New Tools for On-the-Fly Secure Computation

by

Sacha Servan-Schreiber

Sc.B., Brown University, 2019
S.M., Massachusetts Institute of Technology, 2021

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2025

Authored by:	Sacha Servan-Schreiber Department of Electrical Engineering and Computer Science January 24, 2025
Certified by:	Srinivas Devadas Webster Professor of Electrical Engineering and Computer Science Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Department Committee on Graduate Students

New Tools for On-the-Fly Secure Computation

by

Sacha Servan-Schreiber

Submitted to the Department of Electrical Engineering and Computer Science
on January 24, 2025 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

ABSTRACT

Secure computation allows two or more parties, each with a private input, to learn the output of a function computed over their inputs without learning anything else—a fundamental tool with applications to both theoretical cryptography and real-world Internet protocols.

In this thesis, we focus on secure computation protocols where, in a single round of interaction, the parties obtain additive shares that sum to the output. We call this *on-the-fly* secure computation, because such a protocol requires no special setup assumptions and minimizes interaction between parties—two vital properties for real-world deployments.

Unfortunately, the only way to build on-the-fly secure computation for general functions currently requires using *spooky encryption*—a powerful primitive with few prospects for efficient implementations. This reliance on spooky encryption represents a void in our theoretical understanding and creates a barrier for real-world deployments.

This thesis focuses on advancing the theory and practice of secure computation by asking:

Can we realize on-the-fly secure computation without spooky encryption?

In positively resolving this question, we obtain a series of new results that span from novel tools with concretely-efficient, open-source implementations to theoretical paradigms with new implications in several important areas of cryptography. As primary contributions:

- We provide the first concretely-efficient *non-interactive oblivious transfer extension* with plausible post-quantum security, which can be seen as a restricted form of on-the-fly secure computation. Previously, such a result was only known using spooky encryption.
- We initiate the study of *multi-key homomorphic secret sharing*, which allows two parties to compute secret shares of a function over their private inputs in one round, giving the first construction of general on-the-fly secure computation without spooky encryption. Our construction resolves the long-standing open question of obtaining sublinear, two-round secure computation in the common reference string model from weaker assumptions.
- We initiate the study of *simultaneous-message and succinct* secure computation, which generalizes on-the-fly secure computation to have *input* succinctness. Such a result was not known even assuming spooky encryption. As a direct corollary, we get the first trapdoor hash function supporting general predicates from the learning with errors assumption.

Beyond these main contributions, we introduce new approaches for communication-efficient multi-party computation, new constructions of constrained pseudorandom functions and attribute-based non-interactive key exchange, generic compilers for correlation-intractable hashing and rate-1 fully homomorphic encryption, and output-succinct secure computation.

Thesis supervisor: Srinivas Devadas
Title: Webster Professor of Electrical Engineering and Computer Science

Contents

Title page **1**

Abstract **3**

Acknowledgments **10**

1 Introduction **17**
1.1 The Real-World Challenges of Secure Computation 18
1.2 The Complexity of Secure Computation 18
1.2.1 The importance of minimizing round complexity 19
1.3 The Dream: On-the-Fly Secure Computation 20
1.3.1 Perspective: Internet protocols and multi-party computation 21
1.3.2 Perspective: Generalization of Diffie–Hellman key exchange 21
1.3.3 Perspective: Cryptographic primitive vs. protocol 22
1.4 A Taxonomy of Secure Computation Protocols 22
1.4.1 Secure computation from garbled circuits 23
1.4.2 Secure computation from fully homomorphic encryption 24
1.4.3 Secure computation from homomorphic secret sharing 25
1.4.4 Secure computation from spooky encryption 26
1.5 Motivation for Finding New Techniques 26
1.6 Thesis Overview 27
1.7 Thesis Results 28
1.7.1 Results in Part I 28
1.7.2 Results in Part II 33
1.7.3 Results in Part III 37

I New Tools with Practical Applications **41**

2 Constrained Pseudorandom Functions for Inner-Product Predicates **43**
2.1 Introduction 44
2.1.1 Related work 46
2.1.2 Organization 47
2.2 Technical Overview 48
2.2.1 Our approach 48

2.3 Preliminaries . . . 51
2.3.1 Notation . . . 51
2.3.2 Constrained pseudorandom functions . . . 51
2.3.3 RKA-secure PRFs . . . 53
2.4 Adaptive Construction in the Random Oracle Model . . . 54
2.5 A General Framework and Constructions . . . 59
2.5.1 Framework . . . 59
2.5.2 DDH-based construction . . . 63
2.5.3 VDLPN-based construction . . . 66
2.6 CPRFs for Inner-Product Predicates from OWFs . . . 67
2.6.1 Affine RKA-secure PRFs from OWFs . . . 68
2.6.2 CPRF construction from OWFs . . . 71
2.7 Extensions . . . 72
2.7.1 More general constraint predicates . . . 72
2.8 Application to Learning Theory . . . 74
2.9 Evaluation . . . 75
2.9.1 Complexity and benchmarks . . . 75
2.9.2 Comparison to other CPRF constructions . . . 76

3 Oblivious Transfer Extension with a Public-Key Setup 78
3.1 Introduction . . . 79
3.1.1 Our contributions . . . 80
3.2 Technical Overview . . . 82
3.2.1 Background on the BCMPR framework . . . 82
3.2.2 Our approach . . . 84
3.2.3 Two-round OT extension . . . 86
3.2.4 Public-key Setup from Ring LWE . . . 87
3.2.5 Multi-instance security . . . 88
3.3 Preliminaries . . . 89
3.3.1 Notation . . . 89
3.3.2 Cryptographic definitions . . . 89
3.3.3 Oblivious transfer . . . 90
3.4 Shiftable CPRFs . . . 92
3.4.1 Defining shiftable CPRFs . . . 92
3.4.2 Constructing shiftable CPRFs . . . 94
3.4.3 Security analysis . . . 96
3.5 PCFs for ListOT: Framework . . . 96
3.5.1 Defining PCFs for ListOT . . . 96
3.5.2 Framework: PCF for ListOT from IPM-wPRFs . . . 98
3.5.3 Realizing QuietOT from a PCF for ListOT . . . 99
3.6 PCFs for ListOT: Instantiations . . . 101
3.6.1 BIPSW IPM-wPRF instantiation . . . 101
3.6.2 GAR IPM-wPRF instantiation . . . 102
3.6.3 Other instantiations . . . 103
3.7 Public-Key Setup . . . 105

3.7.1 ℓ-instance updatability of shiftable CPRFs . . . 105
3.7.2 Constructing ℓ-instance updatably-secure shiftable CPRFs . . . 107
3.7.3 Defining public-key PCFs for ListOT . . . 107
3.7.4 Constructing public-key PCFs for ListOT . . . 109
3.7.5 Public-key setup from Ring LWE . . . 115
3.8 Implementation and Evaluation . . . 116
3.8.1 Optimizations in the implementation . . . 119
3.9 Application to Large-Scale MPC . . . 121
3.10 Precomputability and Two-Round OT Extension . . . 122
3.10.1 Precomputability . . . 123
3.10.2 Two-round setup from two-round OT . . . 125
3.10.3 Instantiations of QuietOT in the standard model . . . 127
3.11 Deferred Proofs . . . 127
3.11.1 Proof of Theorem 3.4.1 . . . 127
3.11.2 Proof of Theorem 3.5.1 . . . 130
3.11.3 Proof of Theorem 3.7.1 . . . 133
3.11.4 Proof of Theorem 3.7.3 . . . 134

II New Tools with Theoretical Applications 136

4 Distributed Point Functions with a Non-Interactive Setup 138
4.1 Introduction . . . 139
4.1.1 Our results . . . 140
4.1.2 Applications . . . 141
4.2 Technical Overview . . . 143
4.2.1 Building block: Non-interactive multiplication . . . 143
4.2.2 Overview of the NIDPF construction . . . 144
4.3 Preliminaries . . . 149
4.3.1 Notation . . . 150
4.3.2 Additive secret sharing . . . 150
4.3.3 Cryptographic assumptions . . . 150
4.3.4 The NIDLS framework . . . 151
4.3.5 Secret-key homomorphic secret sharing . . . 153
4.3.6 Degree-2 homomorphic secret sharing . . . 154
4.4 Non-Interactive Multiplication . . . 156
4.4.1 NIM with multiplicative output reconstruction . . . 157
4.4.2 Succinct NIM for matrix multiplication . . . 158
4.4.3 Constructions from group-based assumptions . . . 158
4.4.4 Constructions from lattice-based assumptions . . . 160
4.5 Non-Interactive DPF . . . 164
4.5.1 Emulating arithmetic modulo N . . . 166
4.5.2 NIDPF framework . . . 167
4.5.3 Random-payload instantiation from SXDH . . . 170
4.6 Generalization to Non-interactive Computation . . . 175

5 Multi-Key Homomorphic Secret Sharing **177**
5.1 Introduction . . . 178
5.1.1 Applications of multi-key HSS . . . 179
5.2 Technical Overview . . . 183
5.2.1 Notation . . . 183
5.2.2 Background on HSS from group-based assumptions . . . 184
5.2.3 Challenges associated with multi-keyness . . . 186
5.2.4 Solving the synchronization challenges . . . 188
5.2.5 Putting things together . . . 192
5.3 Preliminaries . . . 193
5.3.1 Cryptographic assumptions . . . 193
5.3.2 Cryptographic building blocks . . . 194
5.3.3 Distributed evaluation of RMS programs . . . 198
5.4 Multi-Key Homomorphic Secret Sharing . . . 203
5.4.1 Definition . . . 203
5.4.2 External security . . . 206
5.4.3 Construction . . . 207
5.4.4 Sublinear, two-round secure computation . . . 212
5.5 Attribute-Based Non-Interactive Key Exchange . . . 213
5.6 Public-Key PCFs from MKHSS and Applications . . . 218
5.6.1 Public-key pseudorandom correlation functions . . . 219
5.6.2 Public-key PCFs from MKHSS . . . 221
5.6.3 Multi-party computation with silent preprocessing . . . 225

III Expanding the Frontier **230**

6 Simultaneous-Message and Succinct Secure Computation **232**
6.1 Introduction . . . 233
6.1.1 Our results . . . 235
6.1.2 Related work . . . 238
6.1.3 Chapter organization . . . 239
6.2 Technical Overview . . . 239
6.2.1 Construction from LWE . . . 239
6.2.2 Construction from indistinguishability obfuscation . . . 242
6.3 Preliminaries . . . 245
6.3.1 Notation . . . 245
6.3.2 The learning with errors assumption . . . 246
6.4 Defining SMS Secure Computation . . . 247
6.4.1 Succinct, non-interactive VOLE as SMS . . . 250
6.5 Construction from LWE . . . 251
6.5.1 Preliminaries . . . 251
6.5.2 Construction . . . 253
6.5.3 Setting the parameters . . . 253
6.5.4 Security analysis . . . 257

6.6 Construction from Indistinguishability Obfuscation 261
6.6.1 Preliminaries 261
6.6.2 Construction 265
6.6.3 Setting the parameters 265
6.6.4 Security analysis 265
6.7 Optimizations 268
6.7.1 Unbounded computations 268
6.7.2 Minimizing communication from Bob to Charlie 270
6.7.3 Minimizing computation for Bob 274
6.8 Trapdoor Hashing from SMS 277
6.8.1 Background on TDH and relation to SMS 277
6.8.2 Construction from SMS 279
6.9 Rate-1 FHE from SMS 281
6.9.1 Generic construction from SMS 282
6.9.2 Security analysis 283
6.10 Correlation-Intractable Hashing from SMS 285
6.10.1 Generic construction from SMS 286
6.11 Generic Upgrade to a Simulation-Based Definition 287
6.12 Deferred Proofs 291
6.12.1 Proof of Proposition 6.6.3 291

Acknowledgments

Writing these acknowledgments has been a process of profound reflection, gratitude, and humility. Words feel insufficient to express my appreciation for all the people who have contributed to making this PhD an extraordinary learning experience. When I was accepted to MIT, I was both very excited and deeply apprehensive. The reputation of this institution precedes it—I wasn't sure if I had what it took. Fortunately, there were many people here to help me grow by sharing their first-rate talents and *savoir-faire*. I am deeply grateful to the incredible MIT community for shaping the person I am today.

❀

To my mentors. First and foremost, I'd like to thank Srini Devadas, without whom this whole PhD experience would have taken a very different turn. You were the best advisor I could have hoped to have. What you did, which I may have occasionally taken for granted, was to believe in me and my own path, letting me fall and break my nose until I learned to walk. Thank you for providing me with an academic home and keeping your office door open when I needed advice, whether it was related to a research problem I was thinking through or to managing inevitable life problems. Most of all, thank you for your patience—you had faith in me and, in the end, that made all the difference.

Second, I'm immensely grateful to Geoffroy Couteau, who welcomed me for an extended visit in his lab in Paris. This visit was important to me for both personal and professional reasons. Thank you for fruitful collaborations and sharing with me a fraction of the methods, papers, ideas, and folklore results that you've accrued over the years. It is thanks to you and your willingness to patiently listen to my (many) half-baked ideas that I managed to grow some little seeds into a couple research bonsai. You're a brilliant researcher with an ocean of insights, and I'm humbled to have had the opportunity to work with you.

Third, I'm very appreciative of Elette Boyle, Abhishek Jain, and Yael Kalai for being incredible mentors during two summer internships. I've learned a lot from witnessing each of your unique thinking styles and razor-sharp approaches to tackling difficult problems. This PhD would not have been as fun, and this thesis would not have been as focused, if it were not for these internships and the mental clarity that they provided.

Finally, I'd also like to extend my heartfelt thanks to Dan Berman, Andy van Dam, Piotr Indyk, Tim Kraska, Anna Lysyanskaya, Matteo Riondato, Akshay Srinivasan, Vinod Vaikuntanathan, Daniel Weitzner, and Jacob White for their guidance, advice, collaborations, and support throughout my academic journey.

To my colleagues and collaborators. My PhD has been profoundly shaped by the many inspiring people I was fortunate enough to work with, including: Maxime Bombar, Dung Bui, Lali Devadas, Jules Drean, Clément Ducros, Aparna Gupte, Kevin He, Aditya Hegde, Kyle Hogan, Alexander Koch, Simon Langowski, Nikolas Melissaris, Zack Newman, Cristina Nita-Rotaru, Olya Ohrimenko, Nikola Samardzic, Mayuri Sridhar, Alin Tomescu, Tal Wagner, Jun Wan, Ben Weintraub, Yu Xia, Hanshen Xiao, and Emanuel Zgraggen, All of you have impressed me with your wealth of knowledge and expertise, and I am happy to have had the opportunity to work with all of you on various projects over the years. I'm especially grateful to several close collaborators and friends:

- Aditya, I'm very glad we got to overlap for a week at NTT, which turned out to be one of the most productive weeks of my PhD. It was fun brainstorming and writing with you.
- Alex, thanks for being a wonderful collaborator and friend—we've worked together on so many projects and I've learned a lot from that experience, especially when it comes to mathematical thinking and being a bit more of a stickler for English grammar.
- Dung, thanks for working through tedious ideal functionalities with me and sharing many "deadline lunches" at the IRIF cafeteria.
- Kyle, thanks for being a wonderful collaborator, PRIMES co-mentor, a close friend, and an oracle for life advice—I've learned a lot about precision, organization, research, and aesthetics from you. Also, spending the first months of the pandemic with Chris and you in New Hampshire and Vermont remains among my fondest memories of this time.
- Jules, thanks for being a close friend, PRIMES co-mentor, startup co-founder, and most importantly, introducing me to your best friend Jane—my life would not be as rich if it weren't for your presence and friendship these past years. Looking forward to what's next!
- Lali, thanks for collaborating with me on three out of the five projects in this thesis, sharing the heavy responsibility of TA'ing 6.875 (and making it fun), brainstorming through problems together, and being a fun travel buddy in India.
- Maxime, thanks for teaching me about some cool aspects of code-based cryptography and working through low-level implementation optimizations with me.
- Mayuri, thanks for working on several research problems with me and showing me that it can be nice to walk through Cambridge in the rain.
- Simon, thanks for being a fantastic collaborator on several projects and for countless whiteboard sessions in the office. It's been a lot of fun working, traveling, thinking, and sharing Japantown ramen and A4 pizza with you.
- Zack, thanks for being a great collaborator; I learned a lot from working with you, especially when it comes to writing clean code, being pedantic about LaTeX macros, and taking care of maintaining a linear git commit history.

To my PRIMES students. Albert, Andrew, Eli, Ella, Hyojae, Maya, Patrick, and Simon: thank you for your insightful questions and research enthusiasm; it was a joy watching you all grow and I learned a great deal by working with each one of you.

To the anonymous reviewers. I'm deeply grateful to the security and cryptography community, and especially the reviewers from ASIACRYPT, CRYPTO, EUROCRYPT, NDSS, NSDI, PKC, S&P, and USENIX, who took the time to review my submissions. The quality and thoroughness of the reviews helped me iterate and grow. Thank you for your time.

To the administrative staff. I would like to express my gratitude to the CSAIL and MIT EECS staff, and especially Sally Lee, for dealing with many administrative tasks for me. Moreover, your laughter, bowl of dog treats, and artwork brought life to the office.

To my friends. I feel fortunate to have had the chance to cross paths and spend time with many inspiring humans in CSAIL, in Cambridge, and beyond. Some of these people include Aja, Alana, Albert, Anish, Anne, Ben, Camille, Chris, Derek, Eugenie, Floriane, Giovanni, Guy, Jack, Jasmine, Jon, Kiran, Leah, Lije, Louis, Luke, Manon, Michele, Miyé, Nate, Nicholas, Nicholas (pappou), Nikolas, Oriol, Paul, Peter, Romain, Sarah, Sergio, Shiv, Stella, Sue, Surya, Tanya, Tobin, and Zane. Thank you all for sharing interesting conversations, movies, bike rides, books, hikes, food, runs, apartments, walks, more hikes, music, art, memes, debates, games, and many other activities that significantly enriched my time here. In particular, I'd like to thank a few close friends who made a big difference:

- Alana, for being an inspiration to many and playing a role in the next chapter of my life.
- Anne, for sharing your passion for weird movies, wacky parties, and hacker culture.
- Ben, for caring so deeply, helping so often, and being a conduit to a larger sphere of people.
- Chris, for teaching me about the art of bicycle maintenance and cooking elaborate dinners.
- Jack and Jasmine, for being great neighbors and sharing pancakes on the weekends.
- Leah, for sharing many fun experiences, including making art and diving into research.
- Luke, for going on many cold morning runs and bringing entropy to the people around you.
- Pappou, for being a great godfather and always checking in to see how I was doing.
- Peter, for being an awesome flatmate, dog co-parent, and for many gluttonous Costco runs.
- Manon, for setting the bar on research talks, deep conversations, and rubber duck hunting.
- Miyé, for sharing many adventures in Vermont, including winter hiking to see the eclipse.
- Zane, for sharing thoughts on complexity and quantum theory and indulging in sci-fi TV.

I would have liked to write a personalized message to everyone (including people I may have forgotten to mention above), but this would take too much space in an already long acknowledgments section. Instead, I will resort to the following compression technique invented by Pierre Meyer in his PhD thesis acknowledgments [Mey23]: Generate a long random string, which can be opened to each reader's personalized message using a corresponding key (please contact me for the key, which I'll generate for you *on the fly*). To wit, I leave you with:

```
uOnU9hfpkUS72iHfbRqOGKtd8GoNZEcf9X1HmJDWF18cWnIrdrdQWPSK4S9OUwDqSCwKIryzH94e
1RYAr6cmgbe8jwXvFatZBlPVkBPKd161TODfc5TdqO5bsU4ZDONpXceUKUb3LVyh3FmYam9os65y
```

To my family. Thank you to my mother Olga, whose entrepreneurial spirit [Mar98] (not to mention free spirit) is matched only by her unwavering dedication to my upbringing. You showed me, by example, how to pursue ambitious goals, be independent, and appreciate the value of thoughtful reading and clear writing.

Thank you to my father David (in spirit): Though this thesis is all about removing spooky encryption, I'm still open to your occasional spooky actions at a distance. I'm happy I could share this experience with you in my own way.

To my uncle Edouard, thank you for being both a mentor and guide during my undergraduate years. Your patient tutoring and encouragement led me to discover my love for many aspects of computer science. The hours we spent working through problems together laid the foundation of my academic pursuits.

To my brother Roman, thanks for sharing with me your enthusiasm for building things—I always look forward to seeing what you create.

To my cousin Nina, thank you for providing so much grounding in my life, giving me feedback on my attempts at more literary writing, and sharing so many chats about life and research. I'm proud to have inspired you to pursue your own PhD and look forward to seeing where your ionospheric research takes you (but please stay down to earth).

To my maternal grandparents: Anatole (in spirit), who nurtured me with complex engineering problems at a young age, and Tamara, who loved me for who I was at all times, who read books to me, and whose beautiful soul made me who I am today.

To my paternal grandparents, Jean-Jacques (in spirit) and Sabine: thank you for broadening my horizons through many trips, including Israel and hiking up Mt. Sinai, and for nurturing my mind through your stories and careful preservation of our family history.

To Liliane, for your meticulous attention to detail and genuine care for me each time I visit France. Thank you for sharing crêpes, coffee, and little moments in the kitchen.

To Jane, your love, support, and encouragement have meant more than words can express. Through the ups and downs of the PhD, your cheer in celebrating each victory, your comforting words during each setback, and your energetic presence, have all brought color to my life. Thank you for being a lighthouse in a stormy sea.

Finally, thank you to my dog Elliot. You have been the best writing companion a student could wish for. Thank you for forcefully reminding me to take walks in fresh air and for your unconditional enthusiasm for tug-of-war.

❀

Reflections. Before embarking on the PhD, I walked the Appalachian Trail—spanning from Georgia to Maine—to climb Mt. Katahdin, where I reached the terminus of the trail 2,200 miles from where I started. After four months of walking, putting one foot in front of the other, I finally reached the end. That moment was bittersweet. If I'd had the time, I would have kept walking into Canada and beyond. Now, reaching the terminus of the PhD, the same feeling embraces me, and I remember enjoying my last trail lunch dangling my feet from a scenic overlook. Back then, a young hiker just beginning his southbound journey asked me about the 1,950 miles ahead. In that same spirit, I'd like to leave the following breadcrumbs for my past self, and potentially anyone else who cares to read it:

(1) Read (or watch) the lecture by Richard Hamming titled "You and Your Research." Looking back, I wish I had incorporated more of his advice earlier on. My own experience has taught me that his candid take on the meaning of "work" shines a bright light on many truths that few wish to admit, including myself.

(2) Focus on quality (and maybe read "Zen and the Art of Motorcycle Maintenance" by Robert M. Pirsig, which happens to be Srini's favorite book). Quality takes a long time—a very long time. Finding quality requires fighting one's own demons and instincts that want us to settle for less (admittedly, I still struggle with this fight myself). However, it's important not to let others deter you from your own definition of quality, only you can know what it means, and only you can find it (albeit I often find it elusive).

(3) Find and refine your aesthetic, in research, in ideas, in your presentations. It helps to have it be something you believe is elegant, something you believe is "distilled" to its very essence. In doing so, it helps to understand (possibly at an intuitive level) why you believe it is elegant and why you believe it is pure. This took me years to start honing and I wish I had started the practice sooner. Once you've understood your aesthetic, it's important to find the people that share it.

(4) Be passionate about sharing your ideas, even if they appear half-baked and stupid. In the beginning especially, your ideas are typically not worth safeguarding, as they are not good ideas. The sooner you put them out, the sooner you start iterating on them, the faster you'll get to a place where you can start building something interesting. This is advice you hear over and over again if you listen carefully, whether it's from established researchers or startup founders, yet is often hard to apply in daily life.

(5) Don't be afraid to ask for help and feedback. I've found that most people will be happy to share their thoughts if you ask. Learn to ask, be ready to listen, and always say thanks.

(6) Try not to look back on mistakes and accept that each embarrassing moment was necessary in making who you are today. The learning process, for better or for worse, is messy and filled with mistakes, dejection, and embarrassment. I personally still struggle to accept this fact, but it's echoed by many people pursuing creative tasks—the act of creation necessitates cringing at what you used to think was good and constantly raising the bar.

(7) Teach what you know to understand what you don't know. Few things have been more valuable to me than working with high school students through MIT PRIMES to appreciate the extent of how little I knew myself, and how much there is to discover.

There are these two young fish swimming along and they happen to meet an older fish swimming the other way, who nods at them and says "Morning, boys. How's the water?" And the two young fish swim on for a bit, and then eventually one of them looks over at the other and goes "What the hell is water?"

DAVID FOSTER WALLACE

Chapter 1

Introduction

One of the foundational pillars of modern cryptography is the study of secure computation. It enables the magical goal of allowing two or more parties to compute a function over their private inputs and only learn the result of the computation—nothing more.

Since the initial feasibility results in the 1980s, secure computation has become central to many different areas of cryptography and has far-reaching connections and implications. The seminal works of Yao [Yao86] and Goldreich, Micali, and Wigderson [GMW87] proved that any efficiently-computable function could be securely evaluated by two (or more) parties; specifically ensuring that each party learns nothing beyond the intended output. This breakthrough has led to nearly four decades of research dedicated to improving both the concrete and asymptotic overheads of secure computation, as well as understanding its fundamental limits, often motivated by the goal of solving real-world challenges.

The practical importance of secure computation has grown dramatically as data breaches and widespread tracking has become increasingly pervasive on the Internet. Secure computation offers solutions that guarantee both functionality and privacy, and its potential to affect real-world protocols is reflected in several ongoing standardization efforts in industry as well as academic systems research (e.g., [IKNP03, DPSZ12, KSS13, WYG+17, CGB17, SSLD22, MW22, HDCGZ23, HHC+23, BBC+24, ZPZS24]). For example, the MPC Alliance works with industry to deploy secure computation protocols, the Internet Engineering Task Force (IETF) is developing a proposal for private analytics systems [GPP+24] using specialized secure computation techniques, the National Institute of Standards and Technology is standardizing post-quantum digital signatures [ABC+24] based on the theoretical underpinnings of secure computation, and the World Wide Web Consortium (W3C) is working on an online advertising attribution proposal that uses general secure computation as a building block [W3C21].

Indeed, the impact of secure computation protocols extends far beyond building privacy-preserving Internet systems. It has proven to be a fundamental building block for constructing zero-knowledge proofs [IKOS07], digital signatures [Fen23], and even protecting against side-channel attacks in hardware [ISW03]. These theoretical advances have produced practical applications in blockchains [BCG+14, CM19], digital identities [SLC+24], and more.

1.1 The Real-World Challenges of Secure Computation

Despite the promising potential of secure computation, real-world adoption remains limited to a small set of specific tasks (e.g., simple analytics) and wider deployments face barriers.

The work of Halevi, Lindell, and Pinkas [HLP11] identifies an important gap between secure computation protocols and how most other protocols function on the Internet. This gap takes the form of *interaction*. Secure computation protocols require more rounds of interaction between parties to compute the function—which translates to needing to make additional availability assumptions on parties. In real-world deployments, where parties might be in different geographic regions and where network conditions can vary significantly, requiring multiple rounds of communication introduces latency and imposes often unrealistic availability assumptions on the computing parties. Moreover, while computation and communication costs can be mitigated with better infrastructure (larger Internet cables, more compute power), round complexity is inherently tied to the speed of light, making it a hard lower bound on the total time it takes to execute the protocol. An overarching goal of this thesis is to address this limitation and mitigate the barriers inhibiting practical deployments.

In the remainder of this chapter, we expand on the key metrics that determine the practical feasibility of secure computation by providing a comparison to the way in which regular (i.e., *non*-secure computation) protocols are deployed on the Internet today. This then leads us to an ideal goal—"on-the-fly" secure computation—and we will examine how existing approaches to secure computation compare against it. Understanding these elements will set the stage for the contributions of this thesis, which span from novel theoretical approaches to new practical implementations.

1.2 The Complexity of Secure Computation

To understand the state-of-the-art—both in theory and in practice—when it comes to secure computation, and the remaining challenges inhibiting real-world deployments, we examine three key complexity measures that dictate practical feasibility. These measures help to bridge the gap between theoretical possibilities and practical implementations.

Since the initial feasibility result [Yao86, GMW87], researchers have focused on reducing three complexity metrics in secure computation: (1) computational overhead, (2) communication complexity, and (3) round complexity. In more detail:

The computational complexity captures how much resources each party needs to expend in order to run the protocol.

The communication complexity captures how many bits of information the computing parties need to exchange in order to run the protocol. The notion of communication complexity is especially interesting when described in terms of the size of the circuit description of the function being computed, since only a handful of techniques achieve sublinear communication in the circuit size [Mey23].

The round complexity measures how many sequential messages the parties need to exchange, one after the other, in order to securely evaluate the function. It is known that two rounds are necessary and sufficient to evaluate functions securely between two parties [Yao86, HLP11].

Remark 1 (Counting rounds)**.** *A round consists of messages being sent simultaneously by any subset of parties to any other subset of parties. Importantly, this differs from a "round-trip," which requires messages to go back and forth: a round-trip counts as two rounds since it involves two sequential message transmissions and cannot be parallelized. For example, if Alice sends a message to Bob (round 1), Bob responds to Alice (round 2) based on the first message, and Alice sends another message back (round 3) based on the previous message, this counts as three rounds, even though it only completes one and a half round-trips.*

1.2.1 The importance of minimizing round complexity

Of the aforementioned complexity measures, *round complexity* often places the greatest limits on the real-world adoption of secure computation protocols [HLP11, BCPW15]. More rounds implies more inter-dependent messages between parties, which imposes cumbersome availability assumptions (the parties need to be online and communication channels must remain open) that are often unrealistic to expect on the Internet where churn is high, faults are frequent, and coordination is difficult. Moreover, reducing round complexity is known to have far-reaching *theoretical* implications (making it an extensively studied complexity metric in the secure computation literature [KO04, HLP11, BCPW15, GMPP16]). From a purely practical perspective, however, lower round complexity often leads to simpler protocol designs and implementations, makes it easier to reason about protocol security and results in more maintainable deployments. Anecdotally, both the W3C and the IETF workshops dedicated to the standardization of various secure computation protocols disproportionately favor simpler protocols that involve fewer rounds of interactions. This is because implementations involving multiple rounds require intermediate state management and significantly more complex standardization documents, hindering the ability for a broader audience to reason about them and increasing the likelihood of implementation bugs.

Properties of Internet protocols. To better understand why round complexity is so important to practical deployments and real-world adoption of secure computation—despite being a purely "theoretical" notion—it is helpful to consider how existing client-server protocols function on the Internet today. Specifically, consider the following protocol template between two parties—Alice (e.g., a client) and Bob (e.g., a server)—for computing some function f:

> **Round 1:** Alice sends her input x to Bob over an encrypted channel.
>
> **Round 2:** Bob computes the function f over his input y and the received input x, and sends back $f(x, y)$ to Alice (or publishes it to a bulletin board).

While the above protocol is "insecure," given that it reveals Alice's input to Bob, it has several very appealing features—and models how many Internet protocols operate today. First, it achieves optimal efficiency across all complexity measures: two rounds of communication (each sequential transmission is counted as a round; cf. Remark 1) and no bandwidth or computational overhead for the parties. Second, and more importantly, it has three other "implicit" features, which help to explain its practical appeal:

(1) The protocol can be initiated "on the fly," as there is no need for prior coordination between the parties and no special setup assumptions imposed on the parties. (For now, we gloss over the need for public key infrastructure to establish encrypted channels.)

(2) The protocol offers a non-interactive disclosure of the output: Bob can directly post the computed result to a public bulletin board without having to coordinate with Alice.

(3) The protocol is asynchronous: Alice can go offline immediately after sending her message—there is no need for subsequent interaction with Bob assuming that the result is to be made public (Alice can also get the result at a later point in time).

All three of the above properties remove availability assumptions from the parties, making it easy to run such a protocol over unreliable channels with unreliable parties. Unfortunately, these properties are largely absent in existing secure computation protocols, especially protocols involving many parties, as discussed in depth by Halevi et al. [HLP11].

In particular, with the exception of spooky encryption [DHRW16] (which we will cover in more detail later), all existing approaches to secure computation fail to have some or all of these features, making them of limited practical interest—even if we ignore the communication and computational overheads they incur. Given this state of affairs, we describe an ideal goal for secure computation protocols: remove coordination and availability requirements on participants, matching, to the extent possible, the insecure model outlined above. We call this ideal model, "on-the-fly" secure computation. In particular, we note that lower bounds in secure computation prevent us from completely removing interaction [HLP11].

1.3 The Dream: On-the-Fly Secure Computation

Partially inspired by López-Alt, Tromer and, Vaikuntanathan [LTV12], in this thesis, we will call a secure computation protocol that imposes minimal availability assumptions on the parties an *on-the-fly* secure computation protocol. A little more formally, we will say that a secure computation protocol supports on-the-fly computation if it is a *one-round* protocol where parties obtain secret shares of the computation result.

We will focus primarily on the case where parties get *additive shares* (summing the share obtained by each party, over some algebraic group, gives the result of the computation). Additive reconstruction has several nice properties, such as optimal share size and public reconstruction—the parties can publicly disclose result in a second round (e.g., by posting their shares to a public bulletin board).

As we will explain next, a two-round protocol with non-interactive public disclosure suffices to reach the efficiency properties of the "insecure" protocol outlined in the previous section, and achieves the two-round lower-bound for secure computation [HLP11]. Moreover, two-round protocols often enable treating the first-round messages as public encodings of each party's input, which then gives the following *reusability* of the first-round message in a context where p distinct parties engage in pairwise secure computations:

> **Round 1:** All parties post their first-round messages of the two-round protocol to the public bulletin board, where this message is seen as a public encoding of their input.
>
> **Round 2:** Any distinct *pair* of parties in the set of p parties can use the published first-round messages to locally compute shares of a function evaluation, and use the *non-interactive disclosure* property to publish the result to the bulletin board.

Notice that, from the perspective of any given party, the identity of the second party in

the second round can change without affecting the first round, thanks to the reusability of the first message. This makes the protocol "on-the-fly" in the sense that, after the first round (which is independent of all other parties), the parties never need to interact directly over a secure channel, while still having the ability to engage in a secure computation.

1.3.1 Perspective: Internet protocols and multi-party computation

We emphasize that on-the-fly secure computation mirrors existing deployments of client-server protocols on the Internet, where (1) parties *do not* need to engage in a prior setup phase and (2) where the result can be made public without further interaction. When it comes to secure computation, these features have many practical benefits. For example, an on-the-fly communication pattern enables efficient *pairwise* computation in a multi-party context by reducing the communication in the total number of parties, thanks to reusability of the first message. Specifically, some multi-party computation protocols require pairwise secure two-party computation between all parties, resulting in a quadratic communication complexity $\Omega(p^2)$ in the number of parties p. However, on-the-fly secure computation gives the parties the ability to *reuse* the first-round message with all other parties, as outlined at the beginning of this section. Using these properties, it becomes possible to instantiate multi-party computation protocols with only $O(p)$ communication, reducing the communication by a quadratic factor (cf. Chapters 3 and 5). Indeed, in some cases, this enables multi-party protocols with asymptotically optimal communication complexity.

1.3.2 Perspective: Generalization of Diffie–Hellman key exchange

An on-the-fly secure computation protocol can be seen as a generalization of the Diffie–Hellman key exchange protocol [DH76], which has defined the modern era of cryptography and has become one of the most widely deployed protocols on the Internet [KPW13,KRA+18]. And so, it is perhaps no coincidence that it can be viewed as on-the-fly secure computation.

In the Diffie–Hellman protocol, each party generates a public key that it posts to a public bulletin board, which is directly captured by the first round of the on-the-fly secure computation protocol (with a reusable first round). Then, using the public key of another party, along with its own secret state, each party can locally (i.e., non-interactively) derive a pseudorandom key $k \in \{0,1\}^\lambda$. We can equivalently view k as being an additive secret share of 0^λ, held by each party.

A little more concretely, by viewing k as an XOR-additive secret share of the degenerate all-zeros function, because we have $k \oplus k = 0^\lambda$, it becomes clear that the Diffie–Hellman protocol is an on-the-fly secure computation protocol for computing $f(x, y) = 0$, for all x and y. While, on the surface, this connection may appear contrived, it uncovers an important implication that sets on-the-fly secure computation apart from standard two-round secure computation protocols. Using the theorem of Gilboa, Ishai, Lin, and Tessaro [BGI+18, Proposition 4.7], any on-the-fly secure computation protocol for computing any non-linear function (e.g., even an AND of two bits) already implies the ability to perform non-interactive key exchange *à la* Diffie–Hellman. In particular, this connection implies that on-the-fly secure computation is black-box-separable from simpler primitives such as oblivious transfer.

1.3.3 Perspective: Cryptographic primitive vs. protocol

On-the-fly secure computation can be seen as a *primitive* rather than a *protocol* since it can be described by two polynomial-time algorithms Encode and Decode, rather than interactive Turing machines. In particular, Encode is used by parties to encode their inputs in the first round and Decode is used by parties to decode secret shares of the result. This is illustrated in Figure 1.1. This syntactic simplicity of on-the-fly secure computation makes it align more with cryptographic primitives (e.g., such as non-interactive key exchange, digital signatures, and encryption schemes) rather than protocols. In particular, the above description highlights why on-the-fly secure computation avoids the availability assumptions on parties and, moreover, why spooky encryption [DHRW16] is cast as an *encryption scheme* rather than a *protocol*, even though it directly enables many secure computation protocols, as discussed in Section 1.4.

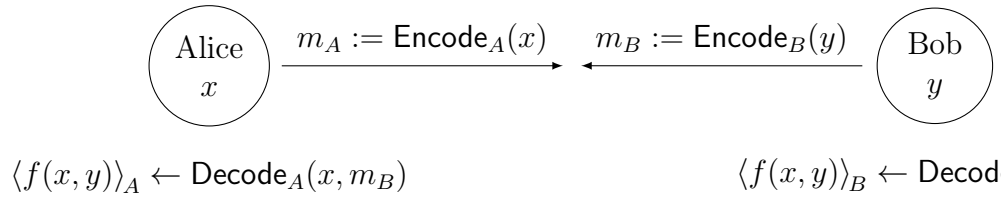


Figure 1.1: On-the-fly secure computation described as a cryptographic primitive. The secret share of party $\sigma \in \{A, B\}$ is denoted as $\langle f(x,y)\rangle_\sigma$, such that $\langle f(x,y)\rangle_A + \langle f(x,y)\rangle_B = f(x,y)$.

1.4 A Taxonomy of Secure Computation Protocols

In this section, we overview existing approaches to secure two-party computation and explain why most of them fail to be *on-the-fly* secure computation protocols. To date, when considering general classes of functions, there are essentially only four tools for secure computation in the two-party setting: (1) garbled circuits [Yao86], (2) fully homomorphic encryption [Gen09], (3) homomorphic secret sharing [BGI16], and (4) spooky encryption [DHRW16]. We ignore approaches such as the GMW protocol [GMW87] (which is tailored to multi-party computation and requires many rounds of interaction) and protocols tailored to specific computations. The above four approaches differ in their core mechanism:

- Garbled circuit schemes transform the computation itself into an encrypted (i.e., "garbled") form that another party can evaluate to get the result.
- Fully homomorphic encryption (FHE) enables computing any function directly on encrypted inputs to obtain an encryption of the result.
- Homomorphic secret sharing (HSS) splits the inputs between parties using secret shares that a function can be evaluated over.
- Spooky encryption enables computing on *independently encrypted* inputs with parties obtaining secret shares of the output after the first round of interaction.

Each approach represents a distinct paradigm for achieving secure computation, with its own set of tradeoffs in terms of complexity measures, cryptographic assumptions, and

practical considerations. Understanding these differences is crucial for appreciating both the evolution of secure computation and the motivation for developing new techniques.

Of the four existing approaches, only spooky encryption can be classified as enabling on-the-fly secure computation. To better understand the power of spooky encryption, the type of secure computation it enables, and how the results presented in this thesis improve on the state-of-the-art, we describe how each of these respective approaches can be used to securely compute a function between two parties (e.g., a client and a server). We focus our attention on the *round complexity* of each approach in a setting where the output is made publicly available (i.e., posted to a bulletin board).

Remark 2 (On multi-party computation)**.** *The focus of this thesis is on secure* two-party *computation and we will primarily provide background techniques in this setting. However, our results have direct applications to multi-party computation as well.*

1.4.1 Secure computation from garbled circuits

The original feasibility result of Yao showing that it is possible to compute any efficiently-computable function securely over inputs provided by two parties, was constructed from a simple building block: oblivious transfer (OT) [Rab81]. In a nutshell, OT is a two-party protocol allowing a sender to obliviously transmit one of two messages to a receiver. The appealing features of the garbled circuit approach are its simplicity and the minimal assumptions (OT is known to be necessary and sufficient for secure computation [Kil88]). Assuming the existence of *two-round* OT [BM90, NP01], it becomes possible to instantiate the following high-level secure computation protocol, where the output is made publicly available:

> **Round 1**: Alice initiates the protocol by sending an encoding of her input x to Bob and keeps the randomness used to generate the encoding of x as her secret state.
>
> **Round 2**: Bob responds by sending an encoding of $f(x, y)$, which consists of a "garbled" circuit computing f, generated using the encoded input x and his input y.
>
> **Round 3**: Alice locally decodes the result $f(x, y)$ by evaluating the garbled circuit using her secret state, and posts the result to a public bulletin board.

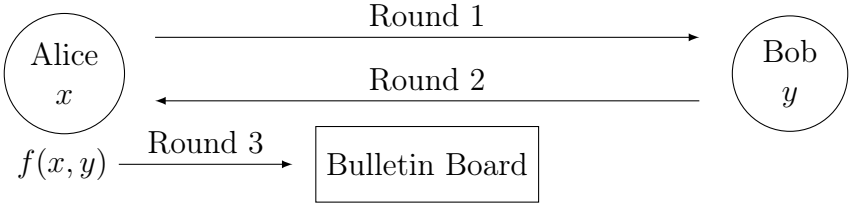


Figure 1.2: Communication pattern of secure computation protocols instantiated using garbled circuits with two-round OT or using FHE. In particular, both approaches require a minimum of three rounds to disclose the output publicly (e.g., to a public bulletin board) since Alice needs to "decode" the result received from Bob using her secret state (e.g., secret key).

At a very high level, Alice's encoding consists of OT messages for obliviously retrieving a circuit evaluation key from Bob that corresponds to her input x. The communication pattern

of this protocol is illustrated in Figure 1.2. In particular, any two-party secure computation protocol making use of garbled circuits requires a minimum of three rounds to make the result public, and therefore fails to achieve the notion of non-interactive disclosure. This is because Alice, who received the "garbled" circuit, needs to use her secret state to evaluate it and obtain the result. Moreover, secure computation from garbled circuits has a total communication proportional to the size of the circuit description, as it places no restrictions on the size of the encodings. While we do not state it as an explicit goal for on-the-fly secure computation, reducing communication complexity is often desirable.

1.4.2 Secure computation from fully homomorphic encryption

Since the breakthrough work of Gentry [Gen09] building FHE, it became possible to securely compute a function between two parties with sublinear communication in the circuit description. Indeed, FHE minimizes the communication complexity by not having the communication grow with the size of the circuit description (in contrast to the garbled circuit approach described above). With FHE, it becomes possible to instantiate the following high-level secure computation protocol:

> **Round 1**: Alice generates an FHE secret key, and sends her encrypted input x to Bob.
>
> **Round 2**: Bob locally evaluates the function over Alice's ciphertext and his own input y to obtain an encryption of the result $f(x, y)$, which he sends to Alice.
>
> **Round 3**: Alice uses her secret key to decrypt the result received from Bob, and posts the output to a public bulletin board.

In particular, because the input is encrypted and only Alice has the secret key, Bob learns nothing about x. Moreover, with (circuit-private [Gen09, vDGHV10]) FHE, the evaluated ciphertext reveals only $f(x, y)$ to Alice.

Notice that the third round is necessary because only Alice has knowledge of the secret key, making it impossible to non-interactively disclose the output: she needs to decrypt it first using her key. Therefore, any secure computation protocol from FHE requires a minimum of three rounds to make the result public (similarly to the garbled circuit approach), and follows the same communication pattern illustrated in Figure 1.2. However, unlike with the garbled circuit approach, the total communication between parties is *sublinear* with respect to the size of the circuit description because the evaluated ciphertext size is independent of the function being computed. In contrast, using garbled circuits, Bob sends an "encoded" circuit to Alice who then acts as the evaluator, necessitating linear communication in the circuit description. This succinctness property of FHE is important for many real-world applications, where one party might have a large input (e.g., a database) or where the function being computed does not admit a small circuit description. Indeed, all the protocols considered in this thesis will achieve different flavors of succinctness, either with respect to the circuit description or even with respect to the inputs of the parties, as we explain later. We note in passing that a protocol that achieves *input* succinctness also achieves succinctness with respect to the size of the circuit description, since the circuit must be at least as large as the length of its inputs (cf. Chapter 6).

1.4.3 Secure computation from homomorphic secret sharing

The seminal work of Boyle, Gilboa, and Ishai [BGI16] introduced the notion of homomorphic secret sharing (HSS) as an alternative to FHE for secure computation, motivated by finding new approaches for breaking the "circuit size barrier" associated with classical garbled-circuit-based secure computation. Using HSS, two parties can compute all polynomial-size branching programs (which are sufficiently powerful to evaluate all circuits in the computational complexity class NC^1) sublinearly in the size of the circuit description, which was previously only known using FHE.

Interestingly, HSS schemes offer an appealing property which is not available with the FHE-based approach: the output of each party consist of additive secret shares of the function evaluation (rather than a ciphertext), which enables recovering the output publicly—without needing to use secret state. This automatically makes it possible to publicly disclose the output in the last round, since additive secret shares just need to be added together to reveal the result. Unfortunately, however, all existing HSS constructions require a minimum of three rounds to securely compute a function. In particular, HSS gives rise to the following secure computation protocol (the communication pattern is illustrated in Figure 1.3):

> **Round 1**: Both parties agree on an HSS public key and derive shares of the secret key using a suitable simultaneous-message (i.e., one round) protocol [OSY21, ADOS22].
>
> **Round 2**: The parties swap their private inputs encrypted using the public key.
>
> **Round 3**: Each party locally evaluates the function over both encrypted inputs to obtain secret shares of the result, which it posts to a public bulletin board.

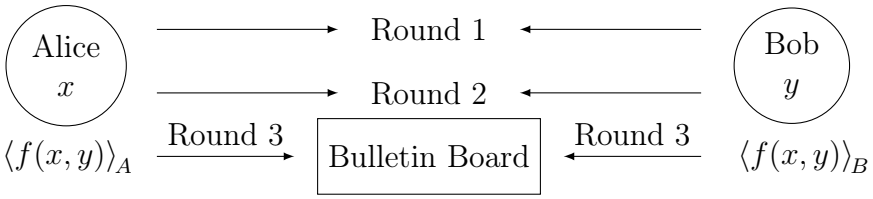


Figure 1.3: Communication pattern of secure computation instantiated using HSS. Similarly to the garbled circuit or FHE-based approach, HSS requires a minimum of three rounds to disclose the output publicly. Note that $f(x,y) = \langle f(x,y)\rangle_A + \langle f(x,y)\rangle_B$, making it possible for anyone to reconstruct the output by adding together the shares posted to the public bulletin board.

The main advantage of HSS over FHE is the non-interactive public disclosure property implied by the fact that no secret state is needed in order to reconstruct the result of the evaluation. In particular, because the parties obtain additive shares of the output $f(x,y)$ at the end of the second round, where the secret share held by party $\sigma \in \{A, B\}$ is denoted by $\langle f(x,y)\rangle_\sigma$, the parties can post their shares to the bulletin board where anyone can reconstruct the output $f(x,y)$ by computing $\langle f(x,y)\rangle_A + \langle f(x,y)\rangle_B$. This additive reconstruction property has many other benefits too, one of them being *compactness*—additive shares are the same size as the message they encode. Unfortunately, however, HSS does not enable on-the-fly secure computation, due to the three-round requirement. We show how to resolve this in Chapter 5 by introducing *multi-key* HSS which eliminates the need for the first round.

1.4.4 Secure computation from spooky encryption

Spooky encryption, introduced in the seminal work of Dodis, Halevi, Rothblum, and Wichs [DHRW16], is the only known approach for building on-the-fly secure computation. Specifically, spooky encryption generalizes the FHE-based approach *and* the HSS-based approach by needing only two rounds of interaction and supporting additive reconstruction.

With spooky encryption, parties can perform the following two-round protocol:

Round 1: Parties exchange their private inputs encrypted under *independent* keys.

Round 2: Each party locally runs a special evaluation algorithm over the encrypted inputs and obtains an additive share of the result,[1] which it posts to the bulletin board.

Notice that after the first round, parties can complete the second round asynchronously (without having to interact with one another) by just posting the second-round messages—the secret shares of the output—to the public bulletin board. This model of communication is depicted in Figure 1.4, and captures to our notion of on-the-fly secure computation.

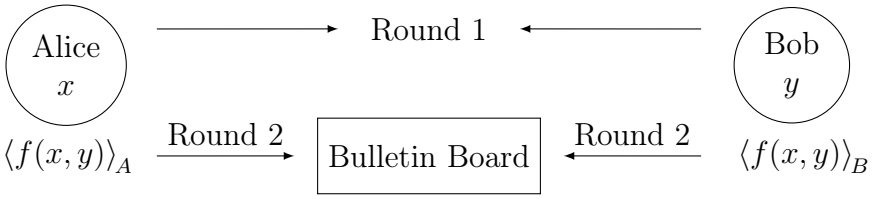


Figure 1.4: Communication pattern of on-the-fly secure computation protocols. After a single, simultaneous round of interaction, the parties can locally derive additive shares of $f(x, y)$—the function f computed over their joint inputs x and y. Note that $f(x, y) = \langle f(x, y)\rangle_A + \langle f(x, y)\rangle_B$, making it possible for anyone to reconstruct the output by adding together the secret shares posted on the public bulletin board.

We stress that spooky encryption gives us on-the-fly secure computation as it enables a protocol with the best possible communication pattern: (1) it requires only a single, simultaneous exchange between parties for them to be able to locally obtain additive secret shares of the result and (2) the shares computed by both parties can be disclosed publicly in a second round. In particular, (1) one round is information-theoretically necessary and (2) is optimal due to the two-round lower bound for secure computation. As a bonus, spooky encryption guarantees *succinctness* with respect to the circuit description, similarly to the FHE and HSS-based approaches.

1.5 Motivation for Finding New Techniques

Given the existence of spooky encryption, which meets our dream goal of on-the-fly secure computation described in Section 1.3, it is natural to ask why pursuing alternative approaches is necessary, let alone of interest to the cryptographic community. Indeed, spooky encryption

[1] This is the "spooky" part of spooky encryption: it converts uncorrelated encrypted inputs generated entirely independently of one another into correlated additive shares of the evaluation result.

achieves strictly more than the goals outlined in Section 1.3 by additionally offering sublinear communication in the circuit description. However, despite this beautiful feasibility result, there remain many unsatisfactory aspects associated with using spooky encryption to instantiate on-the-fly secure computation, which motivates finding alternative approaches.

The first motivation is that spooky encryption is *only* known from powerful assumptions—namely from a strong form of the learning with errors (LWE) assumption or the existence of sub-exponentially-secure indistinguishability obfuscation ($i\mathcal{O}$)—which limits the available avenues we have for instantiating on-the-fly secure computation.

The second motivation is that spooky encryption makes use of "heavy hammer" building blocks—specific constructions of multi-key FHE—which do not currently have known implementations, making it difficult to envision practical deployments.

The exploration of alternative approaches to secure computation serves several fundamental goals. First, discovering new theoretical foundations often reveals unexpected connections between primitives and leads to more efficient constructions—as exemplified by how different approaches to public-key encryption ultimately enabled diverse post-quantum candidates to many cryptographic primitives [BL17, ABC+24].

Second, different constructions often have complementary advantages: while some maximize asymptotic efficiency, others might offer better concrete performance for specific use cases. The ability of constructing the same primitives from a diverse set of cryptographic assumptions typically improves the available options for obtaining practical constructions (Chapters 2 and 3 of this thesis are a direct testimony to this claim).

Third, understanding the minimal assumptions necessary for secure computation helps identify the true boundaries between feasible and infeasible protocols, and what makes certain primitives or assumptions especially powerful and worthwhile of further study. While spooky encryption provides an elegant solution to on-the-fly secure computation, the history of cryptography shows that initial feasibility results often lead to a rich tapestry of alternative approaches, with the alternatives uncovering surprising implications and applications.

For instance, Gentry's original feasibility result for building FHE [Gen09] was based on specific ideal lattice assumptions and not practical. It was only the followup work of Brakerski and Vaikuntanathan [BV11] that constructed FHE from the standard LWE assumption and eventually paved the way for concretely-efficient schemes. In turn, these alternative approaches led to an explosion of lattice-based cryptography [Pei15], enriched our theoretical understanding of the LWE assumption, and resulted in many practical schemes (e.g., [CLP17, HS20, BBB+22]). Similarly, finding new approaches for constructing on-the-fly secure computation promises to unlock both theoretical and practical advances.

1.6 Thesis Overview

This thesis explores new theoretical and practical frontiers in secure computation. Our investigation serves to achieve two main goals: (1) advancing the foundations of secure computation and its broader connections to theoretical cryptography, and (2) developing more practical protocols for real-world deployment.

Part I. In the first part of this thesis, we demonstrate that certain restricted, yet still important, forms of on-the-fly computation can be achieved using lightweight primitives offering

concretely-efficient implementations. These results serve to advance both our theoretical understanding and practical implementations, and have immediate applications to large-scale secure computation involving many parties. Moreover, our constructions and open-source implementations significantly advance the state-of-the-art in terms of performance.

Part II. In the second part of this thesis, we show that spooky encryption is not necessary to build on-the-fly secure computation for general function classes through new approaches from a variety of standard assumptions. Our constructions significantly advance our theoretical understanding of on-the-fly secure computation for complex functions and resolve several open problems along the way. These contributions lay the groundwork for future advances in both theory and practice.

Part III. In the third part of this thesis, we explore the frontier of what was known to be possible, even when assuming powerful building blocks like spooky encryption. We show that we can construct *input succinct* on-the-fly secure computation, which introduces an important succinctness requirement (and thus is a stronger primitive than spooky encryption) and which implies other powerful primitives in cryptography, such as trapdoor hash functions [DGI+19], rate-1 FHE [BDGM19, GH19], correlation-intractable hash function [CGH98], and more. Our formalizations and constructions push the round and communication efficiency of secure computation to the limit, and pave the way for new research directions.

Overall, this thesis opens many promising new directions for future research. On the theoretical side, our results raise intriguing questions about the minimal assumptions necessary for various forms of secure computation and other important cryptographic objects. On the practical side, our results provide new approaches for building efficient protocols tailored to specific real-world scenarios. More broadly, our results demonstrate how theoretical advances in cryptography can directly enable practical improvements in privacy-preserving computation. In sum, this thesis chips away at the current barriers limiting widespread deployment of secure computation by aligning more closely with the availability assumptions on the Internet.

1.7 Thesis Results

The thesis is divided into three parts, covering primitives on the "low-end" of cryptographic primitives, which come with efficient implementations, to the "high-end" of cryptographic primitives that push the frontier of what was previously known to be possible—even from powerful primitives like spooky encryption.

In the following sections, we overview the main results in this thesis and how they pertain to on-the-fly secure computation. A technical overview providing in-depth details on each of these results can be found in the corresponding chapter.

1.7.1 Results in Part I

The first part of this thesis consists of two chapters (Chapters 2 and 3) and covers new tools with applications to practical (i.e., concretely efficient) secure computation. Chapter 2 builds constrained pseudorandom functions—an important cryptographic primitive with applications to secure computation [Ria24]—from a variety of assumptions that enable concretely-efficient

implementations. In Chapter 3, we directly build upon the results from Chapter 2 to realize on-the-fly secure computation for oblivious transfer extension, an important building block in computation protocols.

Results in Chapter 2

Pseudorandom functions (PRFs) are a foundational tool in cryptography and allow a party with a short key K to produce strings that are computationally indistinguishable from random. A useful extension to standard PRFs is the notion of a constrained PRF (CPRF) [BW13,KPTZ13,BGI14]. At a high level, CPRFs are PRFs that allow generating a constrained key K' from the master key K, determined with respect to a predicate. The constrained key K' can be used to evaluate the PRF on all inputs that satisfy the predicate. However, for all other inputs (i.e., that do not satisfy the predicate) the PRF evaluation under K must appear pseudorandom, even conditioned on the constrained key.

In the last decade, CPRFs have seen diverse applications in secure computation [BGI17, BCG+19a, BGIK22, BCM+24] and cryptography more broadly. To date, existing CPRF constructions can be classified into two groups: efficient constructions for very simple constraint predicates and purely theoretical constructions for general predicates. In Chapter 2, we make progress on bridging this gap by building concretely-efficient constrained PRFs for inner-product constraint predicates from a variety of assumptions under a unified framework.

Contributions. The focus of Chapter 2 is building a framework for constructing CPRFs from the well-studied notion of related-key-attack (RKA) secure PRFs [BK03]. This gives us a diverse set of assumptions from which we can instantiate CPRFs for inner-product constraint predicates. Using this framework, we show constructions of CPRFs for inner-product predicates either unconditionally in the random oracle model or from several standard assumptions. In particular, we show instantiations from the decisional Diffie–Hellman (DDH) assumption, the variable-density learning parity with noise (VDLPN) assumption [BCG+20a], and under the minimal assumption that one-way functions exist. The results push both our theoretical understanding on CPRFs, as well as the state-of-the-art when it comes to concrete efficiency. Overall, Chapter 2 helps to shed light on the core building blocks and assumptions that can be used to construct CPRFs and provides several avenues for future work. We summarize the main results in the following informal theorem:

Theorem 1.7.1 (Informal)**.** *There exist CPRFs supporting inner-product constraint predicates under either (1) the random oracle model, (2) the DDH assumption, or (3) the VDLPN assumption. Furthermore, if restricted to a polynomial domain, there exist CPRFs supporting inner-product constraint predicates assuming that one-way functions exist.*

Our constructions in Chapter 2 result in the first practical CPRFs for inner-product predicates. All prior approaches for constructing CPRFs for inner-product predicates either required powerful assumptions [DKN+20] (which were known to be sufficient to instantiate CPRFs for general constraint predicates), or used computationally-inefficient primitives [CMPR23].[2] We provide an open-source implementation for two of our constructions and demonstrate concrete efficiency on a range of parameters.

[2] In a concurrent and independent work, Bui et al. [BCM+24] constructed a CPRF for a slightly more general predicate class they call "inner-product membership," using a non-standard variant of power-DDH.

Theorem 1.7.2 (Informal)**.** *There exists a* concretely-efficient *CPRF supporting inner-product constraint predicates either in the random oracle model or under the DDH assumption.*

We believe that our techniques may help to understand the minimal ingredients necessary for building CPRFs and will be useful in real-world applications.

Open questions. The results in Chapter 2 raise the following open questions:

Question 1. Is it possible to construct CPRFs for general predicates from RKA-secure PRFs?

The results from Chapter 2 focus exclusively on inner-product predicates, capture all linear functions of the constraint as well as predicates described by constant-degree polynomials. A very intriguing open problem is extending this class to *inner-product membership* predicates [BCM+24], which capture a slightly more powerful predicate class. Similarly, it is conceivable that techniques from Couteau, Meyer, Passelègue, and Riahinia [CMPR23], which realizes constrained PRFs from homomorphic secret sharing, can be combined with the ideas from Chapter 2 to build concretely-efficient CPRFs supporting NC^1 predicates.

Question 2. Is it possible to construct suitable RKA-secure PRFs from more assumptions to instantiate our framework with?

A related question is whether there exist other constructions of RKA-secure PRFs that are suitable to instantiate our current framework. While we have shown a diverse set of instantiations, new constructions of RKA-secure PRFs will yield new instantiations of CPRFs.

Question 3. Do CPRFs for inner-product predicates have other practical applications, aside from those described in Chapter 3?

In Chapter 3, we heavily exploit the concrete efficiency of the CPRFs constructed from Chapter 2 to realize oblivious transfer extension. A natural question is whether other practical applications of CPRFs with inner-product constraint predicates exist. While inner-product predicates are relatively weak, the lack of concretely-efficient constructions is arguably a factor in the dearth of existing applications. In particular, CPRFs for puncturing constraints—which are one of the few constraint predicates with efficient implementations—have found numerous practical applications (e.g., [BMO17, SYL+18, SGRR19, BBMHS22, MZRA22, BBD+23, LP23, Fen23]). Given that we already have one practical application in Chapter 3, we believe that more concrete applications for CPRFs will follow.

Question 4. Is it possible to construct CPRFs for inner-product predicates from the minimal assumption that one-way functions exist?

This last question is partially resolved in Chapter 2, where we show that a CPRF for inner-product predicates can be constructed from one-way functions by (1) restricting the domain to be *polynomial* in size with respect to the security parameter and (2) sacrificing concrete efficiency (the parameters of the scheme become unwieldy making it purely of theoretical interest). A very interesting open question is whether it is possible to resolve one or both of these limitations associated with our construction. Indeed, since minimal assumptions often also translate to practical efficiency (by supporting a wider range of instantiations), resolving this question may also help to realize more efficient constructions.

Connection to on-the-fly secure computation. The results in Chapter 2 do not directly pertain to on-the-fly secure computation, however, they become a key building block in Chapter 3 where we construct oblivious transfer extension supporting an on-the-fly setup.

Results in Chapter 3

Secure computation protocols require a large number of oblivious transfers (OTs) [Rab81]. For example, when using garbled circuits [Yao86], the parties need to perform one OT per input bit in the garbled circuit. Even worse, in the GMW multi-party computation protocol [GMW87], each pair of parties need to perform an OT for each AND gate in the circuit, leading to a significant number of OTs per circuit evaluation.

Since OT is closely related to public-key encryption [GKM+00], the computational cost of performing each OT is high and, moreover, requires two or more rounds of interaction to complete. These computation and round complexity costs then directly impact the secure computation protocol.

To address the practical limitations of having to perform a large number of OTs to realize secure computation, the work of Beaver [Bea96] (and the efficient construction of Ishai, Kilian, Nissim, and Petrank [IKNP03]), introduced the notion of OT *extension*. With an OT extension, it becomes possible for two parties to cheaply "extend" a small number of OTs into many (e.g., millions of) OTs using only lightweight symmetric-key operations. These OTs can then be used to instantiate secure computation as before, but with much lower computational complexity; in particular, the *communication and round* complexity now only depends on the OT extension protocol.

While remarkably computationally-efficient constructions of OT extension exist today [RR20,Roy22] (capable of extending many millions of OTs per second), these protocols require *interaction*, both in the setup phase and in the "extension" phase, which have downstream effects on the secure computation protocols in terms of the number of rounds and the communication costs.

Recent work has explored an alternative approach to build *non-interactive* OT extension by replacing the interactive setup with a *public-key* setup [OSY21, BCM+24], where two parties can compute OT extensions locally, using only each other's public keys. This allows any two parties to generate OT extensions *on the fly*—without interaction—by just using their public keys, which mirrors the Diffie–Hellman key-exchange. The state-of-the-art protocols offering such a public-key setup achieve approximately $20,000$ OTs per second [BCM+24], which is still orders of magnitude slower than traditional (interactive) OT extensions. In particular, this performance gap comes from the use of public-key operations per extended OT, which deviates from the classical OT extension [Bea96, IKNP03] protocols where only symmetric-key operations are used. Moreover, all existing approaches offering a public-key setup (barring the trivial approach from spooky encryption) require group-based assumptions, which make them insecure in a post-quantum world.

Contributions. In Chapter 3, we provide a new framework we call QuietOT that results in the first concretely-efficient OT extension protocols with a public-key setup and plausible post-quantum security, resolving the open question of Riahinia [Ria24, Question 4]. In fact, the only prior approach for realizing a plausibly post-quantum-secure construction of OT extension with a public-key setup was the folklore construction from spooky encryption.

In QuietOT, after the initial public-key setup (which can be seen as replacing the "base" OTs with a non-interactive, public-key setup), two parties can perform a practically unbounded number of OTs using only cheap, symmetric-key primitives and only 7 to 33 bits of communication per transferred bit. In contrast, all other approaches require public-key operations for each OT [OSY21, BCM+24] (including the folklore approach via spooky encryption). With QuietOT, we can perform over one million OTs per second on commodity hardware—30 to 100× faster than the state-of-the-art. However, in contrast to communication-efficient approaches—which only incur 3 bits of communication per bit OT—QuietOT incurs slightly more communication: 7 to 33 bits per bit OT. We note that fast OT extension protocols, which do not offer a public-key setup, incur 32 bits or more per bit OT [Roy22], making QuietOT competitive.

Theorem 1.7.3 (Informal)**.** *Assuming the existence of a suitable weak PRF candidate, there exists a concretely-efficient OT extension with a public-key setup and plausible post-quantum security in the random oracle model.*

We realize the public-key setup for QuietOT from the ring learning with errors (RLWE) assumption, which gives us plausible post-quantum security. The public-key setup generalizes to on-the-fly secure computation supporting linear functions (e.g., inner products) [OSY21, CZ22, BCM+24], which we believe may be of independent interest.

Lemma 1.7.1 (Informal)**.** *Under the RLWE assumption, there is a public-key setup protocol for the OT extension protocol from Theorem 1.7.3.*

As an immediate application of QuietOT, we show that it can be used to reduce the communication overhead in multi-party computation protocols where parties engage in pairwise secure computations by allowing parties to reuse their public keys across computations (cf. the discussion in Section 1.3.1). As an independent contribution, we formalize the notion of *multi-instance security* for OT extension with a public-key setup—a crucial property that was overlooked in prior work and essential for secure composition within larger protocols.

Theorem 1.7.4 (Informal)**.** *QuietOT enables communication-efficient pairwise secure computation with a large number of parties.*

QuietOT also yields the first two-round OT extension protocol from a nearly-black-box use of symmetric-key primitives. In particular, QuietOT makes black-box use of an inner-product membership PRF [BCM+24], a variant of a standard PRF which is not implied by a black-box use of a one-way function (in contrast to standard PRFs), but is nonetheless a "Minicrypt"-style [Imp95] assumption with many candidates fitting the criteria. As such, QuietOT partially circumvents the impossibility result of Garg, Mahmoody, Masny, and Meckler [GMMM18] for black-box two-round OT extension from black-box use of one-way functions.

Theorem 1.7.5 (Informal)**.** *Assuming the existence of an inner-product membership PRF, there exists a two-round OT extension protocol in the random oracle model.*

Connection to on-the-fly secure computation. The results in Chapter 3 directly pertain to on-the-fly secure computation by allowing two parties to generate OT instances non-interactively using only each other's public keys; *à la* Diffie–Hellman. Indeed, this model

is captured by an on-the-fly secure computation restricted to computing OT instances and features the same useful properties discussed in Section 1.3.

Open questions. The results in Chapter 3 raise the following open questions:

Question 5. Is it possible to construct an OT extension protocol with a public-key setup, plausible post-quantum security, and optimal communication overhead?

The main limitation of the results in Chapter 3 is that we pay a small amount of extra communication per OT (e.g., 7 bits per OT instead of the optimal 3 bits per OT). Silent OT extension protocols ("silent" refers to optimal communication [BCG+19b]) that also offer a public-key setup (e.g., [OSY21, BCM+24]) currently require public-key operations per OT and are not post-quantum secure. Therefore, managing to find a way to reduce the communication overhead in QuietOT is an interesting open problem.

Question 6. Is it possible to construct a maliciously-secure OT extension protocol with a public-key setup?

The QuietOT framework is realized in the semi-honest model, which is also the model of prior work for OT extension protocols with a public-key setup. An important open problem is constructing a maliciously-secure protocol while still keeping the public-key setup and efficiency features.

Question 7. Is it possible to construct an OT extension protocol with a public-key setup that is as concretely-efficient as traditional OT extension protocols?

Even though QuietOT can generate a million OT extensions per second, it is still roughly 30× slower compared to traditional OT extension protocols (which have no public-key setup). Closing this gap, by either further optimizing QuietOT or finding alternative approaches, is an interesting open problem.

Bibliographic notes

The results in Part I are based on the following works:

- The results in Chapter 2 are based on "Constrained Pseudorandom Functions for Inner-Product Predicates from Weaker Assumptions" [SS24a], a single-author paper that appeared at Asiacrypt 2024.
- The results in Chapter 3 are based on "QuietOT: Lightweight Oblivious Transfer with a Public-Key Setup" [CDD+24], a joint work with Geoffroy Couteau, Lalita Devadas, Srinivas Devadas, and Alexander Koch that appeared at Asiacrypt 2024.

Artifact evaluations. The open-source implementations [SS24b, SS24c] associated with the results from Chapters 2 and 3 underwent artifact evaluations at Asiacrypt 2024 to receive "Artifact Available," "Artifact Functional," and "Artifact Reproducible" badges.

1.7.2 Results in Part II

The second part of this thesis consists of two chapters (Chapters 4 and 5) and covers new tools with theoretical applications by introducing new approaches to on-the-fly secure computation without using spooky encryption.

Chapter 4 introduces the notion of a distributed point function with a non-interactive setup, which has important applications in secure computation, and which we realize from a variety of standard assumptions.

Chapter 5 formalizes the notion of multi-key homomorphic secret sharing and presents the first construction of such a primitive from a group-based assumption, opening a new approach for on-the-fly secure computation for a large class of functions.

Results in Chapter 4

Distributed point functions (DPFs) [GI14, BGI15] have become a fundamental building block in modern secure computation, enabling efficient protocols for private information retrieval [WYG+17, SSLD22], secure database queries [DFL+20, DRPS22], and private analytics [BBC+21, MPD+24, MST24, RZCGP24]. Their significance is particularly evident in multi-party computation protocols, where they enable communication-efficient generation of correlated randomness [BCGI18, SGRR19, BCG+19b] and serve as key components in applications ranging from distributed oblivious RAM [Ds17, VHG23] to private machine learning [RTPB22, YJG+23, JGB+24], making them essential for both theoretical advances and practical implementations in cryptography.

However, classical DPFs are designed to operated in a "trusted dealer" regime where a trusted party generates the shares of the DPF and distributes them to the evaluating parties. In many applications of DPFs, there is no trusted dealer entity, and two parties need to generate the shares themselves by emulating the dealer using a secure computation protocol. Using state-of-the-art protocols for this task requires multiple rounds [Ds17, BGIK22], and the only alternative approach is to use spooky encryption.

Contributions. In Chapter 4, we initiate the study of non-interactive distributed point functions (NIDPFs), which replace the trusted dealer entity with a non-interactive setup. This follows a similar paradigm to how we replaced the interactive oblivious transfers with a public-key setup in Chapter 3. Two immediate applications of NIDPFs are to enable on-the-fly secure computation for a variety of functions, including the equality function and pseudorandom correlation functions. Both of these applications are important to building communication-efficient secure computation protocols.

In addition to formalizing the notion of NIDPFs, we give constructions from a variety of standard cryptographic assumptions including the decisional composite residuosity (DCR) assumption, the symmetric external Diffie–Hellman (SXDH) assumption in pairing groups, the quadratic residuosity (QR) assumption, the enhanced DDH (EDDH) assumption in class groups, and a strong variant of the LWE assumption. These constructions achieve a total communication of $O(N^{2/3})$ for domain size N, which results in sublinear communication in the domain size (a necessary feature for DPFs).

Theorem 1.7.6 (Informal)**.** *There exists an NIDPF scheme achieving $O(N^{2/3})$ communication for domain size N, ignoring polynomial factors in the security parameter, under either (1) the DCR assumption, (2) the QR assumption, (3) the EDDH assumption in class groups, or (4) the SXDH assumption in pairing groups.*

As an independent contribution, we show that our techniques for building NIDPFs generalize to *succinct* on-the-fly secure computation for a restricted class of functions, which

is of independent interest. In particular, we show that a generalization of our NIDPF construction yields the following theorem:

Theorem 1.7.7 (Informal)**.** *Let $\mathcal{P}$ be the set of constant-degree polynomials. There exists an on-the-fly secure computation protocol, with input-succinct communication in the first round, for computing functions of the form $P(x, f(y))$, where $P \in \mathcal{P}$ and $f \in \mathsf{NC}^1$, one party has a (large) input x, and the other party has a (short) input y. The communication in the first round is $o(|x|) + O(|y|)$, ignoring polynomial factors in the security parameter.*

Indeed, Theorem 1.7.7 captures flavors of on-the-fly secure computation that becomes the topic of study in both Chapter 5 and in Chapter 6, and were not previously known from any assumption—not even using spooky encryption. Elaborating, in Chapter 5, we show how to build on-the-fly secure computation for functions in the class NC^1 but without *succinctness* in one of the parties' inputs. And in Chapter 6, we show how to build on-the-fly secure computation for *all* functions (from stronger assumptions compared to Chapter 5) with full succinctness in one of the parties' inputs. As such, the generalized construction from Chapter 4 lies somewhere in between the two results and, interestingly, can be realized from a wider range of assumptions.

Open questions. The results in Chapter 4 raise the following open questions:

Question 8. Is it possible to construct concretely-efficient NIDPFs?

The constructions of NIDPFs in Chapter 4 are currently only of theoretical interest since they require a large number of public-key operations. This is in contrast to standard DPFs which can be realized from cheap, symmetric-key primitives. Finding new constructions of NIDPFs that are concretely efficient is a fascinating open problem with important real-world applications. However, resolving this open problem will likely necessitate new techniques.

Question 9. Can the $O(N^{2/3})$ communication cost be brought down to $O(\log N)$?

Existing *interactive* protocols for setting up a DPF between two parties require optimal $O(\log N)$ communication (ignoring factors in the security parameter). This makes the $O(N^{2/3})$ communication overhead of our NIDPF constructions suboptimal and rather unnatural, which suggests it should be possible to improve succinctness with new techniques.

Question 10. Can the generalization to succinct on-the-fly computation be extended to support a larger class of functions under the same group-based assumptions?

The generalization in Theorem 1.7.7 is restricted to an "unbalanced" class of functions supporting any function in NC^1 for one party's input but only constant-degree polynomials for the other party's input. Finding ways to balance the class of functions that can be computed on each input, or otherwise extend the class of functions supported by this generalization, are interesting open questions. We note that this question is resolved in Chapter 6 under stronger assumptions, namely the learning with errors assumption or assuming the existence of indistinguishability obfuscation.

Results in Chapter 5

Homomorphic secret sharing (HSS) [BGI16] enables parties to locally evaluate functions over secret-shared inputs, serving as a distributed analogue of fully homomorphic encryption

(FHE) [Gen09] (cf. Section 1.4). While HSS schemes have found numerous applications in cryptography, including secure computation with sublinear communication and private information retrieval (see Pierre Meyer's PhD thesis [Mey23] and Mahshid Riahinia's PhD thesis [Ria24] for detailed discussions on many applications of HSS), existing constructions require a correlated setup between the parties. This setup requirement impacts the practical utility of HSS, particularly in downstream applications.

Contributions. In Chapter 5, we formalize the notion of *multi-key* homomorphic secret sharing (MKHSS), which eliminates the need for any correlated setup. In particular, given only a common reference string (CRS), any two parties can directly secret share their inputs with one another and perform local computations, similar to how multi-key FHE [LTV12, MW16] extends standard FHE to the multi-party setting without requiring a prior setup. While MKHSS is readily implied by spooky encryption [DHRW16], constructing it from non-lattice assumptions presents significant technical challenges and was an open problem prior to our work work described in this thesis.

Our approach results in the first MKHSS constructions from a group-based assumption supporting NC^1 computation, specifically the decisional composite residuosity (DCR) assumption, which we capture in the following informal theorem:

Theorem 1.7.8 (Informal)**.** *Under the DCR assumption, there exists a two-party, MKHSS scheme for computing functions in the class* NC^1*.*

Our construction of MKHSS has several immediate applications to secure computation and beyond. Most notably, it immediately implies a sublinear, two-round, two-party secure computation protocol for NC^1 circuits from non-lattice assumptions, resolving a long-standing (folklore) open problem of realizing secure computation in two rounds and sublinear communication without resorting to multi-key FHE.

Theorem 1.7.9 (Informal)**.** *Under the DCR assumption, there exists a sublinear, two-round, two-party secure computation protocol for* NC^1 *computations.*

We also show that our MKHSS construction can be used to build the first attribute-based non-interactive key exchange supporting NC^1 predicates in the standard model. This significantly generalizes password-based non-interactive key exchange, allowing parties to derive a shared key if their secret attributes satisfy a public predicate. Unlike many interactive constructions of key exchange, that often suffer from subtle security flaws [JRX24], our non-interactive approach yields a conceptually simple construction with a straightforward argument for security.

Theorem 1.7.10 (Informal)**.** *Under the DCR assumption, there exists an attribute-based non-interactive key exchange protocol supporting predicates in* NC^1*.*

Finally, we show how MKHSS enables the first public-key pseudorandom correlation functions (PCFs) for any NC^1 correlation from non-lattice assumptions. This leads to significant improvements in silent multi-party computation, achieving $O(p)$ communication complexity in the preprocessing phase for p-party computation. This improves upon all previous protocols not based on spooky encryption, which required $\Omega(p^2)$ communication.

Theorem 1.7.11 (Informal)**.** *Let C be an arithmetic circuit with n inputs, s multiplication gates, and m outputs, instantiated over a ring $\mathcal{R}$. Under the DCR assumption, for any number of parties p, there exists a p-party secure computation protocol for computing C in the preprocessing model, with the following communication complexity:*

- *In the preprocessing phase: $O(p)$ bits in a single broadcast round.*
- *In the online phase: $p \cdot (2s + m)$ ring elements.*

The protocol is secure against a passive adversary corrupting any strict subset of parties.

Open questions. The results in Chapter 5 raise a number of open questions:

Question 11. Are there practical applications for MKHSS restricted to low-depth circuits?

In Chapter 5, we show that the construction is potentially implementable and perhaps concretely practical for low-depth computations (requiring roughly 100ms per multiplication gate in the circuit). An interesting question, therefore, is finding concrete applications where a low-depth circuit is sufficient. We believe that implementing the attribute-based non-interactive key exchange to support *fuzzy* password-based key exchange (e.g., where the "password" is a biometric) could be one such application.

Question 12. Is it possible to construct a multi-party *MKHSS scheme?*

The recent work of Dao, Ishai, Jain and Lin [DIJL23] provides an elegant construction of multi-party (not multi-key) HSS from the sparse learning parity with noise assumption. All prior HSS schemes, barring spooky encryption, were only suitable for a small number of parties (typically only supporting two parties). As such, finding ways of potentially extending their approach to also be multi-key is an interesting open question. In particular, such a result would immediately imply a sublinear, two-round multi-party computation, which we still do not know how to achieve without using multi-key FHE [MW16].

Bibliographic notes

The results in Part II are based on the following works:

- The results in Chapter 4 are based on "Non-Interactive Distributed Point Functions" [BDSS25], a joint work with Elette Boyle and Lalita Devadas, which is to appear at PKC 2025.
- The results in Chapter 5 are based on "Multi-key Homomorphic Secret Sharing" [CDH+25], a joint work with Geoffroy Couteau, Lalita Devadas, Aditya Hegde, and Abhishek Jain, which is to appear at Eurocrypt 2025.

1.7.3 Results in Part III

The third part of this thesis consists of Chapter 6 and explores the limits of secure computation by showing that it is possible to approach the optimal *insecure* solution in terms of rounds and communication complexity, even when one party has a very large input.

Results in Chapter 6

In a two-party secure computation setting where one party has a large input, constructing protocols with minimal communication and rounds is challenging. The natural insecure protocol—where the party with small input sends it to the party with the large input, who then computes the function and broadcasts the result—achieves optimal communication, but clearly fails to achieve privacy for the senders input. This raises a fundamental question: can we design secure protocols that preserve the communication efficiency, round complexity, and non-interactive disclosure properties of this insecure protocol? The answer to this question is not even known from powerful primitives like spooky encryption.

Contributions. In Chapter 6, we introduce simultaneous-message and succinct (SMS) secure computation. In an SMS scheme, two parties, where one party has a large input, simultaneously exchange encoded values in the first round. Following this exchange, they can locally decode secret shares of the result. The key requirements are: simultaneity, succinctness, and additive reconstruction.

Simultaneity requires the parties to exchange messages simultaneously, where the only prior setup allowed is a common reference string. Succinctness requires the communication to be sublinear in the large party's input length (which also implies succinctness in the circuit description). And additive reconstruction ensures that parties can publicly disclose the output in the second round.

Even with a primitive as powerful as spooky encryption, it was not known how to construct a protocol satisfying all three of these requirements. In particular, while spooky encryption gives succinctness in the circuit description, the total communication still grows with the size of the inputs, which SMS does not allow. As such, the formalization of SMS captures the frontier of secure computation.

In Chapter 6, we show that we can construct SMS from *standard assumptions* and uncover several interesting connections to a number of powerful cryptographic primitives. Our main results on building SMS are captured in the following informal theorem:

Theorem 1.7.12 (Informal)**.** *Let Alice be the party with the large input X, and Bob be the party with the small input y, and let f be a function in the family supported by the SMS scheme. There exist the following constructions of SMS schemes:*

- *Under the LWE assumption (with superpolynomial modulus-to-noise ratio) there exists an SMS scheme for all efficiently-computable functions, where:*
  - *Alice's encoding is of size $|f(X,y)|^{2/3} \cdot \mathsf{poly}(\lambda)$.*
  - *Bob's encoding is of size $(|y| + |f(X,y)|^{2/3}) \cdot \mathsf{poly}(\lambda)$.*

  I.e., the encodings are fully independent of $|X|$.
- *Assuming sub-exponentially-secure indistinguishability obfuscation, somewhere statistically binding hash functions, and other standard assumptions, there exists an SMS scheme for all efficiently-computable* batch *functions, where for any batch size L,*
  - *Alice's encoding is of size $\mathsf{poly}(\lambda)$.*
  - *Bob's encoding is of size $\mathsf{poly}(\lambda, |f|, \log L)$.*

I.e., the encodings are at most logarithmically dependent on the batch size but can be linear in the function description.

Applications of SMS. We show that SMS schemes have several immediate applications. We give:

(1) The first trapdoor hash functions (TDH) [DGI+19] for all circuits from LWE, significantly extending prior constructions of TDH which were limited to linear functions.

(2) A generic compiler from any FHE scheme to a rate-1 FHE scheme.

(3) A generic compiler from an SMS scheme to correlation-intractable (CI) hash functions for efficiently-searchable relations. CI hash functions, in turn, have applications to zero-knowledge proofs and more (see Alex Lombardi's PhD thesis [Lom22]).

Finally, we also show that our $i\mathcal{O}$-based construction of SMS for batch functions can be used to realize a two-round secure computation protocol that is succinct in the output length, giving an alternative approach to the construction of Hubáček–Wichs protocol [HW15].

Theorem 1.7.13 (Informal)**.** *Under the LWE assumption (with a superpolynomial modulus-to-noise ratio) there exists (1) a trapdoor hash scheme for all circuits, (2) any FHE scheme can be generically transformed into a rate-1 FHE scheme using SMS, and (3) there exists a generic transformation from SMS to correlation-intractable hash functions for efficiently-searchable relations. Moreover, under sub-exponentially-secure $i\mathcal{O}$, there exists a two-round secure computation protocol with communication sublinear in the output length.*

Open questions. The results in Chapter 6 raise the following open questions:

Question 13. Is it possible to improve the dependence on the output length in Theorem 1.7.12 from $\epsilon = 2/3$ to arbitrary ϵ?

Our construction of SMS from LWE achieves succinctness (in the first round) in both the input and output length. However, the dependence on the output length is only modestly sublinear; improving the succinctness in the output length is an interesting open problem. We note that Question 13 is related to Question 9, since the underlying tools, limiting succinctness to $\epsilon = 2/3$, are the same in both cases.

Question 14. Is it possible to construct an SMS scheme for all functions from $i\mathcal{O}$?

Our construction of SMS from $i\mathcal{O}$ only supports batch functions, where the same function is applied over a (large) batch of inputs. An important open question is whether it is possible to construct SMS for all functions, similarly to the LWE-based approach. In particular, one of the differences between the LWE-based construction and the $i\mathcal{O}$-based construction of SMS, is that the function is fixed at encoding time in the former but not the latter. This would make an $i\mathcal{O}$-based construction supporting all functions potentially more powerful compared to our LWE-based scheme. Therefore, the dual question is asking whether there are there barriers to achieving a "function-adaptive" SMS scheme for all functions.

Question 15. What other connections exist between SMS schemes and existing cryptographic primitives?

In Chapter 6, we already uncover some powerful applications and connections (cf. Theorem 1.7.13). An intriguing open problem is finding more connections between the SMS abstraction and other cryptographic primitives. In particular, TDH supporting *linear predicates* has seen numerous applications across cryptography (e.g., [JK20, JJ21, HJKS22, DGKV22, BDSZ24]); understanding what power is gained from TDH supporting all efficiently-computable functions is an exciting open question.

Question 16. Can our constructions of SMS schemes be extended to a multi-party setting?

Our constructions are focused on the two-party setting. This is the natural scenario to consider when there is one party with a large input. In particular, it is information-theoretically impossible to construct an on-the-fly secure computation protocol with succinctness in two or more inputs simultaneously. However, it *is* conceivable to imagine a scenario where many parties have small inputs and only one party has a large input. In this case, it is possible that our techniques can be extended to a multi-party setting via a similar approach to the one used to extend spooky encryption from two parties to multiple parties [DHRW16].

Bibliographic notes

The results in Chapter 6 are based on "Simultaneous-Message and Succinct Secure Computation" [BJSSS25], a joint work with Elette Boyle, Abhishek Jain, and Akshayaram Srinivasan, which is to appear at Eurocrypt 2025.

Part I

New Tools with Practical Applications

Chapter 2

Constrained Pseudorandom Functions for Inner-Product Predicates

Summary

In this chapter, we provide a novel framework for constructing constrained pseudorandom functions (CPRFs) with inner-product constraint predicates, using ideas from subtractive secret sharing and related-key-attack security.

Our framework can be instantiated using a random oracle or any suitable related-key-attack (RKA) secure pseudorandom function. This results in three new CPRF constructions:

(1) An adaptively-secure construction in the random oracle model.

(2) A selectively-secure construction under the decisional Diffie–Hellman (DDH) assumption.

(3) A selectively-secure construction with a polynomial domain under the assumption that one-way functions exist.

All three instantiations are constraint-hiding and support inner-product predicates, leading to the first constructions of such expressive CPRFs under each corresponding assumption. Moreover, while the OWF-based construction is primarily of theoretical interest, the random oracle and DDH-based constructions are concretely efficient, which we show via an open-source implementation.

2.1 Introduction

Constrained pseudorandom functions (CPRFs) [BW13, KPTZ13, BGI14] are pseudorandom functions (PRFs) with a "default mode" associated with a master key msk, and a "constrained mode" associated with a constrained key csk defined over a predicate C. The constrained key csk can be used to compute the same "default mode" value of the PRF for all inputs x where $C(x) = 0$. However, for all inputs x where $C(x) \neq 0$, the constrained key csk can only be used to compute a pseudorandom value that is computationally independent of the PRF value under msk.

In the basic definition of CPRFs, the constrained key csk can reveal the predicate C (i.e., all inputs x where $C(x) = 0$). For example, the GGM PRF [GGM86], admits *puncturing* constraints [BW13, KPTZ13, BGI14], where the constraint C is a point function that outputs 0 on all-but-one input. In the GGM PRF, csk reveals the punctured point to the constraint key holder. An enhanced definition of CPRFs, first formalized by Boneh, Lewi, and Wu [BLW17] (PKC 2017), requires csk to hide C, and is much more challenging to achieve, even for simple constraints [BLW17, CC17, DKN+20].

Constructing CPRFs for expressive constraint classes under standard assumptions has proven to be a challenging task. Several constructions exist for simple constraint classes, such as prefix-matching, bit-fixing, and constraints expressible by t-CNF formulas (with constant t) under various assumptions, including the minimal assumption that one-way functions exist (see the excellent survey of related works in [DKN+20, Appendix A]). However, even slightly more expressive constraints, such as constraints represented by inner products, constant-degree polynomials, or circuits in NC^1 (the class of functions computable by logarithmic-depth circuits), appear to be much more challenging to construct from standard assumptions [CC17, AMN+18, CVW18, CMPR23].

In a recent work, Couteau, Meyer, Passelègue, and Riahinia [CMPR23] (Eurocrypt 2023) were able to realize CPRFs for NC^1 (but without the constraint-hiding property) from the Decision Composite Residuosity (DCR) assumption, as well as *constraint-hiding* CPRFs with inner-product constraint predicates, through an elegant connection to homomorphic secret sharing [BGI16, BCG+17, BKS19, OSY21]. In contrast, *constraint-hiding* CPRFs for NC^1 are only known under LWE [CC17, CVW18, PS18] (or indistinguishability obfuscation [CRV16, BLW17]) and can even *imply* indistinguishability obfuscation in certain cases [CC17]. Therefore, the result of Couteau et al. [CMPR23] significantly pushes the constraint expressivity of CPRFs under the Decisional Composite Residuosity (DCR) assumption. Prior to their result, the only known constructions for constraint-hiding CPRFs with sufficiently powerful constraint predicates to evaluate inner-product constraints required either the learning with errors (LWE) assumption or non-standard assumptions [BLW17, CC17, PS18]. However, in contrast to other constraint predicates that can be realized from one-way functions [BW13, KPTZ13, BGI14, DKN+20], there is still a significant gap in our understanding of which assumptions are necessary for realizing CPRFs for more expressive constraint classes, such as inner-product and NC^1 predicates.

Motivation. In this chapter, we revisit the assumptions required to construct constraint-hiding CPRFs for inner-product constraint classes. This is motivated by the existence of CPRFs for NC^1 from Diffie–Hellman-style assumptions [AMN+18], as well as constraint-hiding

CPRFs for bit-fixing and (constant sized) t-CNF formulas from the minimal assumption that one-way functions exist [DKN+20]. Understanding what assumptions are required to realize sufficiently expressive CPRFs can shed light on realizing closely related "high-end" cryptographic primitives such as functional encryption [CC17], searchable symmetric encryption [BLW17], attribute-based encryption [AMN+18], and even obfuscation [CC17]. Specifically, in this chapter, we ask the following question.

Under what assumptions do constrained PRFs with inner-product predicates exist?

The motivation for studying inner-product constraints is that they can be used to construct CPRFs with constraint predicates represented by constant-degree polynomials and extensions thereof (see Section 2.8 for details), and are of interest both as a theoretical object and as a practical tool (see Chapter 3).

From a theoretical lens, the fact that inner-product predicates lie somewhere in between t-CNF and NC^1 predicates in terms of expressivity, motivates the study of CPRFs for inner-product predicates under weaker assumptions, with the goal of potentially finding new techniques that could lead to more expressive constraints under weaker assumptions. This was also the motivation behind Attrapadung et al. [AMN+18] and other works examining the assumptions required to build CPRFs. Indeed, Davidson et al. [DKN+20] prove that CPRFs for inner-product predicates imply CPRFs for constant t-CNFs predicates (see [DKN+20, Appendix C] and Section 2.8), which in turn imply CPRFs for bit-fixing predicates.

From a practical perspective, the current lack of any *concretely efficient* CPRF constructions for inner-product predicates,[1] motivates the quest of finding assumptions under which efficient constructions can be realized. This is especially motivated by the hope that concretely efficient constructions of CPRFs for inner-product predicates will lead to interesting real-world applications, as has been the case for the concretely efficient constructions of CPRFs admitting puncturing constraints (e.g., [RW14, BMO17, SYL+18, SGRR19, BCG+20a, HK21, BBMHS22, MZRA22, BBD+23, LP23, Fen23]).

Contributions. In this chapter, we make the following three contributions:

A simple framework. We provide a simple framework that exploits the properties of subtractive secret sharing to construct CPRFs for inner-product predicates. Our framework makes explicit several ideas that have been used implicitly in many prior works on CPRFs (e.g., [BV15, BTVW17, AMN+18, PS18]), and may prove useful in obtaining more results in the future.

New constructions from new assumptions. We construct the first CPRFs for inner-product predicates with (1) adaptive security in the random oracle model, (2) selective security under the decisional Diffie–Hellman (DDH) assumption, and (3) selective security with a polynomial input domain under the minimal assumption that one-way functions (OWFs) exist. All three of our results push the frontier of what was previously known theoretically on CPRFs. Moreover, our constructions are all constraint-hiding by default.

An implementation. Due to the simplicity of our building blocks, we show that our constructions result in the first *practical* constraint-hiding CPRFs under standard assumptions.

[1] To the best of our knowledge, no constraint-hiding CPRF constructions have been implemented to date.

We implement and benchmark our constructions, proving that they are concretely efficient. (All prior constructions of CPRFs for inner-product predicates, including the DCR-based construction of Couteau et al. [CMPR23], require computationally-expensive machinery, making them impractical.)

Applications and extensions. Our framework has the following applications and extensions:

(1) *More complex predicates.* From inner-product constraints, we can build CPRFs for more complex predicates via generic transformations, including constraints represented by *constant degree polynomials* and for the "AND" of d distinct inner-product predicates. In particular, the latter allows us to construct matrix-product constraint predicates, where the constraint is satisfied if and only if $\mathbf{Ax} = \mathbf{0}$, for a constraint matrix $\mathbf{A}$ and input vector $\mathbf{x}$.

(2) *Lower bounds in learning theory.* In learning theory, Membership Query (MQ) learning provides a model for quantifying the "learnability" or complexity of a certain class of functions [Val84]. Informally, in the MQ learning framework, a learner gets oracle access to a function and must approximate the function after making a sufficient number of queries. Cohen et al. [CGV15] introduce a model they call MQ *with Restriction Access* ($\mathrm{MQ}_{\mathsf{RA}}$), where in addition to black-box membership queries, the learner obtains non-black-box access to a restricted subset of the function. Obtaining (negative) results on the learnability of a particular class in the $\mathrm{MQ}_{\mathsf{RA}}$ model can be done using a connection to constrained PRFs.

We discuss these applications further in Section 2.8.

2.1.1 Related work

In Table 2.1, we summarize known constructions of CPRFs for inner-product predicates (including existing constructions for more general predicates such as NC^1 and $\mathsf{P/poly}$) and highlight our results.

CPRFs for inner-product predicates. Attrapadung et al. [AMN+18] construct constrained PRFs for NC^1 (which includes inner-product predicates) from the L-decisional Diffie–Hellman inversion (L-DDHI) in combination with DDH over the quadratic residue subgroup QR_p (they can make their construction adaptively-secure by using a random oracle instead of DDH in QR_p), but their construction is not constraint-hiding. Similarly, Couteau et al. [CMPR23] shows how to construct CPRFs for NC^1 predicates from the DCR assumption through homomorphic secret sharing (but also fail to achieve constraint privacy). Couteau et al. additionally show that their techniques can be used to construct a CPRF from DDH *with a polynomially-bounded input domain.* CPRFs for more general predicates are known from multi-linear maps [BW13, BKM17], indistinguishability obfuscation [BZ14, HKW15, BLW17, HKKW19, AMN+19, DKN+20], and LWE [BV15, CC17, BTVW17, CVW18, PS18], and can be used to instantiate CPRFs with inner-product constraints under those assumptions.

Constraint-hiding CPRFs for inner-product predicates. Davidson et al. [DKN+20] (Crypto 2020) construct (weakly) constraint hiding CPRFs for inner-product predicates from the LWE assumption. Specifically, their construction satisfies a weaker privacy definition, in

	Assumption	Security	Hiding	Predicate	Practical	Comments
	LWE	Selective	✓ / ✗	$\supseteq \mathsf{NC}^1$	✗	
[AMN+18]	L-DDHI	Selective	✗	NC^1	✗	L-DDHI in $\mathsf{QR}_p \wedge$ DDH in $\mathbb{G}$
[AMN+18]	L-DDHI	Adaptive	✗	NC^1	✗	L-DDHI in $\mathsf{QR}_p \wedge$ ROM
[DKN+20]	LWE	Adaptive	✗	IP	✗	Is *weakly* constraint hiding
[CMPR23]	DCR	Selective	✓	IP	✗	
[CMPR23]	DDH	Selective	✓	IP	✗	Polynomial input domain
Theorem 2.4.1	ROM	Adaptive	✓	IP	✓	
Theorem 2.5.2	DDH	Selective	✓	IP	✓	
Theorem 2.5.4	VDLPN	Selective	✓	IP	✗	Only for *weak* CPRFs
Theorem 2.6.3	OWF	Selective	✓	IP	✗	Polynomial input domain

Table 2.1: Related work on CPRFs for Inner-Product (IP) predicates from standard assumptions.
ROM = Random Oracle Model.
DDH = Decisional Diffie–Hellman assumption.
DCR = Decisional Composite Residuosity assumption.
L-DDHI = L-decisional Diffie–Hellman Inversion assumption.
VDLPN = Variable-density Learning Parity with Noise assumption [BCG+20a].

which the adversary does not get access to an evaluation oracle. Constraint-hiding CPRFs for more general predicates (that include inner-product predicates) are known from the LWE assumption [BTVW17, CC17, CVW18, PS18] and indistinguishability obfuscation [BLW17]. To the best of our knowledge, Couteau et al. [CMPR23] are the first realize constraint-hiding CPRFs for inner-product predicates from a non-lattice assumption, specifically from DCR.

One-one CPRFs. Our framework (as well as some prior constructions of CPRFs [AMN+18, DKN+20, CMPR23]) shares some conceptual similarities to the construction of one-one constrained PRFs [PTW20]—an information-theoretic primitive that can be viewed as a CPRF in the "no-evaluation security" model [AMN+18], with applications to conditional disclosure of secrets. However, their constructions cannot be used to realize the standard notion of CPRFs from standard assumptions.

2.1.2 Organization

In Section 2.2, we provide a technical overview highlighting the main ideas behind our framework and constructions. In Section 2.3, we cover the necessary preliminaries on CPRFs and RKA-secure PRFs. In Section 2.4, we present an adaptively-secure CPRF construction in the random oracle model. In Section 2.5, we give a general framework for building selectively-secure CPRFs that we instantiate from RKA-secure PRFs, without the need for a random oracle. In Section 2.6, we show how to instantiate our framework from one-way functions. In Section 2.7, we discuss some generic extensions to our framework. In Section 2.8, we provide some applications. In Section 2.9, we discuss the practical efficiency of our constructions and benchmark our implementation.

2.2 Technical Overview

In this section, we provide an overview of our framework and constructions.

Background on CPRFs. Following prior works [BTVW17, CMPR23], for PRF domain $\mathcal{X}$ and a constraint $C\colon \mathcal{X} \to \{0,1\}$, we write $C(x) = 0$ for "true" (authorized), and $C(x) \neq 0$ for "false" (unauthorized). CPRFs consist of a master secret key msk, which can be used to evaluate the PRF on *all* inputs in the domain. From msk, it must then be possible to efficiently sample a constrained key csk for a given constraint C, which can be used to evaluate the PRF on all inputs x in the domain where $C(x) = 0$. Constraint hiding CPRFs have the added property that C remains hidden given csk. See Section 2.3 for formal definitions.

2.2.1 Our approach

We now explain the main technical ideas that underpin our framework for constructing CPRFs for inner-product predicates. We start by explaining how we can use the idea of subtractive secret sharing to construct a constraint predicate C for inner-product predicates, inspired by Couteau et al. [CMPR23].

The power of subtractive secret sharing. Subtractive secret shares of a value s, which we denote by s_0 and s_1, have the property that $s_0 - s_1 = s$ (over $\mathbb{Z}$). By splitting s into two random shares s_0 and s_1, individually each share is independent of the secret s. To use subtractive secret sharing to construct CPRFs, the main idea is to exploit the *symmetry* between the two shares. Specifically, consider what happens when the secret s is zero. Because we have that $s_0 - s_1 = 0$, it follows that $s_0 = s_1$. This symmetry present in subtractive secret shares has enabled many efficient techniques for distributed computations [GI14, BGI15, BGI16, BCG+17, BCG+19b, BKS19, BCG+20b, OSY21], and surprisingly, also applies to CPRFs [CMPR23]. Specifically, consider the inner-product constraint $C_{\mathbf{z}}$ parameterized by a vector $\mathbf{z}$ and defined as $C_{\mathbf{z}}(\mathbf{x}) = \langle \mathbf{z}, \mathbf{x} \rangle$. Next, denote subtractive secret shares of the constraint vector $\mathbf{z}$ by $\mathbf{z_0}$ and $\mathbf{z_1}$, such that $\mathbf{z_0} - \mathbf{z_1} = \mathbf{z}$. Thanks to the aforementioned symmetry property, for all input vectors $\mathbf{x}$:

- If $\langle \mathbf{z}, \mathbf{x} \rangle = 0$ (i.e., $C_{\mathbf{z}}(\mathbf{x}) = 0$, authorized), then $\langle \mathbf{z_0}, \mathbf{x} \rangle = \langle \mathbf{z_1}, \mathbf{x} \rangle$, and
- If $\langle \mathbf{z}, \mathbf{x} \rangle \neq 0$ (i.e., $C_{\mathbf{z}}(\mathbf{x}) \neq 0$, unauthorized), then $\langle \mathbf{z_0}, \mathbf{x} \rangle \neq \langle \mathbf{z_1}, \mathbf{x} \rangle$.

In words, the constraint is satisfied if and only if both shares of the inner product are equal. Moreover, note that $\mathbf{z_1}$ can be sampled *after* $\mathbf{z_0}$, because $\mathbf{z_0}$ is a random value independent of the "secret" constraint $\mathbf{z}$. We now describe how we can use these properties of subtractive secret sharing to construct a CPRF.

Initial attempt (not secure). Our first idea, which unfortunately turns out to be not secure, is to let the master secret key $\mathsf{msk} = \mathbf{z_0}$, for a random $\mathbf{z_0}$. Then, for a given constraint vector $\mathbf{z}$, the constrained key is computed "on the fly" as $\mathsf{csk} = \mathbf{z_1}$, where $\mathbf{z_1} = \mathbf{z_0} - \mathbf{z}$. The intuition is that for all $\mathbf{x}$ where $\langle \mathbf{z}, \mathbf{x} \rangle = 0$ (i.e., for all authorized $\mathbf{x}$), both the master secret key and the constrained key can be used to derive *the same* key k. Specifically, we can simply let $k = \langle \mathbf{z_0}, \mathbf{x} \rangle = \langle \mathbf{z_1}, \mathbf{x} \rangle$. Using the key k, in conjunction with any PRF F, we can define the output of the evaluation on the input $\mathbf{x}$ to be $F_k(\mathbf{x})$. Additionally, for all $\mathbf{x}$ where $\langle \mathbf{z}, \mathbf{x} \rangle \neq 0$

(i.e., for all unauthorized $\mathbf{x}$), the master key and constrained key derive *different* PRF keys, which results in the constrained key outputting a pseudorandom value.

Unfortunately, while this initial attempt provides the necessary correctness properties, it is not secure for the following two reasons:

(1) the CPRF adversary, knowing the constraint $\mathbf{z}$ and given $\mathbf{z_1}$ can trivially recover $\mathbf{z_0}$ (the master secret key) simply by computing $\mathbf{z_0} = \mathbf{z_1} + \mathbf{z}$, and

(2) in the case where $\langle \mathbf{z}, \mathbf{x} \rangle \neq 0$, the derived key is still *related* to the master key msk, in that $\langle \mathbf{z_1}, \mathbf{x} \rangle = \langle \mathbf{z_0}, \mathbf{x} \rangle - \langle \mathbf{z}, \mathbf{x} \rangle$.

Couteau et al. [CMPR23] resolves these two issues by resorting to HSS. In particular, they only use the value of $\langle \mathbf{z}, \mathbf{x} \rangle$ (which each party can compute a share of given $\mathbf{z_0}$ and $\mathbf{z_1}$, respectively) as a conditional mask in a HSS computation that computes a PRF. As such, they require evaluating a PRF inside of HSS which makes their construction impractical. This is where our approach diverges from the one of Couteau et al. [CMPR23], which we explain next.

Second attempt (secure). To fix our initial attempt, we must first prevent the adversary from recovering the full vector $\mathbf{z_0}$ (the master secret key) from the constrained key $\mathbf{z_1}$, while still guaranteeing the necessary property that $\langle \mathbf{z_0}, \mathbf{x} \rangle = \langle \mathbf{z_1}, \mathbf{x} \rangle$ whenever $\langle \mathbf{z}, \mathbf{x} \rangle = 0$. To achieve this, we exploit the linearity of inner products. Specifically, let $\mathbb{F}$ be a finite field of order at least 2^λ, for a security parameter λ. As before, we let $\mathsf{msk} := \mathbf{z_0}$, for a random $\mathbf{z_0} \in \mathbb{F}^\ell$. However, now we let $\mathsf{csk} := \mathbf{z_1}$, where $\mathbf{z_1} := \mathbf{z_0} - \Delta\mathbf{z}$, for a random scalar "shift" $\Delta \in \mathbb{F}$. Notice that when $\langle \mathbf{z}, \mathbf{x} \rangle = 0$,

$$\langle \mathbf{z_0}, \mathbf{x} \rangle = \langle \mathbf{z_0}, \mathbf{x} \rangle - \Delta \langle \mathbf{z}, \mathbf{x} \rangle = \langle \mathbf{z_0}, \mathbf{x} \rangle - \underbrace{\langle \Delta\mathbf{z}, \mathbf{x} \rangle}_{\substack{\text{By linearity of} \\ \text{inner products}}} = \langle \mathbf{z_1}, \mathbf{x} \rangle ,$$

which still guarantees that the master secret key and constrained key can be used to derive the same PRF key k whenever $C(\mathbf{x}) = 0$. Moreover, because Δ is uniformly random over $\mathbb{F}$ (which has order at least 2^λ), $\mathbf{z_1}$ cannot be used to recover $\mathbf{z_0}$, even with knowledge of the constraint $\mathbf{z}$, thereby preventing the CPRF adversary from recovering the master secret key msk from the constrained key csk. Indeed, if the constraint vector is non-zero, there will be at least one coordinate that is scaled by Δ preventing the adversary from recovering the full master key vector.

Now, with the random shift Δ, we ensure that the constrained key csk does not leak the full master secret key. However, we are still left with the second problem we identified in our initial attempt: the derived PRF keys are still *related* to the master secret key, which does *not* guarantee that the resulting PRF evaluation is pseudorandom to the adversary. To deal with this, we can use a random oracle to instantiate a "correlation-resistant" PRF.

Construction in the random oracle model. One simple way to instantiate the CPRF with correlated keys is to instantiate the PRF with a random oracle H. This forms the basis for our first construction, which we describe in Section 2.4. In a nutshell, we show that, if we use the derived key $k = \langle \mathbf{z_1}, \mathbf{x} \rangle$ with a random oracle H as the PRF, then the construction $F_k(\mathbf{x}) := H(k, \mathbf{x})$ is a secure CPRF. Specifically, the random oracle ensures that

each evaluation is uniformly random, while still guaranteeing both the master secret key and the constrained key derive the same k when the constraint is satisfied.

Removing the random oracle with an RKA-secure PRF. To remove the random oracle requirement, we show that we can use a "special" PRF that remains provably secure when evaluated with different *related* keys. Such PRFs are known as Related-Key-Attack (RKA) secure PRFs [Bih94, BK03] and have been studied extensively [BK03, GL10, BC10, GOR11, BLMR13, ABPP14, AW14, LMR14, CLWG19, BCG+20a], yielding several constructions to choose from. This result is rather surprising, since prior works that require notions of correlation-robustness (e.g., [IKNP03, KS08, PSZ18]) could only be constructed from more powerful assumptions. In contrast, we show that constructing CPRFs with inner-product constraints requires a much weaker flavor of correlation-robustness satisfied by RKA-secure PRFs with affine key-derivation functions. In particular, this weaker notion of correlation-robustness can be instantiated unconditionally leading to our one-way function based CPRF construction in Section 2.6.

Suitable RKA-secure PRFs. As we have informally shown above, a fully "RKA-secure" PRF can be realized with a random oracle to remove correlations in the keys. However, constructions of RKA-secure PRFs exist from several standard assumptions. These constructions achieve security against adversaries that can adaptively query the PRF when keyed on arbitrary functions of the secret key. In particular, we require RKA-security against *affine* functions of the key (see Section 2.3 for definitions), which is a stronger notion compared to standard RKA-security against *additive* functions that is often considered in the literature. The affine function requirement eliminates many RKA-secure PRF constructions (e.g., [BK03, BC10, GL10, BLMR13, AW14, LMR14, CLWG19]), leaving us only with the DDH-based RKA-secure PRF for affine functions of Abdalla et al. [ABPP14].

The DDH-based RKA-secure PRF forms the basis for our first instantiation in the standard model. However, we also show that we can use any (weak) PRF[2] that is RKA-secure against additive functions to instantiate our framework and obtain a (weak) CPRF for inner-product predicates. In particular, this allows us to use the VDLPN-based RKA-secure (weak) PRF of Boyle et al. [BCG+20a].

Additionally, we show that we can adapt the one-way function based RKA-secure PRF of Applebaum and Widder [AW14] to instantiate our framework (under certain restrictions). Specifically, the PRF of Applebaum and Widder [AW14] is only secure against additive functions and requires the number of related keys that the adversary queries to be apriori bounded by some polynomial t (in the security parameter). While these restriction makes their RKA-secure PRF construction have limited applications elsewhere, we find that it is just sufficiently powerful to apply to our framework provided that we bound the magnitude of the input vectors to be polynomial in t and restrict the CPRF to a polynomially-sized domain. However, a problem is that their construction is only proven RKA-secure for *additive* functions of the key, which is not suitable to instantiate our framework. Fortunately, however, we can easily adapt their result to the case of *affine* functions, making it compatible with our framework and leading to the first CPRF construction for inner-product predicates in Minicrypt.

[2] A weak PRF is secure if the adversary only queries it on random inputs.

2.3 Preliminaries

2.3.1 Notation

We let $\mathbb{F}$ denote a finite field (e.g., integers mod p), $\mathbb{Z}$ denote the set of integers, and $\mathbb{N}$ denote the set of natural numbers. We let $\mathbb{F}^*$ denote the set $\mathbb{F} \setminus \{0\}$. We denote by $\mathsf{poly}(\cdot)$ the set of all polynomials and by $\mathsf{negl}(\cdot)$ any negligible function. We occasionally abuse notation and let poly denote a fixed polynomial. For a PRF family $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ and input $x \in \mathcal{X}$, we write $F_k(x)$ to mean $F(k, x)$.

Vectors and matrices. A vector $\mathbf{v} = (v_1, \ldots, v_n)$ is denoted using bold lowercase letters. Scalar multiplication with a vector is denoted $a\mathbf{v} = (av_1, \ldots, av_n)$ and the inner product between two vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted $\langle \mathbf{a}, \mathbf{b} \rangle$.

Sampling and assignment. We let $x \xleftarrow{R} S$ denote a uniformly random sample drawn from a set S. We let $x \leftarrow \mathcal{A}$ denote assignment from a randomized algorithm $\mathcal{A}$ and $x := y$ denote initialization of x to the value of y (which may be the output of a deterministic algorithm).

Efficiency and indistinguishability. By an *efficient* algorithm $\mathcal{A}$ we mean that $\mathcal{A}$ is modeled by a (possibly non-uniform) Turing Machine that runs in probabilistic polynomial time. We write $D_0 \approx_c D_1$ to mean that two distributions D_0 and D_1 are *computationally* indistinguishable to all efficient distinguishers $\mathcal{D}$ and $D_0 \approx_s D_1$ to mean that D_0 and D_1 are *statistically* indistinguishable.

2.3.2 Constrained pseudorandom functions

We start by recalling the syntax and properties of constrained pseudorandom functions (CPRFs). For simplicity, we restrict the definition to 1-key, constraint-hiding CPRFs, which is the definition satisfied by our constructions. We point to Boneh, Lewi, and Wu [BLW17] for a more general definition of constraint-hiding CPRFs (i.e., with polynomial-key security).

Definition 2.3.1 (Constrained Pseudorandom Functions; Adapted from [BLW17, CMPR23])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. A Constrained Pseudorandom Function (CPRF) with key space $\mathcal{K} = \mathcal{K}_\lambda$, domain $\mathcal{X} = \mathcal{X}_\lambda$, and range $\mathcal{Y}$, that supports constraints represented by the class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where $C_\lambda \colon \mathcal{X} \to \{0, 1\}$, consists of the following four algorithms.*

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{msk}$. *The randomized key generation algorithm takes as input a security parameter λ. Outputs a master secret key $\mathsf{msk} \in \mathcal{K}$.*
- $\mathsf{Eval}(\mathsf{msk}, x) \to y$. *The deterministic evaluation algorithm takes as input the master secret key msk and input $x \in \mathcal{X}$. Outputs $y \in \mathcal{Y}$.*
- $\mathsf{Constrain}(\mathsf{msk}, C) \to \mathsf{csk}$. *The randomized constrain algorithm takes as input the master secret key msk and a constraint circuit $C \in \mathcal{C}$. Outputs a constrained key csk.*
- $\mathsf{CEval}(\mathsf{csk}, x) \to y$. *The deterministic constrained evaluation algorithm takes as input the constrained key csk and an input $x \in \mathcal{X}$. Outputs $y \in \mathcal{Y}$.*

We let any auxiliary public parameters PP *be an implicit input to all algorithms. A CPRF must satisfy the following correctness and security properties.*

Correctness. *For all security parameters* λ*, all constraints* $C \in \mathcal{C}$*, and all inputs* $x \in \mathcal{X}$ *such that* $C(x) = 0$ *(authorized), it holds that:*

$$\Pr\left[\mathsf{Eval}(\mathsf{msk}, x) = \mathsf{CEval}(\mathsf{csk}, x) \quad : \quad \begin{array}{r} \mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{csk} \leftarrow \mathsf{Constrain}(\mathsf{msk}, C) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

(1-key, adaptive) Security. *A CPRF is (1-key, adaptively)-secure if for all efficient adversaries* $\mathcal{A}$*, the advantage of* $\mathcal{A}$ *in the following security experiment* $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},b}(\lambda)$ *is negligible in* λ*. Here,* b *denotes the challenge bit.*

1. ***Setup:*** *On input* 1^λ*, the challenger runs* $\mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda)$*, initializes the set* $Q := \emptyset$*, and runs* $\mathcal{A}(1^\lambda)$*.*
2. ***Pre-challenge queries:*** $\mathcal{A}$ *adaptively sends arbitrary inputs* $x \in \mathcal{X}$ *to the challenger. For each* x*, the challenger computes* $y := \mathsf{Eval}(\mathsf{msk}, x)$*, sends* y *to* $\mathcal{A}$*, and proceeds to update* $Q := Q \cup \{x\}$*.*
3. ***Constrain query:*** $\mathcal{A}$ *sends* one *constraint* $C \in \mathcal{C}$ *to the challenger. The challenger computes* $\mathsf{csk} \leftarrow \mathsf{Constrain}(\mathsf{msk}, C)$*, and sends* csk *to* $\mathcal{A}$*.*
4. ***Challenge query:*** *For the single challenge query,* $\mathcal{A}$ *sends input* $x^* \in \mathcal{X}$ *as its challenge query, subject to the restriction that* $x^* \notin Q$ *and* $C(x^*) \neq 0$*. If* $b = 0$*, the challenger computes* $y^* := \mathsf{Eval}(\mathsf{msk}, x^*)$*. Else, if* $b = 1$*, the challenger samples* $y^* \xleftarrow{R} \mathcal{Y}$*. The challenger sends* y^* *to* $\mathcal{A}$*.*
5. ***Post-challenge queries:*** $\mathcal{A}$ *continues to adaptively query the challenger on inputs* $x \in \mathcal{X}$*, subject to the restriction that* $x \neq x^*$*. For each* x*, the challenger computes* $y := \mathsf{Eval}(\mathsf{msk}, x)$ *and sends* y *to* $\mathcal{A}$*.*
6. ***Guess:*** $\mathcal{A}$ *outputs its guess* b'*, which is the output of the experiment.*

$\mathcal{A}$ *wins if* $b' = b$*, and its advantage* $\mathsf{Adv}^{\mathsf{sec}}_{\mathcal{A}}(\lambda)$ *is defined as*

$$\mathsf{Adv}^{\mathsf{sec}}_{\mathcal{A}}(\lambda) := \left| \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},1}(\lambda) = 1] \right|,$$

where the probability is over the randomness of $\mathcal{A}$ *and* KeyGen*.*

Definition 2.3.2 (Constraint Privacy; Adapted from [BLW17, CMPR23])**.** *A CPRF is (1-key, adaptive)-constraint-hiding if for all efficient adversaries* $\mathcal{A}$*, the advantage of* $\mathcal{A}$ *in the following security experiment* $\mathsf{Exp}^{\mathsf{priv}}_{\mathcal{A},b}(\lambda)$ *is negligible in* λ*. Here,* b *denotes the challenge bit.*

1. ***Setup:*** *On input* 1^λ*, the challenger runs* $\mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda)$*, initializes the set* $Q := \emptyset$*, and runs* $\mathcal{A}(1^\lambda)$*.*
2. ***Pre-challenge queries:*** $\mathcal{A}$ *adaptively sends arbitrary input values* $x \in \mathcal{X}$ *to the challenger. For each* x*, the challenger computes* $y := \mathsf{Eval}(\mathsf{msk}, x)$*, sends* y *to* $\mathcal{A}$*, and proceeds to update* $Q := Q \cup \{x\}$*.*

3. ***Constrain query:*** *$\mathcal{A}$ sends a* pair *of constraints $(C_0, C_1) \in \mathcal{C}^2$ to the challenger,* subject to the restriction that *$C_0(x) = C_1(x)$, for all $x \in Q$. The challenger computes $\mathsf{csk}^* \leftarrow \mathsf{Constrain}(\mathsf{msk}, C_b)$, and sends csk^* to $\mathcal{A}$.*
4. ***Post-challenge queries:*** *$\mathcal{A}$ adaptively sends arbitrary input values $x \in \mathcal{X}$ to the challenger, subject to the restriction that $C_0(x) = C_1(x)$. For each x, the challenger computes $y := \mathsf{Eval}(\mathsf{msk}, x)$, and sends y to $\mathcal{A}$.*
5. ***Guess:*** *$\mathcal{A}$ outputs its guess b', which is the output of the experiment.*

$\mathcal{A}$ wins if $b' = b$ and its advantage $\mathsf{Adv}^{\mathsf{priv}}_{\mathcal{A}}(\lambda)$ is defined as

$$\mathsf{Adv}^{\mathsf{priv}}_{\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{priv}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{priv}}_{\mathcal{A},1}(\lambda) = 1]\right|,$$

where the probability is over the randomness of $\mathcal{A}$ and KeyGen.

Definition 2.3.3 ((1-key, selective) Security)**.** *A CPRF as defined in Definition 2.3.1 is said to be (1-key,* selectively*)-secure if the adversary commits to the constraint C* before *querying the challenger [BLW17]. That is, $\mathcal{A}$ sends the constraint C to the challenger before issuing any pre-challenge queries. The same applies to the constraint-privacy definition (Definition 2.3.2).*

Remark 3 (Unique evaluation queries)**.** *Without loss of generality, we can restrict the PRF adversary $\mathcal{A}$ to issuing only* unique *evaluation queries (as was also done in prior PRF formalizations [AW14, AMN+18]). Note that the adversary is already restricted to a unique challenge query in the above definition.*

2.3.3 RKA-secure PRFs

Here, we formalize the notion of related-key attack (RKA)-secure PRFs.

Remark 4 (Find-then-guess security)**.** *We slightly modify the standard defintion of RKA-secure PRFs (e.g., [BK03]) to better align with the syntax of constrained PRFs. In the basic definition, the adversary does not obtain evaluation queries from what is guaranteed to be the output of the PRF F on* some *key. However, we note that this extra evaluation oracle is without loss of generality, and is only added to syntactically simplify our proofs. This definition is known as the find-then-guess PRF security game [CMPR23, Definition 10] and implies the real-or-random PRF security game, albeit with a polynomial loss in security.*

Definition 2.3.4 (Φ-restricted Adversaries)**.** *An efficient RKA-PRF adversary $\mathcal{A}$ is said to be Φ-restricted if its oracle queries have a related-key derivation function ϕ chosen arbitrarily from a set of valid key derivation functions Φ.*

Definition 2.3.5 (Related-Key-Attack Secure PRFs [BK03])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter and $\ell = \ell(\lambda) \in \mathsf{poly}(\lambda)$. Let $\mathcal{F} = \{F_k : \mathcal{X}_\lambda \to \mathcal{Y}\}_{k \in \mathcal{K}_\lambda}$ be a family of functions and $\Phi \colon \mathcal{K}_\lambda \to \mathcal{K}_\lambda$ be a family of related-key derivation functions. $\mathcal{F}$ is said to be an RKA-secure PRF family if for all efficient Φ-restricted adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following security experiment $\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},b}(\lambda)$ is negligible in λ. Here, b denotes the challenge bit.*

1. ***Setup:*** *On input* 1^λ*, the challenger samples* $k \xleftarrow{R} \mathcal{K}_\lambda$*, initializes the set* $Q := \emptyset$*, and runs* $\mathcal{A}(1^\lambda)$*.*
2. ***Pre-challenge queries:*** *For each query* (ϕ, x)*, the challenger computes* $y := F_{\phi(k)}(x)$*, sends* y *to* $\mathcal{A}$*, and proceeds to update* $Q := Q \cup \{(\phi, x)\}$*.*
3. ***Challenge query:*** *For the single challenge query* (ϕ^*, x^*)*,* subject to the restriction that $(\phi^*, x^*) \notin Q$*, the challenger proceeds based on the bit* b *as follows. If* $b = 0$*, the challenger computes* $y^* := F_{\phi^*(k)}(x^*)$*. If* $b = 1$*, the challenger samples* $y^* \xleftarrow{R} \mathcal{Y}$*. The challenger then sends* y^* *to* $\mathcal{A}$*.*
4. ***Post-challenge queries:*** *For each query* (ϕ, x)*,* subject to the restriction that $(\phi, x) \neq (\phi^*, x^*)$*, the challenger computes* $y := F_{\phi^*(k)}(x)$*, and sends* y *to* $\mathcal{A}$*.*
5. ***Guess:*** $\mathcal{A}$ *outputs its guess* b'*, which is the output of the experiment.*

$\mathcal{A}$ *wins if* $b' = b$ *and its advantage* $\mathsf{Adv}^{\mathsf{rka}}_{\mathcal{A}}(\lambda)$ *is defined as*

$$\mathsf{Adv}^{\mathsf{rka}}_{\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},1}(\lambda) = 1]\right|,$$

where the probability is over the randomness of $\mathcal{A}$ *and choice of* k*.*

Definition 2.3.6 (Affine Related-Key Derivation Functions [ABPP18])**.** *Let* $\mathcal{R}$ *be a finite ring and let* $m \geq 1$ *be an integer, let the class* Φ_{aff} *(*aff *for affine) denote the class of functions from* $\mathcal{R}^m$ *to* $\mathcal{R}^m$ *that can be separated into* m *component functions consisting of degree-1 univariate polynomials. That is,*

$$\Phi_{\mathsf{aff}} := \left\{ \boldsymbol{\phi} : \mathcal{R}^m \to \mathcal{R}^m \;\middle|\; \begin{array}{l} \boldsymbol{\phi} = (\phi_1, \ldots, \phi_m); \\ \forall i \in [m],\ \phi_i(k_i) = \gamma_i k_i + \delta_i,\ \gamma_i \neq 0 \end{array} \right\}.$$

Note that $\gamma_i \neq 0$ *is necessary to make the derivation function non-trivial. The definition of Abdalla et al. [ABPP18] uses* $\mathbb{Z}_p$*; here we generalize it to any ring* $\mathcal{R}$*.*

Remark 5. *Note that* Φ_{aff} *captures additive and multiplicative relations, which we denote by* $\Phi_+ \subset \Phi_{\mathsf{aff}}$ *and* $\Phi_\times \subset \Phi_{\mathsf{aff}}$*, respectively.*

2.4 Adaptive Construction in the Random Oracle Model

In this section, we present a basic construction using a random oracle. In Section 2.5.1, we extend this construction into a general framework based on RKA security, which we use in conjunction with RKA-secure PRFs to realize CPRFs for inner-product predicates under DDH, VDLPN, and OWFs.

The main idea is to instantiate the high-level construction presented in Section 2.2 with a random oracle $H \colon \mathcal{K} \times \mathbb{F}^\ell \to \mathcal{Y}$ taking the role of the PRF. Doing so ensures that when $\langle \mathbf{z}, \mathbf{x} \rangle \neq 0$, the output is uniformly random and independent of the constrained key csk, which guarantees that the evaluation under msk is independent of csk. We prove the following theorem.

Adaptive CPRF for Inner-Products from a Random Oracle

Public Parameters. Let λ be a security parameter, $\ell \geq 1$ be an integer, and $\mathbb{F}$ be a finite field of order at least 2^λ. Define a key space $\mathcal{K}$, a range $\mathcal{Y}$, and a suitable choice of efficiently computable deterministic function $\mathsf{map}\colon \mathbb{F} \to \mathcal{K}$. Let $H\colon \mathcal{K} \times \mathbb{F}^\ell \to \mathcal{Y}$ be a function modeled by a random oracle.

$\mathsf{KeyGen}(1^\lambda, \ell)$:

1 : $k_0 \xleftarrow{R} \mathbb{F}$
2 : $\mathbf{z_0} \xleftarrow{R} \mathbb{F}^\ell$
3 : $\mathsf{msk} := (k_0, \mathbf{z_0})$

$\mathsf{Constrain}(\mathsf{msk}, \mathbf{z})$:

1 : **parse** $\mathsf{msk} = (k_0, \mathbf{z_0})$
2 : $\Delta \xleftarrow{R} \mathbb{F}^*$
3 : $\mathbf{z_1} := \mathbf{z_0} - \Delta\mathbf{z}$
4 : **return** $\mathsf{csk} := (k_0, \mathbf{z_1})$

$\mathsf{Eval}(\mathsf{msk}, \mathbf{x})$:

1 : **parse** $\mathsf{msk} = (k_0, \mathbf{z_0})$
2 : $\delta_\mathbf{x} := \langle \mathbf{z_0}, \mathbf{x} \rangle$
3 : $k := \mathsf{map}(k_0 + \delta_\mathbf{x})$
4 : **return** $H(k, \mathbf{x})$

$\mathsf{CEval}(\mathsf{csk}, \mathbf{x})$:

1 : **parse** $\mathsf{csk} = (k_0, \mathbf{z_1})$
2 : $\delta_\mathbf{x} := \langle \mathbf{z_1}, \mathbf{x} \rangle$
3 : $k := \mathsf{map}(k_0 + \delta_\mathbf{x})$
4 : **return** $H(k, \mathbf{x})$

Figure 2.1: Adaptive CPRF construction in the random oracle model.

Theorem 2.4.1. *Let λ be a security parameter, $\ell \geq 1$ be any integer, $\mathbb{F}$ be a finite field of order at least 2^λ, and $\mathcal{K} = \mathcal{K}_\lambda$ be a key space. The construction in Figure 2.1 is a (1-key, adaptively-secure, constraint-hiding) CPRF in the random oracle model.*

Proof. We prove each required property in turn.

Correctness. Correctness follows from the intuition presented in Section 2.2. For all constraints $\mathbf{z}$ and inputs $\mathbf{x}$, whenever $\langle \mathbf{z}, \mathbf{x} \rangle = 0$, we have that

$$\delta_{\mathbf{x}} = \langle \mathbf{z_0}, \mathbf{x} \rangle = \langle \mathbf{z_0}, \mathbf{x} \rangle + \langle \mathbf{z}, \mathbf{x} \rangle = \langle \mathbf{z_0}, \mathbf{x} \rangle + \langle \Delta \mathbf{z}, \mathbf{x} \rangle = \langle \mathbf{z_1}, \mathbf{x} \rangle .$$

Therefore, Eval and CEval in Figure 2.1 compute the same key k, because both Eval and CEval add the same shift $\delta_{\mathbf{x}}$ to the starting key k_0. It then follows that the evaluation is identical under the master key and the constrained key given that F_k is deterministic.

(1-key, adaptive) Security. Our proof consists of a sequence of hybrid games.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the (1-key, adaptive) CPRF security game. We note that here, the challenger provides an oracle $\mathcal{O}_H$ via which the adversary $\mathcal{A}$ queries the random oracle H.

- *Hybrid* $\mathcal{H}_1$. In this hybrid game, we place the following restrictions on the adversary: (1) Each query issued by $\mathcal{A}$ to the challenger (including queries to $\mathcal{O}_H$) must be unique, and (2) *after* issuing the constraint query, the adversary is only allowed to query the pre- and post-challenge oracles on constrained inputs.

  These restrictions are without loss of generality in the 1-key setting, and have been used in prior work (e.g., [AMN+18, DKN+20]). It follows that $\mathcal{A}$'s advantage in $\mathcal{H}_1$ is identical to its advantage in $\mathcal{H}_0$.

- *Hybrid* $\mathcal{H}_2$. In this hybrid game, the challenger starts by sampling responses to the pre-challenge evaluation queries until the constraint query is issued. That is, for a bound q_0 on the number of pre-challenge queries issued by $\mathcal{A}$, the challenger samples $v_1, \ldots, v_{q_0} \xleftarrow{R} \mathcal{Y}$. Then, the challenger responds to the i-th pre-challenge query $\mathbf{x_i}$ with v_i, and programs the random oracle $\mathcal{O}_H$ to output v_i on all future queries r where it holds that $r = k_0 + \langle \mathbf{z_0}, \mathbf{x_i} \rangle$, for some $i \in [q_0]$. After the constrain query, the challenger responds to pre-challenge queries as in $\mathcal{H}_1$.

  Claim. *$\mathcal{A}$'s advantage in $\mathcal{H}_2$ is at most $\mathsf{negl}(\lambda)$ larger compared to $\mathcal{H}_1$.*

  Proof. Let q_H be the total number of random oracle queries issued by the adversary prior to the constrain query, and let $\{r_i\}_{i \in [q_0]}$ be the queries to $\mathcal{O}_H$. We define the event bad_0 as:

  $$\exists (i, j) \in [q_H] \times [q_0] \text{ such that } r_i = (k_0 + \langle \mathbf{z_0}, \mathbf{x_j} \rangle, \mathbf{x_j}) \wedge H(r_i) \neq v_j.$$

  The event bad_0 corresponds to the case where the adversary happens to query the random oracle $\mathcal{O}_H$ on a "bad" input r_i *prior* to the challenger programming $\mathcal{O}_H$ to output v_i, causing the response to be inconsistent with respect to $\mathcal{H}_1$. For any given

pre-challenge query $\mathbf{x_j}$, issued before the constrain query, the probability that $\mathcal{A}$ issues an r_i query to $\mathcal{O}_H$ such that $r_i = (k_0 + \langle \mathbf{z_0}, \mathbf{x_j} \rangle, \mathbf{x_j})$ is equivalent to the probability of guessing k_0, which is bounded by $\frac{1}{|\mathbb{F}|}$. Hence, by a union bound, we get that

$$\Pr_{k_0 \xleftarrow{R} \mathbb{F}} [\mathsf{bad}_0] \leq \frac{q_H \cdot q_0}{|\mathbb{F}|} \leq \frac{q_H \cdot q_0}{2^\lambda} \leq \mathsf{negl}(\lambda),$$

which bounds $\mathcal{A}$'s advantage in $\mathcal{H}_1$ to a negligible function in λ. □

- *Hybrid $\mathcal{H}_3$.* In this hybrid game, we swap the definition of the constrained key and master key. Specifically, in this game, the challenger responds to $\mathcal{A}$'s constrain query $\mathbf{z}$ by sampling $\mathbf{z_1} \xleftarrow{R} \mathbb{F}^\ell$ and sending back $\mathsf{csk} = \mathbf{z_1}$. The challenger then samples $\Delta \xleftarrow{R} \mathbb{F}$, computes $\mathbf{z_0} = \mathbf{z_1} + \Delta \mathbf{z}$, and responds to future evaluation queries using $\mathbf{z_0}$ as the master key.

  Claim. *$\mathcal{A}$'s advantage in $\mathcal{H}_3$ is equivalent to its advantage in $\mathcal{H}_2$.*

  Proof. This change is purely syntactic and therefore does not affect the distribution of the keys. In particular, note that in $\mathcal{H}_2$, all evaluation queries prior to the constrain query are sampled independently of the master key. As such, it can be sampled at the time of the constrain query. □

- *Hybrid $\mathcal{H}_4$.* In this hybrid game, the challenger samples $w_1, \ldots, w_{q_1} \xleftarrow{R} \mathcal{Y}$ as the responses to the q_1 pre- and post-challenge evaluation queries issued following the constrain query. Then, the challenger responds to $\mathcal{A}$'s i-th evaluation query $\mathbf{x_i}$, where $\langle \mathbf{z}, \mathbf{x_i} \rangle \neq 0$ (recall the restriction in $\mathcal{H}_0$), with w_i.

  Claim. *$\mathcal{A}$'s advantage in $\mathcal{H}_4$ is at most $\mathsf{negl}(\lambda)$ larger compared to $\mathcal{H}_3$.*

  Proof. Here, we let q_H be a bound on the total number of random oracle queries issued by the adversary throughout the game and let q_1 be a bound on the number of pre- and post-challenge evaluation queries issued *after* the constrain query. We then define the event bad_1 as:

$$\exists (i, j) \in [q_H] \times [q_1] \text{ such that } r_i = (k_0 + \langle \mathbf{z_0}, \mathbf{x_j} \rangle, \mathbf{x_j}) \wedge H(r_i) \neq w_j,$$

  where r_i is a query to $\mathcal{O}_H$ issued by $\mathcal{A}$ and each $\mathbf{x_j}$ is constrained by assumption. The event bad_1 corresponds to the case where the adversary happens to query $\mathcal{O}_H$ on an input corresponding to a constrained evaluation under the master key msk, causing the response to be inconsistent with respect to the distribution in $\mathcal{H}_3$. For all post-constraint evaluation queries $\mathbf{x_j}$, where $j \in [q_1]$, define $y_j = H(k_0 + \langle \mathbf{z_0}, \mathbf{x_j} \rangle, \mathbf{x_j})$, which is computed identically to a post-constraint evaluation response in hybrid $\mathcal{H}_3$. We claim that y_j is computed independently of the constrained key $\mathsf{csk} = \mathbf{z_1}$. To see this, note that we can equivalently express y_j in terms of $\mathbf{z_1}$ as $y_j = H(k_0 + \langle \mathbf{z_1}, \mathbf{x_j} \rangle + \Delta \langle \mathbf{z}, \mathbf{x_j} \rangle, \mathbf{x_j})$, where

$\langle \mathbf{z}, \mathbf{x_j} \rangle \neq 0$ by assumption. Then, because Δ is uniformly random and independent of $\mathbf{z_1}$ in $\mathcal{H}_3$, each y_j is computed using a random oracle H that is "seeded" by $\Delta \langle \mathbf{z}, \mathbf{x_j} \rangle$, which makes the response independent of $\mathbf{z_1}$. Then, to compute the probability of the event bad_1, over the choice of $\Delta \in \mathbb{F}$, we can apply a union bound over all q_1 post-constraint evaluation queries issued by $\mathcal{A}$ to get

$$\Pr_{\Delta \xleftarrow{R} \mathbb{F}}[\mathsf{bad}_1] \leq \frac{q_H \cdot q_1}{|\mathbb{F}|} \leq \frac{q_H \cdot q_1}{2^\lambda} = \mathsf{negl}(\lambda),$$

which bounds the adversary's advantage in $\mathcal{H}_4$ to a negligible function in λ. □

- *Hybrid* $\mathcal{H}_5$. In this hybrid game, the challenger samples a uniformly random key k to answer the challenge query when the challenge bit is set to $b = 0$.

  Claim. *$\mathcal{A}$'s advantage in $\mathcal{H}_5$ is equivalent to its advantage in $\mathcal{H}_4$.*

  Proof. Note that all evaluation queries in $\mathcal{H}_4$ are sampled independently of Δ. Therefore, Δ is only used by the challenger in $\mathcal{H}_4$ to respond to the challenge query, which is equivalent to sampling a uniformly random and independent key k to answer the challenge query $\mathbf{x}^*$, given that $k_0 + \langle \mathbf{z_0}, \mathbf{x}^* \rangle = k_0 + \langle \mathbf{z_1}, \mathbf{x}^* \rangle + \Delta \langle \mathbf{z}, \mathbf{x}^* \rangle$ is a uniformly random value in $\mathbb{F}$. □

At this point, because H is a random oracle, the response to the challenge query is equivalent to sampling a uniformly random value, making $\mathcal{A}$'s advantage in $\mathcal{H}_5$ exactly equal to 0. Taking into account the other hybrids, we conclude that $\mathcal{A}$'s overall advantage in winning the CPRF security game is therefore negligible.

Remark 6. *An alternative proof strategy is to employ the framework of Attrapadung et al. [AMN+19] and show that $H(k_0 + \langle \mathbf{z_0}, \mathbf{x} \rangle, \mathbf{x})$ is a* no-evaluation *secure CPRF (similar to the CPRF game but the adversary does not get access to an evaluation oracle). They prove that any no-evaluation secure and "collision-resistant" CPRF becomes adaptively secure in the ROM when the output is passed through a random oracle. However, this then necessitates making the construction of the form $H'(H(k_0 + \langle \mathbf{z_0}, \mathbf{x} \rangle, \mathbf{x}))$ or arguing why $H'(H(\cdot))$ is equivalent to $H(\cdot)$ in the ROM. We opt here to prove adaptive security directly for completeness.*

Constraint Privacy. We must prove that for all $\mathbf{z}$ and $\mathbf{z}'$ provided by the adversary $\mathcal{A}$, the constrained key, and all evaluation and challenge queries, do not reveal whether the constraint $\mathbf{z}$ or $\mathbf{z}'$ is used by the challenger.

First, we begin by noting that, even given $(\mathbf{z}, \mathbf{z}', \Delta)$, $\mathbf{z_0} + \Delta\mathbf{z}$ is distributed identically to $\mathbf{z_0} + \Delta\mathbf{z}'$ because $\mathbf{z_0}$ is uniformly random and independent of $\mathbf{z}$ and $\mathbf{z}'$. Therefore, the constrained key, absent the evaluation queries, is efficiently simulatable regardless of the constraint chosen by the challenger.

Now, we must show that this remains the case even when the adversary is given access to the evaluation and challenge oracles. Observe that we can proceed via the same sequence of hybrids used in the security proof. Note that in the game defined in Hybrid $\mathcal{H}_5$, we can view

each constrained query as being answered using a uniformly random key $k_i \in \mathcal{K}$. As such, the evaluation queries on constrained inputs (including the challenge query) are independent of the constraint, which guarantees that $\mathcal{A}$ cannot distinguish between $\mathbf{z}$ and $\mathbf{z}'$ with better than negligible advantage.

This concludes the proof of constraint privacy and the proof of the theorem. ■

Remark 7 (Replacing the random oracle with a correlated-input secure hash)**.** *As noted by several prior works (e.g., [IKNP03, GOR11, DGI+19]), the random oracle model is an overkill when all that is required is a notion of "correlation-robustness." Specifically, in our case, all we require is that H removes specific types of correlations present in its inputs. With this in mind, we can replace the random oracle H with a correlated-input secure hash (CIH) function [IKNP03, GOR11, AMN+18, DGI+19]. At a high level, a CIH is a publicly parameterized function H whose outputs "look random and independent" to a computationally-bounded adversary, even when the inputs are correlated. Specifically, we require the CIH to be secure against* affine *correlations between the inputs. The proof of security for Theorem 2.4.1 then follows the same blueprint, but instead hinges on the correlated-input security of H to ensure that the outputs are computationally indistinguishable from uniform. Unfortunately, we are not aware of an adaptively-secure CIH function construction (to the best of our knowledge, all existing constructions are in the selective-security regime). However, we note that there exist strong connections between CIH functions and RKA-PRFs, as discussed in-depth by Goyal, O'Neill, and Rao [GOR11].*

2.5 A General Framework and Constructions

In this section, we provide a general framework that we can instantiate using RKA-secure PRFs. In Section 2.5.1, we start by extending the ideas behind the construction from Section 2.4 into a general framework. We then prove that this framework yields selectively-secure constraint-hiding CPRFs from any RKA-secure PRF supporting Φ_{aff} key derivation functions. In Sections 2.5.2 and 2.5.3, we plug in the DDH-based and VDLPN-based RKA-secure PRF constructions into the framework. We defer instantiating the framework with our OWF-based RKA-secure PRF to Section 2.6, as there we must first construct a Φ_{aff}-RKA-secure PRF from OWFs.

2.5.1 Framework

Existing constructions of RKA-secure PRFs (e.g., [BC10, ABPP14, AW14, BCG+20a]) have a key that is a *vector* of n field elements. In particular, unlike the ROM-based construction from Figure 2.1, where the derived key is a single field element, here we will need the keys to be in the vector space $\mathbb{F}^n$ (or subfield thereof). Our framework can be seen as an *extended* version of the ROM-based construction, where we can replace the random oracle with an RKA-secure PRF. At a high level, to accommodate keys that consist of vectors of n elements, the idea is to run the construction from Figure 2.1 (but instantiated with an RKA-secure PRF instead of a random oracle) independently n times to derive a key for each coordinate. We describe the framework in Figure 2.2.

Selective CPRF for Inner-Products from RKA-secure PRFs

Public Parameters. Let λ be a security parameter, $n, \ell \geq 1$ be integers, and $\mathbb{F}$ be a finite field. Define a key space $\mathcal{K}$ and range $\mathcal{Y}$, and a suitable choice of efficiently computable deterministic function $\mathsf{map}\colon \mathbb{F}^n \to \mathcal{K}$. Let $\mathcal{F} = \left\{F_k\colon \mathbb{F}^\ell \to \mathcal{Y}\right\}_{k\in\mathcal{K}}$ RKA-secure PRF family supporting affine key-derivation functions.

$\mathsf{KeyGen}(1^\lambda, \ell)$:

1 : $\mathbf{k_0} \xleftarrow{\text{R}} \mathbb{F}^n$
2 : **foreach** $i \in [n]$:
3 : $\quad \mathbf{z_{0}}_i \xleftarrow{\text{R}} \mathbb{F}^\ell$
4 : $\mathsf{msk} := (\mathbf{k_0}, \mathbf{z_{0}}_1, \ldots, \mathbf{z_{0}}_n)$

$\mathsf{Constrain}(\mathsf{msk}, \mathbf{z})$:

1 : **parse** $\mathsf{msk} = (\mathbf{k_0}, \mathbf{z_{0}}_i, \ldots, \mathbf{z_{0}}_n)$
2 : **foreach** $i \in [n]$:
3 : $\quad \Delta_i \xleftarrow{\text{R}} \mathbb{F}$
4 : $\quad \mathbf{z_{1}}_i := \mathbf{z_{0}}_i - \Delta_i \mathbf{z}$
5 : **return** $\mathsf{csk} := (\mathbf{k_0}, \mathbf{z_{1}}_1, \ldots, \mathbf{z_{1}}_n)$

$\mathsf{Eval}(\mathsf{msk}, \mathbf{x})$:

1 : **parse** $\mathsf{msk} = (\mathbf{k_0}, \mathbf{z_{0}}_i, \ldots, \mathbf{z_{0}}_n)$
2 : **foreach** $i \in [n]$:
3 : $\quad \delta_{\mathbf{x}i} := \langle \mathbf{z_{0}}_i, \mathbf{x} \rangle$
4 : $\boldsymbol{\delta}_\mathbf{x} := (\delta_{\mathbf{x}1}, \ldots, \delta_{\mathbf{x}n})$
5 : $k := \mathsf{map}(\mathbf{k_0} + \boldsymbol{\delta}_\mathbf{x})$
6 : **return** $F_k(\mathbf{x})$

$\mathsf{CEval}(\mathsf{csk}, \mathbf{x})$:

1 : **parse** $\mathsf{csk} := (\mathbf{k_0}, \mathbf{z_{1}}_1, \ldots, \mathbf{z_{1}}_n)$
2 : **foreach** $i \in [n]$:
3 : $\quad \delta_{\mathbf{x}i} := \langle \mathbf{z_{1}}_i, \mathbf{x} \rangle$
4 : $\boldsymbol{\delta}_\mathbf{x} := (\delta_{\mathbf{x}1}, \ldots, \delta_{\mathbf{x}n})$
5 : $k := \mathsf{map}(\mathbf{k_0} + \boldsymbol{\delta}_\mathbf{x})$
6 : **return** $F_k(\mathbf{x})$

Figure 2.2: Selectively-secure CPRF framework using RKA-secure PRFs.

Theorem 2.5.1. *Let $\mathbb{K}$ be a subfield of $\mathbb{F}$ and let the PRF key space $\mathcal{K} = \mathbb{K}^n$. Fix* map *to be any non-trivial ring homomorphism applied component-wise. If $\mathcal{F}$ is a family of RKA-secure pseudorandom functions with respect to affine related key derivation functions Φ_{aff}, as defined in Definition 2.3.6, then Figure 2.2 instantiated with $\mathcal{F}$ is a (1-key, selectively-secure, constraint-hiding) CPRF.*

Proof. We prove the required properties in turn.

Correctness. For all constraints $\mathbf{z}$ and inputs $\mathbf{x}$, whenever $\langle \mathbf{z}, \mathbf{x} \rangle = 0$, we have that $\delta_{\mathbf{x}i} = \langle \mathbf{z}_{\mathbf{0}i}, \mathbf{x} \rangle = \langle \mathbf{z}_{\mathbf{0}i}, \mathbf{x} \rangle + \Delta_i \langle \mathbf{z}, \mathbf{x} \rangle = \langle \mathbf{z}_{\mathbf{0}i}, \mathbf{x} \rangle + \langle \Delta_i \mathbf{z}, \mathbf{x} \rangle = \langle \mathbf{z}_{\mathbf{1}i}, \mathbf{x} \rangle \in \mathbb{F}$. Therefore, the resulting $\boldsymbol{\delta}_{\mathbf{x}}$ (as computed in Eval and CEval of Figure 2.1) is the same. Moreover, this holds for all $i \in [n]$, and because map is a ring homomorphism to a subfield of $\mathbb{F}$, the resulting keys are also identical when $\langle \mathbf{z}, \mathbf{x} \rangle = 0$. It then follows that the PRF evaluation is identical under the master key and the constrained key, because both Eval and CEval add the same $\boldsymbol{\delta}_{\mathbf{x}}$.

(1-key, selective) Security. We prove security by a reduction to the RKA-security of $\mathcal{F}$. Our proof consists of a sequence of hybrid games.

- *Hybrid $\mathcal{H}_0$.* This hybrid consists of the (1-key, selective) CPRF security game.
- *Hybrid $\mathcal{H}_1$.* In this hybrid, the challenger first samples the constrained key and *then* samples the master key. Specifically, at the start of the game, given the constraint $\mathbf{z}$ (we are in the selective security regime), the challenger first samples the constrained key $\mathsf{csk} := (\mathbf{k_0}, \mathbf{z}_{\mathbf{1}1}, \ldots, \mathbf{z}_{\mathbf{1}n})$, where $\mathbf{k_0} \xleftarrow{R} \mathbb{F}^n$ and $\mathbf{z}_{\mathbf{1}i} \xleftarrow{R} \mathbb{F}^\ell$, for all $i \in [n]$. Then, the challenger computes the master secret key as $\mathsf{msk} := (\mathbf{k_0}, \mathbf{z}_{\mathbf{0}1}, \ldots, \mathbf{z}_{\mathbf{0}n})$, where $\mathbf{z}_{\mathbf{0}i} := \mathbf{z}_{\mathbf{1}i} + \Delta_i \mathbf{z}$ and $\Delta_i \xleftarrow{R} \mathbb{F}$, for all $i \in [n]$.

  Claim. *$\mathcal{A}$'s advantage in $\mathcal{H}_1$ is identical to $\mathcal{A}$'s advantage in $\mathcal{H}_0$.*

  Proof. The claim follows immediately by observing that the distribution of msk and csk in $\mathcal{H}_1$ is identical to $\mathcal{H}_0$, because the change is merely syntactic. □

- *Hybrid $\mathcal{H}_2$.* In this hybrid game, the challenger does not sample Δ anymore. Instead, it is given access to the following stateful oracle $\mathcal{O}_{\mathsf{rka}}$:

  Oracle $\mathcal{O}_{\mathsf{rka}}$

  Initialize. Sample $\Delta \xleftarrow{R} \mathbb{K}^n$.

  Evaluation. On input a affine function $\phi \in \Phi_{\mathsf{aff}}$ and $\mathbf{x} \in \mathbb{F}^\ell$, return $F_{\phi(\Delta)}(\mathbf{x})$.

  The challenger is then defined as follows.

  1. **Setup:** On input $(1^\lambda, \mathbf{z})$, $\mathcal{B}$ initializes $Q := \emptyset$, samples csk according to $\mathcal{H}_1$ by sampling $\mathbf{k_0} \xleftarrow{R} \mathbb{F}^n$, and $\mathbf{z}_{\mathbf{1}i} \xleftarrow{R} \mathbb{F}^\ell$, for all $i \in [n]$, and runs $\mathcal{A}$ on input $\mathsf{csk} := (\mathbf{k_0}, \mathbf{z}_{\mathbf{1}1}, \ldots, \mathbf{z}_{\mathbf{1}n})$.

2. **Pre-challenge queries:** For each query $\mathbf{x}$ issued by $\mathcal{A}$, the challenger updates $Q := Q \cup \{\mathbf{x}\}$, then does the following to compute y:

   - Compute $a_i := \mathsf{map}(\langle \mathbf{z}, \mathbf{x} \rangle)$ and $b_i := \mathsf{map}(\mathbf{k_0}_i + \langle \mathbf{z_1}_i, \mathbf{x} \rangle)$, for all $i \in [n]$.
   - Set $\phi\colon \mathbf{u} \mapsto \mathbf{a} \circ \mathbf{u} + \mathbf{b}$ where $\mathbf{a} := (a_1, \ldots, a_n)$ and $\mathbf{b} := (b_1, \ldots, b_n)$, and where $\circ$ denotes the component wise (i.e., Hadamard) product.
   - Query $\mathcal{O}_{\mathsf{rka}}$ on input $(\phi, \mathbf{x})$, and forward the response y to $\mathcal{A}$.
     ▷ Note that y is computed by $\mathcal{O}_{\mathsf{rka}}$ as $F_{\mathbf{k}'}(\mathbf{x})$ where
     ▷ $\mathbf{k}' = \mathbf{a} \circ \Delta + \mathbf{b} \in \mathbb{K}^n = \phi(\Delta)$, for $\phi \in \Phi_{\mathsf{aff}}$.

3. **Challenge:** For the single challenge query $\mathbf{x}^*$, subject to $\langle \mathbf{z}, \mathbf{x}^* \rangle \neq 0$ and $\mathbf{x}^* \notin Q$, the challenger does the following. Sample $b \xleftarrow{R} \{0, 1\}$, then:

   - If $b = 0$, then
     - Compute $a_i := \mathsf{map}(\langle \mathbf{z}, \mathbf{x} \rangle)$ and $b_i := \mathsf{map}(\mathbf{k_0}_i + \langle \mathbf{z_1}_i, \mathbf{x}^* \rangle)$, for all $i \in [n]$.
     - Set $\phi^*\colon \mathbf{u} \mapsto \mathbf{a} \circ \mathbf{u} + \mathbf{b}$ where $\mathbf{a} := (a_1, \ldots, a_n)$ and $\mathbf{b} := (b_1, \ldots, b_n)$, where $\circ$ denotes the component-wise product.
     - Query $\mathcal{O}_{\mathsf{rka}}$ on input $(\phi^*, \mathbf{x}^*)$, and forward the response y^* to $\mathcal{A}$.
   - Else if $b = 1$, then
     - Sample $y^* \xleftarrow{R} \mathcal{Y}$ and send y^* to $\mathcal{A}$.

4. **Post-challenge queries:** Answered identically to pre-challenge queries.

Claim. *$\mathcal{A}$'s advantage in $\mathcal{H}_2$ is identical to $\mathcal{A}$'s advantage in $\mathcal{H}_1$.*

Proof. The difference between $\mathcal{H}_2$ and $\mathcal{H}_1$ is again purely syntactic since each output is computed identically in both games, with the only difference being that the challenger now only has access to Δ via the oracle $\mathcal{O}_{\mathsf{rka}}$. □

Claim. *If $\mathcal{F}$ is an RKA-secure PRF for affine related key derivation functions Φ_{aff}, then there does not exist an efficient $\mathcal{A}$ that wins the game defined in $\mathcal{H}_2$ with better than negligible advantage.*

Proof. Suppose, towards contradiction, that there exists an efficient adversary $\mathcal{A}$ for $\mathcal{H}_2$ that wins with non-negligible advantage. Construct an efficient Φ_{aff}-restricted adversary $\mathcal{B}$ that wins the RKA security game for the PRF F_k with the same advantage. $\mathcal{B}$ simply plays the role of the challenger in $\mathcal{H}_2$, forwarding all queries to its own challenger. Note that this makes $\mathcal{B}$'s queries Φ_{aff}-restricted. Therefore, on the one hand, when $\mathcal{B}$ is given access to a truly random function at the challenge phase, its answers are distributed identically to $\mathcal{H}_2$ when the challenger samples $b = 1$. On the other hand, when $\mathcal{B}$ is given access to an RKA-PRF oracle, $\mathcal{B}$'s answers are distributed identically to $\mathcal{H}_2$ when the challenger samples $b = 0$ and queries $\mathcal{O}_{\mathsf{rka}}$. As such, $\mathcal{B}$ has the same advantage as $\mathcal{A}$, which contradicts the RKA-security of $\mathcal{F}$. □

This concludes the proof of (1-key, selective) security.

Constraint Privacy. For constraint privacy, we must show that if $\mathcal{F}$ is an RKA-secure PRF family, then all evaluation and challenge queries remain pseudorandom, regardless of whether constraint $\mathbf{z}$ or $\mathbf{z}'$ is used by the challenger.[3]

Again, note that $\mathbf{z_0}_i + \Delta_i \mathbf{z}$ is distributed identically to $\mathbf{z_0}_i + \Delta_i \mathbf{z}'$, thereby making the constraint key, absent the evaluation queries, efficiently simulatable regardless of the constraint chosen by the challenger. Now, we must show that this remains the case even when the adversary is given access to the evaluation oracles. We prove this via the following lemma. Roughly speaking, the lemma states that if the underlying PRF is RKA-secure, then distinguishing between evaluations under two different related-key derivation functions of the PRF key contradicts the RKA security of the PRF.

Lemma 2.5.1. *Let λ be a security parameter and $\mathcal{F} = \{F_k \colon \mathcal{X} \to \mathcal{Y}\}_{k \in \mathcal{K}}$ be an RKA-secure PRF. Then, for all efficient Φ-restricted adversaries $\mathcal{A}$, the advantage in the following game is negligible in λ.*

1. **Setup:** On input 1^λ, the challenger samples $k \xleftarrow{R} \mathcal{K}$, samples a random bit $b \in \{0,1\}$, initializes the set $Q := \emptyset$, and runs $\mathcal{A}(1^\lambda)$.
2. **Pre-challenge queries:** For each query (ϕ, x), the challenger computes $y := F_{\phi(k)}(x)$, sends y to $\mathcal{A}$, and proceeds to update $Q := Q \cup \{(\phi, x)\}$.
3. **Challenge query:** $\mathcal{A}$ sends challenge query $(\phi_0^*, \phi_1^*, x^*)$, *subject to the restriction that* $(\phi_c^*, x^*) \notin Q,\ \forall c \in \{0,1\}$. The challenger computes $y^* := F_{\phi_b^*(k)}(x^*)$ and sends y^* to $\mathcal{A}$.
4. **Post-challenge queries:** For each query (ϕ, x) *subject to the restriction that* $(\phi, x) \neq (\phi_c^*, x^*), \forall c \in \{0,1\}$, the challenger computes $y := F_{\phi(k)}(x)$, and sends y to $\mathcal{A}$.
5. **Guess:** $\mathcal{A}$ outputs its guess b'.

$\mathcal{A}$ wins if $b' = b$ and its advantage is defined as $|\Pr[\mathcal{A}\ wins] - \frac{1}{2}|$, where the probability is over the internal coins of $\mathcal{A}$ and choice of k.

The lemma follows immediately from a standard hybrid argument. By RKA-security of the PRF F we have that $F_{\phi_0(k)}(x) \approx_c R(x) \approx_c F_{\phi_1(k)}(x)$, where R is a random function. Therefore, a distinguisher would directly contradict the security of the RKA-PRF.

■

2.5.2 DDH-based construction

In this section, we describe the DDH-based RKA-secure PRF construction of Bellare and Cash [BC10] (later extended by Abdalla et al. [ABPP14]) and describe how it fits into the framework described in Figure 2.2 to realize a DDH-based CPRF for inner-product predicates.

RKA-secure PRF from DDH. The multiplicative variant [BC10, ABPP14] of the Naor–Reingold PRF [NR97] is parameterized by an integer $n \geq 1$ and a multiplicative group $\mathbb{G}$ of prime order p with generator g. The PRF key $\mathbf{k} = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n$ consists of n random

[3] Recall that the adversary provides two constraints $\mathbf{z}$ and $\mathbf{z}'$.

elements in $\mathbb{Z}_p^n$ and the input $x \in \{0,1\}^n \setminus \{0^n\}$ is chosen from the set of all non-zero n-bit strings. The PRF NR^* is then defined as:

$$\mathsf{NR}^*((a_1, \ldots, a_n), x) = g^{\prod_{i=1}^{n} a_i^{x_i}}. \tag{2.1}$$

The RKA-secure version of the multiplicative Naor–Reingold PRF is parameterized by a collision-resistant hash function $h\colon \{0,1\}^n \times \mathbb{G}^n \to \{0,1\}^{n-2}$ and is defined as:[4]

$$\mathsf{NR}^*((a_1, \ldots, a_n), 11 \| h(x, g^{a_1}, \ldots, g^{a_n})). \tag{2.2}$$

Abdalla et al. [ABPP14, Section 4] show that Equation 2.2 is an RKA-secure PRF for Φ_{aff}-restricted adversaries. For completeness, we provide a merger of the main theorems from Abdalla et al. [ABPP14] pertaining to this construction here.

Proposition 2.5.1 (Merge of [ABPP14, Theorems 4.5, 5.1, & A1])**.** *Let $\mathbb{G}$ be a multiplicative group of prime order p and let NR^* be defined as in Equation 2.1. Let $\mathcal{H} = \{h\colon \{0,1\}^n \times \mathbb{G}^n \to \{0,1\}^{n-2}\})_{n\in\mathbb{N}}$ be a collision-resistant hash function family. Define the PRF family $\mathcal{F} = \{F_{\mathbf{k}}\colon \{0,1\}^n \to \mathbb{G}\}_{\mathbf{k}\in\mathbb{Z}_p^n}$ to be as in Equation 2.2. Then, if the DDH assumption holds in $\mathbb{G}$, $\mathcal{F}$ is RKA-secure against all efficient, Φ_{aff}-restricted adversaries $\mathcal{A}$.*

Remark 8 (RKA security under DDH)**.** *Abdalla et al. [ABPP14] prove the RKA security of their construction for Φ_{aff}-restricted adversaries under the 1-DDHI assumption (which is known to be equivalent to the Square DDH assumption [BP12]). However, they explicitly note that, by combining Theorems 4.5, 5.1, & A1 (found in the full version of their paper), they obtain the same result under the DDH assumption. This same result was also used by Attrapadung et al. [AMN+18].*

Remark 9 (Supporting vector inputs)**.** NR^* *takes as input a* binary *string $x \in \{0,1\}^n$ as opposed to a vector $\mathbf{x} \in \mathbb{F}^\ell$ as is assumed by our framework. However, we can easily map any $\mathbf{x} \in \mathbb{F}^\ell$ to a binary string of required length via any collision-resistant hash function (CRHF), which is known from the discrete logarithm assumption [Dam88] (implied by DDH, see Section 2.5.2.1), making vector inputs $\mathbf{x} \in \mathbb{F}^\ell$ syntactically cleaner and without any loss of generality. In particular, for a CRHF h, the binary string input x can be computed as $h(\mathbf{x})$. Moreover, since the RKA-secure variant of NR^* already requires hashing the input using a CRHF, this does not introduce additional computational complexity.*

Construction from DDH-based RKA-secure PRF. With the RKA-secure PRF construction of Proposition 2.5.1, we can instantiate the framework of Figure 2.2.

To satisfy the key space and related-key derivation requirements, we must instantiate our framework with the following parameters. Let p be the order of the DDH-hard group $\mathbb{G}$. We set $\mathbb{F}$ to be a field extension of $\mathbb{F}_p$, and let $n = n(\lambda) \in \mathsf{poly}(\lambda)$, following Equation 2.2. Applying Theorem 2.5.1 in conjunction with Proposition 2.5.1 yields:

Theorem 2.5.2. *Assume that the DDH assumption holds in a cyclic group $\mathbb{G}$ of order p. Then, there exists a (1-key, selectively-secure, constraint-hiding) CPRF for inner-product constraint predicates with vectors in $\mathbb{F}_p^\ell$, for any $\ell \geq 1$.*

[4] Note that the prefix "11" ensures that the input is never 0^n, and therefore always in the domain of NR^* [BC10, ABPP14].

Remark 10 (Complexity of the DDH-based construction)**.** *The Naor–Reingold PRF from Equation 2.1 can be evaluated in* NC^1*. Interestingly, the same is true of the RKA-secure variant of Equation 2.2, provided that the collision resistant hash function can be evaluated in* NC^1 *(which is the case of the discrete log based construction [Dam88]; see also Section 2.5.2.1). We will use this later in Section 2.8 when applying our construction to derive lower bounds in learning theory.*

2.5.2.1 Tool: Collision-resistant hashing from discrete logarithms

Here, we describe a construction of a collision-resistant hash function (CRHF) family from the discrete logarithm (DL) assumption that generalizes the construction of Damgård [Dam88] in the natural way. Importantly, this construction is in the complexity class NC^1, which makes the CPRF construction from the DDH assumption (when instantiated with this DL-based CRHF family) have an evaluation function that is computable in the complexity class NC^1.

Construction. Fix a prime-order group $\mathbb{G}$ in which the discrete logarithm problem is hard and let $\mathsf{extract}\colon \mathbb{G} \to \{0,1\}^k$ be a randomness extractor with $\lambda \leq k < \log_2(|\mathbb{G}|)$ and public parameters PP_e. Let $p > 2^\lambda$ be the order of $\mathbb{G}$ and define the CRHF family $\mathcal{H} = \left\{h_{\mathbf{g}}\colon \mathbb{Z}_p^n \to \{0,1\}^k\right\}_{\mathbf{g}\in\mathbb{G}^n}$, parameterized by n random generators $\mathbf{g} = (g_1, \ldots, g_n)$ and public parameters PP consisting of the group description and PP_e, where the function $h_{\mathbf{g}}\colon \mathbb{Z}_p^n \to \{0,1\}^k$ is defined as

$$h_{\mathbf{g}}(\mathbf{x}) = \mathsf{extract}\Big(\prod_{i=1}^{n} g_i^{\mathbf{x}_i}\Big).$$

Claim. *The function family* $\mathcal{H} := \left\{h_{\mathbf{g}}\colon \mathbb{Z}_p^n \to \{0,1\}^k\right\}_{\mathbf{g}\in\mathbb{G}^n}$ *is a CRHF family.*

Proof. Consider the simpler hash function $\hat{h}_{\mathbf{g}}(\mathbf{x}) = \prod_{i=1}^{n} g_i^{\mathbf{x}_i}$ parameterized by $\mathbf{g} = (g_1, \ldots, g_n)$. Suppose, towards contradiction, that there exists an efficient $\mathcal{A}$ that finds a pair of colliding inputs to $\hat{h}_{\mathbf{g}}$ with non-negligible probability $\nu(\lambda)$. Then, on input $(1^\lambda, \mathbb{G}, \mathbf{g})$, $\mathcal{A}$ outputs $(\mathbf{x}, \mathbf{x}')$ such that $\mathbf{x} \neq \mathbf{x}'$ and $\hat{h}_{\mathbf{g}}(\mathbf{x}) = \hat{h}_{\mathbf{g}}(\mathbf{x}')$, with probability at least $\nu(\lambda)$. Therefore, when $\mathcal{A}$ succeeds, we have that $\prod_{i=1}^{n} g_i^{\mathbf{x}_i} = \prod_{i=1}^{n} g_i^{\mathbf{x}'_i}$. We can use $\mathcal{A}$ to solve the discrete logarithm problem as follows. On input a generator g for $\mathbb{G}$ and an element $y \in \mathbb{G}$,

1: Sample $i \xleftarrow{\text{R}} [n]$.
2: Sample $(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \xleftarrow{\text{R}} \mathbb{Z}_p^{n-1} \setminus \{0\}$.
3: Set $\mathbf{g} = (g^{a_1}, \ldots, g^{a_{i-1}}, y, g^{a_{i+1}}, \ldots, g^{a_n})$.
4: Run $\mathcal{A}$ on input $(1^\lambda, \mathbf{g})$ and obtain as output $(\mathbf{x}, \mathbf{x}')$.
5: Compute $z := \sum_{j=1, j\neq i}^{n} a_j \mathbf{x}_j$ and $z' := \sum_{j=1, j\neq i}^{n} a_j \mathbf{x}'_j$.
6: Output $a_i := (z' - z)/(\mathbf{x}_i - \mathbf{x}'_i)$.

We now analyze the reduction. The probability that $\mathbf{x}_i \neq \mathbf{x}'_i$ is at least $\frac{1}{n}$ because i is chosen uniformly from the set $\{1, \ldots, n\}$. Second, observe that

$$\sum_{j=1}^{n} a_j \mathbf{x}_j - \sum_{j=1}^{n} a_j \mathbf{x}'_j = z - z' + a_i(\mathbf{x}_i - \mathbf{x}'_i) = 0,$$

which implies that $(z' - z)/(\mathbf{x}_i - \mathbf{x}'_i) = a_i$. As such, the reduction succeeds with probability $\frac{1}{n}\nu(\lambda)$, which is non-negligible, contradicting the discrete logarithm assumption in $\mathbb{G}$.

Finally, it follows that $h_{\mathbf{g}}$ is a CRHF if $\hat{h}_{\mathbf{g}}$ is a CRHF because $\mathsf{extract}$ is a randomness extractor and $k > \lambda$, making the advantage of $\mathcal{A}$ in the case where it is given outputs of the randomness extractor equivalent to the case where it is given the explicit description of group elements. Specifically, this follows from a random element of $\mathbb{G}$ having at least λ bits of min entropy.

■

2.5.3 VDLPN-based construction

In this section, we show that we can instantiate the framework from Figure 2.2 using any RKA-secure PRF supporting only additive key derivation functions $\Phi_+ \subset \Phi_{\mathsf{aff}}$ over the field $\mathbb{F}_2$. In particular, this allows us to instantiate our framework using the weak PRF candidate of Boyle et al. [BCG+20a] based on the variable density learning parity with noise (VDLPN) assumption. This yields the first construction of a (weak) CPRF for inner-product predicates under a code-based assumption.

RKA-secure weak PRF candidate from VDLPN. For a security parameter λ, the VDLPN-based weak PRF candidate of Boyle et al. [BCG+20a] is parameterized by integers $D = D(\lambda)$, $w = w(\lambda)$, input space $\{0,1\}^n$ and key space $\{0,1\}^n$, where $n := w \cdot D(D-1)/2$. The PRF F_K is defined as:

$$F_K(x) = \bigoplus_{i=1}^{D} \bigoplus_{j=1}^{w} \bigwedge_{k=1}^{i} (K_{i,j,k} \oplus x_{i,j,k}). \tag{2.3}$$

Theorem 2.5.3 (Informal; Adapted from [BCG+20a, Theorem 6.9])**.** *Let λ be a security parameter and suppose that the VDLPN assumption holds with parameters $w(\lambda)$ and $D(\lambda)$. Then, the PRF in Equation 2.3 is an RKA-secure weak PRF with respect to additive key derivation functions Φ_+.*

We will use the following lemma which proves that for the case of $\mathbb{F}_2$, additive and affine RKA security are in fact equivalent in our context:

Lemma 2.5.2. *Let F be a PRF with key space $\mathbb{F}_2^n$ that is secure against Φ_+-restricted adversaries. Then, the framework from Figure 2.2 instantiated with F is a secure CPRF.*

Proof. Consider the proof of Theorem 2.5.1. We look at the queries issued by the CPRF challenger to the RKA oracle $\mathcal{O}_{\mathsf{rka}}$ in Hybrid $\mathcal{H}_2$ of the proof. For each query $\mathbf{x}$ issued by the adversary to the CPRF challenger, the induced affine function $\phi \in \Phi_{\mathsf{aff}}$ is parameterized by vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}_2^n$. Note that $\mathbf{a} = (a_1, \ldots, a_n)$, where $a_i := \langle \mathbf{z}, \mathbf{x} \rangle$. Moreover, $a_i \neq 0$ for all queries that do not satisfy the constraint, which implies that $a_i = 1 \in \mathbb{F}_2$. As such, each (constrained) query issued to the RKA oracle $\mathcal{O}_{\mathsf{rka}}$ by the challenger is an affine function $\phi \in \Phi_{\mathsf{aff}}$ parameterized by $(\mathbf{1}, \mathbf{b})$ and the oracle $\mathcal{O}_{\mathsf{rka}}$ responds with the PRF evaluated using key $\mathbf{k} := \mathbf{1} \circ \Delta + \mathbf{b}$. This is equivalent to an *additive* function $\phi' \in \Phi_+$ simply parameterized by $\mathbf{b}$. The reduction in Theorem 2.5.1 therefore goes through as before, concluding the lemma. ■

Construction from VDLPN-based RKA-secure weak PRF. With the RKA-secure weak PRF construction of Equation 2.3, we can instantiate the framework from Figure 2.2. To satisfy the key space and related-key derivation requirements, we must instantiate our framework with the following parameters. We set $\mathbb{F}$ to be a field extension of $\mathbb{F}_{2^n}$, $n = n(\lambda) \in \mathsf{poly}(\lambda)$, map maps from $\mathbb{F}$ to $\mathbb{F}_2^n$, and $\ell \geq n$ (inputs of length ℓ can be truncated to n before being fed into the PRF, without loss of generality). Applying Theorem 2.5.1 in conjunction with Theorem 2.5.3 and Lemma 2.5.2 yields:

Theorem 2.5.4. *Assume that the VDLPN assumption holds. Then, there exists a (1-key, selectively-secure, constraint-hiding) weak CPRF for inner-product constraint predicates computed over vectors in $\mathbb{F}_2^\ell$, where $\ell \geq n$.*

2.6 CPRFs for Inner-Product Predicates from OWFs

In this section, we instantiate our framework from Section 2.5.1 under the minimal assumption that one-way functions exist. Unlike our constructions in Section 2.5.1, here we will require that the set of possible related keys computed for evaluation queries is bounded in size by a fixed polynomial $t = t(\lambda)$, which forces us to restrict the input domain of the CPRF. Specifically, we show that we can satisfy this requirement *without* placing any restrictions on the CPRF adversary if the CPRF inputs are vectors in $[0, B)^\ell$ with $B \in O(1)$ and $\ell = \ell(\lambda) \in O(\log \lambda)$. These restrictions limit the L_∞-norm of each input vector and make the input domain of the CPRF polynomial in the security parameter. We note that this is the same class of inner-product constraints considered by Davidson et al. [DKN+20] (inner products over $\mathbb{Z}$) from the LWE assumption, albeit here we only obtain a polynomially-sized input domain.

Our construction builds off of a result by Applebaum and Widder [AW14], which constructs a restricted class of RKA-secure PRFs from any PRF and an m-wise independent hash function. Their construction is secure against *additive* relations over a group, provided that the RKA adversary uses at most $t = t(\lambda)$ different related-key derivation functions $\phi_1 \ldots, \phi_t \in \Phi_+$, where $t \ll m$. (We stress, however, that the adversary can query the PRF on an unbounded number of inputs using each of the t different RKA functions.) Because m-wise independent hash functions can be constructed unconditionally [WC81], the resulting RKA-secure PRF can be realized from any PRF, thus relying only on the assumption that one-way functions exist [GGM86, AW14]. More formally, they prove:

Theorem 2.6.1 (Adapted from [AW14])**.** *Let $\mathcal{K} = \{\mathbb{G}_\lambda\}_{\lambda \in \mathbb{N}}$ be a sequence of efficiently computable additive groups, and $t = t(\lambda)$ be an arbitrary fixed polynomial. Then, assuming the existence of a PRF $\mathcal{F} = \{F_k \colon \mathcal{X}_\lambda \to \mathcal{Y}\}_{k \in \mathbb{G}_\lambda}$, there exists an RKA-secure PRF with respect to addition over $\mathcal{K}$ provided that the total number of unique related-key derivation functions queried by the adversary is bounded by t. (The adversary is allowed to query each function on any number of inputs.)*

Unfortunately, we require the PRF to be RKA-secure with respect to *affine* relations Φ_{aff} and therefore cannot apply Theorem 2.6.1 directly. More concretely, the issue with affine (as opposed to additive) relations is that they are not "claw-free," meaning that there exist

pairs of different functions $\phi_1, \phi_2 \in \Phi_{\mathsf{aff}}$ such that for a key $k \in \mathcal{K}$, $\phi_1(k) = \phi_2(k)$. The lack of claw-freeness poses problems in security proofs because, if an adversary is able to find two different $\phi_1, \phi_2 \in \Phi_{\mathsf{aff}}$ such that $\phi_1(k) = \phi_2(k)$, the adversary learns information about k and can then break the RKA-security of the PRF [ABPP14]. To address this, we strengthen Theorem 2.6.1 for the case of Φ_{aff}-restricted adversaries by showing that the number of collisions is bounded by a negligible factor in the security parameter, proving a stronger theorem via their approach. We describe this next.

2.6.1 Affine RKA-secure PRFs from OWFs

In this section, we show how to construct RKA-secure PRFs for affine related-key derviation functions from one-way functions. The framework and proof closely follows that of Applebaum and Widder [AW14] for constructing RKA-secure PRFs from m-wise independent hash functions.

Immunizing PRFs against RKA. The idea of Applebaum and Widder [AW14] is to immunize any regular PRF family $\mathcal{F}$ with key space $\mathcal{K} = \mathcal{K}_\lambda$ against a *bounded* related-key attack, where the adversary makes at most t related key queries (but can make an unbounded number of PRF queries under each related key) for some apriori fixed $t = t(\lambda) \in \mathsf{poly}(\lambda)$. The high-level idea is to use a long key s from a large key space $\mathcal{S}$ (larger than $\mathcal{K}^t$) and use a public hash function h to derive a shorter key $h(s) \in \mathcal{K}$ for $\mathcal{F}$. Here, we generalize their approach to the case of affine functions.

Definition 2.6.1 (t-good Hash Function)**.** *Let λ be a security parameter, $\mathbb{F}$ be finite field of order at least 2^λ, and $\mathcal{K} \subseteq \{0,1\}^\lambda$ be a set of strings. A hash function $h\colon \mathbb{F} \to \mathcal{K}$ is said to be t-good if for any t-tuple of distinct affine function $(\phi_1, \ldots, \phi_t) \in \Phi_{\mathsf{aff}}^t$, the joint distribution of $(h(s), h(\phi_1(s)), \ldots, h(\phi_t(s))$ induced by a random choice of $s \xleftarrow{R} \mathbb{F}$, is ε-close in statistical distance to the uniform distribution over $\mathcal{K}^{t+1}$, for some negligible $\varepsilon = \varepsilon(\lambda)$.*

Definition 2.6.2 (t-good Hash Family)**.** *Let λ be a security parameter, $\mathbb{F}$ be a finite field of order at least 2^λ, and $\mathcal{Z}, \mathcal{K} \subseteq \{0,1\}^\lambda$. A family of hash functions $H = \{h_z\colon \mathbb{F} \to \mathcal{K}\}_{z \in \mathcal{Z}}$ is said to be t-good if with all-but-negligible probability, for a randomly selected $z \xleftarrow{R} \mathcal{Z}$, the hash function h_z is t-good.*

Remark 11 (Relation to correlation-robustness)**.** *We note that a t-good hash function can instantiated via a suitable correlation-robust hash function (and, in particular, a random oracle), which provides an alternative strategy to constructing CPRFs (in the selective security regime) from a random oracle.*

We now prove that if we have a t-good hash family, we can "immunize" any PRF against affine related key attacks. Later, in Lemma 2.6.1, we show how to construct a t-good hash family from m-wise independent hash functions.

Theorem 2.6.2 (Extended from [AW14, Lemma 7.1])**.** *Let λ be a security parameter, $t = t(\lambda) \in \mathsf{poly}(\lambda)$, $\mathbb{F}$ be a finite field of order at least 2^λ, and $\mathcal{Z}, \mathcal{K} \subseteq \{0,1\}^\lambda$. Let $\mathcal{F} = \{F_k\colon \mathcal{X} \to \mathcal{Y}\}_{k \in \mathcal{K}}$ be a PRF family and $H = \{h_z\colon \mathbb{F} \to \mathcal{K}\}_{z \in \mathcal{Z}}$ be a t-good hash family. The PRF family $\mathcal{G} = \{G_{s,z}\colon \mathcal{X} \to \mathcal{Y}\}_{s \in \mathbb{F}, z \in \mathcal{Z}}$, parameterized by a secret $s \xleftarrow{R} \mathbb{F}$ and public*

$z \xleftarrow{R} \mathcal{Z}$, and defined by the mapping $G_{s,z}(x) \mapsto F_k(x)$, where $k := h_z(s)$, is an RKA-secure PRF family against t-bounded Φ_{aff}-restricted adversaries.

Proof. Suppose, towards contradiction, there exists an efficient Φ_{aff}-restricted $\mathcal{A}$ that has non-negligible advantage in the RKA-security game for G. Then, there exists a non-negligible function ν such that,

$$\left| \Pr_{s \xleftarrow{R} \mathbb{F}, z \xleftarrow{R} \mathcal{Z}} [\mathcal{A}^{G_{s,z}}(1^\lambda, z)] - \Pr_{z \xleftarrow{R} \mathcal{Z}} [\mathcal{A}^{R}(1^\lambda, z)] \right| \geq \nu(\lambda),$$

where R is a truly random function.

Then, consider a vector of $t+1$ keys $\mathbf{k} := (k_0, k_1, \ldots, k_t) \in \mathcal{K}^{t+1}$, and define a stateful oracle $\mathcal{O}_{\mathbf{k}}$ as follows.

Oracle $\mathcal{O}_{\mathbf{k}}$

Initialize. Set $Q_\phi := \{\}$, define a dictionary $T := [\,]$, and counter $j := 1$.

Evaluation.

- For each non-RKA query x, output $F_{k_0}(x)$.
- For each RKA query (ϕ, x):
  - If $\phi \in Q_\phi$, retrieve $k_i := T[\phi]$ and output $F_{k_i}(x)$.
  - If $\phi \notin Q_\phi$, set $T[\phi] := k_j$, set $j := j + 1$, and output $F_{k_j}(x)$.

In words, $\mathcal{O}_{\mathbf{k}}$ outputs $F_{k_i}(x)$, and stores the association between ϕ and k_i to answer all future queries involving ϕ using PRF key k_i.

Now, because h_z is t-good, for a random vector $\mathbf{k}$ of $t+1$ keys, we have that

$$\left| \Pr_{\mathbf{k} \xleftarrow{R} \mathcal{K}^{t+1}, z \xleftarrow{R} \mathcal{Z}} [\mathcal{A}^{\mathcal{O}_{\mathbf{k}}}(1^\lambda, z)] - \Pr_{z \xleftarrow{R} \mathcal{Z}} [\mathcal{A}^{G_{s,z}}(1^\lambda, z)] \right| \geq \nu(\lambda) - \mathsf{negl}(\lambda).$$

By a straightforward hybrid argument, it follows that $\mathcal{A}$ has non-negligible advantage in winning the (standard) PRF game by distinguishing between $\mathcal{O}_{\mathbf{k}}$ and the truly random function R, contradicting that $\mathcal{F}$ is a PRF. This proves security against Φ_{aff}-restricted adversaries.

■

The following lemma shows that any $\Omega(\lambda \cdot t^2)$-wise independent hash function with a sufficiently large domain is t-good in the sense of Definition 2.6.1. Moreover, an m-wise independent hash function can be constructed unconditionally for any m (e.g., using a universal hash based on random polynomials [WC81]).

Lemma 2.6.1. *Let λ be a security parameter, $t = t(\lambda) \in \mathsf{poly}(\lambda)$, and H be a family of m-wise independent hash functions with domain $S = \{S_\lambda\}$ and range $K = \{K_\lambda\}$ where*

$m \geq \lambda(3t+5)(t+1)$, $|K_\lambda| = 2^\lambda$, and $|S_\lambda| = 2^{\lambda(2t+6)}$. Then, H is a t-good family of hash function. In particular, for all but a $2^{-\lambda}$ fraction of the functions in H, the distribution of $h_z \xleftarrow{R} H$ is $2^{-0.99\lambda}$-close to uniform.

Proof. The proof is almost identical (occasionally taken verbatim) to the related proof of Applebaum and Widder [AW14, Lemma 7.2] for the case of additive functions. However, it differs in several key places where we must consider *affine* functions and their impact on the corresponding distributions, which has sufficient repercussions to necessitate rewriting the proof in full.

Fix a sequence of t distinct affine functions $\boldsymbol{\phi} := (\phi_0, \phi_1, \ldots, \phi_t)$ where we define ϕ_0 to be the identity function for notational convenience. We say that $h_z \in H$ is ε-good for $\boldsymbol{\phi}$ if for a random s, the distribution $\{h_z(\phi_i(s)\}_{0 \leq i \leq t}$ is ε-close to the uniform distribution over $\mathcal{K}^{t+1}$. In order to bound the statistical distance, we must prove the following claim.

Claim. *For all but a $2^{-2\lambda(t+1)-\lambda}$-fraction of the $h \in H$ the following holds. For every vector of (not necessarily distinct) keys $\mathbf{k} := (k_0, \ldots, k_t) \in \mathcal{K}^{t+1}$,*

$$\Pr_{s \xleftarrow{R} S}\left[\bigwedge_{i=0}^{t} h(\phi_i(s)) = k_i\right] \in \left(\frac{1}{|\mathcal{K}|^{t+1}} \cdot (1 \pm 2^{-0.99\lambda})\right).$$

Proof. Fix a vector of keys $\mathbf{k} \in \mathcal{K}^{t+1}$. For every $s \in S$, define the indicator random variable χ_s which takes on the value 1 if $h(\phi_i(s)) = k_i$ for all $i \in \{0, 1, \ldots, t\}$ and a random choice of $h \in H$. Observe that the random variable $\bar{\chi}$ taking the value of $\Pr_s[\bigwedge_{i=0}^{t} h(\phi_i(s)) = k_i]$, and induced by a choice of h, can be written as $\bar{\chi} = \sum_{s \in S} \frac{\chi_s}{|S|}$. Next, we must prove the following bound:

$$\Pr_{h \xleftarrow{R} H}\left[\bar{\chi} \notin \left(\frac{1}{|\mathcal{K}|^{t+1}} \cdot (1 \pm 2^{-0.99\lambda})\right)\right] \leq 2^{-3\lambda(t+1)-\lambda}, \tag{2.4}$$

which we will later use to prove the claim via a simple union bound. To prove Equation 2.4, observe that since H is an m-wise independent hash family and $m > t+1$, we have that $\mathbb{E}[\chi_s] = 1/|\mathcal{K}|^{t+1}$, for every s. Then, by linearity of expectation, it is easy to see that $\mathbb{E}(\bar{\chi}) = 1/|\mathcal{K}|^{t+1}$. Next, we show that the average of χ_s is concentrated around its expectation. Following the proof of Applebaum and Widder [AW14, Claim 7.3], we can show that the χ_s's are r-wise independent, for $r \geq 3t+5$, which yields a strong concentration bound despite local dependencies in the χ_s's (see the result of Gradwohl and Yehudayoff [GY08] for an overview of the deployed proof strategy).

To formally prove the bound, define a graph G over any pair of $s, s' \in S$ by placing an edge between s and s' if $\phi_i(s) = \phi_j(s')$ for some $i \neq j$. It then follows that the degree of each node in G is at most $d = (t+1)^2$. We claim that for every independent set I in the graph, the random variables $\{\chi_s : s \in I\}$ are r-wise independent (or using the terminology of Gradwohl and Yehudayoff [GY08], the random variables r-agree with G). To show this, consider any independent set $I \subseteq S$. For any r-sized subset $(s_1, \ldots, s_r) \subseteq I$, the value of each random variable χ_{s_j} for $s_j \in I$, solely depends on the value of h evaluated on the set of $t+1$ points $(\phi_0(s_j), \phi_1(s_j), \ldots, \phi_{t+1}(s_j))$. Moreover, observe that for all choices of $t+1$ distinct affine functions $(\phi_1, \ldots, \phi_{t+1})$, all elements of the set $\{\phi_0(s_j), \phi_1(s_j), \ldots, \phi_{t+1}(s_j)\}$ are also distinct with probability at least $1 - \frac{t+1}{2^\lambda}$, since the probability of a collision between

any distinct ϕ_u and ϕ_v is exactly $1/|S| < 2^{-\lambda} \leq 1/|\mathcal{K}|$. It then follows (via a union bound and using the fact that I is an independent set) that the sets $\{\phi_0(s_j), \phi_1(s_j), \ldots, \phi_{t+1}(s_j)\}$ for all $j \in [r]$ are distinct with probability at least $1 - \frac{r(t+1)}{2^\lambda}$.

From the above, we conclude that with all but negligible probability in λ, the image of these sets under a randomly chosen h are statistically independent, since h is m-wise independent for $m \geq r(t+1)$. It then follows that $\chi_{s_1}, \ldots, \chi_{s_r}$ are statistically independent, or in other words, agree with G [GY08]. Applying the bound of [GY08, Corollary 3.2] and taking into account the negligible collision probability computed above, we get that:

$$\Pr_{h \xleftarrow{R} H}\left[\bar{\chi} \notin \left(\frac{1}{|\mathcal{K}|^{t+1}} \cdot (1 \pm \delta)\right)\right] < 4\sqrt{\pi r}\left(\frac{|\mathcal{K}|^{t+1}\sqrt{(d+1)r}}{\delta\sqrt{|S|}}\right)^r + \frac{r(t+1)}{2^\lambda}. \tag{2.5}$$

Then, setting $\delta = 2^{-0.99\lambda}$, $|\mathcal{K}| = 2^\lambda$, $|S| = 2^{(2t+6)\lambda}$, and $r, t \in \mathsf{poly}(\lambda)$, Equation 2.5 is upper-bounded by $2^{-\lambda r} \leq 2^{-3\lambda(t+1)-\lambda}$ for all sufficiently large λ (recall that $d = (t+1)^2$ and $r \geq 3t+5$), and so Equation 2.4 follows. The claim then follows by applying a union bound over all $2^{\lambda(t+1)}$ possible $\mathbf{k} \in \mathcal{K}^{t+1}$, since $\lambda(t+1) - 3\lambda(t+1) - \lambda = -2\lambda(t+1) - \lambda$. ■

To complete the proof of the lemma, note that any h that satisfies the lemma is $2^{-0.99\lambda}$-good (as defined in the beginning of the proof) for the fixed sequence of affine functions $\boldsymbol{\phi}$. Specifically, $(h(\phi_0(s)), \ldots, h_t(\phi_{t+1}(s)))$ has a statistical distance of at most $2^{-0.99\lambda}$ from the uniform distribution. Moreover, as shown above, all but a $2^{-2\lambda(t+1)-\lambda}$-fraction of the $h \in H$ are t-good for the fixed vector $\boldsymbol{\phi}$. By applying a union bound over all possible $2^{2\lambda(t+1)}$ affine functions, we conclude that all but a $2^{-\lambda}$-fraction of the $h \in H$ are t-good, in the sense of Definition 2.6.1, and the lemma follows. ■

2.6.2 CPRF construction from OWFs

Using the RKA-secure PRF construction from Theorem 2.6.2, we can instantiate the framework from Figure 2.2 with $\mathbb{F} = \mathbb{F}_p$, for sufficiently large $p \geq 2^{\lambda(2t+6)}$ as required by Lemma 2.6.1, and $n \geq 1$. However, we must set the input vector domain to $[0, B)^\ell \subset \mathbb{Z}^\ell$ with the vector length ℓ such that $B^\ell \leq t$. Specifically, this ensures that the total number of unique inputs to the t-good hash when deriving affine keys is bounded by $t = t(\lambda) \in \mathsf{poly}(\lambda)$. To see this, note that there are B^ℓ possible values for the inner product $\langle \mathbf{z_0}, \mathbf{x} \rangle + \Delta \langle \mathbf{z}, \mathbf{x} \rangle$ given that $\mathbf{z}$ and $\mathbf{z_0}$ are fixed while $\mathbf{x} \in [0, B)^\ell$ is chosen by the adversary. Hence, we can simply let map be defined by applying n different t-good hash functions component-wise to derive the PRF key in K^n. Then, applying Theorem 2.5.1 in conjunction with Theorem 2.6.2 yields:

Theorem 2.6.3. *Let λ be a security parameter and fix a polynomial $t = t(\lambda) \in \mathsf{poly}(\lambda)$. Assume that one-way functions exist. Then, there exists a (1-key, selectively-secure, constraint-hiding) CPRF for inner-product constraint predicates with $\ell = \ell(\lambda) \in O(\log \lambda)$ and input vectors in the range $[0, B)$ for any constant B such that $B^\ell \leq t$.*

Proof. We recall the proof of Theorem 2.5.1, and in particular Hybrid $\mathcal{H}_2$. In the game defined by $\mathcal{H}_2$, for each query $\mathbf{x}$ issued by the CPRF adversary, the challenger derives the affine function ϕ parameterized by vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n$ where:

- $\mathbf{a} := (a_1, \ldots, a_n)$ with $a_i = \langle \mathbf{z}, \mathbf{x} \rangle$ for all $i \in [n]$.
- $\mathbf{b} := (b_1, \ldots, b_n)$ with $b_i = \langle \mathbf{z}_{\mathbf{0}i}, \mathbf{x} \rangle$ for all $i \in [n]$.

Note that $\mathbf{z}$ and $\mathbf{z}_{\mathbf{0}i}$, for all $i \in [n]$ are fixed at the start of the CPRF game. Therefore, $\mathbf{a}, \mathbf{b}$ are both entirely determined by the query vector $\mathbf{x}$. The RKA oracle $\mathcal{O}_{\mathsf{rka}}$ in $\mathcal{H}_2$ (when instantiated with the immunized RKA-PRF construction of Theorem 2.6.2) computes the RKA key as $h_i(a_i\Delta_i + b_i)$ for all $i \in [n]$, where h_i is an independent t-good hash function and Δ_i is an independent PRF key. We must show that, for all possible sets of queries $Q := \{\mathbf{x}_j \mid 1 \leq j \leq q_E\}$ issued by $\mathcal{A}$ (here q_E is an arbitrary upper bound on the total number of evaluation queries), the number of unique inputs to h_i never exceeds t. This follows from the fact that the number of possible values that $k_i := a_i\Delta_i + b_i$ can take on is bounded by the number of unique values of $\mathbf{x}$, which in turn is bounded by $B^\ell \leq t$, by construction. We stress that there are no restrictions placed on the adversary's queries—the adversary can adaptively query the CPRF challenger and issue any polynomial number of evaluation queries (independently of t). ■

As a corollary, we obtain an analogous result to Theorem 2.6.3 but with an exponential input domain provided that the CPRF adversary makes at most t unique evaluation queries on constrained inputs.

Corollary 2.6.1. *Let λ be a security parameter and fix a polynomial $t = t(\lambda) \in \mathsf{poly}(\lambda)$. Assume that one-way functions exist. Then, there exists a (1-key, selectively-secure, constraint-hiding) CPRF for inner-product constraint predicates for any $\ell \geq 1$ provided that the adversary makes at most t constrained evaluation queries.*

2.7 Extensions

In this section, we describe extensions to CPRFs with inner-product constraints.

2.7.1 More general constraint predicates

It is known (in some cases folklore) that CPRFs for inner-product constraint predicates yield CPRFs with constraints described by constant-degree polynomials, t-CNF formulas (with constant t) [DKN+20], and the "AND" of an arbitrary set of constraint predicates. We explicitly describe these extensions here for completeness. We note that all the presented extensions preserve the constraint-hiding property.

CPRFs for constant-degree polynomials. A CPRF for inner-product constraint predicates can be converted to a CPRF for constraint predicates described by constant-degree polynomials P by associating each entry in the constraint vector $\mathbf{z}$ with a coefficient of P. Specifically, let $\mathbf{z} = (a_d, a_{d-1}, \ldots, a_1, a_0)$ be the coefficients describing the degree-d polynomial over $\mathbb{F}$. Then, for input vectors of the form $\mathbf{x} = (x^d, x^{d-1}, \ldots, x, 1)$, it holds that $P(x) = 0$ if and only if $\langle \mathbf{z}, \mathbf{x} \rangle = 0$.

CPRFs for t-CNF formulas. Any t-CNF formula can be defined as the AND of $d = \binom{m}{t} \cdot 2^t$ NC^0_t circuits, where NC^0_t is the class of NC^0 circuits that read at most t indices of the input

bits [DKN+20]. More formally, a t-CNF circuit $C\colon \{0,1\}^m \to \{0,1\}$ can be defined as:

$$C(x) = \bigwedge_{i=1}^{d} C_i(x) \text{ where } C_i \in \mathsf{NC}^0_t. \tag{2.6}$$

Davidson et al. [DKN+20, Appendix C] provide a simple reduction from CPRFs for inner-product predicates to CPRFs for t-CNF formulas. The high-level idea is to let $\mathbf{x} = (C_1(x), C_2(x), \ldots, C_d(x), -1)$, where the C_i's describe the t-CNF circuit C, as per Equation 2.6. The constraint vector is then defined as $\mathbf{z} = (z_1, \ldots, z_d, w)$, where $z_i = 1$ if the i-th circuit needs to be satisfied and $z_i = 0$ otherwise, and w is the hamming weight of $(z_1, \ldots, z_d)$. It then holds that $\langle \mathbf{z}, \mathbf{x} \rangle = 1$ if and only if $C(\mathbf{x}) = 0$. This reduction to t-CNF formulas implicitly uses the fact that we can describe constraints as the "AND" of many individual, simpler constraints. We describe this trick explicitly, and explain how it applied to constructing constraint predicates described by matrix-vector products.

Conjunction of constraints. Here, we show that if we have a CPRF for a constraint class $\mathcal{C}$, then we can construct a CPRF for the constraint class $\bigwedge_{i=1}^{d} C_i$ where $\forall i, C_i \in \mathcal{C}$. In a nutshell, we can define the CPRF for "AND constraints" as a vector of d CPRFs such that the output is defined to be the addition of all the individual CPRF outputs. It is not difficult to see that the sum of the d individual CPRF outputs will be consistent with the evaluation under the master secret key if and only if all the constraints are satisfied.

Let $\mathsf{CPRF} = (\mathsf{CPRF.KeyGen}, \mathsf{CPRF.Eval}, \mathsf{CPRF.Constrain}, \mathsf{CPRF.CEval})$ be a CPRF for constraints in the class $\mathcal{C}$. We construct the CPRF $\widehat{\mathsf{CPRF}}$ for the AND of d constraints in $\mathcal{C}$ as follows. Let $\oplus$ denote the group operation over the range $\mathcal{Y}$.

- $\widehat{\mathsf{CPRF}}.\mathsf{KeyGen}(1^\lambda, d)$:

  1: Compute $\mathsf{msk}_i \leftarrow \mathsf{CPRF.KeyGen}(1^\lambda)$ for all $i \in [d]$.
  2: Output $\mathsf{msk} := (\mathsf{msk}_1, \ldots, \mathsf{msk}_d)$.

- $\widehat{\mathsf{CPRF}}.\mathsf{Eval}(\mathsf{msk}, x)$:

  1: Parse $\mathsf{msk} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_d)$.
  2: Compute $y_i := \mathsf{CPRF.Eval}(\mathsf{msk}_i, x)$ for all $i \in [d]$.
  3: Output $\bigoplus_{i=1}^{d} y_i$.

- $\widehat{\mathsf{CPRF}}.\mathsf{Constrain}(\mathsf{msk}, \widehat{C})$:

  1: Parse $\mathsf{msk} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_d)$ and $\widehat{C} = (C_1, \ldots, C_d) \in \mathcal{C}^d$.
  2: Compute $\mathsf{csk}^{(i)} \leftarrow \mathsf{CPRF.Constrain}(\mathsf{msk}_i, C_i)$ for all $i \in [d]$.
  3: Output $\mathsf{csk} := (\mathsf{csk}^{(1)}, \ldots, \mathsf{csk}^{(d)})$.

- $\widehat{\mathsf{CPRF}}.\mathsf{CEval}(\mathsf{csk}, x)$:

  1: Parse $\mathsf{csk} = (\mathsf{csk}^{(1)}, \ldots, \mathsf{csk}^{(d)})$.
  2: Compute $y_i := \mathsf{CPRF.CEval}(\mathsf{csk}^{(i)}, x)$ for all $i \in [d]$.
  3: Output $\bigoplus_{i=1}^{d} y_i$.

We prove the following proposition with regards to the above construction.

Proposition 2.7.1. *Let* $\mathsf{CPRF} = (\mathsf{CPRF.KeyGen}, \mathsf{CPRF.Eval}, \mathsf{CPRF.Constrain}, \mathsf{CPRF.CEval})$ *be a CPRF for constraints in the class* $\mathcal{C}$*. Then* $\widehat{\mathsf{CPRF}}$ *is a CPRF for constraint predicates described as* $\bigwedge_{i=1}^{d} C_i$*, where* $C_i \in \mathcal{C}$*. Moreover, if* CPRF *is constraint-hiding, then so is* $\widehat{\mathsf{CPRF}}$*.*

Proof sketch. We briefly sketch the proofs of correctness and security.

Correctness. Correctness holds because if all d constraints $C_1, \ldots, C_d$ are satisfied, then $\widehat{\mathsf{Eval}}$ and $\widehat{\mathsf{CEval}}$ agree on all y_i computed as $\mathsf{CPRF.Eval}(\mathsf{msk}_i, x)$ and $\mathsf{CPRF.CEval}(\mathsf{csk}^{(i)}, x)$, respectively. It then follows that the sum of the outputs is identical under both the master secret key and constrained key.

Security. If at least one $C_1, \ldots, C_d$ is *not* satisfied, then $\mathsf{CPRF.CEval}(\mathsf{csk}^{(i)}, x)$, for at least one $i \in [i]$ will output a pseudorandom value in $\mathcal{Y}$ (by the security of CPRF). By a straightforward hybrid argument, it then follows that $\widehat{\mathsf{CPRF}}.\mathsf{CEval}(\mathsf{csk}^{(i)}, x)$ outputs a pseudorandom value that is independent of the CPRF evaluation under the master key. Constraint hiding follows by a similar hybrid argument. ■

Matrix-vector product constraints. As a corollary of Proposition 2.7.1 and our constructions of CPRF for inner-product predicates, we can construct CPRFs for constraints where the constraint is satisfied if and only if $\mathbf{Ax} = \mathbf{0}$, for some constraint matrix $\mathbf{A}$. Specifically, for a matrix $\mathbf{A} \in \mathbb{F}^{d \times \ell}$ where $(\mathbf{a}_1, \ldots, \mathbf{a}_d) \in (\mathbb{F}^{\ell})^d$ is the vector of rows of $\mathbf{A}$, it holds that $\mathbf{Ax} = \mathbf{0} \iff \bigwedge_{i=1}^{d} \langle \mathbf{a}_i, \mathbf{x} \rangle = 0$.

2.8 Application to Learning Theory

In this section, we highlight known connections between learning theory and CPRFs and provide a corollary that is implied by our CPRF construction from DDH.

Membership queries with restriction access. Motivated by the goal of providing stronger lower bounds in learning theory, Cohen, Goldwasser, and Vaikuntanathan [CGV15] introduce a learning model they call MQ *with Restriction Access* ($\mathrm{MQ}_{\mathsf{RA}}$) and show that CPRFs naturally define a concept class that is not learnable, even when the learner obtains non-black-box access to the function on a restricted subset of the domain. Informally, in the basic MQ learning framework [Val84] (without restriction access), a learner gets oracle access to a function and must approximate the function after a sufficient number of queries. *Restriction access* [DRWY12] is a different model in learning theory, where the learner obtains a non-black-box implementation of the function computing a restricted set of function evaluations. Cohen et al. [CGV15] merge the two models to introduce the model of MQ with Restriction Access ($\mathrm{MQ}_{\mathsf{RA}}$), where in addition to black-box membership queries, the learner obtains non-black-box access to a restricted "simplified" version of the function. We provide the informal definition here, and point the reader to Cohen et al. [CGV15] for details and further discussion.

Definition 2.8.1 (Membership queries with restriction access ($\mathrm{MQ}_{\mathsf{RA}}$) [CGV15])**.** *Let* $\mathcal{C} \colon \mathcal{X} \to \{0,1\}$ *be a concept class, and* $\mathcal{S} = \{S \subseteq \mathcal{X}\}$ *be a collection of subsets of the domain* $\mathcal{X}$*.* $\mathcal{S}$

is the set of allowable restrictions for concepts $f \in \mathcal{C}$. Let Simp *be a "simplification rule" which, for a concept f and restriction S outputs a "simplification" of f restricted to S. An algorithm $\mathcal{A}$ is an $(\epsilon, \delta, \alpha)$-$MQ_{\mathsf{RA}}$ learning algorithm for representation class $\mathcal{C}$ with respect to a restrictions in $\mathcal{S}$ and simplification rule* Simp *if, for every $f \in \mathcal{C}$, $\Pr[\mathcal{A}^{\mathsf{Simp}(f,\cdot)} = h] \geq 1 - \delta$, where h is an ϵ-approximation to f, and furthermore, $\mathcal{A}$ only requests restrictions for an α-fraction of the whole domain $\mathcal{X}$.*

Cohen et al. [CGV15] prove the following theorem (restated here in its informal version since the formal definitions require substantial notation):

Theorem 2.8.1 (Informal)**.** *Suppose $\mathcal{F}$ is a family of constrained PRFs which can be constrained to sets in $\mathcal{S} = \{S \subseteq \mathcal{X}\}$. If $\mathcal{F}$ is computable in circuit complexity class $\mathcal{C}$, then $\mathcal{C}$ is hard to MQ_{RA}-learn with restrictions in $\mathcal{S}$.*

Let $\mathcal{IP} = \left\{\{\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{z}\} \mid \mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{z} \in \mathbb{F}^\ell; \langle \mathbf{x}_i, \mathbf{z} \rangle = 0, \forall i \in [N]\right\}_{N \in \mathbb{N}}$ be the subsets of the input domain $\mathbb{F}^\ell$ that satisfy the inner-product relation with respect to a vector $\mathbf{z}$. Using our CPRFs for inner-product predicates, we immediately obtain the following two corollaries.

Corollary 2.8.1. *Assuming the DDH assumption holds in a cyclic group $\mathbb{G}$, there is a simplification rule such that NC^1 is hard to MQ_{RA}-learn with respect to restrictions in $\mathcal{IP}$.*

In particular, Corollary 2.8.1 uses the fact that our DDH-based CPRF construction can be evaluated in NC^1 (recall Remark 10).

2.9 Evaluation

In this section, we implement and benchmark our CPRF constructions. Our implementation is open source [SS24b]. For each construction, we first analyze the complexity (in terms of multiplication, additions, and invocations of other cryptographic primitives) and then report the concrete performance of our Go (v1.20) implementation benchmarked on an Apple M1 CPU. All benchmarks are performed on a single core.

2.9.1 Complexity and benchmarks

Random oracle construction. The random oracle construction requires computing the inner product in $\mathbb{F}$ followed by a call to a random oracle. We heuristically instantiate the random oracle using the SHA256 hash function. We let the $\mathbb{F} = \mathbb{F}_p$ be a finite field where p is a 128-bit prime. The bottleneck of the construction is computing the inner product (modulo p), which requires a total of ℓ modular multiplications and additions. We report the concrete performance in Table 2.2. Overall, evaluation requires a few microseconds of computation time, ranging from 2μs for small vectors ($\ell = 10$) and 200μs for large vectors ($\ell = 1000$).

DDH-based construction. In the DDH-based construction, the bulk of the required operations are performed modulo p, where p is the order of the DDH-hard group. For a security parameter λ and $n = n(\lambda)$, the CPRF construction requires computing (1) $n\ell$

(ℓ)	10	50	100	500	1000
	2 μs	10 μs	19 μs	98 μs	200 μs

Table 2.2: Concrete evaluation time for our RO-based construction for vectors of length ℓ.

(ℓ)	10	50	100	500	1000
	8 ms	11 ms	16 ms	46 ms	85 ms

Table 2.3: Concrete evaluation time for our DDH-based construction for vectors of length ℓ.

multiplications and $n\ell$ additions (mod p) to compute the inner products between length-ℓ vectors, (2) one invocation of a collision-resistant hash function, and (3) n multiplications (mod p) and $n+1$ group operations in $\mathbb{G}$ to compute the PRF evaluation. This results in a total complexity of $n(\ell+1)$ multiplications (mod p), $n\ell$ additions, $n+1$ group operations, and one invocation of a CRHF. Using the P256 elliptic curve, letting $n = 128$, and using the discrete logarithm based CRHF construction (described in Section 2.5.2.1), each CPRF evaluation requires a few milliseconds to compute (note that in practice, the DL-based CRHF can be replaced with a fixed-key AES or SHA256 hash function for better performance). We report the concrete performance in Table 2.3. The concrete performance is worse for smaller vectors due to constant overheads of computing the CRHF and PRF relative to computing the inner product. For larger vectors, however, the inner product computation dominates the cost.

OWF-based construction. Our OWF-based construction requires computing the inner products over the integers, which requires ℓ multiplications and ℓ additions in $\mathbb{Z}$ to compute inner products. Then, we need to evaluate an m-wise independent hash function. This requires evaluating a random polynomial of degree $m = O(\lambda \cdot t^2)$ with $O(\lambda \cdot t)$-bit coefficients (recall Lemma 2.6.1). Here, we let $\lambda = 40$ as it is a statistical security parameter of the t-good hash function. For very small values of B and ℓ, we obtain reasonable concrete efficiency when evaluating the m-wise independent hash function (less than one second of computation for $B = 2$ and $\ell = 5$ and roughly 50MB public parameters). However, for larger parameters, the concrete efficiency quickly becomes impractical. This blowup is due to the quadratic overhead of Lemma 2.6.1. Additionally, the public parameters quickly become impractically large (in the *petabytes*) as ℓ increases, due to the cubic factor in t needed to describe the random polynomial. Indeed, this description already reaches terabytes in size with $B = 2$ and $\ell = 10$, barring any concretely practical instantiation.

2.9.2 Comparison to other CPRF constructions

Prior CPRF constructions for inner product (and NC^1) predicates [AMN+19, DKN+20, CMPR23] do not have implementations, and due to large parameters or heavy building blocks, are far too inefficient to be implemented. We briefly discuss the concrete efficiency roadblocks associated with these constructions.

The LWE-based CPRF construction of Davidson et al. [DKN+20] is implementable but very inefficient due to the large parameters required for security and computationally expensive building blocks. Specifically, their construction requires computing a linear (in the input size) number of matrix-matrix products, which poses an efficiency roadblock. Similar roadblocks are faced with other LWE-based constructions, even if adapted to the simpler

case of inner-product constraints. While concrete efficiency can be improved by assuming Ring LWE, the concrete costs remain high.

The construction of Attrapadung et al. [AMN+18] is tailored to evaluating NC^1 Boolean circuits and requires computing a linear number of group exponentiations in the degree of the universal NC^1 circuit computing the constraint predicate. While their construction can be theoretically applied to computing inner-product predicates, it does *not* yield a practical solution given the need for emulating field operations *inside* of the NC^1 universal circuit.

The approach of Couteau et al. [CMPR23] based on DCR requires evaluating a PRF using HSS (where the PRF key is encoded as an HSS input share). This requires evaluating a linear (in the degree of the polynomial computing the PRF) number of HSS multiplications. Using a DCR-based variant of the Naor-Reingold PRF necessitates computing $g^{\prod_i^n a_i^{x_i}}$ in HSS, where the key $k = (a_1, \ldots, a_n)$ is the PRF key provided as input. The exceedingly high degree of this polynomial eliminates the possibility of a concretely practical instantiation, since even low-degree polynomials can already be concretely expensive to evaluate in HSS schemes [BCG+17].

Chapter 3

Oblivious Transfer Extension with a Public-Key Setup

Summary

Oblivious Transfer (OT) is at the heart of secure computation and is a foundation for many applications in cryptography. Over two decades of work have led to extremely efficient protocols for evaluating OT instances in the preprocessing model, through a paradigm called OT extension. A few OT instances, generated in an offline phase, can be used to perform many OTs in an online phase efficiently, i.e., with very low communication and computational overheads.

Specifically, traditional OT extension protocols use a small number of "base" OTs, generated using any black-box OT protocol, and convert them into many OT instances using only lightweight symmetric-key primitives. Recently, a new paradigm of OT with a non-interactive *public-key setup* has emerged, which replaces the base OTs with a non-interactive setup: Using only the public key of the other party, two parties can efficiently compute a virtually unbounded number of OT instances "on the fly."

In this chapter, we put forth a novel framework for OT extension with a public-key setup (henceforth, "public-key OT") and concretely efficient instantiations. Implementations of our framework are 30–100× faster when compared to the previous state-of-the-art public-key OT protocols, and remain competitive even when compared to OT extension protocols that *do not* offer a public-key setup. Additionally, our instantiations result in the first public-key OT schemes with plausible post-quantum security.

3.1 Introduction

In its simplest form, Oblivious Transfer (OT) allows a party (called the *receiver*) to privately retrieve one out of two messages from another party (called the *sender*). The receiver has a choice bit b and the sender has a pair of messages (m_0, m_1). Using OT, the receiver learns m_b but learns nothing about m_{1-b}. Moreover, the sender is guaranteed to learn nothing about b. OT is a foundational building block for secure multi-party computation [Kil88], and its applications typically require a large number of oblivious transfers (in the millions or billions). Unfortunately, all existing protocols for OT require public-key cryptography, making them concretely inefficient in many applications. Since it is known that OT cannot be constructed in a black-box manner using only symmetric-key primitives [IR89], this inefficiency is somewhat inherent to the OT problem. Fortunately, however, since the seminal work of Beaver [Bea96] and the efficient construction of Ishai, Kilian, Nissim, and Petrank [IKNP03] (henceforth, IKNP), lightweight OT can be realized by performing a small number of expensive "base" OTs that are then extended (using only lightweight, symmetric-key cryptography) to perform any number of regular OTs. Despite its concrete computational efficiency, the original paradigm of IKNP induces a large communication overhead (λ bits of communication per extended OT). To address this overhead, new paradigms have recently emerged that enable extending base OTs using much less communication. Protocols like SoftSpokenOT [Roy22] directly improve the communication efficiency of IKNP by a small factor k (e.g., $k = 5$) at the cost of some increased computation. Silent OT extension protocols [BCG+19a, BCG+19b, SGRR19, YWL+20, CRR21, OSY21, BCG+22, RRT23] achieve optimal communication (3 bits of communication per OT), but come with a concrete computational overhead that is noticeably larger than SoftSpokenOT (e.g., RRT, the state-of-the-art silent OT [RRT23], being about 8× slower than SoftSpokenOT on machines with AVX instructions).

Our goal: "Diffie–Hellman" for secure computation. The state-of-the-art techniques for efficiently evaluating a large number of OT instances all require the sender and the receiver to initially interact in a distributed setup phase. Contrast this with the simpler task of establishing a secure *communication* channel on the Internet. Thanks to the breakthrough key-agreement protocol of Diffie and Hellman [DH76] in 1976, any pair of parties can locally derive a shared symmetric encryption key directly from the *public key* of the other party. This approach to securing communication has proven to be highly effective, and is now widely deployed [Ope24]. Concretely, this means that over a large network of N parties, all pairs of parties can securely communicate following a one-time public-key setup with $O(N)$ communication, where all parties broadcast their public keys.

The goal of *oblivious transfer with a public-key setup*, first explicitly put forth by Orlandi et al. [OSY21], is to achieve a similar feature for the task of secure *computation* on the Internet. Concretely, over a large network of N parties, if all pairs of parties want to be able to jointly run secure *computation* protocols (which typically requires evaluating many OTs), they must all run the distributed setup pairwise, resulting in $O(N^2)$ communication and simultaneous interactions. However, with a *public-key setup* (or non-interactive "public-key OT" for short), each pair of parties can instead efficiently generate an arbitrary number of pseudorandom OT instances, given only the public key of the other party! These pseudorandom OT instances can then be derandomized in one round to perform regular OTs [Bea95, OSY21, BCM+24].

Unfortunately, despite being very desirable, this feature is not achievable with any of the state-of-the-art OT extension protocols, even in the semi-honest model.

Recently, however, the work of Bui, Couteau, Meyer, Passelègue, and Riahinia [BCM+24] (henceforth, BCMPR) achieved the first practically efficient candidate construction of public-key OT by building a new Pseudorandom Correlation Function (PCF) for the OT correlation, and showing that it admits a public-key setup. Concretely, with a public-key PCF, two parties can, given only each other's public key, locally generate an arbitrary amount of pseudorandom OTs. In turn, these pseudorandom OTs can be used to perform a regular bit-OT in one round of interaction and three bits of communication). While this represents significant progress, their result falls short of providing a fully satisfactory solution to the problem of efficient public-key OT. For one, their protocol is *not* an OT extension, given that it requires (local) public-key operations for *every* OT that it generates. Consequently, it is considerably less efficient than state-of-the-art OT extension protocols. Concretely, BCMPR can generate up to 21K OTs per second, whereas state-of-the-art OT extension protocols can generate several million OTs per second [Roy22]. Additionally, BCMPR is built around group-based primitives, making it not post-quantum secure, and relies on a new assumption they call "Sparse-power DDH" (or SPDDH for short) which is only proven secure in the generic group model.

3.1.1 Our contributions

In this chapter, we make several contributions, which we highlight here and describe in depth in our technical overview of Section 3.2. The primary contribution of this chapter is QuietOT: a novel framework for fast OT extension with a public-key setup. With QuietOT, given only each other's public key, two parties can generate an arbitrary amount of pseudorandom "*ListOTs*," a variant of OT which we introduce, which can be converted into pseudorandom (resp. regular) OTs in one round and a small overhead in communication, e.g., using 4 bits/OT (resp. 7 bits/OT) in one of our instantiations. The only difference between our approach via ListOT and a standard PCF for OT is that the derandomization step incurs slightly more communication (e.g., 7 bits instead of 3 bits). Unlike all prior public-key OT protocols, QuietOT does *not* require public-key operations when generating OTs, making the concrete performance *one to six orders* of magnitude faster compared to the state-of-the-art OT protocols that offer a public-key setup. We show that the base OTs can be replaced with a public-key setup under the standard Ring LWE assumption (with a superpolynomial modulus-to-noise ratio), allowing parties to non-interactively derive a shared key from which they can generate OT extensions. Alternatively, the public-key setup of QuietOT can be replaced by a simple two-round setup using any black-box base OTs, yielding new constructions of two-round OT extension.

We note that state-of-the-art OT extension protocols, such as SoftSpokenOT [Roy22], remain significantly faster than QuietOT (e.g., about 7× faster in the regime where SoftSpokenOT communicates 16 bits/OT). The core advantage of QuietOT over these alternatives lies in its public-key setup: concretely, using QuietOT, two parties can execute the vast majority of the computation *before they even interact*, given only each other's public key. The interactive phase that follows involves solely cheap, non-cryptographic operations (a few XORs per OT). In contrast, using SoftSpokenOT or any state-of-the-art OT extension,

the parties must *first* interact (to generate base OTs) *before* they can run the bulk of the computation and interact again to complete the protocol; this can cause significant delays during which both parties have to stay online. We believe that this precomputation feature of QuietOT is highly desirable in the setting of on-demand pairwise secure computation over a large network. As a bonus, QuietOT communicates less than SoftSpokenOT, and requires only one round of interaction to perform an OT.

Under the hood, our framework combines any "Inner-Product Membership" weak PRF (IPM-wPRF) [BCM+24] with a Shiftable Constrained Pseudorandom Function (ShCPRF), a new primitive that we introduce in Section 3.5 that extends the CPRF construction from Chapter 2. Prior work [OSY21, BCM+24] requires using public-key operations for each OT, translating to expensive group operations under either the Quadratic Residuosity (QR) or DDH assumption (the construction of Orlandi et al. [OSY21], henceforth OSY, is mostly of theoretical interest due to the large number of group exponentiations required). In contrast, we show that an ShCPRF can be constructed unconditionally in the random oracle model by using the construction from Chapter 2. In addition, because IPM-wPRF are lightweight symmetric-key primitives (i.e., they do not require public-key operations to evaluate), our overall OT extension protocol is very efficient. We provide a comparison to related work in Table 3.1.

	OT/s Max. Throughput	**Bits/OT** Communication	**PKS**†	**PQ**	**Assumptions**
IKNP	34,000,000	128	✗	✓	ROM
SoftSpokenOT ($k = 2$)	53,000,000	64	✗	✓	ROM
SoftSpokenOT ($k = 8$)	9,500,000	16	✗	✓	ROM
RRT	6,900,000	3	✗	✓	EC-LPN+ROM
OSY	1	3	✓	✗	QR+ROM
BCMPR	21,000	3	✓	✗	IPM-wPRF+SPDDH+ROM
QuietOT	1,200,000	7	✓	✓	IPM-wPRF+ROM

Table 3.1: An overview of OT extension protocols and their maximum throughput observed across different hardware and parameter settings (full evaluation results provided in Section 3.8). PKS and PQ indicate whether the construction has a *public-key setup* and is plausibly *post-quantum* secure, respectively. For efficiency, nearly all OT extension protocols are instantiated in the Random Oracle Model (ROM). Note that in these constructions, the random oracle assumption can be generically replaced with a suitable correlation-robust hash function. †PKS not implemented.

Additional contributions. In addition to the main contribution of the QuietOT framework, this chapter contributes:

- The first formal treatment of public-key OT when used in a secure multi-party computation over a large network. Our definitions and analysis address several subtle issues with using any public-key OT constructions (including BCMPR and OSY) in a multi-party setting where security must be guaranteed with respect to an adversary corrupting a subset of parties.

- A definition and construction of shiftable CPRFs, a twist on CPRFs where the master key holder can efficiently "shift" the constraint when evaluating the CPRF. This construction plays a crucial role in our framework and may be of independent interest.
- An open-source, optimized implementation of the BCMPR protocol, which we evaluate and compare to QuietOT in Section 3.8.

3.2 Technical Overview

In this section, we provide a detailed overview of our results. In Section 3.2.1, we start by covering the state-of-the-art BCMPR framework for public-key OT. Then, in Section 3.2.2, we cover the main ideas behind our QuietOT framework and the different instantiations of it. In Section 3.2.3, we show how QuietOT implies two-round OT extension and full pre-computability for either the sender or the receiver. In Section 3.2.4, we explain our approach to non-interactive public-key setup under the RLWE assumption. In Section 3.2.5, we overview our definitions of public-key setup with *multi-instance* security, which becomes a crucial building block for applying public-key OT to a multi-party computation setting.

3.2.1 Background on the BCMPR framework

The BCMPR framework constructs a pseudorandom correlation function (PCF) for OT correlations using an IPM-wPRF (an "inner-product membership" weak PRF; more details on this primitive are given later). In addition to the IPM-wPRF requirement, they also require the Sparse-Power DDH (SPDDH) assumption and instantiate their public-key setup from the DCR assumption. In their PCF construction, the sender and receiver can compute OT correlations on-demand: The sender outputs two pseudorandom bits (s_0, s_1) while the receiver outputs (b, s_b), where $b \in \{0, 1\}$ is a pseudorandom choice bit. This correlation can then be converted into a chosen-bit OT with 3 bits of communication in one round of interaction using the transformation of Beaver [Bea95].

At the heart of the BCMPR framework is a Constrained PRF (CPRF) $F = (F.\mathsf{KeyGen}, F.\mathsf{Eval}, F.\mathsf{Constrain}, F.\mathsf{CEval})$ with the constraint predicate set to a weak PRF[1] $f_{\mathbf{z}}$ that outputs a pseudorandom bit. At a high level, a CPRF has two keys: a master key and a constrained key. The constrained key only allows evaluating the PRF when the constraint predicate is satisfied. Hence, the weak PRF f indicates if the input to F is constrained or not, making roughly half the inputs to F constrained. It was known (somewhat folklore) that any CPRF with a weak PRF as a constraint predicate can be used to construct a PCF for OT correlations [BGMM20]. However, prior to BCMPR, all existing CPRF constructions were either not sufficiently expressive to evaluate a weak PRF as a predicate or not concretely efficient enough to result in practical solutions [BV15, BTVW17, CC17, CVW18, PS18, AMN+18, CMPR23]. Therefore, at the core of BCMPR is a construction of a CPRF *just* powerful enough to evaluate a suitable weak PRF candidate as the constraint predicate while remaining concretely efficient in practice. They realize such a CPRF by adapting the classical Naor–Reingold PRF [NR97]. In a nutshell, the BCMPR framework for OT correlations combines:

[1] A weak PRF is only pseudorandom on *uniformly random* inputs.

- A CPRF supporting a special class of *Inner-Product Membership* (IPM) constraints (the constrained key can evaluate the PRF on x if and only if $\langle \mathbf{z}, x \rangle \in S$, for some constraint vector $\mathbf{z} \in \mathcal{R}^n$ defined over a finite ring $\mathcal{R}$ and fixed set S partitioning the ring elements)[2], and
- Any weak PRF $f_{\mathbf{z}} : \{0,1\}^n \to \{0,1\}$ having an evaluation function that can be described as an inner-product membership predicate (which they call an IPM-wPRF). That is, $f_{\mathbf{z}}(x) = 0$ iff $\langle \mathbf{z}, x \rangle \in S_0$ and $f_{\mathbf{z}}(x) = 1$ iff $\langle \mathbf{z}, x \rangle \in S_1$, for a partitioning $S_0 \cup S_1$ of the finite ring $\mathcal{R}$ over which the inner product is defined, and a vector $\mathbf{z} \in \mathcal{R}^n$.

Building on the Naor–Reingold PRF, BCMPR constructs a CPRF supporting IPM predicates in the Random Oracle Model (ROM) [BR93]. In particular, using any IPM-wPRF (which can be realized from a handful of assumptions), coupled with their CPRF construction supporting IPM predicates, allows them to instantiate the following generic template for building a PCF for OT correlations, which will serve as inspiration for our framework as well.

A general PCF template from CPRFs for IPM predicates. The template of BCMPR uses a CPRF F with IPM constraints that evaluates an IPM-wPRF $f_{\mathbf{z}}$ as the predicate, for a vector $\mathbf{z} \in \mathcal{R}^n$ that we will call the wPRF key. The sender gets two master keys $(\mathsf{msk}_0, \mathsf{msk}_1)$ for F, while the receiver obtains two constrained keys $(\mathsf{csk}_0, \mathsf{csk}_1)$. The master keys can be used to evaluate the PRF on the entire domain. In contrast, the constrained key can only be used to evaluate the PRF when the constraint predicate is satisfied. The idea is to have the constrained key csk_0 have $f_{\mathbf{z}}$ as the predicate, and csk_1 have the opposite predicate $1 - f_{\mathbf{z}}$. Notice that given the two constrained keys, the receiver can only evaluate the CPRF using *one* of the two keys for an input x (depending on the value of $f_{\mathbf{z}}(x)$, which is pseudorandom). Moreover, given the IPM-wPRF key $\mathbf{z}$, the receiver can determine which of the two evaluations is constrained for an input x by evaluating the "predicate" $f_{\mathbf{z}}(x)$. The receiver can then compute and output the correlation (b, s_b), consisting of the pseudorandom bit $b = f_{\mathbf{z}}(x)$ and the string $s_b = F.\mathsf{CEval}(\mathsf{csk}_b, x)$. The sender, in contrast, only obtains the master keys $(\mathsf{msk}_0, \mathsf{msk}_1)$, which are independent of the IPM-wPRF key $\mathbf{z}$. As such, the sender can only compute the strings $s_0 = F.\mathsf{Eval}(\mathsf{msk}_0, x)$ and $s_1 = F.\mathsf{Eval}(\mathsf{msk}_1, x)$, consisting of the sender's correlation (s_0, s_1), without learning the pseudorandom bit b computed by the receiver.

Limitations of the general template. The core difficulty associated with the above template (and the BCMPR framework by extension) is finding a CPRF with a predicate class that is sufficiently powerful to evaluate $f_{\mathbf{z}}$. The most efficient construction to date is the constrained Naor–Reingold PRF construction of BCMPR, which (1) requires a new cryptographic assumption, (2) is not post-quantum secure and, (3) necessitates concretely expensive group operations to evaluate, placing an upper limit on practical efficiency of BCMPR (e.g., 21K correlations per second in our optimized implementation). In contrast, OT extension protocols like SoftSpokenOT [Roy22] (which generalizes IKNP) use only lightweight symmetric-key primitives, are post-quantum secure, and are incredibly fast (e.g., achieving several million OTs per second), but do not offer a public-key setup. Unfortunately, improving the efficiency of the BCMPR framework hinges on developing more efficient CPRF constructions for IPM predicates, which appears to be the weakest class of predicates

[2]We slightly abuse notation by interpreting the bit string x as a *vector* of bits.

sufficiently powerful to evaluate any wPRF. Note that a pseudorandom function *cannot* have a linear evaluation, and therefore inner-product equality predicates are inherently insufficient.

3.2.2 Our approach

Intuition. The starting point of our approach is the template construction of BCMPR. As with BCMPR, in our framework, the receiver holds the key $\mathbf{z} \in \mathcal{R}^n$ of an IPM-wPRF and we let the (pseudorandom) selection bit of the receiver be defined as the output $f_{\mathbf{z}}(x)$ of the wPRF f on a random input x. Recall that $f_{\mathbf{z}}(x) = b$ iff $\langle \mathbf{z}, x \rangle \in S_b$ (where S_0, S_1 are a public partitioning of the inner-product range, associated with the IPM-wPRF). The main limitation of BCMPR is the reliance on a CPRF for a class of constraints that contains $f_{\mathbf{z}}$. While they provide an optimized construction, it still requires public-key operations (group exponentiation) for *every* evaluation, and hence for every OT instance.

At this point, we diverge significantly from the BCMPR framework by replacing their CPRF with a far more efficient primitive. Our starting point is the CPRF construction from Chapter 2, which uses only symmetric-key primitives. Concretely, evaluating the CPRF involves computing an inner product and hashing the result; furthermore, the CPRF was shown to be unconditionally secure in the ROM. However, the catch is that the CPRF from Chapter 2 only handles inner-product predicates. That is, given a constraint $\mathbf{z}$, the constrained evaluation with csk on x matches the evaluation with the master key if and only if $\langle \mathbf{z}, x \rangle = 0 \in \mathcal{R}$. Observe that using this much weaker CPRF, the receiver is now only able to evaluate F on all inputs where $\langle \mathbf{z}, x \rangle = 0$[3] (roughly $\frac{1}{|S_b|}$ of all inputs assuming $f(x) = b$, and where $|S_b| \approx |\mathcal{R}|/2$), which is too weak to instantiate the BCMPR template.

Shiftable CPRFs to the rescue. Our first key observation is that (a slight modification of) the CPRF framework introduced in Chapter 2 enjoys an additional *shiftability* property. Concretely, the CPRF evaluation with the master key msk can take an additional *shift* α as input, and provides the following guarantee: the constrained evaluation $F.\mathsf{CEval}(\mathsf{csk}, x)$ is equal to $F.\mathsf{Eval}(\mathsf{msk}, x, \alpha)$ whenever $\langle \mathbf{z}, x \rangle - \alpha = 0$. That is, the constraint is *shifted by* α. Given such a shiftable CPRF for inner products, the sender can now compute $F.\mathsf{Eval}(\mathsf{msk}, x, \alpha)$ for *all possible shifts* $\alpha \in S_0 \cup S_1$. This yields two lists of values: $L_0 = (F.\mathsf{Eval}(\mathsf{msk}, x, \alpha))_{\alpha \in S_0}$ and $L_1 = (F.\mathsf{Eval}(\mathsf{msk}, x, \alpha))_{\alpha \in S_1}$. Our next core observation is that the value $F.\mathsf{CEval}(\mathsf{csk}, x)$ computed by the receiver belongs to exactly one of the two lists, and furthermore, *the index b of the list L_b it belongs to is simply $f_{\mathbf{z}}(x)$*. That is, the receiver knows a pseudorandom value v and pseudorandom "selection bit" $b = f_{\mathbf{z}}(x)$ such that $v \in L_b$. Additionally, by using the constraint $\mathbf{z}$, the receiver can determine the index i in L_b in which v is located (i.e., such that $v = L_b[i]$).

Oblivious transfer from ListOT. So far, we have seen that given a shiftable CPRF for inner-product predicates, the sender and the receiver can generate many instances of the following "correlation:" the sender gets as output two (pseudorandom) lists (L_0, L_1), and the receiver obtains (v, b, i) where $v = L_b[i]$, and b is pseudorandom from the viewpoint of the sender. Importantly, i is *not* pseudorandom, which prevents this from being a true OT correlation. We call "ListOT" this weaker variant of the OT correlation. The name is inspired

[3] We follow the convention of letting $P(x) = 0$ when the predicate P is satisfied.

from *list decoding* [Eli91], where a decoding algorithm for a code is allowed to output a list of code words from which the word can be decoded.[4] Hence, for ListOT, the sender outputs two lists of messages, L_0, L_1, and the receiver outputs a bit b, value v, *and* an index key i, such that v is located at $L_b[i]$. (Later, for ease of notation, L_0 and L_1 will be treated as key-value stores/dictionaries.)

While the pseudorandom ListOT instances are *not* correlations in the strict technical sense (because the distribution of i depends on the secret wPRF key), it is not too hard to see that it still suffices to instantiate a random OT using some additional communication. To see this, observe that given OT inputs (m_0, m_1), the sender simply sends $(L_0[j] \oplus m_0)_{j \in S_0}$ and $(L_1[j] \oplus m_1)_{j \in S_1}$ to the receiver. The receiver recovers m_b by unmasking $L_b[i] \oplus m_b$ using $v = L_b[i]$.[5]

We now explain how we construct efficient ShCPRFs by adapting the framework from Chapter 2 building CPRFs for inner-product predicates from RKA-secure PRFs [Bih94] in the standard model (or in the random oracle model).

Constructing ShCPRFs. We make the observation that in all existing CPRF constructions for inner-product predicates [DKN+20, BCM+24], the master key holder can efficiently compute the set of all possible pseudorandom values evaluated under the constrained key csk. We will focus on the CPRF construction from Chapter 2, instantiated unconditionally using a hash function H modeled as a random oracle. In this construction, the master key msk consists of a random vector $\mathbf{z_0}$ of length n, with elements from some sufficiently large field $\mathbb{F}$.[6] For a constraint vector $\mathbf{z} \in \mathbb{F}^n$, the constrained key is defined as $\mathbf{z_1} = \mathbf{z_0} - \Delta \cdot \mathbf{z}$, where $\Delta \in \mathbb{F} \setminus \{0\}$ is random. Simplifying slightly,[7] the evaluation and the constrained evaluation algorithms are defined as $H(\langle \mathbf{z_0}, x \rangle, x)$ and $H(\langle \mathbf{z_1}, x \rangle, x)$, respectively. Note that when $\langle \mathbf{z}, x \rangle = 0$, it holds that $H(\langle \mathbf{z_0}, x \rangle, x)$ is equal to $H(\langle \mathbf{z_1}, x \rangle, x)$, which guarantees the master key and constrained key evaluations agree. In contrast, when $\langle \mathbf{z}, x \rangle \neq 0$, then $H(\langle \mathbf{z_1}, x \rangle, x)$ is equal to $H(\langle \mathbf{z_0}, x \rangle - \Delta \langle \mathbf{z}, x \rangle, x)$, which is independent of $H(\langle \mathbf{z_0}, x \rangle, x)$ due to Δ. In particular, we observe that when $\langle \mathbf{z}, x \rangle \neq 0$, using $\mathbf{z_0}$ and Δ allows the master key holder to evaluate *all possible* constrained evaluations by computing $H(\langle \mathbf{z_1}, x \rangle + \Delta\alpha, x)$, for all possible inner products $\alpha \in \{\langle \mathbf{z}, x \rangle \mid x \in \{0,1\}^n\}$ associated with the constraint class given by $\mathbf{z}$. We point to Section 3.4 for more details on this ShCPRF construction. Abstractly, we define the master key evaluation algorithm $F.\mathsf{Eval}(\mathsf{msk}, x, \alpha)$ to take a shift α as an additional input while leaving the remaining CPRF algorithms unchanged.

Putting things together: A "PCF" for ListOT. Using the ShCPRF construction sketched above, coupled with an IPM-wPRF $f_{\mathbf{z}}$ with partitioning $S_0 \cup S_1$, the sender with the master secret key msk computes the two lists, L_0 and L_1, corresponding to $b = 0$ and $b = 1$, respectively, as $L_0 = (F.\mathsf{Eval}(\mathsf{msk}, x, \alpha))_{\alpha \in S_0}$, $L_1 = (F.\mathsf{Eval}(\mathsf{msk}, x, \beta))_{\beta \in S_1}$, using a random x. Importantly, note that given the constrained key csk for a constraint vector

[4] We note that ListOT is not related to "list two-party computation" [COSW23], which defines list OT as a security definition for the standard oblivious transfer functionality.

[5] When generating (pseudo)random OTs, this simple approach can be further improved by letting m_0 and m_1 be the first element of L_0 and L_1 respectively, which allows communicating two elements less, for a total of $|S_0| + |S_1| - 2$ bits of communication. Concretely, with our most communication-efficient instance, this translates to only 4 bits of communication per random OT.

[6] Our actual ShCPRF construction is defined using a ring extension for efficiency.

[7] The full construction has an extra additive term to handle the all-zero input $x = 0^n$.

$\mathbf{z}$, the receiver obtains *one* value in L_b, where $b = f_{\mathbf{z}}(x)$. All other values, in both lists, remain pseudorandom from the viewpoint of the receiver. At this stage, our framework can be instantiated using any choice of ShCPRF and any choice of IPM-wPRF. We choose to instantiate the ShCPRF in the random oracle model, as it offers the most concretely-efficient solution. The IPM-wPRF can be realized from several assumptions, as detailed in BCMPR. Indeed, many wPRFs fit the IPM-wPRF framework, including the learning with rounding (LWR)-based wPRF [BPR12], the Goldreich–Applebaum–Raykov (GAR) [Gol11, AR16], and several other low-complexity wPRF candidates, including the Boneh, Ishai, Passelègue, Sahai, and Wu (BIPSW) [BIP+18], and LPN-based candidates [BCG+20a]. (Bui et al. [BCM+24] provide an overview of these different candidates and others.) The BIPSW wPRF candidate is especially well-suited to this framework given that the evaluation (defined in Equation 3.1) is essentially just a rounded inner product computed in $\mathbb{Z}_6$:

$$f_{\mathbf{z}}(x) = \lfloor \langle \mathbf{z}, x \rangle \mod 6 \rceil_2 . \tag{3.1}$$

Note that when $f_{\mathbf{z}}(x) = 0$, then it holds that $\langle \mathbf{z}, x \rangle \pmod 6 \in \{0, 1, 2\}$ and when $f_{\mathbf{z}}(x) = 1$ it holds that $\langle \mathbf{z}, x \rangle \pmod 6 \in \{3, 4, 5\}$. By instantiating the ShCPRF to compute predicates over an extension of $\mathbb{Z}_6$, we can achieve very efficient evaluations using the BIPSW IPM-wPRF (see Section 3.8 for our evaluation).

3.2.3 Two-round OT extension

Using our framework, we obtain a *two-round* OT extension protocol (i.e., the minimal number of rounds needed without a public-key setup). Observe that the sender can independently generate the ShCPRF master secret key, consisting of $\mathbf{z_0}$ and Δ, while the receiver can independently generate the IPM-wPRF key $\mathbf{z}$. For the case where $\mathbf{z} \in \{0,1\}^n$, we can use any two-round string OT protocol repeated in parallel n times as follows. For $i \in [n]$, the sender sets $m_{i,0} = \mathbf{z_0}_i$ and $m_{i,1} = \mathbf{z_0}_i - \Delta$. The receiver uses $\mathbf{z}_i \in \{0,1\}$ as its choice bit to retrieve $m_{i,\mathbf{z}_i} = \mathbf{z_0}_i - \Delta \mathbf{z}_i$, and in this way can recover $\mathsf{csk} := \mathbf{z_0} - \Delta \mathbf{z}$ using n parallel calls to the two-round OT functionality (indeed, because Δ is the same across messages, any *correlated* OT protocol [ALSZ13] is sufficient). In the general case, when $\mathbf{z} \in \mathcal{R}^n$, we can use any two round "reverse" *vector oblivious linear evaluation* (VOLE) protocol [ADI+17, BCG+19a], which directly generalizes correlated OT to work over a ring $\mathcal{R}$. In reverse VOLE, the sender inputs $(\mathbf{b}, x) \in \mathcal{R}^n \times \mathcal{R}$ and the receiver inputs $\mathbf{a} \in \mathcal{R}^n$. The sender gets no output while the receiver obtains $\mathbf{a}x + \mathbf{b}$. By letting the sender input $(\mathbf{z_0}, \Delta)$ and the receiver input $-\mathbf{z}$, we immediately have that the receiver obtains $\mathbf{z_1} = \mathbf{z_0} - \Delta \mathbf{z}$. See Section 3.10 for more details.

Two-round OT extension is known to be impossible under black-box symmetric-key primitives [GMMM18] making our use of an IPM-wPRF a rather weak assumption to circumvent the impossibility result of Garg et al. [GMMM18] (in fact, an IPM-PR$\mathbf{G}$ suffices). In contrast, protocols like IKNP and SoftSpokenOT inherently require three rounds of interaction due to their unconditional instantiations in the random oracle model, and all previous two-round OT extensions (with the exception of Beaver [Bea96], which is not black-box and not concretely efficient) required variants of the LPN assumptions [BCG+19a, YWL+20, BCG+22, RRT23].

Precomputability. A nice feature of our two-round setup is the ability for one party to

precompute all correlations *before even knowing the identity of the other party.* To see this, note that the receiver can precompute all choice bits just using the IPM-wPRF key $\mathbf{z}$ without needing to know the constrained key. Additionally, the receiver can sample a *uniformly random* constrained key $\mathbf{z_1}$ for the ShCPRF and use it to generate ahead-of-time all its ListOT triples (b, i, v). Later, once the identity of the sender is known, the sender can engage with the receiver in a two-round OT protocol to compute the master key $\mathbf{z_0} = \mathbf{z_1} + \Delta\mathbf{z}$ *from* the "constrained key" $\mathbf{z_1}$. Similarly, the sender can alternatively generate all its ListOT instances (L_0, L_1) without needing to know the identity of the receiver by locally sampling Δ and $\mathbf{z_0}$. We provide details on precomputability and more motivation for the notion in Section 3.10.1.

3.2.4 Public-key Setup from Ring LWE

We present a non-interactive distributed setup protocol from RLWE. To the best of our knowledge, this forms the first distributed setup protocol for PCF for ListOT based on a plausibly post-quantum assumption. The goal of this protocol is for the sender with input Δ and receiver with input $\mathbf{z}$ to distributively derive keys $\mathbf{z_0}$ (part of the master secret key) and $\mathbf{z_1}$ (the constrained key), which can be viewed as additive shares of $\Delta \cdot \mathbf{z}$ in a ring $\mathcal{R}$.

Parameters. In order to rely on the security of RLWE, the receiver will "encode" the bits of $\mathbf{z} \in \mathcal{R}^n$ into the coefficients of an element z of a suitable polynomial ring $\mathcal{P}$. The protocol is executed over $\mathcal{P}$ and then, at the end of the protocol, the sender and receiver each "decode" their result back into the ring $\mathcal{R}$ to obtain vectors $\mathbf{z_0}$ and $\mathbf{z_1}$, by parsing each polynomial as a vector of n coefficients (and disregarding any extra coefficients).

Assume that $\mathcal{R} = \mathbb{Z}_t$ is an integer ring, and let $q := n \cdot t \cdot B \cdot 2^{\omega(\log \lambda)}$ (B is some bound on the noise that we compute later). We define $\mathcal{P} := \mathbb{Z}_q[X]/(X^\eta + 1)$, where η is a power of 2 that is larger than n. Let $\chi = \chi(\mathcal{P})$ be a suitable noise distribution over $\mathcal{P}$, such that for $e_0, e_1 \xleftarrow{R} \chi$, it holds that $\|e_0 e_1\|_\infty \leq B/3$, with overwhelming probability.

The protocol proceeds in two phases as follows. During the public-key generation phase, the sender and receiver each broadcast a public key, which is used by the other party in the ShCPRF evaluation key derivation phase.

Step 1: Generating public keys. Fix random $a_0, a_1 \in \mathcal{P}$ as part of the public parameters. To generate public keys, the sender and receiver proceed as follows. These public keys can then be posted to a bulletin board or broadcasted.

Sender

1: Sample secret $s_0 \xleftarrow{R} \chi$.
2: Sample error $e_0 \xleftarrow{R} \chi$.
3: Set $\mathsf{pk}_S = \Delta \cdot a_0 + s_0 a_1 + e_0$.

Receiver

1: Encode $\frac{q}{t} \cdot \mathbf{z}$ as $z \in \mathcal{P}$.
2: Sample secret $s_1 \xleftarrow{R} \chi$.
3: Sample errors $e_1, e_1' \xleftarrow{R} \chi$.
4: Set $\mathsf{pk}_R = (z + s_1 a_0 + e_1, s_1 a_1 + e_1')$.

Step 2: Deriving ShCPRF keys. To derive a master key msk and constrained key csk, respectively, the sender and receiver use the other party's public key to proceed as follows. (Here and throughout, we overload rounding $\lceil \cdot \rfloor_t$ notation to include "rounding" a polynomial coefficient-by-coefficient.)

Sender

1: Compute $z_0 := \lceil \langle \mathsf{pk}_R, (\Delta, s_0) \rangle \rfloor_t$.
2: Decode $z_0 \in \mathcal{P}$ as $\mathbf{z_0} \in \mathcal{R}^n$.
3: Set $\mathsf{msk} := (\mathbf{z_0}, \Delta)$.

Receiver

1: Compute $z_1 := \lceil \mathsf{pk}_S \cdot s_1 \rfloor_t$.
2: Decode $z_1 \in \mathcal{P}$ as $\mathbf{z_1} \in \mathcal{R}^n$.
3: Set $\mathsf{csk} := \mathbf{z_1}$.

Correctness. The inner products computed in the key derivation phase are, in fact, noisy additive shares of $\Delta \cdot z \in \mathcal{P}$, since we have that

$$\begin{aligned}
&\langle \mathsf{pk}_R, (\Delta, s_0) \rangle - (\mathsf{pk}_S \cdot s_1) \\
&= \Delta \cdot z + \Delta \cdot a_0 s_1 + \Delta \cdot e_1 + s_0 a_1 s_1 + s_0 e_1' - \Delta \cdot a_0 s_1 - s_0 a_1 s_1 - e_0 s_1 \\
&= \Delta \cdot z + \underbrace{\Delta \cdot e_1 + s_0 e_1' - e_0 s_1}_{\text{noise}} \approx \Delta \cdot z.
\end{aligned}$$

Note that $\Delta \in \mathbb{Z}_t$ has low norm, so we can bound the magnitude of the noise term $\Delta \cdot e_1 + s_0 e_1' - e_0 s_1$ by B. Hence, by a standard rounding lemma [DHRW16, BKS19], $z_0 - z_1 = \lceil \langle \mathsf{pk}_R, (\Delta, s_0) \rangle \rfloor_t - \lceil \mathsf{pk}_S \cdot s_1 \rfloor_t = \Delta \cdot z \bmod t$. After parsing as vectors over $\mathcal{R}^n$, we have $\mathbf{z_0} - \mathbf{z_1} = \Delta \cdot \mathbf{z}$.

Security. Pseudorandomness of the public keys follows from the RLWE assumption with short secrets (i.e., *normal form* RLWE).[8] In the sender public key, the RLWE sample $s_0 a_1 + e_0$ masks $\Delta \cdot a_0$ and thus the secret key Δ. Similarly, in the receiver public key, the RLWE sample $s_1 a_0 + e_1$ masks z and thus the secret key $\mathbf{z}$. For a complete description of our protocol, its parameters, and proof of security, we refer to Section 3.7.5.

3.2.5 Multi-instance security

An immediate application of QuietOT (and public-key OT schemes in general [OSY21, BCM+24]) is for efficient large-scale MPC. At a high level, with QuietOT, parties can, using just the public keys of all other parties, create pairwise OT channels for the purpose of running a secure computation (e.g., as in the GMW protocol [GMW87]). This application was also described in prior public-key OT constructions [OSY21, BCM+24] but was never formalized. We make the rather subtle observation that existing definitions [OSY21, BCM+24] for public-key OT only require *one-time* security—i.e., privacy for the sender or receiver is not considered when the same public keys are reused with different parties.

To address this gap and properly define public-key OT, we formalize the notion of "multi-instance security" in Section 3.7. In a nutshell, our definition captures a setting where parties (re)use a *long-term* secret (that depends on the public-key) and an *ephemeral* secret that is generated for each new session. We then prove that our public-key setup satisfies multi-instance security.

[8] Normal form RLWE is a standard variant of RLWE where the secret is sampled from the noise distribution instead of uniformly. It is known to be as hard as regular RLWE and is often used for practical schemes [LPR13, ACC+18, dCJV21, MW22].

3.3 Preliminaries

3.3.1 Notation

We let $\mathbb{N}$ denote the set of natural numbers, $\mathbb{Z}$ denote the set of integers, and $\mathbb{G}$ denote a finite group. We let $\mathcal{R}$ denote a finite ring. We denote by $\mathsf{poly}(\cdot)$ the set of all polynomials and by $\mathsf{negl}(\cdot)$ any negligible function. We occasionally abuse notation and let poly denote a fixed polynomial.

Sampling and assignment. We let $x \xleftarrow{R} S$ denote a uniformly random sample drawn from a set S. We let $x \leftarrow \mathcal{A}$ denote assignment from a randomized algorithm $\mathcal{A}$ and $x := y$ denote initialization of x to the value of y (which may be the output of a deterministic algorithm).

Vectors and matrices. We denote a vector $\mathbf{v}$ using bold lowercase letters and a matrix $\mathbf{A}$ using bold uppercase letters. The i-th coordinate of a vector $\mathbf{v}$ is denoted by $\mathbf{v}[i]$ (we will also occasionally abuse notation and write $\mathbf{v}[i]$ to look up a value associated with key i in a key-value list). The i-th bit of a bit-string s is denoted by s_i. For a ring $\mathcal{R}^n$, we define $\Delta \cdot \alpha$ for $\alpha \in \mathcal{R}^n$ as the coordinate-wise scalar product.

Efficiency and indistinguishability. By an *efficient* algorithm $\mathcal{A}$ we mean that $\mathcal{A}$ is modeled by a (possibly non-uniform) Turing Machine that runs in probabilistic polynomial time. We write $D_0 \approx_c D_1$ to mean that two distributions D_0 and D_1 are *computationally* indistinguishable to all efficient distinguishers $\mathcal{D}$ and $D_0 \approx_s D_1$ to mean that D_0 and D_1 are *statistically* indistinguishable.

Rounding. We let $\lfloor x \rceil$ denote the rounding of a real number x to the nearest integer. For integers $q > p \geq 2$, we define the modular rounding function $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ as $\lfloor v \rceil_p = \lfloor (p/q) \cdot v \rceil$.

Party identifiers. We identify parties with letters A and B, and use $\sigma \in \{A, B\}$ to refer to a party. We will slightly abuse notation by letting $\overline{\sigma}$, for some $\sigma \in \{A, B\}$, refer to the party identifier in the singleton set $\{A, B\} \setminus \{\sigma\}$.

3.3.2 Cryptographic definitions

Here, we recall the cryptographic definitions that we will use throughout the chapter. In Section 3.3.2.1, we define the notion of an IPM-wPRF. In Section 3.3.2.2, we cover the definition of Ring LWE and the basics of modular rounding.

3.3.2.1 Inner-Product Membership PRF.

We define the notion of a weak PRF (wPRF)[9] that can be evaluated using the "inner-product membership" formalism introduced by Bui et al. [BCM+24].

Definition 3.3.1 (Inner-Product Membership wPRF (IPM-wPRF) [BCM+24])**.** *Let λ be the security parameter and $\mathcal{R} = \mathcal{R}$ be a finite ring. Let $S_0 = S_0^{(\lambda)}$ be a (polynomially-sized) subset of $\mathcal{R}_\lambda$, and set $S_1 := \mathcal{R} \setminus S_0$. Then, $f := \{f_\lambda \colon \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \{0,1\}\}_{\lambda \in \mathbb{N}}$ is an inner-product*

[9] A weak PRF is pseudorandom on *uniformly random* inputs.

membership weak PRF (IPM-wPRF) family with respect to the partitioning (S_0, S_1)*, if it satisfies the following three properties:*

(1) $\mathcal{K}_\lambda = \mathcal{X}_\lambda = \mathcal{R}_\lambda^n$ *for some* $n = n(\lambda)$*,*

(2) its evaluation can be expressed as an inner product membership, i.e., for each $\lambda \in \mathbb{N}$*,* $\mathbf{z} \in \mathcal{K}_\lambda$*,* $\mathbf{x} \in \mathcal{X}_\lambda$*, we have that*

$$f_{\mathbf{z}}(\mathbf{x}) = \begin{cases} 0, & \text{if } \langle \mathbf{z}, \mathbf{x} \rangle \in S_0 \\ 1, & \text{otherwise (i.e., } \langle \mathbf{z}, \mathbf{x} \rangle \in S_1\text{),} \end{cases}$$

where $\langle \cdot, \cdot \rangle$ *is the standard (simple) inner product on* $\mathcal{R}^n$*, and*

(3) it achieves the standard notion of a secure (weak) PRF [KL07].

3.3.2.2 Ring learning with errors and rounding.

We recall the standard ring learning with errors (RLWE) assumption of Lyubashevsky et al. [LPR10] and its normal form.

Definition 3.3.2 (The Ring LWE assumption [LPR10])**.** *Let* λ *be a security parameter. Let* $\eta = \eta(\lambda), q = q(\lambda) \in \mathbb{N}$ *be polynomial in* λ*. Define the polynomial ring* $\mathcal{P} = \mathbb{Z}_q[X]/(X^\eta + 1)$ *and let* $\chi = \chi(\lambda)$ *be an error distribution over* $\mathcal{P}$*. The* $\mathsf{RLWE}_{\eta,q,\chi}$ *assumption states that for any* $t = t(\lambda) \in \mathsf{poly}(\lambda)$*, it holds that*

$$(\mathbf{a}, s \cdot \mathbf{a} + \mathbf{e}) \approx_c (\mathbf{a}, \mathbf{u}),$$

where $s \xleftarrow{R} \mathcal{P}, \mathbf{a} \xleftarrow{R} \mathcal{P}^t, \mathbf{e} \xleftarrow{R} \chi^t, \mathbf{u} \xleftarrow{R} \mathcal{P}^t$*. The "normal form"* $\mathsf{RLWE}_{\eta,q,\chi}$ *assumption states that this holds even when* $s \xleftarrow{R} \chi$*, and is implied by the original formulation [LPR13, Lemma 2.24].*

Modular rounding. We let $\lfloor x \rceil$ denote the rounding of a real number x to the nearest integer. For integers $q > p \geq 2$, we define the modular rounding function $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ as $\lfloor v \rceil_p = \lfloor (p/q) \cdot v \rceil$.

Rounding lemma. We recall the following "rounding lemma" [DHRW16, BKS19, CZ22]:

Lemma 3.3.1 (Rounding of Noisy Secret Shares)**.** *Let* (t, q) *be two integers such that* t *divides* q*. Fix any* $z \in \mathbb{Z}_q$ *and let* (z_0, z_1) *be any two random elements of* $\mathbb{Z}_q$ *subject to* $z_0 + z_1 = (q/t) \cdot z + e \bmod q$*, where* e *is such that* $q/(t \cdot |e|) \geq \lambda^{\omega(1)}$*. Then, with probability at least* $1 - (|e| + 1) \cdot t/q \geq 1 - \lambda^{-\omega(1)}$*, it holds that* $\lfloor z_0 \rceil_t + \lfloor z_1 \rceil_t = z \bmod t$*, and the probability is over the random choice of* $(z_0, z_1) \in \mathbb{Z}_q \times \mathbb{Z}_q$*.*

3.3.3 Oblivious transfer

We define oblivious transfer functionality in Figure 3.1. The sender inputs two messages m_0 and m_1, while the receiver inputs a choice bit $b \in 0, 1$. The functionality then ensures that the receiver learns only their chosen message m_b (and learns no information on the other message m_{1-b}), and the sender learns nothing about the choice bit b.

Functionality $\mathcal{F}_{\mathsf{OT}}$

Parameters. String length k.

Parties. The functionality interacts with a sender S, receiver R.

Procedure.

1: Wait for input $(m_0, m_1) \in \{0,1\}^k$ from S.
2: Wait for input $b \in \{0,1\}$ from R.
3: Output m_b to the R and $\perp$ to S.

Figure 3.1: Oblivious transfer ideal functionality $\mathcal{F}_{\mathsf{OT}}$.

3.3.3.1 RKA-secure PRFs from random oracles

We recall the definition of RKA-secure PRFs from Chapter 2. In this section, we include, for completeness, a proof of the (folklore) fact that when H is modeled as a random oracle, the function $F_k(x) = H(k, x)$ (for a suitable choice of the domain of k) is an RKA-secure PRF for affine relations.

Lemma 3.3.2. *Let $m = m(\lambda), n = n(\lambda), \ell(\lambda) \in \mathsf{poly}(\lambda)$ and let $\mathcal{R}$ be a ring. Let $H : \mathcal{R}^m \times \mathbb{F}_2^n \to \mathbb{F}_2^\ell$ be a hash function, modeled as a random oracle. Then, the family of functions $\mathcal{F} = \{x \mapsto H(k, x)\}_{k \in \mathcal{R}^m}$ is an RKA-secure PRF for the family Φ of all affine relations $\Phi = \{\phi_{\alpha,\beta} \colon k \mapsto \alpha k + \beta\}_{\alpha \in \mathcal{R}^*, \beta \in \mathcal{R}^m}$. More precisely, any Φ-restricted adversary $\mathcal{A}$ against the RKA security of $\mathcal{F}$, making at most q_e evaluation queries and q_r random oracle queries, has advantage at most*

$$\mathsf{Adv}^{\mathsf{rka}}_{\mathcal{A}}(\lambda) \leq \frac{q_r \cdot q_e}{\min_{\alpha \in \mathcal{R}^*} |\alpha \cdot \mathcal{R}^m|}.$$

Proof. Let Q_e denote the set of evaluation queries (ϕ, x) of $\mathcal{A}$, and let Q_r denote the set of random oracle queries of $\mathcal{A}$. Let us denote by Bad the following event during an execution of the experiment $\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},b}(\lambda)$ (with $b = 0$ or $b = 1$): At the end of the experiment, the challenger computes $(\phi(k), x)$ for each $(\phi, x) \in Q_e$ and raises a flag Bad if $(\phi(k), x) \in Q_r$. If no flag Bad is raised, the challenger returns a flag Good.

Conditioned on the event Good, every answer $R(\phi, x)$ (where $R \colon \Phi \times \mathbb{F}_2^n \to \mathbb{F}_2^\ell$ is a random function) of the evaluation oracle to an evaluation query (ϕ, x) issued by $\mathcal{A}$ in $\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},1}(\lambda)$ is sampled as a fresh random element independent of $\mathcal{A}$'s view. In turn, it is distributed exactly as $H(\phi(k), x)$, because the latter is a fresh uniform random element when $\mathcal{A}$ never queries $(\phi(k), x)$—i.e., it is distributed exactly as in $\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda)$. In other words,

$$\Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda) = 1 \mid \mathsf{Good}] = \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},1}(\lambda) = 1 \mid \mathsf{Good}].$$

This implies

$$\begin{aligned}\mathsf{Adv}^{\mathsf{rka}}_{\mathcal{A}}(\lambda) &= \left|\Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda)=1] - \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda)=1]\right| \\ &= \Pr[\mathsf{Bad}] \cdot \left|\Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},0}(\lambda)=1 \mid \mathsf{Bad}] - \Pr[\mathsf{Exp}^{\mathsf{rka}}_{\mathcal{A},1}(\lambda)=1 \mid \mathsf{Bad}]\right| \\ &\leq \Pr[\mathsf{Bad}].\end{aligned}$$

Now, fix a query $(\phi, x) \in Q_e$, and let $\phi\colon k \mapsto \alpha \cdot k + \beta$. Define $U := \{u \in \mathcal{R}^m \mid \exists v \in \mathbb{F}_2^n, (u,v) \in Q_r\}$. The probability, over the random choice of k, that $(\phi(k), x) \in Q_r$ is at most the probability that there exists $u \in U$ such that $\alpha \cdot k = u - \beta$. Since $\alpha \cdot k$ is a uniformly random element from the ideal $\alpha \cdot \mathcal{R}$, this happens with probability at most $q_r/|\alpha\mathcal{R}| \leq q_r/\min_{\alpha\in\mathcal{R}^*}|\alpha\mathcal{R}^m|$. The lemma then follows by a union bound over all queries in Q_e. ■

3.4 Shiftable CPRFs

In this section, we start by defining the notion of *Shiftable* CPRFs in Section 3.4.1. Then, in Section 3.4.2, we construct ShCPRFs for inner-product predicates by adapting the framework from Chapter 2.

3.4.1 Defining shiftable CPRFs

For simplicity, we restrict the definition to 1-key (rather than multi-key) ShCPRFs and selective security, which is the definition that is satisfied by our construction. The definition overlaps significantly with the definition of (non-shiftable) CPRFs [BW13, KPTZ13, BGI14] but using the classic PRF-style "real-or-random" security game, where all evaluations are either computed using the master key or using a truly random function.

Remark 12 (On the choice of security game)**.** *By using the real-or-random security definition, we manage to get a tight reduction when proving security of our constructions. In contrast, prior work that uses CPRFs to construct PCFs [CMPR23, BCM+24] has a polynomial loss in security in their security proofs, proportional to the number of PCF evaluations, which was an artifact of the original CPRF definition that uses the find-then-guess formulation.*

Remark 13 (Relation to shift-hiding shiftable functions (SHSF))**.** *SHSF are an extension to CPRFs introduced by Peikert and Shiehian [PS18]. In a SHSF, the constrained CPRF evaluation computes $F.\mathsf{Eval}(\mathsf{msk}, x) + f(x)$, for a hidden "shift" function f embedded into the constrained key. In contrast, our notion of Shiftable CPRFs only allows the master key holder to shift the* constraint *when evaluating the CPRF (using the master key) and does not affect the constrained key.*

Definition 3.4.1 (Shiftable Constrained Pseudorandom Functions)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. A Shiftable Constrained Pseudorandom Function (ShCPRF) with domain $\mathcal{X} = \mathcal{X}_\lambda$, range $\mathcal{Y}$, and a finite set of shifts $\mathcal{S}$ that supports constraints represented by the class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$, where $\mathcal{C}_\lambda\colon \mathcal{X} \times \mathcal{S} \to \{0,1\}$, consists of the following four algorithms. We highlight the parts that are specific to shiftable CPRFs.*

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{msk}$. *The randomized key generation algorithm takes as input a security parameter* λ. *It outputs a master secret key* msk.
- $\mathsf{Eval}(\mathsf{msk}, x, \alpha) \to y$. *The deterministic evaluation algorithm takes as input the master secret key* msk, *an input* $x \in \mathcal{X}$, *and a shift* $\alpha \in \mathcal{S}$. *It outputs* $y \in \mathcal{Y}$.
- $\mathsf{Constrain}(\mathsf{msk}, C) \to \mathsf{csk}$. *The randomized constrain algorithm takes as input the master secret key* msk *and a constraint circuit* $C \in \mathcal{C}$. *It outputs a constrained key* csk.
- $\mathsf{CEval}(\mathsf{csk}, x) \to y$. *The deterministic constrained evaluation algorithm takes as input the constrained key* csk *and an input* $x \in \mathcal{X}$. *It outputs* $y \in \mathcal{Y}$.

We let any public parameters PP *be an implicit input to all algorithms. An ShCPRF must satisfy the following correctness, security, and pseudorandomness properties. We let* $\widetilde{\mathcal{F}} = \widetilde{\mathcal{F}}_\lambda$ *denote the set of all functions from* $\mathcal{X} \times \mathcal{S}$ *to* $\mathcal{Y}$.

Correctness. *For all security parameters* λ, *all constraints* $C \in \mathcal{C}$, *and all inputs* $x \in \mathcal{X}$, *there exists an efficiently computable* $\alpha \in \mathcal{S}$ *such that* $C(x, \alpha) = 0$ *(authorized), and for all* $\alpha \in \mathcal{S}$ *where* $C(x, \alpha) = 0$ *it holds that:*

$$\Pr\left[\mathsf{Eval}(\mathsf{msk}, x, \alpha) = \mathsf{CEval}(\mathsf{csk}, x) \quad : \quad \begin{array}{r} \mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{csk} \leftarrow \mathsf{Constrain}(\mathsf{msk}, C) \end{array}\right] = 1 - \mathsf{negl}(\lambda),$$

where the probability space is over the randomness used in KeyGen *and* $\mathsf{Constrain}$.

(1-key, selective) Security. *An ShCPRF is (1-key, selectively)-secure if for all efficient adversaries* $\mathcal{A}$, *the advantage of* $\mathcal{A}$ *in the following security experiment* $\mathsf{Exp}^{\mathsf{shcprf}}_{\mathcal{A},b}(\lambda)$ *is negligible in* λ. *Here,* b *denotes the challenge bit.*

1. ***Setup:*** *On input* 1^λ, *the challenger*
   - *runs* $\mathcal{A}(1^\lambda)$ *who outputs a constraint* $C \in \mathcal{C}$,
   - *computes* $\mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and* $\mathsf{csk} \leftarrow \mathsf{Constrain}(\mathsf{msk}, C)$,
   - *samples a uniformly random function* $R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda$, *and*
   - *sends* csk *to* $\mathcal{A}$.
2. ***Evaluation queries:*** $\mathcal{A}$ *adaptively sends arbitrary inputs* $x \in \mathcal{X}$ *and shifts* $\alpha \in \mathcal{S}$ *to the challenger. For each pair* (x, α), *if* $C(x, \alpha) = 0$, *then the challenger returns* $\perp$. *Otherwise, the challenger proceeds as follows:*
   - *If* $b = 0$, *it computes* $y := \mathsf{Eval}(\mathsf{msk}, x, \alpha)$ *and returns* y.
   - *If* $b = 1$, *it computes* $y := R(x, \alpha)$ *and returns* y.
3. ***Guess:*** $\mathcal{A}$ *outputs its guess* b', *which is the output of the experiment.*

$\mathcal{A}$ *wins if* $b' = b$, *and its advantage* $\mathsf{Adv}^{\mathsf{shcprf}}_{\mathcal{A}}(\lambda)$ *is defined as*

$$\mathsf{Adv}^{\mathsf{shcprf}}_{\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{shcprf}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{shcprf}}_{\mathcal{A},1}(\lambda) = 1]\right|,$$

where the probability is over the randomness of $\mathcal{A}$ and KeyGen.

Pseudorandomness. *An ShCPRF is said to be pseudorandom if for all efficient adversaries $\mathcal{A}$, it holds that*

$$\left| \Pr_{\mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda)} [\mathcal{A}^{\mathsf{Eval}(\mathsf{msk},\cdot,\cdot)}(1^\lambda) = 1] - \Pr_{R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda} [\mathcal{A}^{R(\cdot,\cdot)}(1^\lambda) = 1] \right| = 1 - \mathsf{negl}(\lambda).$$

Remark 14 (The requirement for the pseudorandomness property)**.** *We note that while the pseudorandomness property is implicit in standard CPRF definition (which has a polynomial loss in security) [BW13], in the real-or-random style definition for CPRFs (as in Definition 3.4.1), this property is not immediately satisfied, since the challenger outputs $\perp$ when given an unconstrained query. Hence, we require the separate* pseudorandomness *property to capture the requirement that the function being constrained is indeed a standard PRF.*

3.4.2 Constructing shiftable CPRFs

In this section, we adapt the framework from Chapter 2 constructing CPRFs for inner-product predicates from RKA-secure PRFs. We make the observation that the construction can be easily adapted to fit the shiftable CPRF definition (Definition 3.4.1). In the process, we additionally generalize the construction from Chapter 2 to work over a small ring as opposed to a large field, which makes it integrate better with an IPM-wPRF as the predicate.

The CPRF framework from Chapter 2, in a nutshell. The CPRF framework from Chapter 2 is parameterized by a security parameter λ, finite field $\mathbb{F}$ of order at least 2^λ, and a vector length parameter $n \geq 1$. The master secret key msk consists of a random vector $\mathbf{z_0} \in \mathbb{F}^n$. The constrained key csk for a constraint $\mathbf{z} \in \mathbb{F}^n$ is then defined as $\mathbf{z_1} := \mathbf{z_0} - \Delta\mathbf{z}$, with $\Delta \in \mathbb{F} \setminus \{0\}$ a random non-zero scalar. The main insight behind the framework of Chapter 2 is that for an input $\mathbf{x} \in \mathbb{F}^n$, when $\langle \mathbf{z}, \mathbf{x} \rangle = 0$ (i.e., when the constraint is satisfied), then the inner product $\langle \mathbf{z_0}, \mathbf{x} \rangle$ is equal to $\langle \mathbf{z_1}, \mathbf{x} \rangle$. This fact can be used to derive *identical* PRF keys k and k' under both msk and csk:

$$k = \langle \mathbf{z_0}, \mathbf{x} \rangle = \langle \mathbf{z_1}, \mathbf{x} \rangle - \cancel{\Delta \langle \mathbf{z}, \mathbf{x} \rangle} = \langle \mathbf{z_1}, \mathbf{x} \rangle = k'.$$

In contrast, when $\langle \mathbf{z}, \mathbf{x} \rangle \neq 0$, the $\Delta \langle \mathbf{z}, \mathbf{x} \rangle$-term makes $k \neq k'$. Moreover, because Δ is uniformly random over $\mathbb{F} \setminus \{0\}$ (where $\mathbb{F}$ has order at least 2^λ), $\mathbf{z_1}$ cannot be used to recover $\mathbf{z_0}$, even with knowledge of the constraint $\mathbf{z}$. The evaluation of the CPRF is then defined as $F_{k_0+k}(\mathbf{x})$ (resp. $F_{k_0+k'}(\mathbf{x})$ for the constrained evaluation), where k_0 is a "zero" PRF key used to handle the case where $\mathbf{x} = 0^n$. One caveat, however, is that the derived PRF keys are *highly correlated*, which necessitates choosing F to be a suitable RKA-secure PRF. In Chapter 2, we show that when the PRF F is RKA-secure for affine key-derivation functions (Definition 2.3.6), then the CPRF instantiated with the PRF F is secure. (We note that a random oracle $H \colon \mathbb{F} \times \mathbb{F}^n \to \{0,1\}^*$ is RKA-secure PRF for all non-trivial key-derivation functions, as we prove in Section 3.3.3.1.)

Adding shiftability. We make the simple observation that if we make the master secret key msk also contain Δ, then the we can easily turn the above framework into a *shiftable* CPRF

ShCPRF for Inner-Product Predicates

Public Parameters. Security parameter λ, finite ring $\mathcal{R}$ of order ℓ, integers m such that $m \geq \lambda$, vector length $n \geq 1$, and an RKA-secure PRF family $F\colon \mathcal{R}^m \times \mathcal{R}^n \to \mathcal{Y}$ for affine key-derivation functions.

$\mathsf{ShCPRF.KeyGen}(1^\lambda)$:
1 : $\mathbf{k_0} \xleftarrow{R} \mathcal{R}^m,\ \Delta \xleftarrow{R} \mathcal{R}^m \setminus \{0\}$
2 : $\mathbf{Z_0} \xleftarrow{R} \mathcal{R}^{m \times n}$
3 : $\mathsf{msk} := (\mathbf{k_0}, \mathbf{Z_0}, \Delta)$

$\mathsf{ShCPRF.Constrain}(\mathsf{msk}, \mathbf{z})$:
1 : **parse** $\mathsf{msk} = (\mathbf{k_0}, \mathbf{Z_0}, \Delta)$
2 : $\mathbf{Z_1} := \mathbf{Z_0} - \Delta \mathbf{z}^\top$
3 : **return** $\mathsf{csk} := (\mathbf{k_0}, \mathbf{Z_1})$

$\mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}, \alpha)$:
1 : **parse** $\mathsf{msk} = (\mathbf{k_0}, \mathbf{Z_0}, \Delta)$
2 : $\mathbf{k} := \mathbf{k_0} + \mathbf{Z_0 x} - \Delta \cdot \alpha$
3 : **return** $F_{\mathbf{k}}(\mathbf{x})$

$\mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x})$:
1 : **parse** $\mathsf{csk} = (\mathbf{k_0}, \mathbf{Z_1})$
2 : $\mathbf{k} := \mathbf{k_0} + \mathbf{Z_1 x}$
3 : **return** $F_{\mathbf{k}}(\mathbf{x})$

Figure 3.2: ShCPRF framework for inner-product predicates based on RKA-secure PRFs.

as follows. Specifically, when $\langle \mathbf{z}, \mathbf{x} \rangle \neq 0$, the constrained key computes $k' = \langle \mathbf{z_0}, \mathbf{x} \rangle - \Delta \langle \mathbf{z}, \mathbf{x} \rangle$. The master key holder, with knowledge of Δ, can compute $k = \langle \mathbf{z_0}, \mathbf{x} \rangle - \Delta \cdot \alpha = k'$, where $\alpha = \langle \mathbf{z}, \mathbf{x} \rangle$. This is enough to satisfy the correctness property of Definition 3.4.1 (Shiftable CPRFs). In particular, here the constraint predicate $C(\mathbf{x}, \alpha)$ is 0 if $\langle \mathbf{z}, \mathbf{x} \rangle - \alpha = 0$ and 1 otherwise.

Moving to the ring setting. We require instantiating the ShCPRF with a *small* ring $\mathcal{R}$ (e.g., $\mathcal{R} = \mathbb{Z}_6$) for efficiency purposes. However, to ensure each derived key is still at least λ-bits, we must extend the small ring to a sufficiently large ring $\mathcal{R}'$. As such, we replace the large field $\mathbb{F}$ with a large $\mathcal{R}' = \mathcal{R}^m$, where $m \geq \lambda$ to guarantee λ bits of security (we prove the security of this modification in Lemma 3.3.2). Doing so, however, makes the vectors $\mathbf{z_0}$ and $\mathbf{z_1}$ (now sampled in $(\mathcal{R}')^n$) better denoted as *matrices* from $\mathcal{R}^{m \times n}$. While this induces notational changes, the CPRF construction itself remains almost identical to the one in Chapter 2. In particular, for a constraint $\mathbf{z} \in \mathcal{R}^n$, we now have CPRF keys $\mathbf{k}, \mathbf{k}' \in \mathcal{R}^m$ (as opposed to $k, k' \in \mathbb{F}$ above) derived for an input $\mathbf{x} \in \mathcal{R}^n$ as $\mathbf{k} = \mathbf{Z_0 x} = \mathbf{Z_1 x} + (\Delta \mathbf{z}^\top)\mathbf{x}$ which is equal to $\mathbf{k}' = \mathbf{Z_1 x}$, when the constraint $\langle \mathbf{z}, \mathbf{x} \rangle = 0 \in \mathcal{R}$. We present our ring-based ShCPRF framework in Figure 3.2 and prove security in Section 3.4.3 and Section 3.11.1.

To state more exactly the special type of Shiftable CPRF we obtain, we have the following definition.

Definition 3.4.2 (Shiftable CPRFs for Inner-Product Predicates)**.** *Let $\mathcal{R} = \mathcal{R}_\lambda$ be a finite ring. Let* ShCPRF *be a Shiftable CPRF with domain $\mathcal{X} = \mathcal{R}^n$ for an $n = n(\lambda)$, range $\mathcal{R}$, and finite set of shifts $\mathcal{S} = \mathcal{R}$ that supports constraints represented by a class of circuits $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$,*

such that $\mathcal{C}_\lambda = \{C_{\mathbf{z}} \colon \mathbf{z} \in \mathcal{R}^n\}$*, where the* $C_{\mathbf{z}} \colon \mathcal{X} \times \mathcal{S} \to \{0,1\}$ *are given via*

$$(\mathbf{x}, \alpha) \mapsto \begin{cases} 0, & \text{if } \langle \mathbf{z}, \mathbf{x} \rangle - \alpha = 0, \\ 1, & \text{otherwise.} \end{cases}$$

Then, we identify the constraint circuit $C_{\mathbf{z}}$ *with* $\mathbf{z}$*, i.e., we just write that the constraint as a vector* $\mathbf{z}$*, and call* ShCPRF *a (ring-based)* Shiftable CPRF for inner-product predicates.

3.4.3 Security analysis

Here, we analyze the security of the ShCPRF framework using the proof template of Theorem 2.5.1 from Chapter 2. We adapt it in several key locations to handle the shiftability property and operations in the ring $\mathcal{R}$.

Theorem 3.4.1. *If* $\mathcal{F}$ *is a family of RKA-secure pseudorandom functions with respect to affine related-key derivation functions* Φ_{aff}*, as defined in Definition 2.3.6, then Figure 3.2 instantiated with* $\mathcal{F}$ *is a (1-key, selectively-secure) ShCPRF for inner-product constraint predicates.*

Proof. Deferred to Section 3.11.1. ■

3.5 PCFs for ListOT: Framework

In this section, we formalize our PCF for ListOT framework using the Shiftable CPRF framework from Section 3.4. As mentioned in Section 3.2, ListOT does *not* fulfill the definition of a "correlation" as defined by Boyle et al. [BCG+19b]. Therefore, we cannot use existing definitions of a pseudorandom *correlation* function (PCF), since the correlation is only partially defined. In particular, the problem is that the output of the receiver in ListOT has an additional lookup key i that depends directly on the wPRF key used to compute the pseudorandom bit b, which cannot be efficiently sampled given just the output of the sender. We sidestep these issues by adapting the standard definition of a PCF [BCG+20a] to work with the "partial correlation" that is ListOT. In Section 3.5.1, we define the notion of a PCF for ListOT. Then, in Section 3.5.2, we describe our general framework for constructing a PCF for ListOT. Finally, in Section 3.5.3, we explain how a PCF for ListOT is used to instantiate QuietOT.

3.5.1 Defining PCFs for ListOT

Here, we give a formal definition of (weak) PCF for ListOT. For convenience, we use of the following *distribution of lists* notation.

Definition 3.5.1 (Distribution of Lists)**.** *Let* λ *be a security parameter,* $\mathcal{Y}$ *be a finite set, and* I *be an arbitrary finite index set. We denote by* $\mathcal{D}_{\mathcal{Y}}^{\text{list}}(I)$ *the distribution that outputs a list* $(v_i)_{i \in I}$*, where each* $v_i \xleftarrow{R} \mathcal{Y}$ *is independently sampled at random.*

Definition 3.5.2 (Pseudorandom Correlation Function for ListOT)**.** *Let λ be a security parameter and $\lambda \leq n = n(\lambda) \in \mathsf{poly}(\lambda)$ be an input length. A Pseudorandom Correlation Function (PCF) for ListOT with domain $\mathcal{X} = \mathcal{X}_\lambda$ is defined by a pair of algorithms $\mathsf{PCF} = (\mathsf{KeyGen}, \mathsf{Eval})$ with the following functionality:*

- $\mathsf{KeyGen}(1^\lambda) \to (K_S, K_R)$*. The randomized key generation algorithm takes as input the security parameter λ. It outputs a pair of keys (K_S, K_R).*
- $\mathsf{Eval}(\sigma, K_\sigma, x) \to y_\sigma$*. The deterministic evaluation algorithm takes as input $\sigma \in \{S, R\}$, a key K_σ, and input $x \in \mathcal{X}$. It outputs a string $y_\sigma \in \{0,1\}^*$, where*
  - *if $\sigma = S$ then $y_\sigma = (L_0, L_1)$ for two lists L_0, L_1; otherwise*
  - *if $\sigma = R$ then $y_\sigma = (b, v, \alpha)$ for a bit $b \in \{0,1\}$, a list entry $v \in L_0 \cup L_1$, and a lookup key α.*

We will use $\mathsf{PCF.EvalS}(K_S, x)$ and $\mathsf{PCF.EvalR}(K_R, x)$ as shorthand for the Eval algorithm used by the sender and receiver, respectively. We leave any public parameters PP as an implicit input to all algorithms.

A $\mathsf{PCF} = (\mathsf{KeyGen}, \mathsf{Eval})$ is a (weak) PCF for ListOT *with domain $\mathcal{X} = \mathcal{X}_\lambda$, if the following* correctness, sender security, *and* receiver security *properties hold. In each case, the adversary is given access to $N(\lambda) \in \mathsf{poly}(\lambda)$ samples.*

Pseudorandomness. *For all efficient adversaries $\mathcal{A}$, and all $N \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{pr}}_{\mathcal{A},N}(\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 3.3.

Correctness. *Moreover, we want that for any $\lambda \in \mathbb{N}$ it holds that:*

$$\Pr\left[\; v \neq L_b[\alpha] \quad : \quad \begin{array}{r} (K_S, K_R) \leftarrow \mathsf{PCF.KeyGen}(1^\lambda) \\ x \xleftarrow{R} \mathcal{X}_\lambda \\ (L_0, L_1) := \mathsf{PCF.EvalS}(K_S, x) \\ (b, v, \alpha) := \mathsf{PCF.EvalR}(K_R, x) \end{array} \;\right] \leq \mathsf{negl}(\lambda),$$

i.e., that the (relevant) entry v is at position α of list L_b with a probability that is overwhelming in λ.

Sender Security. *For all efficient adversaries $\mathcal{A}$, there exists a negligible function negl such that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{Ssec}}_{\mathcal{A},N}(\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 3.4.

Receiver Security. *For all efficient adversaries $\mathcal{A}$, there exists a negligible function negl*

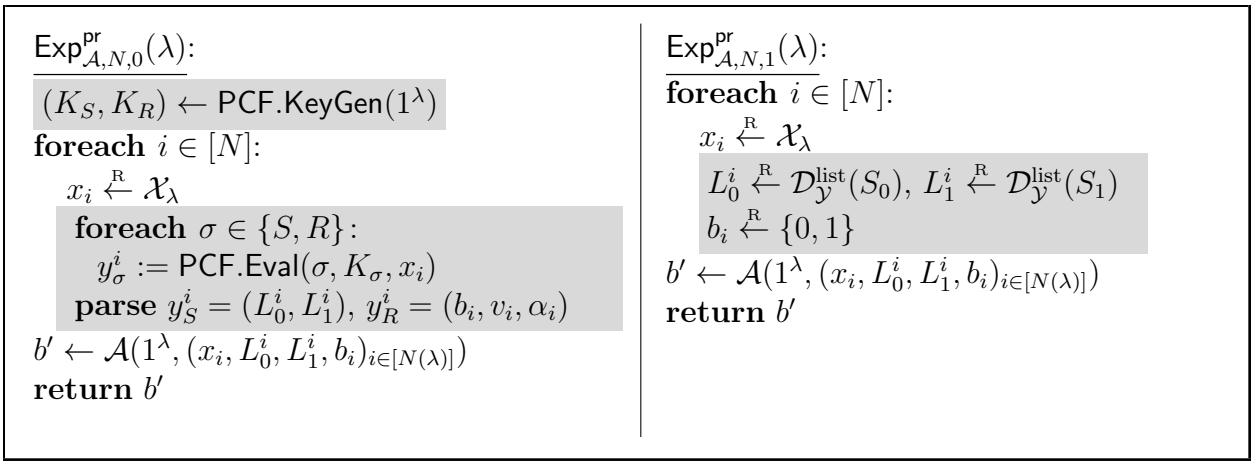


Figure 3.3: (Partially) Pseudorandom outputs of a PCF for ListOT. The distribution $\mathcal{D}^{\text{list}}_{\mathcal{Y}}(\cdot)$ is defined in Definition 3.5.1.

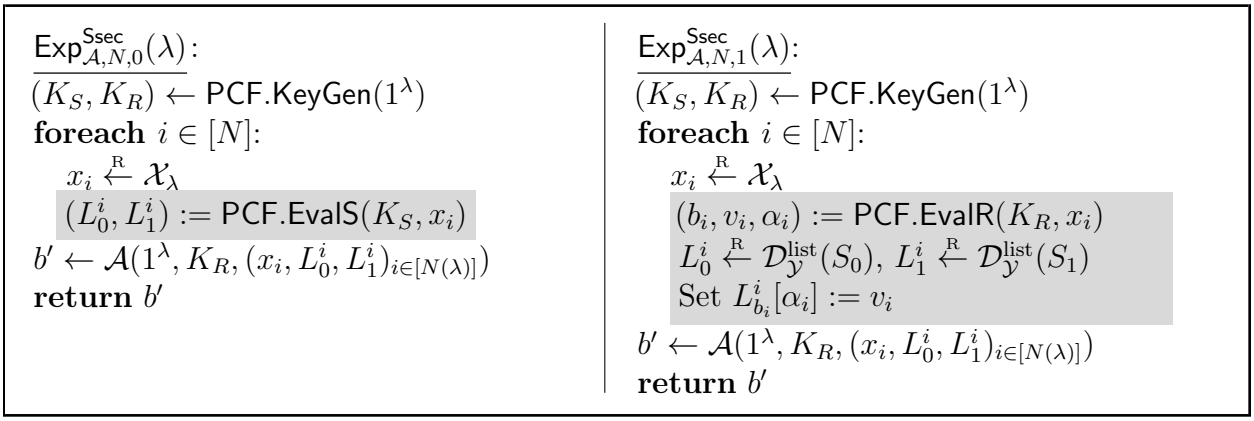


Figure 3.4: Sender security game of a PCF for ListOT. The distribution $\mathcal{D}^{\text{list}}_{\mathcal{Y}}(\cdot)$ is defined in Definition 3.5.1.

such that for all sufficiently large λ*,*

$$\mathsf{Adv}^{\mathsf{Rsec}}_{\mathcal{A},N}(\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{Rsec}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{Rsec}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{Rsec}}_{\mathcal{A},N,b}(\lambda)$*, for* $b \in \{0,1\}$*, is as defined in Figure 3.5.*

3.5.2 Framework: PCF for ListOT from IPM-wPRFs

In Figure 3.6, we describe the framework for constructing a PCF for ListOT by combining a Shiftable CPRF with any IPM-wPRF.

Theorem 3.5.1. *Let* $n = n(\lambda) \in \mathsf{poly}(\lambda)$ *and* $\mathcal{R}$ *be a finite ring of order* q*, with extension parameter* $m \geq \lambda$*. Let* $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ *be a shiftable CPRF for inner-product predicates with domain* $\mathcal{R}^n$*, and* $f\colon \mathcal{R}^n \times \mathcal{R}^n \to \{0,1\}$ *a weak PRF family for*

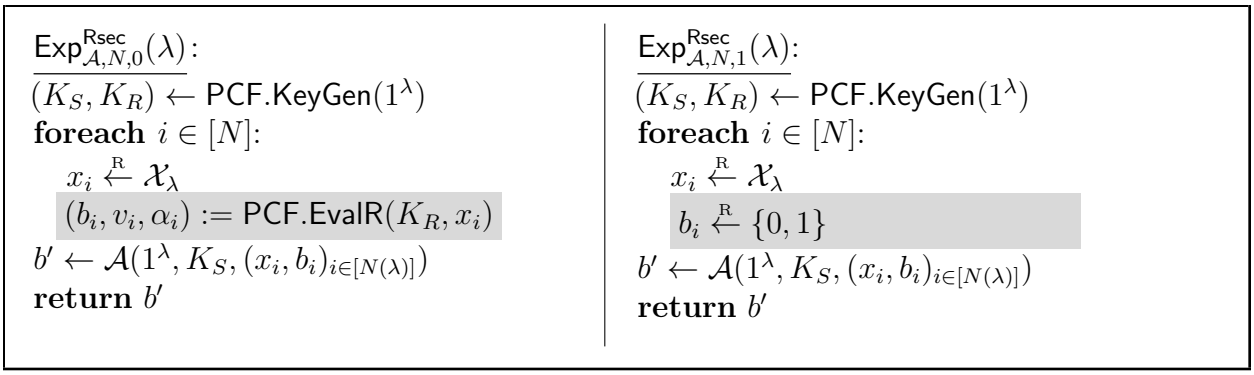


Figure 3.5: Receiver security game of a PCF for ListOT.

inner-product membership with partitioning $S_0 \cup S_1 = \mathcal{R}$. *Then,* $\mathsf{PCF} = (\mathsf{KeyGen}, \mathsf{Eval})$ *from Figure 3.6 is a PCF for ListOT.*

Proof. Deferred to Section 3.11.2. ■

3.5.3 Realizing QuietOT from a PCF for ListOT

To generate random OT correlations, the sender and receiver use the PCF for ListOT to generate pseudorandom ListOT instances. We use the template of Beaver [Bea95] for converting a random ListOT instance into a chosen-bit OT protocol. We describe this transformation in Figure 3.7.

Proposition 3.5.1. *Let* $\mathsf{PCF} = (\mathsf{KeyGen}, \mathsf{Eval})$ *be a PCF for ListOT. Then, the protocol given in Figure 3.7 securely realizes the OT functionality.*

Proof. The OT functionality is defined in Section 3.3.3. By the pseudorandomness property of the PCF for ListOT we have that L_0 and L_1 are pseudorandom lists and b' is a pseudorandom bit if x (input to the PCF) is uniformly random. For an arbitrary choice bit $b \in \{0,1\}$ we consider the two possible cases to prove correctness.

- Case 1: $b = 0$. In this case, $c = b'$ and so $L'_0 = L_{b'} \oplus m_0$ and $L'_1 = L_{1-b'} \oplus m_1$. It then follows that the receiver outputs $(L'_0[\alpha] \oplus m_0) \oplus v$, which equals m_0 by the correctness property of the PCF.
- Case 2: $b = 1$. In this case, $c = 1 - b'$ and so $L'_0 = L_{1-b'} \oplus m_0$ and $L'_1 = L_{b'} \oplus m_1$. It then follows that the receiver outputs $m_1 = (L'_1[\alpha] \oplus m_1) \oplus v$, since we have $v = L_{b'}[\alpha] = L'_1[\alpha]$ by the correctness property of the PCF.

Note that in both cases, the equality holds with overwhelming probability, because correctness of the PCF holds with overwhelming probability (Definition 3.5.2).

Sender security follows directly from the sender security of the PCF which guarantees that (1) the receiver only obtains $L_{b'}[\alpha]$ and (2) all other values in both lists are pseudorandom from the viewpoint of the receiver and therefore guarantees that the receiver only obtains m_b.

Receiver security follows from the fact that b' is a pseudorandom bit (by receiver security of the PCF) and therefore a pseudorandom mask for the receiver's choice bit b. Therefore, the sender learns nothing. ■

PCF for ListOT

Public Parameters.

- Key length $n = n(\lambda) \in \mathsf{poly}(\lambda)$.
- Finite ring $\mathcal{R}$ of order q, with extension parameter $m \geq \lambda$.
- IPM-wPRF family $f : \mathcal{R}^n \times \mathcal{R}^n \to \{0,1\}$ with partitioning $S_0 \cup S_1 = \mathcal{R}$.
- An injective input mapping $\mathsf{map}\colon \mathcal{X}_\lambda \to \mathcal{R}^n$.
- An ShCPRF for inner-product predicates $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ with domain $\mathcal{R}^n$

$\mathsf{PCF.KeyGen}(1^\lambda)$**.**

1: $\mathsf{msk} \leftarrow \mathsf{ShCPRF.KeyGen}(1^\lambda)$
2: $\mathbf{z} \xleftarrow{R} \mathcal{R}^n$. ▷ IPM-wPRF key
▷ Note that $\mathbf{z}$ can also be sampled from a non-uniform distribution over $\mathcal{R}^n$.
3: $\mathsf{csk} \leftarrow \mathsf{ShCPRF.Constrain}(\mathsf{msk}, \mathbf{z})$
4: $K_S := \mathsf{msk}$, $K_R := (\mathsf{csk}, \mathbf{z})$.
5: **return** (K_S, K_R)

$\mathsf{PCF.EvalS}(K_S, x)$**.**

1: **parse** $K_S = \mathsf{msk}$.
2: $\mathbf{x} := \mathsf{map}(x)$.
3: **foreach** $b \in \{0,1\}$ **and** $\alpha \in S_b$:
  1: $y := \mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}, \alpha)$.
  2: $L_b[\alpha] = y$.
4: **return** (L_0, L_1).

$\mathsf{PCF.EvalR}(K_R, x)$**.**

1: **parse** $K_R = (\mathsf{csk}, \mathbf{z})$.
2: $\mathbf{x} := \mathsf{map}(x)$.
3: $v := \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x})$
4: $b := f_{\mathbf{z}}(\mathbf{x})$, $\alpha := \langle \mathbf{z}, \mathbf{x} \rangle \in \mathcal{R}$.
▷ Note that $v = L_b[\alpha]$ in the sender-computed list.
5: **return** (b, v, α).

Figure 3.6: Framework for a PCF for ListOT from any ShCPRF and IPM-wPRF.

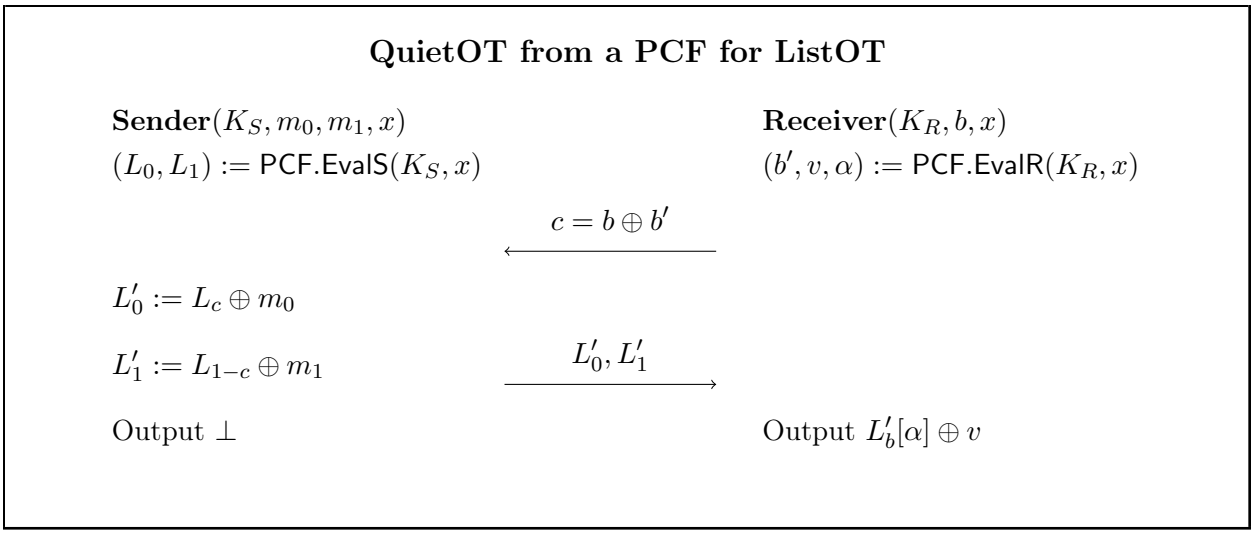


Figure 3.7: QuietOT using a (weak) PCF for ListOT. Note that x (input to the PCF known by both the sender and receiver) is uniformly random, as specified in Definition 3.5.2.

3.6 PCFs for ListOT: Instantiations

In this section, we instantiate the framework from Section 3.5 using either the BIPSW or GAR IPM-wPRF candidate, coupled with the "RKA-PRF" $F_k(x) := H(k, x)$ for a hash function H modeled as a random oracle. For the sake of completeness, we prove in Section 3.3.3.1 that F_k is indeed an RKA-PRF for all affine relations $x \mapsto \alpha x + \beta$ with $\alpha \in \mathcal{R}^*$ and $\beta \in \mathcal{R}^m$, as long as $\min_\alpha(|\alpha \cdot \mathcal{R}^m|) \geq 2^\lambda$. Looking ahead, both our instantiations will satisfy $\min_\alpha(|\alpha \cdot \mathcal{R}^m|) = 2^m$, and we will therefore set $m = \lambda$.

These two instantiations result in our concretely efficient constructions (see Section 3.8). We also describe other instantiations from different assumptions (in particular, replacing the random oracle using an RKA-secure PRF), which have interesting theoretical implications but do not currently result in concretely efficient constructions.

3.6.1 BIPSW IPM-wPRF instantiation

Our main instantiation is based on the BIPSW wPRF candidate, which can be easily viewed as an IPM-wPRF. For a key $\mathbf{z} \in \mathbb{Z}_6^n$ with $n = n(\lambda) \in \mathsf{poly}(\lambda)$, and $\mathbf{x} \in \mathbb{Z}_6^n$, the BIPSW wPRF is defined as: $f_{\mathbf{z}}(\mathbf{x}) = \lfloor \langle \mathbf{z}, \mathbf{x} \rangle \mod 6 \rceil_2$, where $\lfloor \alpha \rceil_2 = 0$ for all $\alpha \in \{0, 1, 2\}$ and $\lfloor \alpha \rceil_2 = 1$ for all $\alpha \in \{3, 4, 5\}$. For a partitioning of $\mathbb{Z}_6$ consisting of $S_0 := \{0, 1, 2\}$ and $S_1 := \{3, 4, 5\}$, we get that $\langle \mathbf{z}, \mathbf{x} \rangle \mod 6 \in S_b \iff f_{\mathbf{z}}(\mathbf{x}) = b$.

We instantiate the framework using the ring $\mathcal{R} = \mathbb{Z}_6$ and set $m \geq \lambda$ (see Lemma 3.3.2). We interpret $\{0, 1\}$ as elements of $\mathbb{Z}_6$ in the natural way (mapping 0 to the additive identity and 1 to the multiplicative identity of $\mathbb{Z}_6$) and define map to be the canonical embedding from $\{0, 1\}^n$ to $\mathbb{Z}_6^n$. The full construction is presented in Figure 3.8 and closely follows the general framework from Figure 3.6. We use the specific ShCPRF construction of Figure 2.1 with the random oracle H as the RKA-secure PRF and explicitly work over the ring $\mathbb{Z}_6$.

PCF for ListOT from the BIPSW IPM-wPRF

Public Parameters.

- Integers $n = n(\lambda) \in \mathsf{poly}(\lambda)$, $m \geq \lambda$ and domain $\mathcal{X}_\lambda = \{0,1\}^n$.
- BIPSW IPM-wPRF family $f : \mathbb{Z}_6^n \times \mathbb{Z}_6^n \to \{0,1\}$ with partitioning:

$$S_0 := \{\alpha \in \mathbb{Z}_6 \mid \alpha < 3\} \text{ and } S_1 := \mathbb{Z}_6 \setminus S_0.$$

- Input mapping $\mathsf{map}\colon \mathcal{X}_\lambda \to \mathcal{R}^n$ is the canonical embedding of $\{0,1\}^n$ in $\mathbb{Z}_6^n$.
- The ShCPRF $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ from Figure 3.2, where the RKA-secure PRF family $F\colon \mathcal{R}^m \times \mathcal{R}^n \to \mathcal{Y}$ is instatiated by a random oracle H, cf. Section 3.3.3.1.

Construction.
$(\mathsf{PCF.KeyGen}, \mathsf{PCF.EvalR}, \mathsf{PCF.EvalS})$ are identical to Figure 3.6 using $\mathcal{R} = \mathbb{Z}_6$.

Figure 3.8: PCF for ListOT from the BIPSW IPM-wPRF.

3.6.2 GAR IPM-wPRF instantiation

Unlike for the BIPSW wPRF, converting the GAR wPRF into an IPM-wPRF is more challenging. For completeness, we describe how Bui et al. [BCM+24] express the evaluation function as an IPM and discuss concrete parameters that we use for our instantiation, which differ from the parameters used to instantiate BCMPR.

The GAR construction. In a nutshell, the GAR construction (when instantiated with the XOR-MAJ predicate [AL16, CDM+18]) has a key $K \in \{0,1\}^n$ and takes as input a string x that is parsed as a tuple of disjoint *sets* $(X_{\mathsf{xor}}, X_{\mathsf{maj}}) \subset [n]^2$ such that $|X_{\mathsf{xor}}| = k$ and $|X_{\mathsf{maj}}| = \ell$, for integers $k = k(\lambda), \ell = \ell(\lambda)$. The evaluation of f_K is then computed as: $(\bigoplus_{i \in X_{\mathsf{xor}}} K[i]) \oplus \mathsf{MAJ}((K[j])_{j \in X_{\mathsf{maj}}})$, where MAJ outputs the majority bit.

The GAR construction as an IPM-wPRF. Converting this evaluation into an inner-product membership can be done as follows. View the evaluation as two separate components: an XOR component and a MAJ component. For each index $i \in X_{\mathsf{xor}}$, let $\mathbf{e}_i$ be the one-hot vector of length n with 1 in its i-th coordinate. First, interpret K as $\mathbf{z}_{\mathsf{xor}} \in \mathbb{Z}_2^n$ and as $\mathbf{z}_{\mathsf{maj}} \in \mathbb{Z}_\ell^n$ by mapping 0 to 0 $\in \mathbb{Z}_2$ (resp. $\mathbb{Z}_\ell$) and 1 to 1 $\in \mathbb{Z}_2$ (resp. $\mathbb{Z}_\ell$)). Then, compute $v_{\mathsf{xor}} = \sum_{i \in X_{\mathsf{xor}}} \langle \mathbf{z}_{\mathsf{xor}}, \mathbf{e}_i \rangle$. Similarly, for each index $j \in X_{\mathsf{maj}}$, let $\mathbf{e}_j$ be the corresponding one-hot vector. Then, compute $v_{\mathsf{maj}} = \sum_{j \in X_{\mathsf{maj}}} \langle \mathbf{z}_{\mathsf{maj}}, \mathbf{e}_j \rangle$. Observe that $v_{\mathsf{maj}} \geq \lceil \frac{\ell}{2} \rceil \iff \mathsf{MAJ}((\mathbf{z}_{\mathsf{maj}}[j])_{j \in X_{\mathsf{maj}}}) = 1$. We define $\mathcal{R} = \mathbb{Z}_2 \times \mathbb{Z}_\ell$, which intuitively allows for computing the "XOR" and "MAJ" components in separate subrings. Therefore, we can view f as an IPM-wPRF with partition:

$$S_0 = \left\{(u,v) \in \mathcal{R} \mid (u = 0 \wedge v > \lfloor \tfrac{\ell}{2} \rfloor) \vee (u = 1 \wedge v \leq \lfloor \tfrac{\ell}{2} \rfloor)\right\} \text{ and } S_1 = \mathcal{R} \setminus S_0.$$

Parameters. We follow the parameter selection process of Bui et al. [BCM+24], which

builds upon the state-of-the-art cryptanalysis of Goldreich's PRG from [AL16, CDM+18, YGJL21, Üna23]. To achieve λ bits of security with a key of length $n = \lambda^\delta$ and a bound n^{1+e} on the number of queries, the analysis of Bui et al. [BCM+24] suggests to use the XOR-MAJ predicate with $\ell_1 = 2 \cdot e + 1$ terms in the XOR, and $\ell_2 = (2\delta/(\delta - 1)) \cdot e + 1$ terms in the MAJ. Concretely, we set $e = 2$ to get a stretch n^3 (looking ahead, we will choose $n = 2^{11}$, hence this corresponds to generating up to 2^{33} OTs) and $\delta = 7/5$ (hence $\delta/(\delta - 1) = 7/2$). This implies that we can set $\ell_1 = 5$ and $\ell_2 = 15$. With these parameters, we must set $\ell = \ell_2 + 1 = 16$ to ensure no wraparound when computing the MAJ predicate on the sum modulo ℓ of the ℓ_2 entries in the corresponding subset. While this analysis indicates that a seed size of $n \geq 128^\delta = 892$ suffices, we set $n = 2048$ which allows us to more efficiently parse uniformly random inputs x into the index sets X_{xor} and X_{maj}, and generate a larger number $\lambda^\delta = 2^{33}$ of oblivious transfers. This results in an extremely conservative parameter set: The estimated bit security of this parameter set, using the state-of-the-art cryptanalysis [AL16, CDM+18, YGJL21, Üna23], is 2^{232}.

Remark 15 (On the recent attack of Fu et al. [FLLL24])**.** *Very recently, the work of Fu et al. [FLLL24] introduced a new attack on Goldreich's PRG that, in particular, breaks the parameters chosen in BCMPR using about 2^{25} calls to a Gaussian elimination routine (a reasonable estimate of the runtime is of the order of 2^{45} operations). This attack exploits the fact that, given enough equations, one can guess that a subset of d seed bits will be zero, and filter for equations where the* MAJ *component includes all these d bits. Then, because a* MAJ *predicate applied to ℓ_2 values, of which d are zero, is highly likely to yield zero, the corresponding XOR-MAJ equation can be viewed as a noisy linear equation. This noisy linear equations can then be solved using information set decoding algorithms.*

While the attack completely breaks the BCMPR parameters, it performs very poorly in our setting due to our choice of a very small ℓ_2 and a very large n. Concretely, to obtain at least n "filtered" equations, their attack requires that $m \cdot \binom{n-d}{\ell_2-d}/\binom{n}{\ell_2} > n$, which in our case implies $d \leq 3$. Using $\ell_2 = 15$, fixing $d = 3$ bits of MAJ to zero yields a "noise rate" of 19%, and the runtime of ISD with this noise rate is extremely large (e.g., on the order of 2^{660} using Prange's algorithm [Pra62], when $n = 2048$).

3.6.3 Other instantiations

While we focus on the BIPSW and GAR IPM-wPRF constructions when instantiating our framework, several other instantiations are possible. First, we can instantiate the framework using a different IPM-wPRF candidate. While BIPSW and GAR appear to be the most efficient candidates to fit the IPM-wPRF template, future wPRF candidates or improved parameters for the LWR wPRF resulting from tighter reductions for the LWR problem, could lead to new instantiations. For example, with the VDLPN wPRF candidate [BCG+20a] (which can be cast as an IPM-wPRF [BCM+24]) and whose concrete security is beginning to be analyzed [CD23], we could potentially have an additional practical instantiation.

Additionally, our framework is not restricted to the random oracle model (albeit, assuming a random oracle can lead to the most practical instantiations, as is the case for other OT extension protocols). As with prior OT extension protocols, we can replace the random oracle with a suitable correlation-robust hash function [IKNP03]. However, we can even go one step

PCF for ListOT from the GAR IPM-wPRF

Public Parameters.

- Integers $n = n(\lambda) \in \mathsf{poly}(\lambda)$ and $m \geq \lambda$ and $\mathcal{R} = \mathbb{Z}_2 \times \mathbb{Z}_\ell$.
- GAR IPM-wPRF family $f : \{0,1\}^n \times \{0,1\}^w \to \{0,1\}$ with parameters $w_0, w_1, w = w(w_0, w_1)$, where $w_0, w_1, w \in \mathbb{N}$, and partitioning:

$$S_0 = \left\{(u,v) \in \mathcal{R} \mid (u = 0 \wedge v > \left\lfloor \tfrac{\ell}{2} \right\rfloor) \vee (u = 1 \wedge v \leq \left\lfloor \tfrac{\ell}{2} \right\rfloor)\right\} \text{ and } S_1 = \mathcal{R} \setminus S_0.$$

- Input mapping $\mathsf{map}\colon \{0,1\}^{\mathsf{poly}(n)} \to \mathcal{R}^n$ where each uniformly random input $x \in \{0,1\}^{\mathsf{poly}(n)}$ is interpreted as a tuple $(x_{\mathsf{xor}}, x_{\mathsf{maj}}) \in \{0,1\}^n \times \{0,1\}^n$ subject to $\mathcal{HW}(x_{\mathsf{xor}}) = w_0$ and $\mathcal{HW}(x_{\mathsf{maj}}) = w_1$, where $\mathcal{HW}(\cdot)$ denotes the Hamming weight of the input. $(x_{\mathsf{xor}}, x_{\mathsf{maj}})$ is interpreted as $(\mathbf{x}_{\mathsf{xor}}, \mathbf{x}_{\mathsf{maj}}) \in \mathcal{R}^n$ in the natural way (by interpreting 0 and 1 as elements of $\mathcal{R}$).
- The ShCPRF $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ from Figure 3.2, where the RKA-secure PRF family $F\colon \mathcal{R}^m \times \mathcal{R}^n \to \mathcal{Y}$ is instatiated by a random oracle H, cf. Section 3.3.3.1.

Construction.

$(\mathsf{PCF.KeyGen}, \mathsf{PCF.EvalR}, \mathsf{PCF.EvalS})$ are as described in Figure 3.6 using the ring $\mathcal{R} = \mathbb{Z}_2 \times \mathbb{Z}_\ell$ and input mapping map defined above.

Figure 3.9: PCF for ListOT from the GAR IPM-wPRF.

further. Note that because the security relies on the security of the ShCPRF, and the CPRF construction from Chapter 2 relies only on a suitable RKA-secure PRF (a property inherited by our construction of shiftable CPRFs), we can instantiate it using other assumptions such as DDH, DCR, or VDLPN. We discuss these (currently purely theoretical) instantiations in Section 3.10.3.

3.7 Public-Key Setup

We define the notion of a public-key setup for our PCF for ListOT. These definitions provide the necessary foundations to apply public-key OT to an MPC setting (e.g., where parties may engage in pairwise OT extensions). While the application of public-key OT to MPC was mentioned in several prior works [OSY21, BCM+24], they did not provide a formal treatment of this application. We find that the standard definition of public-key OT only guarantees "one-instance" security, and does not address the many subtleties that arise when the existing constructions are applied to a multi-party setting where an adversary can corrupt multiple parties. Unfortunately, this makes existing public-key OT constructions potentially insecure if used in such contexts. Here, we lay down the foundations necessary for providing "ℓ-instance"-secure public-key OT, and prove that our main framework satisfies this stronger definition.

3.7.1 ℓ-instance updatability of shiftable CPRFs

Here, we upgrade the ShCPRF definition from Section 3.5 to provide a notion of updatability, allowing two parties to generate an ShCPRF key from "partial" keys.

Definition 3.7.1. *Let* $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ *be a shiftable CPRF with domain* $\mathcal{X} = \mathcal{X}_\lambda$ *and range* $\mathcal{Y}$ *that supports constraints represented by the class of circuits* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *(cf. Definition 3.4.1). We say the ShCPRF is* updatable*, if the key generation algorithm* KeyGen *outputs a master secret key* msk *of the form* $(\mathsf{ltsk}, \mathsf{esk})$*, where* ltsk *is a* long-term secret key*,* esk *is an* ephemeral secret key*, and there exists an efficient* Update *algorithm with the following syntax:*

- $\mathsf{Update}(\mathsf{ltsk}, \mathsf{csk}^*, C) \to \mathsf{esk}'$*. The deterministic update algorithm takes as input a long-term master secret key* ltsk*, a constrained key* csk^**, and a constraint* C*. It outputs an updated ephemeral master secret key* esk'*.*

Moreover, we require that correctness also holds with respect to $\mathsf{msk}' = (\mathsf{ltsk}, \mathsf{esk}')$ *and* csk^**, if* csk^* *is properly formatted. More formally, we have:*

Updatable Correctness. *For all security parameters* λ*, all constraints* $C \in \mathcal{C}$*, all inputs* $x \in \mathcal{X}$*, all properly formatted* csk^**, and all* $\alpha \in \mathcal{S}$ *with* $C(x, \alpha) = 0$ *(authorized), it holds:*

$$\Pr\left[\mathsf{Eval}(\mathsf{msk}', x, \alpha) = \mathsf{CEval}(\mathsf{csk}^*, x) \quad : \quad \begin{array}{r}(\mathsf{ltsk}, \mathsf{esk}) \leftarrow \mathsf{KeyGen}(1^\lambda)\\ \mathsf{esk}' := \mathsf{Update}(\mathsf{ltsk}, \mathsf{csk}^*, C)\\ \mathsf{msk}' := (\mathsf{ltsk}, \mathsf{esk}')\end{array}\right] = 1 - \mathsf{negl}(\lambda),$$

Multi-instance Updatable Security. *For every polynomial $\ell = \ell(\lambda) \in \mathsf{poly}(\lambda)$, an* ShCPRF *is (1-key, selectively, ℓ-instance)-updatably secure if for all efficient adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following security experiment $\mathsf{Exp}^{\mathsf{mi\text{-}shcprf}}_{\mathcal{A},b}(\lambda)$ is negligible in λ. Here, b denotes the challenge bit.*

1. ***Setup:*** *On input 1^λ, the challenger*
   - *runs $\mathcal{A}(1^\lambda)$ and receives ℓ constraints $C_i \in \mathcal{C}$ and the (possibly corrupted) constrained keys csk^*_i, for all $i \in [\ell]$,*
   - *computes $(\mathsf{ltsk}, \mathsf{esk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $\mathsf{esk}'_i := \mathsf{Update}(\mathsf{ltsk}, \mathsf{csk}^*_i, C_i)$,*
   - *sets $\mathsf{msk}_i := (\mathsf{ltsk}, \mathsf{esk}'_i)$, and*
   - *samples a uniformly random function $R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda$, where $\widetilde{\mathcal{F}}_\lambda$ denotes the set of all functions from $[\ell(\lambda)] \times \mathcal{X}_\lambda \times \mathcal{S}$ to $\mathcal{Y}$.*
2. ***Evaluation queries:*** *$\mathcal{A}$ adaptively sends arbitrary inputs $x \in \mathcal{X}$, shifts $\alpha \in \mathcal{S}$, and index $i \in [\ell]$ to the challenger. For each triple (x, α, i), if $C_i(x, \alpha) = 0$, then the challenger returns $\perp$. Otherwise, the challenger proceeds as follows:*
   - *If $b = 0$, it computes $y := \mathsf{Eval}(\mathsf{msk}_i, x, \alpha)$ and returns y.*
   - *If $b = 1$, it computes $y := R(i, x, \alpha)$ and returns y.*
3. ***Guess:*** *$\mathcal{A}$ outputs its guess b', which is the output of the experiment.*

$\mathcal{A}$ wins if $b' = b$, and its advantage $\mathsf{Adv}^{\mathsf{mi\text{-}shcprf}}_{\mathcal{A}}(\lambda)$ is defined as

$$\mathsf{Adv}^{\mathsf{mi\text{-}shcprf}}_{\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{mi\text{-}shcprf}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{mi\text{-}shcprf}}_{\mathcal{A},1}(\lambda) = 1]\right|,$$

where the probability is over the randomness of $\mathcal{A}$ and KeyGen*.*

Pseudorandomness. *Define the following stateful oracle $\mathcal{O}_{\mathsf{mi\text{-}eval}}$:*

Oracle $\mathcal{O}_{\mathsf{mi\text{-}eval}}$

Initialize. Sample $\mathsf{msk}_i \leftarrow \mathsf{KeyGen}(1^\lambda)$, for all $i \in [\ell]$.

Evaluation. On input (x, α, i), return $\mathsf{Eval}(\mathsf{msk}_i, x, \alpha)$.

An ShCPRF is said to be multi-instance pseudorandom if for all efficient adversaries $\mathcal{A}$, it holds that

$$\left|\Pr[\mathcal{A}^{\mathcal{O}_{\mathsf{mi\text{-}eval}}(\cdot,\cdot,\cdot)}(1^\lambda) = 1] - \Pr_{R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda}[\mathcal{A}^{R(\cdot,\cdot,\cdot)}(1^\lambda) = 1]\right| \leq 1 - \mathsf{negl}(\lambda),$$

where the left probability is over the randomness of KeyGen*, and $\widetilde{\mathcal{F}}_\lambda$ is the set of all functions from $[\ell] \times \mathcal{X}_\lambda \times \mathcal{S}$ to $\mathcal{Y}$.*

Updatability for the Ring-based ShCPRF for Inner Products

Public Parameters.

- Security parameter λ.
- Finite ring $\mathcal{R}$ of order ℓ.
- Integers m such that $m \geq \lambda$.
- Vector length $n \geq 1$.
- RKA-secure PRF family $F\colon \mathcal{R}^m \times \mathcal{R}^n \to \mathcal{Y}$ for affine RKA key-derivation functions.

$\mathsf{ShCPRF.KeyGen}(1^\lambda)$:

1 : $\mathsf{ltsk} := \Delta \xleftarrow{R} \mathcal{R}^m \setminus \{0\}$
2 : $\mathsf{esk} := (\mathbf{Z_0}, \mathbf{k_0}) \xleftarrow{R} \mathcal{R}^{m \times n} \times \mathcal{R}^m$
3 : **return** $\mathsf{msk} := (\mathsf{ltsk}, \mathsf{esk})$

$\mathsf{ShCPRF.Update}(\mathsf{ltsk}, \mathsf{csk}, \mathbf{z})$:

1 : **parse** $\mathsf{ltsk} =: \Delta$
2 : **parse** $\mathsf{csk} := (\mathbf{k'_0}, \mathbf{Z_1})$
3 : $\mathbf{Z'_0} := \mathbf{Z_1} + \Delta \mathbf{z}^\top$
4 : **return** $\mathsf{esk}' = (\mathbf{Z'_0}, \mathbf{k'_0})$

The remaining algorithms ($\mathsf{ShCPRF.Constrain}, \mathsf{ShCPRF.Eval}, \mathsf{ShCPRF.CEval}$), are as defined in Figure 3.2. Note that due to the now slightly different format of msk, the algorithms $\mathsf{ShCPRF.Constrain}$ and $\mathsf{ShCPRF.Eval}$ defined in Figure 3.2 need to be adapted to parse msk accordingly.

Figure 3.10: A modified version of the KeyGen, and the additional Update algorithm for the ring-based shiftable CPRF framework from Figure 3.2.

3.7.2 Constructing ℓ-instance updatably-secure shiftable CPRFs

We now describe a simple modification of the ShCPRF construction from Section 3.4 and prove that it satisfies updatable correctness and ℓ-instance updatable security. The construction is given in Figure 3.10.

Theorem 3.7.1. *If $\mathcal{F}$ is a family of RKA-secure pseudorandom functions with respect to affine related key derivation functions Φ_{aff}, as defined in Definition 2.3.6, then for every polynomial $\ell(\lambda) \in \mathsf{poly}(\lambda)$, Figure 3.10 instantiated with $\mathcal{F}$ is a (1-key, updatably, selectively, ℓ-instance)-secure ShCPRF for inner-product predicates.*

Proof. Deferred to Section 3.11.3. ■

3.7.3 Defining public-key PCFs for ListOT

In this section, we introduce the notion of public-key PCFs for ListOT. Our definition is geared towards our main application: non-interactive setup of pairwise OT channels over a large-scale network. Crucially for this application, our security notions incorporate the definition of *ℓ-instance security*, which says (informally) that if a user uses the *same* public

key pk to establish an OT channel with ℓ different parties, then security is maintained even against an adversary corrupting all ℓ parties.

Definition 3.7.2 (ℓ-Instance Public-Key Pseudorandom Correlation Function for ListOT)**.** *Let λ be a security parameter and $\lambda \leq n = n(\lambda) \in \mathsf{poly}(\lambda)$ be an input length. A Public-Key Pseudorandom Correlation Function (pkPCF) for ListOT is defined by a triple of algorithms $\mathsf{pkPCF} = (\mathsf{Gen}, \mathsf{KeyDer}, \mathsf{Eval})$ with the following functionality. We leave the public parameters PP as an implicit input to all algorithms.*

- $\mathsf{pkPCF.Gen}(1^\lambda, \sigma) \to (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$. *The randomized key generation algorithm takes as input the security parameter λ and a party index σ. It outputs a public and private key pair $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$.*
- $\mathsf{pkPCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{\bar{\sigma}}) \to K_\sigma$. *The deterministic key derivation algorithm takes as input a party identifier $\sigma \in \{S, R\}$ (where $\bar{\sigma}$ denotes the other party's identifier), a secret key sk_σ, and a public key $\mathsf{pk}_{\bar{\sigma}}$. It outputs a key K_σ. We assume this algorithm is deterministic.*
- $\mathsf{pkPCF.Eval}(\sigma, K_\sigma, x) \to y_\sigma$. *The deterministic evaluation algorithm takes as input $\sigma \in \{S, R\}$, a key K_σ, and input $x \in \mathcal{X}$. It outputs a string $y_\sigma \in \{0,1\}^*$, where*
  - *if $\sigma = S$ then $y_\sigma = (L_0, L_1)$ for two lists L_0, L_1, and*
  - *if $\sigma = R$ then $y_\sigma = (b, v, \alpha)$ for a bit $b \in \{0,1\}$, a list entry $v \in L_0 \cup L_1$, and a lookup key α.*

We will use $\mathsf{pkPCF.EvalS}(K_S, x)$ and $\mathsf{pkPCF.EvalR}(K_R, x)$ as shorthand for the $\mathsf{pkPCF.Eval}$ algorithm used by the sender and receiver, respectively.

A public key PCF $\mathsf{pkPCF} = (\mathsf{Gen}, \mathsf{KeyDer}, \mathsf{Eval})$ is a (weak) *ℓ-instance public-key PCF for* ListOT*, if the following* correctness*, ℓ-instance sender security, and ℓ-instance receiver* security *properties hold. In each case, the adversary is given access to $N(\lambda) \in \mathsf{poly}(\lambda)$ samples.*

- ***Pseudorandomness.*** *For all efficient adversaries $\mathcal{A}$, and all $N \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all sufficiently large λ,*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{pkpr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pkpr}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

  where $\mathsf{Exp}^{\mathsf{pkpr}}_{\mathcal{A},N,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 3.11.

- ***Correctness.*** *We want that for any $\lambda \in \mathbb{N}$ the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{rl} & (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{pkPCF.Gen}(1^\lambda, \sigma) \text{ for } \sigma \in \{S, R\} \\ & K_\sigma := \mathsf{pkPCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{\bar{\sigma}}) \text{ for } \sigma \in \{S, R\} \\ v \neq L_b[\alpha] \quad : & x \xleftarrow{R} \mathcal{X}_\lambda \\ & (L_0, L_1) := \mathsf{pkPCF.EvalS}(K_S, x) \\ & (b, v, \alpha) := \mathsf{pkPCF.EvalR}(K_R, x) \end{array}\right],$$

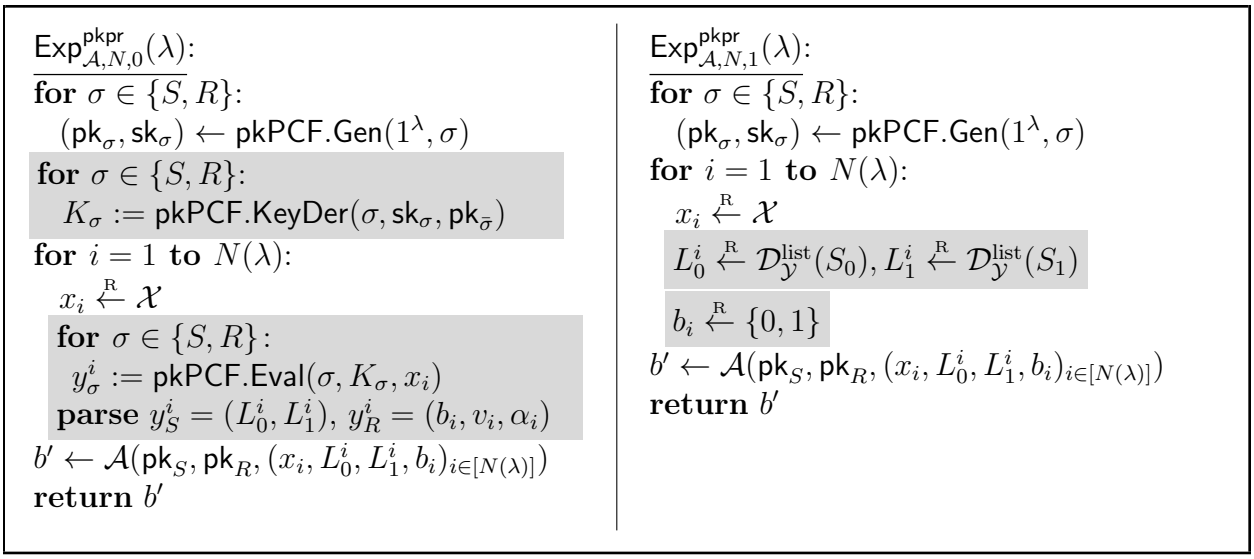


Figure 3.11: (Partially) Pseudorandom outputs of a public-key PCF for ListOT.

i.e., that the (relevant) entry v is at position α of list L_b with a probability that is overwhelming in λ.

- ***ℓ-Instance Sender Security.*** *For all efficient adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,S,0}(\lambda,\ell(\lambda)) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,S,1}(\lambda,\ell(\lambda)) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,S,b}(\lambda,\ell(\lambda))$, for $b \in \{0,1\}$, is as defined in Figure 3.12.

- ***ℓ-Instance Receiver Security.*** *For all efficient adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,R,0}(\lambda,\ell(\lambda)) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,R,1}(\lambda,\ell(\lambda)) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,S,b}(\lambda,\ell(\lambda))$, for $b \in \{0,1\}$, is as defined in Figure 3.13.

3.7.4 Constructing public-key PCFs for ListOT

In this section, we provide a construction of public-key PCFs for ListOT. In a nutshell, our construction is exactly the construction of PCF for ListOT (Figure 3.6) instantiated with an ℓ-instance updatable shiftable CPRF, enhanced with a distributed public-key setup protocol $\mathsf{PKS} = (\mathsf{Gen}, \mathsf{KeyDer})$ to generate the ShCPRF keys. Formally, we define a distributed public-key setup protocol as follows:

Definition 3.7.3 (Distributed Public-Key Setup Protocol)**.** *A distributed, corruptible public-key setup protocol* PKS *is parameterized by an updatable, shiftable CPRF* $\mathsf{ShCPRF} = (\mathsf{KeyGen},$

$\underline{\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,0}(\lambda,\ell(\lambda))}$:
$(\mathsf{pk}_S,\mathsf{sk}_S) \leftarrow \mathsf{pkPCF.Gen}(1^\lambda, S)$
for $i=1$ to ℓ,
  $(\mathsf{pk}^i_R,\mathsf{sk}^i_R) \leftarrow \mathsf{pkPCF.Gen}(1^\lambda, R)$
  $K^i_S := \mathsf{pkPCF.KeyDer}(S,\mathsf{sk}_S,\mathsf{pk}^i_R)$
  $K^i_R := \mathsf{pkPCF.KeyDer}(R,\mathsf{sk}^i_R,\mathsf{pk}_S)$
  for $j=1$ **to** $N(\lambda)$:
    $x_{i,j} \xleftarrow{\mathrm{R}} \mathcal{X}$
    $(L^{i,j}_0, L^{i,j}_1) := \mathsf{PCF.EvalS}(K^i_S, x_{i,j})$
$\mathsf{corr} := ((\mathsf{pk}^1_R,\mathsf{sk}^1_R),\ldots,(\mathsf{pk}^\ell_R,\mathsf{sk}^\ell_R))$
$b' \leftarrow \mathcal{A}(\mathsf{pk}_S, \mathsf{corr}, (x_{i,j}, L^{i,j}_0, L^{i,j}_1)_{i\in[\ell],j\in[N]})$
return b'

$\underline{\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,1}(\lambda,\ell(\lambda))}$:
$(\mathsf{pk}_S,\mathsf{sk}_S) \leftarrow \mathsf{pkPCF.Gen}(1^\lambda, S)$
for $i=1$ to ℓ,
  $(\mathsf{pk}^i_R,\mathsf{sk}^i_R) \leftarrow \mathsf{pkPCF.Gen}(1^\lambda, R)$
  $K^i_S := \mathsf{pkPCF.KeyDer}(S,\mathsf{sk}_S,\mathsf{pk}^i_R)$
  $K^i_R := \mathsf{pkPCF.KeyDer}(R,\mathsf{sk}^i_R,\mathsf{pk}_S)$
  for $j=1$ **to** $N(\lambda)$:
    $x_{i,j} \xleftarrow{\mathrm{R}} \mathcal{X}$
    $(b_{i,j}, v_{i,j}, \alpha_{i,j}) := \mathsf{PCF.EvalR}(K^i_R, x_{i,j})$
    $L^{i,j}_0 \xleftarrow{\mathrm{R}} \mathcal{D}^{\mathrm{list}}_{\mathcal{Y}}(S_0), L^{i,j}_1 \xleftarrow{\mathrm{R}} \mathcal{D}^{\mathrm{list}}_{\mathcal{Y}}(S_1)$
    Set $L^{i,j}_{b_{i,j}}[\alpha_{i,j}] := v_{i,j}$
$\mathsf{corr} := ((\mathsf{pk}^1_R,\mathsf{sk}^1_R),\ldots,(\mathsf{pk}^\ell_R,\mathsf{sk}^\ell_R))$
$b' \leftarrow \mathcal{A}(\mathsf{pk}_S, \mathsf{corr}, (x_{i,j}, L^{i,j}_0, L^{i,j}_1)_{i\in[\ell],j\in[N]})$
return b'

Figure 3.12: ℓ-instance sender security game of a pkPCF for ListOT.

Eval, Constrain, CEval, Update) *with constraint class* $\mathcal{C}$*, and consists of the following algorithms. We leave the public parameters* PP *as an implicit input to all algorithms.*

- $\mathsf{PKS.Gen}(1^\lambda,\sigma,m_\sigma) \to (\mathsf{pk}_\sigma,\mathsf{sk}_\sigma)$. *The randomized key generation algorithm takes as input a party identifier* $\sigma \in \{S,R\}$*, a message* m_σ *where* m_σ *is a long-term secret key* Itsk *if* $\sigma = S$ *and a constraint* $C \in \mathcal{C}$ *if* $\sigma = R$*. It outputs a public-secret key pair.*
- $\mathsf{PKS.KeyDer}(\sigma,\mathsf{sk}_\sigma,\mathsf{pk}_{\overline{\sigma}}) \to K_\sigma$. *The deterministic key derivation algorithm takes as input a party identifier* $\sigma \in \{S,R\}$*, the corresponding secret key* sk_σ*, and the other party's public key* $\mathsf{pk}_{\overline{\sigma}}$*. It outputs an evaluation key* K_σ*.*

We say $\mathsf{PKS} = (\mathsf{Gen},\mathsf{KeyDer})$ *is a distributed corruptible public-key setup if the following one-message protocol realizes the ideal functionality described in Figure 3.14:*

1. *Each sender and receiver runs* PKS.Gen *and broadcasts the resulting public key.*
2. *Each sender (resp. receiver) uses the public key of each receiver (resp. sender) and its own secret key to derive a shared updatable ShCPRF key using* PKS.KeyDer.

Theorem 3.7.2. *Let* $n = n(\lambda) \in \mathsf{poly}(\lambda)$ *and* $\mathcal{R}$ *be a finite ring of order* q*, with extension parameter* $m \geq \lambda$*. Let* $\mathsf{ShCPRF} = (\mathsf{KeyGen},\mathsf{Eval},\mathsf{Constrain},\mathsf{CEval},\mathsf{Update})$ *be an* ℓ*-instance updatable ShCPRF for inner-product predicates with domain* $\mathcal{R}^n$*, and* $f\colon \mathcal{R}^n \times \mathcal{R}^n \to \{0,1\}$ *a weak PRF family for inner-product membership with partitioning* $S_0 \cup S_1 = \mathcal{R}$*. Let* $\mathsf{PKS} = (\mathsf{Gen},\mathsf{KeyDer})$ *be a distributed public-key setup protocol for* ShCPRF*. Let* $\mathsf{pkPCF} = (\mathsf{Gen},\mathsf{KeyDer},\mathsf{Eval})$ *denote the PCF from Figure 3.6 (parameterized by* ShCPRF*), where the algorithm* PCF.KeyGen *is replaced by the distributed public-key setup algorithms:*

- $\mathsf{pkPCF.Gen}(1^\lambda,\sigma)$. *Takes as input a security parameter and party identifier* $\sigma \in \{S,R\}$*, and*

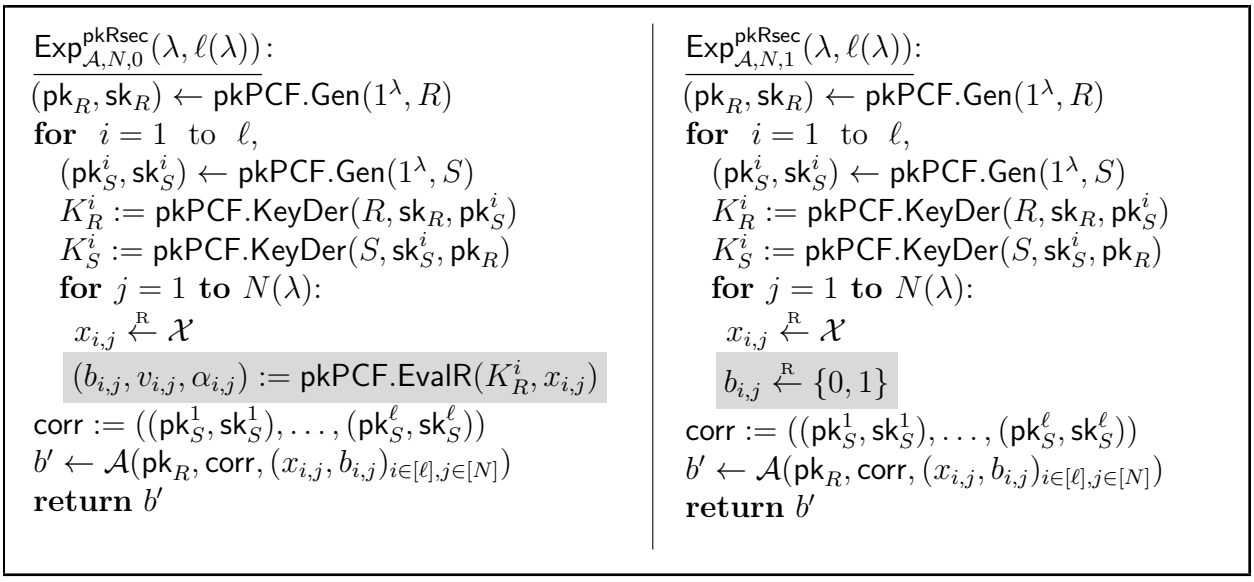


Figure 3.13: ℓ-instance receiver security game of a pkPCF for ListOT.

- *if* $\sigma = S$, *samples* $(\mathsf{ltsk},\mathsf{esk}) \leftarrow \mathsf{ShCPRF.KeyGen}(1^\lambda)$ *and sets* $m_\sigma := \mathsf{ltsk}$,
- *if* $\sigma = R$, *samples a random constraint* $\mathbf{z}$ *over* $\mathcal{R}^n$ *and sets* $m_\sigma := \mathbf{z}$.

Outputs $(\mathsf{pk}_\sigma,\mathsf{sk}_\sigma) \leftarrow \mathsf{PKS.Gen}(1^\lambda,\sigma,m_\sigma)$.

- $\mathsf{pkPCF.KeyDer}(\sigma,\mathsf{sk}_\sigma,\mathsf{pk}_{\bar{\sigma}}) := \mathsf{PKS.KeyDer}(\sigma,\mathsf{sk}_\sigma,\mathsf{pk}_{\bar{\sigma}})$.

Then, pkPCF *is an ℓ-instance secure public-key PCF for ListOT.*

Proof. We consider each property in turn.

Pseudorandomness. We prove pseudorandomness via a sequence of hybrids.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists $\mathsf{Exp}^{\mathsf{pkpr}}_{\mathcal{A},N,0}(\lambda)$ from Figure 3.11, where pkPCF.KeyGen, pkPCF.KeyDer and pkPCF.Eval are as defined in Theorem 3.7.2.

- *Hybrid* $\mathcal{H}_1$. In this hybrid, we rely on the simulator $\mathcal{S}_{\mathsf{PKS}}$ for PKS. $\mathcal{S}_{\mathsf{PKS}}$ sends an empty message to $\mathcal{F}_{\mathsf{PKS}}$ on behalf of the ideal functionality, and receives no output. Then, it emulates the distribution of $(\mathsf{pk}_S,\mathsf{pk}_R)$. By security of the PKS scheme, $\mathcal{H}_0 \approx_c \mathcal{H}_1$. Note that in this hybrid, the distribution of $(\mathsf{pk}_S,\mathsf{pk}_R)$ is independent of $\mathbf{z}$ and msk.

- *Hybrid* $\mathcal{H}_2$. In this hybrid game, we replace each pseudorandom bit b_i, sampled in $\mathcal{H}_1$ using the IPM-wPRF f, with truly random bits sampled uniformly at random. The proof that $\mathcal{H}_2 \approx_c \mathcal{H}_1$ reduces to the pseudorandomness of f and is identical to the proof between hybrids $\mathcal{H}_1$ and $\mathcal{H}_0$ from the pseudorandomness proof of Theorem 3.5.1.

- *Hybrid* $\mathcal{H}_3$. In this hybrid, for all $i \in [N(\lambda)]$, the lists L^i_0, L^i_1 are sampled uniformly from $\mathcal{D}^{\mathrm{list}}_{\mathcal{Y}}(S_0)$ and $\mathcal{D}^{\mathrm{list}}_{\mathcal{Y}}(S_1)$, respectively, where the distribution $\mathcal{D}^{\mathrm{list}}_{\mathcal{Y}}(\cdot)$ is defined in Definition 3.5.1 and consists of uniformly random samples from $\mathcal{Y}$. The proof that $\mathcal{H}_3 \approx_c \mathcal{H}_2$ reduces to the pseudorandomness of ShCPRF and is identical to the proof between hybrids $\mathcal{H}_2$ and $\mathcal{H}_1$ in the pseudorandomness proof of Theorem 3.5.1.

Functionality $\mathcal{F}_{\mathsf{PKS}}$

Parameters. The ideal corruptible functionality $\mathcal{F}_{\mathsf{PKS}}$ is parameterized by an updatable shiftable CPRF $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval}, \mathsf{Update})$ that supports constraints in a constraint class $\mathcal{C}$.

Parties. An adversary $\mathcal{A}$, senders $S_1, \ldots, S_\ell$, and receivers $R_1, \ldots, R_\ell$.

Procedure.
The functionality aborts if it receives any incorrectly formatted message.

Generation phase (one time).

1: Receive a message containing a long-term secret key and an index (ltsk_i, i) from every sender S_i for all $i \in [\ell]$.
2: Receive a message containing a constraint and an index (C_j, j) from receiver R_j, for all $j \in [\ell]$.
3: Send ready to $\mathcal{A}$.

Key derivation phase (repeatable).

1: Receive a message (keyder, j) from a sender S_i and a message (keyder, i) from a receiver R_j, for some $i, j \in [\ell]$.
2: Receive a message from $\mathcal{A}$ which is either empty, contains an ephemeral master secret key esk, or contains a constrained key csk.
3: If $\mathcal{A}$ sent an empty message (i.e., the sender S_i and receiver R_j are both honest), then sample esk uniformly as in KeyGen and compute $\mathsf{csk} \leftarrow \mathsf{Constrain}((\mathsf{ltsk}_i, \mathsf{esk}), C_j)$.
4: If $\mathcal{A}$ sent esk (i.e., the sender S_i is corrupted), then compute $\mathsf{csk} \leftarrow \mathsf{Constrain}((\mathsf{ltsk}_i, \mathsf{esk}), C_j)$.
5: If $\mathcal{A}$ sent csk (i.e., receiver R_j is corrupted), then compute $\mathsf{esk} := \mathsf{Update}(\mathsf{ltsk}_i, \mathsf{csk}, C_j)$.
6: Output (esk, j) to S_i and (csk, i) to R_j.

Figure 3.14: Corruptible ideal functionality for the distributed public key setup.

We observe that $\mathcal{H}_3$ is identical to $\mathsf{Exp}^{\mathsf{pkpr}}_{\mathcal{A},N,1}(\lambda)$, which concludes the proof of pseudorandomness.

Correctness. This is similar to the correctness proof of Theorem 3.5.1. Observe that the entry v output by $\mathsf{PCF.EvalR}(K_R, x)$ is computed as $v := \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x})$, where $\mathbf{x} := \mathsf{map}(x)$ and csk is computed as in the key derivation phase of $\mathcal{F}_{\mathsf{PKS}}$ via $\mathsf{csk} \leftarrow \mathsf{ShCPRF.Constrain}(\mathsf{msk}, \mathbf{z})$ for a constraint $\mathbf{z} \in \mathcal{R}^n$, if the adversary $\mathcal{A}$ is honest, or is a properly formatted csk as output by $\mathcal{A}$, in which case $\mathsf{esk}' := \mathsf{Update}(\mathsf{ltsk}, \mathsf{csk}, \mathbf{z})$. In the first case ($\mathcal{A}$ is honest), correctness follows exactly as in the proof of Theorem 3.5.1, using the correctness of ShCPRF. Similarly, for the second case (with a properly formatted csk given by $\mathcal{A}$), we make use of the updatable correctness of ShCPRF. Hence, using $\alpha = \langle \mathbf{z}, \mathbf{x} \rangle$, the entry v is equal to $\mathsf{Eval}(\mathsf{msk}', x, \alpha)$, where $\mathsf{msk}' := (\mathsf{ltsk}, \mathsf{esk}')$ with $(\mathsf{ltsk}, \mathsf{esk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Then, for the (unique) $b' \in \{0,1\}$ with $\alpha \in S_{b'}$, we have that, with overwhelming probability, $L_{b'}[\alpha] = v$. As before, $b' = b$ by the property of the IPM-wPRF which guarantees that $b := f_{\mathbf{z}}(\mathbf{x}) = 0$ iff $\langle \mathbf{z}, \mathbf{x} \rangle \in S_0$ and $f_{\mathbf{z}}(\mathbf{x}) = 1$ iff $\langle \mathbf{z}, \mathbf{x} \rangle \in S_1$.

Sender Security. We have a sequence of hybrids.

- *Hybrid* $\mathcal{H}_0$. This consists of $\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,0}(\lambda)$ defined in Figure 3.12, where $\mathsf{pkPCF.KeyGen}$, $\mathsf{pkPCF.KeyDer}$ and $\mathsf{pkPCF.Eval}$ are as defined in Theorem 3.7.2. In particular, we note that $\mathsf{pkPCF.EvalS}$ internally runs the updatable ShCPRF $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval}, \mathsf{Update})$.

- *Hybrid* $\mathcal{H}_1$. In this hybrid, for each $i \in [\ell(\lambda)]$ and for each $j \in [N(\lambda)]$, the call to $\mathsf{ShCPRF.Eval}(\mathsf{msk}_i, \mathbf{x}, \alpha)$ inside of the algorithm $\mathsf{pkPCF.EvalS}$ is replaced with a call to $\mathsf{ShCPRF.CEval}(\mathsf{csk}_i, \mathbf{x})$ whenever $(\mathbf{x}, \alpha)$ is authorized (i.e., $\langle \mathbf{z}_i, \mathbf{x} \rangle - \alpha = 0$), where csk_i is part of K_{R_i} in the $\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,0}(\lambda)$ experiment.

  Claim. $\mathcal{H}_1 \approx_s \mathcal{H}_0$.

  Proof. By the updatable correctness of updatable ShCPRFs, we have that for all authorized $(\mathbf{x}_{i,j}, \alpha)$ pairs, $\mathsf{ShCPRF.Eval}(\mathsf{msk}_i, \mathbf{x}_{i,j}, \alpha) = \mathsf{ShCPRF.CEval}(\mathsf{csk}_i, \mathbf{x}_{i,j})$, with overwhelming probability. □

- *Hybrid* $\mathcal{H}_2$. In this hybrid, we rely on the simulator $\mathcal{S}_{\mathsf{PKS}}$ for PKS. $\mathcal{S}_{\mathsf{PKS}}$ simulates pk_S and extracts the constrained keys $(\mathsf{csk}_j)_{j \le \ell}$ computed by the ℓ corrupted receivers from $(\mathsf{pk}_S, \mathsf{sk}_R^j)$. It sends csk_j to $\mathcal{F}_{\mathsf{PKS}}$ on behalf of $\mathcal{A}$ for each corrupted receiver R_j. Note that from this game onward, the updated master secret keys msk_j are not known anymore, which will allow us to invoke the multi-instance updatable security of ShCPRF in the next hybrid.

- *Hybrid* $\mathcal{H}_3$. In this hybrid, the lists $L_0^{i,j}$, $L_1^{i,j}$ are sampled uniformly at random from $\mathcal{D}_{\mathcal{Y}}^{\mathrm{list}}(S_0), \mathcal{D}_{\mathcal{Y}}^{\mathrm{list}}(S_1)$, respectively. Then, for each (i,j) where $(\mathbf{x}_{i,j}, \alpha_{i,j})$ is an authorized pair, find the $b_{i,j} \in \{0,1\}$, such that $\alpha_{i,j} \in S_{b_{i,j}}$, and overwrite $L_{b_{i,j}}^{i,j}[\alpha_{i,j}] := \mathsf{CEval}(\mathsf{csk}_i, \mathbf{x}_{i,j})$.

  Claim. $\mathcal{H}_3 \approx_c \mathcal{H}_2$.

Proof. This follows from the ℓ-instance updatable security of ShCPRF; the proof proceeds essentially as the proof between $\mathcal{H}_3$ and $\mathcal{H}_2$ in the sender security proof of Theorem 3.7.2. □

Observe that $\mathcal{H}_3$ is exactly the experiment $\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,1}(\lambda)$ experiment defined in Figure 3.12, which concludes the proof of sender security.

Receiver Security. We have a sequence of hybrids.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the $\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,0}(\lambda)$ experiment defined in Figure 3.13, where pkPCF is as defined in Theorem 3.7.2.

- *Hybrid* $\mathcal{H}_1$. In this hybrid, we rely on the simulator $\mathcal{S}_{\mathsf{PKS}}$ for PKS. $\mathcal{S}_{\mathsf{PKS}}$ simulates pk_R and extracts the ephemeral keys $(\mathsf{esk}_j)_{j\leq\ell}$ computed by the ℓ corrupted senders from $(\mathsf{pk}_R, \mathsf{sk}^j_S)$. It sends esk_j to $\mathcal{F}_{\mathsf{PKS}}$ on behalf of $\mathcal{A}$ for each corrupted sender S_j. Note that from this game onward, the constraint $\mathbf{z}$ is not known anymore, which will allow us to invoke the wPRF security in the next hybrid.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$.

  Proof. This follows directly from the receiver security property of the PKS scheme, since pk_R is computationally indistinguishable from uniform. □

- *Hybrid* $\mathcal{H}_2$. In this hybrid, we sample a uniformly random function R from the set of all functions from $\mathcal{X}_\lambda$ to $\{0,1\}$, and generate $b_{i,j} := R(x_{i,j})$.

  Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$.

  Proof. The only difference between $\mathcal{H}_2$ and $\mathcal{H}_1$ is that $b_{i,j} = f_{\mathbf{z}}(x_{i,j})$ in $\mathcal{H}_1$, and $b_{i,j} = R(x_{i,j})$ in $\mathcal{H}_2$. As the $x_{i,j}$'s are uniformly random (and pk_R is simulated without using $\mathbf{z}$), any distinguisher between $\mathcal{H}_2$ and $\mathcal{H}_1$ immediately yields a distinguisher for the wPRF f, contradicting the pseudorandomness of $f_{\mathbf{z}}$. □

- *Hybrid* $\mathcal{H}_3$. In this hybrid, each bit $b_{i,j}$ is sampled uniformly at random: $b_{i,j} \xleftarrow{\text{R}} \{0,1\}$. Note that this hybrid is exactly $\mathsf{Exp}^{\mathsf{pkRsec}}_{\mathcal{A},N,1}(\lambda)$.

  Claim. $\mathcal{H}_3 \approx_s \mathcal{H}_2$.

  Proof. Since R is a truly random function, $\mathcal{H}_3$ and $\mathcal{H}_2$ are perfectly indistinguishable conditioned on all $x_{i,j}$'s being distinct. By a straightforward union bound, since all $x_{i,j}$'s are sampled randomly from $\mathcal{X}$, the condition is satisfied except with probability at most $(N \cdot \ell)^2/|\mathcal{X}_\lambda|$, which is negligible in λ because $|\mathcal{X}_\lambda|$ is exponential in λ. □

This concludes the proof of receiver security, and the proof of Theorem 3.7.2. ■

3.7.5 Public-key setup from Ring LWE

Protocol. Here, we first informally describe the distributed public-key setup protocol $(\mathsf{Gen}, \mathsf{KeyDer})$. See Figure 3.15 for a formal construction.

Public-Key Setup for Ring-based PCF for ListOT

Public Parameters. Security parameter λ, finite ring $\mathcal{R} = \mathbb{Z}_t$, vector length $n \geq 1$, integer m such that $m \geq \lambda$, polynomial ring $\mathcal{P} = \mathbb{Z}_q[X]/(X^\eta + 1)$, noise distribution χ over $\mathcal{P}$, and random $a_0, a_1 \in \mathcal{P}$.

$\mathsf{Gen}(1^\lambda, S, \mathsf{ltsk} =: \Delta)$:

1 : $s_0^1, \ldots, s_0^m \xleftarrow{\text{R}} \chi$
2 : $\mathsf{sk}_S := (\Delta, s_0^1, \ldots, s_0^m)$
3 : $e_0^1, \ldots, e_0^m \xleftarrow{\text{R}} \chi$
4 : **foreach** $i \in [m]$:
5 : $\quad \mathsf{pk}_S^i := \Delta_i \cdot a_0 + s_0^i a_1 + e_0^i$
6 : $\mathbf{k}_0 \xleftarrow{\text{R}} \mathcal{R}^m$
7 : **return** $(\mathsf{pk}_S := (\mathbf{k}_0, (\mathsf{pk}_S^i)_{i \leq m}), \mathsf{sk}_S)$

$\mathsf{Gen}(1^\lambda, R, C_{\mathbf{z}})$:

1 : **parse** $\mathbf{z} \in \mathcal{R}^n$ from $C_{\mathbf{z}}$
2 : $s_1 \xleftarrow{\text{R}} \chi$
3 : $\mathsf{sk}_R := (\mathbf{z}, s_1)$
4 : let $z \in \mathcal{P}$ have coeffs. $\frac{q}{t} \cdot (\mathbf{z} \| 0^{\eta - n})$
5 : $e_1, e_1' \xleftarrow{\text{R}} \chi$
6 : $\mathsf{pk}_R := (z + s_1 a_0 + e_1, s_1 a_1 + e_1')$
7 : **return** $(\mathsf{pk}_R, \mathsf{sk}_R)$

$\mathsf{KeyDer}(S, \mathsf{sk}_S, \mathsf{pk}_R)$:

1 : **parse** $\mathsf{sk}_S = (\Delta, s_0^1, \ldots, s_0^m)$
2 : **foreach** $i \in [m]$:
3 : $\quad z_0^i := \langle \mathsf{pk}_R, (\Delta_i, s_0^i) \rangle$
4 : $\quad$ **parse** $z_0^i \in \mathcal{P}$ as $\tilde{\mathbf{z}}_0^i \in \mathbb{Z}_q^n$
5 : $\quad$ **round** $\mathbf{z}_0^i = \lceil \tilde{\mathbf{z}}_0^i \rfloor_t \in \mathbb{Z}_t^n$
6 : $\mathsf{esk} = (\mathbf{k}_0, \mathbf{Z}_0 = (\mathbf{z}_0^i)_{i \leq m})$
7 : **return** $K_S := (\Delta, \mathsf{esk})$

$\mathsf{KeyDer}(R, \mathsf{sk}_R, \mathsf{pk}_S)$:

1 : **parse** $\mathsf{sk}_R = (\mathbf{z}, s_1)$, $\mathsf{pk}_S := (\mathbf{k}_0, (\mathsf{pk}_S^i)_{i \leq m})$
2 : **foreach** $i \in [m]$:
3 : $\quad z_1^i := \mathsf{pk}_S^i \cdot s_1$
4 : $\quad$ **parse** z_1^i as $\tilde{\mathbf{z}}_1^i \in \mathbb{Z}_q^n$
5 : $\quad$ **round** $\mathbf{z}_1^i = \lceil \tilde{\mathbf{z}}_1^i \rfloor_t \in \mathbb{Z}_t^n$
6 : $\mathsf{csk} := (\mathbf{k}_0, \mathbf{Z}_1 := (\mathbf{z}_1^i)_{i \leq m})$
7 : **return** $K_R := (\mathsf{csk}, \mathbf{z})$

Figure 3.15: A distributed public-key setup protocol for the ring-based updatable ShCPRF for inner products. Note that when parsing elements of $\mathcal{P}$ as vectors in $\mathbb{Z}_q^n$, we ignore the last $\eta - n$ coefficients.

We assume $\mathcal{R} = \mathbb{Z}_t$ and work over a polynomial ring $\mathcal{P} = \mathbb{Z}_q[X]/(X^\eta + 1)$ (we will define the RLWE parameters η, q later). The public parameters include random $a_0, a_1 \in \mathcal{P}$. In the public-key generation phase, the sender samples Δ from $\mathcal{R}^m \setminus \{0\}$, m secrets $s_0^1, \ldots, s_0^m$ from a noise distribution, and noise $e_0^1, \ldots, e_0^m$. It publishes as its public key $\mathsf{pk}_S = (\Delta_i \cdot a_0 + s_0^i a_1 + e_0^i)_{i \in [m]}$, along with $\mathbf{k}_0$, which it samples randomly from $\mathcal{R}^m$.

The receiver samples $\mathbf{z}$, a secret s_1 from the noise distribution, and noise e_1, e_1'. It encodes $\frac{q}{t} \cdot \mathbf{z}$ in a polynomial $z \in \mathcal{P}$ and publishes as its public key $\mathsf{pk}_R = (z + s_1 a_0 + e_1, s_1 a_1 + e_1')$.

In the evaluation-key derivation phase, the sender computes, for each $i \in [m]$, the rounded inner product $\lceil \langle \mathsf{pk}_R, (\Delta_i, s_0^i) \rangle \rfloor_t$[10] and parses the resulting m polynomials as a matrix in $\mathbb{Z}_t^{m \times n}$ (ignoring the last $\eta - n$ coefficients when parsing polynomial ring elements as vectors). The receiver computes, for each $i \in [m]$, the rounded product $\lceil \mathsf{pk}_S^i \cdot s_1 \rfloor_t$ and, as before, parses the result as a matrix in $\mathbb{Z}_t^{m \times n}$.

Concrete parameters. We rely on the RLWE assumption (with a superpolynomial modulus-to-noise ratio) in the polynomial ring $\mathcal{P} = \mathbb{Z}_q[X]/(X^\eta + 1)$ for some η a power of 2. We use the normal form (see Definition 3.3.2), where the secret is drawn from the noise distribution rather than uniformly. This is a standard choice in practical RLWE-based schemes with hardness supported by security reductions [LPR13, ACC+18, dCJV21, MW22].

We use a standard choice of error distribution—a discrete Gaussian with standard deviation $\sigma = 3.2$. When applying the rounding lemma (Lemma 3.3.1) to prove correctness of our protocol, note that the failure probability of rounding an element of $\mathbb{Z}_q$ to an element of $\mathcal{R} = \mathbb{Z}_t$ will grow with $\frac{t \cdot B}{q}$, where B is a bound on the magnitude of the error term in the derived polynomials (that depends on χ and η). To ensure correctness with overwhelming probability, i.e., that with probability at least $1 - 2^{40}$, the sender and receiver correctly round all $n \cdot m$ coefficients, we will set $q = t \cdot B \cdot n \cdot m \cdot 2^{40}$.

We need that $\eta \geq n$ (for values of n chosen for both the BIPSW and GAR instantiations), and following post-quantum security standards for normal form RLWE [ACC+18], the choice of $\eta = 2^{12}$ can support up to a 103-bit modulus q, which is more than sufficient for our choice of q.

Public key size. The sender's public key consists of m elements of $\mathcal{P}$, as well as $\mathbf{k}_0 \in \mathcal{R}^m$, so we can bound the key size by $m \cdot (\eta \cdot \log q + \log t)$ bits. The sender's public key is roughly 5.5 MB in size for the BIPSW IPM-wPRF and GAR IPM-wPRF. However, the receiver's public key only consists of 2 elements of $\mathcal{P}$, which makes it roughly 85 kB in size.

Theorem 3.7.3. *The distributed public-key setup protocol* $\mathsf{PKS} = (\mathsf{Gen}, \mathsf{KeyDer})$ *defined in Figure 3.15 securely implements the corruptible ideal functionality represented in Figure 3.14 with respect to the ring-based updatable* ShCPRF *framework defined in Figures 3.2 and 3.10.*

Proof. Deferred to Section 3.11.4. ■

3.8 Implementation and Evaluation

Implementation. We implement QuietOT in C with roughly 1,200 lines of code for the BIPSW and GAR implementations combined. Our implementation is open source [SS24c].

We additionally implement the BCMPR silent OT protocol in C in roughly 600 lines of code using the P-256 elliptic curve implementation available in the OpenSSL library [Ope24]. For OSY, we estimate their runtime by benchmarking the dominant cost of their construction: computing $\lambda = 128$ modular exponentiations in a 3200-bit RSA group. To heuristically instantiate the random oracle $H(\cdot)$, we use fixed-key AES, operating with 128-bit inputs, and truncate the output to a single bit (such an instantiation is shown to be correlation-robust

[10] Recall that we write $\lceil \cdot \rfloor_t$ to denote "rounding" a polynomial coefficient-by-coefficient.

in the ideal cipher model [GKWY20]). To generate pseudorandom inputs for the PCF, we stretch a short seed (common to both the sender and receiver) using AES in CTR mode. Our implementations make use of several optimizations, which are described further in Section 3.8.1. We use the state-of-the-art implementation of existing OT extension protocols (IKNP, SoftSpokenOT, RRT) available in libOTe [RR20] to compare to other OT extension protocols. In order to provide a fairer comparison to existing OT extension protocols, we do *not* include the base OT costs required in SoftSpokenOT and IKNP.

Environment. We run our benchmarks on an AWS `c5.metal` and `t2.small` instances, and on an Apple M1 Pro laptop computer, using a single core. Because network latency and bandwidth can fluctuate leading to high variance, our benchmarks take into account only the processing time required by the sender and receiver. We compare the network overhead between each protocol using the "bits/OT" measure, which provides an objective and consistent comparison between protocols, avoiding network-specific or implementation differences.[11]

Parameters. We fix the security parameter $\lambda = 128$. For BIPSW, we set $n = 768$ and pre-compute inner products with CPRF keys in blocks of 16 bits (this is an optimization we describe in Section 3.8.1). We operate over the ring $\mathbb{Z}_6$, which allows us to use CRT decomposition and pack 128 elements of $\mathbb{Z}_2$ into one machine word. For GAR, we set $n = 2048$, $\ell_1 = 5$ and $\ell_2 = 15$. This allows us to work over the ring $\mathcal{R} = \mathbb{Z}_2 \times \mathbb{Z}_{16}$. The choice of $n = 2048$ is very conservative but allows us to sample indices in $\{1, \ldots, 2048\}$ efficiently without rejection sampling. In turn, this improves concrete performance by allowing us to efficiently generate inputs for the wPRF (all we require is checking that the sampled set of random indices consist of distinct elements). SoftSpokenOT has a tunable tradeoff between communication and computational efficiency parameterized by k. For a given k, SoftSpokenOT requires λ/k communication but increases computation by a factor of $2^k/k$. Small values of k (e.g., $k = 4$) provide a good tradeoff in practice, resulting in 32 bits/OT at an increase of $4\times$ in computation.

Communication costs and comparison. QuietOT with BIPSW as the IPM-wPRF requires 7 bits of communication per chosen-bit OT. For random choice bits, communication is only 6 bits since the receiver does not need to send its masked bit. Moreover, for random OT (when the sender inputs are also random), the messages m_0 and m_1 can be set to the first elements of L_0 and L_1, respectively, reducing communication to $|S_0| + |S_1| - 2$ (or 4 bits when using the BIPSW IPM-wPRF). QuietOT with GAR as the IPM-wPRF requires 33 bits of communication per chosen-bit OT. However, the same logic above reduces communication to 32 bits/OT when the choice bit is random and 30 bits/OT for random OT. QuietOT beats SoftSpokenOT on communication (for reasonable choices of k) when instantiated with BIPSW and remains on-par with SoftSpokenOT in terms of communication when instantiated with GAR. Silent OT protocols (i.e., RRT, BCMPR, OSY) have an optimal 3 bits/OT of communication and 2 bits/OT when the receiver's choice bit is random. This makes QuietOT roughly 2-10$\times$ worse in terms of communication when compared to silent OT.

Computational costs and comparison. The state-of-the-art OT extension protocol is

[11] The libOTe implementation is evaluated on `localhost`, and therefore is somewhat limited by the kernel when transferring data making IKNP slower than SoftSpokenOT.

	$\lvert\mathsf{pk}_{\mathsf{sender}}\rvert$	$\lvert\mathsf{pk}_{\mathsf{receiver}}\rvert$	**OT/s** (M1 Pro)	**OT/s** (c5.metal)	**OT/s** (t2.small)	**Bits/OT**
IKNP			2,592,000	34,174,000	12,264,000	128
SoftSpokenOT ($k = 2$)			2,732,000	52,676,000	33,121,000	64
SoftSpokenOT ($k = 4$)			1,636,000	44,443,000	27,504,000	32
SoftSpokenOT ($k = 8$)			249,000	9,500,000	5,891,000	16
SoftSpokenOT ($k = 16$)			2,000	76,000	49,000	8
RRT			1,230,000	6,856,000	2,492,000	3
OSY	50 kB	1 kB	0.6	0.5	0.3	3
BCMPR (BIPSW)	63 kB	72 kB	15,000	12,000	8,000	3
BCMPR (GAR)	33 kB	38 kB	21,000	17,000	11,000	3
QuietOT (BIPSW)	5.4 MB	84 kB	1,165,000	561,000	362,000	7
with AVX512 support			N/A	1,265,000	N/A	7
QuietOT (GAR)	5.6 MB	88 kB	1,198,000	526,000	336,000	33

Table 3.2: OTs per second on a single core generated by the sender. Note that libOTe is not optimized for M1 since the AVX instructions are not available on M1 processors, hence we report these numbers in gray. The GAR IPM-wPRF cannot benefit from AVX due to limited bit-slicing opportunities. Setup costs are excluded.

SoftSpokenOT. To provide an apples-to-apples comparison of the computational costs while fixing the communication overhead in SoftSpokenOT, we could set $k = 18$ and $k = 4$ in SoftSpokenOT, leading to 7.1 bits/OT and 32 bits/OT, respectively. However, SoftSpokenOT becomes very inefficient with large k which does not make the comparison fair when QuietOT is instantiated using BIPSW. Comparing to SoftSpokenOT with small k and QuietOT (when instantiated with either BIPSW or GAR) shows that QuietOT is roughly one to two orders of magnitude slower. However, we stress that SoftSpokenOT benefits a lot more from advanced hardware instructions than QuietOT, potentially making QuietOT outshine SoftSpokenOT on weak(er) devices. This is evidenced by QuietOT outperforming the SoftSpokenOT implementation on the M1 (where AVX512 is not available). Unfortunately, since the libOTe implementation is not optimized for performance when AVX is disabled, performing a head-to-head comparison difficult. Comparing QuietOT to BCMPR (state-of-the-art *public-key* OT protocol) shows that QuietOT is up to 100× faster in terms of computation while only increasing communication by a few bits.

Public key size. Our public key setup has public keys that are roughly 20 to 60 times larger compared to the public keys in BCMPR and OSY. This is primarily due to the parameters required for the RLWE assumption (see Section 3.7.5). However, as a consequence, we obtain plausible post-quantum security. In practical terms, however, the average web page size is roughly 2 MB as of 2023 [IS22], making the overall key size very reasonable for use on the Internet. Additionally, we note that this is the *sender*'s public key size—the receiver's public key in our construction is only around 90 kB, which could be beneficial to some applications.

3.8.1 Optimizations in the implementation

Our implementation makes use of several optimizations which we detail here.

Optimizing the BIPSW instantiation. We make several observations allowing us to concretely optimize the BIPSW instantiation from Figure 3.8.

Working over the CRT decomposition. All computations over $\mathcal{R} = \mathbb{Z}_6$ can be computed over the CRT decomposition $\mathbb{Z}_2 \times \mathbb{Z}_3 \simeq \mathbb{Z}_6$. This enables applying the following two optimizations:

- *Bit-sliced arithmetic in* $\mathbb{Z}_2$. We can pack the m elements of $\mathbb{Z}_2$ into $\lfloor m/64 \rfloor$ machine words (on 64-bit word processors). This allows parallelizing the $\mathbb{Z}_2$ component of all operations over $\mathcal{R}^m$ by using a single machine instruction for each batch of $\lfloor m/64 \rfloor$ elements of $\mathcal{R}^m$.

- *Bit-sliced arithmetic in* $\mathbb{Z}_3$. While we can also pack the $\mathbb{Z}_3$ elements into $\lfloor 2m/64 \rfloor$ machine words (using two bits for each element of $\mathbb{Z}_3$ on a 64-bit machine), such a packing requires computing operations in $\mathbb{Z}_3$ over the bit-sliced representation. Fortunately, this very problem was explored in WAVE [BCC+23, Appendix B.1], where they provide an efficient bit-sliced representation for computing fast bit-sliced arithmetic over $\mathbb{Z}_3$. In particular, each arithmetic operation in $\mathbb{Z}_3$ requires using only 7 machine instructions.

Our implementation in C. Concretely, we pack ring elements into `uint128_t` types in C (which represent two 64-bit machine words). This allows us to pack the 128 $\mathbb{Z}_2$ elements into one `uint128_t` type and pack the high and low order bits of the 128 $\mathbb{Z}_3$ elements into two `uint128_t` types. We can then perform bit-sliced arithmetic over this packed representation.

Preprocessing inner products. A separate optimization, which we find also applies to the construction of BCMPR when instantiated with the BIPSW IPM-wPRF, is to preprocess the inner products over small chunks of the key. When using the BIPSW wPRF, each matrix-vector product $\mathbf{Z_0 x}$ can be equivalently written as a sum of smaller matrix-vector products. More precisely, let $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{n/t})$ such that $|\mathbf{x}_i| \in \{0,1\}^t$ (we assume that t divides n) and let $\mathbf{Z_0} = (\mathbf{Z_0}_1, \ldots, \mathbf{Z_0}_{n/t})$. Then, by preprocessing all possible 2^t matrix-vector products associated with the i-th column block matrix $\mathbf{Z_0}_i$ and storing the results, we can efficiently look up the result for any input block $\mathbf{x}_i$, saving a factor t in computation at a cost of 2^t in extra storage.

Optimizing the GAR instantiation. Unfortunately, we find fewer ways to optimize the GAR instantation compared to our BIPSW instantiation. In particular, the GAR instantiation does not benefit from preprocessing opportunities that we identify for our BIPSW instantiation. However, we note that we can still take advantage of bit-sliced operations to compute the m operations over $\mathbb{Z}_2$ in parallel. Performing bit-sliced arithmetic over $\mathbb{Z}_{16}$ (when $\ell = 16$), in contrast, is more challenging.

Fast arithmetic over $\mathbb{Z}_{16}$. We found it to be more efficient to *not* pack the $\mathbb{Z}_{16}$ elements and instead just use one byte for each of the 128 elements of $\mathbb{Z}_{16}^{128}$. This allows us to sum modulo 16 by first computing the sum over the integers and then using a bit-mask to reduce modulo 16. In particular, we can sum two elements of $\mathbb{Z}_{16}$ via integer addition (summing two bytes) followed by a bit-wise AND with `0b00001111`, which zeroes-out the carry bit. This

optimization makes summation modulo 16 fast, mitigating the impact of not being able to perform bit-sliced arithmetic for the $\mathbb{Z}_{16}$ elements.

General optimization: Compressing hash inputs via universal hashing. In the ring element representations of both the BIPSW and GAR instantiations, we end up with a tightly packed representation of the $\mathbb{Z}_2$ elements but only a "loosely-packed" representation of the $\mathbb{Z}_3$ elements (resp. $\mathbb{Z}_{16}$ elements). The naive approach would be to feed the entire bit-string representation of the ring elements (the ShCPRF key) and input $\mathbf{x}$ into the random oracle, which is heuristically instantiated using fixed-key AES [GKWY20]. However, this would require breaking up the input string into blocks of 128 bits and then xoring all the resulting AES outputs together. While this solution does not introduce noticeable slowdowns for the BIPSW instantiation,[12] it is not ideal for the GAR instantiation. For the GAR implementation, this approach would require packing all the elements of $\mathbb{Z}_{16}^{128}$ into four AES blocks, which is inefficient since the packing itself is slow (recall that each element is represented as a byte for fast arithmetic operations). However, we additionally need to pack the ShCPRF input $\mathbf{x}$, which would lead to even more overheads. To avoid these inefficiencies, we instead choose to compress the representation of the ring elements and input $\mathbf{x}$ into tightly packed λ-bit strings by using a universal hash, which acts as a randomness extractor for the input to the random oracle. Specifically, we can make use of the leftover hash lemma [HILL99] to extract $\lambda \approx 128$ bits from the representation of the ring elements. This allows us to then only perform one AES call, using the compressed 128-bit representation as input. We do the same for the ShCPRF input $\mathbf{x}$ and the $\mathbb{Z}_2^{128}$ block, leading to a total of three independent AES calls that we then truncate and XOR together. The standard LHL bound requires $128 + 2\kappa$ bits of entropy to extract 128-bits that are statistically close to uniform in the worst-case [HILL99], where κ is a statistical security parameter. However, the generalized LHL bound of Barak [BDK+11] allows us to do better. Specifically, they prove that when extracting randomness to use as a key for a *weak* PRF (we assume that our ShCPRF takes uniformly random inputs and thus is a weak PRF)[13] the LHL bound can be improved to $128 + \kappa$, which saves a factor of two in the entropy loss. Therefore, we can increase the ring dimension m to $128 + 64$ to ensure the universal hashing produces a near-uniform 128 bit key for the ShCPRF, with ≥ 64-bits of *statistical* security. As for the input $\mathbf{x}$, universal hashing provides 128-bits with even more statistical security since under our concrete parameter choice for GAR, we already have 308 bits of entropy in the input $\mathbf{x}$, which already give more than 64-bits of statistical security under the basic LHL bound. Finally, to further improve efficiency, we use an *almost*-universal (as opposed to perfectly universal) hash function, which results in faster implementations while only sacrificing a few bits of statistical security [BDK+11]. In our implementation of the GAR instantiation, we use the open-source `Polymur` [Pet24] almost-universal hash function for this purpose.

[12] In particular, we only need ≈ 80 bits for the $\mathbb{Z}_3$ components (which can be represented with two 128-bit blocks), leaving 96 bits "on the table" into which we can pack the ShCPRF input string. This leads to only three AES calls to instantiate the random oracle.

[13] We can view the inputs to the ShCPRF as having ≥ 128 bits of entropy—even if they are potentially non-uniform—since the input to the ShCPRF in our framework is itself *an input to a weak PRF*.

3.9 Application to Large-Scale MPC

We consider an application where a large number of parties are interacting over a network. Each individual party is associated with a role (sender or receiver) and is identified by its public key. At any time, a pair of parties (S, R) with respective key pairs $(\mathsf{pk}_S, \mathsf{sk}_S)$ and $(\mathsf{pk}_R, \mathsf{sk}_R)$ can engage in a secure computation protocol, which reduces to a large number N of oblivious transfers [GMW87]. Both parties derive PCF keys for ListOT via KeyDer, compute N ListOTs, and use them in N instances of the OT protocol from Figure 3.7. In this setting, we note that the ℓ-instance security notions of public key PCFs for ListOT captures the following security properties:

Sender security. Consider a sender S with key pair $(\mathsf{pk}_S, \mathsf{sk}_S)$ interacting with ℓ corrupted receivers $(R_1, \ldots, R_\ell)$ with public keys $(\mathsf{pk}_{R_1}, \ldots, \mathsf{pk}_{R_\ell})$. The sender has $N \cdot \ell$ message pairs $(m_0^{i,j}, m_1^{i,j})_{i \le N, j \le \ell}$. The sender and the receivers agree on $N \cdot \ell$ uniformly random inputs $(x_{i,j})_{i \le N, j \le \ell}$.

- **Key derivation.** The sender S sets $K_S^j := \mathsf{pkPCF.KeyDer}(S, \mathsf{sk}_S, \mathsf{pk}_{R_j})$, for all $j \in [\ell]$.
- **Oblivious transfers.** For every $i \le N, j \le \ell$, S computes

  $$(L_0^{i,j}, L_1^{i,j}) := \mathsf{pkPCF.EvalS}(K_S^j, x_{i,j}).$$

  For every $j \le \ell$, upon receiving $(c_{i,j})_{i \le N}$ from R_j, S replies with $(L_{c_{i,j}}^{i,j} \oplus m_0^{i,j}, L_{1-c_{i,j}}^{i,j} \oplus m_1^{i,j})_{i \le N}$ (following Figure 3.7).

To argue security in this setting, consider the following sequence of hybrids.

- *Hybrid* $\mathcal{H}_0$. This hybrid is the protocol described above.
- *Hybrid* $\mathcal{H}_1$. In this hybrid game, a simulator $\mathcal{S}$ (who is given the random tapes of the receivers $R_1, \cdots, R_\ell$, samples the lists $(L_0^{i,j}, L_1^{i,j})$ exactly as in $\mathsf{Exp}^{\mathsf{pkSsec}}_{\mathcal{A},N,1}(\lambda, \ell(\lambda))$ from Figure 3.12, and plays the role of the sender exactly as in $\mathcal{H}_0$ afterwards. In this experiment, for any $i \le N$ and $j \le \ell$, the simulator $\mathcal{S}$ first samples $(L_0^{i,j}, L_1^{i,j})$ uniformly at random and then sets $L_{b_{i,j}}^{i,j}[\alpha_{i,j}] := v_{i,j}$, where $(b_{i,j}, v_{i,j}, \alpha_{i,j}) := \mathsf{pkPCF.EvalR}(K_R^i, x_{i,j})$ (note that $\mathcal{S}$ can reconstruct the keys K_R^j from the tapes of the receivers). Importantly, for every (i, j), the list $L_{\overline{b}_{i,j}}^{i,j}$ remains truly random.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$.

  Proof. This claim follows directly from the ℓ-instance sender security property of the public key PCF (cf. Figure 3.12). □

- *Hybrid* $\mathcal{H}_2$. In this hybrid game, the simulator $\mathcal{S}$ first samples pk_S, reconstructs all keys K_R^j using pk_S by deriving sk_R^j from the tape of the receiver R_j, and computes $(b_{i,j}, v_{i,j}, \alpha_{i,j}) := \mathsf{pkPCF.EvalR}(K_R^i, x_{i,j})$. For all $i \le N, j \le \ell$ it sends $b_{i,j}$ to the OT

functionality, and obtains $m_{b_{i,j}}^{(i,j)}$. Finally, $\mathcal{S}$ samples $(L_0^{i,j}, L_1^{i,j})$ uniformly at random, and sets $L_{c_{i,j}}^{i,j}[\alpha_{i,j}] := v_{i,j} \oplus m_{b_{i,j}}^{(i,j)}$. For $j = 1$ to ℓ, $\mathcal{S}$ sends $(L_{c_{i,j}}^{i,j}, L_{\bar{c}_{i,j}}^{i,j})_{i \le N}$ to R_j.

Claim. $\mathcal{H}_2 \approx \mathcal{H}_1$.

Proof. Note that the change in $\mathcal{H}_2$ is purely syntactic, because all values in $L_{\bar{c}_{i,j}}$ and all values $L_{c_{i,j}}[k]$ for $k \neq \alpha_{i,j}$ are uniformly and independently random.

This concludes the proof of sender security.

Receiver security. Conversely, consider a receiver R with key pair $(\mathsf{pk}_R, \mathsf{sk}_R)$ interacting with ℓ corrupted senders $(S_1, \cdots, S_\ell)$ with public keys $(\mathsf{pk}_{S_1}, \cdots, \mathsf{pk}_{S_\ell})$. The receiver has $N \cdot \ell$ selection bits $(b_{i,j})_{i \le N, j \le \ell}$. The senders and the receiver agree on $N \cdot \ell$ uniformly random inputs $(x_{i,j})_{i \le N, j \le \ell}$.

- **Key derivation.** R sets $K_R^j := \mathsf{pkPCF.KeyDer}(R, \mathsf{sk}_R, \mathsf{pk}_{S_j})$, for all $j \in [\ell]$.
- **Oblivious transfers.** For every $i \le N, j \le \ell$, R computes

$$(b'_{i,j}, v_{i,j}, \alpha_{i,j}) := \mathsf{pkPCF.EvalR}(K_R^j, x_{i,j}).$$

  For every $j \le \ell$, the receiver sends $c_{i,j} := b_{i,j} \oplus b'_{i,j}$ to S_j. Upon receiving $(L_0^{i,j}, L_1^{i,j})_{i \le N}$ from S_j, R outputs $m_{i,j} := L_{b_{i,j}}^{i,j}[\alpha_{i,j}] \oplus v_{i,j}$.

The security analysis is straightforward: The simulator samples the bits $b_{i,j}$ uniformly at random, as in the experiment $\mathsf{Exp}_{\mathcal{A},N,1}^{\mathsf{pkRsec}}(\lambda, \ell(\lambda))$ from Figure 3.13. The simulated game is indistinguishable from the honest game by the ℓ-instance receiver security of pkPCF. In this simulated game, $c_{i,j} = b_{i,j} \oplus b'_{i,j}$ perfectly masks $b_{i,j}$, and receiver security follows.

3.10 Precomputability and Two-Round OT Extension

In this section, we discuss additional features offered by our framework and instantiations. In particular, we describe how our framework offers precomputability for either the sender or the receiver, with an efficient "synchronization" protocol using standard building blocks. We additionally discuss several other features offered by our framework, such as two-round OT extension and theoretical instantiations in the standard model.

Tool: Vector Oblivious Linear Evaluation (VOLE). As a building-block for precomputability and two-round OT extension, we first describe Vector OLE [NP06, IPS09] and "reverse" VOLE (reVOLE) [BCG+19b]. We define the ideal functionalities for these primitives in Figure 3.16, when generalized to matrix inputs (this generalization follows immediately from standard VOLE and reVOLE). At a high level, matrix-VOLE allows the receiver to obtain $\mathbf{A} + \mathbf{b}\mathbf{x}^\top$ as output (the sender gets no output), when the sender inputs $(\mathbf{A}, \mathbf{b})$ and the receiver inputs $\mathbf{x}$. In contrast, matrix-reVOLE allows the receiver to obtain $\mathbf{A} + \mathbf{b}\mathbf{x}^\top$ as output, when the sender inputs $(\mathbf{A}, \mathbf{x})$ and the receiver inputs $\mathbf{b}$. Using any matrix-VOLE (or matrix-reVOLE) protocol, we obtain precomputability for the sender and receiver via simple building blocks (VOLE and reVOLE).

Parameters. Finite ring $\mathcal{R}$ and integers $m, n \geq 1$.

Parties. The functionality interacts with a sender S and a receiver R.

Ideal functionality $\mathcal{F}_{\mathsf{VOLE}}$:

1: Wait for input $(\mathbf{A}, \mathbf{b}) \in \mathcal{R}^{m \times n} \times \mathcal{R}^m$ from S.
2: Wait for input $\mathbf{x} \in \mathcal{R}^n$ from R.
3: Compute $\mathbf{C} := \mathbf{A} + \mathbf{b}\mathbf{x}^\top$, and output $\mathbf{C}$ to R and $\perp$ to S.

Ideal functionality $\mathcal{F}_{\mathsf{reVOLE}}$:

1: Wait for input $(\mathbf{A}, \mathbf{x}) \in \mathcal{R}^{m \times n} \times \mathcal{R}^n$ from S.
2: Wait for input $\mathbf{b} \in \mathcal{R}^m$ from R.
3: Compute $\mathbf{C} := \mathbf{A} + \mathbf{b}\mathbf{x}^\top$, and output $\mathbf{C}$ to R and $\perp$ to S.

Figure 3.16: Ideal functionalities for matrix-VOLE ($\mathcal{F}_{\mathsf{VOLE}}$) [NP06, IPS09].

3.10.1 Precomputability

Here, we describe how either the sender or receiver can precompute all their inputs ahead of time, *without needing to know the identity of the other party.* Our definition for precomputability is inspired by the blueprint laid out by Couteau, Meyer, Passelègue, and Riahinia [CMPR23].[14] Moreover, we explain how VOLE or (reVOLE) can be used to efficiently synchronize the parties after one of the parties precomputes their OT messages.

Remark 16. *The definition of precomputability in [CMPR23, Def. 7] has a couple downsides which we address with Definition 3.10.1. In particular, their definition does not rule out a "trivially precomputable" scheme where* KeyGen_0 *outputs both the keys (and the other party's key would be fixed by having it as part of the auxiliary information passed to* KeyGen_1*). In this case, the first party has both keys and we do not get meaningful security for the applications when using this setup. Additionally, their definition provides no formalization of the interactive "synchronization" protocol, which we resolve by defining the ideal functionality used by the parties to agree on common keys (without changing the first party's key) following one party's precomputation.*

Definition 3.10.1 (Precomputable PCF for ListOT)**.** *Let λ be a security parameter and $\lambda \leq n = n(\lambda) \in \mathsf{poly}(\lambda)$ be an input length. Let $\mathsf{PCF} = (\mathsf{KeyGen}, \mathsf{Eval})$ be a PCF for ListOT. We say that* PCF *is* sender precomputable *if there exist efficient algorithms KeyGen_S, $\mathsf{KeyGenRev}_S$, and an interactive key generation protocol KG_R between the sender S and a receiver R, with the following syntax:*

- $\mathsf{KeyGen}_S(1^\lambda) \to K_S$*. The randomized key generation algorithm takes as input the security parameter. It outputs a key K_S.*
- $\mathsf{KeyGenRev}_S(K_S) \to r$*. The deterministic reversal algorithm takes as input a sender's key, and outputs the sampling randomness $r \in \{0,1\}^*$. (This is used to formalize a notion of oblivious sampleability of the key.)*

[14] Note that *simultaneous* precomputability is not possible [CMPR23].

Parties. The functionality interacts with a sender S and a receiver R.

Ideal functionality $\mathcal{F}_{\mathsf{KG},R}$:

1: Wait for input K_S from S.
2: Wait for input ("`KeyGen`") from R.
3: Compute $K_R := \mathsf{KeyDer}_R(K_S)$, where KeyDer_R is an algorithm that computes a fitting receiver's key (i.e., such that the distribution of (K_S, K_R) is identical to the output of $\mathsf{PCF.KeyGen}(1^\lambda)$) from a valid sender's key K_S. Then, output K_R to R and $\perp$ to S.

Figure 3.17: Ideal functionality for the protocol that securely establishes a fitting key for the other party, after one party's key has been generated ahead of time.

- $\mathsf{KG}_R\langle S, R\rangle$ *is an interactive protocol, where* S *has input* K_S *(as output by the first algorithm), and* R *has no input. After the protocol,* R *outputs a key* K_R*, while* S *outputs* $\perp$*.*

Importantly, we have that the keys generated by this process are identically distributed to those generated by $\mathsf{PCF.KeyGen}$*. That is, we first require that the sender's key is* obliviously sampleable *(independently of the receiver's key), a notion that is inspired by Canetti and Fischlin [CF01]. Formally,*

(1) Generating the sender's key via KeyGen_S *is perfectly indistinguishable from the generating it via* $\mathsf{PCF.KeyGen}$*:*

$$\left\{K_S \mid K_S \leftarrow \mathsf{KeyGen}_S(1^\lambda)\right\} \equiv \left\{K_S \mid (K_S, K_R) \leftarrow \mathsf{PCF.KeyGen}(1^\lambda)\right\},$$

and

(2) the algorithm $\mathsf{KeyGenRev}_S$ *returns, for a sender's key* K_S*, the randomness that is used to generate it using* KeyGen_S*:*

$$\left\{(K_S, r) \mid r \xleftarrow{R} \{0,1\}^{\mathsf{poly}(\lambda)}; K_S := \mathsf{KeyGen}_S(1^\lambda; r)\right\} \approx_c \left\{ (K_S, r) \;\middle|\; \begin{array}{r} (K_S, K_R) \leftarrow \mathsf{PCF.KeyGen}(1^\lambda) \\ r \leftarrow \mathsf{KeyGenRev}_S(K_S) \end{array} \right\}.$$

Additionally, we require that KG_R *securely realizes the functionality* $\mathcal{F}_{\mathsf{KG},R}$ *described in Figure 3.17.*

Receiver precomputability. *This is defined in the exact analogous way, where all instances of* S *and* R *are exchanged. Hence, it asks for the existence of* KeyGen_R*,* $\mathsf{KeyGenRev}_R$*, and an interactive key generation protocol* KG_S *that fulfills the respective analogous properties.*

The main motivation of precomputability is that one party can locally generate a key and use it to precompute all evaluations. Then, at a later point in time, another party can use an interactive protocol that securely establishes a key that "fits" to the key that was used by the first party for the precomputation. We now turn to describing how we can achieve

precomputability for either the sender or the receiver.

Sender precomputability. KeyGen_S simply outputs the ShCPRF master key, exactly as in the construction. This gives us sender precomputability for Figure 3.6, because in the construction, we have $K_S = \mathsf{msk} = (\mathbf{k_0}, \mathbf{Z_0})$ (an ShCPRF master key as defined in Figure 3.2) which is sampled independently of the receiver's key K_R. Then, to obtain K_R, the receiver can invoke a matrix-VOLE protocol, defined in Figure 3.16, where the sender inputs $(\mathbf{Z_0}, \Delta)$ while the receiver inputs $-\mathbf{z}$ (note that $\Delta \in \mathcal{R}^m$ and $\mathbf{z} \in \mathcal{R}^n$). The receiver then obtains $\mathbf{Z_1} = \mathbf{Z_0} - \Delta\mathbf{z}^\top$ as output, which is distributed identically to $\mathbf{Z_1}$ in Figure 3.2. Note that $\mathbf{k_0}$, which is common to both msk and csk, can simply be sent over by the sender. As such, the receiver successfully derives $K_R = ((\mathbf{k_0}, \mathbf{Z_1}), \mathbf{z})$, which matches the expected receiver key.

Receiver precomputability. Showing that we also get *receiver* precomputability is slightly more involved. First, note that we can let the receiver sample $\mathbf{k_0}$, $\mathbf{Z_1}$, and $\mathbf{z}$ uniformly, which together form $K_R = ((\mathbf{k_0}, \mathbf{Z_1}), \mathbf{z})$. Therefore, we can define KeyGen_R to output these elements, which are distributed identically to the receiver's key in the construction. Using K_R, the receiver can precompute all the OT messages for Figure 3.7.

Then, the challenge is finding a way for the *sender* (who has Δ) to obtain the "master key" $\mathsf{msk} = (\mathbf{k_0}, \mathbf{Z_0})$ where $\mathbf{Z_0} = \mathbf{Z_1} + \Delta\mathbf{z}^\top$. This can be achieved by invoking a matrix-reVOLE protocol, defined in Figure 3.16, where now the receiver (playing the role of the sender) inputs $(\mathbf{Z_1}, \mathbf{z})$ and the sender (playing the role of the receiver) inputs $\Delta \in \mathcal{R}^m$. The sender obtains as output $\mathbf{Z_0} = \mathbf{Z_1} + \Delta\mathbf{z}^\top$, which is distributed identically to $\mathbf{Z_0}$ in Figure 3.2. Once more, the receiver can simply send $\mathbf{k_0}$ to the sender. This allows the sender to recover $K_S = (\mathbf{k_0}, \mathbf{Z_0})$, as required.

Application: Fast OTs in a client-server model. To motivate the notion of precomputability, consider the following setting: a weak client will want to, at some point in the future, run a secure two-party computation protocol with some servers, the identity of which is yet unknown. During an idle period, the weak client generates its key $K_\sigma \leftarrow \mathsf{KeyGen}_\sigma(1^\lambda)$, for $\sigma \in \{S, R\}$, and precomputes a large number of ListOTs (either list pairs (L_0, L_1) if the client has a sender role, or triples (b, v, α)). When the client decides on a server they want to generate OTs with, a cheap distributed protocol is performed (by our above discussion, this can be done with a single length-n VOLE or reVOLE with our construction) to provide the key $K_{1-\sigma}$ to the server. The powerful server can then quickly compute its share of the ListOTs on the fly, and engage in the two-party computation with the client. In a typical scenario, the client could be a phone; at night, the phone automatically prepares ListOTs. Then, whenever the phone holder decides to browse a website, the phone could enable a two-party computation with the website (e.g., to securely get restaurant recommendations, find matching profiles, or be served with a targeted ad) using only cheap computations on its side (indeed, the bottleneck shifts entirely towards communication).

3.10.2 Two-round setup from two-round OT

In this section, we describe how our framework yields a protocol for two-round OT extension. Since two-round OT is clearly optimal, two-round OT extension provides the best possible performance one could hope for (in terms of rounds and use of symmetric vs. public-key primitives). However, due to the impossibility result of Garg et al. [GMMM18], two-round OT

extension is impossible to instantiate unconditionally in the ROM, and therefore *necessitates* a non-black-box assumption. Here, we show that using any black-box two-round OT protocol, we get a two-round key-derivation protocol for our PCF for ListOT from Figure 3.6. This then allows us to build a two-round OT extension. Coupled with our instantiation of QuietOT in the standard model (see Section 3.10.3), we show that any IPM-wPRF suffices to achieve two-round OT extension and circumvent the impossibility of Garg et al. [GMMM18] using a broad family of Minicrypt primitives.

Lemma 3.10.1 (VOLE and reVOLE from OT)**.** *Two-round OT implies two-round VOLE and reVOLE, as defined in Figure 3.16.*

Proof. Two-round reverse VOLE can be constructed using n parallel calls to a two-round OLE functionality (where the receiver inputs $\mathbf{x} \in \mathcal{R}^n$ coordinate-by-coordinate and the sender inputs $\mathbf{A}$ column-by-column and $\mathbf{b}$). In turn, two-round OLE can be constructed from $O(m \log_2 |\mathcal{R}|)$ parallel calls to a two-round OT functionality using the protocol of Gilboa [Gil99] for ring $\mathcal{R}^m$. The same holds for reVOLE, since reVOLE is trivially implied by OLE [ALSZ15, ADI+17]. ■

Theorem 3.10.1 (Informal)**.** *There exists a two-round key derivation protocol for Figure 3.6 from any two-round OT protocol.*

Proof. The proof follows from the protocols described in Section 3.10.1 and using Lemma 3.10.1. Consider the PCF for ListOT framework of Figure 3.6. As shown in Section 3.10.1, using a reVOLE protocol, the receiver can obtain K_R with one call to the reVOLE functionality by inputting $-\mathbf{z}$ and having the sender input $(\mathbf{Z_0}, \Delta)$. Again, since $\mathbf{k_0}$ is common to both the sender and receiver in Figure 3.6, it can be transmitted separately. By Lemma 3.10.1, this reVOLE functionality can be instantiated using $O(n \cdot m \log_2(|\mathcal{R}|))$ parallel calls to a two-round OT functionality, which implies that the receiver can derive K_R in two rounds. ■

3.10.2.1 Two-round OT extension

Note that, just using $\mathbf{z}$ (the wPRF key) and the random x_i, the receiver can precompute all the OT messages sent to the sender in Figure 3.7, without needing to know csk (recall that $K_R = (\mathsf{csk}, \mathbf{z})$). Specifically, by (1) computing all the bits destined for the sender in Figure 3.7 using $\mathbf{z}$, and (2) executing the two-round key-derivation protocol from Theorem 3.10.1 in parallel with Figure 3.7, the receiver obtains csk and all the information it needs to decode the response messages received from the sender. A little more concretely,

Round 1: Receiver → Sender. In the first round, the receiver, with choice bits $\mathbf{b} = (b_1, \ldots, b_N)$, starts by computing all the bit masks $\mathbf{b}'$ using the wPRF key $\mathbf{z}$ and x and sends $\mathbf{c} = \mathbf{b} \oplus \mathbf{b}'$ to the sender in Figure 3.7. In addition, the receiver also sends the reVOLE message used to derive csk in parallel with its choice bits.

Round 2: Sender → Receiver. The sender computes the lists L_0 and L_1 as in Figure 3.7, then using the masked bits $(c_i \in \mathbf{c})_{i \in [N]}$ received from the receiver, it responds with its reVOLE response message and lists $(L'_{i,0}, L'_{i,1})_{i \in [N]}$. The receiver can then locally (1) reconstruct csk from the reVOLE response message and (2) recover the messages exactly as in Figure 3.7.

3.10.3 Instantiations of QuietOT in the standard model

An interesting feature of our PCF for ListOT framework (Figure 3.6) is that security (for the sender) reduces entirely to the ShCPRF, as seen in the proof of Theorem 3.5.1. While the simplest instantiation of the ShCPRF framework (Section 3.5) is using a random oracle, *any* suitable RKA-secure PRF suffices. In particular, as was shown in Chapter 2, we can use the VDLPN wPRF candidate of Boyle et al. [BCG+20a] to instantiate a (weak) CPRF (in turn giving us a weak ShCPRF by extension). Coupled with the work of Bui et al. [BCM+24], which shows that the VDLPN wPRF candidate is actually an IPM-wPRF, we can instantiate QuietOT solely based on the VDLPN assumption. Of course, using alternative CPRF constructions supporting inner-product predicates based on DDH (cf. Chapter 2), DCR [CMPR23], or LWE [DKN+20] is also an option. However, while such instantiations are interesting when viewed from a theoretical lens, they do not lead to practical constructions given their "public-key" nature.

Remark 17. *We note that Applebaum, Harnik, and Ishai [AHI11] have shown that it is possible to instantiate the IKNP OT extension protocol assuming RKA-secure PRFs, which coupled with our framework, makes studying the relationship between RKA-security and OT extension an interesting direction for future work.*

Remark 18 (Generating random inputs)**.** *In practice, the inputs to the PCF (which are used as inputs to the IPM-wPRF) need to be uniformly random. The random oracle model immediately implies a common random string available to both parties to use as inputs. However, if we replace the random oracle with an RKA-secure PRF, then the parties need a different way to obtain uniformly random inputs. One idea is to settle for* pseudorandom *inputs and have both parties obtain a common seed for a PRG (or alternatively obtain a PRF key), which they can expand into many (pseudo)random inputs $x_1, \ldots, x_N$ to generate N correlations. Unfortunately, such an approach is only heuristically secure in the general case, since there exist counter-examples to the security of wPRFs when evaluated using (public) pseudorandomness [PS08]. However, for the BIPSW and VDLPN wPRF candidates, security is believed to hold even when evaluated using public pseudorandomness, as shown in a recent work of Brzuska et al. [BCE+24].*

3.11 Deferred Proofs

3.11.1 Proof of Theorem 3.4.1

Proof. We prove correctness, security, and pseudorandomness in turn.

Correctness. Consider a constraint $\mathbf{z} \in \mathcal{R}^n$ and input $\mathbf{x} \in \mathcal{R}^n$ such that $\langle \mathbf{z}, \mathbf{x} \rangle = 0$ (i.e, the constraint is satisfied). It holds that $\mathbf{k} = \mathbf{k}_0 + \mathbf{Z_0 x} = \mathbf{k}_0 + \mathbf{Z_1 x} + (\Delta \mathbf{z}^\top)\mathbf{x} = \mathbf{k}_0 + \mathbf{Z_1 x}$. Therefore, the resulting $\mathbf{k}$ is identical in Eval and CEval of Figure 3.2 for the same input $\mathbf{x}$. Correctness then follows from the correctness of F. For shiftability correctness, for all $\langle \mathbf{z}, \mathbf{x} \rangle - \alpha \neq 0$, using shift α, then $\mathbf{k} = \mathbf{Z_0 x} - \Delta\alpha = \mathbf{k} = \mathbf{Z_0 x} - \Delta\alpha + \Delta \langle \mathbf{z}, x \rangle - \Delta \langle \mathbf{z}, \mathbf{x} \rangle = \mathbf{Z_1 x} + \Delta \langle \mathbf{z}, \mathbf{x} \rangle - \Delta\alpha$. Therefore, when $\alpha = \langle \mathbf{z}, \mathbf{x} \rangle$, the resulting $\mathbf{k}$ is identical in Eval and CEval of Figure 3.2 for the same input $\mathbf{x}$ and correctness follows.

(1-key, selective) Security. We prove security by a reduction to the RKA-security of $\mathcal{F}$. Our proof consists of a sequence of hybrid games.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the (1-key, selective) ShCPRF security game defined in Definition 3.4.1.

- *Hybrid* $\mathcal{H}_1$. In this hybrid, during setup, the challenger first samples the constrained key and *then* samples the master key. Specifically, at the start of the game, given the constraint $\mathbf{z} \in \mathcal{R}^n$, the challenger first samples a constrained key $\mathsf{csk} := (\mathbf{k_0}, \mathbf{Z_1})$, where $\mathbf{k_0} \xleftarrow{R} \mathcal{R}^m$ and $\mathbf{Z_1} \xleftarrow{R} \mathcal{R}^{m \times n}$. Then, the challenger computes the master secret key as $\mathsf{msk} := (\mathbf{k_0}, \mathbf{Z_0}, \Delta)$, where $\Delta \xleftarrow{R} \mathcal{R}^m$, $\mathbf{Z_0} := \mathbf{Z_1} + \Delta \mathbf{z}^\top$ and $\mathbf{k}_0$ is as in csk. The difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is purely syntactic. In particular, it follows that the distribution of msk and csk in $\mathcal{H}_1$ is identical to $\mathcal{H}_0$.

- *Hybrid* $\mathcal{H}_2$. In this hybrid game, the challenger does not sample Δ anymore. Instead, the challenger is given access to the following stateful oracle $\mathcal{O}_{\mathsf{rka}}$:

  Oracle $\mathcal{O}_{\mathsf{rka}}$

  Initialize. Sample $\Delta \xleftarrow{R} \mathcal{R}^m$.

  Evaluation. On input $\phi \in \Phi_{\mathsf{aff}}$ and $\mathbf{x} \in \mathcal{R}^n$, return $F_{\phi(\Delta)}(\mathbf{x})$.

  The challenger is then defined as follows.

  1. **Setup:** On input 1^λ, the challenger
     - runs $\mathcal{A}(1^\lambda)$ who outputs a constraint $\mathbf{z}$;
     - samples csk according to $\mathcal{H}_1$ by sampling $\mathbf{k_0} \xleftarrow{R} \mathcal{R}^m$, $\mathbf{Z_1} \xleftarrow{R} \mathcal{R}^{m \times n}$;
     - samples a uniformly random function $R \xleftarrow{R} \widetilde{\mathcal{F}}_\lambda$, where $\widetilde{\mathcal{F}}_\lambda$ is the set of all functions with domain $\mathcal{X} \times \mathcal{S}$ and range $\mathcal{Y}$; and
     - sends csk to $\mathcal{A}$.
  2. **Evaluation queries:** For each query $(\mathbf{x}, \alpha)$ from $\mathcal{A}$, if $C_{\mathbf{z}}(x, \alpha) = 0$, then the challenger responds with $\perp$. Otherwise, the challenger proceeds as follows:
     - If $b = 0$, it computes $a := \langle \mathbf{z}, \mathbf{x} \rangle - \alpha$ and $\mathbf{b} := \mathbf{k_0} + \mathbf{Z_1}\mathbf{x}$, defines the affine function $\phi\colon \mathbf{u} \mapsto a\mathbf{u} + \mathbf{b}$, queries $\mathcal{O}_{\mathsf{rka}}$ on input $(\phi, \mathbf{x})$, and forwards the response y to $\mathcal{A}$.

       ▷ We note that y is computed by $\mathcal{O}_{\mathsf{rka}}$ as $F_{\mathbf{k}'}(\mathbf{x})$ where
       ▷ $\mathbf{k}' = a\Delta + \mathbf{b} \in \mathcal{R}^m = (\langle \mathbf{z}, \mathbf{x} \rangle - \alpha)\Delta + \mathbf{b} = \phi(\Delta)$, for some $\phi \in \Phi_{\mathsf{aff}}$.
     - If $b = 1$, it computes $y := R(x, \alpha)$ and returns y,

  Claim. *$\mathcal{A}$'s advantage in $\mathcal{H}_2$ is identical to $\mathcal{A}$'s advantage in $\mathcal{H}_1$.*

Proof. The difference between $\mathcal{H}_2$ and $\mathcal{H}_1$ is again purely syntactic since each output is computed identically in both games. However, note that the challenger now only has access to Δ via the oracle $\mathcal{O}_{\mathsf{rka}}$. □

Claim. *There does not exist an efficient $\mathcal{A}$ with greater than negligible advantage in $\mathcal{H}_2$ assuming $\mathcal{F}$ is an RKA-secure PRF with respect to affine related key derivation functions Φ_{aff}.*

Proof. Note that the challenger in $\mathcal{H}_2$ is already playing the role of a Φ_{aff}-restricted adversary when querying the oracle $\mathcal{O}_{\mathsf{rka}}$ to answer the evaluation queries. The reduction to RKA security of $\mathcal{F}$ is therefore immediate. □

This concludes the proof of (1-key, selective) security.

Pseudorandomness. We prove the pseudorandomness property by a reduction to the RKA-security of $\mathcal{F}$. Our proof consists of a sequence of hybrid games.

- *Hybrid $\mathcal{H}_0$.* In this hybrid, the adversary is given oracle access to $\mathsf{Eval}(\mathsf{msk}, \cdot, \cdot)$.
- *Hybrid $\mathcal{H}_1$.* In this hybrid, the reduction emulates the answers of the oracle queries of $\mathcal{A}$ as follows. It samples $\mathbf{Z}_0 \xleftarrow{\mathrm{R}} \mathcal{R}^{m \times n}$ and $\Delta \xleftarrow{\mathrm{R}} \mathcal{R}^m \setminus \{0\}$. In addition, the reduction interacts with the following oracle $\mathcal{O}_{\mathsf{rka}}$:

  Oracle $\mathcal{O}_{\mathsf{rka}}$

  Initialize. Sample $\mathbf{k}_0 \xleftarrow{\mathrm{R}} \mathcal{R}^m$.

  Evaluation. On input $\phi \in \Phi_{\mathsf{aff}}$ and $\mathbf{x} \in \mathcal{R}^n$, return $F_{\phi(\mathbf{k}_0)}(\mathbf{x})$.

  Given a query $(\mathbf{x}, \alpha)$, the reduction defines $\phi_\alpha : \mathbf{k} \mapsto \mathbf{k} + \mathbf{Z}_0\mathbf{x} - \Delta \cdot \alpha$, and queries $\mathcal{O}_{\mathsf{rka}}$ on input $(\phi_\alpha, \mathbf{x})$, and forwards the response to $\mathcal{A}$. Observe that the answers to $\mathcal{A}$'s queries in $\mathcal{H}_1$ are always equal to $\mathsf{Eval}(\mathsf{msk}, \mathbf{x}, \alpha)$ for $\mathsf{msk} := (\mathbf{k}_0, \mathbf{Z}_0, \Delta)$, hence $\mathcal{A}$'s advantage in $\mathcal{H}_1$ is identical to $\mathcal{A}$'s advantage in $\mathcal{H}_0$.
- *Hybrid $\mathcal{H}_2$.* In this hybrid, the answer of $\mathcal{O}_{\mathsf{rka}}$ on a query $(\mathbf{x}, \alpha)$ is computed as $R(\phi_\alpha, \mathbf{x})$, where R is a uniformly random function from the set $\widetilde{\mathcal{F}}_\lambda$ of all functions from $\Phi_{\mathsf{aff}} \times \mathcal{S}$ to $\mathcal{Y}$. By the RKA-security of the PRF family, $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable.
- *Hybrid $\mathcal{H}_3$.* In this hybrid, we sample a uniformly random function $R \xleftarrow{\mathrm{R}} \widetilde{\mathcal{F}}_\lambda$, where $\widetilde{\mathcal{F}}_\lambda$ is the set of all functions from $\mathcal{R}^n \times \mathcal{S}$ to $\mathcal{Y}$, and all queries of $\mathcal{A}$ are answered with R. Observe that for any two queries $(\mathbf{x}_0, \alpha_0)$ and $(\mathbf{x}_1, \alpha_1)$, it holds that $(\phi_{\alpha_0}, \mathbf{x}_0) = (\phi_{\alpha_1}, \mathbf{x}_1)$ iff $(\mathbf{x}_0, \alpha_0) = (\mathbf{x}_1, \alpha_1)$, hence $\mathcal{H}_3$ is perfectly indistinguishable from $\mathcal{H}_2$.

This concludes the proof of pseudorandomness and the proof of Theorem 3.4.1. ■

3.11.2 Proof of Theorem 3.5.1

Proof. We prove pseudorandomness, correctness, sender security, and receiver security in turn.

Pseudorandomness. We prove pseudorandomness via the following two hybrid games.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)$ from Figure 3.3, where $\mathsf{PCF.KeyGen}$ and $\mathsf{PCF.Eval}$ are as defined in Figure 3.6 (PCF for ListOT framework).

- *Hybrid* $\mathcal{H}_1$. In this hybrid game, we replace each pseudorandom bit b_i computed in $\mathcal{H}_0$ using the IPM-wPRF f, with truly random bits.

  Claim. $\mathcal{H}_0 \approx_c \mathcal{H}_1$.

  Proof. Suppose, towards a contradiction, that there exists an efficient $\mathcal{A}$ that distinguishes between $\mathcal{H}_0$ and $\mathcal{H}_1$ with non-negligible advantage $\nu(\lambda)$. Since the only difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is that the pseudorandom bits in $\mathcal{H}_0$ are replaced with uniformly random bits in $\mathcal{H}_1$, the reduction to the wPRF pseudorandomness of f is immediate. (For this, note that $\mathbf{z}$ is independent of msk.) □

- *Hybrid* $\mathcal{H}_2$. In this hybrid, for all $i \in [N(\lambda)]$, the lists L_0^i, L_1^i are sampled uniformly from $\mathcal{D}_{\mathcal{Y}}^{\text{list}}(S_0)$ and $\mathcal{D}_{\mathcal{Y}}^{\text{list}}(S_1)$, respectively, where the distribution $\mathcal{D}_{\mathcal{Y}}^{\text{list}}(\cdot)$ is defined in Definition 3.5.1 and consists of uniformly random samples from $\mathcal{Y}$.

  Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$.

  Proof. Suppose, towards a contradiction, that there exists an efficient $\mathcal{A}$ that distinguishes between $\mathcal{H}_2$ and $\mathcal{H}_1$ with non-negligible advantage $\nu(\lambda)$. We can then construct an efficient $\mathcal{B}$ that contradicts the pseudorandomness property of the ShCPRF (Definition 3.4.1). Note that in Figure 3.6, the list entries of L_0^i, L_1^i are sampled as

  $$s_0^i := \mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}_i, \alpha_0) \text{ and } s_1^i := \mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}_i, \alpha_1),$$

  for all $\alpha_0 \in S_0$ and $\alpha_1 \in S_1$, where $\mathbf{x}_i := \mathsf{map}(x_i) \in \mathcal{R}^n$, and then assembled into the two lists L_0^i, L_1^i.

  Given oracle access to $\mathcal{O}$, which is either a random function $R(\cdot,\cdot)$ or the algorithm $\mathsf{ShCPRF.Eval}(\mathsf{msk}, \cdot, \cdot)$, we construct $\mathcal{B}$ as follows:

  1. For all $i \in [N(\lambda)]$,
     - sample $x_i \xleftarrow{R} \mathcal{X}_\lambda$ and set $\mathbf{x}_i := \mathsf{map}(x_i) \in \mathcal{R}^n$,
     - query $(\mathbf{x}_i, \alpha_0)$ to the oracle on all $\alpha_0 \in S_0$ to get response s_{α_0},
     - query $(\mathbf{x}_i, \alpha_1)$ to the oracle on all $\alpha_1 \in S_1$ to get response s_{α_1},
     - sample bit b_i uniformly at random.

2. Then, assemble the lists L_0^i, L_1^i and run $\mathcal{A}$ on input $(1^\lambda, (x_i, L_0^i, L_1^i, b_i)_{i\in[N(\lambda)]})$ and output as it does.

Observe that the lists $(L_0^i, L_1^i)_{i\in[N(\lambda)]}$ output by $\mathcal{B}$ are distributed identically to the lists in Figure 3.6 if $\mathcal{B}$ is given oracle access to the ShCPRF, and distributed as uniformly random lists when $\mathcal{B}$ is given oracle access to a random function. Therefore, the distribution given to $\mathcal{A}$ is identical to $\mathcal{H}_1$ or $\mathcal{H}_2$, allowing $\mathcal{B}$ to win the pseudorandomness game of the ShCPRF with the same advantage. □

We observe that $\mathcal{H}_2$ is already identical to $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)$, which concludes the proof of pseudorandomness.

Correctness. Observe that v, as output by evaluating $\mathsf{PCF.EvalR}(K_R, x)$, is computed as $v := \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x})$, where $\mathbf{x} := \mathsf{map}(x)$ and $\mathsf{csk} \leftarrow \mathsf{ShCPRF.Constrain}(\mathsf{msk}, \mathbf{z})$ for a random constraint $\mathbf{z}$. Hence, by correctness of the ShCPRF, we know that there exists a shift $\alpha \in S_0 \cup S_1$, such that $\mathbf{x}$ and α are authorized. More specifically, $C(\mathbf{x}, \alpha) := \langle \mathbf{z}, \mathbf{x} \rangle - \alpha = 0$, and with overwhelming probability, v is equal to the list entry calculated via $\mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}, \alpha)$. (Note that α is calculated in PCF.EvalR exactly in this way.) Hence, for the (unique) $b' \in \{0, 1\}$ with $\alpha \in S_{b'}$, we have that, with overwhelming probability, $L_{b'}[\alpha] = v$. Finally, $b' = b$ by the property of the IPM-wPRF which guarantees that $b := f_{\mathbf{z}}(\mathbf{x}) = 0$ iff $\langle \mathbf{z}, \mathbf{x} \rangle \in S_0$ and $f_{\mathbf{z}}(\mathbf{x}) = 1$ iff $\langle \mathbf{z}, \mathbf{x} \rangle \in S_1$.

Sender Security. Informally, this follows from the fact that by the shiftable CPRF security, for any (possibly not uniform)[15] $x_i \xleftarrow{\text{R}} \mathcal{X}$ and given only csk for a constraint $\mathbf{z}$, all constrained values are pseudorandom to the adversary. However, for the given $\mathbf{x}_i := \mathsf{map}(x_i)$, it is authorized only exactly for this constraint $\mathbf{z}$ with shift α_i (because $S_0 \cup S_1 = \mathcal{R}$ and $\alpha_i = \langle \mathbf{z}, \mathbf{x} \rangle \in \mathcal{R}$ is the unique value such that $\langle \mathbf{z}, \mathbf{x} \rangle - \alpha_i = 0$). More formally, we have a sequence of hybrids.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the $\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,0}(\lambda)$ experiment defined in Figure 3.4, where $\mathsf{PCF} = (\mathsf{KeyGen}, \mathsf{EvalS}, \mathsf{EvalR})$ are as defined in Figure 3.6 (PCF for ListOT framework). In particular, we note that PCF.EvalS internally runs the ShCPRF $\mathsf{ShCPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$.

- *Hybrid* $\mathcal{H}_1$. In this hybrid, for each $i \in [N(\lambda)]$, the call to $\mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}, \alpha)$ inside of PCF.EvalS is replaced with a call to $\mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x})$ whenever $(\mathbf{x}, \alpha)$ is authorized (i.e., $\langle \mathbf{z}, \mathbf{x} \rangle - \alpha = 0$), where csk is part of K_R in $\mathsf{Exp}^{\mathsf{Ssec}}_{\mathcal{A},N,0}(\lambda)$.

  Claim. $\mathcal{H}_0 \approx_s \mathcal{H}_1$.

  Proof. By the correctness of shiftable CPRFs, we have that for all authorized $(\mathbf{x}_i, \alpha_i)$ pairs,

  $$\mathsf{ShCPRF.Eval}(\mathsf{msk}, \mathbf{x}_i, \alpha_i) = \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x}_i),$$

  with overwhelming probability. □

[15] For example, in the GAR instantiation, the uniformly random input is mapped to a non-uniform vector in the ring.

- *Hybrid* $\mathcal{H}_2$. In this hybrid, the lists L_0^i, L_1^i are sampled uniformly at random from $\mathcal{D}_{\mathcal{Y}}^{\text{list}}(S_0), \mathcal{D}_{\mathcal{Y}}^{\text{list}}(S_1)$. Then, for each i where $(\mathbf{x}_i, \alpha_i)$ is an authorized pair, find the $b_i \in \{0,1\}$, such that $\alpha_i \in S_{b_i}$, and overwrite $L_{b_i}[\alpha_i] := \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{x}_i)$.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_2$.

  Proof. The claim follows directly from the security of the shiftable CPRF. Namely, let $\mathcal{A}$ be an efficient adversary with a non-negligible advantage of distinguishing between $\mathcal{H}_1$ and $\mathcal{H}_2$. Then, we define the following adversary $\mathcal{B}$ to the (1-key, selective) security experiment $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{shcprf}}(\lambda)$ for ShCPRF. When $\mathcal{B}$ is queried for a constraint, it samples a random $\mathbf{z} \xleftarrow{R} \mathcal{R}^n$ (or via some distribution over $\mathcal{R}^n$) and outputs it. When $\mathcal{B}$ is then run with a key csk, it sets $K_R := (\mathsf{csk}, \mathbf{z})$. Then, for each $i \in [N(\lambda)]$, it samples an $x_i \xleftarrow{R} \mathcal{X}$, maps it via $\mathbf{x}_i := \mathsf{map}(x_i)$, and computes the corresponding authorized shift, denoted by α_i (recall that there always exists an efficiently computable shift, by the correctness property of ShCPRF). Then, for all $b \in \{0,1\}$ and all $\alpha \in S_b \setminus \{\alpha_i\}$, it calls its evaluation oracle with $\mathbf{x}_i$ and shift α, and receives response $y_{i,\alpha}$. For the remaining authorized entry pair $(\mathbf{x}_i, \alpha_i)$ it computes $y_{i,\alpha_i} := \mathsf{CEval}(\mathsf{csk}, \mathbf{x}_i)$, as in the previous hybrid. It then assembles these entries into the two lists L_0^i, L_1^i, according to whether the respective shift belongs to S_0 or S_1, and sends K_R and (x_i, L_0^i, L_1^i) to $\mathcal{A}$. Finally, $\mathcal{B}$ outputs what $\mathcal{A}$ outputs.

  Observe that $\mathcal{B}$ is an efficient algorithm, making $N \cdot (|\mathcal{R}| - 1)$ oracle queries. (Note that for each query issued by the adversary, $\mathcal{B}$ needs to perform $|\mathcal{R}| - 1$ queries to the shiftable CPRF oracle.) Moreover, if $\mathcal{B}$ is given inputs from the real game $\mathsf{Exp}_{\mathcal{B},0}^{\mathsf{shcprf}}(\lambda)$, then this perfectly simulates hybrid $\mathcal{H}_1$ for $\mathcal{A}$, and if it is given inputs from the ideal game $\mathsf{Exp}_{\mathcal{B},1}^{\mathsf{shcprf}}(\lambda)$, then this perfectly simulates hybrid $\mathcal{H}_2$ for $\mathcal{A}$. As such, $\mathcal{B}$ has the same advantage as $\mathcal{A}$. □

  Notice that $\mathcal{H}_2$ is distributed identically to $\mathsf{Exp}_{\mathcal{A},N,1}^{\mathsf{Ssec}}(\lambda)$, the experiment defined in Figure 3.4, which concludes the proof of sender security.

Receiver Security. Receiver security follows from the fact that $K_S = \mathsf{msk}$ and x_i are independent of the IPM-wPRF key $\mathbf{z}$, the x_i are sampled uniformly at random, and because f is a wPRF with range $\{0,1\}$. Formally, we prove receiver security via a sequence of hybrids.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the $\mathsf{Exp}_{\mathcal{A},N,0}^{\mathsf{Rsec}}(\lambda)$ experiment defined in Figure 3.5, where $\mathsf{PCF} = (\mathsf{KeyGen}, \mathsf{EvalS}, \mathsf{EvalR})$ are as defined in Figure 3.6.

- *Hybrid* $\mathcal{H}_1$. In this hybrid, we sample a uniformly random function R from the set of all functions from $\mathcal{X}_\lambda$ to $\{0,1\}$, and generate $b_i := R(x_i)$.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$.

  Proof. The only difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is that $b_i = f_{\mathbf{z}}(\mathsf{map}(x_i))$ in $\mathcal{H}_0$, and $b_i = R(x_i)$ in $\mathcal{H}_1$. As the x_i's are uniformly random (and $K_S = \mathsf{msk}$ is generated independently of $\mathbf{z}$), any distinguisher between $\mathcal{H}_0$ and $\mathcal{H}_1$ immediately yields a distinguisher against the pseudorandomness of $f_{\mathbf{z}}$. □

- *Hybrid* $\mathcal{H}_1$. In this hybrid, each bit b_i is sampled uniformly at random: $b_i \xleftarrow{R} \{0,1\}$. Note that this hybrid is exactly $\mathsf{Exp}^{\mathsf{Rsec}}_{\mathcal{A},N,1}(\lambda)$.

  Claim. $\mathcal{H}_2 \approx_s \mathcal{H}_1$.

  Proof. Since R is a truly random function, $\mathcal{H}_2$ and $\mathcal{H}_1$ are perfectly indistinguishable conditioned on all x_i's being distinct. By a straightforward union bound, since all x_i's are sampled randomly from $\mathcal{X}$, the condition is satisfied except with probability at most $N^2/|\mathcal{X}_\lambda|$, which is negligible in λ because $|\mathcal{X}_\lambda|$ is exponential in λ (we require this for wPRF security anyway). □

This concludes the proof of receiver security and the proof of Theorem 3.5.1.

■

3.11.3 Proof of Theorem 3.7.1

Proof. We prove each property in turn.

Correctness. Correctness follows directly from the proof of Theorem 3.4.1 (correctness proof of the non-updatable ShCPRF, cf. Section 3.11.1).

Updatable Correctness. Consider an arbitrary $\mathsf{csk} := (\mathbf{k'_0}, \mathbf{Z_1}) \in \mathcal{R}^m \times \mathcal{R}^{m\times n}$, a constraint $\mathbf{z} \in \mathcal{R}^n$, and an input $\mathbf{x} \in \mathcal{R}^n$. Let $\alpha := \langle \mathbf{z}, \mathbf{x} \rangle$. Given $\mathsf{ltsk} := \Delta \xleftarrow{R} \mathcal{R}^m$, we have $\mathsf{msk}' = (\mathsf{ltsk}, \mathsf{esk}')$ with $\mathsf{esk}' = (\mathbf{Z'_0} := \mathbf{Z_1} + \Delta\mathbf{z}^\top, \mathbf{k'_0})$. Then, denoting $\mathbf{k}_\alpha := \mathbf{k'_0} + \mathbf{Z'_0}\mathbf{x} - \Delta\alpha$, it holds that $\mathbf{k}_\alpha = \mathbf{k'_0} + (\mathbf{Z_1} + \Delta\mathbf{z}^\top)\mathbf{x} - \Delta\alpha = \mathbf{k'_0} + \mathbf{Z_1}\mathbf{x} + \cancel{\Delta(\langle \mathbf{z}, \mathbf{x} \rangle - \alpha)} = \mathbf{k'_0} + \mathbf{Z_1}\mathbf{x}$. Therefore, $F_{\mathbf{k}_\alpha}(\mathbf{x}) = \mathsf{ShCPRF.CEval}(\mathsf{csk}, \mathbf{z}, \mathbf{x})$.

(1-key, selective, ℓ-instance) Updatable Security. We prove security by a reduction to the RKA-security of $\mathcal{F}$. Our proof consists of a sequence of hybrid games.

- *Hybrid* $\mathcal{H}_0$. This hybrid game consists of the (1-key, selective, ℓ-instance) updatable security game. Specifically, at the start of the game, the challenger is given the constraints $\mathbf{z}_i$ and the constrained keys $\mathsf{csk}_i := (\mathbf{k_0}_i, \mathbf{Z_1}_i)$, where $\mathbf{k_0}_i \in \mathcal{R}^m$ and $\mathbf{Z_1}_i \in \mathcal{R}^{m\times n}$ for all $i \in [\ell]$. Then, the challenger computes the updated master secret key as $\mathsf{msk}'_i := (\mathsf{ltsk} := \Delta, \mathsf{esk}_i := (\mathbf{k_0}_i, \mathbf{Z_0}_i))$, where $\mathbf{Z_0}_i := \mathbf{Z_1}_i + \Delta\mathbf{z}_i^\top$ with $\Delta \xleftarrow{R} \mathcal{R}^m$.

- *Hybrid* $\mathcal{H}_1$. In this hybrid, we modify the updatable security game $\mathcal{H}_0$ as follows: given $(\mathbf{z}_i, \mathsf{csk}_i)_{i\le\ell}$, the challenger does not sample Δ anymore and instead interacts with the oracle $\mathcal{O}_{\mathsf{rka}}$ from the proof of Theorem 3.4.1. Similarly to the proof of Theorem 3.4.1, for evaluation query $(\mathbf{x}, \alpha, i)$ from $\mathcal{A}$ with $C_{\mathbf{z}_i}(\mathbf{x}, \alpha) \neq 0$, the challenger computes $a := \langle \mathbf{z}_i, \mathbf{x} \rangle - \alpha$ and $\mathbf{b} := \mathbf{k_0}_i + \mathbf{Z_1}_i\mathbf{x}$.

  - If $b = 0$, the challenger queries $\mathcal{O}_{\mathsf{rka}}$ on input $(\phi, \mathbf{x})$, where $\phi\colon \mathbf{u} \mapsto a\mathbf{u} + \mathbf{b}$ and forwards the response to $\mathcal{A}$.
  - If $b = 1$, the challenger returns $y := R(i, \mathbf{x}, \alpha)$.

Observe that for each query $(\mathbf{x}, \alpha, i)$, it holds that $\mathbf{k}_\alpha = \mathbf{k_0}_i + \mathbf{Z_1}_i\mathbf{x} + \Delta(\langle \mathbf{z}_i, \mathbf{x}\rangle - \alpha)$ is equal to $\phi(\mathbf{x}) = a\mathbf{x} + \mathbf{b}$. Hence, the answers of the challenger to all queries issued by $\mathcal{A}$ in $\mathcal{H}_1$ are computed identically to $\mathcal{H}_0$.

- *Hybrid $\mathcal{H}_2$.* This hybrid game consists of the RKA security game for $\mathcal{F}$ with respect to affine related key derivation functions Φ_{aff}.

  Claim. *If there exists an efficient adversary $\mathcal{A}$ for $\mathcal{H}_1$ that wins with non-negligible advantage, then there exists an efficient Φ_{aff}-restricted adversary $\mathcal{B}$ that wins the $\mathcal{H}_2$ game (RKA security game) with the same advantage as $\mathcal{A}$.*

  Proof. The challenger in $\mathcal{H}_1$ is already playing the role of a Φ_{aff}-restricted adversary when querying the oracle $\mathcal{O}_{\mathsf{rka}}$ to answer the evaluation queries. The reduction to RKA security of $\mathcal{F}$ is therefore immediate. □

Pseudorandomness. The proof of pseudorandomness follows directly from the proof of Theorem 3.4.1.

This concludes the proof of Theorem 3.7.1. ■

3.11.4 Proof of Theorem 3.7.3

Generation phase. For every corrupted sender S_k, the simulator reconstructs $\mathsf{ltsk}_k := \Delta \in \mathcal{R}^m$ (using their random tape) and sends (ltsk_k, k) on their behalf to $\mathcal{F}_{\mathsf{PKS}}$. For every corrupted receiver R_l, the simulator reconstructs the constraint $C_l := \mathbf{z}$ (using their random tape) and sends (C_l, l) to $\mathcal{F}_{\mathsf{PKS}}$ on their behalf. We now emulate each key derivation phase between a sender $S := S_k$ and a receiver $R := R_l$.

Case 1: Both parties are honest. For each $i \in [m]$, the sender computes $z_0^i := \lceil \langle \mathsf{pk}_R, (\Delta_i, s_0^i)\rangle \rfloor_t \in \mathcal{P}$, the receiver computes $z_1^i := \left\lceil \mathsf{pk}_S^i \cdot s_1 \right\rfloor_t \in \mathcal{P}$ and they each parse the first n coefficients of their respective polynomial as vectors $\mathbf{z}_0^i, \mathbf{z}_1^i \in \mathbb{Z}_t^n$. Observe that $\langle \mathsf{pk}_R, (\Delta_i, s_0^i)\rangle$ and $\mathsf{pk}_S^i \cdot s_1$ are in fact noisy additive shares of $\Delta_i \cdot z$ over $\mathcal{P}$, since we have that

$$
\begin{aligned}
&\left\langle \mathsf{pk}_R, \left(\Delta_i, s_0^i\right)\right\rangle - \left(\mathsf{pk}_S^i \cdot s_1\right) \\
&= \Delta_i \cdot z + \Delta_i \cdot a_0 s_1 + \Delta_i \cdot e_1 + s_0^i a_1 s_1 + s_0^i e_1' - \Delta_i \cdot a_0 s_1 - s_0^i a_1 s_1 - e_0^i s_1 \\
&= \Delta_i \cdot z + \underbrace{\Delta_i \cdot e_1 + s_0^i e_1' - e_0^i s_1}_{\text{noise}} \approx \Delta_i \cdot z.
\end{aligned}
$$

Recall that the coefficients of z are $(q/t) \cdot (\mathbf{z} \| 0^{\eta - n})$, while $\Delta_i \in \mathbb{Z}_t$ (where $t \ll q$), and $e_1, s_0^i, e_1', e_0^i, s_1$ are all drawn from χ, which we take to be a discrete Gaussian with standard deviation σ. By basic concentration inequalities, we have that $B = 3 \cdot (8\sigma)^2 \cdot \eta$ is a bound on the magnitude of $\Delta \cdot e_1 + s_0^i e_1' - e_0^i s_1$ with overwhelming probability. Further, note that by the normal form of RLWE, the public keys pk_R and pk_S are pseudorandom, so the shares $\langle \mathsf{pk}_R, (\Delta_i, s_0^i)\rangle$ and $\mathsf{pk}_S^i \cdot s_1$ are as well. Hence, by the rounding lemma (Lemma 3.3.1), since

we set $q = t \cdot B \cdot n \cdot m \cdot 2^{40}$, the probability that a single component is rounded incorrectly is at most $\frac{1}{nm} \cdot 2^{-40}$. Thus, by the union bound over all $n \cdot m$ components of the shares, for all $i \in [m]$ we have that (after parsing the ring elements) $\mathbf{z}_0^i - \mathbf{z}_1^i = \Delta_i \cdot \mathbf{z} \in \mathbb{Z}_t^n$, with very high probability.

Case 2: Sender S is corrupted. The view of the corrupted sender is $(a_0, a_1, z + s_1 a_0 + e_1, s_1 a_1 + e_1')$, where $a_0, a_1 \xleftarrow{R} \mathcal{P}$ and $s_1, e_1, e_1' \xleftarrow{R} \chi$. By the normal form of RLWE we have that $(a_0, a_1, s_1 a_0 + e_1, s_1 a_1 + e_1') \approx_c (a_0, a_1, u_0, u_1)$, where $u_0, u_1 \xleftarrow{R} \mathcal{P}$. Hence, the simulator emulates the view of the sender using a random $\mathsf{pk}_R := (u_0, u_1) \xleftarrow{R} \mathcal{P}^2$. Eventually, the simulator recovers $\mathsf{sk}_S = (\Delta, s_0^1, \ldots, s_0^m)$, computes $\mathsf{esk} := (\mathbf{k}_0, \mathbf{Z}_0) := \mathsf{KeyDer}(S, \mathsf{sk}_S, \mathsf{pk}_R)$ and sends esk on behalf of the ideal adversary to the functionality. The output of R in the ideal world is equal to $(\mathbf{k}_0, \mathbf{Z}_0 - \Delta \mathbf{z}^\top)$, which is identical to R's output in the real world (by the same analysis as for Case 1).

Case 3: Receiver R is corrupted. In each key derivation phase with a sender, the view of the corrupted receiver is $(a_0, a_1, \mathbf{k}_0, (\Delta_i \cdot a_0 + s_0^i + e_0^i)_{i \in [m]})$, where $a_0, a_1 \xleftarrow{R} \mathcal{P}, \mathbf{k}_0 \xleftarrow{R} \mathcal{R}^m$, and for each $i \in [m]$, $s_0^i, e_0^i \xleftarrow{R} \chi$. By the normal form RLWE assumption, this is computationally indistinguishable from $(a_0, a_1, \mathbf{k}_0, (u^i)_{i \in [m]})$, where $a_0, a_1 \xleftarrow{R} \mathcal{P}$ and for each $i \in [m]$, $u_i \xleftarrow{R} \chi$, via a straightforward hybrid argument. Hence, the simulator emulates the view of the receiver using a random $\mathsf{pk}_S := (\mathbf{k}_0, (u^i)_{i \in [m]}) \xleftarrow{R} \mathcal{R}^m \times \mathcal{P}^m$. Eventually, the simulator recovers $\mathsf{sk}_R = (\mathbf{z}, s_1)$, computes $\mathsf{csk} := (\mathbf{k}_0, \mathbf{Z}_1) := \mathsf{KeyDer}(R, \mathsf{sk}_R, \mathsf{pk}_S)$, and sends csk on behalf of the ideal adversary to the functionality. The output of S in the ideal world is equal to $(\mathbf{k}_0, \mathbf{Z}_1 + \Delta \mathbf{z}^\top)$, which is identical to S's output in the real world (by the same analysis as for Case 1).

Part II

New Tools with Theoretical Applications

Chapter 4

Distributed Point Functions with a Non-Interactive Setup

Summary

Distributed point functions (DPFs) are a useful cryptographic primitive enabling a dealer to distribute short keys to two parties, such that the keys encode additive secret shares of a secret point function. However, in many applications of DPFs, no single dealer entity has full knowledge of the secret point function, necessitating the parties to run an interactive protocol to emulate the setup. Prior works have aimed to minimize complexity metrics of such distributed setup protocols, e.g., *round complexity*, while remaining black-box in the underlying cryptography.

We construct *non-interactive* DPFs (NIDPF), which have a one-round (*simultaneous-message*, semi-honest) setup protocol, removing the need for a trusted dealer. Specifically, our construction allows each party to publish a special "public key" to a public channel or bulletin board, where the public key encodes the party's secret function parameters. Using the public key of another party, any pair of parties can *locally* derive a DPF key for the point function described by the two parties' joint parameter choices.

We realize NIDPF from an array of standard assumptions, including DCR, SXDH, QR, and LWE. Each party's public key is of size $O(N^{2/3})$, for point functions with a domain of size N, which leads to a sublinear communication setup protocol. The only prior approach to realizing such a non-interactive setup required using multi-key fully homomorphic encryption or indistinguishability obfuscation.

As immediate applications of our construction, we obtain "public-key setup" protocols for several existing constructions of pseudorandom correlation generators and round-efficient protocols for secure comparisons.

4.1 Introduction

A point function, denoted by $P_{i,v}$, is a function that evaluates to a message v on input i, and evaluates to zero on all other inputs $j \neq i$ in its domain. A distributed point function (DPF) [GI14, BGI15] allows a trusted dealer to distribute short keys to two parties, where the keys jointly encode a point function $P_{i,v}$ for parameters (i, v) chosen by the dealer. Individually, a DPF key does not reveal any information about the secret index i or message v to the party. However, using their key, each party can locally "evaluate" the point function on a public input x, to obtain an additive *secret share* of $y := P_{i,v}(x)$.

DPFs are the backbone of many useful primitives and protocols relating to multi-party computation (MPC). In particular, DPFs enable communication-efficient generation of correlated randomness in MPC protocols [BCGI18, SGRR19, BCG+19a, BCG+19b, BCG+20b, BCG+20a, YWL+20, AS22, BCG+22, BBC+24], can be used to instantiate distributed oblivious RAM [Ds17, VHG23], privacy-preserving machine learning [RTPB22, YJG+23, JGB+24], private database queries [WYG+17, DFL+20, DRPS22, SSLD22] and analytics [BBC+21, MPD+24, MST24, RZCGP24], and mixed-mode secure computation [BGI19, BCG+21].

However, in many of these applications, there is no trusted dealer that can generate and distribute the DPF keys to the parties. Instead, the trusted dealer is emulated by the parties via a distributed key generation protocol [Ds17, BGIK22, VSH22, VHG23], which the parties invoke to obtain their respective DPF keys. More concretely, in a distributed generation protocol, each party holds a *secret share* of the parameters (i, v). After invoking the protocol, the parties end up with DPF keys that correspond to the point function $P_{i,v}$, such that neither party learns the parameters (i, v) in the process.

Early approaches to distributed key generation simply used generic secure computation, resulting in protocols with at least two rounds of communication, while being non-black-box in the underlying cryptographic primitives. It was shown by Doerner and shelat [Ds17] how to achieve black-box distributed two-party key generation with logarithmically many communication rounds. Later, the DPF construction of Boyle et al. [BGIK22] admitted a 5-round black-box protocol. In both approaches, the DPF key size is polylogarithmic in the domain size N. If one instead relaxes the key size, e.g., to $N^{1/2}$, these approaches can yield black-box distributed generation protocols with round complexity as low as two sequential calls to 1-out-of-$N^{1/2}$ oblivious transfer, resulting in a small constant number of rounds (e.g., four rounds when using two-round OT).

At first glance, it is tempting to think that lower-bounds from the MPC literature [HLP11] would set the minimum number of rounds required for a distributed DPF generation protocol to two. However, upon closer inspection, we observe that because the parties obtain a *key* (which in some DPF constructions can even be distributed pseudorandomly [BGI15, BGIK22]), a DPF generation protocol is *not* subjected to the two-round lower bound because each key can be efficiently simulated (indeed, this directly extends the model of public-key setup for oblivious transfer from Chapter 3). In particular, we can hope to achieve a *non-interactive* generation protocol mimicking non-interactive key exchange protocols like Diffie–Hellman [DH76]. Indeed, spooky encryption [DHRW16] already gives such a protocol through the use of multi-key fully homomorphic encryption (FHE) or indistinguishability obfuscation ($i\mathcal{O}$). However, to date, this has been the *only* known approach to realizing a "non-interactive" protocol (a protocol

where each party only needs to read the other party's public key to locally derive a joint DPF key).

4.1.1 Our results

In this chapter, we put forth and study the notion of a "non-interactive" DPF, and demonstrate constructions from new assumptions. This is motivated by the search for (round-efficient) protocols for eliminating the dealer, that do not require heavy tools like multi-key FHE, and can be instantiated from an array of standard assumptions.

Non-interactive DPFs. Our definition of a *non-interactive* DPF (NIDPF) enables two parties to locally (non-interactively) derive DPF keys by simply reading each other's public keys from a bulletin board. More generally, this model is captured by a one-round, simultaneous-message semi-honest protocol. A simultaneous-message communication pattern captures the interaction of non-interactive key exchange protocols like Diffie–Hellman: (1) two parties exchange messages simultaneously, then (2) each party can use the other party's message to locally derive a joint output (key). Such a model of communication is highly desirable because the first message can be reused (i.e., the message of the first party can be reused indefinitely with many different parties) and the parties do *not* need to be online at the same time to participate.

The problem of generating DPF keys in a simultaneous-message protocol is much more challenging compared to key exchange. This is due to the fact that a DPF setup requires the *total communication* between parties (i.e., the size of the public keys) to be sublinear in the domain of the point function. This requirement is generally challenging to achieve—indeed, the only way we currently know of achieving such succinctness is via multi-key FHE [DHRW16]. Moreover, this connection to "multi-key"-like primitives is inherent, as we remark on later.

Constructing NIDPF. Our primary contribution is to show that, perhaps surprisingly, we can rely on simple cryptography and assumptions to achieve the sublinearity requirements. In particular, inspired by the recent work of Abram, Roy, and Scholl [ARS24], we show that we can realize NIDPF schemes "directly," without going through heavier primitives like multi-key FHE. A NIDPF scheme immediately implies non-interactive key exchange, and thus public-key encryption, which eliminates the possibility of using only lightweight symmetric-key cryptography (e.g., one-way functions). However, we are able to realize NIDPFs from many standard assumptions, including the decisional composite residuosity (DCR) assumption, symmetric external Diffie–Hellman (SXDH) assumption, quadratic residuosity (QR) assumption, the enhanced Diffie–Hellman (EDDH) assumption in class groups, and the learning with errors (LWE) assumption. We summarize our results in Table 4.1 and Theorem 4.1.1.

Theorem 4.1.1 (Informal)**.** *Let N be a domain size. There exists a non-interactive DPF (NIDPF) with a key size $O(N^{2/3})$ and evaluation time $O(N^{5/3})$ under either (1) the DCR assumption, (2) the QR assumption, (3) the EDDH assumption and the uniformity assumption in class groups, (4) the SXDH assumption, or (5) the LWE assumption with a superpolynomial-modulus-to-noise ratio. Here, $O(\cdot)$ hides polynomial factors in the security parameter.*

	Assumption	Transparent Setup	Key Size
[DHRW16, XW23]	LWE / $i\mathcal{O}$+DDH	✓	$\log(N)$
Theorems 4.5.1 & 4.5.2	LWE / RLWE	✓	$N^{2/3}$
	DCR / QR	✗	$N^{2/3}$
	SXDH* / Class Groups+EDDH	✓	$N^{2/3}$

Table 4.1: Summary of our instantiations of NIDPF with domains of size N. Constants and polynomial factors (in the security parameter) are ignored in the asymptotic key size for readability. See Section 4.3 for details on the cryptographic assumptions used. *The SXDH-based construction only supports random payloads (output messages).

As an independent contribution, we define a new abstraction that we call *non-interactive multiplication*, which captures all existing "non-interactive" primitives from a recent line of work. In particular, we identify a surprising (but rather obvious in retrospect) connection between our abstraction and Homomorphic Secret Sharing (HSS). This connection results in constructions of "succinct *multi-key* HSS" (restricted to a special class of computations) from a variety of assumptions, including DDH, DCR, and the EDDH assumption in class groups. To the best of our knowledge, the only prior approaches for such non-interactive computation required using multi-key FHE techniques [DHRW16, XW23]. More concretely, our abstraction allows us to adapt the recent result of Abram et al. [ARS24] constructing succinct (but not multi-key) HSS for "special RMS" programs into a multi-key (i.e., non-interactive), albeit restricted to a slightly weaker class of functions. Specifically, unlike with standard HSS, our construction does not require a correlated setup between parties and only requires a common reference string. Moreover, the additional succinctness property allows one party to have a large input x while maintaining that the input share is succinct in the size of x. We summarize this generalization of our techniques in Theorem 4.1.2 and provide more details in Section 4.6.

Theorem 4.1.2 (Informal)**.** *Let* HSS *be an HSS scheme for the function class* $\mathcal{F}$ *and let* $\mathcal{P}$ *be the set of constant-degree polynomials. There exists a succinct, multi-key HSS scheme for computing functions of the form* $P(x, f(y))$*, where* $P \in \mathcal{P}$ *and* $f \in \mathcal{F}$*, one party has a (large) input* x*, and the other party has a (short) input* y*. Moreover, the total size of both parties' input shares is* $o(|x|) + O(|y|)$*, ignoring polynomial factors in the security parameter.*

4.1.2 Applications

We describe two immediate applications of our NIDPF construction. The primary application is replacing multi-round DPF setup protocols with a non-interactive "public key" setup. In particular, many applications of DPFs require two parties, each holding a secret share of an index $t \in [N]$, to generate DPF keys (through a secure setup protocol) that encode a point function parameterized by t. Concretely, Alice and Bob hold shares $t_A, t_B \in [N]$, such that $t_A + t_B = t \bmod N$, jointly generate DPF keys for the point function $P_{t,1}$. (We assume additive secret sharing of the index t, following [BGIK22, BBC+24]; some protocols also

consider bit-wise XOR secret shares of t, however, applications typically require working with additive secret sharing, e.g., [BCG+20b, BCCD23, BBC+24].)

PCGs with a "public-key" setup. Pseudorandom Correlation Generators (PCGs) [BCGI18, BCG+19a, BCG+19b, SGRR19, BCG+20a, BCG+20b, BCG+22, AS22, BBC+24] are a cryptographic primitive enabling parties to generate long pseudorandom correlations given access to short correlated seeds. In particular, to jointly generate long correlations, it suffices for parties to first execute a secure computation protocol to jointly sample the short PCG seeds, and then *locally* expand them into a large number of pseudorandom correlations. PCG constructions exist for a variety of correlations, including oblivious transfer (OT), vector olivious linear evaluation (VOLE), and Beaver triple correlations, and make heavy use of DPFs. Indeed, the dominant cost of the setup protocol for these constructions is jointly generating DPF keys [SGRR19, BCG+20b, AS22, BBC+24].

Interestingly, a recent line of work [OSY21, BCM+24] has shown that when it comes to OT/VOLE correlations specifically, the parties do not need to engage in the initial interactive setup protocol. Instead, two parties can non-interactively derive a pair of seeds that enables them to expand their correlations locally. Such PCGs are said to have a "public-key setup" protocol, which follows the same non-interactive communication pattern we motivated in Section 4.1. However, to date, the only such "public-key PCG" constructions that exist are for the OT/VOLE correlation [OSY21, BCM+24]. It has remained an open problem to realize public-key PCGs for other correlation types (e.g., Beaver triple correlations), for which we have constructions of PCGs from a variety of standard assumptions but no corresponding public-key setup protocol. By instantiating the DPF in existing PCG constructions (for further classes of correlations) with an NIDPF, it becomes possible to obtain a semi-honest "public-key setup" protocol for the PCG.

We note that we fully resolve the open problem of building public-key correlation functions (which imply PCGs) for many correlation types in Chapter 5 under the DCR assumption. However, our NIDPF constructions are still of interest to building public-key PCGs, since they can be realized from a wider range of assumptions and are more likely to provide a path towards practical constructions.

Mixed-mode secure computation in one round. Recent works on mixed-mode secure computation, beginning with the work of Boyle et al. [BCG+21], have demonstrated that the round and communication complexity of MPC protocols can be improved by using DPFs to help directly evaluate complex functions such as comparisons and equality of secret-shared values, without needing to express the computations as Boolean or arithmetic circuits.

For example, consider the case of securely computing secret shares of an equality predicate evaluated between a public threshold t and secret shared input x, in a two-party setting. The idea, at a high level, is to have a trusted dealer distribute DPF keys to two parties for the point function $P_{t+r,1}$ that evaluates to 1 on index $t + r$, where r is uniformly random in the domain. The dealer additionally distributes additive shares of r to the parties. The parties, holding additive shares of a value x (assumed to be in the domain of the point function), can publicly open the value $y := x + r$ by locally masking their shares of x with their shares of r and broadcasting the result. Then, observe that by evaluating the DPF on input y, the parties obtain shares of 1 if and only if $x + r = t + r$, which is the case if and only if $x = t$.

Because the DPF is "one-time-use" due to the masking term r, the efficiency gains obtained

by such a protocol depend heavily on the efficiency of the DPF setup protocol used by the parties to emulate the dealer. However, when using a NIDPF to replace the dealer, the parties can *simultaneously*: (1) choose their own share $\langle r \rangle_\sigma$ of the random mask r, (2) broadcast their masked share $\langle x \rangle_\sigma + \langle r \rangle_\sigma$, and (3) generate and send their NIDPF key message for the point function $P_{t+r,1}$. This already enables the parties to locally compute additive shares of the t-equality predicate on x, yielding a *single (simultaneous) round* protocol for the equality predicate computation.

4.2 Technical Overview

In this section, we provide a detailed technical overview of our NIDPF construction. The main building block we use in our construction is a novel abstraction we call non-interactive multiplication (NIM), which we overview in Section 4.2.1, and which we view as a contribution of independent interest. In Section 4.2.2, we provide an overview of our NIDPF construction.

4.2.1 Building block: Non-interactive multiplication

At a high level, a NIM allows two parties, Alice and Bob, each holding a ring element as input, to obtain secret shares of the multiplication of their inputs by exchanging one message simultaneously (or posting their message to a public bulletin board). The NIM abstraction captures several primitives recently introduced in various contexts. In particular, NIM directly implies *non-interactive* variants of OT [BM90], VOLE [OSY21, ADOS22, ARS24, BCM+24] (including Chapter 3), and inner-products [CZ22], where parties obtain secret-shares of the computation by exchanging one message simultaneously.

The NIM abstraction also captures the case where Alice and Bob have *matrices* as inputs, and wish to compute secret shares of the matrix product. In particular, a NIM scheme for matrix multiplication allows Alice with a matrix $\mathbf{A}$ and Bob with a matrix $\mathbf{B}$ to compute additive secret shares of $\mathbf{AB}$. Surprisingly, a NIM scheme for matrix multiplication can have *sublinear communication* (in the size of one input matrix), using techniques developed in two recent works that build *succinct* VOLE [ARS24, BCM+24]. This succinct NIM variant is the main building block we use in Section 4.5 to construct NIDPFs. We obtain the following instantiations of succinct NIM for matrix multiplication:

Theorem 4.2.1 (Informal; Implicit in [ARS24, BCM+24])**.** *Let $\mathcal{R}$ be a ring and N, ℓ, m be integer parameters. For $\ell = N^{2/3}$ and $m = N^{1/3}$, there exists a succinct NIM scheme computing shares of $\mathbf{AB}$ with $O(N^{2/3})$ communication, for all matrices $\mathbf{A} \in \mathcal{R}^{\ell \times m}$ and $\mathbf{B} \in \mathcal{R}^{m \times m}$, if one of the following assumptions hold: (1) DCR, (2) QR, (3) an "enhanced" DDH assumption in class groups, (4) LWE with a superpolynomial modulus-to-noise ratio, or (5) SXDH in bilinear groups when the NIM output sharing is defined multiplicatively. In the above, $O(\cdot)$ hides polynomial factors in the security parameter.*

Our proof of Theorem 4.2.1 follows from ideas presented in the recent work of Abram et al. [ARS24] in their construction of "succinct" Homomorphic Secret Sharing (HSS) [BGI16] and a concurrent work of [BCM+24] constructing succinct non-interactive VOLE. In particular, their constructions internally use the ability to multiply a matrix $\mathbf{A}$ by another matrix

$\mathbf{B} := \Delta \cdot \mathbf{I}$ (where Δ is a scalar and $\mathbf{I}$ is the identity matrix), with sublinear communication in the size of $\mathbf{A}$. However, in both these works, the primary goal was constructing a non-interactive VOLE scheme. Non-interactive VOLE, in and of itself, neither implies NIDPFs nor succinct NIM for matrix multiplication, and is overall a weaker primitive. In this chapter, we show that their constructions not only generalize to *any* matrix $\mathbf{B}$ (of appropriate size)—while still preserving sublinear communication—but also use it as a building block to construct NIDPFs in Section 4.5.

Comparison to HSS. While at first glance, it may appear as though using HSS would be sufficient to construct NIM, there are subtle yet important distinctions between these primitives which make them very different. Concretely, we show in Section 4.4 that our NIM abstraction implies a form of (2-party) *multi-key* HSS, in an analogous sense to multi-key FHE, which is a stronger primitive compared to standard HSS.

In particular, the NIM abstraction has a universal setup that consists of a common reference string used by everyone, which enables us to realize a truly "non-interactive" (or multi-key) primitive, eliminating the requirement for multi-round setup protocols. In contrast, HSS [BGI16, BCG+17], including succinct HSS [ARS24], requires a *trusted setup process* to distribute evaluation keys to each party before any computation can be performed, and the setup cannot be used in a computation involving other parties. In a network with many parties, each *pair* of parties needs to generate a unique pair of HSS evaluation keys and can only compute over inputs assigned to them, resulting in quadratic communication overheads. Compare this with *multi-key* HSS (see Chapter 5) or spooky encryption [DHRW16], where any pair of parties can compute a function "on the fly" over their joint inputs and *without* needing to perform a joint setup process ahead of time to do so. As such, we view our NIM abstraction as a potential stepping-stone to uncovering new constructions for efficient MPC. Indeed, in Section 4.6, we generalize the ideas we use to realize our NIDPF constructions to realize a form of multi-key HSS for a restricted class of computations, which may prove to have additional applications.

4.2.2 Overview of the NIDPF construction

A NIDPF consists of a generation algorithm Gen and evaluation algorithm Eval. We let $\mathcal{R}$ be a finite ring and the message space of the NIDPF. A party Alice, with point function parameters (t_A, v_A), uses Gen to generate a public key pk_A and a secret key sk_A. Bob does the same with (t_B, v_B). Then, using the public key of the other party in conjunction with their own secret key, Alice and Bob can locally derive a DPF key encoding the point function with parameters $(t_A + t_B, v_A + v_B)$. Importantly, the public keys generated by Alice and Bob need to be *short*—sublinear in the truth-table size of the point function.

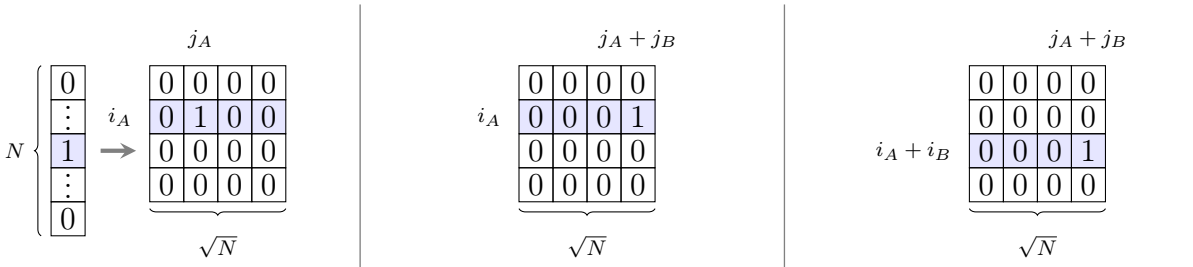
Similarly to some other DPF constructions (and all non-generic DPF key generation protocols) [CBM15, Ds17, BGIK22], our construction of NIDPF is tailored to the "full-domain evaluation" regime, where the parties obtain the output of the point function evaluated on all inputs in the domain. We let N denote the domain size of the point function, and view the point function evaluation on the entire domain (i.e., for every $x \in [N]$) as being a one-hot vector $\mathbf{u} \in \mathcal{R}^N$, where $\mathbf{u}[t] = 1$ and t is the special index.

For suitable choices of N, we can represent $x \in [N]$ by (i, j) where $i = x \pmod{\ell}$ and

$j = x \pmod{m}$ for some coprime integers $\ell, m \in [N]$ such that $N = \ell \cdot m$. Such a mapping (*à la* Chinese Remainder Theorem) allows us to interpret the length-N vector $\mathbf{u}$ as a $\ell \times m$ matrix, while still preserving arithmetic modulo N via the residue number system. This places a restriction on N, which ideally we would like to avoid. In Section 4.5.1, we sketch an alternative approach that works for arbitrary (non-coprime) integers ℓ, m, but has a $2\times$ cost in efficiency.

At a high level, our approach to realizing our NIDPF construction is the following. Assume that Alice parses her index $t_A \in [N]$ as $t_A = (i_A, j_A) \in [\ell] \times [m]$ and Bob parses his index t_B as $(i_B, j_B) \in [\ell] \times [m]$. The goal is to have Alice and Bob derive secret shares of the $\ell \times m$ matrix, where the $(i_A + i_B, j_A + j_B)$-th entry is non-zero. In particular, note that this matrix will be reinterpreted as the unit vector $\mathbf{u}$ with non-zero coordinate $t = t_A + t_B \in [N]$, where $t = i_A + i_B \pmod{\ell}$ and $t = j_A + j_B \pmod{m}$. This is equivalent to the full-domain evaluation of the point function $P_{t,1}$.[1]

Our construction achieves this in two steps, which we overview next.



(a) Alice's initial input vector parsed as a matrix.

(b) Secret-shared matrix held by the parties after Step I.

(c) Secret-shared matrix held by the parties after Step II.

Figure 4.1: Running example used in the overview of the NIDPF construction. The matrix represents the full evaluation of the point function with a domain of size N, where the parameters ℓ, m (as defined in Definition 4.4.1) are $\ell = m = \sqrt{N}$, for simplicity.

Step I: Shifting the columns using NIM. Alice begins by defining a $\ell \times m$ matrix $\mathbf{A}$ with 1 at entry (i_A, j_A) and zeros elsewhere. This is illustrated for the case where $\ell = m = \sqrt{N}$ in Figure 4.1a. Then, the main idea is to obliviously "shift" this matrix by Bob's input (i_B, j_B).

First, we observe that we can perform one dimension of this shift using a matrix multiplication: Bob defines the $m \times m$ *cyclic shift* matrix $\mathbf{S}_{j_B}$ that shifts each column of $\mathbf{A}$ cyclically to the right by j_B (wrapping around modulo m). Using NIM, Alice and Bob can non-interactively compute shares of the matrix $\mathbf{AS}_{j_B}$. Note that $\mathbf{AS}_{j_B}$ is a matrix where the only non-zero entry is located at row i_A and column $j_A + j_B$. This is illustrated in Figure 4.1b for our running example. By applying Theorem 4.2.1, we have that the communication between Alice and Bob in this process is *sublinear* in N. Moreover, by the security of the NIM scheme (see Section 4.4), Bob does not learn the value of (i_A, j_A) and Alice does not learn the value of j_B.

Finally, by interpreting the resulting shares back to a vector $\mathbf{y}$, the parties obtain secret shares corresponding to the full-evaluation of the point function $P_{i_A \cdot m + j_A + j_B, 1}$. Unfortunately,

[1] For now, we assume that the point function outputs $v = 1$ at the special index i and later generalize to arbitrary outputs.

this is not quite what we want, since our goal is for the parties to obtain shares of the full-evaluation corresponding to the point function $P_{(i_A+i_B)\cdot m+j_A+j_B,1} \equiv P_{t_A+t_B,1}$. In particular, notice that following the cyclic shift, Alice still knows which *row* of the resulting matrix contains the non-zero index, since the row index does not currently depend on Bob's input i_B. To remedy this, we need a way for Bob to cyclically shift the rows of the resulting secret-shared matrix by his secret index i_B, which ideally could be done by multiplying the result with another "shift matrix" parameterized by i_B.

Sadly, multiplying by another shift matrix is not possible, since this would require a "NIM" for the degree-3 computation $\mathbf{S}_{i_B}(\mathbf{A}\mathbf{S}_{j_B})$, where $\mathbf{S}_{j_B}$ cyclically shifts the columns of $\mathbf{A}$ by j_B and $\mathbf{S}_{i_B}$ cyclically shifts the rows by i_B. We do not know how to realize such a primitive for degree-3 computations (*even* if we sacrifice the succinctness requirement) without going through "high-end" tools like multi-key FHE [DHRW16].

However, we show that Bob can cyclically shift the rows using degree-2, secret-key HSS (the weakest form of non-trivial HSS [BGI16], which can be instantiated from a wide range of assumptions). In particular, our usage of HSS to let Bob cyclically shift the rows is only possible *after* computing the NIM to cyclically shift the columns, as will become apparent later. We stress that secret-key HSS alone cannot be used to directly build NIDPFs—for one, the NIDPF abstraction directly implies public key encryption, while secret-key HSS (even for higher degree computations) does not [DIJL23].

Step II: Shifting the rows with degree-2 HSS. Our idea is to compose degree-2 HSS with NIM to allow Bob to obliviously cyclically shift the rows *and* columns of Alice's matrix $\mathbf{A}$. This composition with HSS is inspired by the multi-party DPF construction of Abram et al. [ARS24], where they use HSS to obliviously select an appropriate cyclic shift of a one-hot vector by computing an inner product with all possible shifts. However, to apply this idea to the non-interactive setting, there are several challenges we need to overcome.

The first challenge is that HSS schemes, even for degree-2 functions, require a *trusted setup process*, which would prevent us from getting a non-interactive solution (the parties would need to engage in a multi-round setup protocol).

The second challenge is that HSS does not enable computing degree-2 functions on additive secret shares. Instead, typical HSS schemes (following [BGI16]) require parties to have "memory shares" and "input shares" of the secret values in order to perform computations over them. In particular, degree-2 HSS allows two parties to locally compute an additive sharing of xy from an input share of a value x and memory share of a value y. At a very high level, an input share of a message $x \in \mathcal{R}$ is just an encryption of x; and memory shares of a message $y \in \mathcal{R}$ are additive shares of the tuple $(x, x \cdot \mathsf{sk})$, where $\mathsf{sk} \in \mathcal{R}^k$ is the secret key used to encrypt the input share (see Section 4.3.6 for additional background).

If the parties can somehow obtain memory shares of $\mathbf{A}\mathbf{S}_{j_B}$ and input shares of an input provided by Bob, then using HSS for computing degree-2 functions, we have the following solution for cyclically shifting the rows. First, Bob defines the one-hot vector $\mathbf{e}_{i_B}$ representing his row index i_B and sends HSS input shares of $\mathbf{e}_{i_B}$ to Alice. Let $\mathbf{T} := \mathbf{A}\mathbf{S}_{j_B}$ which, for now, we assume Alice and Bob hold memory shares of at the end of Step I. Then, the parties locally define the list of ℓ "shifted" matrices $\mathbf{T}_1, \ldots, \mathbf{T}_\ell$, such that $\mathbf{T}_i$ is the matrix $\mathbf{T}$ with the rows cyclically shifted down by i. Finally, using HSS, the parties compute the following

degree-2 equation to "obliviously select" $\mathbf{T}_{i_B}$:

$$\langle \mathbf{e}_{i_B}, (\mathbf{T}_1, \ldots, \mathbf{T}_\ell) \rangle = \mathbf{T}_{i_B}. \tag{4.1}$$

Observe that this allows Alice and Bob to compute shares of the one-hot matrix with a non-zero entry at index $(i_A + i_B, j_A + j_B)$, as required. A similar idea underpins the multi-party DPF construction of Abram et al. [ARS24]. However, their requirement for a trusted setup makes their approach for obtaining the necessary compatible input and memory shares inherently interactive. We make use of the following two ideas to avoid interaction:

Idea I: Bob generates the HSS setup. To avoid needing a trusted setup, we exploit the fact that Bob knows the full input $\mathbf{e}_{i_B}$, which means that he can act as the trusted dealer to generate the HSS setup in our case. Moreover, we observe that *secret-key* HSS suffices, since only Bob needs to encrypt his input $\mathbf{e}_{i_B}$. This allows us to use the most basic form of HSS, making it easy to instantiate from many standard assumptions, including a novel instantiation we present in Section 4.5.3.1 from the SXDH assumption in bilinear groups.

Idea II: NIM outputs memory shares. To make the output of the NIM compatible with HSS, we need a way for Alice and Bob to obtain memory shares of the matrix $\mathbf{T}$ rather than additive shares. To achieve this, we use the following trick from prior work [CMPR23, ARS24] to generate memory shares. Observe that if Bob multiplies his cyclic shift matrix $\mathbf{S}_{j_B}$ by any scalar $c \in \mathcal{R}$, the output of the NIM will be an additive share of $c \cdot \mathbf{A}\mathbf{S}_{j_B}$. This can be generalized to computing $\mathsf{sk} \otimes \mathbf{A}\mathbf{S}_{j_B}$ in the natural way. Then, the idea is to have Alice and Bob engage in two copies of the NIM protocol simultaneously. In both cases Alice inputs $\mathbf{A}$. Bob, on the other hand, inputs the cyclic shift matrix $\mathbf{S}_{j_B}$ in one instance, and the scaled matrix $\mathsf{sk} \otimes \mathbf{S}_{j_B}$ in the other. Together, the NIM outputs produce an HSS memory share of $\mathbf{A}\mathbf{S}_{j_B}$ under Bob's secret key. In parallel to this, Bob generates an HSS input share for his vector $\mathbf{e}_{i_B}$ and an evaluation key ek_A, which he sends to Alice. Then, using HSS, Alice and Bob compute shares of the inner product from Equation 4.1.

Examining the communication costs. The communication is dominated by the NIM encodings and the length of $\mathbf{e}_{i_B}$, which is $O(\ell)$ (ignoring $\mathsf{poly}(\lambda)$ factors). Thus, the total communication is $O(\ell + m^2)$, which is sublinear in the domain size N using Theorem 4.2.1. Moreover, because this whole protocol only requires one simultaneous exchange of information, Alice and Bob can simply post their messages in the form of a public key, which aligns with our design goals.

Arbitrary payload. The above overview captures a NIDPF construction where the non-zero output (i.e., the payload) of the point function at the special index is the scalar $1 \in \mathcal{R}$. However, to satisfy a more general definition, we need to allow Alice and Bob to also jointly specify the payload v as the sum of their individual payloads v_A and v_B.

To achieve this, we observe that we can use the same "scaling trick" used by Bob to compute the product with his secret key sk to allow either Alice or Bob to specify v_A or v_B as the output. Specifically, it is enough for one of the parties (say Alice) to simply multiply their input matrix by v_σ before generating the NIM encoding. This enables a "half-chosen" variant of the NIDPF, where only one of the two parties is allowed to specify the (secret) payload.

To generalize this to the case where the output is message $v = v_A + v_B$ jointly defined by the two parties, the parties can engage in two instances of the half-chosen protocol, in parallel, and add the resulting shares together. We explain this further in Section 4.5.

4.2.2.1 Random-payload NIDPF from SXDH

In some applications of DPFs, the payload can be *random* and only determined by the random coins of the generation algorithms.[2] Here, we overview a construction of such a NIDPF under the SXDH assumption in bilinear groups. In a nutshell, a bilinear group consists of a triple of cyclic groups: $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ with an efficient map (pairing) $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let g_1, g_2, and g_T be generators for $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, respectively. Then, for all g_1^x and g_2^y, it holds that $e(g_1^x, g_2^y) = g_T^{xy} \in \mathbb{G}_T$. This feature enables computing the multiplication "in the exponent" of the bilinear group.

Idea: Replacing Degree-2 HSS with a pairing. The main idea behind our NIDPF construction in bilinear groups is to follow the template outlined in Section 4.2.2 but replace Step II with a "multiplication in the exponent" using the pairing. We first construct a special succinct NIM scheme from the DDH assumption (over any suitable cyclic group $\mathbb{G}$), which outputs the result of the matrix product "in the exponent" of the group $\mathbb{G}$, restricting the ring $\mathcal{R}$ to $\mathbb{Z}_p$. This variant of NIM is appealing since it allows us to use *any* DDH group $\mathbb{G}$ to compute the matrix multiplication, but is limiting in that we cannot obtain the memory shares required for the HSS computation in Step II of Section 4.2.2. However, we observe that we don't need to do so if we have a pairing! Specifically, we can instead replace the HSS computation in Step II by computing the multiplication using the pairing, as sketched above.

More concretely, in our DDH-based succinct NIM variant, the parties obtain *multiplicative* shares $g^{\mathbf{R}_A}$ and $g^{\mathbf{R}_B}$, respectively, such that $g^{\mathbf{R}_A} \cdot g^{\mathbf{R}_B} = g^{\mathbf{AB}}$ (where the notation $g^{\mathbf{M}}$ denotes the matrix of group elements $g^{m_{ij}}$ for all entries $m_{ij} \in \mathbf{M}$). Therefore, by instantiating this NIM scheme in the group $\mathbb{G}_1$ of a bilinear group, the parties obtain multiplicative shares of $g_1^{\mathbf{T}}$, rather than additive shares of $\mathbf{T}$ (defined in Step II above). Nonetheless, the parties can still define multiplicative shares of the vectors $\mathbf{T}_1, \ldots, \mathbf{T}_\ell$ "in the exponent" of g_1, as before. Then, Bob can encrypt his one-hot vector $\mathbf{e}_{i_B}$ with the aim of selecting the appropriate $\mathbf{T}_{i_B}$. However, instead of using HSS to do so, Bob simply uses ElGamal encryption in the group $\mathbb{G}_2$ to compute:

$$(g_2^{r_j}, g_2^{e_{i_B,j}} h_2^{r_j}),\ \forall j \in [m],$$

where $h_2 := g_2^{\mathsf{sk}}$ is an ElGamal public key for an sk known to Bob, and each r_j is uniformly random. Since now we need DDH to hold in both $\mathbb{G}_1$ and $\mathbb{G}_2$, we must rely on the SXDH assumption.

Given these ciphertexts, Alice and Bob compute the inner product from Equation 4.1 "in the exponent" using the pairing and obtain multiplicative shares of the inner product in $\mathbb{G}_T$:

$$g_T^{\left\langle \mathbf{e}_{i_B}, (\mathbf{T}_1, \ldots, \mathbf{T}_\ell) \right\rangle} = g_T^{\mathbf{T}_{i_B}}. \tag{4.2}$$

We can view this as replacing the HSS scheme used by Bob in the overview of Section 4.2.2 with

[2] Note that a secret sharing of the random payload can be derived non-interactively by each party summing all entries of its length-N DPF evaluation vector.

a "multiplicative HSS" scheme from pairings: i.e., where the HSS outputs are multiplicatively, rather than additively, secret shared. However, as with the first scheme, Alice and Bob still need to compute "memory shares" for this multiplicative variant of HSS. Memory shares now take on the form $(g_1^{\mathbf{T}}, g_1^{\mathsf{sk}\cdot\mathbf{T}})$, which can be obtained by having Bob scale his matrix by sk.

Converting from multiplicative to additive shares. Now the issue we face is the following. The result in Equation 4.2 is a *multiplicative* sharing of the full-domain evaluation, which does not correspond to the desired additive shares we need for the NIDPF. To solve this problem, we need a way to "bring down" the exponent and convert it to additive shares. One way to achieve this would be using the Distributed Discrete Logarithm (DDLog) procedure [BGI16].

Using the DDLog algorithm, Alice and Bob can derive additive shares of $\mathbf{T}_{i_B}$ by applying DDLog to each entry of $g_T^{\mathbf{T}_{i_B}}$. However, there is now a problem of *correctness* for the resulting output shares of the NIDPF. Specifically, DDLog has a $1/\mathsf{poly}(\lambda)$ error (in which case it outputs a uniformly random value in $\{0, 1, \ldots, M\}$), which would translate to a $1 - 1/\mathsf{poly}(\lambda)$ correctness for the output shares of the NIDPF. Having a (non-negligible) correctness error is undesirable, and prevents applying the resulting NIDPF in many contexts. We show how to sidestep this problem by making an important observation regarding use of the DDLog procedure, which we explain next.

Random payload. Surprisingly, we show that the error can in fact be avoided entirely when constructing a NIDPF with a random payload (i.e., a NIDPF which outputs a random message at the special index). In particular, we observe that existing constructions of DDLog have *no error* when given multiplicative shares of the identity element g^0 [BGI16, DKK20]. Inspired by this observation, we show that we can obtain a NIDPF with random payloads. We observe that $\mathbf{T}_{i_B}$ is a one-hot matrix, and so has only one non-zero entry. By having the parties set their payload share to a uniformly random scalar, they can further ensure the non-zero value of $\mathbf{T}_{i_B}$ has high (pseudo)entropy to both parties. Thus, we can simply use a PRF F_K with outputs in $\mathbb{Z}_M$ to generate additive shares from the multiplicative shares. To see this, note that:

- for the multiplicative shares g^{x_0}, g^{x_1} of the *non-zero* entry $g^{x_0+x_1}$ of $\mathbf{T}_{i_B}$, $F_K(g^{x_0}) - F_K(g^{-x_1})$ is a pseudorandom value in $\mathbb{Z}_M$ (even given the PRF key K); however,
- for all multiplicative shares g^{x_0}, g^{x_1} such that $g^{x_0} \cdot g^{x_1} = g^0$, we have $F_K(g^{x_0}) - F_K(g^{-x_1}) = 0$.

This means that parties obtain pseudorandom shares of zero on all entries except for the entry with the non-zero value, where they obtain shares of some pseudorandom value. We provide details on the DDLog algorithm and our NIDPF construction from SXDH in Section 4.5.

4.3 Preliminaries

In this section, we provide the necessary notational and cryptographic preliminaries that we use our construction of NIDPF.

4.3.1 Notation

We let $\mathbb{N}$ denote the set of natural numbers, $\mathbb{Z}$ denote the set of integers, and $\mathbb{G}$ denote a finite group. We let $\mathcal{R}$ denote a finite ring. We denote by $\mathsf{poly}(\cdot)$ the set of all polynomials and by $\mathsf{negl}(\cdot)$ any negligible function. We occasionally abuse notation and let poly denote a fixed polynomial.

Vectors and matrices. We denote a vector $\mathbf{u}$ using bold lowercase letters and let $\mathbf{u}[i]$ denote the i-th coordinate of $\mathbf{u}$. We denote matrices $\mathbf{A}$ with bold uppercase letters and let $\mathbf{A}[i, j]$ denote the element of $\mathbf{A}$ located at the i-th row, j-th column.

Sampling and assignment. We let $x \xleftarrow{\text{R}} S$ denote a uniformly random sample drawn from a set S. We let $x \leftarrow \mathcal{A}$ denote assignment from a randomized algorithm $\mathcal{A}$ and $x := y$ denote initialization of x to the value of y (which may be the output of a deterministic algorithm).

Efficiency and indistinguishability. By an *efficient* algorithm $\mathcal{A}$ we mean that $\mathcal{A}$ is modeled by a (possibly non-uniform) Turing Machine that runs in probabilistic polynomial time. We write $D_0 \approx_c D_1$ to mean that two distributions D_0 and D_1 are *computationally* indistinguishable to all efficient distinguishers $\mathcal{D}$ and $D_0 \approx_s D_1$ to mean that D_0 and D_1 are *statistically* indistinguishable.

Rounding. We let $\lfloor x \rceil$ denote the rounding of a real number x to the nearest integer. For integers $q > p \geq 2$, we define the modular rounding function $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ as $\lfloor v \rceil_p = \lfloor (p/q) \cdot v \rceil$.

Party identifiers. We identify parties with letters A and B, and use $\sigma \in \{A, B\}$ to refer to a party. We will slightly abuse notation by letting $\overline{\sigma}$, for some $\sigma \in \{A, B\}$, refer to the party identifier in the singleton set $\{A, B\} \setminus \{\sigma\}$.

4.3.2 Additive secret sharing

We define the function $\mathsf{Share}_{\mathbb{G}}(\cdot)$ to be the (randomized) function that outputs a tuple of additive shares in $\mathbb{G}$, such that each share is individually uniformly random over $\mathbb{G}$. For simplicity, we will denote the tuple of additive shares of a secret s by $(\langle s \rangle_0, \langle s \rangle_1)$, such that $\langle s \rangle_0 + \langle s \rangle_1 = s \in \mathbb{G}$.

4.3.3 Cryptographic assumptions

In this section, we present the cryptographic assumptions we build NIDPFs from, including the DDH assumption, the SXDH assumption, and the NIDLS framework.

Definition 4.3.1 (Decisional Diffie–Hellman (DDH) Assumption)**.** *Let λ be a security parameter. Let $\mathbb{G}$ be a cyclic group of prime order $p = p(\lambda) \in \mathsf{poly}(\lambda)$ with generator g. The DDH assumption states that: $(g, g^a, g^b, g^{ab}) \approx_c (g, g^a, g^b, g^c)$, where $a, b, c \xleftarrow{R} \mathbb{Z}_p$.*

Definition 4.3.2 (Symmetric External Diffie–Hellman (SXDH) Assumption)**.** *Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be a bilinear group, where $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ are cyclic groups of prime order $p = p(\lambda) \in \mathsf{poly}(\lambda)$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map. Let g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The SXDH assumption states that the DDH assumption (cf. Definition 4.3.1) holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Definition 4.3.3 (Learning with Errors Assumption)**.** *Let χ denote a discrete Gaussian noise distribution. Let $n = n(\lambda), m = m(\lambda)$, and $q = q(\lambda)$, all polynomial in λ. The learning with errors (LWE) assumption states that $(\mathbf{A}, \mathbf{A}^\top \mathbf{s} + \mathbf{e}) \approx_c (\mathbf{A}, \mathbf{u})$ where $\mathbf{A} \leftarrow\!\$\, \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow\!\$\, \mathbb{Z}_q^n, \mathbf{e} \leftarrow\!\$\, \chi^m$, and $\mathbf{u} \leftarrow\!\$\, \mathbb{Z}_q^m$.*

Definition 4.3.4 (Distributed Discrete Logarithm [BGI16])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter and $\epsilon = \epsilon(\lambda)$. Let $\mathbb{G}$ be an arbitrary cyclic group with generator g and let $1 \leq M \ll |\mathbb{G}|$ be an integer. Let $\mathsf{crs} := (\mathbb{G}, g, M)$ be a common reference string. An efficient algorithm DDLog solves the distributed discrete logarithm in $\mathbb{G}$ with ϵ-correctness, if for all $x \in \mathbb{Z}_M$ and every pair of elements $h_A, h_B \in \mathbb{G}$ such that $h_A \cdot h_B = g^x$,*

$$\Pr\Big[\ \langle z \rangle_A - \langle z \rangle_B = x \quad : \quad \langle z \rangle_\sigma := \mathsf{DDLog}(\mathsf{crs}, h_\sigma),\ \forall \sigma \in \{A, B\}\ \Big] \geq \epsilon(\lambda).$$

4.3.4 The NIDLS framework

The non-interactive discrete log sharing (NIDLS) framework [ADOS22] abstracts several HSS constructions [OSY21, RS21]. The NIDLS framework defines a finite Abelian group $\mathbb{G} = F \times H$, where the discrete log problem is easy in F and assumed to be computationally intractable in H. Essentially, this allows two parties to non-interactively compute secret shares of a discrete log in F.

Definition 4.3.5 (NIDLS Framework [ADOS22])**.** *The NIDLS framework consists of three efficient algorithms $(\mathsf{GGen}, \mathcal{D}, \mathsf{DDLog})$ with the following functionality:*

- $\mathsf{GGen}(1^\lambda) \to \mathsf{crs} := (\mathbb{G}, F, H, g, p, t, \mathsf{aux})$. *The randomized group generation algorithm takes as input the security parameter and outputs a common reference string crs which consists of:*
  - *finite Abelian group $\mathbb{G}$,*
  - *subgroups F and H such that $\mathbb{G} = F \times H$,*
  - *generator g and order p of F,*
  - *positive integer t,*
  - *and auxiliary information aux.*
- $\mathcal{D}(1^\lambda, \mathsf{crs}) \to (h, \rho)$. *The randomized sampling algorithm takes as input the security parameter and common reference string, and outputs a group element $h \in \mathbb{G}$ along with some auxiliary information ρ.*
- $\mathsf{DDLog}(\mathsf{crs}, h) \to s$. *The deterministic distributed discrete log algorithm takes as input a common reference string and a group element, and outputs an element $s \in \mathbb{Z}_p$.*

The above functionality needs to satisfy the following properties:

Correctness. *For all security parameters $\lambda \in \mathbb{N}$ and efficient adversaries $\mathcal{A}$, there exists a*

negligible function negl *such that*

$$\Pr\left[\langle s\rangle_A - \langle s\rangle_B = m \pmod p \quad : \quad \begin{array}{r} \mathsf{crs} := (\mathbb{G}, F, H, g, p, t, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (h_A, m) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ h_B := g^m \cdot h_A \\ \langle s\rangle_A := \mathsf{DDLog}(\mathsf{crs}, h_A) \\ \langle s\rangle_B := \mathsf{DDLog}(\mathsf{crs}, h_B) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. *For all security parameters* $\lambda \in \mathbb{N}$*, it holds that:*

$$\left\{ (\mathsf{crs}, h, \rho, h^r) \;\middle|\; \begin{array}{r} \mathsf{crs} := (\mathbb{G}, F, H, g, p, t, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (h, \rho) \leftarrow \mathcal{D}(1^\lambda, \mathsf{crs}) \\ r \xleftarrow{R} [t] \end{array} \right\}$$

$$\approx_s \left\{ (\mathsf{crs}, h, \rho, h') \;\middle|\; \begin{array}{r} \mathsf{crs} := (\mathbb{G}, F, H, f, p, t, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (h, \rho) \leftarrow \mathcal{D}(1^\lambda, \mathsf{crs}) \\ h' \xleftarrow{R} \langle h\rangle \end{array} \right\}.$$

I.e., the group elements h^r *and* h' *are* statistically *indistinguishable.*

Known instantiations. The NIDLS framework has been instantiated in the Paillier group under the DCR assumption, in class groups under a variant of the DDH assumption (see below), and in the group of elements in $\mathbb{Z}_n^*$ with a Jacobi symbol of 1 (under the quadratic residuosity assumption), where n is the product of two large random safe primes. We refer to [ADOS22, ARS24] for formal definitions of these instantiations.

To instantiate "ElGamal-like" encryption in class groups, we will need to use the Enhanced DDH assumption [ARS24]. This assumption states that given the parameters of the NIDLS group and $\ell + 1$ group elements $g_0, \ldots, g_\ell$ sampled from $\mathcal{D}$ (along with the corresponding auxiliary information $\rho_0, \ldots, \rho_\ell$), it is hard to distinguish between $(g_0^w, \ldots, g_\ell^w)$ for a random w and $(f^{r_0} \cdot g_0^w, \ldots, f^{r_\ell} \cdot g_\ell^w)$ for random $r_0, \ldots, r_\ell \in \mathbb{Z}_q$.

Definition 4.3.6 (The ℓ-ary Enhanced DDH Assumption [ARS24])**.** *Let* GGen *and* $\mathcal{D}$ *be as defined in Definition 4.3.5. The* ℓ*-ary Enhanced DDH (*ℓ*-EDDH) assumption in the NIDLS*

framework states that:

$$\left\{ \begin{array}{l} \mathsf{crs} \\ h_0, \ldots, h_\ell \\ \rho_0, \ldots, \rho_\ell \\ h_0^w, \ldots, h_\ell^w \end{array} \middle| \begin{array}{r} (\mathbb{G}, F, H, g, q, t, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (h_j, \rho_j) \leftarrow \mathcal{D}(1^\lambda, \mathsf{crs}),\ \forall j \in \{0, 1, \ldots, \ell\} \\ w \xleftarrow{R} [t] \end{array} \right\}$$

$$\approx_c \left\{ \begin{array}{l} \mathsf{crs} \\ h_0, \ldots, h_\ell \\ \rho_0, \ldots, \rho_\ell \\ g^{r_0} \cdot h_0^w, \ldots, g^{r_\ell} \cdot h_\ell^w \end{array} \middle| \begin{array}{r} (\mathbb{G}, F, H, g, q, t, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (h_j, \rho_j) \leftarrow \mathcal{D}(1^\lambda, \mathsf{crs}),\ \forall j \in \{0, 1, \ldots, \ell\} \\ w \xleftarrow{R} [t] \\ r_j \xleftarrow{\$} \mathbb{Z}_q, \forall j \in \{0, 1, \ldots, \ell\} \end{array} \right\}.$$

4.3.5 Secret-key homomorphic secret sharing

Here, we provide the full definition of Homomorphic Secret Sharing (HSS).

Definition 4.3.7 (Secret-Key HSS; Adapted from [BGI16, BKS19, DIJL23])**.** *Let λ be a security parameter, $\mathcal{R}$ be a finite ring, and $\mathcal{F}$ be a class of ℓ input functions defined over $\mathcal{R}$. A (secret key) HSS scheme with message space $\mathcal{R}$ consists of six efficient algorithms* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Eval}, \mathsf{Output})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B))$. *The randomized setup algorithm takes as input the security parameter. It outputs a secret key* sk *and a pair of HSS evaluation keys* $(\mathsf{ek}_A, \mathsf{ek}_B)$.
- $\mathsf{Share}(\mathsf{sk}, x) \to (\llbracket x \rrbracket_A, \llbracket x \rrbracket_B)$. *The randomized share algorithm takes as input the secret key* sk *and message $x \in \mathcal{R}$. It outputs an input share of x.*
- $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, f, \llbracket \mathbf{x} \rrbracket_\sigma) \to \langle\langle y \rangle\rangle_\sigma$. *The deterministic evaluation algorithm takes as input the party identifier $\sigma \in \{A, B\}$, an evaluation key* ek_σ*, function $f \in \mathcal{F}$, and input shares of $\mathbf{x} := (x_1, \ldots, x_\ell)$. It outputs a memory share of $y := f(\mathbf{x})$.*
- $\mathsf{Output}(\sigma, \langle\langle y \rangle\rangle_\sigma) \to \langle y \rangle_\sigma$. *The deterministic output algorithm takes as input the party identifier $\sigma \in \{A, B\}$ and a memory share of y. It outputs a share of y.*

When $\mathsf{Alg} \in \{\mathsf{Share}, \mathsf{Output}\}$ *is given as input vector of input (or memory) shares, it outputs the vector obtained by evaluating* Alg *on each coordinate of the input vector independently.*

The above algorithms must satisfy correctness and security:

Correctness. *We say the HSS scheme is ϵ-correct, for some $0 < \epsilon \leq 1$, if for all functions $f \in \mathcal{F}$, and for all vectors $\mathbf{x} \in \mathcal{R}^\ell$, it holds that:*

$$\Pr\left[\langle z \rangle_A - \langle z \rangle_B = f(\mathbf{x}) \quad : \quad \begin{array}{r} (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\llbracket \mathbf{x} \rrbracket_A, \llbracket \mathbf{x} \rrbracket_B) \leftarrow \mathsf{Share}(\mathsf{sk}, \mathbf{x}) \\ \langle\langle y \rangle\rangle_\sigma := \mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, f, \llbracket \mathbf{x} \rrbracket_\sigma) \\ \langle z \rangle_\sigma := \mathsf{Output}(\sigma, \langle\langle y \rangle\rangle_\sigma) \end{array} \right] \geq \epsilon - \mathsf{negl}(\lambda).$$

Security. *For every $\sigma \in \{A, B\}$, and all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$ we have that:*

$$\Pr\left[\begin{array}{rr} & (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{Setup}(1^\lambda) \\ & (\mathbf{x}_0, \mathbf{x}_1, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{ek}_\sigma) \\ b' = b \quad : & b \xleftarrow{R} \{0,1\} \\ & (\llbracket \mathbf{x}_b \rrbracket_A, \llbracket \mathbf{x}_b \rrbracket_B) \leftarrow \mathsf{Share}(\mathsf{sk}, \mathbf{x}_b) \\ & b' \leftarrow \mathcal{A}(\mathsf{st}, \llbracket \mathbf{x}_b \rrbracket_\sigma) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where for all $b \in \{0,1\}$, $\mathbf{x}_b \in \mathcal{R}^\ell$ and $k = k(\lambda) \in \mathsf{poly}(\lambda)$.

4.3.6 Degree-2 homomorphic secret sharing

Here, we define the most minimal form of Homomorphic Secret Sharing (HSS), which will be sufficient for our construction of NIDPFs. The definition is adapted from a more general definition of secret-key HSS (cf. Definition 4.3.7) and is satisfied by existing HSS constructions in the NIDLS framework and from lattice-based assumptions.

Definition 4.3.8 (Degree-2 Secret-Key HSS; Adapted from [BGI16, DIJL23])**.** *Let λ be a security parameter and $\mathcal{R}$ be a finite ring. A Degree-2 (secret key) HSS scheme with message space $\mathcal{R}$ consists of four efficient algorithms $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Convert}, \mathsf{Mult})$ with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B))$*. The randomized setup algorithm takes as input the security parameter and outputs a secret key sk and a pair of HSS evaluation keys $(\mathsf{ek}_A, \mathsf{ek}_B)$.*
- $\mathsf{Share}(\mathsf{sk}, x) \to (\llbracket x \rrbracket_A, \llbracket x \rrbracket_B)$*. The randomized share algorithm takes as input the secret key sk and message $x \in \mathcal{R}$. It outputs a pair of input shares of x.*
- $\mathsf{Convert}(\sigma, \mathsf{ek}_\sigma, \llbracket x \rrbracket_\sigma) \to \langle\langle x \rangle\rangle_\sigma$*. The deterministic conversion algorithm takes as input the party identifier $\sigma \in \{A, B\}$, an evaluation key ek_σ, and input share of x. It outputs a memory share of x.*
- $\mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, \llbracket x \rrbracket_\sigma, \langle\langle y \rangle\rangle_\sigma) \to \langle z \rangle_\sigma$*. The deterministic multiplication algorithm takes as input the party identifier $\sigma \in \{A, B\}$, an evaluation key ek_σ, an input share of x, and a memory share of y. It outputs a share of z.*

When $\mathsf{Alg} \in \{\mathsf{Share}, \mathsf{Convert}, \mathsf{Mult}\}$ is given as input a vector of input (or memory) shares, it outputs the vector obtained by evaluating Alg on each coordinate of the input vector independently.

The above algorithms must satisfy correctness and security:

Correctness. *For all security parameters $\lambda \in \mathbb{N}$, and for all messages $x, y \in \mathcal{R}$, we say the HSS scheme is ϵ-correct, for some $0 < \epsilon \leq 1$ if there exists a negligible function $\mathsf{negl}(\cdot)$ such*

that:

$$\Pr\left[\begin{array}{l} \langle z\rangle_A - \langle z\rangle_B = xy \quad : \\ \end{array} \begin{array}{r} (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\llbracket x \rrbracket_A, \llbracket x \rrbracket_B) \leftarrow \mathsf{Share}(\mathsf{sk}, x) \\ (\llbracket y \rrbracket_A, \llbracket y \rrbracket_B) \leftarrow \mathsf{Share}(\mathsf{sk}, y) \\ \langle\langle y \rangle\rangle_\sigma := \mathsf{Convert}(\sigma, \mathsf{ek}_\sigma, \llbracket y \rrbracket_\sigma),\ \forall \sigma \in \{A, B\} \\ \langle z \rangle_\sigma := \mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, \llbracket x \rrbracket_\sigma, \langle\langle y \rangle\rangle_\sigma) \end{array}\right] \geq \epsilon - \mathsf{negl}(\lambda).$$

Security. *For all security parameters $\lambda \in \mathbb{N}$ such that, for every $\sigma \in \{A, B\}$, and all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$*,*

$$\Pr\left[\begin{array}{l} b' = b \quad : \\ \end{array} \begin{array}{r} (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbf{x}_0, \mathbf{x}_1, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{ek}_\sigma) \\ b \xleftarrow{R} \{0,1\} \\ (\llbracket \mathbf{x}_b \rrbracket_A, \llbracket \mathbf{x}_b \rrbracket_B) \leftarrow \mathsf{Share}(\mathsf{sk}, \mathbf{x}_b) \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \llbracket \mathbf{x}_b \rrbracket_\sigma) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where for all $b \in \{0,1\}$, $\mathbf{x}_b \in \mathcal{R}^\ell$ and $k = k(\lambda) \in \mathsf{poly}(\lambda)$.

4.3.6.1 Memory shares in HSS schemes

All existing HSS constructions in the NIDLS framework [ADOS22, ARS24], and direct constructions from DDH [BGI16, BCG+17] or lattice-based assumptions [BKS19], are constructed using the following template. The HSS secret key sk is a vector of ring elements from the ring $\mathcal{R}$ and corresponds to the decryption key of some additively-homomorphic encryption scheme with message space $\mathcal{R}$, supporting some form of linear (or nearly-linear) decryption. The evaluation keys $(\mathsf{ek}_A, \mathsf{ek}_B)$ are additive shares of the secret key sk. For degree-2 computations, *input shares* are simply encryptions of the message x under sk, while *memory shares* consist of additive shares of x and $\mathsf{sk} \cdot x$. Multiplication of an input share of x with a memory share of y can then be computed as follows. First, using the homomorphism of the encryption scheme, compute an encryption of the additive share of $z = x \cdot y$ by multiplying the encrypted message with the additive share of y. Second, using the linear decryption property of the encryption scheme, compute the decryption of the resulting ciphertext using the additive share of $y \cdot \mathsf{sk}$ to recover the additive share of z.

We formalize the property of "multiplication by a memory share," which we will use in our NIDPF construction. We note that several prior works (e.g., [CMPR23, ARS24]) make use of such "multiplication by memory shares," without explicitly formalizing the property.

Definition 4.3.9 (Multiplication by Memory Shares)**.** *Let* $\mathsf{HSS} = (\mathsf{Input}, \mathsf{Share}, \mathsf{Eval})$ *(cf. Definition 4.3.7) with a finite ring $\mathcal{R}$ as the message space. We say an HSS scheme supports* multiplication by a memory share *if the following three properties are simultaneously satisfied:*

(1) The secret key of the HSS scheme is a vector $\mathsf{sk} \in \mathcal{R}^k$, for some $k \in \mathbb{N}$.

(2) A memory share $\langle\langle y \rangle\rangle_\sigma$ for any message $y \in \mathcal{R}$ consists of an additive share of the tuple $(y, y \cdot \mathsf{sk})$, defined over $\mathcal{R}$.

(3) There exists an efficient, deterministic algorithm $\mathsf{MultEval}$ *with the same syntax as* Eval*, such that for all messages $y \in \mathcal{R}$, all memory shares $(\langle\langle y \rangle\rangle_A, \langle\langle y \rangle\rangle_B)$ of y, all input shares $(\llbracket x \rrbracket_A, \llbracket x \rrbracket_B)$ of $x \in \mathcal{R}$, and all functions f in the family of functions computable by* HSS*, it holds that:*

$$\Pr\left[\langle z \rangle_A - \langle z \rangle_B = y \cdot f(x) \quad : \quad \begin{array}{r} \langle z \rangle_\sigma := \mathsf{MultEval}(\sigma, \mathsf{ek}_\sigma, f, \llbracket x \rrbracket_\sigma, \langle\langle y \rangle\rangle_\sigma), \\ \forall \sigma \in \{A, B\} \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

In words, any computation evaluated by the HSS scheme can be "pre-multiplied" by a value y given only a memory share of y.

We note that Definition 4.3.8 explicitly captures property (3) from Definition 4.3.9. Importantly to us, Definition 4.3.9 is satisfied by all existing HSS constructions, including HSS construction from the DDH [BGI16,BCG+17,BGI17], DCR [OSY21,RS21], QR [OSY21,ADOS22] (for degree-2 computations), class groups [ADOS22], and LWE [BKS19].

4.4 Non-Interactive Multiplication

In this section, we define the notion of non-interactive multiplication. As mentioned in the technical overview, this definition captures the core ingredient used in several prior works, including the non-interactive OT construction of [BM90], non-interactive VOLE (e.g., [OSY21, ARS24, BCM+24]), and the notion of non-interactive inner-products [CZ22]. We believe that our abstraction is of independent interest and may aid in further studying the applications of these primitives. Indeed, in Section 4.6, we show that we can bootstrap the NIM abstraction to compute more expressive functions in a "non-interactive" manner.

Definition 4.4.1 (Non-Interactive Multiplication)**.** *Let λ be a security parameter and $\mathcal{R}$ be a finite ring. A Non-Interactive Multiplication (NIM) scheme consists of five efficient algorithms,*

$$\mathsf{NIM} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma \in \{A,B\}}),$$

with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$*. The randomized setup algorithm takes as input the security parameter and outputs a common reference string (CRS)* crs*.*
- $\mathsf{Encode}_\sigma(\mathsf{crs}, v) \to (\mathsf{pe}_\sigma, \mathsf{st}_\sigma)$*. The randomized encoding algorithm is parameterized by a party identifier $\sigma \in \{A, B\}$. It takes as input the CRS* crs *and message v. It outputs a public encoding* pe_σ *and secret state* st_σ*.*
- $\mathsf{Decode}_\sigma(\mathsf{crs}, \mathsf{pe}_{\overline{\sigma}}, \mathsf{st}_\sigma) \to \langle z \rangle_\sigma$*. The deterministic decoding algorithm is parameterized by a party identifier $\sigma \in \{A, B\}$. It takes as input the CRS* crs*, public encoding* $\mathsf{pe}_{\overline{\sigma}}$ *belonging to the other party, and a secret state* st_σ *belonging to party σ. It outputs a share of z over $\mathcal{R}$.*

The above functionality must satisfy correctness and security:

Correctness. *For all security parameters $\lambda \in \mathbb{N}$ and every pair of elements $x, y \in \mathcal{R}$, a NIM scheme is said to be correct if there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\Pr\left[\begin{array}{c|rl} & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, x) \\ \langle z\rangle_A - \langle z\rangle_B = xy \quad : & (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, y) \\ & \langle z\rangle_A := \mathsf{Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A) \\ & \langle z\rangle_B := \mathsf{Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. *For all efficient adversaries $\mathcal{A}$, for all $\sigma \in \{A, B\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\Pr\left[\begin{array}{c|rl} & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & (v_0, v_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b = b' \quad : & b \xleftarrow{R} \{0,1\} \\ & (\mathsf{pe}_\sigma, \mathsf{st}_\sigma) \leftarrow \mathsf{Encode}_\sigma(\mathsf{crs}, v_b) \\ & b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pe}_\sigma) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

In words, the public encoding hides the message.

4.4.1 NIM with multiplicative output reconstruction

We also define *multiplicative* rather than additive reconstruction for NIM, which will serve us in instantiating a NIDPF in bilinear groups. In this case, Decode_σ outputs a group element Z_σ for $\sigma \in \{A, B\}$, such that $Z_A/Z_B = g^{xy}$, where g is a generator of a cyclic group $\mathbb{G}$.

Definition 4.4.2 (Multiplicative Reconstruction)**.** *A NIM scheme* NIM *is said to have* multiplicative reconstruction *if the correctness property of Definition 4.4.1 is instead stated as follows.*

Multiplicative-Output Correctness. *Let $\mathbb{G}$ be an Abelian group of order p with generator g. For all security parameters $\lambda \in \mathbb{N}$ and every pair of elements $x, y \in \mathbb{Z}_p$, a NIM scheme (instantiated with $\mathcal{R} = \mathbb{Z}_p$) is said to be correct if there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\Pr\left[\begin{array}{c|rl} & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, x) \\ Z_A \cdot Z_B = g^{xy} \quad : & (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, y) \\ & Z_A := \mathsf{Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A) \\ & Z_B := \mathsf{Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Remark 19 (General reconstruction)**.** *We note that we could have defined NIM (Definition 4.4.1) to have an arbitrary reconstruction algorithm, that we then instantiate either as being addition or multiplication. However, because for most applications of NIM the additive reconstruction property is more desirable, we choose to instead provide two definitions noting that the more general abstraction is possible.*

4.4.2 Succinct NIM for matrix multiplication

When computing non-interactive *matrix* multiplication, we can realize a "batch NIM" scheme that achieves sublinear encoding size relative to the size of the output matrix, which translates to sublinearity with respect to one of the party's inputs (or the size of the joint output). Note that we cannot, in general, require succinctness in *both* of the parties inputs since this would contradict information-theoretic lower bounds [ARS24].

Definition 4.4.3 (ϵ-succinct Matrix NIM)**.** *A NIM scheme for matrix multiplication is said to be* ϵ-succinct*, for some* $0 \leq \epsilon < 1$*, if for all security parameters* $\lambda \in \mathbb{N}$*, every CRS* crs*, integers* $\ell, m, k \in \mathbb{N}$ *such that* $\ell > k$*, and every pair of matrices* $(\mathbf{M}_A, \mathbf{M}_B) \in \mathcal{R}^{\ell \times m} \times \mathcal{R}^{m \times k}$*, it holds that for* $N := \ell \cdot m$*,*

$$|\mathsf{pe}_A| \leq N^{\epsilon} \cdot \mathsf{poly}(\lambda, \log |\mathcal{R}|),$$

where $(\mathsf{pe}_A, \_) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, \mathbf{M}_A)$*. In words, the public encoding generated by the party with the larger matrix is sublinear in the size of its matrix.*

Remark 20 (Connection to "Bilinear HSS")**.** *Abram et al. [ARS24] define the notion of Bilinear HSS, which is conceptually related to our formalization of succinct NIM. While the notions share some similarities, succinct NIM is a stronger definition due to the non-interactivity requirement. Bilinear HSS, in contrast, captures an "HSS-like" syntax, where a trusted setup process distributes keys to the parties. Succinct NIM follows straightforwardly from Bilinear HSS with (1) Strong Hasher Privacy, (2) Strong Matrix Privacy, (3) Transparent Hasher Privacy, and (4) Transparent Matrix Privacy, using the terminology and definitions from [ARS24]. However, Property (4) is not defined in [ARS24], even though we believe that it can be easily be inferred as a variant of (3).*

4.4.3 Constructions from group-based assumptions

Our group-based construction of succinct NIM follows from the construction of succinct non-interactive VOLE protocols [ARS24, BCM+24]. Let m be an integer. Let $\mathbb{G}$ be a suitable finite-order group with generators $h_0, h_1, \ldots, h_m$ and let g be a generator for a subgroup of $\mathbb{G}$ with order p (not necessarily prime). The CRS consists of the group $\mathbb{G}$ and the generators $\mathsf{crs} := (g, h_0, h_1, \ldots, h_m)$, sampled according to some distribution we will define later. The first step in realizing matrix products is non-interactively computing the *inner-product* between two vectors [CZ22, ARS24, BCM+24]. To achieve this, we start with the construction from [ARS24]: Alice has an input vector $\mathbf{a} \in \mathbb{Z}_p^m$ and Bob has an input vector $\mathbf{b} \in \mathbb{Z}_p^m$. The goal is for the parties to obtain shares of the inner product $\langle \mathbf{a}, \mathbf{b} \rangle$ in one simultaneous round of communication. To achieve this, Alice encodes her input vector

$\mathbf{a} = (a_1, \ldots, a_m)$ by computing a Pedersen commitment $d := h_0^r \prod_{i=1}^m h_i^{a_i}$, where r is uniformly random.[3] Bob encodes his input vector $\mathbf{b}$ by computing a "batched" ElGamal-like encryption $E := (h_0^s, g^{b_1} h_1^s, \ldots, g^{b_m} h_m^s)$, where s is uniformly random. Surprisingly, just these values are enough for Alice and Bob to obtain shares of the inner product. In particular, given Bob's public encoding $E = (e_0, e_1, \ldots, e_m)$, Alice computes:

$$Z_A := e_0^r \cdot \prod_{i=1}^m e_i^{a_i} = (h_0^{sr} \cdot g^{b_1 a_1} h_1^{sa_1} \cdots g^{b_m a_m} h_m^{sa_m}) = g^{\langle \mathbf{a}, \mathbf{b} \rangle} \cdot h_0^{sr} \cdot \prod_{i=1}^m h_i^{sa_i},$$

and given Alice's public encoding d, Bob computes:

$$Z_B := d^s = h_0^{rs} (\prod_{i=1}^m h_i^{a_i})^s = h_0^{sr} \cdot \prod_{i=1}^m h_i^{sa_i}.$$

Observe that $Z_A \cdot (Z_B)^{-1} = g^{\langle \mathbf{a}, \mathbf{b} \rangle}$, meaning that the parties obtain *multiplicative* shares of the inner-product. To obtain *additive* shares from the multiplicative shares (i.e., to "bring down" the exponent), the parties can use the distributed discrete logarithm (DDLog) procedure [BGI16] (see also Definition 4.3.5). However, an algorithm for solving the DDLog in an *arbitrary* group $\mathbb{G}$ requires tolerating a polynomial correctness error [DKK20], which is undesirable. Fortunately, however, in was shown in [ADOS22] that for certain instantiations of $\mathbb{G}$, the DDLog procedure can be made "error free" provided that g is chosen to be a generator of a subgroup in which the discrete logarithm is easy.[4] Using the DDLog algorithm, and choosing g according to the non-interactive discrete logarithm sharing (NIDLS) framework of Abram et al. [ADOS22], Alice and Bob can non-interactively derive additive shares of $\langle \mathbf{a}, \mathbf{b} \rangle$, as required.

Succinct matrix products from inner products. Using a non-interactive protocol for inner-products we can clearly construct NIM for matrix multiplication by invoking the inner-product protocol in parallel. In particular, simultaneous round protocols have the appealing feature that the first messages can be reused, meaning that we can "mix-and-match" encodings of different vectors generated by Alice and Bob.

Thus, it is enough for Alice to encode her matrix $\mathbf{A} \in \mathbb{Z}_p^{\ell \times m}$ by generating a commitment d_i using randomness r_i for the row vector $\mathbf{a}_i$, for each $i \in [\ell]$. The randomness masks the entries of $\mathbf{A}$, so Alice's public encoding does not leak information. Likewise, Bob encodes his matrix $\mathbf{B} \in \mathbb{Z}_p^{m \times k}$ by encrypting the column vectors $\mathbf{b}_i$, for each $i \in [k]$. These encryptions hide the entries of $\mathbf{B}$ by the semantic security of the NIDLS-ElGamal encryption, so Bob's public encoding also does not leak information.

This allows Alice and Bob to then non-interactively compute shares of the product $\mathbf{AB}$ by locally computing the inner-products between the $\mathbf{a}_i$'s and $\mathbf{b}_j$'s. We refer to Figure 4.2 for a formal construction of the succinct NIM for matrix multiplication from group-based assumptions.

As observed in [ARS24, BCM+24], the above protocol has a special property that enables a *communication-succinct* variant for certain parameters of ℓ, m, and k: Alice's message is of

[3] Note that Alice's encoding is information-theoretically hiding, regardless of how the CRS was generated.

[4] Examples of such groups include the Paillier group $\mathbb{Z}_{n^2}^*$ and the group of quadratic residues [ADOS22].

size $O(\ell)$ and is entirely independent of m! Moreover, Bob's message only depends on m and k and is independent of ℓ.

In particular, when the size of the output matrix is $N = \ell \cdot k$, there exists a sublinear protocol (in N) by letting $\ell = N^{2/3}$ and choosing m such that $m \cdot k = N^{2/3}$. To see this, note that Alice's encoding is of size $\ell = N^{2/3}$ while Bob's encoding is of size $m \cdot k = N^{2/3}$, resulting in an optimal communication tradeoff with respect to N. This "balancing trick" works for any matrices $\mathbf{A}$ and $\mathbf{B}$, but has previously only been used in realizing succinct VOLE [ARS24, BCM+24]. Our construction of non-interactive DPFs (Section 4.5) is the first to exploit this property explicitly for general matrix multiplication, which proves that this technique may be of independent interest.

Lemma 4.4.1 (Extended from [ARS24]). *Let λ be a security parameter, $\mathcal{R}$ be a finite ring (as defined below), ℓ, m be integer parameters, and $N = \ell \cdot m$. There exist the following instantiations of succinct NIM with $O(N^{2/3})$ encoding size, $O(N)$ encoding time, and $O(N^{4/3})$ decoding time, for all matrices $\mathbf{A} \in \mathcal{R}^{\ell \times m}$ and $\mathbf{B} \in \mathcal{R}^{m \times m}$, where $O(\cdot)$ hides a factor of $\mathsf{poly}(\lambda, \log |\mathcal{R}|)$:*

(1) under the DCR assumption over the Paillier group $\mathbb{Z}^*_{n^2}$, when $\mathcal{R} \subseteq \mathbb{Z}_n$;

(2) under the QR assumption over the RSA group $\mathbb{Z}^*_n$ where n is the product of two large safe-primes, when $\mathcal{R} = \mathbb{Z}_2$;

(3) under the $N^{1/3}$-ary EDDH assumption and the uniformity assumption in class groups, when $\mathcal{R} = \mathbb{Z}_p$, for any suitable prime $p = \Omega(2^\lambda)$; and

(4) under the DDH assumption in a cyclic group $\mathbb{G}$ of order p when $\mathcal{R} = \mathbb{Z}_p$ and when the negligible correctness error of Definition 4.4.1 is relaxed to $1/T^2$, for $T = T(\lambda) \in \mathsf{poly}(\lambda)$, where the Decode is allowed to run in time $O(T)$.

4.4.3.1 Multiplicative-output NIM from DDH

We observe that NIM (with multiplicative output reconstruction; Definition 4.4.2) can be instantiated under the DDH assumption over a suitable cyclic group $\mathbb{G}$ with generators g and $h_1, \ldots, h_m$. To see this, observe that if we forego the DDLog procedure in the overview above, then the parties *already* obtain multiplicative shares Z_A and Z_B such that $Z_A / Z_B = g^{\langle \mathbf{a}, \mathbf{b} \rangle}$. This then carries through to the succinct NIM instantiation via the balancing trick. We will use this variant of NIM in Section 4.5 to realize a NIDPF scheme in bilinear groups under the SXDH assumption (an analog of the DDH assumption for bilinear groups).

4.4.4 Constructions from lattice-based assumptions

The LWE and RLWE constructions follow a similar template to the NIDLS-based approach. The idea, first described in [ARS24], is to replace the Pedersen commitment computed by Alice with an SIS-based commitment[5] and replace the ElGamal encryption computed by Bob with a dual-Regev variant.

For LWE parameters (n, q, χ) and plaintext space $\mathbb{Z}_p$ for $p \ll q$, Alice and Bob encode their input vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^m$ as elements of $\mathbb{Z}_q^m$, in the natural way. For an integer $t \gg n$,

[5]The Short Integer Solution (SIS) problem [Ajt96] is implied by LWE.

Succinct NIM for Matrix Multiplication from Groups

Public Parameters. Matrix dimensions ℓ, m, k. NIDLS framework $(\mathsf{GGen}, \mathcal{D}, \mathsf{DDLog})$.

$\mathsf{NIM.Setup}(1^\lambda)$:

$\mathsf{crs}_{\mathbb{G}} \leftarrow \mathsf{GGen}(1^\lambda)$
foreach $i \in \{0, \ldots, m\}$:
  $(h_i, \rho) \leftarrow \mathcal{D}(1^\lambda, \mathsf{crs}_{\mathbb{G}})$
return $\mathsf{crs} := (\mathsf{crs}_{\mathbb{G}}, h_0, \ldots, h_m)$

$\mathsf{NIM.Encode}_A(\mathsf{crs}, \mathbf{A})$:

parse $\mathsf{crs} = (\mathsf{crs}_{\mathbb{G}}, h_0, \ldots, h_m)$
foreach $i \in [\ell]$:
  $r_i \leftarrow\$ [t]$
  $d_i := h_0^{r_i} \prod_{j=1}^{m} h_j^{a_{i,j}}$
$\mathsf{pe}_A := (d_1, \ldots, d_\ell)$
$\mathsf{st}_A := (\mathbf{A}, r_1, \ldots, r_\ell)$
return $(\mathsf{pe}_A, \mathsf{st}_A)$

$\mathsf{NIM.Encode}_B(\mathsf{crs}, \mathbf{B})$:

parse $\mathsf{crs} = (\mathsf{crs}_{\mathbb{G}}, h_0, \ldots, h_m)$
foreach $i \in [k]$:
  $s_i \leftarrow\$ [t]$
  $E_{i,0} := h_0^{s_i}$
  foreach $j \in [m]$: $E_{i,j} = g^{b_{j,i}} h_j^{s_i}$
$\mathsf{pe}_B := (E_{i,0}, \ldots, E_{i,m})_{i \in [k]}$
$\mathsf{st}_B := (s_1, \ldots, s_k)$
return $(\mathsf{pe}_B, \mathsf{st}_B)$

$\mathsf{NIM.Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A)$:

parse $\mathsf{crs} = (\mathsf{crs}_{\mathbb{G}}, h_0, \ldots, h_m)$
parse $\mathsf{pe}_B = (E_{j,0}, \ldots, E_{j,m})_{j \in [k]}$
parse $\mathsf{st}_A = (\mathbf{A}, r_1, \ldots, r_\ell)$
foreach $(i, j) \in [\ell] \times [k]$:
  $Z_{i,j} := E_{j,0}^{r_i} \cdot \prod_{j'=1}^{m} E_{j,j'}^{a_{i,j'}}$
  $\mathbf{Z}_A[i, j] := \mathsf{DDLog}(\mathsf{crs}_{\mathbb{G}}, Z_{i,j})$
return $\mathbf{Z}_A$

$\mathsf{NIM.Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B)$:

parse $\mathsf{crs} = (\mathsf{crs}_{\mathbb{G}}, h_0, \ldots, h_m)$
parse $\mathsf{pe}_A = (d_1, \ldots, d_\ell)$
parse $\mathsf{st}_B = (s_1, \ldots, s_k)$
foreach $(i, j) \in [\ell] \times [k]$:
  $D_{i,j} := d_i^{s_j}$
  $\mathbf{Z}_B[i, j] := \mathsf{DDLog}(\mathsf{crs}_{\mathbb{G}}, D_{i,j})$
return $\mathbf{Z}_B$

Figure 4.2: Group-based NIM construction.

determined by the SIS problem, let $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{U} \in \mathbb{Z}_q^{n \times t}$ be matrices. The CRS consists of the LWE parameters, and matrices $\mathbf{V}$ and $\mathbf{U}$.

Then, Alice computes a commitment $\mathbf{d} := \mathbf{U}\mathbf{r} + \mathbf{V}\mathbf{a}$, where $\mathbf{r} \in \mathbb{Z}_q^t$ is a random vector from a short distribution that she saves as her secret state. The ElGamal encryption computed by Bob is replaced with a dual-Regev encryption variant of the form: $(\mathbf{e}_0, \mathbf{e}_1)$ where

$$\mathbf{e}_0 := \mathbf{V}^\top \mathbf{s} + \lfloor q/p \rceil \, \mathbf{b} + \mathsf{noise} \quad \text{and} \quad \mathbf{e}_1 := \mathbf{U}^\top \mathbf{s} + \mathsf{noise},$$

for a secret $\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ that he saves as his secret state. As before, Alice and Bob publish $\mathbf{d}$ and $\mathbf{e} := (\mathbf{e}_0, \mathbf{e}_1)$. Again, it holds that $\mathbf{d}$ is of size n (the LWE security parameter), which is independent of the vector length m. For correctness, it is enough to note that:

$$\begin{aligned}
(\mathbf{e}_0^\top \cdot \mathbf{a} + \mathbf{e}_1^\top \cdot \mathbf{r}) - (\mathbf{s}^\top \cdot \mathbf{d}) &= (\mathbf{s}^\top \cdot \mathbf{V} + \lfloor q/p \rceil \cdot \mathbf{b}^\top + \mathsf{noise}) \cdot \mathbf{a} + (\mathbf{s}^\top \cdot \mathbf{U} + \mathsf{noise}) \cdot \mathbf{r} \\
&\quad - \mathbf{s}^\top \cdot (\mathbf{U}\mathbf{r} + \mathbf{V}\mathbf{a}) \\
&= \lfloor q/p \rceil \cdot \mathbf{b}^\top \cdot \mathbf{a} + (\mathsf{noise}) \cdot \mathbf{a} + (\mathsf{noise}) \cdot \mathbf{r} \\
&= \lfloor q/p \rceil \cdot \mathbf{b}^\top \cdot \mathbf{a} + \mathsf{noise} \\
&\approx \lfloor q/p \rceil \cdot \langle \mathbf{a}, \mathbf{b} \rangle \bmod p.
\end{aligned}$$

Using the rounding lemma [DHRW16, BKS19], it holds that for sufficiently large q relative to p, we can locally round the shares such that

$$\Pr\Big[\ \big\lfloor (\mathbf{e}_0^\top \cdot \mathbf{a} + \mathbf{e}_1^\top \cdot \mathbf{r}) \big\rceil_p - \big\lfloor (\mathbf{s}^\top \cdot \mathbf{d}) \big\rceil_p = \big\lfloor (\mathbf{e}_0^\top \cdot \mathbf{a} + \mathbf{e}_1^\top \cdot \mathbf{r}) - (\mathbf{s}^\top \cdot \mathbf{d}) \big\rceil_p \ \Big] \geq 1 - \mathsf{negl}(\lambda),$$

where $\lfloor \cdot \rceil_p$ denotes the modular rounding from $\mathbb{Z}_q$ to $\mathbb{Z}_p$. To ensure security for Bob, we require that the LWE assumption holds for $\mathbf{V}$ and $\mathbf{U}$, i.e., $\mathbf{V}^\top \mathbf{s} + \mathsf{noise}$ and $\mathbf{U}^\top \mathbf{s} + \mathsf{noise}$ look uniform, so Bob's public encoding $(\mathbf{e}_0, \mathbf{e}_1)$ computationally hides his secret input $\mathbf{b}$. To ensure security for Alice, we require that SIS holds for $\mathbf{U}$ (which follows from a hybrid argument), i.e., $\mathbf{U}\mathbf{r}$ looks uniform, so Alice's public encoding $\mathbf{d}$ computationally hides her secret input $\mathbf{a}$. We refer to Figure 4.3 for a formal description of the succinct NIM for matrix multiplication from LWE. The security of the full construction follows from the security of the dot product construction and a standard hybrid argument. We note that the construction can be equivalently instantiated from the Ring LWE (RLWE) assumption by replacing the matrices with a suitable polynomial ring.

Remark 21. *We note the the non-interactive VOLE protocol from Chapter 3 can be easily extended to support inner products (and even batch OLE by working over a spitting ring) while still remaining non-interactive. However, the key difference with the construction from Chapter 3 is that both Alice's and Bob's encodings are computationally hiding whereas the construction described here has Alice's encoding be a statistically hiding commitment. Having one input be statistically hiding is important for achieving input succinctness: otherwise the encodings must grow linearly with the input size.*

Succinct NIM for Matrix Multiplication from Lattices

Public Parameters. Matrix dimensions ℓ, m, k. Plaintext space $\mathbb{Z}_p$. LWE parameters n and $q \gg p$, error distribution χ. SIS parameter $t \gg n$, short distribution $\mathcal{B}$.

Notation. We write $\mathbf{B}[:, i]$ to denote the ith column of the matrix $\mathbf{B}$.

$\mathsf{NIM.Setup}(1^\lambda)$:

$\mathbf{V} \leftarrow\$ \mathbb{Z}_q^{n \times m}$
$\mathbf{U} \leftarrow\$ \mathbb{Z}_q^{n \times t}$
return $\mathsf{crs} := (\mathbf{V}, \mathbf{U})$

$\mathsf{NIM.Encode}_A(\mathsf{crs}, \mathbf{A})$:

parse $\mathsf{crs} = (\mathbf{V}, \mathbf{U})$
foreach $i \in [\ell]$:
  $\mathbf{r}_i \leftarrow\$ \mathcal{B}^t$
  $\mathbf{d}_i := \mathbf{U}\mathbf{r}_i + \mathbf{V}\mathbf{A}[i,]$
$\mathsf{pe}_A := (\mathbf{d}_1, \ldots, \mathbf{d}_\ell)$
$\mathsf{st}_A := (\mathbf{A}, \mathbf{r}_1, \ldots, \mathbf{r}_\ell)$
return $(\mathsf{pe}_A, \mathsf{st}_A)$

$\mathsf{NIM.Encode}_B(\mathsf{crs}, \mathbf{B})$:

parse $\mathsf{crs} = (\mathbf{V}, \mathbf{U})$
foreach $i \in [k]$:
  $\mathbf{s}_i \leftarrow\$ \mathbb{Z}_q^n,\ \mathbf{w}_i, \mathbf{w}'_i \leftarrow\$ \chi^m$
  $\mathbf{e}_{i,0} := \mathbf{V}^\top \mathbf{s}_i + \lfloor q/p \rceil\, \mathbf{B}[:, i] + \mathbf{w}_i$
  $\mathbf{e}_{i,1} := \mathbf{U}^\top \mathbf{s}_i + \mathbf{w}'_i$
$\mathsf{pe}_B := (\mathbf{e}_{i,0}, \mathbf{e}_{i,1})_{i \in [k]}$
$\mathsf{st}_B := (\mathbf{s}_1, \ldots, \mathbf{s}_k)$
return $(\mathsf{pe}_B, \mathsf{st}_B)$

$\mathsf{NIM.Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A)$:

parse $\mathsf{crs} = (\mathbf{V}, \mathbf{U})$
parse $\mathsf{pe}_B = (\mathbf{e}_{i,0}, \mathbf{e}_{i,1})_{j \in [k]}$
parse $\mathsf{st}_A = (\mathbf{A}, \mathbf{r}_1, \ldots, \mathbf{r}_\ell)$
foreach $(i, j) \in [\ell] \times [k]$:
  $\mathbf{Z}_A[i, j] := \mathbf{e}_{j,0}^\top \cdot \mathbf{A}[i,] + \mathbf{e}_{j,1}^\top \cdot \mathbf{r}_i$
return $\mathbf{Z}_A$

$\mathsf{NIM.Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B)$:

parse $\mathsf{crs} = (\mathbf{V}, \mathbf{U})$
parse $\mathsf{pe}_A = (\mathbf{d}_1, \ldots, \mathbf{d}_\ell)$
parse $\mathsf{st}_B = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$
foreach $(i, j) \in [\ell] \times [k]$:
  $\mathbf{Z}_B[i, j] := \mathbf{s}_j^\top \cdot \mathbf{d}_i$
return $\mathbf{Z}_B$

Figure 4.3: Lattice-based NIM construction.

4.5 Non-Interactive DPF

In this section, we define and construct Non-Interactive DPFs (NIDPFs). Our construction makes use of the NIM abstraction and constructions from Section 4.4.

Definition 4.5.1 (Non-Interactive Distributed Point Function)**.** *Let λ be a security parameter and $\mathbb{G}$ be a cyclic group. A non-interactive distributed point function (NIDPF) with input domain $\mathbb{Z}_N$ consists of four efficient algorithms,*

$$\mathsf{NIDPF} = (\mathsf{Setup}, (\mathsf{Gen}_\sigma, \mathsf{KeyDer}_\sigma, \mathsf{Eval}_\sigma)_{\sigma\in\{A,B\}}),$$

with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter and outputs a common reference string* crs.
- $\mathsf{Gen}_\sigma(\mathsf{crs}, t_\sigma, v_\sigma) \to (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$. *The randomized generation algorithm is parameterized by a party identifier $\sigma \in \{A, B\}$. It takes as input the* crs*, an index $t_\sigma \in \mathbb{Z}_N$, and a message $v_\sigma \in \mathbb{G}$. It outputs a public key* pk_σ *and secret key* sk_σ.
- $\mathsf{KeyDer}_\sigma(\mathsf{crs}, \mathsf{pk}_{\overline{\sigma}}, \mathsf{sk}_\sigma) \to \kappa_\sigma$. *The deterministic key derivation algorithm is parameterized by a party identifier $\sigma \in \{A, B\}$. It takes as input the* crs*, public key* $\mathsf{pk}_{\overline{\sigma}}$ *belonging to the other party, and a secret key* sk_σ *belonging to party σ. It outputs a DPF key κ_σ for party σ.*
- $\mathsf{Eval}_\sigma(\mathsf{crs}, \kappa_\sigma, x) \to \langle y\rangle_\sigma$. *The deterministic evaluation algorithm is parameterized by a party identifier $\sigma \in \{A, B\}$. It takes as input the* crs*, the party's DPF key κ_σ, and an input $x \in \mathbb{Z}_N$. It outputs a share of the evaluation result y.*

Let $P_{i,v} \colon \mathbb{Z}_N \to \mathbb{G}$ be the point function that outputs 0 for all inputs $x \neq i$ and outputs v otherwise. The above functionality must satisfy the following correctness and security properties:

Correctness. *For all security parameters $\lambda \in \mathbb{N}$, every pair of indices $t_A, t_B \in \mathbb{Z}_N$ such that $t = t_A + t_B \in \mathbb{Z}_N$, every pair of messages $v_A, v_B \in \mathbb{G}$ such that $v = v_A + v_B \in \mathbb{G}$, and every input $x \in \mathbb{Z}_N$, a NIDPF scheme is correct if there exists a negligible function* negl *such that:*

$$\Pr\left[\begin{array}{rl} & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{Gen}_\sigma(\mathsf{crs}, t_\sigma, v_\sigma),\ \forall\sigma \in \{A, B\} \\ \langle y\rangle_A - \langle y\rangle_B = P_{t,v}(x) \quad : & \kappa_A := \mathsf{KeyDer}_A(\mathsf{crs}, \mathsf{pk}_B, \mathsf{sk}_A) \\ & \kappa_B := \mathsf{KeyDer}_B(\mathsf{crs}, \mathsf{pk}_A, \mathsf{sk}_B) \\ & \langle y\rangle_\sigma := \mathsf{Eval}_\sigma(\mathsf{crs}, \kappa_\sigma, x),\ \forall\sigma \in \{A, B\} \end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins used by Gen.

Security. *For all efficient adversaries $\mathcal{A}$, for all $\sigma \in \{A, B\}$, there exists a negligible function*

$\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{rcl} & & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & & (t_0, v_0, t_1, v_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b = b' & : & b \xleftarrow{R} \{0,1\} \\ & & (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{Gen}_\sigma(\mathsf{crs}, t_b, v_b) \\ & & b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pk}_\sigma) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

In words, the public key computationally hides the encoded index and message.

We now define two variants of NIDPF that we will consider in this chapter. The first variant, which we call a *half-chosen* NIDPF, only allows one of the parties to specify the output message, forcing the second party to input $\perp$ to Gen. The second variant, which we call a *random-output* NIDPF, does not allow the parties to specify the output message: the output v is uniformly random and determined solely based on the random coins of Gen_A and Gen_B.

Definition 4.5.2 (Half-Chosen NIDPF)**.** *We say that a NIDPF scheme has a* half-chosen payload *if for a fixed $\sigma \in \{A, B\}$, Gen_σ only accepts $v_\sigma = 0$.*

Definition 4.5.3 (Random-Payload NIDPF)**.** *We say that a NIDPF scheme has a* random payload *if both Gen_A and Gen_B do not take any message parameter, and the NIDPF correctness property from Definition 4.5.1 instead holds with respect to a random payload $v \in \mathbb{G}$ (determined by the random coins of Gen).*

Lemma 4.5.1 (Half-Chosen NIDPF $\implies$ NIDPF)**.** *Given a half-chosen NIDPF with (public and secret) encoding size S and evaluation time T, we can obtain a NIDPF with encoding size size $2S$ and evaluation time $2T$.*

The lemma follows directly from the composition theorem of function secret sharing [BGI15, Section 3.2]. In particular, the parties run two instances of the half-chosen NIDPF in parallel, where each party specifies its own payload in turn by reversing roles, then the outputs of the two instances are summed together.

Remark 22 (Full-domain evaluation)**.** *Our construction will focus on settings where the evaluation algorithm Eval is applied on all inputs in the domain $\mathbb{Z}_N$ (in which case we need to assume that N is polynomial in the security parameter, for efficiency). That is, given a NIDPF scheme $\mathsf{NIDPF} = (\mathsf{Setup}, (\mathsf{Gen}_\sigma, \mathsf{KeyDer}_\sigma, \mathsf{Eval}_\sigma)_{\sigma \in \{A,B\}})$, we will denote by $\mathsf{EvalAll}_\sigma$ the algorithm that runs $\mathsf{NIDPF.Eval}_\sigma$ on every input $x \in \mathbb{G}$. As such, $\mathsf{EvalAll}_\sigma$ only takes as input the crs and key κ_σ.*

This setting captures a motivated range of applications and implemented systems, including constructions of pseudorandom correlation generators, private "reading" applications such as Private Information Retrieval, and private "writing" applications such as secure distributed storage, voting, and aggregation. (See, e.g., [BGIK22] for further discussion.)

We additionally remark that all present black-box distributed DPF setup protocols require domains of feasible size. Indeed, removing this limitation while remaining black-box in the underlying cryptography would seem to pose a significant challenge.

4.5.1 Emulating arithmetic modulo N

Here, we briefly describe two natural approaches for representing arithmetic over the inputs of Alice and Bob. We want our construction to give parties the ability to generate keys for the point function with a non-zero index at $t_A + t_B \bmod N$. Unfortunately, our NIDPF construction requires Alice and Bob to parse their inputs $t_A, t_B \in [N]$ as $(i_A, j_A), (i_B, j_B) \in \mathbb{Z}_\ell \times \mathbb{Z}_m$, where $N = \ell \cdot m$ and only allows them to compute a DPF key encoding a point function with special index:

$$(i_A + i_B \bmod \ell) \cdot m + (j_A + j_B \bmod m). \tag{4.3}$$

If we parse the indices t_A and t_B of each party in the simplest way, i.e., $t_A = i_A \cdot m + j_A$ and $t_B = i_B \cdot m + j_B$, the above operation does not capture addition of t_A and t_B modulo N. Specifically, it is possible that $j_A + j_B$ has a "carry bit" b in the case when $j_A + j_B \geq m$, which then has to be added to the $i_A + i_B$ component as:

$$(i_A + i_B + b \bmod \ell) \cdot m + (j_A + j_B \bmod m). \tag{4.4}$$

Concretely, in our NIDPF construction, this will require shifting the rows of the matrix $\mathbf{T}$ by $i_B + b$ in the case that the carry bit is set (recall Step II from Section 4.2.2).

Remark 23 (Random point function)**.** *We remark that in the case that Alice and Bob need a point function with a* random *non-zero index, they do not need to emulate addition modulo N and can instead simply sample uniformly random (i_σ, j_σ) for their inputs to the NIDPF.*

Here, we present two approaches for emulating addition (modulo N) using arithmetic represented over ℓ and m, as in Equation 4.3.

Approach I: Emulating arithmetic via a residue number system. As described in Section 4.2, we can let ℓ be coprime to m, which immediately allows us to emulate arithmetic modulo N in a residue number system, using ℓ and m as the coprime moduli. In this case, we no longer need to worry about the carry bit, since we can compute locally modulo $\mathbb{Z}_\ell$ and $\mathbb{Z}_m$ and then map back to $\mathbb{Z}_N$. Alice and Bob represent their indices $t_A, t_B \in [N]$ as (i_A, j_A) and (i_B, j_B) where

$$\begin{aligned} t_A &\equiv i_A \pmod{\ell}, & t_B &\equiv i_B \pmod{\ell}, \\ t_A &\equiv j_A \pmod{m}, & t_B &\equiv j_B \pmod{m}. \end{aligned}$$

After executing the protocol, they hold secret shares of a matrix that is nonzero in location $(i_A + i_B \pmod{\ell}, j_A + j_B \pmod{m})$. Let α, α' be integers such that $\alpha\ell + \alpha' m = 1$, which exist since ℓ and m are coprime. Alice and Bob can each map location (i, j) of their $\ell \times m$ matrix to location $l \in [N]$ in a vector of length N where $l \equiv i\alpha' m + j\alpha\ell \pmod{N}$. Suppose t is the resulting one-hot index in the vector Alice and Bob now hold shares for. By the Chinese Remainder Theorem, we have that

$$\left.\begin{aligned} t &\equiv i_A + i_B \pmod{\ell} \\ t &\equiv j_A + j_B \pmod{m} \end{aligned}\right\} \implies t \equiv t_A + t_B \pmod{N}.$$

Approach II: Emulating the carry. For some applications, it may be inconvenient or impossible for ℓ and m to be coprime, such as if $N = \ell \cdot m$ is a power of 2. An alternative strategy we can use in this case is to prevent the "erasure" of the carry bit modulo m. Specifically, we observe that if the cyclic shift is performed modulo $2m$, then we do not lose information on the carry: the non-zero entry of Alice's matrix $\mathbf{A} \in \mathcal{R}^{\ell \times 2m}$ will either contain the non-zero entry in the "left half" or the "right half" of the columns depending on whether or not the carry occurred. At this point, Alice and Bob will hold shares of a matrix $\mathbf{T}$ that can be parsed as $[\mathbf{T}_0\ \mathbf{T}_1]$, where $\mathbf{T}_0, \mathbf{T}_1 \in \mathcal{R}^{\ell \times m}$ and $\mathbf{T}_{1-b} = \mathbf{0}$ when b is the value of the carry bit. Then, Alice and Bob can cyclically shift their rows of $\mathbf{T}_1$ down by one (this operation is a linear function over their shares of $\mathbf{T}_1$), and compute $\mathbf{T} := \mathbf{T}_0 + \mathsf{ShiftDown}(\mathbf{T}_1, 1)$. Observe that because $\mathbf{T}_{1-b}$ is always zero, they obtain shares of the matrix $\mathbf{T}$ that has exactly one non-zero entry and the rows cyclically shifted down precisely if the carry bit is set. Note that this approach works regardless of the choice of ℓ and m.

4.5.2 NIDPF framework

Here, we formalize the NIDPF construction, closely following the technical overview from Section 4.2.2. We present a construction for the "half-chosen payload" (Definition 4.5.2) NIDPF in Figure 4.4, which can be extended to satisfy the full NIDPF definition via Lemma 4.5.1 (however, for the applications described in Section 4.1.2, the payload is public, and so the half-chosen variant sufficient on its own).

Our construction uses the following auxiliary functions as building blocks.

Auxiliary functions. We define two deterministic functions that simplify the presentation of our NIDPF construction in Figure 4.4.

- $\mathsf{Shift}\colon \mathcal{R}^{\ell \times m} \times [\ell] \to \mathcal{R}^{\ell \times m}$. Shift takes as input a $\ell \times m$ matrix (for arbitrary integers ℓ, m) and a shift $i \in [\ell]$. It outputs the matrix with the rows cyclically shifted down by i.
- $\mathsf{Mat2Vec}\colon \mathcal{R}^{\ell \times m} \to \mathcal{R}^{\ell \cdot m}$. $\mathsf{Mat2Vec}$ takes as input a $\ell \times m$ matrix (for arbitrary integers ℓ, m) and outputs the vector obtained by concatenating the rows of the matrix together.

Proposition 4.5.1. *Let* NIM *be a succinct NIM scheme (Definition 4.4.3) and let* HSS *be a (degree-2, secret-key) HSS scheme (Definition 4.3.8). The construction presented in Figure 4.4 is a half-chosen NIDPF (Definition 4.5.2).*

Proof. We prove correctness and security in turn.

Correctness. By correctness of NIM, we have that

$$\mathbf{U}_A + \mathbf{U}_B = \mathbf{AE} = \left[\mathbf{AS}_{j_B} \;\middle|\; \mathsf{sk}_B \cdot \mathbf{AS}_{j_B} \;\middle|\; \cdots \;\middle|\; \mathsf{sk}_k \cdot \mathbf{AS}_{j_B}\right].$$

In words, $\mathbf{U}_\sigma$ is an HSS memory share of $\mathbf{T} = \mathbf{AS}_{j_B}$. Recall that the vector $\mathbf{e}_{i_B}$ has entries $e_{i_B,j} = 0$ for $j \neq i_B$ and $e_{i_B,i_B} = 1$. Then, by correctness of HSS, we have that

$$\begin{aligned}\mathbf{Z}_A - \mathbf{Z}_B &= \sum_{j \in [\ell]} (\mathbf{Z}_A^{(j)} - \mathbf{Z}_B^{(j)}) = (\mathbf{Z}_A^{(i_B)} - \mathbf{Z}_B^{(i_B)}) + \mathbf{0} \\ &= \mathbf{T}_{i_B} = \mathsf{Shift}(\mathbf{T}, i_B) = \mathsf{Shift}(\mathbf{AS}_{j_B}, i_B).\end{aligned}$$

NIDPF Framework

Public Parameters. Domain size N and matrix dimensions ℓ, m such that $\ell \cdot m = N$. Set of cyclic shift matrices $\mathcal{S}_m = \{\mathbf{S}_j : j \in [m]\}$, where $\mathbf{S}_j$ is the j-th canonical cyclic shift matrix. Succinct NIM scheme $\mathsf{NIM} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma\in\{A,B\}})$ and (degree-2, secret-key) HSS scheme $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Convert}, \mathsf{Mult})$. We instantiate the full-domain evaluation algorithm $\mathsf{NIDPF.EvalAll}$, described in Remark 22.

$\mathsf{NIDPF.Setup}(1^\lambda)$:

1 : $\mathsf{crs} \leftarrow \mathsf{NIM.Setup}(1^\lambda)$
2 : **return** crs

$\mathsf{NIDPF.Gen}_A(\mathsf{crs}, t_A, v)$:

1 : **parse** $t_A = (i_A, j_A) \in [\ell] \times [m]$
2 : $\mathbf{A} := \mathbf{0} \in \mathcal{R}^{\ell\times m},\ \mathbf{A}[i_A, j_A] := v$
3 : $(\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, \mathbf{A})$
4 : $(\mathsf{pk}_A, \mathsf{sk}_A) := (\mathsf{pe}_A, \mathsf{st}_A)$
5 : **return** $(\mathsf{pk}_A, \mathsf{sk}_A)$

$\mathsf{NIDPF.KeyDer}_\sigma(\mathsf{crs}, \mathsf{pk}_{\overline{\sigma}}, \mathsf{sk}_\sigma)$:

1 : **parse** $(\mathsf{pk}_{\overline{\sigma}}, \mathsf{sk}_\sigma) = (\mathsf{pe}_{\overline{\sigma}}, \mathsf{ek}_\sigma, [\![\mathbf{e}_{i_B}]\!]_\sigma, \mathsf{st}_\sigma)$
2 : $\mathbf{U}_\sigma := \mathsf{Decode}_\sigma(\mathsf{pe}_{\overline{\sigma}}, \mathsf{st}_\sigma)$
3 : **parse** $\mathbf{U}_\sigma = \langle\!\langle \mathbf{T} \rangle\!\rangle_\sigma$
4 : $\kappa_\sigma := (\langle\!\langle \mathbf{T} \rangle\!\rangle_\sigma, \mathsf{ek}_\sigma, [\![\mathbf{e}_{i_B}]\!]_\sigma)$
5 : **return** κ_σ

$\mathsf{NIDPF.Gen}_B(\mathsf{crs}, t_B, \bot)$:

1 : **parse** $t_B = (i_B, j_B) \in [\ell] \times [m]$
2 : $(\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$
3 : **parse** $\mathsf{sk} = (\mathsf{sk}_B, \ldots, \mathsf{sk}_k) \in \mathcal{R}^k$
▷ k is determined by the HSS scheme.
4 : $([\![\mathbf{e}_{i_B}]\!]_A, [\![\mathbf{e}_{i_B}]\!]_B) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, \mathbf{e}_{i_B})$
▷ $\mathbf{e}_{i_B}$ is the i_B-th canonical unit vector.
5 : $\mathbf{E} := \left[\mathbf{S}_{j_B} \mid \mathsf{sk}_B \cdot \mathbf{S}_{j_B} \mid \cdots \mid \mathsf{sk}_k \cdot \mathbf{S}_{j_B}\right]$
6 : $(\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, \mathbf{E})$
7 : $\mathsf{pk}_B := (\mathsf{pe}_B, \mathsf{ek}_A, [\![\mathbf{e}_{i_B}]\!]_A)$
8 : $\mathsf{sk}_B := (\mathsf{st}_B, \mathsf{ek}_B, [\![\mathbf{e}_{i_B}]\!]_B)$
9 : **return** $(\mathsf{pk}_B, \mathsf{sk}_B)$

$\mathsf{NIDPF.EvalAll}_\sigma(\mathsf{crs}, \kappa_\sigma)$:

1 : **parse** $\kappa_\sigma = (\langle\!\langle \mathbf{T} \rangle\!\rangle_\sigma, \mathsf{ek}_\sigma, [\![\mathbf{e}_{i_B}]\!]_\sigma)$
▷ $[\![\mathbf{e}_{i_B}]\!]_\sigma = ([\![e_{i_B,1}]\!]_\sigma, \ldots, [\![e_{i_B,k}]\!]_\sigma)$.
2 : **foreach** $j \in [\ell]$:
3 : $\quad \langle\!\langle \mathbf{T}_j \rangle\!\rangle_\sigma := \mathsf{Shift}(\langle\!\langle \mathbf{T} \rangle\!\rangle_\sigma, i)$
4 : $\quad \mathbf{Z}_\sigma^{(j)} := \mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, [\![e_{i_B,j}]\!]_\sigma, \langle\!\langle \mathbf{T}_j \rangle\!\rangle_\sigma)$
5 : $\mathbf{Z}_\sigma := \sum_{j=1}^{\ell} \mathbf{Z}_\sigma^{(j)}$
6 : $\langle \mathbf{y} \rangle_\sigma := \mathsf{Mat2Vec}(\mathbf{Z}_\sigma)$
7 : **return** $\langle \mathbf{y} \rangle_\sigma$

Figure 4.4: NIDPF framework.

$\mathbf{A}$ is zero everywhere except location (i_A, j_A), so $\mathbf{AS}_{j_B}$ is zero everywhere except location $(i_A, j_A + j_B)$, and $\mathsf{Shift}(\mathbf{AS}_{j_B}, i_B)$ is zero everywhere except location $(i_A + i_B, j_A + j_B)$, as desired.

Security. Note that $\mathsf{pk}_A = \mathsf{pe}_A$ where pe_A is a public encoding generated by Encode_A for message $\mathbf{A}$ defined by t_A, v. This computationally hides t_A and v by NIM security. To prove security for $\mathsf{pk}_B = (\mathsf{pe}_B, \mathsf{ek}_A, [\![\mathbf{e}_{i_B}]\!]_A)$, we proceed via a hybrid argument.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the NIDPF game instantiated using the construction from Figure 4.4 with index t_0 and payload v_0 (i.e., when $b = 0$).
- *Hybrid* $\mathcal{H}_1$. In this hybrid, we set $\mathbf{E} := \mathbf{0} \in \mathcal{R}^{m \times m(k+1)}$.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$ *assuming the security of NIM.*

  Proof. Suppose an efficient adversary $\mathcal{A}$ distinguishes between $\mathcal{H}_1$ and $\mathcal{H}_0$. The existence of $\mathcal{A}$ contradicts NIM security: the reduction asks the NIM challenger for pe_B to be either a public encoding of $\mathbf{E}$ or $\mathbf{0}$, samples $(\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$ itself, and uses $\mathcal{A}$ to break NIM security of Encode. □
- *Hybrid* $\mathcal{H}_2$. In this hybrid, we set $\mathbf{e}_{i_B} := \mathbf{0} \in \mathcal{R}^\ell$.

  Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$ *assuming the security of HSS.*

  Proof. Suppose an efficient adversary $\mathcal{A}$ distinguishes between $\mathcal{H}_2$ and $\mathcal{H}_1$. The existence of $\mathcal{A}$ contradicts HSS security: the reduction asks the HSS challenger for pe_B to be either a share of $\mathbf{e}_{i_B}$ or $\mathbf{0}$, samples $\mathsf{crs} \leftarrow \mathsf{NIM.Setup}(1^\lambda)$, computes $\mathsf{pe}_B \leftarrow \mathsf{Encode}_B(\mathsf{crs}, \mathbf{0})$ itself, and finally uses $\mathcal{A}$ to break HSS security of $\mathsf{HSS.Share}$. □

At this point, there is no information on t_0 and v_0. By continuing with the same sequence of hybrids in reverse order, we can show that $\mathcal{H}_0$ is indistinguishable from the NIDPF game instantiated using the construction from Figure 4.4 with index t_1 and payload v_1 (i.e., when $b = 1$). As such, no efficient adversary can distinguish between the $b = 0$ and $b = 1$ case with better than negligible advantage, which concludes the proof of security. ■

Theorem 4.5.1. *There exist the following instantiations of Figure 4.4 with a $O(N^{2/3})$ public key size, $O(N)$ key generation time, $O(N^{4/3})$ key derivation time, and $O(N^{5/3})$ full domain evaluation time, where $O(\cdot)$ hides a factor of $\mathsf{poly}(\lambda, \log|\mathcal{R}|)$:*

- *under the DCR assumption over the Paillier group $\mathbb{Z}^*_{n^2}$, when $\mathcal{R} \subseteq \mathbb{Z}_n$;*
- *under the QR assumption over the RSA group $\mathbb{Z}^*_n$ where n is the product of two large safe-primes, when $\mathcal{R} = \mathbb{Z}_2$;*
- *under the $N^{1/3}$-ary EDDH assumption and the uniformity assumption in class groups, when $\mathcal{R} = \mathbb{Z}_p$, for any suitable prime $p = \Omega(2^\lambda)$; and*
- *under the LWE/RLWE assumption with a superpolynomial modulus-to-noise ratio, when $\mathcal{R} = \mathbb{Z}_p$, for any integer p.*

The class group and LWE/RLWE instantiations have a transparent setup.

Proof. We set parameters $\ell = N^{2/3}$ and $m = N^{1/3}$. Since the HSS secret key length $k = k(\lambda) = \mathsf{poly}(\lambda)$, we will ignore factors of k in our analysis below.

The size of the public key generated for the larger matrix with dimensions $\ell \times m$ grows with the number of rows, resulting in an asymptotic size of $N^{2/3}$. The size of the public key generated for a smaller matrix with dimensions $m \times m$ grows with the size of the matrix, resulting in an asymptotic size of $N^{1/3} \cdot N^{1/3} = N^{2/3}$. Key generation time is dominated by the NIM encoding algorithm for a matrix of size $\ell \times m$, resulting in an asymptotic runtime of $N^{2/3} \cdot N^{1/3} = N$.

Key derivation time is dominated by the NIM decoding algorithm for matrices with dimensions $\ell \times m$ and $m \times m$, resulting in an asymptotic runtime of $N^{2/3} \cdot N^{1/3} \cdot N^{1/3} = N^{4/3}$. Full domain evaluation time is dominated by running the HSS multiplication algorithm ℓ times for a matrix with dimensions $\ell \times m$, resulting in an asymptotic runtime of $N^{2/3} \cdot N^{2/3} \cdot N^{1/3} = N^{5/3}$. ■

4.5.3 Random-payload instantiation from SXDH

Here, we provide a construction of Figure 4.4 with random payload (see Definition 4.5.3) from the SXDH assumption over bilinear groups. Our starting point is the observation that if we replace the NIM in Figure 4.4 with the multiplicative-output NIM (Definition 4.4.2), we can avoid the error introduced from the DDLog procedure converting the multiplicative shares into additive shares. However, by having the output of NIM be multiplicative, we lose the ability to compute the HSS multiplication in $\mathsf{EvalAll}$, since HSS requires additive memory shares.

We overcome this by constructing a new degree-2 HSS scheme satisfying Definition 4.3.8 and which has "multiplicative" memory shares (i.e., additive memory shares "in the exponent") that are compatible with the outputs of the multiplicative NIM.

4.5.3.1 Degree-2 HSS with Multiplicative Memory Shares

In Figure 4.5, we construct a (secret-key, degree-2) HSS scheme satisfying Definition 4.3.8 under the SXDH assumption in bilinear groups.[6] The construction follows the standard template for realizing HSS in cyclic groups. However, one difference is that we define input shares over the group $\mathbb{G}_1$ and memory shares over the group $\mathbb{G}_2$, which allows us to compute the multiplication using a pairing operations. This slightly complicates the scheme since now we need to convert input shares to memory shares using $\mathsf{Convert}$ by "hopping between groups," which necessitates defining two independent encryptions of the message in $\mathsf{HSS.Share}$ when generating an input share. Importantly, the encryptions in $\mathbb{G}_1$ need to be generated using an encryption key α that is independent from the encryption key β used to encrypt the messages in $\mathbb{G}_2$ (otherwise the security of the encryption would be trivially broken via the pairing).

[6] The construction can easily be made *public key*, but we present the secret-key variant for consistency with the rest of the chapter.

Degree-2 Secret-key HSS from SXDH

Public Parameters. Bilinear group of order p defined by $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$. For convenience, we define $g_T := e(g_1, g_2) \in \mathbb{G}_T$. Additive secret sharing algorithm $\mathsf{Share}_{\mathbb{G}}(\cdot)$ outputting two-out-of-two shares in the group $\mathbb{G}$ (see Section 4.3.2). Distributed discrete logarithm algorithm DDLog (Definition 4.3.4) and an integer $2 \leq M \leq \mathsf{poly}(\lambda)$ defining the message space $\{0, 1, \ldots, M-1\}$.

$\mathsf{HSS.Setup}(1^\lambda)$:

1 : $\alpha, \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_p,\ \gamma := \alpha\beta,\ \mathsf{sk} := (\alpha, \beta, \gamma)$
2 : $(\langle\alpha\rangle_A, \langle\alpha\rangle_B) \leftarrow \mathsf{Share}_{\mathbb{Z}_p}(\alpha)$
3 : $(\langle\beta\rangle_A, \langle\beta\rangle_B) \leftarrow \mathsf{Share}_{\mathbb{Z}_p}(\beta)$
4 : $(\langle\gamma\rangle_A, \langle\gamma\rangle_B) \leftarrow \mathsf{Share}_{\mathbb{Z}_p}(\gamma)$
5 : **foreach** $\sigma \in \{A, B\}$:
6 : $\quad \mathsf{ek}_\sigma := (\langle\alpha\rangle_\sigma, \langle\beta\rangle_\sigma, \langle\gamma\rangle_\sigma)$
7 : **return** $(\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B))$

$\mathsf{HSS.Share}(\mathsf{sk}, x)$:

1 : $r, r' \xleftarrow{\mathrm{R}} \mathbb{Z}_p$
2 : **foreach** $\sigma \in \{A, B\}$:
3 : $\quad [\![x]\!]_\sigma := (g_1^r,\ g_1^{x+\alpha\cdot r},\ g_2^{r'},\ g_2^{x+\beta\cdot r'})$
4 : **return** $([\![x]\!]_A, [\![x]\!]_B)$

$\mathsf{HSS.Convert}(\sigma, \mathsf{ek}_\sigma, [\![x]\!]_\sigma)$:

1 : **parse** $\mathsf{ek}_\sigma = (\langle\alpha\rangle_\sigma, \langle\beta\rangle_\sigma, \langle\gamma\rangle_\sigma)$
2 : **parse** $[\![x]\!]_\sigma = (\_, \_, E_0, E_1)$
3 : $\langle\!\langle x \rangle\!\rangle_\sigma := (E_1 \cdot E_0^{-\langle\beta\rangle_\sigma}, E_1^{\langle\alpha\rangle_\sigma} \cdot E_0^{-\langle\gamma\rangle_\sigma})$
4 : **return** $\langle\!\langle x \rangle\!\rangle_\sigma$

$\mathsf{HSS.Mult}(\sigma, \mathsf{ek}_\sigma, [\![x]\!]_\sigma, \langle\!\langle y \rangle\!\rangle_\sigma)$:

1 : **parse** $[\![x]\!]_\sigma = (D_0, D_1, \_, \_)$
2 : **parse** $\langle\!\langle y \rangle\!\rangle_\sigma = (S_\sigma^{(0)}, S_\sigma^{(1)})$
3 : $Z_\sigma := e(D_0, S_\sigma^{(1)}) \cdot e(D_1, S_\sigma^{(0)})$
4 : $\tau := 0$ **if** $\sigma = A$; **else** $\tau := 1$
5 : $\langle z \rangle_\sigma := \mathsf{DDLog}((\mathbb{G}_T, g_T, M), Z_\sigma^{-1^\tau})$
6 : **return** $\langle z \rangle_\sigma$

Figure 4.5: Degree-2 secret-key HSS from SXDH.

Proposition 4.5.2. *The construction presented in Figure 4.5 satisfies Definition 4.3.8 (degree-2, secret-key) HSS with $\epsilon = 1 - 1/\mathsf{poly}(\lambda)$ correctness assuming the SXDH assumption holds in the bilinear group $\mathbb{G} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$.*

Proof. We prove correctness and security in turn.

Correctness. To simplify analysis later on, we show that Convert outputs memory shares that are partially consistent with the template outlined in Section 4.3.6.1. Notice that given an input share $[\![x]\!]_\sigma$ of the form $(g_1^r,\ g_1^{x+\alpha\cdot r},\ g_2^{r'},\ g_2^{x+\beta\cdot r'})$, $\langle\!\langle x\rangle\!\rangle_\sigma$ consists of a tuple of multiplicative shares of $(x, x\cdot\alpha)$ defined over $\mathbb{G}_2$, since

$$E_1 \cdot E_0^{-\langle\beta\rangle_\sigma} = g_2^{x+\beta\cdot r'} \cdot (g_2^{r'})^{-\langle\beta\rangle_\sigma} = g_2^{\langle x\rangle_\sigma} \text{ and}$$
$$E_1^{-\langle\alpha\rangle_\sigma} \cdot E_0^{-\langle\gamma\rangle_\sigma} = (g_2^{x+\beta\cdot r'})^{\langle\alpha\rangle_\sigma} \cdot (g_2^{r'})^{-\langle\alpha\beta\rangle_\sigma} = g_2^{\langle x\cdot\alpha\rangle_\sigma},$$

which follows the template from Section 4.3.6.1, when memory shares are defined multiplicatively in the group $\mathbb{G}_2$ with respect to secret key α. Specifically, we can "ignore" the secret key β, which is only used to convert from input to memory shares and does not aid in decryption when performing HSS.Mult.

Now, we turn to proving correctness, as defined in Definition 4.3.8. First, observe that Z_σ, as computed in HSS.Mult (line 3), is defined as

$$Z_A = e(g_1^r, g_2^{\langle -y\cdot\alpha\rangle_A}) \cdot e(g_1^{x+\alpha\cdot r}, g_2^{\langle y\rangle_A}) = g_T^{\langle -\alpha\cdot(ry)\rangle_A} \cdot g_T^{\langle yx+\alpha\cdot(ry)\rangle_A} = g_T^{\langle xy\rangle_A}$$
$$Z_B = e(g_1^r, g_2^{\langle -y\cdot\alpha\rangle_B}) \cdot e(g_1^{x+\alpha\cdot r}, g_2^{\langle y\rangle_B}) = g_T^{\langle -\alpha\cdot(ry)\rangle_B} \cdot g_T^{\langle yx+\alpha\cdot(ry)\rangle_B} = g_T^{\langle xy\rangle_B}.$$

Therefore, party $\sigma \in \{A, B\}$ obtains Z_σ, which is a multiplicative share of g_T^{xy}. By correctness of the DDLog, the parties obtain an additive share of $z := xy$ with probability $\epsilon := 1 - \mathsf{poly}(\lambda)$. As such, the error of the HSS scheme is also ϵ.

Security. We prove security via a hybrid argument.

- *Hybrid $\mathcal{H}_0$.* This hybrid consists of the HSS security game instantiated using the construction in Figure 4.5 with message $\mathbf{x}_b$.

- *Hybrid $\mathcal{H}_1$.* In this hybrid, we replace the evaluation key ek_σ produced by HSS.Setup (and given to the adversary) with uniformly random value in $\mathbb{Z}_p^3$. It is trivial to verify that this hybrid game is perfectly indistinguishable from $\mathcal{H}_0$.

- *Hybrid $\mathcal{H}_2$.* In this hybrid, we replace the elements of $\mathbb{G}_1$ of each input share produced by HSS.Share with uniformly random group elements in $\mathbb{G}_1$.

  Claim. *$\mathcal{H}_2 \approx_c \mathcal{H}_1$ assuming DDH in $\mathbb{G}_1$.*

  Proof. Notice that in $\mathcal{H}_1$, the adversary $\mathcal{A}$ receives a vector of input shares $[\![\mathbf{x}_b]\!]_\sigma :=$

$(\llbracket x_b^{(1)} \rrbracket_\sigma, \ldots, \llbracket x_b^{(k)} \rrbracket_\sigma)$ distributed as:

$$\left((g_1^{r_1}, g_1^{x_b^{(1)}+\alpha\cdot r_1}, g_2^{r'_1}, g_2^{x_b^{(1)}+\beta\cdot r'_1}), \ldots, (g_1^{r_k}, g_1^{x_b^{(k)}+\alpha\cdot r_k}, g_2^{r'_k}, g_2^{x_b^{(k)}+\beta\cdot r'_k})\right)$$
$$= \left((g_1^{r_1}, g_1^{x_b^{(1)}} h_1^{r_1}, g_2^{r'_1}, g_2^{x_b^{(1)}} h_2^{r'_1}), \ldots, (g_1^{r_k}, g_1^{x_b^{(k)}} h_1^{r_k}, g_2^{r'_k}, g_2^{x_b^{(k)}} h_2^{r'_k})\right),$$

where $h_1 := g_1^\alpha$ and $h_2 := g_2^\beta$. In turn, this is distributed identically to:

$$\left((g_1^{r_1}, \ldots, g_1^{r_k}, g_1^{x_b^{(1)}} h_1^{r_1}, \ldots, g_1^{x_b^{(k)}} h_1^{r_k}), (g_2^{r'_1}, \ldots, g_2^{r'_k}, g_2^{x_b^{(1)}} h_2^{r'_1}, \ldots, g_2^{x_b^{(k)}} h_2^{r'_k})\right),$$

by rearranging the terms. Note that all elements of the input share from $\mathbb{G}_2$ are independent of the $\mathbb{G}_1$ elements (they are computed with different randomness r'_i and using a different secret key β, sampled independently of r_i and α).

Suppose, towards contradiction, that $\mathcal{A}$ distinguishes between $\mathcal{H}_2$ and $\mathcal{H}_1$ with non-negligible advantage. By a separate hybrid argument, it follows that $\mathcal{A}$ distinguishes between some $(g_1^{r_i}, g_1^{x_b^{(i)}} h_1^{r_i})$ and uniformly random $(u_i, v_i) \in \mathbb{G}_1^2$, for some $i \in [k]$ (note that all elements of $\mathbb{G}_2$ in each input share are still computed as in hybrid $\mathcal{H}_1$). This contradicts the DDH assumption in $\mathbb{G}_1$, since by a straightforward reduction, the adversary is able to distinguish between $(g_1^\alpha, g_1^{r_i}, g_1^{\alpha r_i})$ and (g_1^α, u_i, v_i). □

- *Hybrid $\mathcal{H}_3$.* In this hybrid, we replace the elements of $\mathbb{G}_2$ of each input share produced by $\mathsf{HSS.Share}$ with uniformly random group elements in $\mathbb{G}_2$.

  Claim. *$\mathcal{H}_3 \approx_c \mathcal{H}_2$ assuming DDH in $\mathbb{G}_2$.*

  Proof. The proof follows the same strategy as in the proof of the previous claim, except that now the adversary contradicts the DDH assumption in $\mathbb{G}_2$. □

At this point, we have proven that under the SXDH assumption, $\mathcal{A}$ cannot distinguish between $\llbracket \mathbf{x}_b \rrbracket_\sigma$ and uniform tuple over $(\mathbb{G}_1^2 \times \mathbb{G}_2^2)^k$, with better than negligible advantage. In turn, it follows that $\mathcal{A}$'s distinguishing advantage between $\llbracket \mathbf{x}_0 \rrbracket_\sigma$ and $\llbracket \mathbf{x}_1 \rrbracket_\sigma$ is negligible, which concludes the proof. ■

4.5.3.2 Random "DDLog" procedure

In Figure 4.5, correctness of the output shares depends on the correctness of the DDLog procedure. However, in cyclic groups $\mathbb{G}$, the DDLog procedure has an inherent $1/\mathsf{poly}(\lambda)$ error [BGI16, DKK20], which the NIDLS framework overcomes by using specific groups $\mathbb{G}$ (e.g., $\mathbb{Z}_{n^2}^*$) and requiring different assumptions (e.g., DCR). This is why Figure 4.5 only achieves $\epsilon = 1 - 1/\mathsf{poly}(\lambda)$ correctness, in general. The crucial observation we make here is that the DDLog procedure has *no error* when given multiplicative shares of $g^0 \in \mathbb{G}$ and,

moreover, because we additionally only require uniformly random payloads in the case where the parties hold multiplicative shares of g^u, for some $u \neq 0$, we can construct a trivial algorithm "DDLog" procedure using just a PRF, as we show in the following lemma.

Lemma 4.5.2 (Random Distributed Discrete Logarithm)**.** *Let $\mathbb{G}$ be an arbitrary cyclic group with generator g and $1 \leq M \ll |\mathbb{G}|$ be an integer. There exists an efficient algorithm* DDLog *satisfying Definition 4.3.4 such that:*

(1) For all elements $h_A, h_B \in \mathbb{G}$ where $h_A \cdot h_B = g^0$,

$$\Pr\Big[\ \langle z\rangle_A - \langle z\rangle_B = 0 \quad : \quad \langle z\rangle_\sigma := \mathsf{DDLog}((\mathbb{G}, g, M), h_\sigma),\ \forall \sigma \in \{A, B\}\ \Big] = 1;$$

(2) For all $h_A, h_B \in \mathbb{G}$ where $h_A \cdot h_B \neq g^0$, it holds that for all $\sigma \in \{A, B\}$:

$$\Big\{\ (\langle z\rangle_\sigma, h_{\overline{\sigma}}) \quad \Big|\quad \langle z\rangle_\sigma := \mathsf{DDLog}((\mathbb{G}, g, M), h_\sigma)\ \Big\} \approx_c \Big\{\ (\langle z\rangle_\sigma, h_{\overline{\sigma}}) \quad \Big|\quad \langle z\rangle_\sigma \xleftarrow{R} \mathbb{Z}_M\ \Big\}.$$

Proof. We construct DDLog satisfying the two required properties using any PRF family $F\colon \{0,1\}^\lambda \times \mathbb{G} \to \mathbb{Z}_M$. Define DDLog as $((\mathbb{G}, g, M), h_\sigma) \mapsto F_K((h_\sigma)^{-1^\tau})$,[7] where K is a public uniformly random PRF key. Then, (1) the output of DDLog consists of pseudorandom shares of zero whenever $h_A \cdot h_B = g^0$ and (2) the output of DDLog for all non-zero values is uniformly random in $\mathbb{Z}_M$ (thus satisfying the second property).

To see (1), note that if $h_A \cdot h_B = g^0$, then it holds that $h_A = h_B^{-1}$, which in turn implies that $F_K(h_A) - F_K(h_B) = 0$, with probability 1.

To see (2), note that if $h_A \cdot h_B = g^u$, for some non-zero u with high (pseudo)entropy independent of the PRF key K, then DDLog outputs two pseudorandom and independent elements of $\mathbb{Z}_M$, which guarantees computational indistinguishability by the security of the PRF. ■

Theorem 4.5.2. *Under the SXDH assumption over a bilinear group, there exists an instantiation of Figure 4.4 with random payloads (cf. Definition 4.5.3) in the message space $\mathbb{Z}_M$, for any integer M, with a $O(N^{2/3})$ public key size, $O(N)$ key generation time, $O(N^{4/3})$ key derivation time, and $O(N^{5/3})$ full domain evaluation time, where $O(\cdot)$ hides a factor of $\mathsf{poly}(\lambda, \log|\mathcal{R}|)$.*

Proof. The construction consists of Figure 4.4 instantiated with a multiplicative-output NIM (Definition 4.4.2) and the degree-2 HSS construction from SXDH (Figure 4.5) using the modified DDLog from Lemma 4.5.2.

The public key size, key generation time, and evaluation time follows from the proof of Theorem 4.5.1. Then, by Lemma 4.5.2 we immediately get correctness and a random payload on the non-zero coordinate. However, to additionally ensure pseudorandomness of the payload given the PRF key K, the parties must set their payload to a uniformly random scalar. That is, party σ sets the non-zero coordinate to be Δ_σ (for some uniformly random $\Delta_\sigma \xleftarrow{R} \mathbb{Z}_p$, where p is the prime order of $\mathbb{G}$). Then, the parties obtain multiplicative shares of

[7] Here, DDLog is implicitly parameterized by the party identifier σ and the global PRF key K. We leave this implicit for readability.

a uniformly random value in $\Delta_A \cdot \Delta_B \in \mathbb{Z}_p$ which guarantees the resulting PRF output is pseudorandom.

■

Remark 24 (Guaranteeing a non-zero output)**.** *We remark that because the payload is uniformly random in $\mathbb{Z}_M$, it may be zero with noticeable probability if M is small. To guarantee a negligible probability of the payload being zero, we can choose the $M \geq 2^\lambda$ so as to ensure the output is non-zero with all but negligible probability. In particular, the* DDLog *procedure of Lemma 4.5.2 does not require M to be polynomial in the security parameter.*

4.6 Generalization to Non-interactive Computation

In this section, we show that we can generalize the ideas behind our NIDPF construction to construct succinct, two-party "multi-key" HSS for a restricted class of computations. In particular, with multi-key HSS, two parties can provide inputs to a computation without having to agree on a common public key ahead of time (similarly to spooky encryption, which is based on multi-key FHE [DHRW16]). We consider a setting where Alice has a large input x and Bob has a short input y, where we will require succinctness in the large input (similarly to Definition 4.4.3 and succinct HSS [ARS24]). Then, by exchanging input shares, Alice and Bob can compute secret shares of $P(x, f(y))$, where f is any NC^1 function and P is a constant-degree polynomial.

Abram et al. [ARS24] achieved succinct HSS for a similar, slightly larger computational class which they call "Special RMS" programs. Effectively, this class supports computations of the form

$$P(x_A, x_B, f(y_A, y_B)),$$

where again $f \in \mathsf{NC}^1$ and P is a constant-degree polynomial, and Alice and Bob can each contribute inputs x_σ, y_σ for each computation role. In their construction, they have $x = (x_A, x_B)$ as a "special" input share and $y = (y_A, y_B)$ as a standard HSS input share. However, since they let the HSS input y be defined by *both* parties, this prevents their construction from being "multi-key" or, in other words, non-interactive. That is, in their construction, the parties need to have a common HSS public key to generate the inputs y_A and y_B used in the evaluation of f through HSS.[8]

We show that we can extend succinct NIM for "special RMS" programs provided that only one party specifies the input to the function f—which is exactly the generalization of our NIDPF construction in Section 4.5. That is, instead of letting both parties provide inputs to the function f, we consider computations of the form $P(x, f(y_\sigma))$, where only party σ is allowed to input y. However, we still get the succinctness property since the total communication incurred is only $o(|x|) + O(|y_\sigma|)$,[9] which is sublinear in the size of the large input. Formally, we capture the following extension to NIM for computing general functions:

[8] In particular, this approach requires a trusted setup to distribute the evaluation keys to both parties before parties can share their inputs y_A and y_B, respectively.

[9] Ignoring polynomial factors in the security parameter.

Definition 4.6.1 (Extended NIM)**.** *Let ℓ_0, ℓ_1 be integers. An* extended NIM *scheme for a function class $\mathcal{F}$, consists of five efficient algorithms,*

$$\mathsf{ExtNIM} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma \in \{A,B\}}),$$

with the same syntax and security property as defined in Definition 4.4.1 except that Decode_σ takes an additional input $f \in \mathcal{F}$. The algorithms must satisfy the following extended correctness property:

Extended Correctness. *For all security parameters $\lambda \in \mathbb{N}$, every function $f \in \mathcal{F}$, and every pair of inputs $x, y \in \{0,1\}^{\ell_0} \times \{0,1\}^{\ell_1}$, an extended NIM scheme is said to be correct if there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\Pr\left[\begin{array}{rl} & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, x) \\ \langle z \rangle_A - \langle z \rangle_B = f(x,y) \quad : & (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, y) \\ & \langle z \rangle_A := \mathsf{Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A, f) \\ & \langle z \rangle_B := \mathsf{Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B, f) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

We can additionally consider succinctness, analogously to Definition 4.4.3; we specify the succinctness directly in the following theorem:

Theorem 4.6.1. *Let $\ell_0, \ell_1, \ell_2 \in \mathbb{N}$, let $\mathcal{R}$ be a ring, and let $\mathcal{P}$ be the set of all constant-degree polynomials defined over $\mathcal{R}$. Let $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Eval})$ be a secret-key HSS scheme (cf. Definition 4.3.7) for the function class $\mathcal{F}\colon \{0,1\}^{\ell_1} \to \mathcal{R}^{\ell_2}$ and satisfying multiplication by memory shares (cf. Definition 4.3.9). Then, there exists an extended NIM (cf. Definition 4.6.1) for the class of functions described by $P(x, f(y))$, where $P \in \mathcal{P}$, $f \in \mathcal{F}$, $x \in \mathcal{R}^{\ell_0}$ and $y \in \{0,1\}^{\ell_1}$. Moreover, the size of the public encoding output by Encode_A is bounded by:*

$$\mathsf{poly}(\lambda, \log|\mathcal{R}|) \cdot (\ell_0)^\epsilon,$$

for some $0 \leq \epsilon < 1$.

Proof (sketch). The main idea is that Alice, given an HSS input share of y from Bob, as well as the evaluation key ek_A, can locally compute memory shares of $f(y)$ under Bob's secret key sk. Simultaneously, Alice and Bob can use NIM to compute memory shares of x under Bob's secret key sk. More concretely, Alice inputs x and Bob inputs sk, to instantiate non-interactive VOLE [ARS24, BCM+24]. This then allows Alice and Bob to compute $x \cdot f(y)$ via the "multiplication by memory shares" supported by the HSS scheme (recall Definition 4.3.9). This immediately generalizes to computing $P(x, f(y))$, for any constant-degree polynomial P. To see this, note that Alice can input the vector $\mathbf{x}$ corresponding to the coefficients of all the monomials of P and Bob can input y for the function $\hat{f}$ that outputs a vector $\mathbf{y} := (1, f(y)^1, f(y)^2, \ldots, f(y)^{d-1})$, where $d \in O(1)$ is the degree of P. Then, $P(x, f(y)) = \langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i \cdot f(y)^{i-1}$. Moreover, when using succinct NIM (Definition 4.4.3), Alice's encoding is sublinear in ℓ_0 and Bob's encoding is independent of d. ■

Chapter 5

Multi-Key Homomorphic Secret Sharing

Summary

Homomorphic secret sharing (HSS) is a distributed analogue of fully homomorphic encryption (FHE) where following an input-sharing phase, parties can locally compute a function over their private inputs to obtain secret shares of the function output.

Over the last decade, HSS schemes have been constructed from an array of different assumptions. However, all existing HSS schemes, except ones based on assumptions known to imply multi-key FHE, require a public-key infrastructure (PKI) or a correlated setup between parties. This limitation carries over to many applications of HSS.

In this chapter, we construct *multi-key homomorphic secret sharing* (MKHSS), where given only a common reference string (CRS), two parties can secret share their inputs to each other and then perform local computations as in HSS, eliminating the need for PKI or correlated setups. Specifically, we present the first MKHSS scheme that supports all NC^1 computations from the Decisional Composite Residuosity (DCR) assumption.

Our construction implies the following applications in the CRS model:

- **Succinct two-round secure computation.** We construct succinct, two-round, two-party secure computation for NC^1 circuits. Previously, such a result was only known from spooky encryption and required using multi-key FHE.
- **Attribute-based NIKE.** We construct non-interactive key exchange (NIKE) protocols where two parties agree on a key if and only if their secret attributes satisfy a public NC^1 predicate. This significantly generalizes the notion of password-based NIKE.
- **Public-key PCFs.** We construct public-key pseudorandom correlation functions (PCFs) for any NC^1 correlation. This yields the first public-key PCFs for Beaver triples (and more) from non-lattice assumptions.
- **Silent MPC.** We construct an p-party secure computation protocol in the silent preprocessing model where the preprocessing phase has communication $O(p)$, ignoring polynomial factors. All prior protocols that do not rely on spooky encryption require $\Omega(p^2)$ communication.

5.1 Introduction

In a homomorphic secret sharing (HSS) [BGI16] scheme supporting a class of functions $\mathcal{F}$, two or more parties, each with a share of a secret x, can compute a function $f \in \mathcal{F}$ over x to get a share of the result $f(x)$. For security, any strict subset of the shares must hide the secret x. Importantly, the shares must be homomorphic, in that they must support local (non-interactive) function evaluations such that for any $f \in \mathcal{F}$, the output shares correspond to an additive sharing of $f(x)$.

The key property of HSS is *succinctness*, namely, the size of the input and output shares must be independent of the size of the circuit computing the function over the secret-shared inputs. HSS can be viewed as a distributed analogue of fully homomorphic encryption [Gen09]: it allows two parties to securely compute a function over their private inputs succinctly with respect to the circuit size.

Starting from the seminal work of Boyle, Gilboa, and Ishai [BGI16], a key goal of HSS is to build secure computation schemes from assumptions other than learning with errors (LWE). By now, many HSS schemes are known [BGI16, DHRW16, BGI17, BCG+17, BKS19, BCG+19b, CM21, OSY21, RS21, ADOS22, DIJL23] from a variety of standard assumptions that vary in the number of parties (most schemes support two parties), the class of supported functions (most schemes support NC^1 computations), and correctness error. In particular, while some HSS schemes have a non-negligible correctness error, they still have applications to secure computation [BGI17, DIJL23].

Over the last decade, HSS schemes have enabled a variety of applications in cryptography. These applications include secure computation with sublinear communication [BGI16, Cou19, CM21, DIJL23], private information retrieval [GI14, BGI16], pseudorandom correlation generators [BCGI18, BCG+19b], and constrained pseudorandom functions [CMPR23].

Homomorphic secret sharing with multiple inputs. The basic notion of HSS only enables computations over the private input of a single party. To support multiple private inputs, the notion of public-key HSS [BGI17, ADOS22] was proposed. In a public-key HSS scheme, following a CRS setup, there is a public-key setup phase where the parties sample and publish their respective public keys. Using these public keys, the parties locally derive a common public key and use it for secret sharing their inputs with one another.[1] This enables the parties to perform secure computations over their private inputs, encrypted under the common public key.

Intuitively, public-key HSS can be viewed as an analogue of *threshold* fully homomorphic encryption [AJL+12]—a multi-party version of FHE in the public-key infrastructure (PKI) model. The key drawback of this notion is the necessity of the PKI setup, which itself relies on a CRS, prior to the input sharing phase. This limitation, in turn, affects the applications of HSS. For example, the PKI requirement carries over in the application of HSS to sublinear secure computation, imposing a minimum of three rounds, which is suboptimal [HLP11].

Furthermore, while HSS (with negligible correctness error) for NC^1 computations implies pseudorandom correlation functions (PCFs) for NC^1 correlations [BGMM20, CMPR23] (assuming the existence of pseudorandom functions computable in NC^1), the same is not known

[1] Alternatively, a correlated setup can take place where the parties obtain shares of a secret key belonging to a common public key.

	Assumption	CRS	Transparent Setup	Computational Class
[DHRW16]	$i\mathcal{O}$+DDH	✗	None	P/poly
[DHRW16, XW23]	LWE	✓	✓	P/poly*
Theorem 5.4.1	DCR	✓	✗	NC^1

Table 5.1: Construction of MKHSS realized in this work and comparison to prior work.
*Requires making circular security assumptions to obtain a scheme for all circuits.

for *public-key* PCFs. A public-key PCF [OSY21, BCM+24] is a much stronger primitive that allows two parties to generate correlated randomness "on the fly," without needing to engage in a correlated setup ahead of time.

Multi-key homomorphic secret sharing. In this chapter, we study *multi-key homomorphic secret sharing* (MKHSS), which does not require any PKI or other correlated randomness setup. Instead, given only a CRS, the parties can directly secret share their inputs to each other. In this sense, MKHSS can be viewed as an analogue of multi-key FHE [LTV12, MW16]—a multi-party version of FHE that enables computing over the private data of multiple entities, without needing PKI.

MKHSS for multiple parties is readily implied by spooky encryption [DHRW16], which is currently only known from assumptions known to imply FHE. Similarly to the foundational work of Boyle et al. [BGI16], which initiated the study of HSS as an alternative to FHE, we investigate the feasibility of MKHSS from assumptions not known to imply FHE. As we discuss next, MKHSS enables new applications that were not previously known from public-key HSS, similarly to how multi-key FHE enabled many new applications relative to the older (and weaker) notion of threshold FHE.

We show that *multi-key* HSS is possible from group-based assumptions in the two-party setting and prove the following theorem:

Theorem 5.1.1 (Informal)**.** *Under the DCR assumption, there exists a two-party, multi-key homomorphic secret sharing scheme for computing all functions in the class* NC^1.

In particular, our MKHSS scheme has a negligible correctness error and supports a large message space, similarly to non-multi-key HSS constructions from DCR [OSY21, RS21, ADOS22]. We compare our result to the state-of-the-art in Table 5.1.

5.1.1 Applications of multi-key HSS

We show that many of the applications that multi-key FHE implies are also possible from MKHSS in the two-party setting, thanks to our construction. We briefly summarize the applications of our scheme. Technical details surrounding these applications can be found in Sections 5.4.4, 5.5 and 5.6 of this chapter.

Application I: Sublinear, two-round secure computation. As was shown by Boyle et al. [BGI17], standard HSS already gives two-party secure computation in three rounds and

with sublinear communication in the circuit size. At a high level, the three-round protocol proceeds as follows:

Round 1: Agree on a public key and derive shares of the secret key.

Round 2: Exchange inputs encrypted under the public key.

Round 3: Locally evaluate the function and output the resulting shares.

The question of constructing *two-round* sublinear secure computation protocols from group-based assumptions has remained open. In particular, only *multi-key* FHE was known to be sufficient to instantiate sublinear two-round secure computation in the CRS model.

Our construction gives the first realization of two-round secure computation in the CRS model from an assumption that was not known to imply multi-key FHE. Concretely, multi-key HSS immediately implies the following sublinear, two-round secure computation protocol:

Round 1: Exchange inputs encrypted under independent public keys.

Round 2: Locally evaluate the function and output the resulting shares.

This protocol achieves *reusability* of the first round messages [AJJM20, BL20, BGMM20, AJJM21, BJKL21] in the following sense: the parties can compute different functions over their inputs without having to recompute their first round messages. Furthermore, a party can reuse its first-round message in different computations with different parties. In particular, this enables one party to even go offline after sending the first message, and only later complete the computation asynchronously.

Theorem 5.1.2 (Informal)**.** *Under the DCR assumption, there exists a sublinear, two-party, two-round secure computation protocol for* NC^1 *computations.*

Application II: Attribute-based NIKE. We show that our construction of MKHSS also implies *attribute-based* non-interactive key exchange (ANIKE) supporting NC^1 predicates.

An ANIKE scheme involves two parties, each with their own secret attribute. The requirement is that the parties can derive a shared key if their secret attributes jointly satisfy a public predicate. However, if their attributes do not satisfy the predicate, then they derive independent keys (and do not learn that the predicate was unsatisfied).

ANIKE captures password-based NIKE as well as its extensions such as *fuzzy* password-based NIKE, where parties can derive a shared key only if they share approximately-matching passwords (e.g., those derived from biometrics). In general, ANIKE is well-suited for applications that involve authenticating complicated credentials before providing sensitive information. We briefly summarize the construction (details are provided in Section 5.5):

Public key: Alice samples an MKHSS public and secret key, and generates input shares of her secret attribute x_A and a random shift. Her public key consists of her MKHSS public key and the input share of her attribute and shift. Bob computes his public key in a symmetric manner.

Key derivation Alice, given (1) her secret key, (2) her own input share, and (3) Bob's public key with an input share of his secret attribute x_B, uses MKHSS to evaluate the program which computes the predicate and multiplies the result by the random shift

of each party. Bob evaluates his shares exactly as Alice does. The key for each party consists of the subtractive share output by the MKHSS evaluation.

If the predicate evaluates to 0, which we define as the predicate being satisfied, Alice and Bob end up with pseudorandom subtractive shares of 0, i.e., the same key. On the other hand, if the predicate evaluates to 1, Alice and Bob end up with subtractive shares of a random value (i.e., independent keys) because of the random shifts.

To the best of our knowledge, all existing constructions of key exchange which support NC^1 predicates require interaction and/or assume some idealized model [KKL+16, Mel22]. We realize the first *non-interactive* construction in the standard model. Thanks to the non-interactivity property, our ANIKE scheme—and the associated security proof—is conceptually very simple. In contrast, interactive constructions of ANIKE have many moving parts, have complicated proofs as a result, and many constructions have been broken due to subtle flaws [JRX24].

Theorem 5.1.3 (Informal)**.** *Under the DCR assumption, there exists an attribute-based non-interactive key exchange protocol supporting predicates in* NC^1*.*

Public-key PCFs for NC^1 correlations. Modern secure computation protocols are realized in the preprocessing model [Bea95, DPSZ12]. In this model, during an "offline" preprocessing phase, the computing parties generate many *pseudorandom correlations* that are independent of any function they will later compute. Then, during an online phase, the parties use the stored correlations to compute a function over their inputs in a secure protocol. Thanks to the correlated randomness, the online phase has far greater efficiency by not requiring any cryptographic operations. Pseudorandom correlation functions [BCG+20a] (PCFs) push this model of secure computation to the limit by allowing parties to obtain a short key that they can use to locally, and "on-demand," to generate correlated pseudorandomness for use in the online phase.

Starting with the work of Orlandi, Scholl, and Yakoubov [OSY21], which introduced the concept of a public-key PCF, parties can *non-interactively* derive a PCF key using only the other party's public key. The advantage of public-key PCFs is that any pair of parties can generate correlated randomness on the fly, using only each other's public keys. However, existing constructions of public-key PCFs [OSY21, BCM+24] (including the construction from Chapter 3 for ListOT), are restricted to the OT/VOLE correlation (or weaker variants thereof). In particular, barring spooky encryption, constructing a public-key PCF for even Beaver triple correlations has, so far, remained elusive. In Section 5.6, we use our MKHSS construction to build the first public-key PCF for any NC^1 correlation from the DCR assumption.

Theorem 5.1.4 (Informal)**.** *Under the DCR assumption, there exists a public-key pseudorandom correlation function for* NC^1 *correlations.*

Silent, secure multi-party computation. For multi-party computation protocols instantiated in the preprocessing model, the typical choice of correlated randomness consists of Beaver triples. A p-party Beaver triple consists of additive shares of (a, b, ab), where a, b are random elements in some finite ring. In practice, the cost of securely generating the correlated randomness in the preprocessing phase often dominates the overall cost of the protocol. To

mitigate this cost, a relatively recent line of work [BCGI18, BCG+19a, BCG+19b, BCG+20a] has introduced the *silent preprocessing model*, in which the correlated randomness is replaced by correlated *pseudo*randomness computed by a PCF.

Thanks to recent advances in homomorphic secret sharing and PCFs, there exist silent preprocessing protocols for generating suitable correlated pseudorandomness from standard assumption such as DCR, DDH in class groups, and variants of LPN [OSY21, RS21, ADOS22, BCG+20b, BCCD23]. However, in the context of p-party secure computation, all known methods (that do not rely on spooky encryption) incur an $\Omega(p^2) \cdot \mathsf{poly}(\lambda)$ communication overhead in the preprocessing phase. This overhead stems from the fact that all known constructions of HSS and PCFs are, barring some exceptions (cf. Remark 25), restricted to the setting of *two participants*, and generating p-party correlations via these primitives requires all pairs of parties to interact.

Remark 25. *Nearly all HSS and PCF constructions are restricted to the two-party setting. However, some exceptions include the p-party HSS scheme from sparse LPN of Dao, Ishai, Jain, and Lin [DIJL23], which cannot be used to generate correlated randomness due to its imperfect correctness, the 4-party DCR-based HSS scheme of Boyle, Couteau, and Meyer [BCM23], and the 8-party scheme of Couteau and Kumar [CK24]. We also note that this restriction also applies to pseudorandom correlation generators (PCGs) [BCG+19b].*

Using MKHSS, we show how to construct a multi-party, public-key PCF for Beaver triples over any finite ring. At a high level, using our multi-party public-key PCF for NC^1, p parties can simultaneously broadcast their public keys on a public channel. Then, any pair of parties can, without any interaction, derive two-party Beaver triples. Using the two-party shares, all parties can locally aggregate their two-party Beaver triples to obtain (an arbitrary number of) p-party Beaver triples. Using these precomputed (and pseudorandom) Beaver triples, the parties can then run any efficient p-party non-cryptographic protocol to securely compute a target function (e.g., via the GMW protocol [GMW87]).

As a direct corollary of our MKHSS construction, there exists a p-party protocol securely computing an arithmetic circuit C with s multiplication gates and m outputs over a ring $\mathcal{R}$ with the following communication complexity:

- **Preprocessing**: The parties communicate $p \cdot \mathsf{poly}(\lambda)$ bits in a single broadcast round.
- **Online**: The parties communicate $p \cdot (2s + m)$ elements of $\mathcal{R}$.

This yields a quadratic communication improvement over the state-of-the-art [BBC+24] approach for secure computation in the preprocessing phase. However, we note that our construction is still primarily of theoretical interest because our MKHSS construction is not concretely efficient for general computations.

Theorem 5.1.5 (Informal)**.** *Let C be an arithmetic circuit with n inputs, s multiplication gates, and m outputs, instantiated over a ring $\mathcal{R}$. Under the DCR assumption, for any number of parties p, there exists a p-party secure computation protocol for computing C in the preprocessing model, with the following communication complexity:*

- *In the preprocessing phase: $O(p)$ bits in a single broadcast round.*
- *In the online phase: $p \cdot (2s + m)$ ring elements.*

The protocol is secure against a passive adversary corrupting any strict subset of parties.

Chapter organization. We provide an in-depth technical overview in Section 5.2 capturing the details of our construction. In Section 5.3, we provide the necessary preliminaries related to our construction. In Section 5.4, we provide our formal definition of MKHSS. In Section 5.4.3, we describe our construction of MKHSS from DCR. In Sections 5.5 and 5.6, we provide detailed constructions of these applications.

5.2 Technical Overview

In this section, we provide a technical overview of our MKHSS construction. We organize the overview into the following subsections:

- *Background.* In Section 5.2.1, we define the notation that we use. Then, in Section 5.2.2, we describe the basic template underpinning all existing group-based HSS constructions and provide other relevant background.
- *Challenges.* In Section 5.2.3, we explain the challenges involved in adapting the basic HSS template into a *multi-key* HSS construction.
- *Construction.* In Section 5.2.4, we explain the new ideas that allow us to build MKHSS from DCR. Then, in Section 5.2.5, we overview the full construction.

5.2.1 Notation

We briefly provide some relevant notation for this overview, see Section 5.3 for more details. We let λ denote the security parameter. We let $a \leftarrow \mathsf{Alg}$ denote the output of a (possibly randomized) algorithm Alg and $a \xleftarrow{R} S$ denote a uniformly random sampling from the set S. Assignment of a value b to a variable a is denoted $a := b$. We denote three types of "secret shares" which form the backbone of existing HSS scheme abstractions [BGI16] (see Section 5.2.2 for background) as follows:

- *Input shares* of a message x are denoted by $[\![x]\!]$.
- *Memory shares* of a message x are denoted by $\langle\!\langle x \rangle\!\rangle$.
- *Subtractive shares* of a message x are denoted by $\langle x \rangle$.[2]

This notation is used to describe the *set* of shares of the message x. When referring to a single party's share, we write $[\![x]\!]_\sigma$, $\langle\!\langle x \rangle\!\rangle_\sigma$, and $\langle x \rangle_\sigma$, where $\sigma \in \{A, B\}$ is the party identifier (e.g., Alice and Bob's subtractive shares of x are denoted $\langle x \rangle_A$ and $\langle x \rangle_B$, respectively). We define these share types and describe how they are used to realize an HSS scheme next.

[2]Subtractive shares (z_A, z_B) of a message x are defined over the integers such that $x = z_A - z_B$.

5.2.2 Background on HSS from group-based assumptions

Here, we describe a *simplified* template capturing the basics of existing group-based HSS constructions [BGI16, BCG+17, OSY21, RS21, ADOS22]. Relative to the full constructions, this minimal template omits some important details in the interest of clarity. Our primary goal here is to capture the essential components needed to understand our multi-key HSS approach. Because all known group-based HSS schemes are in the two-party setting, we will call these parties Alice and Bob throughout this overview.

Instantiating the group. Group-based HSS constructions require an Abelian group $\mathbb{G}$ in which a suitable subgroup indistinguishability assumption holds (e.g., DDH in cyclic groups, DCR in the Paillier group, or similar assumptions in class groups). We will use g to denote the generator of $\mathbb{G}$. We will also use a "special" generator h for a suitable subgroup of $\mathbb{G}$ in which the discrete logarithm is computationally easy. Concretely, for the Paillier [Pai99] group $\mathbb{G} = \mathbb{Z}^*_{N^2}$, g is a random generator of a hidden-order subgroup, and $h = (N+1)$ which generates a subgroup of order N.

Correlated setup. All existing HSS schemes require some form of correlated setup process to generate a common public key and distribute "evaluation keys" to the two parties. More concretely, in group-based constructions, the setup produces a public key $f := g^{-s}$ and secret shares the secret key s between Alice and Bob. Following the trusted setup, each party can generate *input shares* of a private input x by encrypting it under the public key f with an "ElGamal-style" encryption over $\mathbb{G}$. Looking ahead, the main challenge in realizing *multi-key* HSS is replacing the entire trusted setup process with a common reference string. In particular, while it was shown that it is possible to reduce the correlated setup down to one round of interaction [BGI17, ADOS22] in the PKI model (i.e., when all parties know each other's public keys), removing this round of interaction has remained an open problem.

Input shares. An input share of a message x (we will define the message space later) under the public key f consists of two "ElGamal-like" ciphertexts in the group, where the first ciphertext encrypts $x \cdot s$ (recall, s is the secret key) and the second ciphertext encrypts x. All operations over the messages are performed "in the exponent" of the subgroup of $\mathbb{G}$ generated by h. An HSS input share of a message x given to party $\sigma \in \{A, B\}$ is denoted as $[\![x]\!]_\sigma$ and defined as:

$$[\![x]\!]_\sigma := \Big(\underbrace{(g^r,\ h^{x \cdot s} f^r)}_{\text{Ciphertext 1}},\ \underbrace{(g^{r'},\ h^x f^{r'})}_{\text{Ciphertext 2}} \Big), \tag{5.1}$$

where $r, r' \xleftarrow{R} \mathbb{Z}_N$. Note that all parties get the same ciphertexts; while it is possible to add private state to the input shares, group-based HSS schemes satisfy the property that at least one component of the input share is identical across parties.

In addition to input shares, existing HSS schemes define an "intermediate" sharing used during a computation called a *memory share*, which we describe next.

Memory shares. A memory share of a message x held by party $\sigma \in \{A, B\}$ is denoted as $\langle\!\langle x \rangle\!\rangle_\sigma$ and is defined as a tuple of secret shares, consisting of subtractive shares of the message x and $x \cdot s$. In particular, a memory share is the secret-shared analog of an input share.

That is,

$$\langle\langle x \rangle\rangle_\sigma := \Big(\langle x \cdot s \rangle_\sigma, \langle x \rangle_\sigma \Big). \tag{5.2}$$

Using the definition of an input share and a memory share, we can now describe how existing group-based HSS schemes evaluate functions.

5.2.2.1 Evaluating functions

The template for evaluating functions on the input shares, introduced by Boyle et al. [BGI16], is to emulate the program via a set of multiplication and addition instructions. In particular, the idea is to show that it is possible to compute a restricted-multiplication straight-line (RMS) program [Cle90]—a special model of computation that is known to be sufficiently powerful to evaluate all branching programs (and the class of functions in NC^1).[3]

In a nutshell, RMS programs are defined to take a set of input values and require maintaining the following rules. Each input to the program can either be (1) converted into a memory value or (2) multiplied by a memory value to produce a memory value of the product [Cle90, BGI16]. Moreover, (3) memory values can be added together to produce a memory value of the sum. In particular, what an RMS program does *not* allow is multiplying two memory values together, since that would imply the ability to compute all functions.

In existing group-based HSS schemes, non-interactively evaluating RMS programs over input shares boils down to computing (2)—a multiplication between an input share and a memory share. That is, given an input share of x and a memory share of y, it should be possible for each party to *locally* derive a memory share of xy. Once this single requirement is satisfied, meeting the other requirements becomes relatively straightforward.

The fact that the input and memory shares defined in Equations (5.1) and (5.2) enable computing a memory share of the product is not difficult to show, but requires using one crucial ingredient: the distributed discrete logarithm (DDLog) procedure [BGI16, OSY21, RS21, ADOS22], which we briefly explain here.

Tool: The Distributed Discrete Logarithm. The DDLog procedure enables local conversion of multiplicative shares to subtractive shares as follows. Given *multiplicative* shares of any value x in the group $\mathbb{G}$, where one party holds $h^{\langle x \rangle_A} g^{\langle 0 \rangle_A}$ and the other party holds $h^{\langle x \rangle_B} g^{\langle 0 \rangle_B}$ such that[4]

$$h^{\langle x \rangle_A} g^{\langle 0 \rangle_A} \cdot h^{-\langle x \rangle_B} g^{-\langle 0 \rangle_B} = h^x,$$

the DDLog procedure allows party-σ to obtain a subtractive share $\langle x \rangle_\sigma$. The details of the distributed discrete log procedure do not matter for the purposes of this overview, and we will treat it as a black-box algorithm satisfying the above "share conversion" property. However, it does play a vital role in computing multiplications between input shares and memory shares in all existing group-based HSS schemes, as we explain next.

Computing a multiplication in HSS. Computing a multiplication between an input share and a memory share is done in two steps. The idea is to exploit (1) the additive homomorphism

[3] See also Section 5.3 for background on RMS programs.

[4] Note that h is the group element of order N in $\mathbb{Z}^*_{N^2}$.

of memory shares, (2) the additive homomorphism of the ElGamal-style encryption, and (3) the linear decryption process. First, the parties compute the multiplication "in the exponent" of the group. Then, using DDLog, the resulting multiplicative shares are converted back to memory shares. In more detail:

Step I: Computing a multiplication "in the exponent." Given an input share $[\![x]\!]_\sigma$ and a memory share $\langle\!\langle y \rangle\!\rangle_\sigma$, party-$\sigma$ (for $\sigma \in \{A, B\}$) computes:

$$\left((g^r)^{\langle y \cdot s \rangle_\sigma} \cdot (h^{x \cdot s} f^r)^{\langle y \rangle_\sigma},\ (g^{r'})^{\langle y \cdot s \rangle_\sigma} \cdot (h^x f^{r'})^{\langle y \rangle_\sigma}\right) = \left(h^{\langle xy \cdot s \rangle_\sigma} g^{\langle 0 \rangle_\sigma},\ h^{\langle xy \rangle_\sigma} g^{\langle 0 \rangle_\sigma}\right).$$

To see the equality, recall that $f = g^{-s}$.

Notice that each party now holds a *multiplicative share* of $(xy \cdot s,\ xy)$, which corresponds to the party having the correct memory share "in the exponent" of the group. The next step is converting this back to a subtractive share via the DDLog procedure described above.

Step II: Conversion to memory shares. By applying the DDLog procedure to each component of the above multiplicative share, the parties locally recover subtractive shares of $(xy \cdot s,\ xy)$, i.e., a memory share of xy. To see this, it suffices to observe that:

$$\left(\mathsf{DDLog}(h^{\langle xy \cdot s \rangle_\sigma} g^{\langle 0 \rangle_\sigma}),\ \mathsf{DDLog}(h^{\langle xy \rangle_\sigma} g^{\langle 0 \rangle_\sigma})\right) = \left(\langle xy \cdot s \rangle_\sigma,\ \langle xy \rangle_\sigma\right) = \langle\!\langle xy \rangle\!\rangle_\sigma.$$

At this point, the parties hold memory shares of the desired product, and can continue multiplying other input shares with the newly derived memory share. This enables the computation of RMS programs, as we briefly explain next.

Computing RMS programs. Observe that if the parties are additionally given memory shares of 1 (e.g., as part of the correlated setup), then they can locally convert any input share into a memory share by computing a multiplication by 1. All in all, this is now sufficient to evaluate the three operations required for RMS programs: (1) An input can be converted to a memory value, (2) an input can be multiplied by a memory value, and (3) any two memory values can be added together to provide a memory value of the sum.

With the above template for how to construct HSS for RMS programs, we are now ready to list some of the challenges and pitfalls associated with constructing *multi-key* HSS.

5.2.3 Challenges associated with multi-keyness

Before we dive in, we emphasize that the problem of eliminating the correlated setup comes down to two things. First, the parties need to obtain a memory sharing of 1 under some joint secret key derived on the fly. Second, they need a way to obtain input shares encrypted under this joint key. If these two problems were magically resolved, then the computation of RMS programs follows.

In particular, the difficulty lies primarily in getting a "re-encryption" of an HSS input share under some joint key, *without* any interaction or correlated setup.[5] We call this the

[5]We note that a common reference string is still allowed in this model; what we need to avoid is any setup process that distributes correlated secrets to parties, which bars solutions based in the PKI model where parties are allowed to exchange public keys in a setup round.

problem of *synchronizing* input shares. To understand this better, consider two input shares generated as in Equation 5.1 but defined under two *independent* public keys $f_A := g^{-s_A}$ and $f_B := g^{-s_B}$. In particular, consider the input shares generated by each party independently:

$$\text{Party-}A\text{'s input share: } \Big((g^{r_A},\ h^{x\cdot s_A} f_A^{r_A}),\ (g^{r'_A},\ h^{x} f_A^{r'_A})\Big)$$
$$\text{Party-}B\text{'s input share: } \Big((g^{r_B},\ h^{y\cdot s_B} f_B^{r_B}),\ (g^{r'_B},\ h^{y} f_B^{r'_B})\Big).$$

The problem is that, given the input shares (and public key) of the other party, it is unclear how the parties can evaluate RMS programs over their independent input shares. While one party, say Alice, can give Bob a share of her secret key s_A, which would then allow the two parties to compute an RMS program *using Alice's input shares*, the multi-key problem arises when trying to compute a program using both the inputs of Alice *and* Bob. This is where prior work resorts to an extra round of communication: parties first agree on a joint public-key in the first round and then share their inputs using this joint key in the second round [BGI17, OSY21, ADOS22]. In the multi-key setting, the question becomes:

How can Alice and Bob non-interactively *obtain a "synchronized" input share under a joint public key?*

Interestingly, this question can be *partially* resolved by leveraging the structure of ElGamal-style encryption. In particular, given Alice's public key f_A, Bob can compute a joint public key $f := f_A \cdot f_B = g^{-(s_A+s_B)}$.

Observe that Alice and Bob can actually interpret their own keys as being "trivial" memory shares of 1 under the joint secret key $s = s_A + s_B$, since $(s_A, 1)$ and $(s_B, 0)$ form subtractive shares of $(s, 1)$, satisfying the invariant of Equation 5.2. Then, for an input share sent by Alice, Bob can compute a "partially synchronized" input share under the joint public key as:

$$\Big((g^{r_A},\ h^{x\cdot s_A} f_A^{r_A}\cdot (g^{r_A})^{-s_B}),\ (g^{r'_A},\ h^{x} f_A^{\cdot r_A}\cdot (g^{r'_A})^{-s_B})\Big) = \Big((g^{r_A},\ h^{x\cdot s_A} f^{r_A}),\ (g^{r'_A}, h^{x} f^{r'_A})\Big),$$

which defines a valid ciphertext tuple under the joint secret key.

Moreover, given that Alice generated the input share, she can trivially re-encrypt it on her end under the joint key $f := g^{-(s_A+s_B)}$ using the same randomness r_A and r'_A (reusing the randomness r_A, r'_A ensures that Alice and Bob obtain the exact same synchronized input share at the end).

This idea *almost* gives a valid input share under the joint public key. The only issue is that the "synchronized" share still has Alice's secret key s_A encrypted in the first component. Unfortunately, while seemingly minor, this is a major obstacle in achieving multi-key HSS. In particular, the above idea fails to give an encryption of $x \cdot (s_A + s_B)$ and thus the resulting ciphertexts do not constitute a valid input share with respect to the joint public key. This prevents the parties from computing RMS programs (indeed, it is not even possible to convert such a share to a memory share, let alone compute a multiplication).

Intuitively, the reason why Alice and Bob are able to synchronize the encryption of x (and not $x \cdot s_A$) is because they can both compute $g^{r'_A \cdot s_B}$: Alice using her knowledge of r'_A and Bob using his knowledge of s_B. Upon closer inspection, this was made possible because *both* r'_A

and s_B are random, which means giving out $g^{r'_A}$ and g^{s_B} does not compromise security and makes it possible to compute $(g^{s_B})^{r'_A} = (g^{r'_A})^{s_B}$ *à la* Diffie–Hellman key exchange [DH76].

In contrast, we run into trouble when doing the same with the encryption of $x \cdot s_A$. Getting an encryption of $x \cdot (s_A + s_B)$ seems to require Alice and Bob to compute $h^{x \cdot s_B}$. This seems challenging for two reasons. First, unlike in the previous case, it is insecure to send h^x or h^{s_B} since discrete logarithms are easy over the subgroup generated by h. However, even if we were to use g, Alice cannot send g^x because x is not random and therefore g^x leaks information on x.

5.2.4 Solving the synchronization challenges

To get around the synchronization challenges outlined in Section 5.2.3, we have to take inspiration from existing constructions of HSS and carefully string together several ideas and observations, which we explain next.

Outline. In Section 5.2.4.1, we start by describing how we can get a step closer to multi-key HSS by avoiding the need for the parties to have encryptions of the secret key as part of the input shares and instead only giving out "implicit" encryptions of the key. In Section 5.2.4.2, we show that defining the joint secret key multiplicatively paves the way to synchronization if we additionally sample the secret keys in a special way (which we describe in Section 5.2.4.3). The new way of sampling secret keys results in "full synchronization" of the input shares but introduces a correctness problem. In Section 5.2.4.4, we resolve this introduced correctness problem by moving to a generalized Paillier-ElGamal encryption scheme. Finally, in Section 5.2.4.5, we show how parties can non-interactively generate subtractive shares of the joint encryption key. We overview the final construction in Section 5.2.5.

5.2.4.1 Step 1: Removing encryptions of the secret key

Abram et al. [ADOS22] observe that it is possible to define a "flipped" ElGamal-like encryption by reversing the role of g and the public key in a ciphertext. A surprising feature of flipped encryption is that "input-to-memory conversion" automatically yields a subtractive share of $x \cdot s$ when decrypted with a share of the correct decryption key s. In more detail, if Alice generates her input share as:

$$[\![x]\!]_A := \Big((h^x g^{r_A},\ f_A^{r_A}),\ (g^{r'_A},\ h^x f_A^{r'_A}) \Big),$$

then the first (highlighted) component can only be decrypted by computing $(h^x g^{r_A})^{s_A} \cdot (f_A^{r_A}) = h^{x \cdot s_A}$, which corresponds to a decryption of the desired result. (Note that it is still possible to decrypt h^x in the usual way using the second ciphertext present in the input share.)

The hope is that, by not having an "explicit" encryption of the secret key, we can mitigate the challenges we ran into in Section 5.2.3. Specifically, using the "flipped" encryption trick above, an input share of a message x under Alice's public key is of the form:

$$\Big((h^x g^{r_A}, f_A^{r_A}),\ (g^{r'_A}, h^x f_A^{r'_A}) \Big) \in \mathbb{Z}_{N^2}^* \times \mathbb{Z}_{N^2}^*.$$

However, at this point, the modified input share does not appear to bring us any closer

to resolving the synchronization challenge. In particular, it is unclear how Bob can compute $g^{r_A \cdot s_B}$ without being given g^{r_A} (which would break security of the encryption scheme). Despite this, in the next few steps, this modified encryption of the input share will play a crucial role in achieving synchronization.

5.2.4.2 Step 2: Defining the joint secret key multiplicatively

Next, observe that the parties can equivalently define their joint public key as $f := g^{-s_A \cdot s_B}$ using Diffie–Hellman [DH76]. Ignoring, for the moment, the challenge of then deriving subtractive shares of the product $s_A \cdot s_B$ (we will resolve this later, in Section 5.2.4.5) we first show how Alice and Bob can derive partially synchronized input shares, as described in Section 5.2.3, under this newly defined joint public key. While in and of itself, it is still not clear how it helps solve the synchronization issue, it paves the way for our next trick, described in Section 5.2.4.3, which does result in full synchronization.

Recall that the goal of synchronization in MKHSS is to take an HSS input share of x generated under Alice's public key and transform it into an HSS input share under the joint key. In this case, Alice and Bob need to locally obtain an input share of the form:

$$\left(\left(h^x g^{r_A}, f^{r_A}\right),\ \left(g^{r'_A}, h^x f^{r'_A}\right) \right),$$

where $f = g^{-s_A \cdot s_B}$.

As before, Alice can trivially synchronize her own share by simply re-encrypting x under the joint public key f and reusing the same randomness r, r' she used to generate the original share. By doing so, Alice obtains a new HSS input share of the form:

$$\left(\left(h^x \cdot g^{r_A}, f^{r_A}\right),\ \left(g^{r'_A}, h^x f^{r'_A}\right) \right), \tag{5.3}$$

which is distributed exactly as an input share under the joint public key f.

Now, we try again to let Bob synchronize by computing

$$\left(\left(h^x \cdot g^{r_A}, (f_A^{r_A})^{s_B}\right),\ \left(g^{r'_A}, (h^x f_A^{r'_A})^{s_B}\right) \right) = \left(\left(h^x \cdot g^{r_A}, f^{r_A}\right),\ \left(g^{r'_A}, h^{x \cdot s_B} f^{r'_A}\right) \right).$$

However, we run into a similar barrier to the one we faced with our prior synchronization attempt described in Section 5.2.3—the second component is an encryption of $x \cdot s_B$ under the joint public key f whereas we need it to be just an encryption of x.

The crucial insight we make next is that the partial synchronization introduced a *superfluous* multiplication by s_B. In contrast, our previous attempt had the exact opposite problem: it was *missing* a multiplication by s_B. Intuitively, while finding a way to obliviously multiply by Bob's secret key s_B appears impossible (as we already explained in Section 5.2.3), it turns out that canceling out the superfluous multiplication by s_B is quite easy, as we explain next.

5.2.4.3 Step 3: Use special secret keys

At this point, to explain how we cancel out the superfluous multiplication by s_B, we need to work explicitly over the Paillier group $\mathbb{Z}^*_{N^2}$ and use Paillier–ElGamal encryption [CS02,

BCP03, DJ03]. Concretely, g is a random generator of a subgroup of $\mathbb{Z}_{N^2}^*$ of order $\phi(N)/4$ and $h = (N+1)$ is a generator of another subgroup of order N.

The goal now is to ensure that the secret key s_B is automatically "canceled out" in the second component of $\left(g^{r'_A}, h^{x \cdot s_B} f^{r'_A}\right)$. To make this happen, our idea is to change the space from which the secret keys are sampled. Rather than sampling s_σ from the set $\{1, \ldots, N\}$ (as done in Paillier–ElGamal [DJ03]), we instead sample the secret keys from the set

$$\{N+1,\ 2N+1,\ 3N+1,\ \ldots,\ (N-1) \cdot N + 1\},$$

which guarantees that all sampled secret keys satisfy $s_\sigma \equiv 1 \bmod N$. Observe that because the secret keys are sampled over the integers, this requirement can be easily satisfied by first sampling $s'_\sigma \xleftarrow{R} \{1, \ldots, N-1\}$ and then defining $s_\sigma := s'_\sigma \cdot N + 1 \in \mathbb{Z}$. While at first glance this may appear to be an odd choice for sampling the secret keys, it turns out to be just the trick to achieve full synchronization (and does not harm security, as we will show later).

Specifically, using the fact that the secret key is congruent to 1 mod N and the fact that h has order N in $\mathbb{Z}_{N^2}^*$, we get that $h^{s_B} = h^{i \cdot N + 1 \pmod N} = h \in \mathbb{Z}_{N^2}^*$. This property allows Bob to then synchronize the encryption of the message x as above because $(h^x f_A^{r_A})^{s_B} = h^x f^{r_A}$, which is a proper encryption of x under public key f with randomness r, and matches Alice's synchronized encryption of x computed in Equation 5.3.

The high-level intuition for why sampling the key in this way does not impact security is the following. First, observe that the public key g^{-s} in Paillier–ElGamal, computed with a secret key $s \xleftarrow{R} [N]$, is close to a random subgroup element generated by g. Then, because g has order $\phi(N)/4$, and N is co-prime to $\phi(N)$, a public key $g^{-s'}$ computed with $s' := N \cdot s$, is statistically close to g^{-s}, given that$s$ mod $\phi(N)/4$ is statistically close to $s \cdot N$ mod $\phi(N)/4$. As such, the new sampling results in a public key that is *statistically* close to a standard Paillier–ElGamal public key.

While we now resolved the synchronization problem completely, we are not out of the woods yet. By sampling the secret keys in this way, we lose the ability to derive shares of $x \cdot s$ as required for HSS computations! Specifically, it is no longer possible to compute shares of $x \cdot s$, since now $h^{x \cdot s} = h^x$, making it impossible to compute the multiplication.

5.2.4.4 Step 4: Moving to the Damgård–Jurik–ElGamal

We are now in a situation where we face two competing requirements:

1. We need to sample the secret keys we need to sample the secret keys s_A and s_B such that $s_A \pmod N \equiv s_B \pmod N \equiv 1 \pmod N$ in order for Bob to locally synchronize the encryption of x.
2. At the same time, we want to maintain our ability to compute multiplicative shares of $h^{x \cdot s}$ using the "flipped" decryption technique, without having the group's order cancel out the secret key s.

Despite these two requirements appearing mutually exclusive, our next idea allows us to get around this. Instead of encrypting messages exclusively in $\mathbb{Z}_{N^2}^*$, we can encrypt them both in $\mathbb{Z}_{N^2}^*$ *and* in $\mathbb{Z}_{N^{w+1}}^*$, for some $w > 2$, by using the generalized Paillier–ElGamal encryption scheme of Damgård and Jurik [DJ03]. In $\mathbb{Z}_{N^{w+1}}^*$, the group element h has order

N^w, which allows us to encrypt x separately in $\mathbb{Z}^*_{N^2}$ and then duplicate this in $\mathbb{Z}^*_{N^{w+1}}$, such that $h^{x \cdot s} \in \mathbb{Z}^*_{N^2} \equiv h^x$ and $h^{x \cdot s} \in \mathbb{Z}^*_{N^{w+1}} \equiv h^{x \cdot s \pmod{N^w}}$, for sufficiently large w so that $x \cdot s$ does not exceed N^w, allowing us to satisfy both requirements. This now gives Bob the ability to fully synchronize with Alice and allows us to satisfy the required invariants to evaluate branching programs over the synchronized input shares.

The final problem we still need to resolve is giving the parties a way to locally obtain subtractive shares of the joint secret key $s_A \cdot s_B$. We glossed over this problem in Section 5.2.4.2 when we defined the joint public key as being $g^{-s_A \cdot s_B}$.

5.2.4.5 Step 5: Getting shares of the secret key

The final ingredient we need is a non-interactive solution for computing subtractive secret share of $s := s_A \cdot s_B$. Fortunately, we can easily achieve this by using non-interactive multiplication (NIM), as described and constructed in Chapter 4.

Recall that a NIM scheme allows Alice and Bob to generate shares of the multiplication of their respective inputs by exchanging public encodings. To allow the parties to locally derive shares of the joint secret key, we have the parties provide NIM encodings as part of their public keys. Then, by encoding their secret keys as the message, Alice and Bob can locally derive subtractive shares of the joint secret key $s = s_A \cdot s_B$ (defined over the integers) using just the public key of the other party, which is exactly a share of the joint secret key they require.

Constructing NIM from DCR. As was shown in Chapter 4 (specifically, Section 4.4), we can construct a NIM scheme from the DCR assumption. We sketch the construction here for completeness. The scheme is essentially a simplification of non-interactive NIM for matrix products described in Chapter 4 and proceeds in two steps: (1) compute the multiplication "in the exponent" of the group and then (2) compute the DDLog to obtain subtractive shares over the integers.

Let N be a suitable composite modulus and let g and h be random generators of $\mathbb{Z}^*_{N^2}$ that are part of the CRS. The protocol is instantiated over the ring $\mathcal{R} = \mathbb{Z}_\ell$, where for correctness we need $\ell < 2^{-\lambda} \cdot \sqrt{N}$. The high-level idea behind the NIM construction is to have:

- Alice's public encoding consist of a commitment $g^{r_A} h^x$ to her element x and
- Bob's public encoding consist of an encryption $(g^{r_B}, (N+1)^y h^{r_B})$ of his element y,

where r_A and r_B are random elements of $\mathbb{Z}_N$.

Then, given Alice's encoding $\mathsf{pe}_A := g^{r_A} h^x$, Bob derives $Z_B := (g^{r_A} h^x)^{-r_B} = g^{-r_A r_B} h^{-x r_B}$. Similarly, given Bob's encoding $\mathsf{pe}_B := (g^{r_B}, (N+1)^y h^{r_B})$, Alice derives $Z_A := (g^{r_B})^{r_A} \cdot ((N+1)^y h^{r_B})^x$. It is not hard to see that Z_A and Z_B form multiplicative shares of $(N+1)^{xy \bmod N}$ since:

$$\begin{aligned} Z_A \cdot Z_B &= ((g^{r_B})^{r_A} \cdot ((N+1)^y h^{r_B})^x) \ \cdot \ (g^{r_A} h^x)^{-r_B} \\ &= (g^{r_A r_B} \cdot (N+1)^{xy} h^{x r_B}) \ \cdot \ (g^{-r_A r_B} h^{-x r_B}) \\ &= (N+1)^{xy}. \end{aligned}$$

Therefore, by applying the DDLog procedure to Z_A and Z_B, the parties recover subtractive

shares of xy mod N. Moreover, because $x, y < 2^{-\lambda} \cdot \sqrt{N}$, we have that, with all but negligible probability, the shares $\langle xy \rangle_A$ and $\langle xy \rangle_B$ are subtractive shares over the integers, by the correctness of the DDLog algorithm.

5.2.5 Putting things together

We now bring all the above ideas together and overview our full multi-key HSS construction (see Section 5.4.3 for the full scheme). The CRS consist of $(N, g, \mathsf{crs}_{\mathsf{nim}})$, where $\mathsf{crs}_{\mathsf{nim}}$ is the CRS of the NIM scheme.

Key generation. Each party samples a secret key $s_\sigma \xleftarrow{R} \{i \cdot N + 1 \mid 1 \leq i \leq N - 1\}$ and sets the corresponding encryption public key to $f_\sigma := g^{-s_\sigma}$. Additionally, each party generates a public NIM encoding pe_σ of s_σ (keeping the private NIM state, denoted st_σ, to itself). The MKHSS public key of party σ is defined as $\mathsf{pk}_\sigma := (f_\sigma, \mathsf{pe}_\sigma)$, and the secret key $\mathsf{sk}_\sigma := (s_\sigma, \mathsf{st}_\sigma)$.

Input sharing. Alice shares her message x_A (without knowing the identity of Bob) by generating two input shares:

$$[\![x_A]\!]_A := \Big(\big(x_A, r_A, r'_A\big), \big((N+1)^{x_A} \cdot g^{r_A}, f_A^{r_A}\big),\ \big(g^{r'_A}, (N+1)^{x_A} f_A^{r'_A}\big) \Big)$$
$$[\![x_A]\!]_B := \Big(\big((N+1)^{x_A} \cdot g^{r_A}, f_A^{r_A}\big),\ \big(g^{r'_A}, (N+1)^{x_A} f_A^{r'_A}\big) \Big),$$

where her own share of her own message $[\![x_A]\!]_A$, contains private "state" $\big(x_A, r_A, r'_A\big)$ that is not made available in the other, public share $[\![x_A]\!]_B$. We stress that $[\![x_A]\!]_B$ is generated *without* knowledge of Bob's identity (the subscript B refers to any party that may later want to play the role of Bob and synchronize with Alice). Alice publishes $(\mathsf{pk}_A, [\![x_A]\!]_B)$. Bob (or any other party) can generate input shares under their own—and independent—public keys in the same way as Alice does above.

Key synchronization. Once any two parties, playing the role of Alice and Bob, decide to perform an HSS computation on their respective input shares, they can syncrhonize their encryption public keys by computing a Diffie–Hellman key exchange to derive a joint encryption key $f := g^{-s_A \cdot s_B} = g^{-s}$. Then, they can use the NIM encodings from the public keys to locally derive a secret share of the joint decryption secret key s.

Input synchronization. Finally, Alice and Bob synchronize Alice's MKHSS input share under their joint public key as follows. Note that synchronization of Bob's MKHSS input share follows by performing the symmetric synchronization computation.

We will denote a synchronized input share of a message x by $\{\{x\}\}$.

1. Alice synchronizes her own input share of x_A under the joint public encryption key f by using her private state (x_A, r_A, r'_A) and simply re-encrypting the message as:

$$\{\{x\}\} := \Big(\big((N+1)^{x_A} \cdot g^{r_A}, f^{r_A}\big),\ \big(g^{r'_A}, (N+1)^{x_A} f^{r'_A}\big) \Big) \in (\mathbb{Z}^*_{N^{w+1}})^2 \times (\mathbb{Z}^*_{N^2})^2.$$

2. Bob synchronizes with Alice by using the joint public encryption key f and his secret

key s_B and computing:

$$\{\{x\}\} := \Big(\big((N+1)^{x_A} \cdot g^{r_A}, (f_A^{r_A})^{s_B}\big),\ \big(g^{r'_A}, \big((N+1)^{x_A} f_A^{r'_A}\big)^{s_B}\big)\Big)$$
$$= \Big(\big((N+1)^{x_A} \cdot g^{r_A}, f^{r_A}\big),\ \big(g^{r'_A}, (N+1)^{x_A} f^{r'_A}\big)\Big) \in (\mathbb{Z}^*_{N^{w+1}})^2 \times (\mathbb{Z}^*_{N^2})^2.$$

In particular, observe that $\{\{x_A\}\}$ is identical for both parties. We show that $\{\{x_A\}\}$ is a suitable HSS input share under the joint key f, making it possible for Alice and Bob to then compute any NC^1 function over this synchronized input share. The idea then extends to synchronizing with Bob's input share (under the same joint public key f) and to synchronizing many input shares provided by both parties.

5.3 Preliminaries

In this section, we cover the notation that we will use throughout the chapter.

General notation. We let $\mathbb{N}$ denote the set of natural numbers, $\mathbb{Z}$ denote the set of integers, $\mathbb{G}$ denote a finite group, and $\mathcal{R}$ denote a finite ring. A reduction modulo t, for any positive integer t, yields a representative in the range $\mathbb{Z}_t = \{-\lfloor t/2 \rfloor, \ldots, \lfloor (t-1)/2 \rfloor\}$. We denote by $\mathsf{poly}(\cdot)$ the set of all polynomials and by $\mathsf{negl}(\cdot)$ any negligible function. We occasionally abuse notation and let poly denote a fixed polynomial.

Vectors and matrices. We denote a vector $\mathbf{v}$ using bold lowercase letters and a matrix $\mathbf{A}$ using bold uppercase letters. The i-th coordinate of a vector $\mathbf{v}$ is denoted by $\mathbf{v}[i]$. We will occasionally write $(v_i)_{i=1}^n$ to denote the vector $(v_1, \ldots, v_n)$.

Vector group operations. For all $\mathbf{g} \in \mathbb{G}^\ell$ and $\mathbf{x} \in \mathbb{Z}^\ell$, we use $\langle \mathbf{g}, \mathbf{x} \rangle$ to denote $\langle \mathbf{g}, \mathbf{x} \rangle = \prod_{i=1}^\ell g_i^{x_i}$, where $\mathbf{g} = (g_1, \ldots, g_\ell)$ and $\mathbf{x} = (x_1, \ldots, x_\ell)$.

Sampling and assignment. We let $x \xleftarrow{\mathrm{R}} S$ denote a uniformly random sample drawn from a set S. We let $x \leftarrow \mathcal{A}$ denote assignment from a randomized algorithm $\mathcal{A}$ and $x := y$ denote initialization of x to the value of y (which may be the output of a deterministic algorithm).

Efficiency. By an *efficient* algorithm $\mathcal{A}$ we mean that $\mathcal{A}$ is modeled by a (possibly non-uniform) Turing Machine that runs in probabilistic polynomial time.

Probability and indistinguishability. We let $\Pr[E : A]$ denote the probability of an event E in an experiment defined by executing A. For two probability ensembles $\{A_i\}_i$ and $\{B_i\}_i$, we use $\{A_i\}_i \equiv \{B_i\}_i$ to denote that the ensembles are identical, $\{A_i\}_i \approx_s \{B_i\}_i$ to denote that the ensembles are statistically close and $\{A_i\}_i \approx_c \{B_i\}_i$ to denote that the ensembles are computationally indistinguishable.

Subtractive Sharing. Let $\mathcal{R}$ be a ring. We use $\langle x \rangle^{\mathcal{R}} \in \mathcal{R}^2$ where $\langle x \rangle^{\mathcal{R}} = (\langle x \rangle_A^{\mathcal{R}}, \langle x \rangle_B^{\mathcal{R}})$ to denote a subtractive sharing of $x \in \mathcal{R}$ such that $\langle x \rangle_A^{\mathcal{R}} - \langle x \rangle_B^{\mathcal{R}} = x$. For ease of notation, we use $\langle x \rangle = (\langle x \rangle_A, \langle x \rangle_B)$ to denote the subtractive sharing over the integers when $\mathcal{R} = \mathbb{Z}$.

5.3.1 Cryptographic assumptions

We will make use of the Decisional Composite Residuosity (DCR) assumption [Pai99].

Assumption 1 (Decisional Composite Residuosity Assumption)**.** *Let* GenPQ *be a randomized algorithm that, on input the security parameter* λ*, outputs two distinct, sufficiently large, random safe primes* p *and* q*. The DCR assumption states that:*

$$\left\{ (N, g_0) \;\middle|\; \begin{array}{r} (p,q) \leftarrow \mathsf{GenPQ}(1^\lambda) \\ N := pq \\ g_0 \xleftarrow{R} \mathbb{Z}^*_{N^2} \end{array} \right\} \approx_c \left\{ (N, g_1) \;\middle|\; \begin{array}{r} (p,q) \leftarrow \mathsf{GenPQ}(1^\lambda) \\ N := pq \\ g_0 \xleftarrow{R} \mathbb{Z}^*_{N^2} \\ g_1 := g_0^N \end{array} \right\}.$$

5.3.2 Cryptographic building blocks

We will make use of the following two building blocks: Non-interactive key exchange, non-interactive multiplication (introduced in Chapter 4 and recalled here for convenience), and the Damgård–Jurik encryption scheme.

Non-Interactive Key Exchange. Here, we provide a basic definition of non-interactive key exchange, which will suffice for our applications and constructions.

Definition 5.3.1 (Non-Interactive Key Exchange [DH76, CKS08, FHKP13])**.** *Let* $\lambda \in \mathbb{N}$ *be a security parameter. A non-interactive key exchange (NIKE) scheme consists of algorithms* $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$*. The randomized setup algorithm takes as input the security parameter* λ *and outputs a common reference string* crs*.*
- $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$*. The randomized key generation algorithm takes as input the CRS* crs*. It outputs a public key* pk *and secret key* sk*.*
- $\mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_i, \mathsf{sk}_j) \to K$*. The deterministic key derivation algorithm takes as input the CRS* crs*, a public key* pk_i*, and a secret key* sk_j*. It outputs a key* $K \in \{0,1\}^\lambda$*.*

The above algorithms must satisfy the following properties:

Correctness. *For all security parameters* $\lambda \in \mathbb{N}$*, it holds that:*

$$\Pr\left[K_A = K_B \quad : \quad \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}_A, \mathsf{sk}_A) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ (\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ K_A \leftarrow \mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_B, \mathsf{sk}_A) \\ K_B \leftarrow \mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_A, \mathsf{sk}_B) \end{array} \right] = 1.$$

Security. *For all efficient adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\, b = b' \quad : \quad \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}_A, \mathsf{sk}_A) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ (\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ K_0 \leftarrow \mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_A, \mathsf{sk}_B) \\ K_1 \xleftarrow{R} \{0,1\}^\lambda \\ b \xleftarrow{R} \{0,1\} \\ b' \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_A, \mathsf{pk}_B, K_b) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

In particular, this security definition for NIKE is known as "CKS-light" security [FHKP13], which is known to be polynomially equivalent to stronger notions of NIKE.

Damgård–Jurik–ElGamal encryption scheme. We first recall the Damgård–Jurik "ElGamal" encryption scheme in Figure 5.1. The scheme is proven secure under the DCR assumption [DJ03] (see also [CS02, BCP03]). For convenience, we extend the scheme to support the "flipped" encryptions via a FlipEncrypt algorithm.

For completeness, we prove the security of the extended DJEG encryption scheme presented in Figure 5.1.

Lemma 5.3.1. *Let λ be a security parameter. If the DCR assumption holds, then the encryption scheme presented in Figure 5.1 satisfies the standard notion of semantic security (i.e., CPA-security).*

Proof. The proof of semantic security follows a similar proof made in [BCCS24, Section 4.4] and proceeds with a simple hybrid argument. Here, we adapt the proof to the generalized Damgård–Jurik–ElGamal setting.

- *Hybrid $\mathcal{H}_0$.* This hybrid consist of a ciphertext (c_0, c_1) as generated by DJEG.Encrypt in Figure 5.1.

- *Hybrid $\mathcal{H}_1$.* In this hybrid, we change how the randomness r is sampled in DJEG.Encrypt, and sample r uniformly from $\{0, 1, \ldots, N^{w+1}\}$ instead of $\{0, 1, \ldots, N\}$.

  Claim. $\mathcal{H}_1 \approx_s \mathcal{H}_0$.

  Proof. This hybrid is statistically close to the previous one by the fact that g and f have order $\phi(N)/4$, which is coprime to N. We note that we implicitly use the fact that GenPQ outputs safe primes making g, as sampled in Figure 5.1, a generator for the subgroup of order $\phi(N)/4$ with overwhelming probability. □

- *Hybrid $\mathcal{H}_2$.* In this hybrid, we change how the public key f is sampled in DJEG.KeyGen by sampling f as a uniformly random $2N$-th residue. That is, $f := (g')^{2N} \in \mathbb{Z}^*_{N^2}$, where $g' \xleftarrow{\mathrm{R}} \mathbb{Z}^*_{N^2}$.

  Claim. $\mathcal{H}_2 \approx_s \mathcal{H}_1$.

Damgård–Jurik–ElGamal Encryption Scheme [DJ03]

Public Parameters. A generator GenPQ with respect to which the DCR assumption holds.

$\mathsf{DJEG.Setup}(1^\lambda)$:

1 : $(p, q) \leftarrow \mathsf{GenPQ}(1^\lambda)$
2 : $N := pq$
3 : $g_0 \xleftarrow{R} \mathbb{Z}^*_{N^2}$
4 : $g := (g_0)^{2N} \in \mathbb{Z}^*_{N^2}$
5 : **return** $\mathsf{crs} := (N, g)$

$\mathsf{DJEG.KeyGen}(\mathsf{crs})$:

1 : **parse** $\mathsf{crs} = (N, g)$
2 : $s \xleftarrow{R} [N]$
3 : $f := g^s$
4 : $(\mathsf{pk}, \mathsf{sk}) := (f, s)$
5 : **return** $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{DJEG.Decrypt}(\mathsf{sk}, \mathsf{ct})$:

1 : **parse** $\mathsf{ct} = (c_0, c_1)$
2 : $c' := c_1/(c_0)^{\mathsf{sk}}$
3 : $x := \dfrac{c' - 1}{N^w}$
4 : **return** x

$\mathsf{DJEG.Encrypt}(\mathsf{crs}, \mathsf{pk}, x, w)$:

1 : **parse** $\mathsf{crs} = (N, g)$
2 : **parse** $\mathsf{pk} = f$
3 : $r \xleftarrow{R} \{0, 1, \ldots, N\}$
4 : $c_0 := g^r \bmod N^{w+1}$
5 : $c_1 := (N+1)^x f^r \bmod N^{w+1}$
6 : **return** $\mathsf{ct} := (c_0, c_1)$

$\mathsf{DJEG.FlipEncrypt}(\mathsf{crs}, \mathsf{pk}, x, w)$:

1 : **parse** $\mathsf{crs} = (N, g)$
2 : **parse** $\mathsf{pk} = f$
3 : $r \xleftarrow{R} \{0, 1, \ldots, N\}$
4 : $c_0 := (N+1)^x g^r \bmod N^{w+1}$
5 : $c_1 := f^r \bmod N^{w+1}$
6 : **return** $\mathsf{ct} := (c_0, c_1)$

Figure 5.1: The DJEG encryption scheme.

Proof. By the definition of $g = (g_0)^{2N}$, it is a generator for the subgroup of the $2N$-th residues with overwhelming probability (again, using the fact that N is a composite of safe primes). Then, it suffices to note that in $\mathcal{H}_1$ we have $f = g^s$, which is a uniformly random $2N$-th residue when g is a generator for the subgroup of $2N$-th residues and s is sampled uniformly from $\mathbb{Z}_N$. □

- *Hybrid* $\mathcal{H}_3$. In this hybrid, we change how the public key f is sampled in DJEG.KeyGen by sampling f as a uniformly random square from $\mathbb{Z}^*_{N^2}$.

  Claim. $\mathcal{H}_3 \approx_c \mathcal{H}_2$ *assuming DCR.*

  Proof. The claim follows from a direct reduction to the DCR assumption. Notice that f is sampled as a $2N$-th residue in $\mathcal{H}_2$ and a random square of $\mathbb{Z}^*_{N^2}$ in $\mathcal{H}_3$. The reduction thus has at most a factor of two loss in advantage in the DCR game. □

  Remark 26. *We note that, thanks to CRT decomposition,* $\mathbb{Z}_{N^w \cdot \phi(N)}$ *is isomorphic to* $\mathbb{Z}_{N^w} \times Z_{\phi(N)}$ *because* N *is coprime to* $\phi(N)$*. Using this, any element* c *in* $\mathbb{Z}^*_{N^{w+1}}$ *can be written as* $c = (1+N)^a g^b \bmod N^{w+1}$*, for some* $(a, b) \in \mathbb{Z}_{N^w} \times \mathbb{Z}_{\phi(N)}$*, since all elements in* $\mathbb{Z}^*_{N^w \cdot \phi(N)}$ *can be decomposed into this form. Moreover, for a random* c*, with overwhelming probability* $1 - \frac{p+q-1}{N}$*, we have that* $a \neq 0$ *and coprime to* N*.*

- *Hybrid* $\mathcal{H}_4$. In this hybrid, the ciphertext elements c_0 and c_1 are sampled as uniformly random elements of $\mathbb{Z}^*_{N^{w+1}}$.

  Claim. $\mathcal{H}_4 \approx_s \mathcal{H}_3$.

  Proof. We claim that, in hybrid $\mathcal{H}_3$, (c_0, c_1) is already statistically close to the uniform distribution over $\mathbb{Z}^*_{N^{w+1}} \times \mathbb{Z}^*_{N^{w+1}}$. To see this, we first note that g generates a subgroup of order $\phi(N)/4$ and, therefore, the element c_0 *statistically* reveals only the value $r_0 = r \bmod \phi(N)/4$. Moreover, using Remark 26, c_1 can be rewritten as follows:

  $$c_1 = (1+N)^x \cdot f^r = (1+N)^{ar_1 + x \bmod N^w} \cdot g^{b \cdot r_0 \bmod \phi(N)/4} \bmod N^{w+1},$$

  where $r_0 = r \bmod \phi(N)/4$ and $r_1 = r \bmod N^w$. Then, conditioned on r_0, r_1 is statistically close to a uniformly random element by the fact that N and $\phi(N)$ are coprime. By the above, we have that $ar_1 + x \bmod N^w$ is statistically close to a uniformly random element of $\mathbb{Z}_{N^w}$ given r_0 (recall that a is coprime to N, with overwhelming probability). Combined, we have that (c_0, c_1) are statistically close to a uniformly random tuple of elements sampled from $\mathbb{Z}^*_{N^{w+1}}$. □

We have now concluded the proof of semantic security for the DJEG scheme when the ciphertext is generated using DJEG.Encrypt. We note that a very similar hybrid argument applies to proving that ciphertexts output by the "flipped" encryption DJEG.FlipEncrypt are computationally indistinguishable from uniform under the DCR assumption. A little more formally, starting with $\mathcal{H}_3$, the element f is distributed identically to g (both are random squares in $\mathbb{Z}^*_{N^2}$) which enables interchanging them in c_0 and c_1. This concludes the proof. ■

Non-interactive multiplication. Here, we recall the definition of NIM from Chapter 4. We change the syntax of the encoding and decoding algorithms to not depend on the party identifier, which simplifies our MKHSS construction. This change is without loss of generality since each party's public and private encoding can consist of public and private encodings generated by playing the role of both parties, increasing the size of the public encoding by a factor of two.

Definition 5.3.2 (Non-Interactive Multiplication)**.** *Let λ be a security parameter, $\mathcal{R}$ be a finite ring. A non-interactive multiplication (NIM) scheme consists of three algorithms* $\mathsf{NIM} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter and outputs a common reference string* crs.
- $\mathsf{Encode}(\mathsf{crs}, x) \to (\mathsf{pe}_\sigma, \mathsf{st}_\sigma)$. *The randomized encoding algorithm takes as input the CRS* crs *and a ring element $x \in \mathcal{R}$. It outputs a public encoding* pe_σ *and secret state* st_σ.
- $\mathsf{Decode}(\mathsf{crs}, \mathsf{pe}_{1-\sigma}, \mathsf{st}_\sigma) \to \langle z \rangle_\sigma$. *The deterministic decoding algorithm takes as input the CRS* crs*, another party's public encoding* $\mathsf{pe}_{1-\sigma}$*, and secret state* st_σ*. It outputs a subtractive secret share of z.*

The above functionality must satisfy correctness and security, which are defined as follows:

Correctness. *For all security parameters $\lambda \in \mathbb{N}$ and every pair of elements $x, y \in \mathcal{R}$, a NIM scheme is said to be correct if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{rcr} & & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & & (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}(\mathsf{crs}, x) \\ \langle z \rangle_A - \langle z \rangle_B = xy & : & (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}(\mathsf{crs}, y) \\ & & \langle z \rangle_A := \mathsf{Decode}(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A) \\ & & \langle z \rangle_B := \mathsf{Decode}(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. *A NIM scheme is said to be* secure *if for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, and all $\sigma \in \{A, B\}$, we have that*

$$\Pr\left[\begin{array}{rcr} & & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & & (x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b' = b & : & b \xleftarrow{R} \{0,1\} \\ & & (\mathsf{pe}_\sigma, \mathsf{st}_\sigma) \leftarrow \mathsf{Encode}(\mathsf{crs}, x_b) \\ & & b' \leftarrow \mathcal{A}\left(\mathsf{pe}_\sigma, \mathsf{st}\right) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $x_0, x_1 \in \mathcal{R}$.

5.3.3 Distributed evaluation of RMS programs

In this section, we present a unifying template for reasoning about distributed evaluation of HSS input shares, which not only captures HSS evaluation from prior works but will

also be useful in proving the correctness of our constructions. Note that our focus here is only on correctness of HSS evaluation, assuming both parties already hold shares of all program inputs. How these inputs are securely shared between the parties will be discussed in subsequent sections.

Restricted Multiplication Straight-line (RMS) programs. An RMS program is an arithmetic circuit over integers with the restriction that every multiplication is between an input value and an intermediate value of the computation, called a memory value. Most existing HSS schemes support evaluating RMS programs. Boyle et al. [BGI16] show that the class of polynomial-size RMS programs includes the class of polynomial-size branching programs, which is in turn known to contain the class of NC^1 circuits.

Definition 5.3.3 (Restricted Multiplication Straight-line Program [Cle90, BGI16])**.** *A restricted multiplication straight-line (RMS) program P consists of a magnitude bound $B \in \mathbb{N}$ and an arbitrary sequence of the following four instructions.*

- $\mathsf{Convert}(\mathsf{I}_x) \to \mathsf{M}_x$: *Load the value of the input wire I_x to the memory wire M_x.*
- $\mathsf{Add}(\mathsf{M}_x, \mathsf{M}_y) \to \mathsf{M}_z$: *Add the values of the memory wires M_x and M_y and assign the result to the memory wire M_z.*
- $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y) \to \mathsf{M}_z$: *Multiply the value of the input wire I_x by the value of the memory wire M_y and assign the result to the memory wire M_z.*
- $\mathsf{Output}(\mathsf{M}_z) \to z$: *Output the value of the memory wire M_z.*

If at any step of the execution, the size of a memory value exceeds the bound B, the output of the program on the corresponding input is defined to be $\perp$. The size of an RMS program, denoted by $|P|$, is defined as the number of instructions.

Primitives required for distributed evaluation. The distributed, non-interactive evaluation of RMS programs in group-based HSS schemes rely on two primitives. The first is HSS shares of the inputs, which satisfy a property we abstract as "exponent-linear decoding." This property intuitively captures the decryption process in ElGamal-style public-key encryption schemes instantiated over various groups (e.g., DDH-hard cyclic groups, the Paillier group, class groups, etc.). The second is the distributed discrete logarithm algorithm introduced in [BGI16], which serves as the foundation of all existing group-based HSS constructions. In Lemma 5.3.3, we show that these components suffice for distributed evaluation of any RMS program. This framework captures HSS constructions of Boyle et al. [BGI16] from DDH (the BHHO-based scheme), as well as the HSS constructions by Abram et al. [ADOS22] based on either the DCR assumption or DDH-like assumptions in Paillier and class groups. Looking ahead, although inputs in our multi-key HSS construction are encoded differently from prior works, they still satisfy the exponent-linear decoding property, which in turn allows distributed evaluation of RMS programs.

Definition 5.3.4 (Exponent-Linear Decoding)**.** *Let $\mathbb{G}$ be an abelian group, let $\mathbb{H} \subseteq \mathbb{G}$ be a finite cyclic subgroup of order t with generator h and let $\ell \in \mathbb{N}$. We let $\{\{x\}\} := (\mathbf{c}_1, \ldots, \mathbf{c}_\ell) \in \mathbb{G}^{\ell \times \ell}$ be an encoding of an integer x with base-h exponent-linear decoding under the decoding key $\mathbf{k} = (k_1, \ldots, k_\ell) \in \mathbb{Z}^\ell$ if for all $i \in [\ell]$, we have $\langle \mathbf{c}_i, \mathbf{k} \rangle = h^{x \cdot k_i}$.*

Remark 27 (Exponent-linear decoding in HSS schemes)**.** *In ElGamal-based HSS, $\mathbb{H}$ is simply the group $\mathbb{G}$ itself. However, in the Paillier–ElGamal-based HSS (and variants thereof), $\mathbb{H}$ is a subgroup of $\mathbb{G}$ in which the discrete logarithm (base h) is efficiently computable. In all cases, the decryption procedure in these schemes consists of a linear function ("in the exponent") with the secret key and hence why existing scheme satisfy Definition 5.3.4.*

Definition 5.3.5 (Distributed Discrete Logarithm)**.** *Let $\mathbb{G}$ be an abelian group, let $\mathbb{H} \subseteq \mathbb{G}$ be a finite cyclic subgroup of order t with generator h, let ε be a real number and B_{dl} be a positive integer, where $0 \leq \varepsilon < 1$ and $B_{\mathsf{dl}} < t$. An efficient algorithm DDLog is an ε-correct, B_{dl}-bounded, base-h algorithm for distributed discrete logarithm, if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all integers x where $|x| \leq B$ and all $f \in \mathbb{G}$ we have*

$$\Pr_{r \xleftarrow{R} \{0,1\}^\lambda} [\mathsf{DDLog}(f \cdot h^x; r) - \mathsf{DDLog}(f; r) \not\equiv x \bmod t] \leq \varepsilon + \mathsf{negl}(\lambda).$$

Lemma 5.3.2 (DDlog in DCR; Adapted from [OSY21, Lemma 3.3])**.** *In the Paillier group $\mathbb{Z}^*_{N^2}$, for all $\lambda \in \mathbb{N}$, and all integers $B_{\mathsf{dl}} < N \cdot 2^{-\lambda}$, there exists an $2^{-\lambda}$-correct, B_{dl}-bounded, base-$(N+1)$ algorithm for distributed discrete logarithm.*

Template for distributed evaluation. We conclude this section by describing an algorithm in Figure 5.2 for distributed evaluation of RMS programs, using PRFs, a DDLog algorithm and encodings of inputs that are exponent-linear decodeable. The proof of correctness closely follows that of group-based HSS constructions in prior works; however, we revisit the details here for completeness.

Lemma 5.3.3. *Let $\mathbb{G}$ be an Abelian group, $\mathbb{H} \subseteq \mathbb{G}$ be a finite cyclic subgroup of order t with generator h, DDLog be an ε-correct, B_{dl}-bounded, base-h algorithm for distributed discrete logarithm, and F_1 and F_2 be secure PRFs. Then, for all polynomials $\mathsf{poly}(\cdot)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all $\mathbf{k} = (k_1, \ldots, k_{\ell-1}, 1) \in \mathbb{Z}^\ell$, all $\mathbf{k}_A \in \mathbb{Z}^\ell$, all RMS programs P with bound B, all $x_1, \ldots, x_m \in \mathbb{Z}$ and $\{\{x_1\}\}, \ldots, \{\{x_m\}\} \in \mathbb{G}^{\ell \times \ell}$, the algorithm DEval described in Figure 5.2 satisfies*

$$\Pr\left[\begin{array}{c} \langle z \rangle_A - \langle z \rangle_B \\ \neq \\ P(x_1, \ldots, x_m) \end{array} : \begin{array}{r} k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}} \xleftarrow{R} \{0,1\}^\lambda \\ \mathbf{k}_B := \mathbf{k}_A - \mathbf{k} \\ \mathsf{ek}_\sigma := (k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}}, \mathbf{k}_\sigma),\ \forall \sigma \in \{A, B\} \\ \langle z \rangle_\sigma := \mathsf{DEval}(\sigma, \mathsf{ek}_\sigma, (\{\{x_1\}\}, \ldots, \{\{x_m\}\}), P),\ \forall \sigma \in \{A, B\} \end{array}\right] \leq \varepsilon \cdot \ell \cdot |P| + \mathsf{negl}(\lambda),$$

where each $|k_i| \leq B_{\mathsf{sk}}$ for some $B_{\mathsf{sk}} \in \mathbb{N}$, each $\{\{x_i\}\}$ is an encoding of x_i with base-h exponent-linear decoding under $\mathbf{k}$, $P(x_1, \ldots, x_m) \neq \perp$, $\ell \leq \mathsf{poly}(\lambda)$, $|P| \leq \mathsf{poly}(\lambda)$, $B \cdot B_{\mathsf{sk}} \leq B_{\mathsf{dl}}$ and $B \cdot B_{\mathsf{sk}} \cdot 2^\lambda < t$.

Proof. Observe that for every memory value M_x in the RMS program, party σ computes a share $\langle\!\langle x \rangle\!\rangle_\sigma$. We must show that the output produced by each party is a subtractive sharing of $P(x_1, \ldots, x_m)$. At a high level, we will show this by proving that DEval maintains the invariant that $\langle\!\langle x \rangle\!\rangle = (\langle\!\langle x \rangle\!\rangle_A, \langle\!\langle x \rangle\!\rangle_B)$ forms a subtractive sharing of $x \cdot \mathbf{k}$, for every memory

Distributed Evaluation of RMS Program

Public Parameters. Abelian group $\mathbb{G}$ and finite cyclic subgroup $\mathbb{H} \subseteq \mathbb{G}$ of order t with generator h. Base-h distributed discrete logarithm algorithm DDLog. PRF F_1 with output space $\{0,1\}^\lambda$ and a PRF F_2 with output space $\mathbb{Z}_t$.

$\mathsf{DEval}(\sigma, \mathsf{ek}_\sigma, (\{\{x_1\}\}, \ldots, \{\{x_m\}\}), P)$:

Parse $\mathsf{ek}_\sigma = (k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}}, \langle\langle 1 \rangle\rangle_\sigma)$.

For each $\mathsf{id} \in [|P|]$, evaluate the id-th instruction as follows:

- $\mathsf{Convert}\colon \mathsf{M}_x \leftarrow \mathsf{I}_x$:
  1: Execute the $\mathsf{Mult}(\{\{x\}\}, \langle\langle 1 \rangle\rangle_\sigma)$ instruction to compute $\langle\langle x \rangle\rangle_\sigma$.
- $\mathsf{Mult}\colon \mathsf{M}_{xy} \leftarrow \mathsf{I}_x \cdot \mathsf{M}_y$:
  1: Parse $\{\{x\}\} = (\mathbf{c}_1, \ldots, \mathbf{c}_\ell)$.
  2: For $i \in [\ell]$:
    2.1: $f_\sigma^{(i)} \coloneqq \langle \mathbf{c}_i, \langle\langle y \rangle\rangle_\sigma \rangle$.
    2.2: $\langle z_i \rangle_\sigma \coloneqq \mathsf{DDLog}(f_\sigma^{(i)}; F_1(k_1^{\mathsf{prf}}, \mathsf{id}\|i)) + F_2(k_2^{\mathsf{prf}}, \mathsf{id}\|i) \bmod t$.
  3: $\langle\langle xy \rangle\rangle_\sigma \coloneqq (\langle z_1 \rangle_\sigma, \ldots, \langle z_i \rangle_\sigma)$.
- $\mathsf{Add}\colon \mathsf{M}_{x+y} \leftarrow \mathsf{M}_x + \mathsf{M}_y$:
  1: $\langle\langle x + y \rangle\rangle_\sigma \coloneqq \langle\langle x \rangle\rangle_\sigma + \langle\langle y \rangle\rangle_\sigma$.
- $\mathsf{Output}\colon z \leftarrow \mathsf{M}_z$:
  1: Parse $\langle\langle z \rangle\rangle_\sigma = (\langle z_1 \rangle_\sigma, \ldots, \langle z_i \rangle_\sigma)$.
  2: Return $\langle z_\ell \rangle_\sigma$.

Figure 5.2: Distributed evaluation of RMS program.

value M_x. Then, since the last component of $\mathbf{k}$ is 1, the parties obtain a subtractive sharing of the program output upon evaluating DEval.

Note that for $\mathsf{Add}(\mathsf{M}_x, \mathsf{M}_y)$ instructions, the above invariant holds trivially due to the additive homomorphism of subtractive sharing.

For $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y)$ instructions, the exponent-linear decoding property allows party-B to compute a $f_B^{(i)} \in \mathbb{G}$ and party-A to compute $f_A^{(i)} = f_B^{(i)} \cdot h^{xy \cdot k_i}$, for each component k_i of the decoding key. The parties can then compute a subtractive sharing of $xy \cdot \mathbf{k}$ using DDLog.

Let the output of the correctness experiment be defined as 1 if $\langle z \rangle_A - \langle z \rangle_B = P(x_1, \ldots, x_m)$ and defined as 0 otherwise. We will prove that this output is 1 with probability $\varepsilon \cdot \ell \cdot |P| + \mathsf{negl}(\lambda)$.

We first use a simple hybrid argument to replace the pseudorandom outputs of the PRFs with uniformly random values.

- *Hybrid* $\mathcal{H}_0$. This hybrid is the output of the experiment, as defined above.

- *Hybrid* $\mathcal{H}_1$. This hybrid is identical to the previous hybrid, except that $\langle z_i \rangle_\sigma$ in DEval is computed as $\langle z_i \rangle_\sigma := \mathsf{DDLog}(f_\sigma^{(i)}; r_\sigma^{(i)}) + \hat{r}_\sigma^{(i)} \bmod t$, where $r_\sigma^{(i)} \in \{0,1\}^\lambda$ and $\hat{r}_\sigma^{(i)} \in \mathbb{Z}_t$ are the outputs of truly random functions evaluated at $\mathsf{id} \| i$.

  Claim. $\mathcal{H}_0 \overset{c}{\approx} \mathcal{H}_1$.

  Proof. The claim follows by the pseudorandomness of the PRFs F_1 and F_2. □

Now that we have uniformly random shares, we will prove that the experiment's output is 1, except with a probability of at most $\varepsilon \cdot \ell \cdot |P| + \mathsf{negl}(\lambda)$. To do so, we first show that if the input for a multiplication satisfies the invariant, the invariant will also hold for the product with probability at least $1 - \varepsilon - \mathsf{negl}(\lambda)$. Then, we use this to derive a lower bound on the probability that the output of the experiment is 1 in hybrid $\mathcal{H}_1$ above.

Claim. *For each multiplication instruction* $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y)$ *evaluated in* DEval*, we have*

$$\Pr[\langle\!\langle xy \rangle\!\rangle_A - \langle\!\langle xy \rangle\!\rangle_B \neq xy \cdot \mathbf{k} \mid \langle\!\langle y \rangle\!\rangle_A - \langle\!\langle y \rangle\!\rangle_B = y \cdot \mathbf{k}] \leq \varepsilon \cdot \ell + \mathsf{negl}(\lambda).$$

Proof. Consider any arbitrary $i \in [\ell]$. Since $\{\{x\}\} = (\mathbf{c}_1, \ldots, \mathbf{c}_\ell)$ is exponent-linear decodable under $\mathbf{k} = (k_1, \ldots, k_\ell)$, we have

$$h^{xy \cdot k_i} = \langle \mathbf{c}_i, y \cdot \mathbf{k} \rangle = \langle \mathbf{c}_i, \langle\!\langle y \rangle\!\rangle_A - \langle\!\langle y \rangle\!\rangle_B \rangle = f_A^{(i)} \cdot \left(f_B^{(i)}\right)^{-1}$$
$$\implies f_A^{(i)} = h^{xy \cdot k_i} \cdot f_B^{(i)},$$

where the second equality follows from the fact that $\langle\!\langle y \rangle\!\rangle_A - \langle\!\langle y \rangle\!\rangle_B = y \cdot \mathbf{k}$.

Let $\langle z_i' \rangle_\sigma = \mathsf{DDLog}(f_\sigma^{(i)}; r_\sigma^{(i)})$, where $r_\sigma^{(i)}$ is the output of a truly random function. Since P is B-bounded and $P(x_1, \ldots, x_m) \neq \perp$, we have $|xy| \leq B$. Along with the fact that $|k_i| \leq B_{\mathsf{sk}}$ and $B \cdot B_{\mathsf{sk}} \leq B_{\mathsf{dl}}$, it follows from the correctness of DDLog that $\langle z_i' \rangle_A - \langle z_i' \rangle_B \equiv xy \cdot k_i \bmod t$ with a probability of at least $1 - \varepsilon - \mathsf{negl}(\lambda)$. Moreover, since $\langle z_i \rangle_\sigma = \langle z_i' \rangle_\sigma + \hat{r}_\sigma^{(i)} \pmod t$, where $\hat{r}_\sigma^{(i)} \in \mathbb{Z}_t$, we have $\langle z_i \rangle_A - \langle z_i \rangle_B \equiv xy \cdot k_i \bmod t$, except with a probability of at most $\varepsilon + \mathsf{negl}(\lambda)$.

Conditioned on the event that there was no error in DDLog, $\langle z_i \rangle_A$ and $\langle z_i \rangle_B$ are uniformly random subtractive shares over $\mathbb{Z}_t$ since $\hat{r}_\sigma^{(i)}$ is uniformly random in $\mathbb{Z}_t$. This implies that $\langle z_i \rangle_A - \langle z_i \rangle_B$ does not wrap around t with overwhelming probability.

In more detail, since $|xy \cdot k_i| < B \cdot B_{\mathsf{sk}}$, $\langle z_i \rangle_A - \langle z_i \rangle_B$ wraps around t only when $-t/2 \leq \langle z_i \rangle_B \leq -t/2 + B \cdot B_{\mathsf{sk}}$ or $t/2 - B \cdot B_{\mathsf{sk}} \leq \langle z_i \rangle_B \leq t/2$. The size of this interval is $2B \cdot B_{\mathsf{sk}}$ and since $\langle z_i \rangle_A$ is a uniformly random subtractive share over $\mathbb{Z}_t$, the probability that $\langle z_i \rangle_A - \langle z_i \rangle_B$ wraps around is at most $2B \cdot B_{\mathsf{sk}}/t < 2 \cdot 2^{-\lambda}$, which is negligible. Thus, it follows that $(\langle z_i \rangle_A, \langle z_i \rangle_B)$ constitute a subtractive sharing of $xy \cdot k_i$ over the integers except with a probability of at most $\varepsilon + \mathsf{negl}(\lambda)$, where the probability is over the correctness of DDLog as well as the possibility of wrap-around when converting additive shares over $\mathbb{Z}_t$ to subtractive shares over integers.

Finally, observe that $\langle\!\langle xy \rangle\!\rangle_A - \langle\!\langle xy \rangle\!\rangle_B = xy \cdot \mathbf{k}$ if and only if $\langle z_i \rangle_A - \langle z_i \rangle_B = xy \cdot k_i$, for every $i \in [\ell]$. Therefore, each $(\langle z_i \rangle_A, \langle z_i \rangle_B)$ constitutes a subtractive sharing of $xy \cdot k_i$, except with

a probability of at most $\varepsilon + \mathsf{negl}(\lambda)$, since they are computed with independently sampled randomness. By a union bound, and the fact that $\ell \leq \mathsf{poly}(\lambda)$, we have

$$\Pr[\langle\langle xy \rangle\rangle_A - \langle\langle xy \rangle\rangle_B \neq xy \cdot \mathbf{k} \mid \langle\langle y \rangle\rangle_A - \langle\langle y \rangle\rangle_B = y \cdot \mathbf{k}] \leq \varepsilon \cdot \ell + \mathsf{negl}(\lambda).$$

□

We will argue that if the invariant is true for memory values corresponding to the output of the first $\mathsf{id}-1$ instructions of the RMS program P, then the invariant is true for the memory value that corresponds to the output of the id-th instruction, except with a probability of at most $\varepsilon \cdot \ell + \mathsf{negl}(\lambda)$. In more detail, observe that if the id-th instruction is an addition instruction $\mathsf{Add}(\mathsf{M}_x, \mathsf{M}_y)$, then it follows from the additive homomorphism of subtractive sharing that the invariant holds for M_{x+y} with probability 1 since $\langle\langle x+y \rangle\rangle_A - \langle\langle x+y \rangle\rangle_B = \langle\langle x \rangle\rangle_A + \langle\langle y \rangle\rangle_A - \langle\langle x \rangle\rangle_B - \langle\langle y \rangle\rangle_B = x + y$. Similarly, if the id-th instruction is a multiplication instruction $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y)$, then it follows from our previous claim that the invariant holds for M_{xy} except with a probability of at most $\varepsilon \cdot \ell + \mathsf{negl}(\lambda)$. Finally, if the id-th instruction is a $\mathsf{Convert}(\mathsf{I}_x)$ instruction, then DEval runs the same steps as for evaluating $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_1)$, where $\langle\langle 1 \rangle\rangle_\sigma = \mathbf{k}_\sigma$. Observe that $(\mathbf{k}_A, \mathbf{k}_B)$ is, by definition, a subtractive sharing of $1 \cdot \mathbf{k}$, which implies from our previous claim that $(\langle\langle x \rangle\rangle_A, \langle\langle x \rangle\rangle_B)$ constitutes a subtractive sharing of $x \cdot \mathbf{k}$, except with a probability of at most $\varepsilon \cdot \ell + \mathsf{negl}(\lambda)$. Thus, the invariant holds true for the output M_x of the $\mathsf{Convert}(\mathsf{I}_x)$ instruction.

Since the last component of the decoding key $k_\ell = 1$, we have $\langle z \rangle_A - \langle z \rangle_B = P(x_1, \ldots, x_m)$ in this hybrid when the invariant is true for the memory value M_z corresponding to the output instruction $\mathsf{Output}(\mathsf{M}_z)$. The probability that the invariant does not hold for a memory value is at most $\varepsilon \cdot \ell + \mathsf{negl}(\lambda)$, since the randomness is freshly sampled for each instruction. It thus follows from a straightforward union bound and the fact that $|P| \leq \mathsf{poly}(\lambda)$ that $\langle z \rangle_A - \langle z \rangle_B = P(x_1, \ldots, x_m)$ and the output of the experiment is 1 in hybrid $\mathcal{H}_1$, except with a probability of at most $\varepsilon \cdot \ell \cdot |P| + \mathsf{negl}(\lambda)$. Moreover, since $\mathcal{H}_0 \overset{c}{\approx} \mathcal{H}_1$, it follows that the output of the experiment is 1 in hybrid $\mathcal{H}_0$, except with a probability of at most $\varepsilon \cdot \ell \cdot |P| + \mathsf{negl}(\lambda)$. This concludes the proof. ■

5.4 Multi-Key Homomorphic Secret Sharing

We formalize the notion of MKHSS in Section 5.4.1. Then, we construct MKHSS from DCR in Section 5.4.3.

5.4.1 Definition

We define multi-key HSS (MKHSS) in Definition 5.4.1. An MKHSS scheme allows a party, given a common reference string, to locally generate a key pair and share its input using its public key. These shares can then be used with the input shares computed by any other party (generated using its own public key), to compute subtractive shares of a program's output, evaluated on the joint inputs. This ability to compute shares of the input, independent of the other party's key—indeed, even before knowing the identity of the other party—is the key property of MKHSS schemes.

Let $[\![x]\!]^A_A$ denote Alice's share of her input x and let $[\![x]\!]^A_B$ denote the share of x intended for other parties. The security of the scheme requires that $[\![x]\!]^A_B$—which can be viewed as a ciphertext that enables computing on x—preserves privacy of Alice's input. In contrast, the definition does not impose any security requirements on Alice's share $[\![x]\!]^A_A$, since she already knows the input x as well as the secret key corresponding to the public key used to generate the shares.

We first introduce some additional notation and then proceed with the definition.

Notation. We denote by $[\![x]\!]^\sigma = ([\![x]\!]^\sigma_A, [\![x]\!]^\sigma_B)$ an input sharing of a message x generated using party σ's HSS public key. Additionally, we occasionally write $[\![\mathbf{x}]\!]^\sigma = ([\![x_1]\!]^\sigma, \ldots, [\![x_\ell]\!]^\sigma)$ to denote a *tuple* of input shares of $\mathbf{x} \in \mathcal{R}^\ell$, where $\mathbf{x} = (x_1, \ldots, x_\ell)$. For some party identifier $\sigma \in \{A, B\}$, we write $1 - \sigma$ as shorthand for the "other party identifier" $\overline{\sigma} \in \{A, B\} \setminus \{\sigma\}$.

Definition 5.4.1 (Multi-Key Homomorphic Secret Sharing)**.** *A multi-key homomorphic secret sharing (MKHSS) scheme for a program class $\mathcal{P}$, defined over a ring $\mathcal{R}$, and having a message space $\mathcal{M} \subseteq \mathcal{R}$ consists of four efficient algorithms* $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter and outputs a common reference string (CRS)* crs.
- $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$. *The randomized key generation algorithm, independently run by each party, takes as input the CRS* crs *and outputs a public and private key pair* $(\mathsf{pk}, \mathsf{sk})$ *for the party.*
- $\mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x) \to ([\![x]\!]^\sigma_A, [\![x]\!]^\sigma_B)$. *The randomized share algorithm takes as input the CRS* crs*, the party identifier $\sigma \in \{A, B\}$, the party's public key* pk_σ*, and a message $x \in \mathcal{M}$. It outputs a pair of input shares $([\![x]\!]^\sigma_A, [\![x]\!]^\sigma_B)$ encoding the message x.*
- $\mathsf{Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![\mathbf{x}_A]\!]^A_\sigma, [\![\mathbf{x}_B]\!]^B_\sigma, P) \to \langle z \rangle^\mathcal{R}_\sigma$. *The deterministic evaluation algorithm takes as input the CRS* crs*, the party identifier $\sigma \in \{A, B\}$, the party's secret key* sk_σ*, the public key of another party* $\mathsf{pk}_{1-\sigma}$*, two tuples $[\![\mathbf{x}_A]\!]^A_\sigma$ and $[\![\mathbf{x}_B]\!]^B_\sigma$ of the party's input shares (where the tuples are generated by different parties using* Share*), and a program description P. It outputs a subtractive share (over the ring $\mathcal{R}$) of the evaluation result z.*

An MKHSS scheme must satisfy the following correctness and security properties:

Correctness. *An MKHSS scheme is said to be* ε-correct*, for some $\varepsilon \in [0, 1)$, if for all $\lambda \in \mathbb{N}$, all $2m$-input programs $P \in \mathcal{P}$, and all $\mathbf{x}_A, \mathbf{x}_B \in \mathcal{M}^m$, we have*

$$\Pr\left[\begin{array}{c} \langle z \rangle^\mathcal{R}_A - \langle z \rangle^\mathcal{R}_B \\ \neq \\ P(\mathbf{x}_A, \mathbf{x}_B) \end{array} : \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{KeyGen}(\mathsf{crs}),\ \forall \sigma \in \{A, B\} \\ ([\![\mathbf{x}_\sigma]\!]^\sigma_A, [\![\mathbf{x}_\sigma]\!]^\sigma_B) \leftarrow \mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathbf{x}_\sigma),\ \forall \sigma \in \{A, B\} \\ \langle z \rangle^\mathcal{R}_\sigma \leftarrow \mathsf{Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![\mathbf{x}_A]\!]^A_\sigma, [\![\mathbf{x}_B]\!]^B_\sigma, P),\ \forall \sigma \in \{A, B\} \end{array}\right] \leq \varepsilon + \mathsf{negl}(\lambda),$$

where $\mathbf{x}_\sigma = (x^{(1)}_\sigma, \ldots, x^{(m)}_\sigma)$ for each $\sigma \in \{A, B\}$. Note that we slightly abuse notation by

letting $\mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathbf{x}_\sigma)$ *denote running* $\mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x_\sigma^{(i)})$ *separately for each* i*. If* $\varepsilon = 0$*, we simply say that the MKHSS is* correct*.*

Security. *An MKHSS scheme is said to be* secure *if for all efficient adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\sigma \in \{A, B\}$*, we have that*

$$\Pr\left[\, b' = b \quad : \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ (x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_\sigma) \\ b \xleftarrow{R} \{0,1\} \\ (\llbracket x_b \rrbracket_A^\sigma, \llbracket x_b \rrbracket_B^\sigma) \leftarrow \mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x_b) \\ b' \leftarrow \mathcal{A}\left(\llbracket x_b \rrbracket_{1-\sigma}^\sigma, \mathsf{st}\right) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Comparison to public-key HSS. In a public-key HSS scheme (e.g., [BGI16, Definition 2.2]), the Setup algorithm outputs a public key pk and two private evaluations keys ek_A and ek_B. Any party—not necessarily those holding the evaluation keys—can then share their input using pk, which in turn allows the servers holding the evaluation keys to non-interactively compute on all shared inputs. Thus, compared to an MKHSS scheme, a public-key HSS scheme allows computing on the inputs of several parties; but this comes at the cost of requiring a correlated setup or, alternatively, a PKI [BGI17, OSY21, ADOS22], which implies a two-round sharing protocol in the CRS model.

While an MKHSS scheme and a public-key HSS scheme might initially seem incomparable, it is not too hard to see that the former implies the latter. Specifically, given an MKHSS scheme MKHSS, a public-key HSS scheme can be constructed as follows.

- The setup algorithm computes crs using $\mathsf{MKHSS.Setup}$, generates two keys pairs $(\mathsf{pk}_A, \mathsf{sk}_A)$ and $(\mathsf{pk}_B, \mathsf{sk}_B)$ using $\mathsf{MKHSS.KeyGen}$ and outputs $\mathsf{pk} := (\mathsf{crs}, \mathsf{pk}_A, \mathsf{pk}_B)$, $\mathsf{ek}_A := \mathsf{sk}_A$, and $\mathsf{ek}_B := \mathsf{sk}_B$.
- The HSS share of an input x is computed using pk by first computing a subtractive sharing, $\langle x \rangle = (\langle x \rangle_A, \langle x \rangle_B)$, and then computing MKHSS shares of $\langle x \rangle_\sigma$ using pk_σ i.e.,

$$\left(\llbracket \langle x \rangle_A \rrbracket_A^A, \llbracket \langle x \rangle_A \rrbracket_B^A\right) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, A, \mathsf{pk}_A, x)$$
$$\left(\llbracket \langle x \rangle_B \rrbracket_A^B, \llbracket \langle x \rangle_B \rrbracket_B^B\right) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, B, \mathsf{pk}_B, x).$$

  Thus, $(\llbracket \langle x \rangle_A \rrbracket_A^A, \llbracket \langle x \rangle_B \rrbracket_A^B)$ constitutes the HSS share of x for the server holding ek_A and $(\llbracket \langle x \rangle_A \rrbracket_B^A, \llbracket \langle x \rangle_B \rrbracket_B^B)$ is the HSS share for the server holding ek_B. In particular, although party-σ might learn $\langle x \rangle_\sigma$, the security of MKHSS ensures the privacy of $\langle x \rangle_{1-\sigma}$, thereby preserving the privacy of x.
- To evaluate a program P on the shared inputs, the servers use $\mathsf{MKHSS.Eval}$ to evaluate a program P' that first reconstructs the inputs and then evaluates P.

This gives a public-key HSS scheme for all programs P for which the corresponding program P' can be evaluated using the MKHSS scheme. Specifically, for an MKHSS scheme supporting

$\underline{\mathsf{E}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B,0}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
foreach $\sigma \in \{A, B\}$:
  $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$
  $(\llbracket \mathbf{x}_\sigma \rrbracket^\sigma_A, \llbracket \mathbf{x}_\sigma \rrbracket^\sigma_B) \leftarrow \mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathbf{x}_\sigma)$
  $\langle z \rangle^{\mathcal{R}}_\sigma := \mathsf{Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, \llbracket \mathbf{x}_A \rrbracket^A_\sigma, \llbracket \mathbf{x}_B \rrbracket^B_\sigma, P)$
$b \leftarrow \mathcal{A}(\mathsf{pk}_A, \mathsf{pk}_B, \llbracket \mathbf{x}_A \rrbracket^A_B, \llbracket \mathbf{x}_B \rrbracket^B_A, \langle z \rangle^{\mathcal{R}}_A, \langle z \rangle^{\mathcal{R}}_B)$
return b

$\underline{\mathsf{E}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B,1}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
foreach $\sigma \in \{A, B\}$:
  $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$
  $(\llbracket \mathbf{x}_\sigma \rrbracket^\sigma_A, \llbracket \mathbf{x}_\sigma \rrbracket^\sigma_B) \leftarrow \mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathbf{x}_\sigma)$
$\langle z \rangle^{\mathcal{R}}_B \xleftarrow{\text{R}} \mathcal{R}$
$\langle z \rangle^{\mathcal{R}}_A := \langle z \rangle^{\mathcal{R}}_B + P(\mathbf{x}_A, \mathbf{x}_B)$
$b \leftarrow \mathcal{A}(\mathsf{pk}_A, \mathsf{pk}_B, \llbracket \mathbf{x}_A \rrbracket^A_B, \llbracket \mathbf{x}_B \rrbracket^B_A, \langle z \rangle^{\mathcal{R}}_A, \langle z \rangle^{\mathcal{R}}_B)$
return b

Figure 5.3: External security experiment for MKHSS.

polynomial-size RMS programs—which is the focus of this chapter—this implies a public-key HSS scheme for polynomial-size RMS programs.

5.4.2 External security

We introduce an additional security notion for MKHSS, which will be important for our applications. The notion strengthens the correctness property of MKHSS by requiring, informally, that the output shares of any HSS evaluation are indistinguishable from uniformly random subtractive shares of the output over the ring $\mathcal{R}$.

Definition 5.4.2 (External Security of Multi-Key Homomorphic Secret Sharing)**.** *An MKHSS scheme* $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ *for a program class* $\mathcal{P}$*, defined over a ring* $\mathcal{R}$*, is* externally secure *if for all* $\lambda \in \mathbb{N}$*, all* $2m$*-input programs* $P \in \mathcal{P}$*, all* $\mathbf{x}_A, \mathbf{x}_B \in \mathcal{M}^m$*, and all efficient adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\mathsf{Adv}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B}(\lambda) := \left|\Pr\left[\ \mathsf{E}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B,A}(\lambda) = 1\ \right] - \Pr\left[\ \mathsf{E}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B,B}(\lambda) = 1\ \right]\right| \leq \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{E}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B,b}(\lambda)$ *is defined in Figure 5.3.*

Getting external security, generically. We now show a simple transformation for converting any MKHSS scheme $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ into an MKHSS scheme MKHSS^* that satisfies external security. The idea is to use a non-interactive key

exchange (NIKE) to derive a common pseudorandom key, which we use to randomize the output shares. Let $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer})$ be a NIKE scheme (cf. Definition 5.3.1) and let G be a PRG (note that MKHSS implies the existence of NIKE, generically [BGI+18]). We describe the transformation to external security in Figure 5.4; the main idea is to have $\mathsf{MKHSS}^*.\mathsf{Eval}$ derive a common pseudorandom string K, which is then used to randomize the output with the help of the PRG.

Claim. *The MKHSS scheme described in Figure 5.4 satisfies Definition 5.4.2 (external security).*

Proof (sketch). The proof of external security of MKHSS^* is almost immediate and can be shown with a simple hybrid argument. First, invoke the security of NIKE to replace K with a fresh random string. Second, invoke the PRG security to replace $G(K)$ with a fresh random value R. Finally, invoke the correctness of MKHSS to conclude that $\langle z \rangle_A^{\mathcal{R}} + R, \langle z \rangle_B^{\mathcal{R}} + R$ are distributed as pseudorandom subtractive shares of $P(\mathbf{x}_A, \mathbf{x}_B)$ over the ring $\mathcal{R}$. ■

5.4.3 Construction

In this section, we present the full MKHSS construction from DCR.

5.4.3.1 MKHSS construction

We present the full MKHSS construction in Figure 5.5. Each party samples a secret key $s_\sigma \xleftarrow{R} \{i \cdot N + 1 \mid 1 \leq i \leq N - 1\}$ such that $s_\sigma \equiv 1 \bmod N$. The public key pk_σ of each party consists of the group element $f_\sigma := g^{-s_\sigma}$ and public NIM encoding of s_σ. Alice and Bob then synchronize their keys and respective input shares as described in the overview. In particular, because the input shares are nearly identical to the input shares in the Paillier–ElGamal constructions of (non-multi-key) HSS [OSY21, RS21], the correctness of evaluation for branching programs is almost immediate. Moreover, security reduces to the semantic security of the Damgård–Jurik encryption scheme and the NIM scheme, which can both be based on the security of the DCR assumption in $\mathbb{Z}_{N^2}^*$.

Concrete performance estimates. We note that the construction in Figure 5.5 is potentially implementable. Finding ways to further optimize it is an interesting direction for future work. In particular, the main overhead of Figure 5.5 is exponentiation in $\mathbb{Z}_{N^3}^*$. Each RMS multiplication in our construction requires two exponentiations: one exponentiation in $\mathbb{Z}_{N^3}^*$ and one in $\mathbb{Z}_{N^2}^*$, which require roughly 30 milliseconds and 15 milliseconds on high-end hardware, respectively, when using a 3072-bit modulus N. Therefore, 45 to 60 milliseconds per multiplication gives a rough estimate of performance for a non-optimized implementation. This results in roughly 20 multiplications per second. In contrast, (non-multi-key) HSS can achieve upwards of 100 multiplications per second on high-end hardware [BCG+17].

Theorem 5.4.1. *Let λ is the security parameter and let $N = N(\lambda)$ be the output of GGen as defined in Figure 5.1. If the DCR assumption holds, then the construction described in Figure 5.5 is an MKHSS scheme for the class of polynomial sized RMS programs with bound $B < 2^{-\lambda} \cdot N$ and message space $\mathbb{Z}_B$.*

External-Security Transformation for MKHSS

Parameters. Let $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ be any MKHSS scheme, let $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer})$ be any NIKE scheme, and let $G : \{0,1\}^\lambda \to \mathcal{R}$ be a PRG.

$\mathsf{MKHSS}^*.\mathsf{Setup}(1^\lambda)$:

1 : $\mathsf{crs}_{\mathsf{mkhss}} \leftarrow \mathsf{MKHSS.Setup}(1^\lambda)$
2 : $\mathsf{crs}_{\mathsf{nike}} \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$
3 : $\mathsf{crs} := (\mathsf{crs}_{\mathsf{mkhss}}, \mathsf{crs}_{\mathsf{nike}})$
4 : **return** crs

$\mathsf{MKHSS}^*.\mathsf{KeyGen}(\mathsf{crs})$:

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{mkhss}}, \mathsf{crs}_{\mathsf{nike}})$
2 : $(\mathsf{pk}^{\mathsf{nike}}, \mathsf{sk}^{\mathsf{nike}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{crs}_{\mathsf{nike}})$
3 : $(\mathsf{pk}^{\mathsf{mkhss}}, \mathsf{sk}^{\mathsf{mkhss}}) \leftarrow \mathsf{MKHSS.KeyGen}(\mathsf{crs}_{\mathsf{mkhss}})$
4 : $\mathsf{pk}^* := (\mathsf{pk}^{\mathsf{nike}}, \mathsf{pk})$
5 : $\mathsf{sk}^* := (\mathsf{sk}^{\mathsf{nike}}, \mathsf{sk})$
6 : **return** $(\mathsf{pk}^*, \mathsf{sk}^*)$

$\mathsf{MKHSS}^*.\mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x)$:

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{mkhss}}, \_)$
2 : $(\llbracket x \rrbracket^\sigma_A, \llbracket x \rrbracket^\sigma_B) \leftarrow \mathsf{Share}(\mathsf{crs}_{\mathsf{mkhss}}, \sigma, \mathsf{pk}_\sigma, x)$
3 : **return** $(\llbracket x \rrbracket^\sigma_A, \llbracket x \rrbracket^\sigma_B)$

$\mathsf{MKHSS}^*.\mathsf{Eval}(\mathsf{crs}, \sigma, \mathsf{sk}^*_\sigma, \mathsf{pk}^*_{1-\sigma}, \llbracket \mathbf{x}_A \rrbracket^A_\sigma, \llbracket \mathbf{x}_B \rrbracket^B_\sigma, P)$:

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{mkhss}}, \mathsf{crs}_{\mathsf{nike}})$
2 : **parse** $\mathsf{pk}^*_{1-\sigma} = (\mathsf{pk}^{\mathsf{nike}}_{1-\sigma}, \mathsf{pk}^{\mathsf{mkhss}}_{1-\sigma})$
3 : **parse** $\mathsf{sk}^*_\sigma = (\mathsf{sk}^{\mathsf{nike}}_\sigma, \mathsf{sk}^{\mathsf{mkhss}}_\sigma)$
4 : $\langle z \rangle^{\mathcal{R}}_\sigma := \mathsf{Eval}(\mathsf{crs}, \sigma, \mathsf{sk}^{\mathsf{mkhss}}_\sigma, \mathsf{pk}^{\mathsf{mkhss}}_{1-\sigma}, \llbracket \mathbf{x}_A \rrbracket^A_\sigma, \llbracket \mathbf{x}_B \rrbracket^B_\sigma, P)$
5 : $K := \mathsf{KeyDer}(\mathsf{pk}^{\mathsf{nike}}_{1-\sigma}, \mathsf{sk}^{\mathsf{nike}}_\sigma)$
6 : **return** $\langle z \rangle^{\mathcal{R}}_\sigma + G(K)$

Figure 5.4: External-security transformation for MKHSS.

Construction of MKHSS from DCR

Public Parameters. Let $S_{\mathsf{sk}} := \{i \cdot N + 1 \mid 1 \le i \le N - 1\}$ be the secret key space and let B be a bound on the message space. Let $\mathsf{NIM} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ be a NIM scheme. We will use the algorithms $\mathsf{ExpLinEncS}$ and $\mathsf{ExpLinEncR}$ defined in Figure 5.6.

$\mathsf{MKHSS.Setup}(1^\lambda, w)$:

$(N, g) \leftarrow \mathsf{DJEG.Setup}(1^\lambda)$
$\mathsf{crs}_{\mathsf{nim}} \leftarrow \mathsf{NIM.Setup}(1^\lambda)$
$k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}} \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$
$\mathsf{crs} := (N, g, \mathsf{crs}_{\mathsf{nim}}, k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}})$
return crs

$\mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x)$:

parse $\mathsf{crs} = (N, g, \mathsf{crs}_{\mathsf{nim}})$
parse $\mathsf{pk}_\sigma = (\mathsf{pe}_\sigma, f_\sigma)$
$r, r' \xleftarrow{\mathrm{R}} \mathbb{Z}_N$
$\mathsf{ct} \leftarrow \mathsf{DJEG.FlipEncrypt}(f_\sigma, x, 5; r)$
$\mathsf{ct}' \leftarrow \mathsf{DJEG.Encrypt}(f_\sigma, x, 1; r')$
$[\![x]\!]^\sigma_\sigma := \big((x, r, r'), (\mathsf{ct}, \mathsf{ct}')\big)$
$[\![x]\!]^\sigma_{1-\sigma} := (\mathsf{ct}, \mathsf{ct}')$
return $([\![x]\!]^\sigma_A, [\![x]\!]^\sigma_B)$

$\mathsf{MKHSS.KeyGen}(\mathsf{crs})$:

parse $(N, g, \mathsf{crs}_{\mathsf{nim}})$ **from** crs
$s \xleftarrow{\mathrm{R}} S_{\mathsf{sk}},\ f := g^{-s}$
$(\mathsf{pe}, \mathsf{st}) \leftarrow \mathsf{NIM.Encode}(\mathsf{crs}_{\mathsf{nim}}, s)$
$\mathsf{pk} := (\mathsf{pe}, f)$
$\mathsf{sk} := (\mathsf{st}, s)$
return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![\mathbf{x}_A]\!]^A_\sigma, [\![\mathbf{x}_B]\!]^B_\sigma, P)$:

parse $(\mathsf{crs}_{\mathsf{nim}}, k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}})$ **from** crs
parse $\mathsf{sk}_\sigma = (\mathsf{st}_\sigma, s_\sigma)$
parse $\mathsf{pk}_{1-\sigma} = (\mathsf{pe}_{1-\sigma}, f_{1-\sigma})$
$f := (f_{1-\sigma})^{s_\sigma}$ ▷ Derive common key.
$\langle z \rangle_\sigma := \mathsf{NIM.Decode}(\mathsf{crs}_{\mathsf{nim}}, \mathsf{pe}_{1-\sigma}, \mathsf{st}_\sigma)$
$\mathbf{k}_\sigma := (\langle z \rangle_\sigma, 1)$ **if** $\sigma = A$ **else** $\mathbf{k}_\sigma := (\langle z \rangle_\sigma, 0)$
for $i \in [m]$:
  $\{\!\{x^{(i)}_\sigma\}\!\} := \mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x^{(i)}_\sigma]\!]^\sigma_\sigma)$
  $\{\!\{x^{(i)}_{1-\sigma}\}\!\} := \mathsf{ExpLinEncR}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x^{(i)}_{1-\sigma}]\!]^{1-\sigma}_\sigma)$
$\mathsf{ek}_\sigma := (k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}}, \mathbf{k}_\sigma)$
$\{\!\{\mathbf{x}\}\!\} := (\{\!\{x^{(1)}_A\}\!\}, \ldots, \{\!\{x^{(m)}_A\}\!\}, \{\!\{x^{(1)}_B\}\!\}, \ldots, \{\!\{x^{(m)}_B\}\!\})$
return $\mathsf{DEval}(\sigma, \mathsf{ek}_\sigma, \{\!\{\mathbf{x}\}\!\}, P)$

Figure 5.5: Construction of MKHSS from DCR.

Proof. We prove correctness and privacy in turn.

Correctness. Recall that the correctness property requires that parties obtain a subtractive

$\mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x]\!]^\sigma_\sigma)$:

1 : **parse** $[\![x]\!]^\sigma_\sigma = \big((x, r, r'), (\_, \_)\big)$
2 : **parse** $\mathsf{sk}_\sigma := (\_, s_\sigma)$
3 : **parse** $\mathsf{pk}_{1-\sigma} = (\_, f_{1-\sigma})$
4 : $(c_0, c_1) := \mathsf{DJEG.FlipEncrypt}(f_{1-\sigma}, x, w; r)$
5 : $(c'_0, c'_1) := \mathsf{DJEG.Encrypt}(f_{1-\sigma}, x, 2; r')$
6 : $\{\!\{x\}\!\} := ((c_0, (c_1)^{s_\sigma}), (c'_0, (c'_1)^{s_\sigma}))$
7 : **return** $\{\!\{x\}\!\}$

$\mathsf{ExpLinEncR}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x]\!]^{1-\sigma}_\sigma)$:

1 : **parse** $[\![x]\!]^{1-\sigma}_\sigma = \big((c_0, c_1), (c'_0, c'_1)\big)$
2 : **parse** $\mathsf{sk}_\sigma = (\_, s_\sigma)$
3 : $\{\!\{x\}\!\} := ((c_0, (c_1)^{s_\sigma}), (c'_0, (c'_1)^{s_\sigma}))$
4 : **return** $\{\!\{x\}\!\}$

Figure 5.6: Exponent-linear encoding algorithms used as subroutines in the MKHSS construction.

sharing of the program output upon evaluation.

We first prove that the encoding $\{\!\{x\}\!\}$ derived by the parties in $\mathsf{MKHSS.Eval}$ is (1) the same for both parties and (2) exponent-linear decodable.

Claim. *For all integers $x \in \mathbb{Z}_N$ and all $\sigma \in \{A, B\}$, we have*

$$\{\!\{x\}\!\} = \mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x]\!]^\sigma_\sigma) = \mathsf{ExpLinEncR}(\mathsf{sk}_{1-\sigma}, \mathsf{pk}_\sigma, [\![x]\!]^\sigma_{1-\sigma}),$$

where $([\![x]\!]^\sigma_A, [\![x]\!]^\sigma_B) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x)$. Moreover, $\{\!\{x\}\!\}$ is base-$(N+1)$ exponent-linear decodable under the decoding key $\mathbf{k} = (s_A \cdot s_B,\ 1)$.

Proof. We consider the case where $\sigma = A$; a symmetric argument follows for the other case.

Inspecting $\mathsf{MKHSS.Share}$, we have $[\![x]\!]^A_A = ((x, r, r'), (\mathsf{ct}, \mathsf{ct}'))$ and $[\![x]\!]^A_B = (\mathsf{ct}, \mathsf{ct}')$, where

$$\mathsf{ct} = ((N+1)^x \cdot g^r,\ f^r_A) \text{ and } \mathsf{ct}' = (g^{r'},\ (N+1)^x \cdot f^{r'}_A).$$

Party-A computes $\{\!\{x\}\!\}$ in $\mathsf{ExpLinEncS}$ as

$$\begin{aligned}
\{\!\{x\}\!\} &= \Big(((N+1)^x \cdot g^r,\ (f^r_B)^{s_A}),\ (g^{r'},\ ((N+1)^x \cdot f^{r'}_B)^{s_A})\Big) \in (\mathbb{Z}^*_{N^6})^2 \times (\mathbb{Z}^*_{N^2})^2 \\
&= \Big(((N+1)^x \cdot g^r,\ f^r),\ (g^{r'},\ (N+1)^{x \cdot s_A} \cdot f^{r' \cdot s_A}_B)\Big) \\
&= \Big(((N+1)^x \cdot g^r,\ f^r),\ (g^{r'},\ (N+1)^x \cdot f^{r'})\Big),
\end{aligned}$$

where the third equality follows from the fact that $s_A \equiv 1 \pmod N$ and $(N+1)$ has order N in $\mathbb{Z}^*_{N^2}$. Now, observe that party-B computes $\{\!\{x\}\!\}$ in $\mathsf{ExpLinEncR}$ as

$$\begin{aligned}
\{\!\{x\}\!\} &= \Big(((N+1)^x \cdot g^r,\ (f^r_A)^{s_B}),\ (g^{r'},\ ((N+1)^x \cdot f^{r'}_A)^{s_B})\Big) \in (\mathbb{Z}^*_{N^6})^2 \times (\mathbb{Z}^*_{N^2})^2 \\
&= \Big(((N+1)^x \cdot g^r,\ f^r),\ (g^{r'},\ (N+1)^{x \cdot s_B} \cdot f^{r' \cdot s_B}_A)\Big) \\
&= \Big(((N+1)^x \cdot g^r,\ f^r),\ (g^{r'},\ (N+1)^x \cdot f^{r'})\Big).
\end{aligned}$$

Therefore, both parties obtain the same encoding $\{\!\{x\}\!\}$.

We are left to show that $\{\{x\}\} = (\mathbf{c}_0, \mathbf{c}_1)$ is base-$(N+1)$ exponent-linear decodable under $\mathbf{k} = (k_1, k_2) = (s_A \cdot s_B,\ 1)$. Observe that

$$\langle \mathbf{c}_0, \mathbf{k} \rangle = ((N+1)^x \cdot g^r)^{s_A \cdot s_B} \cdot f^r = (N+1)^{x \cdot s_A \cdot s_B} \cdot g^{r \cdot s_A \cdot s_B} \cdot g^{-s_A \cdot s_B \cdot r} = (N+1)^{x \cdot s_A \cdot s_B} \in \mathbb{Z}^*_{N^6},$$
$$\langle \mathbf{c}_1, \mathbf{k} \rangle = ((g^{r'})^{s_A \cdot s_B} \cdot (N+1)^x \cdot f^{r'} = g^{r \cdot s_A \cdot s_B} \cdot (N+1)^x \cdot g^{-s_A \cdot s_B \cdot r} = (N+1)^x \in \mathbb{Z}^*_{N^2},$$

which proves that $\{\{x\}\}$ is base-$(N+1)$ exponent-linear decodable. (Recall that we abuse notation by denoting $\langle \cdot, \cdot \rangle$ as computing the inner product "in the exponent" of the group.) In particular, $s_A \cdot s_B \leq ((N-1) \cdot N)^2 < N^4$. Furthermore, because $x \leq N$, we have that $x \cdot s_A \cdot s_B$ does not overflow modulo N^5. □

Finally, to complete the proof of correctness, it suffices to note that by the correctness of NIM, the parties obtain subtractive shares of $s_A \cdot s_B$, and so $\mathbf{k}_\sigma$ is a subtractive share of $\mathbf{k}$ as defined above.

In sum, it follows that parties run DEval with encodings of the input that are base-$(N+1)$ exponent-linear decodable. Finally, since $B < 2^{-\lambda} \cdot N$ and DDLog is a B-bounded (resp. $(B \cdot N^4)$-bounded) base-$(N+1)$ algorithm for distributed discrete logarithm with negligible correctness error in $\mathbb{Z}^*_{N^2}$ (resp. $\mathbb{Z}^*_{N^6}$), it follows from Lemma 5.3.3 that the MKHSS scheme satisfies the correctness property for all polynomial-size RMS programs P.

Security. Recall that the security property requires that the input share $[\![x]\!]^\sigma_{1-\sigma}$ of party-$(1-\sigma)$, ensures the privacy of an input x shared using party-σ's public key pk_σ.

Consider any efficient adversary $\mathcal{A}$ for the security experiment defined in Definition 5.4.1. Let the output of the security experiment be defined as 1 if $\mathcal{A}$'s output b' is equal to the challenge bit b; else let the output of the experiment be defined as 0. We will use a hybrid argument to show that the output of the experiment is 1 with probability of at most $1/2 + \mathsf{negl}(\lambda)$.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the output of the experiment when run with adversary $\mathcal{A}$ when the challenge bit is $b = 0$.

- *Hybrid* $\mathcal{H}_1$. This hybrid game is identical to the previous hybrid, except that the secret key s_σ is sampled uniformly at random from $[N]$ in $\mathsf{MKHSS.KeyGen}$, which matches the distribution of the secret key in the DJEG encryption scheme.

  Claim. $\mathcal{H}_1 \approx_s \mathcal{H}_0$.

  Proof. Note that in $\mathcal{H}_0$ the public key is computed as $f = g^{N \cdot i} g$ for some $i \in [N-1]$ while in $\mathcal{H}_1$ it is computed as g^i for $i \in [N]$. Because g has order $\phi(N)/4$, and for a random $i \in [N-1]$, $i \pmod{\phi(N)/4}$ is statistically close to $i \cdot N \bmod \phi(N)/4$ (since $\phi(N)$ is co-prime to N), it follows that g^i and $(g^N)^i$ are both statistically close to the uniform distribution. To conclude the proof, it suffices to note that $(g^N)^i \cdot g$ is also close to uniform. □

- *Hybrid* $\mathcal{H}_2$. In this hybrid game, pe is replaced with an encoding of zero. That is, $(\mathsf{pe}_\sigma, \_) \leftarrow \mathsf{NIM.Encode}(\mathsf{crs}_{\mathsf{nim}}, 0)$.

Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$ *assuming the security of NIM.*

Proof. The claim follows immediately from the security of the NIM scheme. □

- *Hybrid* $\mathcal{H}_3$. In this hybrid game, we replace the DJEG encryptions with encryptions of x_1.

  Claim. $\mathcal{H}_3 \overset{s}{\approx} \mathcal{H}_2$ *by the semantic security of the DJEG encryption scheme.*

  Proof. The claim follows by a straightforward hybrid argument replacing the two encryptions of x_0 with encryptions of x_1 and invoking the semantic security of the DJEG scheme. □

- *Hybrid* $\mathcal{H}_4$. In this hybrid game, we reverse the changes made in $\mathcal{H}_2$ and encode the secret key s using the NIM scheme.

  Claim. $\mathcal{H}_4 \approx_c \mathcal{H}_3$ *assuming the security of NIM.*

  Proof. The claim follows immediately from the security of the NIM scheme. □

- *Hybrid* $\mathcal{H}_5$. In this hybrid game, we reverse the changes made in $\mathcal{H}_1$ and sample the secret key as in the construction.

  Claim. $\mathcal{H}_5 \approx_c \mathcal{H}_4$.

  Proof. The proof follows the same argument used to prove that $\mathcal{H}_1 \approx_c \mathcal{H}_0$. □

To complete the proof, observe that $\mathcal{H}_5$ is exactly the output of the experiment when the challenge bit $b = 1$. Since we have shown that $\mathcal{H}_0 \approx_c \mathcal{H}_5$, it follows that $\mathcal{A}$ wins the MKHSS security game with probability of at most $1/2 + \mathsf{negl}(\lambda)$. ■

5.4.4 Sublinear, two-round secure computation

In this section, we discuss how our above MKHSS construction can be applied to achieve a sublinear, two-round, two-party secure computation protocol. Sublinear here means that the communication cost is bounded by a fixed polynomial in the total length of the inputs, outputs and the security parameter, but is independent of the circuit evaluated. We refer the reader to Goldreich [Gol06] for the standard security definitions of two-party secure computation.

An MKHSS scheme with negligible correctness error immediately implies a sublinear two-round secure computation protocol, as outlined in Section 5.1.1. Consequently, we obtain the following corollary of Theorem 5.4.1.

Corollary 5.4.1 (Sublinear protocols for RMS programs)**.** *Under the DCR assumption, there exists a sublinear two-round, two-party, secure computation protocol with semi-honest security for evaluating polynomial-size RMS programs in the common reference string model.*

5.5 Attribute-Based Non-Interactive Key Exchange

An intriguing application of MKHSS is the ability to perform policy-based key-exchange. In particular, two parties, Alice and Bob, each have secret attributes x_A and x_B, respectively. For a public predicate C, Alice and Bob obtain the same secret key if and only if $C(x_A, x_B) = 0$ (the predicate is satisfied), and independently distributed keys otherwise. In this process, nothing about Bob's secret attribute is leaked to Alice, as from her perspective she always receives a random key, and vice versa.

Kolesnikov et al. [KKL+16] present an *interactive* solution for this problem using using garbled circuits and supporting general predicates.[6] Many related notions of attribute-based key-exchange also exist, including witness-authenticated key exchange (see the overview of Melissaris [Mel22]). We also note that attribute-based key exchange generalizes the widely-used notion of password-authenticated key exchange, where the predicate essentially checks that Alice and Bob hold the same secret attribute (or password).

To the best of our knowledge, we construct the first attribute-based *non-interactive* key-exchange (ANIKE) protocol for NC^1 predicates in the standard model. In particular, we show that MKHSS for polynomial-size RMS programs implies ANIKE with predicates from the same function class.

We present a universally composable (UC) corruptible ideal functionality for attribute-based non-interactive key exchange, which we then instantiate using MKHSS. This is a more desirable security guarantee for key exchange than a weaker property-based definition, since it composes with other primitives (key exchange is often used as a building block in larger protocols).

The ideal functionality is defined as follows. In the initialization phase (which happens once), the functionality receives an attribute x from every party. The adversary is assumed to statically corrupt an arbitrary subset of these parties. In the key exchange phase (which is repeatable), and instantiated between a pair of parties Alice and Bob, the functionality receives a request from Alice and Bob, which consists of a predicate, and outputs a fresh key to each party. If repeated with the same predicate, the functionality outputs the same keys deterministically. The pair of keys output by the functionality are defined as follows, depending on whether the parties are honest or corrupted and whether the predicate is satisfied.

First we consider the case where both parties are honest:

- If Alice and Bob's attributes satisfy the predicate, the functionality samples a fresh key k which it sends to both parties.
- If their attributes do not satisfy the predicate, the functionality independently samples two keys k_A and k_B, then sends k_A to Alice and k_B to Bob.

Second, we consider the case where one of the parties is corrupted:

- If both parties' attributes satisfy the predicate, the adversary gets to specify the key k, which the functionality sends to both parties.
- If their attributes do not satisfy the predicate, the functionality samples a uniformly

[6] They define attribute-based key exchange in a client-server model, which is equivalent to our notion.

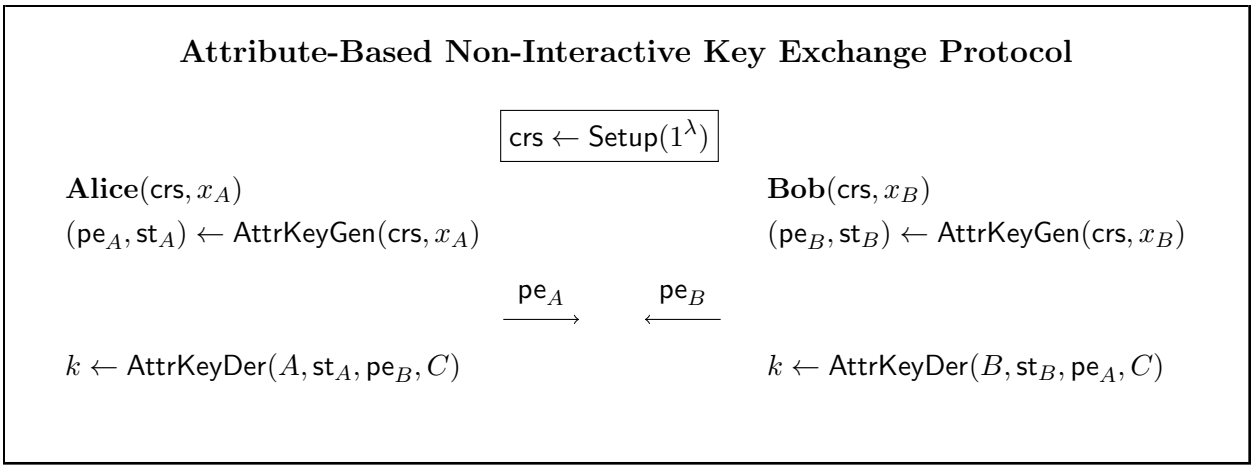


Figure 5.7: The ANIKE protocol using the algorithms specified in Definition 5.5.1.

random key to send to the uncorrupted party.

We refer to Functionality 1 for the full specification and define the algorithms, properties, and key exchange protocol that we will instantiate in Definition 5.5.1.

Definition 5.5.1 (Attribute-Based Non-Interactive Key Exchange)**.** *An attribute-based non-interactive key exchange (ANIKE) protocol with attribute space $\mathcal{X}$ consists of three efficient algorithms* $(\mathsf{Setup}, \mathsf{AttrKeyGen}, \mathsf{AttrKeyDer})$*, which are used to instantiate the two-party protocol described in Figure 5.7, and which have the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$*. The randomized setup algorithm takes as input the security parameter and outputs a common reference string (CRS)* crs*.*
- $\mathsf{AttrKeyGen}(\mathsf{crs}, x) \to (\mathsf{st}_\sigma)$*. The randomized attribute encoding algorithm takes as input the CRS* crs *and an attribute $x \in \mathcal{X}$. It outputs a public encoding* pe *and private state* st*.*
- $\mathsf{AttrKeyDer}(\sigma, \mathsf{st}_\sigma, \mathsf{pe}_{1-\sigma}, C) \to k$*. The deterministic key derivation algorithm takes as input the party identifier $\sigma \in \{A, B\}$, the CRS* crs*, the party's secret state* st_σ*, the other party's public encoding* $\mathsf{pe}_{1-\sigma}$*, and a circuit C describing the predicate. It outputs a key k.*

Security. We say that the ANIKE protocol is secure if it realizes the corruptible ideal functionality described in Functionality 1 against a semi-honest adversary assuming an authenticated channel.

Construction. Our construction is parameterized by an MKHSS scheme supporting polynomial-size RMS programs. We assume, without loss of generality, that the MKHSS message space $\mathcal{M}$ is a finite field $\mathbb{F}$, such that $|\mathbb{F}| \geq 2^\lambda$. Such a setup can always be achieved by working in the field extension of $\mathbb{F}_2$.

Remark 28. *For simplicity, in our construction, we let the algorithm* $\mathsf{AttrKeyGen}$ *be parameterized by a party identifier $\sigma \in \{A, B\}$. This allows us to construct* $\mathsf{AttrKeyGen}$

Functionality $\mathcal{F}_{\mathsf{anike}}$

Parties. The functionality is parameterized by a set of parties and an adversary $\mathcal{A}$ that statically corrupts an arbitrary subset of the parties.

Procedure. The functionality aborts if it receives any incorrectly formatted messages.

- *One-time initialization phase.*

  1: Receive a message $(\texttt{Init}, \mathsf{id}_\sigma, x)$ containing an attribute x from every party with identifier σ.
  2: Initialize a lookup table of generated keys T.
  3: Send ready to $\mathcal{A}$.

- *Repeatable key exchange phase between Alice and Bob.*

  1: Receive messages $(\texttt{KeyAgree}, \mathsf{id}_B, C)$ from Alice and $(\texttt{KeyAgree}, \mathsf{id}_A, C)$ from Bob, where C is a circuit describing a predicate.
  2: Receive a message from $\mathcal{A}$, which is either empty, contains a key and an identifier $\sigma \in \{\mathsf{id}_A, \mathsf{id}_B\}$, or contains two keys and both identifiers.
  3: If $(\mathsf{id}_A, \mathsf{id}_B, C) \in T$:
    3.1 Set $(k_A, k_B) := T[(\mathsf{id}_A, \mathsf{id}_B, C)]$.
  4: Else if $\mathcal{A}$ sent an empty message, i.e., the Alice and Bob are both honest:
    4.1 If $C(x_A, x_B) = 0$: sample k_A uniformly and set $k_B = k_A$.
    4.2 Else if $C(x_A, x_B) = 1$: sample k_A, k_B uniformly and independently.
  5: Else if $\mathcal{A}$ sent k_σ, i.e., party-σ is corrupted and party $\overline{\sigma} \in \{A, B\} \setminus \{\sigma\}$ is honest:
    5.1 If $C(x_A, x_B) = 0$: set $k_{\overline{\sigma}} := k_\sigma$.
    5.2 Else if $C(x_A, x_B) = 1$: sample $k_{\overline{\sigma}}$ uniformly.
  6: If $\mathcal{A}$ sent k_A and k_B, i.e., both parties are corrupted:
    6.1 Do nothing.
  7: Set $T[(\mathsf{id}_A, \mathsf{id}_B, C)] = (k_A, k_B)$.
  8: Output k_A to Alice and k_B to Bob.

Functionality 1: Corruptible ideal functionality for attribute-based non-interactive key exchange.

"asymmetrically" by having it be defined differently depending on whether Alice or Bob runs it. This change is without loss of generality, since the party-agnostic version of $\mathsf{AttrKeyGen}$ *can be recovered by having the parties play both roles simultaneously and agree on their respective roles in the key-derivation phase.*

To encode her attribute x_A, Alice maps it into the field $\mathbb{F}$. She also samples a PRF key K which will be used to generate pseudorandom shifts $\Delta_A \in \mathbb{F}$ in the key derivation phase. Specifically, these shifts are used to ensure the derived keys are uniformly random when the

predicate C is unsatisfied.

Alice then samples MKHSS keys $(\mathsf{pk}_A, \mathsf{sk}_A)$ and computes shares $(\llbracket K_A \| x_A \rrbracket_A^A, \llbracket K_A \| x_A \rrbracket_B^A)$ of $K_A \| x_A$. Her public encoding consists of her MKHSS public key pk_A and Bob's share $\llbracket K_A \| x_A \rrbracket_B^A$, while her private encoding consists of her MKHSS secret key sk_A and her own share $\llbracket K_A \| x_A \rrbracket_A^A$. Bob encodes his attribute x_B analogously.

Given her own private encoding and Bob's public encoding, Alice now holds her MKHSS secret key sk_A, Bob's MKHSS public key pk_B, and her shares $\llbracket K_A \| x_A \rrbracket_A^A$ and $\llbracket K_B \| x_B \rrbracket_A^B$. She homomorphically evaluates the program P_C, where P_C is defined as:

$$P_C(K_A || x_A, K_B || x_B) = \underbrace{F_{K_A}(A_{\mathsf{id}} \| B_{\mathsf{id}} \| C)}_{\Delta_A} \cdot \underbrace{F_{K_B}(A_{\mathsf{id}} \| B_{\mathsf{id}} \| C)}_{\Delta_B} \cdot C(x_A, x_B).$$

That is, P_C computes the predicate $C(x_A, x_B)$ and then multiplies the result by $\Delta_A \cdot \Delta_B$, derived from the PRF. Bob symmetrically evaluates his shares for the same program P_C using his MKHSS secret key sk_B and Alice's MKHSS public key pk_A. Thus, if $C(x_A, x_B) = 0$, Alice and Bob end up with subtractive shares of 0, i.e., the same key. On the other hand, if $C(x_A, x_B) \neq 0$, Alice and Bob end up with shares of $\Delta_A \cdot \Delta_B$, i.e., independent pseudorandom keys.

We refer to Figure 5.8 for a formal description of our construction.

Theorem 5.5.1. *Assuming the existence of an MKHSS scheme* MKHSS *for polynomial-size RMS programs and the existence of PRFs in* NC^1*, the construction described in Figure 5.8 is an attribute-based non-interactive key exchange supporting predicates described by polynomial-size RMS programs.*

Proof. We show that Figure 5.8 securely realizes the functionality $\mathcal{F}_{\mathsf{anike}}$ described in Functionality 1 by constructing a simulator which simulates the view of the corrupted party and interacts with $\mathcal{F}_{\mathsf{anike}}$ on behalf of the ideal adversary.

- *Initialization phase.* For every corrupted party, the simulator obtains their attribute x and sends it to $\mathcal{F}_{\mathsf{anike}}$ on behalf of the ideal adversary.

  We now emulate a key derivation phase between two parties, Alice and Bob.

- *Case 1: Both parties are honest.* By correctness of MKHSS, we have that

  $$k_A - k_B = P_C(\Delta_A \| x_A, \Delta_B \| x_B) = C(x_A, x_B) \cdot \Delta_A \cdot \Delta_B \in \mathbb{F}.$$

  Thus, if $C(x_A, x_B) = 0$, we have that $k_A = k_B$, with all but negligible probability. Moreover, the tuple (k_A, k_B) is computationally indistinguishable from a uniformly random tuple $(k_A, k_B) \in \mathbb{F} \times \mathbb{F}$ subject to $k_A = k_B$, by the external security Definition 5.4.2. As such, when the predicate is satisfied, k_A and k_B matches the output of the ideal functionality.

  On the other hand, if $C(x_A, x_B) \neq 1$, we have $k_A = k_B + C(x_A, x_B) \cdot \Delta_A \cdot \Delta_B$, where Δ_A and Δ_B are the secret pseudorandom shifts output by the PRF evaluated under independent keys (by the correctness of MKHSS and the definition of the evaluated program P_C).

Attribute-Based Non-Interactive Key Exchange from MKHSS

Public Parameters. Let $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ be an MKHSS scheme with external security (cf. Definition 5.4.2) for polynomial-size RMS programs defined over the finite field $\mathbb{F}$, where $|\mathbb{F}| \geq 2^\lambda$. Let $F : \{0,1\}^\lambda \times \{0,1\}^\star \to \mathbb{F}$ be a PRF. Let $F_K \colon \{0,1\}^\star \to \mathbb{F}$ be a PRF with keys sampled from $\{0,1\}^\lambda$, such that $F_k(x)$ is computable by a polynomial-size RMS program over $\mathbb{F}$.

The evaluated program. Define P_C to be the program that, on input $K_A \| x_A, K_B \| x_B$ outputs $F_{K_A}(A_{\mathsf{id}} \| B_{\mathsf{id}} \| C) \cdot F_{K_B}(A_{\mathsf{id}} \| B_{\mathsf{id}} \| C) \cdot C(x_A, x_B)$, where C is the attribute predicate.

$\mathsf{NIKE.Setup}(1^\lambda)$:

1 : $\mathsf{crs} \leftarrow \mathsf{MKHSS.Setup}(\lambda)$
2 : **return** crs

$\mathsf{NIKE.AttrKeyGen}(\mathsf{crs}, \sigma, x)$:

1 : $K \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MKHSS.KeyGen}(\mathsf{crs})$
3 : $(\llbracket K \| x \rrbracket^\sigma_A, \llbracket K \| x \rrbracket^\sigma_B) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}, (K \| x))$
4 : $\mathsf{pe} := (\mathsf{pk}, \llbracket K \| x \rrbracket^\sigma_{1-\sigma})$
5 : $\mathsf{st} := (\mathsf{sk}, \llbracket K \| x \rrbracket^\sigma_\sigma)$
6 : **return** $(\mathsf{pe}, \mathsf{st})$

$\mathsf{NIKE.AttrKeyDer}(\sigma, \mathsf{st}_\sigma, \mathsf{pe}_{1-\sigma}, C)$:

1 : **parse** $\mathsf{pe}_{1-\sigma} = (\mathsf{pk}_{1-\sigma}, \llbracket K_{1-\sigma} \| x_{1-\sigma} \rrbracket^{1-\sigma}_\sigma)$
2 : **parse** $\mathsf{st}_\sigma = (\mathsf{sk}_\sigma, \llbracket K_\sigma \| x_\sigma \rrbracket^\sigma_\sigma)$
3 : $k \leftarrow \mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, \llbracket K_A \| x_A \rrbracket^A_\sigma, \llbracket K_B \| x_B \rrbracket^B_\sigma, P_C)$
4 : **return** k

Figure 5.8: Attribute-based non-interactive key exchange from MKHSS.

We proceed to show that (k_A, k_B) is computationally indistinguishable from a random tuple via a simple hybrid argument:

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the tuple (k_A, k_B), as computed using AttrKeyDer by each party in Figure 5.8.
- *Hybrid* $\mathcal{H}_1$. In this hybrid, we replace k_B with a uniformly random key and set

$k_A = k_B + C(x_A, x_B) \cdot \Delta_A \cdot \Delta_B$. This hybrid is computationally indistinguishable from the previous one by the external security of MKHSS.

– *Hybrid* $\mathcal{H}_2$. In this hybrid, we sample (k_A, k_B) as a uniformly random tuple over $\mathbb{F} \times \mathbb{F}$. This hybrid is computationally indistinguishable from the previous one by the pseudorandomness of $\Delta_A \cdot \Delta_B$ (which are generated internally using the PRF). To see this, it suffices to note that, in $\mathcal{H}_1$, we already have that k_B is computationally indistinguishable from a random field element conditioned on k_A, since we can view $\Delta_A \cdot \Delta_B$ as being a uniformly random element.

At this point, it suffices to note that $\mathcal{H}_2$ is distributed identically to the output of the ideal functionality when the predicate is not satisfied. This concludes the proof for the case where both parties are honest.

- *Case 2: Alice is corrupted.* The simulator computes $(\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{MKHSS.KeyGen}(\mathsf{crs})$ and $(\llbracket \mathbf{0} \rrbracket_B^A, \llbracket \mathbf{0} \rrbracket_B^B) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, B, \mathsf{pk}_B, \mathbf{0})$, to define $\mathsf{pe}_B := (\mathsf{pk}_B, \llbracket \mathbf{0} \rrbracket_A^B)$, where $\mathbf{0} := 0^{\lambda + |x|}$. The simulated view of Alice consists of crs and pe_B. Eventually, the simulator recovers x_A and $\mathsf{st}_A = (\mathsf{sk}_A, \llbracket K_A \| x_A \rrbracket_A^A)$. It computes

$$k_A := \mathsf{MKHSS.Eval}(\mathsf{crs}, A, \mathsf{sk}_A, \mathsf{pk}_B, \llbracket K_A \| x_A \rrbracket_A^A, \llbracket \mathbf{0} \rrbracket_A^B),$$

and sends k_A to the functionality on behalf of the ideal adversary. The functionality outputs k_A to Alice and k_B to Bob. By the security of MKHSS and a straightforward hybrid argument, the joint distribution of Bob's message and the output of both parties in the real world is indistinguishable from the simulation of Bob's message and the output of the ideal functionality.

- *Case 3: Bob is corrupted.* This case follows by symmetry.

■

Corollary 5.5.1. *Assuming the existence of PRFs in* NC^1*, there exists an attribute-based non-interactive key exchange scheme supporting predicates described by polynomial-size RMS programs under the DCR assumption.*

5.6 Public-Key PCFs from MKHSS and Applications

Practical secure computation protocols are realized in the preprocessing model [DPSZ12]: During an "offline" preprocessing phase, the computing parties generate a large amount of *pseudorandom correlations* that are independent of any function. Then, during an online phase, the parties use the stored correlations to compute a function over their inputs in a secure protocol. The advantage of this model is that it pushes the bulk of the communication and computation costs of the function-dependent online phase to the preprocessing phase. Pseudorandom correlation generators (PCGs) and functions (PCFs) push this model of secure computation to the limit by allowing parties to locally expand a short key into a virtually unbounded amount of correlated randomness. However, traditional approaches still require the parties to run an interactive protocol to generate their key.

Public-key PCFs. A *public key* PCF (PK-PCF) is a PCF equipped with a non-interactive key distribution protocol. Public-key PCFs were originally introduced in the work of Orlandi et al. [OSY21] and formalized in the recent work of Bui et al. [BCM+24]. PK-PCFs are motivated by their direct application to secure computation in the preprocessing model.

Let $\mathcal{Y}$ be a correlation such that a secure access to random samples from $\mathcal{Y}$ enables efficient information-theoretic two-party computation (typically, $\mathcal{Y}$ can sample an oblivious transfer correlation, Beaver triples, authenticated Beaver triples, or other types of correlated randomness, depending on the application at hand). A PK-PCF for $\mathcal{Y}$ induces the following appealing template for communication-efficient secure two-party computation:

> **Non-interactive preprocessing.** Ahead of time, all participants P_i of a secure computation network upload their PK-PCF public key pk_i to some public bulletin board.
>
> **Fast, online secure computation.** Whenever two parties P_i and P_j want to securely compute a function, they can retrieve each other's public keys, non-interactively derive correlated PCF keys, and generate as many pseudorandom samples from $\mathcal{Y}$ as they need to enable a fast online phase for a two-party protocol in the correlated randomness model (e.g., GMW [GMW87] or SPDZ [DPSZ12]).

In this section, we construct PK-PCFs for all correlations in NC^1. As another application of our construction, in Section 5.6.3, we show a protocol for generating *multi-party* correlations with quadratic improvement in communication relative to the prior constructions of PCFs.

5.6.1 Public-key pseudorandom correlation functions

In this section, we provide some background and a formal definition of PK-PCFs, following the formalization of Bui et al. [BCM+24].

The standard PCF (and PK-PCF) definition requires the correlation to be "reverse sampleable" which, roughly speaking, means that given any (possibly adversarially generated) share of the target correlation, the other (honest) share can be efficiently sampled. We note that all additive correlations, which will be the target of our constructions, are reverse sampleable.

Remark 29 (Notation)**.** *In this section, we will identify the two parties using indices* 0 *and* 1*, instead of letters A and B as we did in previous sections. This simplifies notation when we define* multi-party *PCFs in Section 5.6.3.*

Definition 5.6.1 (Reverse-Sampleable Correlation)**.** *Let λ be a security parameter and $n = n(\lambda) \in \mathsf{poly}(\lambda)$ be an output length. Define two efficient algorithms $\mathcal{Y}$ and* RSample *with the following syntax:*

- $\mathcal{Y}(1^\lambda) \to (y^0, y^1)$*. The randomized correlation sampling algorithm takes as input the security parameter and outputs a pair $(y^0, y^1) \in \{0,1\}^n \times \{0,1\}^n$ defining a correlation.*
- $\mathsf{RSample}(1^\lambda, \sigma, y^\sigma) \to y^{1-\sigma}$*. The deterministic reverse-sampling algorithm takes as input the security parameter, an index $\sigma \in \{0,1\}$, and a string $y^\sigma \in \{0,1\}^n$. It outputs a string $y^{1-\sigma} \in \{0,1\}^n$.*

We say that $\mathcal{Y}$ defines a reverse-sampleable *correlation if for all $\sigma \in \{0,1\}$, it holds that:*

$$\left\{ (y^0, y^1) \;\middle|\; (y^0, y^1) \leftarrow \mathcal{Y}(1^\lambda) \right\} \approx_s \left\{ (y^0, y^1) \;\middle|\; \begin{array}{r} (\hat{y}^0, \hat{y}^1) \leftarrow \mathcal{Y}(1^\lambda) \\ y^\sigma := \hat{y}^\sigma \\ y^{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y^\sigma) \end{array} \right\}.$$

$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{pkPCF.Setup}(1^\lambda)$
$(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{pkPCF.KeyGen}(\mathsf{crs}, \sigma),\ \forall \sigma \in \{0,1\}$
foreach $i \in [N]$:
  $x_i \xleftarrow{\mathrm{R}} \{0,1\}^n$
  $(y_i^0, y_i^1) \leftarrow \mathcal{Y}(1^\lambda)$
$b \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1, (x_i, y_i^0, y_i^1)_{i\in[N]})$
return b

$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{pkPCF.Setup}(1^\lambda)$
$(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{pkPCF.KeyGen}(\mathsf{crs}, \sigma),\ \forall \sigma \in \{0,1\}$
$K_\sigma := \mathsf{pkPCF.KeyDer}(\mathsf{crs}, \sigma, \mathsf{pk}_{1-\sigma}, \mathsf{sk}_\sigma),\ \forall \sigma \in \{0,1\}$
foreach $i \in [N]$:
  $x_i \xleftarrow{\mathrm{R}} \{0,1\}^n$
  $y_i^\sigma := \mathsf{pkPCF.Eval}(\mathsf{crs}, \sigma, K_\sigma, x_i),\ \forall \sigma \in \{0,1\}$
$b \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1, (x_i, y_i^0, y_i^1)_{i\in[N]})$
return b

Figure 5.9: Pseudorandom $\mathcal{Y}$-correlated outputs for a weak PK-PCF.

$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{pkPCF.Setup}(1^\lambda)$
$(\mathsf{pk}_{\hat{\sigma}}, \mathsf{sk}_{\hat{\sigma}}) \leftarrow \mathsf{pkPCF.KeyGen}(\mathsf{crs}, \hat{\sigma}),\ \forall \hat{\sigma} \in \{0,1\}$
$K_{1-\sigma} := \mathsf{pkPCF.KeyDer}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathsf{sk}_{1-\sigma})$
foreach $i \in [N]$:
  $x_i \xleftarrow{\mathrm{R}} \{0,1\}^n$
  $y_i^{1-\sigma} := \mathsf{pkPCF.Eval}(\mathsf{crs}, 1-\sigma, K_{1-\sigma}, x_i)$
$b \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1, \sigma, \mathsf{sk}_\sigma, (x_i, y_i^{1-\sigma})_{i\in[N]})$
return b

$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{pkPCF.Setup}(1^\lambda)$
$(\mathsf{pk}_{\hat{\sigma}}, \mathsf{sk}_{\hat{\sigma}}) \leftarrow \mathsf{pkPCF.KeyGen}(\mathsf{crs}, \hat{\sigma}),\ \forall \hat{\sigma} \in \{0,1\}$
$K_\sigma := \mathsf{pkPCF.KeyDer}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathsf{sk}_{1-\sigma})$
foreach $i \in [N]$:
  $x_i \xleftarrow{\mathrm{R}} \{0,1\}^n$
  $y_i^\sigma := \mathsf{pkPCF.Eval}(\mathsf{crs}, \sigma, K_\sigma, x_i)$
  $y_i^{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_i^\sigma)$
$b \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1, \sigma, \mathsf{sk}_\sigma, (x_i, y_i^{1-\sigma})_{i\in[N]})$
return b

Figure 5.10: Security of game for a weak PK-PCF. Here, RSample is as defined in Definition 5.6.1.

Definition 5.6.2 (Public-Key Pseudorandom Correlation Function [BCM+24])**.** *Let λ be a security parameter, $\mathcal{Y}$ be a reverse-sampleable correlation with output length $n = n(\lambda) \in \mathsf{poly}(\lambda)$, and $\lambda \leq m = m(\lambda) \in \mathsf{poly}(\lambda)$ be an input length. A Public-Key Pseudorandom Correlation Function (PK-PCF) for $\mathcal{Y}$ is defined by a tuple of algorithms $\mathsf{pkPCF} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{KeyDer}, \mathsf{Eval})$ with the following functionality:*

- $\mathsf{pkPCF.Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter λ and outputs a common reference string (CRS)* crs.
- $\mathsf{pkPCF.KeyGen}(\mathsf{crs}, \sigma) \to (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$. *The randomized key generation algorithm takes as input the CRS* crs *and a party identifier $\sigma \in \{0,1\}$. It outputs a public and secret key pair $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$ for the party.*

- $\mathsf{pkPCF.KeyDer}(\mathsf{crs}, \sigma, \mathsf{pk}_{1-\sigma}, \mathsf{sk}_\sigma) \to K_\sigma$. *The deterministic key derivation algorithm takes as input the CRS* crs*, a party identifier* $\sigma \in \{0,1\}$*, the public key* $\mathsf{pk}_{1-\sigma}$ *of another party, and the secret key* sk_σ *of this party. It outputs an evaluation key* K_σ *for this party.*
- $\mathsf{pkPCF.Eval}(\mathsf{crs}, \sigma, K^\sigma, x) \to y_\sigma$. *The deterministic evaluation algorithm takes as input the CRS, the party identifier* $\sigma \in \{0,1\}$*, an evaluation key* K^σ*, and an input* $x \in \{0,1\}^m$*. It outputs a string* $y^\sigma \in \{0,1\}^n$*.*

We say $\mathsf{pkPCF} = (\mathsf{KeyGen}, \mathsf{Eval})$ *is a PK-PCF for the correlation* $\mathcal{Y}$*, if the following two properties hold:*

Correctness / Pseudorandom* $\mathcal{Y}$*-correlated outputs. *For every* $\sigma \in \{0,1\}$*, all efficient adversaries* $\mathcal{A}$*, and all* $N = N(\lambda) \in \mathsf{poly}(\lambda)$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\mathsf{Adv}^{\mathsf{pr}}_{\mathcal{A},N}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,b}(\lambda)$*, for* $b \in \{0,1\}$*, is as defined in Figure 5.9. In particular, the adversary is given access to* N *samples.*

Security. *For all* $\sigma \in \{0,1\}$*, and all efficient adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\mathsf{Adv}^{\mathsf{sec}}_{\mathcal{A},N,\sigma}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,b}(\lambda)$*, for* $b \in \{0,1\}$*, is as defined in Figure 5.10 (again, with the adversary given* N *samples).*

Remark 30 (Weak vs. Strong PCFs)**.** *We remark that, contrary to pseudorandom functions, the default notion of a PCF is a* weak *PCF, where the inputs are chosen uniformly at random. A PCF (as defined in Definition 5.6.2) can be generically converted to a strong PCF using a random oracle.*

5.6.2 Public-key PCFs from MKHSS

In this section, we provide a construction of a PK-PCF for any additive correlations computable by RMS programs (which includes the class NC^1). We start by defining additive correlations:

Definition 5.6.3 (Additive Correlation)**.** *Let* λ *be a security parameter, and let* $n_y = n_y(\lambda) \in \mathsf{poly}(\lambda)$ *be an input length and* $n_z = n_z(\lambda) \in \mathsf{poly}(\lambda)$ *be an output length. We say that* $\mathcal{Y}$ *(as defined in Definition 5.6.1) is an* additive correlation *over a ring* $\mathcal{R}$ *defined by a function family* $\{C_\lambda \colon \mathcal{R}^{n_y} \times \mathcal{R}^{n_y} \to \mathcal{R}^{n_z}\}_{\lambda \in \mathbb{N}}$ *if* $\mathcal{Y}(1^\lambda)$ *outputs pairs of samples* $((y_0, z_0), (y_1, z_1))$*, where* $(y_\sigma, z_\sigma) \in \mathcal{R}^{n_y} \times \mathcal{R}^{n_z}$ *are uniformly random conditioned on* $z_0 + z_1 = C_\lambda(y_0, y_1)$ *We will drop the subscript* λ *when clear from context.*

Remark 31. *Additive correlations are naturally reverse-samplable. To see this, observe that given* (y_σ, z_σ)*, it is possible to efficiently sample* $y_{1-\sigma} \leftarrow \mathcal{R}^{n_y}$*, and set* $z_{1-\sigma} := C(y_0, y_1) - z_\sigma$*.*

We present our PK-PCF construction in Figure 5.11.

Public-Key PCF from MKHSS

Public Parameters. Let $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ be an externally secure MKHSS scheme for polynomial-size RMS programs defined over a ring $\mathcal{R}$. Let $n_k = n_k(\lambda)$ and $n_x = n_x(\lambda)$ be polynomials denoting the PRF key length and output length, respectively. Let $F_k \colon \mathcal{R}^m \to \mathcal{R}^{n_x}$ be a PRF with keys sampled from $\mathcal{R}^{n_k}$, such that $F_k(\mathbf{x})$ is computable by a polynomial-size RMS program over $\mathcal{R}$. Let $\mathcal{Y}$ be an additive correlation over a ring $\mathcal{R}$ defined by a correlation circuit C computable by polynomial-size RMS programs.

The evaluated program. For all vectors $\mathbf{x} \in \mathcal{R}^m$, define the program $P_{\mathbf{x}} \colon \mathcal{R}^{n_k} \times \mathcal{R}^{n_k} \to \mathcal{R}$ to be the polynomial-size RMS program that, on input $k_0, k_1 \in \mathcal{R}^{n_k}$, computes $\mathbf{y}_\sigma \coloneqq F_{k_\sigma}(\mathbf{x}) \in \mathcal{R}^{n_x}$, for all $\sigma \in \{0,1\}$, and outputs $C(\mathbf{y}_0, \mathbf{y}_1)$.

$\mathsf{pkPCF.Setup}(1^\lambda)$:

1 : $\mathsf{crs} \leftarrow \mathsf{MKHSS.Setup}(\lambda)$
2 : **return** crs

$\mathsf{pkPCF.KeyDer}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma^{\mathsf{pcf}}, \mathsf{pk}_{1-\sigma}^{\mathsf{pcf}})$:

1 : **return** $\mathsf{k}_\sigma \coloneqq (\mathsf{sk}_\sigma^{\mathsf{pcf}}, \mathsf{pk}_{1-\sigma}^{\mathsf{pcf}})$

$\mathsf{pkPCF.KeyGen}(\mathsf{crs}, \sigma)$:

1 : $k \leftarrow \mathcal{R}^{n_k}$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MKHSS.KeyGen}(\mathsf{crs})$
3 : $(\llbracket k \rrbracket_0^\sigma, \llbracket k \rrbracket_1^\sigma) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}, k)$
4 : $\mathsf{pk}_\sigma^{\mathsf{pcf}} \coloneqq (\mathsf{pk}, \llbracket k \rrbracket_{1-\sigma}^\sigma)$
5 : $\mathsf{sk}_\sigma^{\mathsf{pcf}} \coloneqq (\mathsf{sk}, \llbracket k \rrbracket_\sigma^\sigma, k)$
6 : **return** $(\mathsf{pk}_\sigma^{\mathsf{pcf}}, \mathsf{sk}_\sigma^{\mathsf{pcf}})$

$\mathsf{pkPCF.Eval}(\mathsf{crs}, \sigma, \mathsf{k}_\sigma, \mathbf{x})$:

1 : **parse** $\mathsf{k}_\sigma \coloneqq ((\mathsf{sk}_\sigma, \llbracket k_\sigma \rrbracket_\sigma^\sigma, k_\sigma), (\mathsf{pk}_{1-\sigma}, \llbracket k_{1-\sigma} \rrbracket_\sigma^{1-\sigma}))$
2 : $\mathbf{y}_\sigma \coloneqq F_{k_\sigma}(\mathbf{x})$
3 : $z_\sigma \coloneqq \mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, \llbracket k_\sigma \rrbracket_\sigma^\sigma, \llbracket k_{1-\sigma} \rrbracket_\sigma^{1-\sigma}, P_{\mathbf{x}})$
4 : **return** $(\mathbf{y}_\sigma, z_\sigma)$

Figure 5.11: Public-key PCF for NC^1 from MKHSS.

5.6.2.1 Security analysis.

We now turn to the security analysis of the PK-PCF from Figure 5.11.

Theorem 5.6.1 (Security of PK-PCF)**.** *Assuming the existence of an externally-secure MKHSS scheme* MKHSS *for polynomial-size RMS programs over a finite ring* $\mathcal{R}$ *and the existence of PRFs in* NC^1*, the construction described in Figure 5.11 is a PK-PCF for arbitrary additive correlations that can be described by polynomial-size RMS programs over* $\mathcal{R}$*.*

Pseudorandomness. Consider the following sequence of hybrid games.

- *Hybrid* $\mathcal{H}_0$*.* This hybrid game consists of the pseudorandomness experiment $\mathsf{E}^{\mathsf{pkpr}}_{\mathcal{A},N,0}$.
- *Hybrid* $\mathcal{H}_1$*.* In this hybrid game, the outputs (z_0, z_1) are computed by first sampling $z_1 \xleftarrow{R} \mathcal{R}$, and then setting $z_0 := z_1 + P_{\mathbf{x}}(k_0, k_1)$.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$ *assuming the external security of* MKHSS*.*

  Proof. An efficient distinguisher immediately contradicts the external security property of MKHSS (cf. Definition 5.4.1). □

- *Hybrid* $\mathcal{H}_2$*.* In this hybrid game, the challenger is given oracle access to $F_{k_\sigma}(\cdot)$, for all $\sigma \in \{0,1\}$. Instead of computing F_{k_σ} using k_σ, the challenger obtains the PRF evaluation by querying the respective oracles. Then, the challenger samples $z_1 \xleftarrow{R} \mathcal{R}$, and set $z_0 := z_1 + C(\mathbf{y}_0, \mathbf{y}_1)$.

  Claim. $\mathcal{H}_2 \approx_s \mathcal{H}_1$*.*

  Proof. By definition of $P_{\mathbf{x}}$ in Figure 5.11, the output distribution is identical to that of $\mathcal{H}_1$. □

- *Hybrid* $\mathcal{H}_3$*.* This hybrid game proceeds as $\mathcal{H}_2$, except that the key k in pkPCF.KeyGen is replaced by 0. That is, pkPCF.KeyGen computes $(\llbracket k \rrbracket_0^\sigma, \llbracket k \rrbracket_1^\sigma) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}, 0)$.

  Claim. $\mathcal{H}_3 \approx_c \mathcal{H}_2$ *assuming the security of* MKHSS*.*

  Proof. The claim follows immediately by the security of MKHSS. □

- *Hybrid* $\mathcal{H}_4$*.* This hybrid game proceeds as $\mathcal{H}_3$, except that the PRF oracles F_{k_σ}, for $\sigma \in \{0,1\}$, are replaced with *random* oracles H_σ. Hence, $\mathbf{y}_\sigma$ is computed as $\mathbf{y}_\sigma := H_\sigma(\mathbf{x})$, for all $\sigma \in \{0,1\}$.

  Claim. $\mathcal{H}_4 \approx_c \mathcal{H}_3$ *assuming the security of the PRF.*

Proof. The claim follows from the standard PRF security property (note that we can apply the PRF security since the shares $(\llbracket k_\sigma \rrbracket_0^\sigma, \llbracket k_\sigma \rrbracket_1^\sigma)$ do not depend on the PRF key k_σ anymore). □

- *Hybrid* $\mathcal{H}_5$. In this hybrid game, the challenger samples $\mathbf{y}_\sigma \xleftarrow{R} \mathcal{R}^{n_y}$, for all $\sigma \in \{0,1\}$.

  Claim. $\mathcal{H}_5 \approx_s \mathcal{H}_3$.

  Proof. Observe that the probability of any two inputs $\mathbf{x}$ to the random oracle H_σ colliding is negligible, hence $\mathcal{H}_5$ is statistically indistinguishable from $\mathcal{H}_4$. □

At this point, it suffices to note that $\mathcal{H}_5$ is equivalent to the experiment $\mathsf{E}^{\mathsf{pkpr}}_{\mathcal{A},N,1}$, concluding the proof.

Security. We now turn to proving the security of our PK-PCF. We proceed as above via a sequence of hybrid games.

- *Hybrid* $\mathcal{H}_0$. This hybrid game consists of the security experiment $\mathsf{E}^{\mathsf{pksec}}_{\mathcal{A},N,0}$.
- *Hybrid* $\mathcal{H}_1$. In this hybrid game, the challenger computes $z_{1-\sigma}$ by first computing

  $$z_\sigma \leftarrow \mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, \llbracket k_\sigma \rrbracket_\sigma^\sigma, \llbracket k_{1-\sigma} \rrbracket_\sigma^{1-\sigma}, C_{\mathbf{x}})$$

  and then setting $z_{1-\sigma} := z_\sigma + (-1)^{1-\sigma} C_{\mathbf{x}}(\mathbf{y}_0, \mathbf{y}_1)$.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$ *assuming the correctness of* MKHSS.

  Proof. The claim follows directly from the correctness property of MKHSS. □

  Hybrid $\mathcal{H}_2$. This hybrid game is identical to $\mathcal{H}_1$ except that pkPCF.KeyGen outputs:

  $$(\llbracket k_{1-\sigma} \rrbracket_0^{1-\sigma}, \llbracket k_{1-\sigma} \rrbracket_1^{1-\sigma}) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, 1-\sigma, \mathsf{pk}, 0).$$

  Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$ *assuming the security of* MKHSS.

  Proof. The claim follows directly from the security property of MKHSS. □

- *Hybrid* $\mathcal{H}_3$. In this hybrid game, the challenges is given oracle access to $F_{k_{1-\sigma}}$. The challenger uses this oracle access to compute $\mathbf{y}_{1-\sigma} := F_{k_{1-\sigma}}(\mathbf{x}) \in \mathcal{R}^{n_x}$.

  Claim. $\mathcal{H}_3 \approx_s \mathcal{H}_2$.

  Proof. By definition of $P_{\mathbf{x}}$, the output distribution is identical to that of $\mathcal{H}_2$. □

- *Hybrid* $\mathcal{H}_4$. This hybrid game is identical to $\mathcal{H}_3$ except that the oracle for $F_{k_{1-\sigma}}$ is now replaced with a random oracle H. Hence, $\mathbf{y}_{1-\sigma}$ is computed as $\mathbf{y}_{1-\sigma} := H(\mathbf{x})$.

  Claim. $\mathcal{H}_4 \approx_c \mathcal{H}_3$ *assuming the security of the PRF.*

  Proof. The claim follows from the security of the PRF. In particular, note that we can apply the PRF security since the shares $(\llbracket k_{1-\sigma} \rrbracket_0^{1-\sigma}, \llbracket k_{1-\sigma} \rrbracket_1^{1-\sigma})$ do not depend on $k_{1-\sigma}$ anymore. □

- *Hybrid* $\mathcal{H}_5$. In this hybrid game, the challenger samples $\mathbf{y}_{1-\sigma} \xleftarrow{R} \mathcal{R}^{n_y}$.

  Claim. $\mathcal{H}_5 \approx_s \mathcal{H}_4$.

  Proof. Observe that the probability of any two inputs $\mathbf{x}$ to H colliding is negligible, hence this hybrid is statistically indistinguishable from $\mathcal{H}_4$. □

At this point, it suffices to note that $\mathsf{E}^{\mathsf{pksec}}_{\mathcal{A},N,1}$ uses the natural reverse-sampling algorithm for additive correlations. This concludes the proof.

Remark 32 (On strong PK-PCFs)**.** *We note that because we use a standard PRF in our construction, our PK-PCF can be shown to be a* strong *PCF. Alternatively, we can substitute the PRF for a* weak *PRF and follow the same proof.*

5.6.3 Multi-party computation with silent preprocessing

Building upon the PK-PCF introduced in Section 5.6.2, we introduce a *multi-party* PK-PCF for generating Beaver triple correlations, and discuss the direct implications to secure computation. We note that we can only support degree-2 correlations (e.g., Beaver triples) in the multi-party setting when using two-party PCFs as a building block. The same limitation applies to prior constructions of multi-party correlation generators from two-party building blocks [BCG+19b].

Defining multi-party PK-PCFs. We start by introducing the notion of multi-party PK-PCF (Definition 5.6.4). Our definition generalizes the notion of PK-PCF to more than two parties in a natural way. Note that for simplicity, we "absorb" the key derivation procedure into $\mathsf{MKHSS.Eval}$. That is, in our formal definition, $\mathsf{MKHSS.Eval}$ directly takes as input the secret key sk_i of a party and the public keys $(\mathsf{pk}_j)_{j\neq i}$ of the other parties. This is without loss of generality, as we can always define KeyDer to output $\mathsf{k}_i := (\mathsf{sk}_i, (\mathsf{pk}_j)_{j\neq i})$. Indeed, we note that this is exactly what our MKHSS-based PK-PCF construction in Figure 5.11 does.

Definition 5.6.4 (Multi-Party Public-Key Pseudorandom Correlation Function)**.** *A multi-party PK-PCF for a p-party correlation* $\mathcal{Y}$ *is defined by a tuple of algorithms* $\mathsf{mpkPCF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval})$*, with the following template:*

- $\mathsf{mpkPCF.Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter and outputs a common reference string (CRS)* crs.

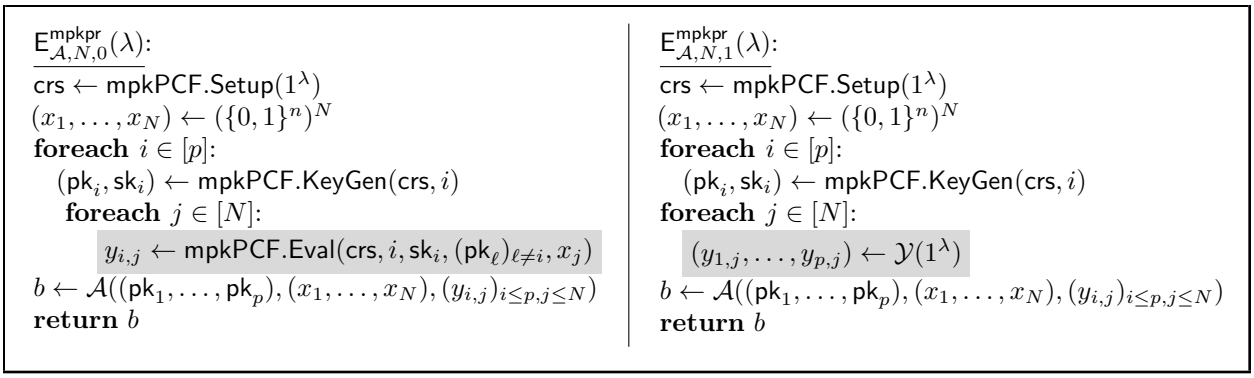


Figure 5.12: Pseudorandomness of a multi-party public-key PCF for a p-party correlation $\mathcal{Y}$.

- $\mathsf{mpkPCF.KeyGen}(\mathsf{crs}, i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$. *The randomized key generation algorithm takes as input the CRS and an index $i \in [p]$, outputs a pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ of public and private* mpkPCF *keys.*
- $\mathsf{mpkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_j)_{j \neq i}, x) \to y_i$. *The deterministic evaluation algorithm takes as input an index i, the secret key sk_i, the public keys $(\mathsf{pk}_j)_{j \neq i}$, and an input $x \in \{0,1\}^n$. It outputs a string y_i.*

A multi-party PK-PCF must satisfy the following pseudorandomness *and* security *properties:*

Correctness / Pseudorandom $\mathcal{Y}$-correlated Outputs. *For all efficient adversaries $\mathcal{A}$, and all $N = N(\lambda) \in \mathsf{poly}(\lambda)$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{mpkpr}}_{\mathcal{A},N}(\lambda) := \left|\Pr[\mathsf{E}^{\mathsf{mpkpr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{E}^{\mathsf{mpkpr}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{E}^{\mathsf{mpkpr}}_{\mathcal{A},N,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 5.12.

Security. *There exists an efficient algorithm* $\mathsf{RSample}\colon (1^\lambda, i^*, (y_i)_{i \neq i^*}) \mapsto y_{i^*}$ *such that for every efficient adversary $\mathcal{A}$, $N = N(\lambda) \in \mathsf{poly}(\lambda)$, and every $i^* \in [p]$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{mpkpr}}_{\mathcal{A},N}(\lambda, i^*) := \left|\Pr[\mathsf{E}^{\mathsf{mpkpr}}_{\mathcal{A},N,0}(\lambda, i^*) = 1] - \Pr[\mathsf{E}^{\mathsf{mpksec}}_{\mathcal{A},N,1}(\lambda, i^*) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{E}^{\mathsf{mpksec}}_{\mathcal{A},N,b}(\lambda, i^*)$, for $b \in \{0,1\}$, is as defined in Figure 5.13.

Construction. We construct a multi-party PK-PCF for the p-party Beaver triple correlation over a ring $\mathcal{R}$. Let $\mathcal{B}$ denote the p-party correlation that, on input λ, samples p uniformly random triples $(a_i, b_i, c_i) \xleftarrow{\text{R}} \mathcal{R}^3$ conditioned on $(\sum_i a_i) \cdot (\sum_i b_i) = \sum_i c_i$. We represent our construction on Figure 5.14.

At a high level, the construction of Figure 5.14 is a direct extension of the PK-PCF construction from Figure 5.11. In particular, the generalization from two parties to p parties is fairly straightforward. In a little more detail, our multi-party PK-PCF is realized as follows:

- Each party P_i generates an MKHSS keys pair $(\mathsf{pk}^{\mathsf{mkhss}}_i, \mathsf{sk}^{\mathsf{mkhss}}_i)$, samples a PRF key k_i, and shares k_i into $(\llbracket k \rrbracket^\sigma_0, \llbracket k \rrbracket^\sigma_1)$ using $\mathsf{MKHSS.Share}$, for each $\sigma \in \{0,1\}$. Then, P_i sets:

$\underline{\mathsf{E}^{\mathsf{mpksec}}_{\mathcal{A},N,0}(\lambda, i^*)}$:
$\mathsf{crs} \leftarrow \mathsf{mpkPCF.Setup}(1^\lambda)$
$(x_1, \ldots, x_N) \leftarrow (\{0,1\}^n)^N$
foreach $i \in [p]$:
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mpkPCF.KeyGen}(\mathsf{crs}, i)$
  foreach $j \in [N]$:
    $y_{i,j} \leftarrow \mathsf{mpkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_\ell)_{\ell \neq i}, x_j)$
$b \leftarrow \mathcal{A}((\mathsf{pk}_1, \ldots, \mathsf{pk}_p), \mathsf{sk}_{i^*}, (x_j)_{j \leq N}, (y_{i,j})_{i \neq i^*, j \leq N})$
return b

$\underline{\mathsf{E}^{\mathsf{mpksec}}_{\mathcal{A},N,1}(\lambda, i^*)}$:
$\mathsf{crs} \leftarrow \mathsf{mpkPCF.Setup}(1^\lambda)$
$(x_1, \ldots, x_N) \leftarrow (\{0,1\}^n)^N$
foreach $i \in [p]$:
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mpkPCF.KeyGen}(\mathsf{crs}, i)$
  foreach $j \in [N]$:
    $y_{i,j} \leftarrow \mathsf{mpkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_\ell)_{\ell \neq i}, x_j)$
  $y_{i^*,j} \leftarrow \mathsf{RSample}(1^\lambda, i^*, (y_{i,j})_{i \neq i^*})$
$b \leftarrow \mathcal{A}((\mathsf{pk}_1, \ldots, \mathsf{pk}_p), \mathsf{sk}_{i^*}, (x_j)_{j \leq N}, (y_{i,j})_{i \neq i^*, j \leq N})$
return b

Figure 5.13: Security of a multi-party public-key PCF for a p-party correlation $\mathcal{Y}$.

$\mathsf{sk}_i := (\mathsf{sk}, [\![k]\!]^0_1, [\![k]\!]^1_1, k)$ and $\mathsf{pk}_i := (\mathsf{pk}, [\![k]\!]^0_0, [\![k]\!]^1_0)$.

- On input $\mathbf{x}$, each party P_i defines $(a_i, b_i) := F_{k_i}(\mathbf{x})$.
- Finally, each pair of parties P_i, P_j, using their MKHSS shares of k_i and k_j, computes additive shares of $F_{k_i}(\mathbf{x}) \cdot F_{k_j}(\mathbf{x})$. Each party P_i aggregates all the shares computed in this way into c_i.

By correctness of the MKHSS scheme, it holds that:

$$\sum_i c_i = \sum_{i,j} F_{k_i}(\mathbf{x}) \cdot F_{k_j}(\mathbf{x}) = \left(\sum_i a_i\right) \cdot \left(\sum_i b_i\right).$$

Theorem 5.6.2 (Security of multi-party PK-PCF)**.** *Assuming the existence of an externally-secure MKHSS scheme* MKHSS *for polynomial-size RMS programs and a PRF in* NC^1*, the construction described in Figure 5.14 is a multi-party PK-PCF for Beaver triple correlations.*

Proof (sketch). The proof is essentially identical to the proof of Theorem 5.6.1. ■

Application to secure computation. A multi-party PK-PCF for the p-party Beaver triple correlation immediately implies a p-party semi-honest secure computation protocol for a general arithmetic circuit C over $\mathcal{R}$ in the *silent preprocessing model* (see Boyle et al. [BCGI18, BCG+19a, BCG+19b, BCG+20a] for discussions on this model):

Preprocessing phase. Each party P_i runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mpkPCF.KeyGen}(i)$ and broadcasts pk_i over a public channel.

Silent expansion. For each multiplication gate in C, each party P_i computes $(a_i, b_i, c_i) := \mathsf{pkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_j)_{j \neq i}, \mathbf{x})$, where $\mathbf{x}$ is a fresh common randomness.

Online phase. The parties run the information-theoretic GMW protocol, consuming one Beaver triple for each multiplication gate computed in the preprocessing phase.

The fact that GMW can be securely instantiated using the correlated pseudorandomness generated by a (multi-party, public key) PCF follows from the fact that the latter suffices to instantiate a *corruptible* functionality for generating correlated randomness, and GMW is

Multi-Party Public-key PCF from MKHSS

Public Parameters. Let $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ be an MKHSS scheme for polynomial-size RMS programs defined over a ring $\mathcal{R}$. Let $F_k : \mathcal{R}^m \to \mathcal{R}^2$ be a PRF with keys sampled from $\mathcal{R}^{n_k}$, such that $F_k(\mathbf{x})$ is computable by a polynomial-size RMS program over $\mathcal{R}$.

The evaluated program. For all vectors $\mathbf{x} \in \mathcal{R}^m$, define $P_{\mathbf{x}} \colon \mathcal{R}^{n_k} \times \mathcal{R}^{n_k} \to \mathcal{R}$ to be the function that, on input $k_0, k_1 \in \mathcal{R}^{n_k}$, computes $(a_\sigma, b_\sigma) := F_{k_\sigma}(\mathbf{x})$, for all $\sigma \in \{0,1\}$, and outputs $a_0 b_1 + a_1 b_0$.

$\mathsf{mpkPCF.Setup}(1^\lambda)$:

1 : $\mathsf{crs} \leftarrow \mathsf{MKHSS.Setup}(\lambda)$
2 : **return** crs

$\mathsf{mpkPCF.KeyGen}(\mathsf{crs}, i)$:

1 : $k \xleftarrow{\text{R}} \mathcal{R}^{n_k}$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MKHSS.KeyGen}(\mathsf{crs})$
3 : **foreach** $\sigma \in \{0,1\}$:
4 : $\quad (\llbracket k \rrbracket_0^\sigma, \llbracket k \rrbracket_1^\sigma) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}, k)$
5 : $\mathsf{pk}_i := (\mathsf{pk}, \llbracket k \rrbracket_0^0, \llbracket k \rrbracket_0^1)$
6 : $\mathsf{sk}_i := (\mathsf{sk}, \llbracket k \rrbracket_1^0, \llbracket k \rrbracket_1^1, k)$
7 : **return** $(\mathsf{pk}_i, \mathsf{sk}_i)$

$\mathsf{mpkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_j)_{j \neq i}, \mathbf{x})$:

1 : $(a_i, b_i) := F_{k_i}(\mathbf{x})$
2 : **parse** $\mathsf{sk}_i = (\mathsf{sk}, \llbracket k_i \rrbracket_1^0, \llbracket k_i \rrbracket_1^1, k_i)$
3 : $c_i := a_i \cdot b_i$
4 : **foreach** $j \in [p] \setminus \{i\}$:
5 : $\quad$ **parse** $\mathsf{pk}_j = (\mathsf{pk}, \llbracket k_j \rrbracket_0^0, \llbracket k_j \rrbracket_0^1)$
6 : $\quad$ **if** $j > i$ **then** $\sigma := 0$ **else** $\sigma := 1$
7 : $\quad c_i := c_i + \mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_i, \mathsf{pk}_j, \llbracket k_i \rrbracket_1^\sigma, \llbracket k_j \rrbracket_0^{1-\sigma}, P_{\mathbf{x}})$
8 : **return** (a_i, b_i, c_i)

Figure 5.14: Multi-party Public-key PCF.

provably secure given ideal access to a corruptible correlated randomness functionality. We refer the reader to Boyle et al. [BCG+19b] for more detailed discussion about this approach. Then, plugging in our construction of (statistically correct) MKHSS from DCR, we get the following corollary:

Corollary 5.6.1. *Assume the DCR assumption holds. For any polynomial number of parties p, for any polynomial-size arithmetic circuit C with n inputs, s multiplication gates, and m outputs over a ring $\mathcal{R}$, there exists a p-party protocol securely computing C in the preprocessing model against an adversary passively corrupting up to $p-1$ parties with the following communication:*

- *In the preprocessing phase, the parties communicate $p \cdot \mathsf{poly}(\lambda)$ bits in a single round of broadcast.*
- *In the online phase, the parties communicate $p \cdot (2s + m)$ elements of $\mathcal{R}$.*

Previously, the best-known multi-party protocols with silent preprocessing (under assumptions not known to imply spooky encryption) were constructed using either HSS (from DCR or DDH over class groups [OSY21, RS21, ADOS22]), programmable 2-party PCGs (from ring-LPN [BCG+20b], or quasi-abelian syndrome decoding [BCCD23]). All these approaches incurred a *quadratic* communication overhead $\tilde{\Omega}(p^2) \cdot \mathsf{poly}(\lambda)$ in the number of parties p, in the preprocessing phase. Our construction is the first to achieve $p \cdot \mathsf{poly}(\lambda)$ communication overhead in the preprocessing phase, which is quasi-optimal.

Part III

Expanding the Frontier

Chapter 6

Simultaneous-Message and Succinct Secure Computation

Summary

In this chapter, we put forth and instantiate a new primitive we call *simultaneous-message and succinct (SMS)* secure computation. An SMS scheme enables a *minimal* communication pattern for secure computation in the following scenario: Alice has a large private input X, Bob has a small private input y, and Charlie wants to learn $f(X, y)$ for a public function f.

Given a common reference string (CRS) setup phase, an SMS scheme for a function f is instantiated with two parties holding inputs X and y, and has the following structure:

- The parties *simultaneously* exchange a single message.
- Communication is *succinct*, scaling sublinearly in the size of X and the output $f(X, y)$.
- Without further interaction, the parties locally derive additive secret shares of $f(X, y)$.

Indeed, Alice and Bob simultaneously send each other a message using the CRS and their private inputs. Using the transcript and their private state, the parties locally derive additive secret shares of $f(X, y)$, which they can send to Charlie. As such, an SMS scheme incurs a communication cost to Charlie that is only twice that of the function output length. Importantly, the size of Alice's message does not grow with the size of her input X, and both Alice's and Bob's first-round messages grow sublinearly in the size of the output. Additionally, Alice's or Bob's view provides no information about the other party's input besides the output of $f(X, y)$, even if colluding with Charlie. We show that SMS schemes give:

(1) A direct construction of trapdoor hash functions (TDH) for the same class of functions as the one supported by the SMS scheme.

(2) A simple and generic compiler for obtaining compact, rate-1 fully homomorphic encryption (FHE) from any non-compact FHE scheme.

(3) A simple and generic compiler for obtaining correlation-intractable (CI) hash functions that are secure against all efficiently-searchable relations.

In turn, under the learning with errors (LWE) assumption, we obtain the first construction of TDH for all functions and generic approaches for obtaining rate-1 FHE and CI hashing.

6.1 Introduction

Consider the following scenario: Alice has a *large* private input X, Bob has a *small* private input y, and Charlie wants to learn the output $f(X, y)$ of some public function f evaluated over the inputs of Alice and Bob. To achieve this with optimal communication cost, Bob can simply send his input to Alice, who then computes the output $f(X, y)$, and sends it to Charlie. This simple, but clearly insecure, protocol achieves the following communication complexity:

- The communication between Alice and Bob is only $|y|$, and in particular, independent of the length of Alice's input X and the output of the function $f(X, y)$.
- The total communication to Charlie is simply the length of the function output (and, in particular, independent of Alice and Bob's input lengths).

Furthermore, this protocol requires only a single message from Bob to Alice, and then from Alice to Charlie. In this chapter, we ask the following.

Can we design a secure computation protocol that preserves, to the extent possible, the communication complexity and the communication pattern of the above insecure protocol?

Specifically, we will consider secure computation protocols [Yao86, GMW87] with security against semi-honest adversaries who may corrupt either of the two input parties (Alice or Bob) together with the output party (Charlie).

Simultaneous-Message and Succinct Secure Computation. To answer this question, we investigate a minimal model of computation that we refer to as *simultaneous-message and succinct* (SMS) secure computation. In an SMS scheme, following a setup phase that outputs a common reference string (CRS),[1] the interaction proceeds as follows:

- **Encode:** Alice and Bob encode their private inputs into public encoding pe_A and pe_B, respectively, and exchange these encodings in a simultaneous round of communication.
- **Decode:** Given her private state and public encodings, Alice (resp., Bob) computes a share z_A (resp., z_B) and sends it to Charlie, who can locally reconstruct the output.

Communication between parties in both encode and decode phases is simultaneous. Moreover, for any big input X, small input y, and a function f, we require that the decoded values z_A and z_B computed by Alice and Bob form an *additive sharing* of $f(X, y)$, which allows Charlie to reconstruct $f(X, y) = z_A \oplus z_B$. Note that because additive shares are information-theoretically the same size as the output, the communication cost of the decode phase is only twice that of the insecure protocol. We further require the following *succinctness* property for the encode phase: the size of the public encodings pe_A and pe_B is succinct with respect to Alice's input length $|X|$ and the function output length.[2] Finally, we require standard simulation-based security against semi-honest adversaries.

In summary, compared to the "optimal" insecure protocol, SMS requires two additional messages: one from Alice to Bob, and another message from Bob to Charlie. In the secure

[1] In our constructions, we only need a common random string.

[2] Note that by communication complexity lower bounds, we cannot expect to achieve communication sublinear in the input lengths of both parties.

setting, these additional messages are necessary to prevent input-resetting attacks [HLP11]. However, we show that Bob's message can, in some cases, be as short as the security parameter λ. In this sense, the communication model of SMS is *minimal*.

We investigate the feasibility of constructing SMS schemes and present positive results as well as several applications. Before we proceed to describe our results, we first compare SMS with some related notions in cryptography.

Comparison with succinct protocols. SMS can be viewed as extending the notion of private simultaneous messages [FKN94] along two dimensions. First, SMS allows *collusion* between an input party and the output party (and hence, requires an additional round of communication). Second, SMS requires succinct communication in the input size and function description.

The study of "circuit-succinct" secure communication has a rich history in cryptography. Arguably, it was most popularized by fully homomorphic encryption [Gen09], which yields secure communication with communication independent of the size of the circuit representation of the function. If we relax the *simultaneous-message* requirement, and allow one party to send its message after the other, we can also obtain "input-succinctness" in the size of one of the party's inputs by using FHE or its "dual" notion of laconic function evaluation [CDG+17, QWW18]. However, neither of these two approaches yield SMS. For example, using (circuit-private) FHE, one can attempt to design an SMS protocol as follows: (1) Bob sends an encryption of his input y to Alice, (2) Alice homomorphically computes the encryption of $f(X, y)$ and sends it to Charlie, and (3) Bob sends his secret key to Charlie. Clearly, this protocol is insecure against collusion with Alice, since Charlie gets the secret key. Furthermore, the output phase does not admit additive reconstruction and requires communicating more than just the function output.

SMS bears resemblance to the notion of homomorphic secret sharing (HSS) (e.g., [BGI16]), most notably in the requirement of additive reconstruction and succinctness in the circuit size. However, SMS and HSS are incomparable. For one, HSS (and the more powerful notion of spooky encryption [DHRW16]) does not require succinctness in the input length. Moreover, HSS and spooky encryption support an *adaptive* choice of functions: the function to be computed can be determined *after* the input encoding phase. SMS, in contrast, does not necessarily require this property (although, as we will discuss later, one of our constructions achieves it).

Because of the input-succinctness requirement, SMS is closer to the recently-proposed notion of *succinct* HSS [ARS24], which extends HSS to require succinctness with respect to one of the inputs (in addition to succinctness with respect to the function description). However, succinct HSS still requires a correlated randomness setup (thus, it cannot support the minimal communication pattern of SMS) and current schemes are only suitable for very restricted function classes.

Comparison with two-round secure computation. We note that SMS is stronger than two-round secure computation [Yao86, BL18, GS18] because of the input and circuit succinctness properties in addition to the additive reconstruction properties. Indeed, because of the additive reconstruction requirement alone, SMS—even for a single AND computation—implies non-interactive key exchange via the reduction of Boyle et al. [BGI+18], and therefore is black-box separable from oblivious transfer [GKM+00].

Applications. SMS implies the previously studied notion of trapdoor hashing (TDH) Döttling et al. [DGI+19]. A trapdoor hash scheme is a protocol between two parties—a sender and a receiver—in the common reference string (CRS) model. Given the CRS (referred to as the hash key in the original work), the sender can compute a digest d of its input X, while the receiver can encode a private function f into an evaluation key ek. Given these encoded values, the sender and the receiver can compute an additive secret sharing of $f(X, y)$. A TDH scheme requires two properties: (1) sender succinctness, namely, the size of the digest d must be sublinear in the sender's input length, and (2) receiver privacy, namely, the evaluation key must hide the function f.

It is easy to see that SMS implies TDH by assigning the role of the sender to Alice and the role of the receiver to Bob. In fact, SMS is *stronger* than TDH since the function f is "decoupled" from Bob's input, and hence, we can require both Alice and Bob's messages to be of size sublinear in Alice's input length. Döttling et al. [DGI+19] (and subsequent works [BKM20, GHO20]) constructed TDH schemes for linear functions from a variety of standard assumptions, and constructing TDH for larger function classes has remained open. As we will discuss shortly, our positive results on SMS (combined with the above implication) break this barrier by realizing the first TDH for all depth-d circuits from the learning with errors assumption [Reg05]. By additionally assuming and the circular security of LWE, we get the first TDH for all polynomial-size circuits.

We also demonstrate direct applications of SMS to other powerful primitives, including rate-1 fully homomorphic encryption [BDGM19, GH19], correlation-intractable hash functions, and output-succinct secure computation.

6.1.1 Our results

We initiate the study of SMS protocols and present several positive results and applications.

SMS from LWE. Our first result is an SMS scheme for all depth-bounded computations, assuming the hardness of learning with errors (LWE). Specifically, for all depth-d circuits, we construct an SMS scheme where the communication complexity of the encode phase grows with d, but is otherwise sublinear in the input and output length of the function being computed. This first result is captured in the following theorem:

Theorem 6.1.1 (Informal)**.** *Let $\mathcal{F}$ be the family of all functions that are computable by depth-d circuits. Assuming the hardness of learning with errors (with a superpolynomial modulus-to-noise ratio), there exists an SMS scheme for any function $f \in \mathcal{F}$, where in the encode phase, the size of Alice's message is $|f(X, y)|^{\epsilon} \cdot \mathsf{poly}(\lambda, d)$ and the size of Bob's message is $(|y| + |f(X, y)|^{\epsilon}) \cdot \mathsf{poly}(\lambda, d)$. Here, λ is the security parameter and $\epsilon = (2/3)$. By additionally assuming the circular-security of LWE, we obtain an SMS scheme where the message size is independent of the circuit size.*

SMS from indistinguishability obfuscation. Our second result is an SMS scheme for all polynomial-size *batch* computations, assuming the existence of sub-exponentially-secure indistinguishability obfuscation [BGI+01, GGH+13, JLS21], sub-exponentially secure one-way

functions and somewhere statistically-binding (SSB) hash functions [HW15].[3] In the batch setting, Alice holds as input a long vector $X := (x_1, \ldots, x_L)$ and Bob holds an input y; and they wish to compute $f(x_1, y), \ldots, f(x_L, y)$ given a public function f in some function family $\mathcal{F}$. Here, we relax the succinctness requirement for the encoding phase: The total communication must be at most polylogarithmically dependent on the batch size L, but can grow with the size of the circuit description.

Theorem 6.1.2 (Informal)**.** *Let $\mathcal{F}$ be the family of all functions that are computable by polynomial-size circuits. Assuming the existence of (1) sub-exponentially secure indistinguishability obfuscation, (2) sub-exponentially secure one-way functions, (3) somewhere statistically binding hash functions (with perfect binding), and (4) the existence of injective one-way functions, there exists an SMS scheme that supports batch computation of functions in $\mathcal{F}$. For any batch size L, the size of both Alice' and Bob's messages in the encode phase is $\mathsf{poly}(\lambda, |f|, \log L)$, where λ is the security parameter.*

The above protocol supports *adaptive* choice of functions during the decode phase. That is, the parties can compute their public encodings in the encode phase *independently* of the function. Then, the public encodings can be *reused* for computing the decode phase for any choice of batch functions.

We next discuss applications of the above results.

Application I: Trapdoor hashing beyond linear functions. Assuming the hardness of LWE, we obtain a construction of trapdoor hash functions for all depth-d circuits. By additionally assuming the circular-security of LWE, we obtain a construction of trapdoor hash functions for all polynomial-size circuits. This significantly improves upon the state of the art, where trapdoor hash functions were only known for linear functions. This result follows immediately from Theorem 6.1.1 combined with the aforementioned direct implication from SMS to trapdoor hashing, however, we provide a formal construction in Section 6.8 for completeness.

Application II: Rate-1 fully homomorphic encryption, generically. In a rate-1 fully homomorphic encryption scheme, the message to ciphertext length ratio (i.e., rate) is $1 - o(1)$. A rate-1 FHE scheme was first constructed under the LWE assumption by Brakerski, Döttling, Garg, and Malavolta [BDGM19] and Gentry and Halevi [GH19]. At a high level, their constructions intricately combine FHE schemes with rate-1 linearly homomorphic encryption to compress ciphertexts.

We show that an SMS scheme can be used to transform *any* FHE scheme into a rate-1 FHE scheme. Our construction is quite simple and generic. We briefly sketch the transformation below and provide more details in Section 6.9.

Consider any FHE scheme with a poor rate and let the decryption algorithm of this FHE scheme be described by a function f that takes as input a ciphertext ct and a secret key sk, and outputs the message. To compress the ciphertexts of this scheme, we use an SMS scheme that performs the computation of f. In more detail:

[3] We additionally require the existence of injective one-way functions and perfect binding for the SSB hash.

- *New keys.* Let $(\mathsf{pk}, \mathsf{sk})$ be a public key and secret key pair of the underlying FHE scheme. The public key of the new FHE scheme consists of the tuple $(\mathsf{pk}, \mathsf{pe}_B)$ where pe_B is Bob's public encoding output by the SMS scheme and computed using the secret key sk as his private input. The new secret key is simply Bob's private state st_B output by the SMS scheme.

- *Ciphertext compression.* Let ct be a homomorphically-evaluated ciphertext ct of the underlying FHE scheme with poor rate. To compress this ciphertext, we first use ct as the input to the SMS scheme to compute Alice's public encoding pe_A in the encode phase. Then, we use Bob's public encoding pe_B (which is now part of the new public key) to compute Alice's decoded value z_A. The compressed ciphertext is simply the tuple (pe_A, z_A).

- *Decryption.* To decrypt the ciphertext (pe_A, z_A), we first use pe_A and Bob's private state st_B (which is part of the new secret key) to compute Bob's decoded value z_B. The plaintexts are recovered as $z_A \oplus z_B$.

Arguing correctness and security. The correctness of the scheme follows by inspection. Intuitively, we use SMS to decrypt the ciphertexts, which results in additive shares of the messages. Because the public encodings are sublinear in the size of the ciphertext, the rate asymptotically approaches 1. The security of the scheme follows from the security of the underlying FHE scheme as well as security for Bob in the SMS scheme (we do not require security for Alice here). The full transformation and proof of security are provided in Section 6.9.

Application III: Correlation-intractable hash functions, generically. Correlation-intractable (CI) hash functions [CGH98] are functions whose input-output pairs behave in a similar way to a random function in that they do not satisfy any "bad" correlations. Specifically, a hash function family H_{hk} is said to be correlation intractable for a relation class $\mathcal{R}$, if for any relation $R \in \mathcal{R}$, no efficient adversary given the hash key hk can find an input-output pair that satisfies R.

Recently, CI hashing has found numerous applications in cryptography, most notably in achieving new constructions of non-interactive zero knowledge proofs (see, e.g., [CCH+19, PS19, JJ21]) and succinct non-interactive arguments (see, e.g., [CHK+19, CJJ21, JKKZ21, CJJ22]) in the standard model. These applications are obtained by using CI hashing to securely instantiate the Fiat–Shamir paradigm [FS87] for round-collapsing interactive proofs.

In Section 6.10, we show a simple and generic construction of CI hashing for efficiently-searchable relations from SMS. Using Theorem 6.1.1, we obtain CI hashing for relations searchable by depth-bounded circuits from LWE. Previously, CI hashing from LWE was known using the work of Peikert and Shiehian [PS19]. However, our construction of CI hashing from SMS is generic, and uses the observation of Brakerski, Koppula, and Mour [BKM20] that CI hashing can be constructed from trapdoor hashing. See Section 6.10 for details on our transformation.

	Assumption	Input Succinctness	Output Succinctness	Function Class
[DHRW16]	LWE / $i\mathcal{O}$+DDH	✗	✓	All Circuits
[OSY21]	DCR / LWE	✗	✗	Degree-2
[ARS24, BCM+24]	DCR / QR / LWE	✓	✓	Degree-2
Theorem 6.5.2	LWE	✓	✓	Depth-d Circuits
Theorem 6.6.3	$i\mathcal{O}$+SSB	✓	✓	All Batch Circuits
Theorem 6.7.1	Circular LWE	✓	✓	All Circuits

Table 6.1: Constructions of SMS.

6.1.2 Related work

In this section, we discuss some connections between SMS and other related notions in cryptography.

Output-succinct secure computation. The work of Hubáček and Wichs [HW15] investigated the feasibility of secure computation with *total* communication complexity that is sublinear in the function output length. Using indistinguishability obfuscation ($i\mathcal{O}$) [BGI+01, GGH+13] and other standard assumptions, they constructed interactive protocols with sublinear communication using a large common random string (or large random tapes for the parties) of length proportional to the output length. In fact, they demonstrated that the use of program obfuscation (with a large CRS) is necessary for this task. This implication does *not* hold for SMS since the communication between the input parties and the output party (Charlie) does grow with the output length. In particular, SMS only requires the encodings (first round messages) to be succinct in the input and output size. Nonetheless, in Section 6.7, we show that our $i\mathcal{O}$-based construction of SMS can be extended to have a succinct second round message from Bob to Charlie. This extension gives an alternative construction of the secure computation protocols considered by Hubáček and Wichs.

Succinct homomorphic secret sharing. The recent work of Abram, Roy, and Scholl [ARS24] constructs *succinct* homomorphic secret sharing (HSS). Similarly to the PSM model, HSS allows a correlated setup to take place between between Alice and Bob. However, the additional succinctness requirement considered in [ARS24], in conjunction with the collusion guarantees of HSS, make the notion more closely related to SMS. In the construction of succinct HSS, Alice with a large vector $\mathbf{a}$ and a short input x, and Bob with short input y, can compute a function of the form $\langle \mathbf{a}, f(x, y)\rangle$ with communication that is sublinear in $|\mathbf{a}|$. Concretely, their constructions result in $\sqrt{|\mathbf{a}|} + |x| + |y|$ communication and require three rounds of interaction when including the correlated setup between the parties.

Indeed, the core primitive used to realize succinct HSS, that turns out to also be instrumental in realizing SMS (cf. Section 6.5.2), is the recently introduced notion of succinct non-interactive VOLE (NIVOLE) [ARS24, BCM+24]. We show that succinct NIVOLE can be cast as an SMS scheme for degree-2 functions, given that it does not require a correlated setup between parties [BCM+24].

We give a comparison of our results with related notions in Table 6.1.

6.1.3 Chapter organization

We begin with a technical overview of our constructions in Section 6.2. Section 6.3 introduces the necessary preliminaries and notation. In Section 6.4, we formally define SMS and explore its relationships with other primitives. Section 6.5 presents our LWE-based construction for depth-d circuits, which we extend to all circuits in Section 6.7. We then detail our $i\mathcal{O}$-based construction for batch function evaluations in Section 6.6. Section 6.7 covers various optimizations and extensions, including an extension to our $i\mathcal{O}$-based construction of SMS that compresses the second-round message to achieve output-succinct secure computation. Finally, we demonstrate applications: a compiler to rate-1 FHE in Section 6.9 and a compiler to correlation-intractable hashing in Section 6.10.

6.2 Technical Overview

In this section, we give an overview of our constructions of SMS schemes from LWE (Section 6.2.1) and from indistinguishability obfuscation (Section 6.2.2).

6.2.1 Construction from LWE

The high-level idea is to start with the ABE scheme of Boneh et al. [BGG+14], which has two useful algorithms EvalPK and EvalCT that can be used in a "black-box" way [GVW15,QWW18]. The CRS consists of α matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\alpha$. Let C be a depth-d arithmetic circuit producing m-bit outputs and let C_i be the circuit that outputs the i-th bit of C. The two algorithms have the following syntax:

- $\mathsf{EvalPK}(\mathsf{crs}, C) \to \mathbf{A}_C$. Takes as input the CRS and a circuit description C producing m bit outputs; it outputs a list $\mathbf{A}_C := (\mathbf{A}_{C_i})_{i=1}^m$ of size $m \cdot \mathsf{poly}(\lambda, d)$.
- $\mathsf{EvalCT}(\mathsf{crs}, \mathsf{ct}, C, x) \to \mathbf{w}_C$. Takes as input a ciphertext vector $\mathsf{ct} := (\mathbf{s}^\top(\mathbf{A}_i + x_i \cdot \mathbf{G}) + \mathbf{e}_i^\top)_{i\in[\alpha]}$ encrypting each bit of the input x to the circuit C; it outputs $\mathbf{w}_C := (\mathbf{s}^\top(\mathbf{A}_{C_i} + C_i(x) \cdot \mathbf{G}) + \tilde{\mathbf{e}}_i^\top)_{i\in[m]}$, encrypting each bit of the output $C(x)$.

Here, $\mathbf{G}$ is the standard gadget matrix (see Definition 6.5.1 for a formal definition).

Inspired by the ideas that underpin the predicate encryption scheme of Gorbunov, Vaikuntanathan, and Wee [GVW15] and laconic function evaluation of Quach, Wee, and Wichs [QWW18], our idea is to use EvalPK to commit Alice to her large input X by hard-coding it into a large circuit C. More concretely, Alice defines C to be the circuit that only takes as input an FHE-encrypted input of y and outputs the (encrypted) FHE evaluation of $f(X, y)$.[4] Alice sends Bob $\mathbf{A}_C$—the output of EvalPK when evaluated on her large circuit C. Bob, on the other hand, encrypts his input y under a suitable FHE scheme and sends it to Alice. Surprisingly, we will show that with just a few tweaks, these values are sufficient to construct an SMS scheme.

We now proceed to explain our initial attempt to realize SMS. While this first approach does not work out-of-the-box, it gives us the right insights and framework to build off of.

[4] We need to assume that the FHE evaluation of a depth-d circuit can itself be represented by a depth-d' circuit, for $d, d' \in \mathsf{poly}(\lambda)$, which is the case for existing LWE-based FHE schemes.

Setup. Let α denote the length (in bits) of an FHE ciphertext encrypting an ℓ-bit input. For now, we define the common *random* string crs to simply consist of α random matrices $(\mathbf{A}_1, \ldots, \mathbf{A}_\alpha)$; later, we will add more matrices to crs, while still keeping it uniformly random.

Step 1: Generating the encodings. Alice computes $\mathsf{EvalPK}(\mathsf{crs}, C)$ to obtain $\mathbf{A}_C$, where C is as defined above. This $\mathbf{A}_C$ implicitly commits Alice to X. Bob samples a secret vector $\mathbf{s}$ such that the first coordinate of $\mathbf{s}$ is 1, and generates a secret key sk for a leveled fully homomorphic encryption scheme (cf. Definition 6.3.2). He encrypts his input y under the FHE scheme using sk to obtain the bits of the ciphertext $\mathsf{ct}_{\mathsf{FHE}} := (\mathsf{ct}^{(1)}_{\mathsf{FHE}} \| \ldots \| \mathsf{ct}^{(\alpha)}_{\mathsf{FHE}}) \in \{0,1\}^\alpha$. Then, he generates

$$\mathsf{ct} := \left(\mathbf{s}^\top (\mathbf{A}_i + \mathsf{ct}^{(i)}_{\mathsf{FHE}} \cdot \mathbf{G}) + \mathbf{e}_i^\top\right)_{i \in [\alpha]},$$

where each $\mathbf{e}_i$ is sampled from a B-bounded error distribution. Note that Bob's encoding is independent of Alice's input length L.

Step 2: Simultaneous communication. Alice sends $\mathbf{A}_C$ to Bob, which is of size $m \cdot \mathsf{poly}(\lambda, d)$, and Bob sends ct to Alice, which is of size $\ell \cdot \mathsf{poly}(\lambda, d)$. Here, d denotes the circuit depth and is implicit in the LWE parameters.

Step 3: Local decoding. With the encodings from Step 1, Alice uses EvalCT to compute:

$$\begin{aligned}\mathbf{w}_C &:= \left(\mathbf{s}^\top (\mathbf{A}_{C_i} + C_i(\mathsf{ct}_{\mathsf{FHE}}) \cdot \mathbf{G}) + \tilde{\mathbf{e}}_i^\top\right)_{i \in [m]} \\ &= \left(\mathbf{s}^\top (\mathbf{A}_{C_i} + \mathsf{FHE.Enc}(\mathsf{sk}, f(X,y))[i] \cdot \mathbf{G}) + \tilde{\mathbf{e}}_i^\top\right)_{i \in [m]}.\end{aligned}$$

Then, because we set $\mathbf{s}[1] = 1$, the first coordinate of the i-th entry of $\mathbf{w}_C$ is:

$$(\mathbf{s}^\top \mathbf{A}_{C_i})[1] + C_i(y) + \tilde{\mathbf{e}}_i^\top[1] = (\mathbf{s}^\top \mathbf{A}_{C_i})[1] + \mathsf{FHE.Enc}(\mathsf{sk}, f(X,y))[i] + \tilde{\mathbf{e}}_i^\top[1]. \tag{6.1}$$

To see equality, it is helpful to note that the first column of the "gadget" matrix $\mathbf{G}$ (see Definition 6.5.1) is the vector $(1, 0, \ldots, 0)$. Moreover, by using a suitable FHE scheme, we can guarantee that FHE evaluation of a depth-d circuit can itself be evaluated by a bounded-depth circuit C.

We can view Equation 6.1 as a "noisy" share z_A of $\mathsf{FHE.Enc}(\mathsf{sk}, f(X,y))[i]$, since Bob can compute his corresponding "noisy" share as $z_B := (\mathbf{s}^\top \mathbf{A}_{C_i})[1]$ using his key $\mathbf{s}$ and the $\mathbf{A}_{C_i}$ he receives from Alice. With this, Alice and Bob now have "noisy shares" of the i-th bit of the ciphertext $\mathsf{FHE.Enc}(\mathsf{sk}, f(X,y))$, since

$$\begin{aligned}z_A - z_B &\approx \underbrace{(\mathbf{s}^\top \mathbf{A}_{C_i})[1] + \mathsf{FHE.Enc}(\mathsf{sk}, f(X,y))[i]}_{\text{Alice's share from Equation 6.1}} - \underbrace{(\mathbf{s}^\top \mathbf{A}_{C_i})[1]}_{\text{Bob's share}} \\ &\approx \mathsf{FHE.Enc}(\mathsf{sk}, f(X,y))[i].\end{aligned}$$

Unfortunately, it is easy to observe that this does not get us any closer to obtaining an SMS scheme, since the two parties could just as easily have obtained a "trivial" secret sharing of $\mathsf{FHE.Enc}(\mathsf{sk}, f(X,y))[i]$ by having Alice compute the FHE evaluation directly (and setting z_A to the result) and Bob setting z_B to zero. Jumping ahead, we will resolve this by "non-interactively" decrypting the evaluated FHE ciphertext using extensions to the EvalPK

and EvalCT algorithms. However, we first point out some additional problems we need to resolve.

We briefly highlight the issues associated with the above attempt:

(1) It does not result in the parties having shares of $f(X, y)$, rather they hold shares of the *encryption*, which is trivial to achieve using just FHE.

(2) The parties have *noisy* shares, which does not correspond to the additive-reconstruction we desire.

(4) It fails to provide privacy for Alice, since the algorithm EvalPK, as defined in [BGG+14], makes no guarantees about the privacy of the circuit C given $\mathbf{A}_C$ (recall that C contains Alice's input).

(3) While it does give input succinctness for Alice's input, it does not have output succinctness, since $|\mathbf{A}_C| = m \cdot \mathsf{poly}(\lambda, d)$ grows linearly with the size of the output m.

We now explain how we resolve these obstacles in our construction.

Obliviously decrypting the FHE evaluation. The first problem to address is that Alice and Bob obtain shares of the *encryption* of the result rather than a share of $f(X, y)$. Therefore, our first priority is letting Alice somehow "decrypt" the FHE ciphertext privately under Bob's secret key sk. Fortunately, this very problem was also faced in the predicate encryption scheme of Gorbunov et al. [GVW15]. In particular, they show that $(\mathsf{EvalPK}, \mathsf{EvalCT})$ can be "augmented" to compute circuits of the form $\mathsf{IP} \circ C$, where IP is the class of inner products. Moreover, while the input to C needs to be public (known to Alice), the inner product can be *private* (only known to Bob). Therefore, we can assume an FHE scheme with a near-linear decryption property and let Bob input the FHE secret decryption key sk as the private "inner product" input. With this modification, and by slightly abusing notation by using C_i to denote $\mathsf{IP} \circ C'_i$ (where C'_i is defined to output a vector in the ring $\mathbb{Z}_q$ corresponding to the FHE ciphertext encrypting the i-th bit of the circuit C), Alice and Bob obtain:

$$\underbrace{(\mathbf{s}^\top \mathbf{A}_{C_i})[1] + \overbrace{\langle \mathsf{FHE.Enc}(\mathsf{sk}, f_i(X, y)), \mathsf{sk} \rangle}^{(q/2)\cdot f_i(X,y)+\mathsf{noise}}}_{\text{Alice's share } z_A^{(i)}} - \underbrace{(\mathbf{s}^\top \mathbf{A}_{C_i})[1]}_{\text{Bob's share } z_B^{(i)}} \approx f_i(X, y),$$

where f_i computes the i-th bit of the function f. Note that this gives us a "noisy" secret sharing of the correct result! Indeed, the output of the circuit C consists of $m \cdot \beta$ ring elements, corresponding to the bit-wise encryptions of $f(X, y)$. Then, the inner product with the secret key $\mathsf{sk} \in \mathbb{Z}_q^\beta$ results in m noisy shares (over $\mathbb{Z}_q$) at the end, where the noise comes from the near-linear decryption.

Rounding of noisy shares. To convert these noisy shares to additive shares we can apply the "rounding lemma" [DHRW16, BKS19] to locally round-away the error:

Lemma 6.2.1 (Rounding of noisy secret shares [DHRW16, BKS19])**.** *Let (p, q) be two integers such that p divides q. Fix any $z \in \mathbb{Z}_q$ and let (z_0, z_1) be any two random elements of $\mathbb{Z}_q$ subject to $z_0 + z_1 = (q/p) \cdot z + e \bmod q$, where e is such that $q/(p \cdot |e|) \geq \lambda^{\omega(1)}$. Then, with probability at least $1 - (|e| + 1) \cdot p/q \geq 1 - \lambda^{-\omega(1)}$, it holds that $\lfloor z_0 \rceil_p + \lfloor z_1 \rceil_p = z \bmod p$, and the probability is over the random choice of $(z_0, z_1) \in \mathbb{Z}_q \times \mathbb{Z}_q$.*

Using the rounding lemma, and setting the LWE parameters of the FHE scheme to have a superpolynomial modulus-to-noise ratio, Alice and Bob locally derive shares $z_A^{(i)}$ and $z_B^{(i)}$, for all $i \in [m]$, such that $z_A^{(i)} - z_B^{(i)} = f_i(X, y)$, as desired. More concretely, we have:

$$\underbrace{\left\lfloor (\mathbf{s}^\top \mathbf{A}_{C_i})[1] + (q/2) \cdot f_i(X, y) + \mathsf{noise} \right\rceil_2}_{\text{Alice's share } z_A^{(i)}} - \underbrace{\left\lfloor (\mathbf{s}^\top \mathbf{A}_{C_i})[1] \right\rceil_2}_{\text{Bob's share } z_B^{(i)}} = f_i(X, y),$$

where noise corresponds to the sum of the FHE decryption and EvalCT errors. (In our actual construction, we additionally add pseudorandom "shares of zero" to ensure the distribution is near-uniform before applying the rounding lemma.)

Statistical hiding for Alice. The final problem we need to take care of is ensuring that $\mathbf{A}_C$ leaks no information about C (which contains Alice's input X). Fortunately, a related problem was faced by Quach et al. [QWW18] when constructing laconic function evaluation. They show an efficient transformation that can be applied to EvalPK (and EvalCT) such that $\mathbf{A}_C$ statistically hides C, which also hides Alice's input X in our construction.

Adding output succinctness. At this point we have a protocol that is input succinct: Bob's encoding is independent of f and only $\mathsf{poly}(\lambda, \ell)$ bits in size (which is independent of Alice's input size L). However, the encoding still grows with the output length $|f(X, y)|$, since $\mathbf{A}_C$ has to be at least as large as the output length m. To get output succinctness, we observe that we can cast Bob's computation as a vector oblivious linear evaluation (VOLE) [ADI+17]: Alice has input $\mathbf{A}_{C_i} \in \mathbb{Z}_q^{n \times k}$ and Bob has input $\mathbf{s} \in \mathbb{Z}_q^n$, and Bob needs to learn the linear evaluation $\mathbf{s}^\top \mathbf{A}_{C_i}$. It is therefore enough for Alice and Bob to compute *shares* of $\mathbf{s}^\top \mathbf{A}_{C_i}$ (in particular, Bob never needs to have $\mathbf{A}_{C_i}$ "in the clear"). To accomplish this, we "bootstrap" using using SMS for VOLE. That is, using an SMS scheme for a "batched" variant of VOLE, Alice encodes each matrix $\mathbf{A}_{C_i} \in \mathbb{Z}_q^{n \times k}$ (Alice has m such matrices) and Bob encodes his secret $\mathbf{s}$. Using the public encodings, the two parties then locally (without further communication) obtain additive shares of $\mathbf{s}^\top \mathbf{A}_{C_i}$ as output, for all $i \in [m]$. Moreover, using existing constructions of succinct, non-interactive VOLE [ARS24, BCM+24], which we show can be cast as an SMS scheme, the total communication of this protocol is $O(m^{2/3}) \cdot \mathsf{poly}(\lambda, d)$, giving us sublinearity in the output size.

6.2.2 Construction from indistinguishability obfuscation

We now overview our construction of SMS from $i\mathcal{O}$, where we realize SMS for polynomially-sized *batch* functions. In this setting, Alice has a vector of inputs $X = (x_1, \ldots, x_L)$ and Bob has an ℓ-bit input y. At a high level, our protocol is reminiscent of the *insecure* protocol sketched in Section 6.1, where Bob simply sends his input y to Alice. However, to provide privacy for Bob, we use $i\mathcal{O}$ to hide y inside an obfuscated program.

In more detail, the main idea is to have Bob send an obfuscated program for a universal circuit, with his short input y hardcoded in it. The program evaluates the function $f(x_i, y) \oplus R_i$, where f and x_i are given as input and R_i is a pseudorandom mask that depends on f and x_i. To allow Bob to compute R_i on his end, which becomes his share of $f(x_i, y)$, Alice commits to her inputs X in such a way that she can locally decommit to any input x_i later on. Then, R is computed as the output of a PRF on the commitment to the batch and the index i of

the input x_i in the batch X. This allows Bob to locally derive his share of $f(x_i, y)$ without knowledge of Alice's inputs.

Concretely, we use SSB hashing [HW15], the standard tool to use in conjunction with $i\mathcal{O}$. The general template of $\mathsf{SSB} + i\mathcal{O}$ [HW15, OPWW15] is to hardcode the hash key hk and commitment c_X (informally, we will refer to the hash as a "commitment") into the obfuscated program. Then, when Alice runs the program, she can provide a local opening to x_i. Because the commitment is *statistically binding* at some index i, it becomes easy to prove security via a hybrid argument that switches out where the SSB hash is statistically vs. computationally binding. At a high level, this makes two adjacent hybrid programs functionally equivalent, which then enables invoking $i\mathcal{O}$ security to prove computational indistinguishability between the hybrids.

Unfortunately, we cannot directly apply this template to realize SMS. The problem is that Bob does not have Alice's commitment (hash) when he generates the obfuscated program at encoding time! At first, this problem appears insurmountable, because Alice (who gets the program that includes Bob's input y) can potentially extract y by running it with many different inputs $X' \neq X$ and learn information from the output (i.e., perform a resetting attack [HLP11]). Indeed, given that the program cannot check whether Alice correctly opens the x_i's relative to the hash she sent to Bob, it may appear that this approach is doomed to fail. This very problem underpins the impossibility result of Hubáček and Wichs [HW15].

We get around this, however, by leaning on the fact that the output to the parties are pseudorandom. That is, we only need to guarantee the output of Alice is a valid secret share of the output if she provides the correct commitment—even if Alice equivocates, she obtains a pseudorandom string that leaks no information on y.

We now turn to explaining our construction.

Setup. The setup consists of generating the public SSB hash key hk. For some specific instantiations of the construction, we can have hk be randomly distributed and hence only need a common random string setup (see Section 6.6).

Step 1: Generating the encodings. Alice computes a commitment to her large (batch) input X using an SSB hash function with the key hk. Alice's public encoding simply consists of the hash output that we denote as c_X. Bob, on the other hand, generates an obfuscation of a program P that has a PRF key K and his input y hardwired inside. This program takes as input *some* commitment c'_X (which may be different from c_X), a batch element x_i, an index i, an opening π_i (generated with respect to c'_X), and the description of a function f. The program first checks that $\mathsf{SSB.Verify}(\mathsf{hk}, c'_X, x_i, i, \pi_i) = 1$ (i.e., the opening is accepting) and then outputs $\mathcal{U}(f, x_i, y) \oplus R_i$, where $\mathcal{U}$ is the universal circuit and R_i is a pseudorandom mask computed as $F_K(c'_X \| f \| i)$. We emphasize that c'_X that is fed as input to the program need not match with c_X that Alice sent to Bob as her public encoding. Moreover, we stress that the PRF evaluation $F_K(c'_X \| f \| i)$ used to compute the output mask does not include the input x_i nor opening π_i, which are provided as input to the program. Jumping ahead, this will be necessary so as to let Bob recompute the mask R_i "on his side" at decoding time, without knowledge of (x_i, π_i).

Remark 33. *We note that because the obfuscated program is generated for a universal circuit $\mathcal{U}$ that takes the function f as input, neither Alice nor Bob need to know the function f (except*

its size) when generating their respective encodings. This makes our $i\mathcal{O}$-based construction satisfy a stronger, function-adaptive variant of SMS, which we define formally in Section 6.4.

Step 2: Simultaneous communication. Alice sends c_X to Bob, which is of size $\mathsf{poly}(\lambda)$, and Bob sends the obfuscated program P to Alice, which is of size $\ell \cdot \mathsf{poly}(\lambda, \log L)$. In particular, the program P sketched above grows logarithmically with the batch size due to its dependence on the index i, for each input in the batch.

Step 3: Local decoding. For any function f (chosen adaptively at decoding time), Alice evaluates the obfuscated program on input (c'_X, x_i, i, π_i, f). Observe that if the verification passes, the program outputs $f(x_i, y) \oplus R_i$. Bob, on his end, computes $R_i := F_K(c_X \| f_i \| i)$, using his PRF key K. Observe that if $c_X = c'_X$, then Alice and Bob have shares of $f(x_i, y)$:

$$f(x_i, y) = \underbrace{(f(x_i, y) \oplus R_i)}_{\text{Alice's share}} \oplus \underbrace{R_i.}_{\text{Bob's share}}$$

They can repeat this process for all $i \in [L]$, and also any $f \in \mathcal{F}$, to obtain the shares of the function computed over all of Alice's L inputs $x_i \in X$.

Ideas behind the proof of security. As mentioned above, the commitment sent by Alice cannot be hardcoded into the program that is obfuscated by Bob. This makes the proof security more involved, since we need to consider all possible commitments that Alice can input into the program.

Let's start with Alice's security, which is the simpler case to analyze. Alice's message to Bob consists of an SSB hash of her private input. However, note that SSB hashing does *not* guarantee hiding of the input, making it possible to learn something about Alice's input message from the resulting hash. Our solution to this problem is to have Alice individually commit to each input (in her batch of inputs) using a regular, perfectly-binding and computationally-hiding commitment scheme, and then SSB hash the full set of commitments rather than her "raw" inputs. This still ensures a one-to-one mapping from inputs to the values hashed using the SSB hashing, and, in particular, preserves somewhere statistical binding. We modify the program sent by Bob to take a local opening to the SSB hash and also the opening to the underlying perfectly-binding commitment. With this modification, Alice's security reduces to the computational hiding of the commitment scheme.

Now we examine Bob's security. Bob's message to Alice consists of the obfuscated program P that has his private input y hardwired inside it. This program outputs $f(x_i, y) \oplus F_K(c'_X \| f \| i)$ if Alice is able to produce valid openings to x_i with respect to c'_X. If we assume that the obfuscation scheme satisfies VBB security [BGI+01], then the only thing that Alice can learn is this output. Because the output is masked with a PRF evaluation, this should intuitively provide no information about Bob's input.[5] However, as we detail below, care must be taken as the obfuscation only satisfies indistinguishability-based security.

To argue Bob's security, we change the obfuscated program P to a dummy program P^{sim} that just outputs $F_K(c'_X \| f \| i)$. Once we make this change, we can rely on the security of $i\mathcal{O}$ to remove the hardwired input y from the program. To switch the obfuscation from the real

[5] This requires a delicate argument as we must ensure that Alice can only provide valid openings to a single x_i for any i. This is argued using the somewhere binding property of the SSB hashing.

program to this dummy program, we rely on the punctured programming approach of Sahai and Waters [SW14]. Specifically, we consider a canonical ordering of the inputs $(c'_X \| f \| i)$ to the PRF and replace the obfuscated program to output a dummy PRF evaluation rather than the actual output, one input at a time, via a sequence of hybrids. But to make this change at a specific input $(c'_X \| f \| i)$, and switch from one hybrid to the next, we need to ensure that $F_K(c'_X \| f \| i)$ is used to mask only one output in the program P. This means that Alice should not be able to provide valid openings for two different x_i's with respect to the same hash c'_X. For this purpose, we rely on the somewhere binding property of the SSB hashing and the binding property of the underlying commitment scheme. Specifically, c'_X statistically determines Alice's input x_i if the SSB hashing is made to be binding at position i. Furthermore, the hiding property of SSB hashing allows us to switch the hashing key to be binding at location i without the adversary noticing this change. This allows us to make the security proof go through. Finally, because we have an exponential number of hybrids, we need to complexity leverage and rely on the sub-exponential security of $i\mathcal{O}$ and the PRF used to mask the outputs.

The full construction and proof are presented in Section 6.6.

6.3 Preliminaries

6.3.1 Notation

In this section, we cover the notation that we will use throughout the chapter.

Circuit and function classes. We define a class of circuits $\mathcal{C}$ to be a set of circuits, where each circuit $C \in \mathcal{C}$ has associated depth and size parameters. Unless otherwise stated, we will write $\mathcal{C} = \left\{C \colon \mathbb{Z}_q^\ell \to \mathbb{Z}_q^m\right\}$ to mean the set of all depth-d arithmetic (or Boolean in the case of $q = 2$) circuits of polynomial size that take ℓ inputs and produce m outputs in $\mathbb{Z}_q$. We will occasionally write C_i to indicate the circuit that computes the i-th output of a circuit C. Unless otherwise stated, we refer to a function family $\mathcal{F}$ as the set of functions represented by circuits in $\mathcal{C}$.

General notation. We let $\mathbb{N}$ denote the set of natural numbers. Unless otherwise stated, we will use $\mathsf{poly}(\cdot)$ to denote the set of all polynomials.

Sampling and assignment. We let $x \xleftarrow{\mathrm{R}} S$ denote a uniformly random sample drawn from S. We let $x \leftarrow \mathcal{A}$ denote assignment from a possibly randomized algorithm $\mathcal{A}$. We let $x := y$ denote initialization of x to the value of y.

Vectors and matrices. We denote a vector $\mathbf{v}$ using bold lowercase letters and a matrix $\mathbf{A}$ using bold uppercase letters. The i-th coordinate of a vector $\mathbf{v}$ is denoted by $\mathbf{v}[i]$. We will occasionally write $(v_i)_{i=1}^n$ to denote the vector $(v_1, \ldots, v_n)$. The i-th bit of a bit-string s is denoted by s_i.

Rounding. We let $\lfloor x \rceil$ denote the rounding of a real number x to the nearest integer. For integers $q \geq p \geq 2$, we define the modular rounding function $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ as $\lfloor v \rceil_p = \lfloor (p/q) \cdot v \rceil$.

Efficiency. By an *efficient* algorithm $\mathcal{A}$ we mean that $\mathcal{A}$ is modeled by a (possibly non-uniform) Turing Machine that runs in probabilistic polynomial time.

Indistinguishability. We write $D_0 \approx_c D_1$ to mean that two distributions D_0 and D_1 are *computationally* indistinguishable to all efficient distinguishers $\mathcal{D}$ and $D_0 \approx_s D_1$ to mean that D_0 and D_1 are *statistically* indistinguishable distributions.

6.3.2 The learning with errors assumption

Here, we recall the learning with errors (LWE) assumption [Reg05].

Definition 6.3.1 (Learning With Errors)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter and let χ_τ denote a discrete Gaussian distribution over $\mathbb{Z}_q$ with noise parameter τ. The learning with errors (LWE) assumption $\mathsf{LWE}_{n,k,q,\tau}$ holds for the parameters $n = n(\lambda)$, $k = k(\lambda)$, $q = q(\lambda)$, $\tau = \tau(\lambda)$ if*

$$\left(\mathbf{A}, \mathbf{s}^\top \mathbf{A} + \mathbf{e}^\top\right) \approx_c (\mathbf{A}, \mathbf{u}),$$

where $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times k}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \xleftarrow{R} \chi_\tau^k$, and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^k$.

Fully homomorphic encryption. We recall here the definition of fully homomorphic encryption (FHE) [Gen09].

Definition 6.3.2 (Fully Homomorphic Encryption)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter and $\mathcal{M} = \mathcal{M}(\lambda)$ be a message space. A fully homomorphic encryption (FHE) scheme consists of four algorithms $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ with the following syntax:*

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$. *The randomized key generation algorithm takes as input the security parameter λ. It outputs a public key pk and a secret key sk.*
- $\mathsf{Enc}(\mathsf{pk}, x) \to \mathsf{ct}$. *The randomized encryption algorithm takes as input the public key pk and a message $x \in \mathcal{M}$. It outputs a ciphertext ct.*
- $\mathsf{Eval}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) \to \mathsf{ct}'$. *The deterministic evaluation algorithm takes as input the public key pk, an ℓ-argument, m-output function f, and a tuple of ℓ ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ encrypting messages $(x_1, \ldots, x_\ell)$. It outputs a tuple of m evaluated ciphertexts $(\mathsf{ct}'_1, \ldots, \mathsf{ct}'_m)$.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to x$. *The deterministic decryption algorithm takes as input the secret key sk and a ciphertext ct. It outputs a message x.*

When Dec takes as input a tuple of ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$ it is understood to mean that Dec is applied individually to each ciphertext in the tuple.

The above algorithms must satisfy the following properties:

Correctness. *There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all sets of messages $x_1, \ldots, x_\ell \in \mathcal{M}$ and all ℓ-argument, m-output functions f that can be represented by a*

polynomial-size circuit, we have:

$$\Pr\left[\begin{array}{c} \mathsf{Dec}(\mathsf{sk},(\mathsf{ct}'_1,\ldots,\mathsf{ct}'_m)) \\ = f(x_1,\ldots,x_\ell) \end{array} \;:\; \begin{array}{r} (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk},x_i), \forall i \in [\ell] \\ (\mathsf{ct}'_1,\ldots,\mathsf{ct}'_m) \leftarrow \mathsf{Eval}(\mathsf{pk},f,(\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell)) \end{array}\right] \geq 1-\mathsf{negl}(\lambda),$$

where the probability is over the randomness of KeyGen *and* Enc.

Compactness. *For all sets of messages $x_1,\ldots,x_\ell \in \mathcal{M}$ and all ℓ-argument, m-output functions f, it holds that:*

$$|\mathsf{Eval}(\mathsf{pk},f,\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell)| = \mathsf{poly}(\lambda,m),$$

for some fixed polynomial $\mathsf{poly}(\cdot)$*. That is, the output length of* Eval *is independent of the input length ℓ and function description f.*

Security. *For all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{l} b=b' \end{array} \;:\; \begin{array}{l} (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (x_0,x_1,\mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pk}) \\ b \xleftarrow{R} \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk},x_b) \\ b' \leftarrow \mathcal{A}(\mathsf{st},\mathsf{ct}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Definition 6.3.3 (Secret-Key Fully Homomorphic Encryption)**.** *An FHE scheme* $\mathsf{FHE} = (\mathsf{KeyGen},\mathsf{Enc},\mathsf{Eval},\mathsf{Dec})$ *is a* secret-key *FHE scheme if it satisfies the syntax, correctness, and compactness properties of Definition 6.3.2 without the public key* pk*.*

6.4 Defining SMS Secure Computation

In this section, we first present the formal definition of SMS, some natural extensions of it, and discuss connections to other primitives.

Definition 6.4.1 (Simultaneous-Message and Succinct Secure Computation)**.** *Let λ be a security parameter, $L = L(\lambda)$ be the input length of Alice (who has the large input X), $\ell = \ell(\lambda)$ be the input length of Bob (who has the small input y), $m = m(\lambda)$ be the output length, where L,ℓ,m are all polynomial in λ, and let*

$$\mathcal{F} = \left\{f\colon \{0,1\}^L \times \{0,1\}^\ell \to \{0,1\}^m\right\}$$

be a family of functions. A simultaneous-message and succinct (SMS) secure computation scheme for $\mathcal{F}$ consists of a tuple of five efficient algorithms,

$$\mathsf{SMS} = \big(\mathsf{Setup},(\mathsf{Encode}_A,\mathsf{Decode}_A),(\mathsf{Encode}_B,\mathsf{Decode}_B)\big),$$

with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter and outputs a common reference string (CRS)* crs.
- $\mathsf{Encode}_\sigma(\mathsf{crs}, f, x) \to (\mathsf{pe}_\sigma, \mathsf{st}_\sigma)$. *The randomized encoding algorithm is parameterized by a party identifier* $\sigma \in \{A, B\}$. *It takes as input the CRS* crs, *a description of a function* $f \in \mathcal{F}$, *and an input* x. *It outputs a public encoding* pe_σ *and secret state* st_σ.
- $\mathsf{Decode}_\sigma(\mathsf{crs}, f, \mathsf{pe}_{\overline{\sigma}}, \mathsf{st}_\sigma) \to z_\sigma$. *The deterministic decoding algorithm is parameterized by a party identifier* $\sigma \in \{A, B\}$. *It takes as input the CRS* crs, *a description of a function* $f \in \mathcal{F}$, *the public encoding* $\mathsf{pe}_{\overline{\sigma}}$ *belonging to party* $\overline{\sigma} \neq \sigma$, *and secret state* st_σ *belonging to party* σ. *It outputs an* m*-bit string* $z_\sigma \in \{0,1\}^m$.

The above functionality must satisfy correctness, and security, and succinctness:

Correctness. *For all security parameters* $\lambda \in \mathbb{N}$, *every pair of inputs* $(X, y) \in \{0,1\}^L \times \{0,1\}^\ell$, *and all functions* $f \in \mathcal{F}$, *an SMS scheme is said to be correct if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{rcl} & & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & & (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, f, X) \\ z_A \oplus z_B = f(X,y) & : & (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, f, y) \\ & & z_A := \mathsf{Decode}_A(\mathsf{crs}, f, \mathsf{pe}_B, \mathsf{st}_A) \\ & & z_B := \mathsf{Decode}_B(\mathsf{crs}, f, \mathsf{pe}_A, \mathsf{st}_B) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. *For all efficient adversaries* $\mathcal{A}$, *for all* $\sigma \in \{A, B\}$, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{rcl} & & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & & (x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b = b' & : & b \xleftarrow{R} \{0,1\} \\ & & (\mathsf{pe}_\sigma, \mathsf{st}_\sigma) \leftarrow \mathsf{Encode}_\sigma(\mathsf{crs}, x_b) \\ & & b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pe}_\sigma) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

In words, the public encoding computationally hides the input of the party.

ϵ-Input Succinctness. *An SMS scheme is said to be* ϵ-input succinct, *for some* $\epsilon \in [0, 1)$, *if for all security parameters* $\lambda \in \mathbb{N}$, *all* $\sigma \in \{A, B\}$, *every CRS* crs, *all inputs* $X, y \in \{0,1\}^L \times \{0,1\}^\ell$, *all output lengths* m, *and every* pe_σ *in the support of* Encode_σ, *it holds that*

$$|\mathsf{pe}_\sigma| \leq \mathsf{poly}(\lambda, \ell, m) \cdot L^\epsilon,$$

for some fixed polynomial poly. *In words, the public encoding generated by each party is sublinear in the size of the large input. If* $\epsilon = 0$, *then we say the SMS scheme is* fully input succinct.

We now define an additional (optional) property of output succinctness for SMS schemes. Similarly to the input succinctness property of Definition 6.4.1, output succinctness states

that the encoding of each party must be sublinear in the function output length. In contrast to input succinctness—which is an integral property of Definition 6.4.1—the notion of SMS remains interesting even if the encodings are not succinct in the function output length. Indeed, output succinctness can, in some cases, be meaningless (e.g., when computing functions that output a single bit).

Definition 6.4.2 (ϵ-Output Succinctness)**.** *An SMS scheme is said to be* ϵ-output succinct*, for some* $\epsilon \in [0,1)$*, if for all security parameters* $\lambda \in \mathbb{N}$*, all* $\sigma \in \{A, B\}$*, every CRS* crs*, all inputs* $X, y \in \{0,1\}^L \times \{0,1\}^\ell$*, all output lengths* $m \in \mathbb{N}$*, and all* pe_σ *in the support of* Encode_σ*, it holds that*

$$|\mathsf{pe}_\sigma| \leq \mathsf{poly}(\lambda, L, \ell) \cdot m^\epsilon,$$

for some fixed polynomial $\mathsf{poly}(\cdot)$*. In words, the public encoding generated by both parties is sublinear in the size of the output length. If* $\epsilon = 0$*, then we say the SMS scheme is* fully output succinct*.*

Definition 6.4.3 (Succinctness)**.** *We say an SMS scheme is* ϵ-succinct *if it is both* ϵ*-input and* ϵ*-output succinct. If* $\epsilon = 0$*, we say it is* fully succinct*.*

Definition 6.4.4 (Function Adaptive)**.** *We say an SMS scheme is* function adaptive *if* Encode_A *and* Encode_B *take as input the function size* $|f|$ *in place of the function* f*.*

Definition 6.4.5 (Additive Reconstruction)**.** *We say an SMS scheme has* additive *(resp.* subtractive*) reconstruction if the outputs of* Decode_σ *are defined over a finite Abelian group* $\mathbb{G}$ *(resp. over the integers) and the correctness property requires* $z_A + z_B \in \mathbb{G}$ *(resp.* $z_A - z_B \in \mathbb{Z}$*) equals* $f(X, y)$*.*

Definition 6.4.6 (Batch-Succinct SMS)**.** *Let* $L, l, \ell, m \in \mathbb{N}$ *be parameters of the SMS scheme. Let* $\mathcal{F}$ *be a family of batch functions, such that for each* $f \in \mathcal{F}$*,* f *takes a batch of inputs* $X \in (\{0,1\}^l)^L$ *and an input* $y \in \{0,1\}^\ell$*, and computes some function* $g(X[i], y)$ *with* m*-bit outputs, for each* $i \in [L]$*. An SMS scheme for the family* $\mathcal{F}$ *is said to be* ϵ-batch-succinct *if for all security parameters* $\lambda \in \mathbb{N}$*, all* $\sigma \in \{A, B\}$*, every CRS* crs*, all input batches* $X \in (\{0,1\}^l)^L$*, and all inputs* $y \in \{0,1\}^\ell$*, it holds that*

$$|\mathsf{pe}_\sigma| \leq \mathsf{poly}(\lambda, \ell, l, m) \cdot L^\epsilon.$$

In words, the public encoding generated by each party grows sublinearly in the batch size.

Remarks on the definition of SMS. We provide some observations pertaining to Definitions 6.4.1 and 6.4.2.

Remark 34 (Relation to a simulation-based definition)**.** *In Definition 6.4.1, we provide a game-based definition where the adversary must distinguish between encodings of two different adversarially-chosen messages. This definition is easier to work with and is conceptually simpler. In Section 6.11, we prove that this game-based definition can be generically transformed into a simulation-based definition modeled by an ideal functionality.*

Remark 35 (Common random string)**.** *We define the CRS as a common reference string for generality. In particular, some NIVOLE protocols (e.g., [ARS24, BCM+24]) satisfying Definition 6.4.1 have a structured common reference string. However, our constructions have a common* random *string.*

Remark 36 (On input succinctness)**.** *We note that input succinctness for both parties* simultaneously *is information-theoretically impossible to achieve, as already observed by Abram et al. [ARS24] in the context of succinct homomorphic secret sharing. In particular, we cannot even have an* insecure *protocol satisfying input succinctness for both parties simultaneously.*

Remark 37 (On output succinctness)**.** *Unlike input succinctness,* output succinctness *is only an interesting notion when satisfied for both parties. In particular, if only one of the public encodings is output-succinct, then the exchange of encodings will not necessarily be (i.e., if the information on the full output is present in one of the encodings).*

Remark 38 (Post-composition with linear functions)**.** *Definition 6.4.1 can be used to compute functions of the form:* $\mathbf{g} \otimes f(X, y)$*, where the decoding algorithm additionally takes the linear transformation* $\mathbf{g}$ *as input. Note that such* post composition *by linear functions (e.g., see [BGI15]) is automatically implied by the additive reconstruction property of SMS.*

Remark 39 (Public-key PCFs are not SMS)**.** *Interestingly, public-key pseudorandom correlation functions (PCFs) [OSY21, BCM+24] (see also Chapters 3 and 5) can be viewed as* fully output-succinct *instantiations of SMS but fail to achieve* input succinctness*! In particular, in a public-key PCF, each party's input is of size* $\mathsf{poly}(\lambda)$ *making the encoding linear in the input size (PCF key). In contrast, the encoding size (a party's public key, using PCF terminology) is independent of the output length, making PCFs fully output-succinct.*

6.4.1 Succinct, non-interactive VOLE as SMS

Here, we show how *succinct* Non-Interactive VOLE (NIVOLE) [ARS24, BCM+24] fits into our SMS definition. In a succinct NIVOLE scheme, Alice with a length-L input vector $\mathbf{a}$ and Bob with a scalar Δ (here, Bob's input length $\ell = 1$) compute additive shares of the vector $\Delta \cdot \mathbf{a}$, in a semi-honest, simultaneous-message protocol. Moreover, the succinctness property states that the communication of this single-round protocol is sublinear in L.

Definition 6.4.7 (Non-Interactive VOLE)**.** *We say that SMS instantiated with the function family* $\mathcal{F} = \left\{ f_p \colon \mathbb{Z}_p^L \times \mathbb{Z}_p \to \mathbb{Z}_p^L \mid f_p \colon (\mathbf{a}, \Delta) \mapsto \Delta \cdot \mathbf{a} \right\}_{p \in \mathbb{N}}$*, is an SMS scheme for non-interactive VOLE, denoted* NIVOLE*. We drop the subscript* p *from* f_p *when the ring* $\mathbb{Z}_p$ *is clear from context. We also omit the function* f_p *from* Encode_σ *and* Decode_σ *when the ring* $\mathbb{Z}_p$ *is fixed.*

Theorem 6.4.1 (Succinct NIVOLE from LWE [ARS24])**.** *For any integer* p*, assuming the hardness of LWE with a superpolynomial modulus-to-noise ratio, there exists an* $(2/3)$*-succinct scheme for NIVOLE instantiated over* $\mathbb{Z}_p$*.*

Batch non-interactive VOLE. We remark that we can view NIVOLE itself as a batch computation, since the same Δ is applied to all entries of Alice's large input $\mathbf{a}$. We define the following extension to NIVOLE which explicitly satisfies batch-SMS for NIVOLE.

Definition 6.4.8 (Batch NIVOLE)**.** *We define Batch NIVOLE for computing L matrix-vector product using the same vector. Concretely,*

$$\mathsf{BNIVOLE} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma\in\{A,B\}}),$$

where Encode_A takes as input a list of matrices $(\mathbf{A}_i)_{i=1}^L$, where each $\mathbf{A}_i \in \mathbb{Z}_p^{\ell\times l}$, and Encode_B takes as input a vector $\mathbf{b} \in \mathbb{Z}_p^\ell$. Then, for all $\sigma \in \{A, B\}$, Decode_σ outputs an additive share of $(\mathbf{b}^\top \mathbf{A}_i)_{i=1}^L$.

Lemma 6.4.1. *If there exists a succinct NIVOLE scheme with ϵ-succinctness, then there exists a batch NIVOLE scheme with ϵ-succinctness.*

Proof sketch. The construction is very simple: It suffices to run ℓ instances of NIVOLE in parallel, where Alice inputs the rows of the matrix $\mathbf{H} \in \mathbb{Z}_p^{(\ell\times l)\cdot L}$ consisting of matrices $(\mathbf{A}_i)_{i=1}^L$ concatenated together, and Bob inputs his vector $\mathbf{b}$. By the succinctness of NIVOLE, multiplying each entry of $\mathbf{b}$ by the corresponding row vector of $\mathbf{H}$ requires $(l \cdot L)^\epsilon$ communication (in particular, Alice's public encoding is sublinear in L). Then, by the post-composition with linear functions (cf. Remark 38), the columns of the resulting matrix can be summed together to obtain $\mathbf{b}^\top\mathbf{H}$. Moreover, this protocol satisfies Definition 6.4.6, since Alice's encoding is of size $\mathsf{poly}(\lambda, \ell) \cdot (l \cdot L)^\epsilon \leq \mathsf{poly}(\lambda, \ell, l) \cdot L^\epsilon$. ■

Remark 40. *We note, in passing, that in the case of succinct NIVOLE, input and output succinctness are equivalent definitions, since the output length is identical to the input length of the party with the large input vector.*

6.5 Construction from LWE

In this section, we present our LWE-based construction of SMS achieving both input and output succinctness.

6.5.1 Preliminaries

In this section, we present the necessary definitions and building blocks that we will use in our LWE-based construction of SMS.

Auxiliary functions. Here, we recall the algorithms EvalPK and EvalCT introduced in Boneh et al. [BGG+14] and the extensions of Gorbunov et al. [GVW15].

Definition 6.5.1 (Gadget Matrix [MP12])**.** *Let $q \geq 2$ be an integer. We call*

$$\mathbf{g} := (1, 2, \ldots, 2^{\lceil \log q\rceil - 1}) \in \mathbb{Z}_q^{\lceil \log q\rceil}$$

the gadget vector. *We call $\mathbf{G} := \mathbf{g} \otimes \mathbf{I}_n$ the* gadget matrix.

Definition 6.5.2 (Auxiliary Evaluation Algorithms [BGG+14, GVW15])**.** *Let α, β be integer parameters and let* crs *be a common random string (CRS). The auxiliary evaluation algorithms are two efficient and deterministic procedures* $(\mathsf{EvalPK}, \mathsf{EvalCT})$ *with the following syntax:*

- $\mathsf{EvalPK}(\mathsf{crs}, C) \to \mathbf{A}_{\mathsf{IP} \circ C}$. *Takes as input the CRS* crs *and a circuit* $C \colon \{0,1\}^\alpha \to \mathbb{Z}_q^\beta$*, and outputs a matrix* $\mathbf{A}_{\mathsf{IP} \circ C} \in \mathbb{Z}_q^{n \times k}$.
- $\mathsf{EvalCT}(\mathsf{crs}, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, C, x) \to \mathbf{w}_{\mathsf{IP} \circ C}$. *Takes as input the CRS* crs*, a list of vectors* $(\mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta)$*, the circuit* C*, and outputs a vector* $\mathbf{w}_{\mathsf{IP} \circ C} \in \mathbb{Z}_q^k$.

Lemma 6.5.1 (Adapted from [GVW15])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter and $B = B(\lambda)$ be an integer bound. Under the* $\mathsf{LWE}_{n,k,q,\tau}$ *assumption with $k := n\lceil \log q \rceil$, there exist algorithms* $(\mathsf{EvalPK}, \mathsf{EvalCT})$ *satisfying Definition 6.5.2 for all integers $\alpha = \alpha(\lambda), \beta = \beta(\lambda)$ that are polynomial in the security parameter, such that for all common random strings of the form:* $\mathsf{crs} := (\mathbf{A}_1, \ldots, \mathbf{A}_\alpha, \mathbf{B}_1, \ldots, \mathbf{B}_\beta) \in (\mathbb{Z}_q^{n \times k})^{\alpha+\beta}$*, for all $\alpha+\beta$ vectors $\mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta \in \mathbb{Z}_q^k$, all $\mathbf{s} \in \mathbb{Z}_q^n$, all $(x, \mathbf{y}) \in \{0,1\}^\alpha \times \mathbb{Z}_q^\beta$, and all arithmetic circuits $C \colon \{0,1\}^\alpha \to \mathbb{Z}_q^\beta$ of depth d, if it holds that:*

$$\forall i \in [\alpha],\ \mathbf{u}_i = \mathbf{s}^\top (\mathbf{A}_i + x_i \cdot \mathbf{G}) + \mathbf{e}_i^\top \text{ and } ||\mathbf{e}_i||_\infty \leq B,$$
$$\forall i \in [\beta],\ \mathbf{v}_i = \mathbf{s}^\top (\mathbf{B}_i + \mathbf{y}[i] \cdot \mathbf{G}) + \mathbf{e}_i^\top \text{ and } ||\mathbf{e}_i||_\infty \leq B,$$

then it also holds that $\mathbf{w}_{\mathsf{IP} \circ C} := \mathsf{EvalCT}(\mathsf{crs}, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, C, x)$ *is of the form:*

$$\mathbf{w}_{\mathsf{IP} \circ C} = \mathbf{s}^\top (\mathbf{A}_{\mathsf{IP} \circ C} + \langle C(x), \mathbf{y} \rangle \cdot \mathbf{G}) + \mathbf{e}^\top \text{ with } ||\mathbf{e}||_\infty \leq (k+1)^d \cdot B,$$

where $\mathbf{A}_{\mathsf{IP} \circ C} := \mathsf{EvalPK}(\mathsf{crs}, C)$ *and* $\mathbf{G}$ *is the gadget matrix from Definition 6.5.1.*

Remark 41 (Public and private inputs)**.** *We stress that* EvalCT *does not take* $\mathbf{y}$ *as input. As such, we can view* EvalCT *as taking a "public" input x and "private" input* $\mathbf{y}$ *(encoded in the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_\beta$), and using these to evaluate functions of the form $\langle C(x), \mathbf{y} \rangle$. Inspired by Gorbunov et al. [GVW15]'s approach for building predicate encryption, we will use this fact to let* $\mathbf{y}$ *be a secret decryption key that will allow Alice to obliviously decrypt an FHE ciphertext output by C.*

Function-hiding. We now formalize the transformation used implicitly in the work of Quach et al. [QWW18] to make $\mathbf{A}_C$ (as output by EvalPK) statistically hiding.

Lemma 6.5.2 (Function-hiding Public Keys)**.** *Let γ be an integer. There exist efficient wrapper algorithms* $\widetilde{\mathsf{EvalPK}}$ *and* $\widetilde{\mathsf{EvalCT}}$ *defined as in Figure 6.1 such that:*

(1) $\widetilde{\mathsf{EvalPK}}$ *and* $\widetilde{\mathsf{EvalCT}}$ *satisfy the properties defined in Lemma 6.5.1,*

(2) if it holds that $\forall i \in [\gamma]$, $\mathbf{t}_i = \mathbf{s}^\top (\mathbf{C}_i + \mathbf{e}_i^\top)$, where $||\mathbf{e}_i||_\infty \leq B$, then it holds that the error magnitude in $\mathbf{w}_{\mathsf{IP} \circ C}$ *(output by* $\widetilde{\mathsf{EvalCT}}$*) is at most an additive factor γB larger compared to the bound in Lemma 6.5.1, and*

(3) if $\gamma \geq 2nk\lceil \log q \rceil$ and q is prime (see Lemma 6.5.1 for parameter details), then $\mathbf{A}_C$ is statistically close to the uniform distribution over $\mathbb{Z}_q^{n \times k}$.

QWW18 Function-Hiding Transformation

$\widetilde{\mathsf{EvalPK}}(\mathsf{crs}, C, \mathbf{C}_1, \ldots, \mathbf{C}_\gamma)$

1 : $r \xleftarrow{R} \{0,1\}^\gamma$
2 : $\mathbf{A}'_C \leftarrow \mathsf{EvalPK}(\mathsf{crs}, C)$
3 : $\mathbf{A}_C := \mathbf{A}'_C + \sum_{i=1}^{\gamma} r_i \mathbf{C}_i$
4 : **return** $(\mathbf{A}_C, r)$

$\widetilde{\mathsf{EvalCT}}(\mathsf{crs}, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, C, x, \mathbf{t}_1, \ldots, \mathbf{t}_\gamma, r)$

1 : **parse** $r = r_1 \| \cdots \| r_\gamma$
2 : $\mathsf{ct}' \leftarrow \mathsf{EvalCT}(\mathsf{crs}, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, C, x)$
3 : $\mathbf{w}_{\mathsf{IP} \circ C} := \mathbf{w}'_{\mathsf{IP} \circ C} + \sum_{i=1}^{\gamma} r_i \mathbf{t}_i$
4 : **return** $\mathbf{w}_{\mathsf{IP} \circ C}$

Figure 6.1: Function-hiding transformation of Quach et al. [QWW18].

The transformation from Figure 6.1 is implicit in Appendix A of Quach et al. [QWW18]; we extract it here as a standalone wrapper for algorithms EvalPK *and* EvalCT*.*

Theorem 6.5.1 (FHE with Near-linear Decryption [BV11, BGV12, GSW13])**.** *Under the* $\mathsf{LWE}_{n,k,q,\tau}$ *assumption (cf. Definition 6.3.1), there exists a fully homomorphic encryption scheme* $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *for computing depth-d circuits, where the secret keys are vectors in* $\mathbb{Z}_q^\beta$ *and where the evaluation algorithm* $\mathsf{FHE.Eval}$ *outputs a vector* $\mathsf{ct} \in \mathbb{Z}_q^\beta$ *such that, for the corresponding secret key* $\mathbf{s}$*, it holds that (with probability 1):*

$$\langle \mathsf{ct}, \mathbf{s} \rangle = \lfloor q/p \rceil \cdot \mathsf{FHE.Dec}(\mathbf{s}, \mathsf{ct}) + e \bmod q$$

for some error $e \in \mathbb{Z}_q$ *and where* $|e| < (k+1)^d \cdot \mathsf{poly}(\lambda)$*. Under the circular-security of the* $\mathsf{LWE}_{n,k,q,\tau}$ *assumption,* FHE *can be used to compute all polynomial-size circuits of unbounded depth such that* $|e| < (k+1) \cdot \mathsf{poly}(\lambda)$*. Moreover, for all circuits C of depth d,* $\mathsf{FHE.Eval}(C, \cdot)$ *can itself be computed by a circuit of depth* $d \cdot \mathsf{polylog}(\lambda)$ *and size* $|C| \cdot \mathsf{poly}(\lambda, d)$*.*

6.5.2 Construction

Our construction is presented in Figure 6.2 and closely follows the technical overview.

6.5.3 Setting the parameters

For the LWE-based construction, we make use of Lemma 6.5.1 to evaluate an FHE evaluation on a ciphertext. In the construction, the circuit C computes $\mathsf{FHE.Eval}(f, \cdot)$ on some input ciphertext ct, where f is a function that is represented by a depth-d circuit. Therefore, using Theorem 6.5.1, the circuit C must have depth $d' \geq d \cdot \mathsf{polylog}(\lambda) \in \mathsf{poly}(\lambda, d)$.

We can set the LWE parameters n, k, q, τ, required for Lemma 6.5.1 and Theorem 6.5.2, as follows. Let $n = \mathsf{poly}(\lambda)$ and let B be an integer bound on the noise distribution determined by the parameter τ. We let $k = n\lceil \log q \rceil$. Then, for correctness, we need $q > 2^{\mathsf{poly}(\lambda, d')}$, for some polynomial $\mathsf{poly}(\cdot)$, subject to $q > 4 \cdot (k+1)^{d'} \cdot B \cdot \lambda^{\omega(1)}$.

SMS from LWE

Public Parameters. Let m be the function output length, n, k, q, χ_τ be LWE parameters (cf. Definition 6.3.1) as required by Lemma 6.5.1 and Theorem 6.5.1, $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a secret-key FHE scheme satisfying Definition 6.3.3, $F\colon \{0,1\}^\lambda \times [m] \to \mathbb{Z}_q$ be a PRF, α be the length (in bits) of an FHE ciphertext encrypting ℓ bits, β be the length (in elements of $\mathbb{Z}_q$) of the FHE secret key from Theorem 6.5.1, and γ be as required in Lemma 6.5.2.

$\mathsf{SMS.Setup}(1^\lambda)$

1 : $\mathsf{crs}_{\mathsf{aux}} := (\mathbf{A}_1, \ldots, \mathbf{A}_\alpha, \mathbf{B}_1, \ldots, \mathbf{B}_\beta) \xleftarrow{\mathrm{R}} (\mathbb{Z}_q^{n\times k})^{\alpha+\beta}$

2 : $\mathsf{crs}_{\mathsf{rnd}} := (\mathbf{C}_1, \ldots, \mathbf{C}_\gamma) \xleftarrow{\mathrm{R}} (\mathbb{Z}_q^{n\times k})^{\gamma}$

3 : $\mathsf{crs}_{\mathsf{bvole}} \leftarrow \mathsf{BNIVOLE.Setup}(1^\lambda)$ ▷ See Theorem 6.4.1.

4 : $K \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$ ▷ PRF key for randomizing output shares.

5 : **return** $\mathsf{crs} := (\mathsf{crs}_{\mathsf{aux}}, \mathsf{crs}_{\mathsf{rnd}}, \mathsf{crs}_{\mathsf{bvole}}, K)$

- See Figure 6.3 for the description of the encoding algorithms $\mathsf{SMS.Encode}_\sigma$.
- See Figure 6.4 for the description of the decoding algorithms $\mathsf{SMS.Decode}_\sigma$.

Figure 6.2: Simultaneous-Message Succinct Secure Computation from LWE.

SMS from LWE: Encoding Algorithms

$\mathsf{SMS.Encode}_A(\mathsf{crs}, f, X)$

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{aux}}, \mathsf{crs}_{\mathsf{rnd}}, \mathsf{crs}_{\mathsf{bvole}}, K)$
2 : Define an arithmetic circuit $C(\cdot)$ computing $\mathsf{FHE.Eval}(f, (X, \mathsf{ct}))$ on input $\mathsf{ct} \in \{0,1\}^\alpha$, where X is hardcoded as an input in C.
3 : Let C_i be the circuit that computes the i-th output bit of C.
4 : **foreach** $i \in [m]$:
5 : $\quad (\mathbf{A}_{C_i}, r_i) \leftarrow \widetilde{\mathsf{EvalPK}}(\mathsf{crs}_{\mathsf{aux}}, C_i, \mathsf{crs}_{\mathsf{rnd}})$ ▷ See Lemma 6.5.2.
6 : $(\mathsf{pe}_A^{\mathsf{bvole}}, \mathsf{st}_A^{\mathsf{bvole}}) \leftarrow \mathsf{BNIVOLE.Encode}_A(\mathsf{crs}_{\mathsf{bvole}}, (\mathbf{A}_{C_i})_{i=1}^m)$ ▷ See Definition 6.4.8.
7 : $\mathsf{pe}_A := \mathsf{pe}_A^{\mathsf{bvole}}, \quad \mathsf{st}_A := (\mathsf{st}_A^{\mathsf{bvole}}, r_1, \ldots, r_m)$
8 : **return** $(\mathsf{pe}_A, \mathsf{st}_A)$

$\mathsf{SMS.Encode}_B(\mathsf{crs}, f, y)$

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{aux}}, \mathsf{crs}_{\mathsf{rnd}}, \mathsf{crs}_{\mathsf{bvole}}, K)$
2 : $\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ subject to $\mathbf{s}[1] = 1$
3 : $\mathsf{sk} \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$
4 : $\mathsf{ct}_y \leftarrow \mathsf{FHE.Enc}(\mathsf{sk}, y)$
5 : **parse** $\mathsf{ct}_y = c_1 \| c_2 \| \cdots \| c_\alpha \in \{0,1\}^\alpha$
6 : **foreach** $i \in [\alpha]$:
7 : $\quad \mathbf{e}_i \xleftarrow{\mathrm{R}} \chi_\tau^k, \quad \mathbf{u}_i := \mathbf{s}^\top(\mathbf{A}_i + c_i \cdot \mathbf{G}) + \mathbf{e}_i^\top$ ▷ Bit-wise encryptions of ct_y.
8 : **parse** $\mathsf{sk} = \mathsf{sk}_1 \| \mathsf{sk}_2 \| \cdots \| \mathsf{sk}_\beta \in \mathbb{Z}_q^\beta$ ▷ See Theorem 6.5.1.
9 : **foreach** $i \in [\beta]$:
10 : $\quad \mathbf{e}'_i \xleftarrow{\mathrm{R}} \chi_\tau^k, \quad \mathbf{v}_i := \mathbf{s}^\top(\mathbf{B}_i + \mathsf{sk}_i \cdot \mathbf{G}) + \mathbf{e}_i'^\top$
11 : **foreach** $i \in [\gamma]$:
12 : $\quad \mathbf{e}''_i \xleftarrow{\mathrm{R}} \chi_\tau^k, \quad \mathbf{t}_i := \mathbf{s}^\top \mathbf{C}_i + \mathbf{e}_i''^\top$
13 : $(\mathsf{pe}_B^{\mathsf{bvole}}, \mathsf{st}_B^{\mathsf{bvole}}) \leftarrow \mathsf{BNIVOLE.Encode}_B(\mathsf{crs}_{\mathsf{bvole}}, \mathbf{s})$
14 : $\mathsf{pe}_B := (\mathsf{ct}_y, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, \mathbf{t}_1, \ldots, \mathbf{t}_\gamma, \mathsf{pe}_B^{\mathsf{bvole}}), \quad \mathsf{st}_B := \mathsf{st}_B^{\mathsf{bvole}}$
15 : **return** $(\mathsf{pe}_B, \mathsf{st}_B)$

Figure 6.3: Encoding algorithms for SMS from LWE (Figure 6.2).

SMS from LWE: Decoding Algorithms

$\mathsf{SMS.Decode}_A(\mathsf{crs}, f, \mathsf{pe}_B, \mathsf{st}_A)$

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{aux}}, \mathsf{crs}_{\mathsf{rnd}}, \mathsf{crs}_{\mathsf{bvole}}, K)$
2 : **parse** $\mathsf{pe}_B = (\mathsf{ct}_y, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, \mathbf{t}_1, \ldots, \mathbf{t}_\gamma, \mathsf{pe}_B^{\mathsf{bvole}})$
3 : **parse** $\mathsf{st}_A = (\mathsf{st}_A^{\mathsf{bvole}}, r_1, \ldots, r_m)$
4 : **foreach** $i \in [m]$:
5 : $\quad \mathbf{w}_{\mathsf{IP} \circ C_i} \leftarrow \widetilde{\mathsf{EvalCT}}(\mathsf{crs}, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, C_i, \mathsf{ct}_y, \mathbf{t}_1, \ldots, \mathbf{t}_\gamma, r_i)$
6 : $(\mathbf{d}_A^{(i)})_{i=1}^m \leftarrow \mathsf{BNIVOLE.Decode}_A(\mathsf{crs}_{\mathsf{bvole}}, \mathsf{pe}_B^{\mathsf{bvole}}, \mathsf{st}_A^{\mathsf{bvole}})$
7 : **foreach** $i \in [m]$:
8 : $\quad z_A^{(i)} := \left\lfloor (\mathbf{d}_A^{(i)} + \mathbf{w}_{\mathsf{IP} \circ C_i})[1] + F_K(i) \right\rceil_2$
9 : **return** $(z_A^{(1)}, \ldots, z_A^{(m)})$

$\mathsf{SMS.Decode}_B(\mathsf{crs}, f, \mathsf{pe}_A, \mathsf{st}_B)$

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{aux}}, \mathsf{crs}_{\mathsf{rnd}}, \mathsf{crs}_{\mathsf{bvole}}, K)$
2 : **parse** $\mathsf{pe}_A = \mathsf{pe}_A^{\mathsf{bvole}}$ **and** $\mathsf{st}_B = \mathsf{st}_B^{\mathsf{bvole}}$
3 : $(\mathbf{d}_B^{(i)})_{i=1}^m \leftarrow \mathsf{BNIVOLE.Decode}_B(\mathsf{crs}_{\mathsf{bvole}}, \mathsf{pe}_A^{\mathsf{bvole}}, \mathsf{st}_B^{\mathsf{bvole}})$
4 : **foreach** $i \in [m]$:
5 : $\quad z_B^{(i)} := \left\lfloor \mathbf{d}_B^{(i)}[1] + F_K(i) \right\rceil_2$
6 : **return** $(z_B^{(1)}, \ldots, z_B^{(m)})$

Figure 6.4: Decoding algorithms for SMS from LWE (Figure 6.2).

6.5.4 Security analysis

Here, we analyze the correctness and security of the SMS construction from Figure 6.2. We prove the following theorem.

Theorem 6.5.2. *Let λ be a security parameter and $d = d(\lambda) \in \mathsf{poly}(\lambda)$ be a circuit depth. Assume that the LWE assumption holds with a superpolynomial modulus-to-noise ratio. Then, Figure 6.2 is an SMS scheme satisfying Definition 6.4.1 for all functions that can be represented by polynomial-size, depth-d circuits. Furthermore, the scheme achieves full input succinctness and $(2/3)$-output succinctness (cf. Definition 6.4.2).*

Proof. We prove each required property in turn.

Correctness. We argue correctness for the i-th output bit, for any $i \in [m]$. By construction, we have that $\mathbf{z}_A^{(i)}$, as computed by Alice, is:

$$
\begin{aligned}
\mathbf{w}_{\mathsf{IP} \circ C_i} &:= \widetilde{\mathsf{EvalCT}}(\mathsf{crs}, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, C_i, \mathsf{ct}_y, \mathbf{t}_1, \ldots, \mathbf{t}_\gamma, r_i) \\
&= \mathbf{s}^\top (\mathbf{A}_C + \langle C_i(\mathsf{ct}_y), \mathsf{sk} \rangle \cdot \mathbf{G}) + \mathbf{e}^\top && \triangleright \text{ Follows from Lemmas 6.5.1 and 6.5.2.} \\
&= \mathbf{s}^\top \left(\mathbf{A}_C + \langle \mathsf{FHE.Eval}(f_i, (X, \mathsf{ct}_y)), \mathsf{sk} \rangle \cdot \mathbf{G}\right) + \mathbf{e}^\top && \triangleright \text{ Definition of } C_i. \\
&= \mathbf{s}^\top \left(\mathbf{A}_C + \langle \mathsf{FHE.Enc}(\mathsf{sk}, f_i(X, y)), \mathsf{sk} \rangle \cdot \mathbf{G}\right) + \mathbf{e}^\top && \triangleright \text{ Correctness of FHE.} \\
&= \mathbf{s}^\top \left(\mathbf{A}_C + \left(f_i(X, y)\frac{q}{2} + \mathbf{e}'^\top\right) \cdot \mathbf{G}\right) + \mathbf{e}^\top && \triangleright \text{ Near-linear decryption.} \\
&= \mathbf{s}^\top \mathbf{A}_C + \mathbf{s}^\top \left(\left(f_i(X, y)\frac{q}{2} + \mathbf{e}'^\top\right) \cdot \mathbf{G}\right) + \mathbf{e}^\top,
\end{aligned}
$$

where f_i is the function that computes the i-th bit of f.

The first equality follows directly from the correctness of $\widetilde{\mathsf{EvalCT}}$, as defined in Lemmas 6.5.1 and 6.5.2. The second equality follows from the definition of the circuit C_i, the properties of $\mathsf{FHE.Eval}$ from Theorem 6.5.1, and choice of parameters in Section 6.5.3. The third equality follows from the correctness of the FHE scheme and the fact that X can, without loss of generality, be converted to a ciphertext by having ct_y contain auxiliary encryptions of 0 and 1 (allowing any evaluator to convert their plaintext input into ciphertexts encrypted under the secret key). The fourth equality follows from the near-linear decryption property of the FHE scheme.

Then, because we set $\mathbf{s}[1] = 1$, we have that $\mathbf{w}_{\mathsf{IP} \circ C_i}[1] = (\mathbf{s}^\top \mathbf{A}_C)[1] + f_i(X, y)\frac{q}{2} + e' + e$, where e' and e are the first entries of $\mathbf{e}'$ and $\mathbf{e}$, respectively. To see this, it is helpful to note that the first column of $\mathbf{G}$ is of the form $(1, 0, \ldots, 0)^\top$; see Definition 6.5.1.

Then, by correctness of $\mathsf{BNIVOLE}$, we have that:

$$
\begin{aligned}
&\mathsf{BNIVOLE.Decode}_A(\mathsf{crs}_{\mathsf{bvole}}, \mathsf{pe}_B^{\mathsf{bvole}}, \mathsf{st}_A^{\mathsf{bvole}})[i] \\
&\quad - \mathsf{BNIVOLE.Decode}_B(\mathsf{crs}_{\mathsf{bvole}}, \mathsf{pe}_A^{\mathsf{bvole}}, \mathsf{st}_B^{\mathsf{bvole}})[i] = \mathbf{d}_A^{(i)} - \mathbf{d}_B^{(i)} = \mathbf{s}^\top \mathbf{A}_{C_i},
\end{aligned}
$$

with all but negligible probability.

Therefore, we have that with all but negligible probability, $\mathbf{z}_A^{(i)}$ and $\mathbf{z}_B^{(i)}$, as computed in

$\mathsf{SMS.Decode}_A$ and $\mathsf{SMS.Decode}_B$, respectively, satisfy:

$$\begin{aligned}\mathbf{z}_A^{(i)} - \mathbf{z}_B^{(i)} &= \mathbf{d}_A^{(i)} + \mathbf{w}_{\mathsf{IP}\circ C_i} - \mathbf{d}_B^{(i)} \\ &= \mathbf{w}_{\mathsf{IP}\circ C_i} - \mathbf{s}^\top \mathbf{A}_{C_i} \\ &= \left(\mathbf{s}^\top \mathbf{A}_{C_i} + \mathbf{s}^\top \left(\left(f_i(X,y)\frac{q}{2} + \mathbf{e}'^\top\right)\cdot \mathbf{G}\right) + \mathbf{e}^\top\right) - \mathbf{s}^\top \mathbf{A}_{C_i}.\end{aligned}$$

And so it holds that, with all but negligible probability,

$$\mathbf{z}_A^{(i)}[1] - \mathbf{z}_B^{(i)}[1] = \mathbf{w}_{\mathsf{IP}\circ C_i}[1] - \mathbf{s}^\top \mathbf{A}_{C_i}[1] = f_i(X,y)\frac{q}{2} + e' + e.$$

Importantly, we have that e' (the FHE decryption error) and e (the EvalCT evaluation error) are bounded in magnitude by Lemma 6.5.1 and Theorem 6.5.1. Specifically, we have that $\max(|e|, |e'|) \leq (k+1)^d \cdot \mathsf{poly}(\lambda)$, and so it holds that $|e + e'| \leq 2(k+1)^d \cdot \mathsf{poly}(\lambda)$. Therefore, if we have that $q > 4(k+1)^d \cdot \lambda^{\omega(1)}$ (which necessitates assuming LWE security holds with a superpolynomial modulus-to-noise ratio), then by Lemma 6.2.1 and the above analysis, we get that:

$$\Pr\Big[\ \Big\lfloor \mathbf{z}_A^{(i)}[1] + F_K(i)\Big\rceil_2 - \Big\lfloor \mathbf{z}_B^{(i)}[1] + F_K(i)\Big\rceil_2 = \Big\lfloor \mathbf{z}_A^{(i)}[1] - \mathbf{z}_B^{(i)}[1]\Big\rceil_2 = f_i(X,y)\ \Big] \geq 1 - \mathsf{negl}(\lambda),$$

where the probability is over randomness of $\mathbf{s}$ and the PRF key K. In particular, the PRF ensures a pseudorandom distribution over $\mathbb{Z}_q$ (and is equivalent to randomizing the subtractive shares), which then allows us to apply Lemma 6.2.1. It follows that the outputs of $\mathsf{SMS.Encode}_A$ and $\mathsf{SMS.Encode}_B$ form subtractive shares of all m output bits of $f(X,y)$, with all but negligible probability in λ.

This concludes the proof of correctness.

Succinctness. We now briefly analyze the input and output succinctness. For input succinctness, observe that Alice's encoding pe_A consists only of the $\mathsf{BNIVOLE}$ public encoding of the batch NIVOLE SMS scheme involving m matrices, where each matrix is of size $\mathsf{poly}(\lambda, d)$ by Definition 6.5.2. In particular, we get full *input* succinctness, since the size of pe_A is independent of Alice's input length $|X|$ and only depends on the output length m.

Then, for output succinctness, by Lemma 6.4.1 and Theorem 6.4.1 we have that $|\mathsf{st}_A^{\mathsf{bvole}}| \leq \mathsf{poly}(\lambda, d) \cdot m^\epsilon$ with $\epsilon = (2/3)$, where d is the circuit depth and is implicit in the choice of modulus q. Moreover, Bob's encoding pe_B consists of the batch NIVOLE public encoding, which has size $\mathsf{poly}(\lambda, d) \cdot m^\epsilon$ with $\epsilon = (2/3)$ along with (1) an FHE ciphertext encrypting the input y under the secret key sk and (2) $(\alpha + \beta + \gamma)$ ciphertexts encrypted under $\mathbf{s}$. These ciphertexts all have, at most, a linear dependence on $|y|$, and thus have a combined length of $|y| \cdot \mathsf{poly}(\lambda, d)$. Thus, Alice's and Bob's encoding are both sublinear in the output length m, which proves the ϵ output succinctness property.

Security for Alice. We show that Alice's encoding reveals no information on her private input X. Note that Alice's public encoding pe_A consists of $\mathbf{A}_C$ computed according to $\widetilde{\mathsf{EvalPK}}(\mathsf{crs}, C, \mathbf{C}_1, \ldots, \mathbf{C}_\gamma)$, where C has X hardcoded inside it. By Lemma 6.5.2, we have that $\mathbf{A}_C$ is statistically close to uniform, and so indistinguishability holds trivially.

Security for Bob. We prove that the real view of the adversary is computationally

indistinguishable to a simulated view. First, we construct the following efficient simulator $\mathcal{S}$ for pe_B (Bob's public encoding from Figure 6.2).

$\mathcal{S}$: On input crs,

- Parse $\mathsf{crs} = (\_, \_, \mathsf{crs}_{\mathsf{bvole}})$.
- Sample $\mathbf{u}_1, \ldots, \mathbf{u}_\alpha$, $\mathbf{v}_1, \ldots, \mathbf{v}_\beta$, and $\mathbf{t}_1, \ldots, \mathbf{t}_\gamma$ uniformly at random.
- Sample $\mathsf{sk} \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$.
- Compute $\mathsf{ct}_y \leftarrow \mathsf{FHE.Enc}(\mathsf{sk}, 0)$.
- Compute $\mathsf{pe}_B^{\mathsf{bvole}} \leftarrow \mathsf{BNIVOLE.Encode}_B(\mathsf{crs}_{\mathsf{bvole}}, 0)$.
- Output $\mathsf{pe}_B := (\mathsf{ct}_y, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, \mathbf{t}_1, \ldots, \mathbf{t}_\gamma, \mathsf{pe}_B^{\mathsf{bvole}})$.

We prove that the output of $\mathcal{S}$ is computationally indistinguishable from the real view under the LWE assumption using a hybrid argument.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of pe_B computed exactly according to Encode_B in Figure 6.2.

- *Hybrid* $\mathcal{H}_1$. In this hybrid, we replace $\mathsf{pe}_B^{\mathsf{bvole}}$ in pe_B with the encoding of zero. That is, $\mathsf{pe}_B^{\mathsf{bvole}}$ is generated according to $\mathsf{BNIVOLE.Encode}_B(\mathsf{crs}_{\mathsf{bvole}}, 0)$.

  Claim. $\mathcal{H}_0 \approx_c \mathcal{H}_1$ *under the LWE assumption.*

  Proof. Computational indistinguishability between $\mathcal{H}_0$ and $\mathcal{H}_1$ follows immediately from the security of $\mathsf{BNIVOLE}$, which is realized under LWE with a superpolynomial modulus-to-noise ratio (cf. Theorem 6.4.1). □

- *Hybrid* $\mathcal{H}_2$. In this hybrid, we make $\mathbf{s}$ uniformly random in $\mathbb{Z}_q^n$ and no longer set $\mathbf{s}[1] = 1$.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_2$ *under the LWE assumption.*

  Proof. Computational indistinguishability between $\mathcal{H}_1$ and $\mathcal{H}_2$ follows immediately from the leakage-resilience of the LWE assumption [GKPV10]; in particular, the LWE assumption is tolerant to any constant number of coordinates of the secret being set to a fixed public value. □

- *Hybrid* $\mathcal{H}_3$. In this hybrid, we replace $\mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, \mathbf{t}_1, \ldots, \mathbf{t}_\beta$ in pe_B (which all depend on the secret $\mathbf{s}$) with uniformly random values sampled independently from $\mathbb{Z}_q^k$.

  Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_3$ *under the LWE assumption.*

Proof. Suppose, towards contradiction, that $\mathcal{H}_2 \not\approx_c \mathcal{H}_3$, then there exists an efficient distinguisher $\mathcal{A}$ distinguishing between $\mathcal{H}_2$ and $\mathcal{H}_3$ with non-negligible advantage.

Let $k' := \alpha + \beta + \gamma$. We construct an efficient distinguisher $\mathcal{B}$ for the LWE problem that receives as input $k' = k'(\lambda) \in \mathsf{poly}(\lambda)$ LWE challenge samples $(\mathbf{R}_i, \mathbf{r}_i)_{i=1}^{k'}$, where $(\mathbf{r}_1, \ldots, \mathbf{r}_{k'})$ are either all uniformly random and independent or distributed as $\mathbf{s}^\top \mathbf{R}_i + \mathbf{e}_i$, for all $i \in [k']$.

$\mathcal{B}$ proceeds as follows:

1. Sample $\mathsf{sk} \xleftarrow{R} \mathsf{FHE.KeyGen}(1^\lambda)$ and computes ct_y exactly as in Figure 6.2. Let $\mathsf{sk} := (\mathsf{sk}_1, \ldots, \mathsf{sk}_\beta)$ and $\mathsf{ct}_y = c_1 \| c_2 \| \cdots \| c_\alpha \in \{0,1\}^\alpha$.
2. For each $i \in [\alpha]$, set $\mathbf{u}_i := \mathbf{r}_i$ and set $\mathbf{A}_i := \mathbf{R}_i - c_i \mathbf{G}$.
3. For each $i \in [\beta]$, set $\mathbf{v}_i := \mathbf{r}_{\alpha+i}$ and set $\mathbf{B}_i := \mathbf{R}_{\alpha+i} - \mathsf{sk}_i \mathbf{G}$.
4. For each $i \in [\gamma]$, set $\mathbf{t}_i := \mathbf{r}_{\alpha+\beta+i}$ and $\mathbf{C}_i := \mathbf{R}_{\alpha+\beta+i}$.
5. Set $\mathsf{crs}_{\mathsf{aux}} := (\mathbf{A}_1, \ldots, \mathbf{A}_\alpha, \mathbf{B}_1, \ldots, \mathbf{B}_\beta)$, $\mathsf{crs}_{\mathsf{rnd}} := (\mathbf{C}_1, \ldots, \mathbf{C}_\gamma)$.
6. Set $\mathsf{crs} := (\mathsf{crs}_{\mathsf{aux}}, \mathsf{crs}_{\mathsf{rnd}}, \mathsf{crs}_{\mathsf{bvole}}, K)$, where $\mathsf{crs}_{\mathsf{rnd}}$, $\mathsf{crs}_{\mathsf{bvole}}$ and K are sampled as in Figure 6.2.
7. Set $\mathsf{pe}_B := (\mathsf{ct}_y, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, \mathbf{t}_1, \ldots, \mathbf{t}_\gamma, \mathsf{pe}_B^{\mathsf{bvole}})$, where $\mathsf{pe}_B^{\mathsf{bvole}}$ is distributed exactly as in $\mathcal{H}_2$.
8. Output as $\mathcal{A}(\mathsf{crs}, \mathsf{pe}_B)$ does.

We now argue that $\mathcal{B}$ wins with the same advantage as $\mathcal{A}$. Observe that when $\mathcal{B}$ receives LWE samples $(\mathbf{r}_1, \ldots, \mathbf{r}_{k'})$, then we have that:

- $\mathbf{u}_i = \mathbf{r}_i = \mathbf{s}^\top \mathbf{R}_i + \mathbf{e}_i = \mathbf{s}^\top (\mathbf{A}_i + c_i \mathbf{G}) + \mathbf{e}_i$, for all $i \in [\alpha]$.
- $\mathbf{v}_i = \mathbf{r}_{\alpha+i} = \mathbf{s}^\top \mathbf{R}_{\alpha+i} + \mathbf{e}_{\alpha+i} = \mathbf{s}^\top (\mathbf{B}_i + \mathsf{sk}_i \mathbf{G}) + \mathbf{e}_{\alpha+i}$, for all $i \in [\beta]$.
- $\mathbf{t}_i = \mathbf{r}_{\alpha+\beta+i} = \mathbf{s}^\top \mathbf{R}_{\alpha+\beta+i} + \mathbf{e}_{\alpha+\beta+i} = \mathbf{s}^\top \mathbf{C}_i + \mathbf{e}_{\alpha+\beta+i}$, for all $i \in [\gamma]$.

and so $\mathcal{A}$ receives pe_B distributed identically to hybrid $\mathcal{H}_2$.

In contrast, if $\mathcal{B}$ receives uniformly random samples $(\mathbf{r}_1, \ldots, \mathbf{r}_{k'})$, then all $\mathbf{u}_i$, $\mathbf{v}_i$, $\mathbf{t}_i$ are uniformly random, which is distributed identically to hybrid $\mathcal{H}_3$. Therefore, $\mathcal{B}$ succeeds with the same advantage as $\mathcal{A}$, contradicting the LWE assumption. □

- *Hybrid $\mathcal{H}_4$.* In this hybrid, we replace ct_y with an encryption of zero.

  Claim. *$\mathcal{H}_3 \approx_c \mathcal{H}_4$ under the LWE assumption.*

  Proof. Computational indistinguishability between $\mathcal{H}_4$ and $\mathcal{H}_3$ follows immediately from the semantic security of FHE, and hence from LWE. □

At this point, it suffices to note that hybrid $\mathcal{H}_4$ is distributed identically to the output of $\mathcal{S}$, which concludes the proof of security for Bob.

This concludes the proof of Theorem 6.5.2. ■

Construction without output-succinctness. We remark that the proof of Theorem 6.5.2 does not make use of the BNIVOLE hiding property when arguing security for Alice; only when arguing security for Bob. This is because we already have statistical hiding for Alice's input thanks to the function-hiding transformation from Figure 6.1. Alternatively, we could avoid using the statistical-hiding transformation and just rely on BNIVOLE security to hide Alice's matrices as output by EvalPK; however, this would make the scheme less modular in the following sense. If the output-succinctness property (Definition 6.4.2) is *not* required, the construction from Figure 6.2 can simply have Alice's public encoding consist of $(\mathbf{A}_{C_i})_{i=1}^m$ such that Bob can locally compute $(\mathbf{s}^\top \mathbf{A}_{C_i})_{i=1}^m$. Specifically, we can simply remove the use of BNIVOLE in Figure 6.2, without changing the proof of security.

6.6 Construction from Indistinguishability Obfuscation

In this section, we construct SMS from $i\mathcal{O}$ in conjunction with other assumptions. Our construction supports the computation of *batch* functions over a large batch of short inputs provided by Alice and a short input provided by Bob. Concretely, we assume that Alice has a large batch of inputs $X = (x_1, \ldots, x_L)$ and Bob has a small input y, such that $|x_i| \approx |y|$. Using our construction, Alice and Bob can compute $f(x_i, y)$ for all $i \in [L]$, and for any function $f \in P/\mathsf{poly}$ determined adaptively *at decoding time*. We obtain input-output succinctness with respect to the batch size L.

Compared to our LWE-based construction from Section 6.5, our $i\mathcal{O}$-based construction supports all circuits in P/poly and is function adaptive (cf. Definition 6.4.4), allowing Alice and Bob to agree on the function they wish to compute over the entire batch or even for each individual entry in the batch.

6.6.1 Preliminaries

In this section, we provide the necessary preliminaries related to the $i\mathcal{O}$-based construction.

Indistinguishability obfuscation. Indistinguishability obfuscation ($i\mathcal{O}$) [BGI+01] satisfies the property that the obfuscation of two "functionally equivalent" circuits C_0 and C_1 are computationally indistinguishable.

Definition 6.6.1 (Indistinguishability Obfuscation [BGI+01])**.** *An efficient uniform algorithm* iO *is said to be an indistinguishability obfuscator for a class of circuits* $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *if the following properties hold:*

Correctness. *For all* $\lambda \in \mathbb{N}$*, for all* $C \in \mathcal{C}_\lambda$*, and for every input* x *to* C*:*

$$\Pr\Big[\ \widetilde{C} \leftarrow \mathsf{iO}(1^\lambda, C) \quad : \quad \widetilde{C}(x) = C(x)\ \Big] = 1,$$

where the probability is over the randomness of iO*.*

Security. *For all efficient distinguishers* $\mathcal{D}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, for all pairs of circuits* $C_0, C_1 \in \mathcal{C}_\lambda$ *such that* $C_0(x) = C_1(x)$ *on all inputs* x*,*

$$\Big|\Pr\Big[\ \mathcal{D}(\widetilde{C}_0) \quad : \quad \widetilde{C}_0 \leftarrow \mathsf{iO}(1^\lambda, C_0)\ \Big] - \Pr\Big[\ \mathcal{D}(\widetilde{C}_1) \quad : \quad \widetilde{C}_1 \leftarrow \mathsf{iO}(1^\lambda, C_1)\ \Big]\Big| \leq \mathsf{negl}(\lambda),$$

where the probability is over the randomness of iO *and* $\mathcal{D}$.

Somewhere statistically binding hashing. Here, we recall the definition of somewhere statistically binding (SSB) hashing [HW15].

Definition 6.6.2 (Somewhere Statistically Binding Hashing [HW15])**.** *Let* λ *be a security parameter,* s *be a block length and* $\Sigma = \{0,1\}^s$ *be the block alphabet, and* $m = m(s) \in \mathsf{poly}(\lambda)$ *be the output length of the hash. Let* $p = p(s) \in \mathsf{poly}(\lambda)$ *be the opening size. A somewhere statistically binding (SSB) hash with local opening consists of four efficient algorithms* $\mathsf{SSB} = (\mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ *with the following syntax:*

- $\mathsf{KeyGen}(1^\lambda, 1^s, L, i) \to \mathsf{hk}$*. The randomized key generation algorithm takes as input the security parameter* λ*, a block length* s*, an input length* $L \leq 2^\lambda$*, and an index* $i \in [L]$*. It outputs a public hashing key* hk*.*
- $\mathsf{Hash}(\mathsf{hk}, X) \to c_X$*. The deterministic hashing algorithm takes as input the hash key* hk *and an input* $X = (x_1, \ldots, x_L) \in \Sigma^L$*. It outputs the hash value* $c_X \in \{0,1\}^m$*.*
- $\mathsf{Open}(\mathsf{hk}, x_j, j) \to \pi$*. The (possibly randomized) opening algorithm takes as input the hash key* hk*, a value* $x_j \in \Sigma$*, and an index* $j \in [L]$*. It creates an opening* $\pi \in \{0,1\}^p$*.*
- $\mathsf{Verify}(\mathsf{hk}, c_X, u, j, \pi) \to 0/1$*. The deterministic verification algorithm takes as input the hash key* hk*, a hash output* $c_X \in \{0,1\}^m$*, a value* $u \in \Sigma$*, an index* $j \in [L]$*, and an opening* $\pi \in \{0,1\}^p$*. It outputs a* 0 *(reject) or* 1 *(accept).*

The above functionality must satisfy the following properties:

Correctness. *For any block length* s*, any input length* L*, any pair of indices* $i, j \in [L]$*, and all* $X = (x_1, \ldots, x_L) \in \Sigma^L$*, it holds that:*

$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, c_X, x_j, j, \pi) = 1 \quad : \quad \begin{array}{r} \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^s, L, i) \\ c_X := \mathsf{Hash}(\mathsf{hk}, X) \\ \pi \leftarrow \mathsf{Open}(\mathsf{hk}, x_j, j) \end{array}\right] = 1.$$

Index Hiding. *For all block lengths* s*, all input lengths* $L \leq 2^\lambda$*, and all pairs of indices* $i_0, i_1 \in [L]$*,*

$$\left\{ \mathsf{hk} \;\middle|\; \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^s, L, i_0) \right\} \approx_c \left\{ \mathsf{hk} \;\middle|\; \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^s, L, i_1) \right\}.$$

Somewhere Statistically Binding. *The hash key* hk *is said to be* statistically binding with respect to the opening *for an index* $i \in [L]$ *if there do not exist any values* c_X*,* $x_i \neq x_i'$*, and openings* π, π' *such that* $\mathsf{Verify}(\mathsf{hk}, c_X, x_i, i, \pi) = \mathsf{Verify}(\mathsf{hk}, c_X, x_i', i, \pi') = 1$*. Formally, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all block lengths* s*, all input lengths* L*, and*

any index $i \in [L]$,

$$\Pr\left[\begin{array}{c} \exists x_i, x_i' \in \Sigma \text{ such that } x_i \neq x_i' \\ \wedge \\ \mathsf{Verify}(\mathsf{hk}, c_X, x_i, i, \pi) = 1 \quad : \quad \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^s, L, i) \\ \wedge \\ \mathsf{Verify}(\mathsf{hk}, c_X, x_i', i, \pi') = 1 \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

The hash function is said to be perfectly binding with respect to the opening *if the above probability is zero.*

Remark 42 (On perfect binding)**.** *Any perfectly-correct, rate-1 oblivious transfer (OT) scheme implies an SSB hash function with perfect binding via the transformation given in the work of Kalai et al. [KLVW23]. Moreover, the hash key is guaranteed to be indistinguishable from random if the OT receiver's message in the OT scheme is pseudorandom (which is indeed the case for constructions based on QR/DCR). We note that the construction of Hubáček and Wichs [HW15] when instantiated with a perfectly-correct FHE scheme (based on LWE) [BGV12] also gives an SSB hash construction with perfect binding with respect to the opening.*

Theorem 6.6.1 ([HW15, KLVW23])**.** *Under either the QR, DCR, or the LWE assumption, there exists a construction of SSB hashing that has perfect binding with respect to the opening.*

Puncturable PRFs. We recall the notion of puncturable PRFs (PPRFs).

Definition 6.6.3 (Puncturable Pseudorandom Function [BW13, KPTZ13, BGI14])**.** *Let* λ *be a security parameter,* $\mathcal{X} = \mathcal{X}_\lambda$ *be the domain, and* $\mathcal{Y}$ *be the range. A puncturable pseudorandom function (PPRF) consists of three efficient algorithms* $\mathsf{PPRF} = (\mathsf{KeyGen}, \mathsf{Puncture}, \mathsf{Eval})$ *with the following syntax:*

- $\mathsf{KeyGen}(1^\lambda) \to K$*. The randomized key generation algorithm takes as input the security parameter* λ *and outputs a master key* K*.*
- $\mathsf{Puncture}(K, x^*) \to K^*$*. The randomized puncture algorithm takes as input a master key* K *and an input* $x^* \in \mathcal{X}$*. It outputs a punctured key* K^**.*
- $\mathsf{Eval}(K, x) \to y$*. The deterministic evaluation algorithm takes as input a key* K *(which may be the punctured key) and an input* $x \in \mathcal{X}$*. It outputs a value* $y \in \mathcal{Y}$*.*

The above functionality must satisfy the following properties:

Correctness. *For all* $\lambda \in \mathbb{N}$*, every choice of punctured input* $x^* \in \mathcal{X}$*, and every* $x \in \mathcal{X} \setminus \{x^*\}$*, it holds that:*

$$\Pr\left[\mathsf{PPRF.Eval}(K, x) = \mathsf{PPRF.Eval}(K^*, x) \quad : \quad \begin{array}{r} K \leftarrow \mathsf{KeyGen}(1^\lambda) \\ K^* \leftarrow \mathsf{Puncture}(K, x^*) \end{array}\right] = 1.$$

Security. *A puncturable PRF is a said to be* selective-puncturing secure *if for all efficient adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following security experiment $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},b}(\lambda)$ is negligible in λ. Here, b denotes the challenge bit.*

1. *The challenger runs $\mathcal{A}(1^\lambda)$.*
2. *The adversary $\mathcal{A}$ sends a challenge $x^* \in \mathcal{X}$ to the challenger.*
3. *The challenger samples a master key $K \leftarrow \mathsf{KeyGen}(1^\lambda)$, and computes the punctured key $K^* \leftarrow \mathsf{Puncture}(K, x^*)$. Then, the challenges does the following:*
   - *If $b = 0$, compute $y := \mathsf{PPRF.Eval}(K, x^*)$ and respond with (K^*, y).*
   - *If $b = 1$, sample $y \xleftarrow{R} \mathcal{Y}$ and respond with (K^*, y).*
4. *$\mathcal{A}$ outputs a guess b', which is the output of the experiment.*

$\mathcal{A}$ wins if $b' = b$, and its advantage $\mathsf{Adv}^{\mathsf{sec}}_{\mathcal{A}}(\lambda)$ is defined as

$$\mathsf{Adv}^{\mathsf{sec}}_{\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},1}(\lambda) = 1]\right|,$$

where the probability is over the randomness of $\mathcal{A}$ and KeyGen *and* Puncture*.*

Remark 43 (t-puncturable PRF)**.** *We will also use the notion of a t-puncturable PRF [BCG+19a], which can be realized in a black-box way using a 1-puncturable PPRF. A t-puncturable PRF is defined exactly as in Definition 6.6.3, except that the adversary is given a key punctured on t inputs and obtains t (real-or-random) challenges from the challenger. A simple and black-box construction of a t-puncturable PRF involves running t independent instances of a 1-puncturable PRF and defining the output to be the bit-wise XOR of all instances [BCG+19a].*

Theorem 6.6.2 (Existence of PPRFs [BW13, KPTZ13, BGI14])**.** *Assuming the existence of sub-exponentially-secure one-way functions, there exits a sub-exponentially-secure puncturable PRF (resp. t-puncturable PRF) with selective puncturing security.*

Commitment scheme. We require any commitment scheme with perfect binding. Such commitment schemes are known from injective one-way functions [BOV03].

Definition 6.6.4 (Perfectly-Binding Commitment Scheme)**.** *Let λ be a security parameter and $\mathcal{M}$ be a commitment message space. A perfectly-binding commitment scheme consists of an efficient algorithm $\mathsf{Commit}(x; r) \to \widehat{x}$ that takes as input a message $x \in \mathcal{M}$ and randomness $r \in \{0,1\}^\lambda$, and outputs a commitment $\widehat{x}$.* Commit *must satisfy the following properties:*

Perfect Binding. *For all $x, x' \in \mathcal{M}$ and every $r, r' \in \{0,1\}^\lambda$,*

$$\Pr\Big[\; x \neq x' \quad : \quad \mathsf{Commit}(x; r) = \mathsf{Commit}(x'; r') \;\Big] = 0.$$

Computational Hiding. *For all efficient adversaries $\mathcal{A}$, there exists a negligible function*

$\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\mathcal{A}(\widehat{x}_b, \mathsf{st}) \quad : \quad \begin{array}{r}(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda) \\ r \xleftarrow{R} \{0,1\}^\lambda \\ b \xleftarrow{R} \{0,1\} \\ \widehat{x}_b := \mathsf{Commit}(x_b; r)\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

6.6.2 Construction

We present our construction in Figure 6.5. Our construction follows closely the ideas presented in Section 6.2.2 and makes use of an SSB hash function with perfect binding and $i\mathcal{O}$. Because the SSB hashing is performed over the set of *commitments* to Alice's inputs, we set the SSB hash input block length (denoted s) to be $\mathsf{poly}(\lambda, l)$ to accommodate the size of the commitment.

Remark 44 (On the use of the puncturable PRF)**.** *We note that the construction itself does not make use of the puncturable PRF, and instead treats it as a regular PRF. The requirement for a puncturable PRF appears only in the proof of security (see Section 6.12.1).*

The obfuscated program. Bob's obfuscated program is defined in Program 1 and is parameterized by a universal circuit $\mathcal{U} \in P/\mathsf{poly}$ that takes as input the tuple (f, x_i, y), consisting of a function description f, an input x_i, and an input y. The circuit $\mathcal{U}$ outputs $f(x_i, y)$. We assume that the size of $\mathcal{U}$ is polynomial in the security parameter λ, Alice's input length l and Bob's input length ℓ.

6.6.3 Setting the parameters

Here, we explain how we need to set the parameters for the underlying primitives to achieve security for our full construction by complexity leveraging.

Let λ be the security parameter for the $i\mathcal{O}$-based SMS construction. We let the security parameter of the SSB hash, denoted λ_{ssb}, be the same as λ. Let $\lambda_{\mathsf{io}} = q(\lambda)$ and $\lambda_{\mathsf{pprf}} = q(\lambda)$ be polynomial in the security parameter λ, for some polynomial $q(\lambda) \in \mathsf{poly}(\lambda)$, which we will set later.

Next, we let ϵ_{io}, ϵ_{pprf}, and ϵ_{ssb}, denote the advantage of any efficient distinguisher $\mathcal{D}$ in the $i\mathcal{O}$ security game (cf. Definition 6.6.1), the PPRF security game (cf. Definition 6.6.3), and the SSB index-hiding game (cf. Definition 6.6.2), respectively. Furthermore, let the PPRF domain length be $n = \mathsf{poly}(\lambda, \log L)$, where L is Alice's batch length. Define $\epsilon \geq \max(\epsilon_{\mathsf{io}}, \epsilon_{\mathsf{pprf}})$. Then, we have that $1/2^{q^\epsilon}$ bounds the advantage of the $\mathcal{D}$ in the $i\mathcal{O}$ game and the PPRF game.

Now, we need to set q such that $2^n/2^{q^\epsilon}$ is negligible in λ. This can be achieved by choosing q such that $q^\epsilon \geq O(n + \lambda)$, which remains polynomial in the security parameter.

6.6.4 Security analysis

In this section, we analyze the correctness and security of our $i\mathcal{O}$-based construction of SMS from Figure 6.5. We prove the following theorem:

SMS from $i\mathcal{O}$

Public Parameters. We let L denote the size of Alice's batch of inputs $X = (x_1, \ldots, x_L)$, l denote the size of each input in the batch, such that $x_i \in \{0,1\}^l$ for all $i \in [L]$, ℓ denote the size of Bob's input y, Commit be a commitment scheme with perfect binding as defined in Definition 6.6.4, $\mathsf{SSB} = (\mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ be an SSB hash function (cf. Definition 6.6.2), and $\mathsf{PPRF} = (\mathsf{KeyGen}, \mathsf{Puncture}, \mathsf{Eval})$ be a puncturable PRF.

$\mathsf{SMS.Setup}(1^\lambda)$

1 : $s := \mathsf{poly}(\lambda, l)$
2 : $\mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^s, L, 0)$
3 : **return** $\mathsf{crs} := \mathsf{hk}$

$\mathsf{SMS.Encode}_A(\mathsf{crs}, 1^{|\mathcal{F}|}, X)$

1 : **parse** $\mathsf{crs} = \mathsf{hk}$
2 : **parse** $X = (x_1, \ldots, x_L)$
3 : **foreach** $i \in [L]$:
4 : $\quad r_i \xleftarrow{R} \{0,1\}^\lambda$
5 : $\widehat{x}_i := \mathsf{Commit}(x_i; r_i)$
6 : $\widehat{X} := (\widehat{x}_1, \ldots, \widehat{x}_L)$
7 : $c_{\widehat{X}} := \mathsf{SSB.Hash}(\mathsf{hk}, \widehat{X})$
8 : $\mathsf{pe}_A := c_{\widehat{X}}$
9 : $\mathsf{st}_A := (X, \widehat{X}, c_{\widehat{X}}, r_1, \ldots, r_L)$
10 : **return** $(\mathsf{pe}_A, \mathsf{st}_A)$

$\mathsf{SMS.Encode}_B(\mathsf{crs}, 1^{|\mathcal{F}|}, y)$

1 : **parse** $\mathsf{crs} = \mathsf{hk}$
2 : $K \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda)$
3 : $\widetilde{P} \leftarrow \mathsf{iO}(1^\lambda, P)$, where P has (hk, K, y) hardcoded in it.
4 : $\mathsf{pe}_B := \widetilde{P}$
5 : $\mathsf{st}_B := K$
6 : **return** $(\mathsf{pe}_B, \mathsf{st}_B)$

$\mathsf{SMS.Decode}_A(\mathsf{crs}, f, \mathsf{pe}_B, \mathsf{st}_A)$

1 : **parse** $\mathsf{crs} = \mathsf{hk}$ **and** $\mathsf{pe}_B = \widetilde{P}$
2 : **parse** $\mathsf{st}_A = (X, \widehat{X}, c_{\widehat{X}}, r_1, \ldots, r_L)$
3 : **parse** $X = (x_1, \ldots, x_L)$ **and** $\widehat{X} = (\widehat{x}_1, \ldots, \widehat{x}_L)$
4 : **foreach** $i \in [L]$:
5 : $\quad \pi_i := \mathsf{Open}(\mathsf{hk}, \widehat{x}_i, i)$
6 : $\quad z_A^{(i)} := \widetilde{P}(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$
7 : **return** $\mathbf{z}_A := (z_A^{(1)}, \ldots, z_A^{(L)})$

$\mathsf{SMS.Decode}_B(\mathsf{crs}, f, \mathsf{pe}_A, \mathsf{st}_B)$

1 : **parse** $\mathsf{crs} = \mathsf{hk}$ **and** $\mathsf{pe}_A = c_{\widehat{X}}$ **and** $\mathsf{st}_B = K$
2 : **foreach** $i \in [L]$:
3 : $\quad z_B^{(i)} := \mathsf{PPRF.Eval}(K, c_{\widehat{X}} \| f \| i)$
4 : **return** $\mathbf{z}_B := (z_B^{(1)}, \ldots, z_B^{(L)})$

Figure 6.5: Simultaneous-Message and Succinct Secure Computation from $i\mathcal{O}$.

Program 1: (Parameterized by a universal circuit $\mathcal{U}$)

Hardcoded: (hk, K, y).
Input: $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$.
Procedure:

1: **if** $\widehat{x}_i = \mathsf{Commit}(x_i; r_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}_i, i, \pi_i) = 1$ **then**
2: $\quad R_i := \mathsf{PPRF.Eval}(K, c_{\widehat{X}} \| f \| i)$
3: $\quad d := \mathcal{U}(f, x_i, y)$
4: $\quad$ **return** $d \oplus R_i$
5: **else return** $\perp$

Figure 6.6: Program obfuscated by Bob in Figure 6.5.

Theorem 6.6.3. *Assuming the existence of sub-exponentially-secure indistinguishability obfuscation and the existence of sub-exponentially-secure one-way functions, in addition to the existence of somewhere statistically binding hash functions (with perfect binding) and injective one-way function, Figure 6.5 is a $O(\log \log L / \log L)$-succinct SMS scheme supporting all batched function families in P/poly, where L is the batch size.*

In Proposition 6.6.1 we prove the correctness and succinctness of the construction. In Proposition 6.6.2, we prove security for Alice. Then, in Proposition 6.6.2, we prove security for Bob.

Proposition 6.6.1. *Figure 6.5 satisfies the correctness properties of Definition 6.4.1 and achieves $O(\log \log L / \log L)$-batch-succinctness (cf. Definition 6.4.6).*

Proof. Consider Program 1. For each $i \in [L]$, notice that if π_i is a valid opening for $\widehat{x}_i$ with respect to the SSB hash value $c_{\widehat{X}}$ and index i, and (x_i, r_i) is a valid decommitment for $\widehat{x}_i$, then the output of the obfuscated program is $z_A^{(i)} := f(x_i, y) \oplus \mathsf{PPRF.Eval}(K, c_{\widehat{X}} \| f \| i)$. Bob's output, on the other hand, is $z_B^{(i)} := \mathsf{PPRF.Eval}(K, c_{\widehat{X}} \| f \| i)$. The parties thus obtain additive shares of $f(x_i, y)$, by the correctness of $i\mathcal{O}$. Furthermore, because this holds for all $i \in [L]$, the output of Decode_σ is an additive share of the function applied to components of Alice's batch of inputs X, as required.

Finally, we note that the size of the obfuscated circuit depends polynomially on the input and output lengths ℓ, l, m, but only depends *logarithmically* on the batch size L (the dependence on L comes from domain of the PRF needing to be large enough to accommodate the index i of the batch input). Hence, we have that the size of the program obfuscated by Bob is of size $\mathsf{poly}(\lambda, \ell, l, m, \log L)$, which gives $\epsilon = O(\log \log L / \log L)$ batch-succinctness since $L^{O(\log \log L / \log L)} = \mathsf{polylog}(L)$. ■

Security for Alice. Proving security for Alice is relatively straightforward and comes down to the computational-hiding property of the commitment scheme. We prove the following proposition.

Proposition 6.6.2. *Figure 6.5 satisfies the security property of Definition 6.4.1 for Alice assuming the commitment scheme* Commit *is computationally hiding.*

Proof. Consider the following efficient simulator $\mathcal{S}$ for pe_A.

$\mathcal{S}$: On input crs,

- Parse $\mathsf{crs} = \mathsf{hk}$.
- Sample $r_1, \ldots, r_L \xleftarrow{R} \{0,1\}^\lambda$.
- Set $\widehat{x}_i := \mathsf{Commit}(0; r_i)$, for all $i \in [L]$.
- $c_{\widehat{X}} := \mathsf{SSB.Hash}(\mathsf{hk}, (\widehat{x}_1, \ldots, \widehat{x}_L))$.
- Output $\mathsf{pe}_A := c_{\widehat{X}}$.

We now argue that the output of $\mathcal{S}$ is computationally indistinguishable to the output of Encode_A. Suppose, towards contradiction, that there exists an efficient distinguisher $\mathcal{A}$ that distinguishes, with non-negligible advantage, between the view produced by $\mathcal{S}$ and pe_A produced by Encode_A in Figure 6.5. Notice that the only difference in the simulated view compared to Encode_A is that each $\widehat{x}_i$ is a commitment to zero. As such, by a straightforward hybrid argument, $\mathcal{A}$ breaks the computational hiding of Commit, which raises a contradiction. This concludes the proof. ■

Security for Bob. Proving security for Bob is more involved and requires carefully removing the presence of Bob's input y from the obfuscated program via a sequence of hybrids. More concretely, our proof strategy involves iterating over all possible inputs $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$ to Bob's program and carefully "puncturing" the program at the j-th canonical input to the PRF by programming the output to be a uniformly random string that is independent of the input y. In the process, we make use of the SSB hash to guarantee functional equivalence and index hiding, which then allows us to invoke $i\mathcal{O}$ security.

This overall strategy requires us to consider an *exponential* (in the SSB hash security parameter) number of hybrids, which requires complexity leveraging and assuming sub-exponential security of the underlying primitives (namely, sub-exponentially secure $i\mathcal{O}$ and one-way functions). See Section 6.6.3 for how we set parameters to obtain sub-exponential security.

Proposition 6.6.3. *Figure 6.5 satisfies the security property of Definition 6.4.1 for Bob assuming the security of the SSB hash (cf. Definition 6.6.2), the existence of injective one-way functions, the existence of sub-exponentially secure indistinguishability obfuscation (cf. Definition 6.6.1), and the existence of sub-exponentially secure one-way functions.*

Proof. Deferred to Section 6.12.1. ■

6.7 Optimizations

In this section, we discuss optimizations that we can make to our LWE-based and $i\mathcal{O}$-based constructions that further push the communication and computational efficiency.

6.7.1 Unbounded computations

We show that our LWE-based construction can be upgraded to support *unbounded* computations assuming the circular security of LWE. In particular, the recent result of Hsieh, Lin, and

Luo [HLL23] show how to construct algorithms EvalPK and EvalCT supporting unbounded computations while satisfying the properties of Lemma 6.5.1. In particular, we can use the following lemma (where the highlighted parts indicate the difference with Lemma 6.5.1).

Lemma 6.7.1 (Adapted from [HLL23, Theorem 13] and [GVW15, Lemma 3.2])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. Under the circular security of the $\mathsf{LWE}_{n,k,q,\tau}$ assumption (as defined in [HLL23, Assumption 1]) with $k := n\lceil \log q \rceil$, there exist algorithms $(\mathsf{EvalPK}, \mathsf{EvalCT})$ satisfying Definition 6.5.2 for all integers $\alpha = \alpha(\lambda), \beta = \beta(\lambda)$ that are polynomial in the security parameter, such that for all common random strings of the form:*

$$\mathsf{crs} := (\mathbf{A}_1, \ldots, \mathbf{A}_\alpha, \mathbf{B}_1, \ldots, \mathbf{B}_\beta) \in (\mathbb{Z}_q^{n \times k})^{\alpha+\beta},$$

for all $\alpha + \beta$ vectors $\mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta \in \mathbb{Z}_q^k$, all $\mathbf{s} \in \mathbb{Z}_q^n$, all $(x, \mathbf{y}) \in \{0,1\}^\alpha \times \mathbb{Z}_q^\beta$, and all arithmetic circuits $C \colon \{0,1\}^\alpha \to \mathbb{Z}_q^\beta$ of polynomial size and unbounded depth, if it holds that:

$$\forall i \in [\alpha],\ \mathbf{u}_i = \mathbf{s}^\top(\mathbf{A}_i + x_i \cdot \mathbf{G}) + \mathbf{e}_i^\top \text{ and } ||\mathbf{e}_i||_\infty \leq B,$$
$$\forall i \in [\beta],\ \mathbf{v}_i = \mathbf{s}^\top(\mathbf{B}_i + \mathbf{y}[i] \cdot \mathbf{G}) + \mathbf{e}_i^\top \text{ and } ||\mathbf{e}_i||_\infty \leq B,$$

then it also holds that for $\mathbf{w}_{\mathsf{IP} \circ C} := \mathsf{EvalCT}(\mathsf{crs}, \mathbf{u}_1, \ldots, \mathbf{u}_\alpha, \mathbf{v}_1, \ldots, \mathbf{v}_\beta, C, x)$,

$$\mathbf{w}_{\mathsf{IP} \circ C} = \mathbf{s}^\top(\mathbf{A}_{\mathsf{IP} \circ C} + \langle C(x), \mathbf{y} \rangle \cdot \mathbf{G}) + \mathbf{e}^\top \text{ with } ||\mathbf{e}||_\infty \leq B,$$

where $\mathbf{A}_{\mathsf{IP} \circ C} := \mathsf{EvalPK}(\mathsf{crs}, C)$ and $\mathbf{G}$ is the gadget matrix from Definition 6.5.1.

Proof (sketch). The lemma follows from [HLL23, Theorem 13] coupled with [GVW15, Lemma 3.2]. [HLL23, Theorem 13] proves the existence of EvalPK and EvalCT as defined in Definition 6.5.2 (without the extension to the class $\mathsf{IP} \circ C$) while [GVW15, Lemma 3.2] provides a generic extension to the class $\mathsf{IP} \circ C$. In particular, the proof of [GVW15, Lemma 3.2] follows directly from the properties satisfied by these two algorithms and therefore also applies to the (unbounded) variants. ■

As an immediate corollary of Lemma 6.7.1, we have that the LWE-based construction of SMS works for any *unbounded* depth function (with a polynomially-sized circuit representation) assuming the circular security of LWE. In particular, we note that Lemma 6.5.2 (the function-hiding transformation), introduces a fixed additive factor overhead to the error and is not impacted by the underlying instantiation of EvalPK and EvalCT. As such, we have all the ingredients to instantiate Figure 6.2 for unbounded-depth computations. We obtain the following theorem:

Theorem 6.7.1. *Let λ be a security parameter. Assume that the circular security of the LWE with a superpolynomial modulus-to-noise ratio. Then, Figure 6.2 is an SMS scheme satisfying Definition 6.4.1 for all functions that can be represented by polynomial-size circuits. Furthermore, the scheme achieves full input succinctness and (2/3)-output succinctness.*

6.7.2 Minimizing communication from Bob to Charlie

We observe that in our $i\mathcal{O}$-based construction (cf. Figure 6.5), Bob's output consists solely of a pseudorandom string computed by the PRF evaluated on input $(c_{\widehat{X}} \| f \| i)$, for i ranging from 1 to L.

At first glance, it may appear that Bob can simply send the PRF key K to Charlie, making the communication overhead from Bob to Charlie optimal (similarly to the communication overhead in the naive FHE-based example described in Section 6.1). However, this idea is obviously insecure in the case where Charlie colludes with Alice, since it would allow Alice to recover the PRF value on *all* inputs to the obfuscated program and perform a resetting attack.

Nonetheless, we show that we can easily tweak this idea by resorting to a *constrained* PRF (CPRF), which restricts the domain of the PRF that Charlie is allowed to evaluate. Specifically, a CPRF generalizes the notion of a puncturable PRF and can be constrained on an arbitrary predicate (not just the puncturing "index" predicate). We recall the formal definition of CPRFs from Chapter 2.

Our idea is to have Bob constrain the PRF on all inputs outside of the set

$$S = \big\{ (c_{\widehat{X}} \| f) \| i \mid i \in [L] \big\}.$$

In particular, given a constrained key constrained to the set S, only L values of the PRF can be evaluated, and these values correspond exactly to the output of Bob. However, because the predicate has a succinct description of size $O(\log L)$, the communication from Bob to Charlie is only polylogarithmic in the batch size L (ignoring polynomial factors in the security parameter).

Connection to Hubáček–Wichs. We additionally observe that when the communication from Bob to Charlie is made succinct, the resulting SMS scheme immediately implies a two-round secure computation protocol that is succinct in the output length. This shares a close connection with the protocol of Hubáček and Wichs [HW15] (and thus inherits the same impossibility results discussed therein related to secure computation protocols where the total communication is succinct in the output length). Specifically, in an SMS scheme, we can assume that Alice and Charlie are colluding. Therefore, Bob can simply send the succinct output message to Alice, who can then locally recover the output. This protocol is two rounds and is similar in flavor to the "multi-decryption" protocol of Hubáček and Wichs [HW15, Section 3.2] (indeed, our construction generalizes their notion to any batch computation).

Programming the output. Interestingly, to prove our optimized construction secure, we need to resort to the same techniques used by Hubáček and Wichs [HW15]. That is, in order to simulate the view of Alice, the simulator needs to program the output (which is now much longer than the view of Bob) and therefore, the simulator has no choice but to program the output into the random coins of Alice. This makes the construction only possible in the honest-but-curious model, where the simulator can specify the randomness used by Alice.[6]

[6]We note that this model can be upgraded to a malicious setting in the random oracle model, since the random oracle can be programmed to produce the correct random coins needed for the simulation.

Hubáček and Wichs [HW15] provide several impossibility results ruling out the alternative approaches, and even a weaker "honest-but-deterministic" adversarial model.

Compressed decoding and construction. In Definition 6.7.1, we formalize the notion of a compressed decoding procedure for Bob, which we will instantiate by adapting the $i\mathcal{O}$-based construction from Figure 6.5 in Figure 6.7.

Definition 6.7.1 (Compressed Decoding)**.** *We say an SMS scheme supports a compressed decoding procedure for Bob if the algorithm* Decode_B *can be decomposed into two efficient algorithms* $(\mathsf{DecodeCompressed}, \mathsf{DecodeExpand})$ *with the following syntax:*

- $\mathsf{DecodeCompressed}(\mathsf{crs}, f, \mathsf{pe}_A, \mathsf{st}_B) \to \tilde{z}_B$*. The deterministic compressed decoding algorithm takes as input the CRS* crs*, a function* $f \in \mathcal{F}$*, the public encoding* pe_A *belonging to Alice, and secret state* st_B *belonging to Bob. It outputs a compressed string* $\tilde{z}_B$*.*
- $\mathsf{DecodeExpand}(\tilde{z}_B) \to z_B$*. The deterministic expansion algorithm takes as input the string* $\tilde{z}_B$ *output by* $\mathsf{DecodeCompressed}$*. It outputs an* m*-bit string* $z_B \in \{0,1\}^m$*.*

The above algorithms must satisfy the following correctness, succinctness, and security properties.

Correctness. *For all inputs* input *in the domain of* Decode_B*, it holds that:*

$$\Pr\Big[\ \mathsf{Decode}_B(\mathsf{input}) = \mathsf{DecodeExpand}(\mathsf{DecodeCompressed}(\mathsf{input}))\ \Big] = 1.$$

Compactness. *There exists an* $\epsilon \in [0,1)$ *such that for all security parameters* $\lambda \in \mathbb{N}$*, every input* input *in the domain of* Decode_B*, it holds that:*

$$|\mathsf{DecodeCompressed}(\mathsf{input})| \leq \mathsf{poly}(\lambda) \cdot |\mathsf{Decode}(\mathsf{input})|^{\epsilon}.$$

Security. *Let* $\mathsf{poly}(\cdot)$ *be a fixed polynomial. There exists an efficient simulator* $\mathcal{S}$ *such that for all* crs *in the support* Setup*, for all* $f \in \mathcal{F}$*, and all inputs* X, y*,*

$$\left\{ (\mathsf{coins}, \mathsf{pe}_B, \tilde{z}_B) \;\middle|\; \begin{array}{r} \mathsf{coins} \xleftarrow{R} \{0,1\}^{\mathsf{poly}(\lambda,|X|)} \\ (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, f, X; \mathsf{coins}) \\ (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, f, y) \\ \tilde{z}_B := \mathsf{DecodeCompressed}(\mathsf{crs}, f, \mathsf{pe}_A, \mathsf{st}_B) \end{array} \right\} \approx_c \mathcal{S}(\mathsf{crs}, f, f(X,y), X).$$

Compressed Decoding

Parameters. Let $\mathsf{CPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ be a constrained PRF.

Changes to Encode_A and Encode_B in Figure 6.5.

- Encode_A "augments" $X = (x_1, \ldots, x_L)$ to consist of L tuples

$$X^{\mathsf{aug}} := \big((x_1, r_1), \ldots, (x_L, r_L)\big),$$

  where $r_1, \ldots, r_L \xleftarrow{R} \{0,1\}^m$ are random and of the same length as the output length m.
- Encode_B obfuscates an augmented program described in Program 2, which ignores the randomness component of each input tuple but is otherwise identical to Program 1.

$\mathsf{SMS.DecodeCompressed}_B(\mathsf{crs}, f, \mathsf{pe}_A, \mathsf{st}_B)$

1 : **parse** $\mathsf{crs} = \mathsf{hk}$ **and** $\mathsf{pe}_A = c_{\widehat{X}}$ **and** $\mathsf{st}_B = K$
2 : $S := \{c_{\widehat{X}} \| f \| i \mid i \in [L]\}$
3 : Define the predicate P: $x \mapsto x \in S$
4 : $\mathsf{csk} \leftarrow \mathsf{CPRF.Constrain}(K, P)$
5 : $\tilde{z}_B := (c_{\widehat{X}}, f, \mathsf{csk})$
6 : **return** $\tilde{z}_B$

$\mathsf{SMS.DecodeExpand}(\tilde{z}_B)$

1 : **parse** $\tilde{z}_B = (c_{\widehat{X}}, f, \mathsf{csk})$
2 : **foreach** $i \in [L]$:
3 : $\quad z_B^{(i)} := \mathsf{CPRF.CEval}(\mathsf{csk}, c_{\widehat{X}} \| f \| i)$
4 : **return** $\mathbf{z}_B := (z_B^{(1)}, \ldots, z_B^{(L)})$

Figure 6.7: Compressed decoding procedure for the $i\mathcal{O}$-based SMS scheme from Figure 6.5.

Program 2: (Parameterized by a universal circuit $\mathcal{U}$)

Hardcoded: (hk, K, y).
Input: $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$.
Procedure:

1: **if** $\widehat{x}_i = \mathsf{Commit}(x_i; r_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}_i, i, \pi_i) = 1$ **then**
2: $\quad$ **parse** $x_i = (x_i', \_)$ ▷ Parse x_i as a tuple and ignore the second component.
3: $\quad R_i := \mathsf{PPRF.Eval}(K, c_{\widehat{X}} \| f \| i)$
4: $\quad d := \mathcal{U}(f, x_i', y)$
5: $\quad$ **return** $d \oplus R_i$
6: **else return** $\perp$

Figure 6.8: Program obfuscated by Bob in Figure 6.7.

Proposition 6.7.1. *The compressed decoding procedure from Figure 6.7 satisfies Definition 6.7.1 with respect to the $i\mathcal{O}$-based SMS scheme from Figure 6.5.*

Proof. We prove each property in turn. We note, in passing, that CPRFs are known for all constraint predicates assuming $i\mathcal{O}$ and one-way functions [BZ14, BLW17].

Correctness. We first note that Program 1 and Program 2 are functionally equivalent and so

the output computed by Alice in Decode_A is unchanged. Correctness then follows immediately from the correctness of the CPRF (cf. Definition 2.3.1 in Chapter 2). In particular, the CPRF correctness guarantees that the evaluation on all inputs in the set S matches the evaluation of the CPRF on under the master key K. Therefore, when considering the subset of the domain on which the PRF is evaluated in Decode_B as defined in Figure 6.5, the evaluation using the constrained key csk is guaranteed to be identical.

Compactness. The CPRF already guarantees that $|\mathsf{csk}| \in \mathsf{poly}(\lambda, |P|)$, where P is predicate. Then, because our predicate is a range predicate over the domain $[L]$, it has size $O(\log L)$. We therefore have that $|\mathsf{csk}| = m^{\epsilon} \cdot \mathsf{poly}(\lambda)$ with $\epsilon = \log\log L / \log L$, where m is the output length.

Security. We first describe the simulator $\mathcal{S}$.

$\mathcal{S}$: On input $(\mathsf{crs},\ f,\ f(X,y), X)$:

- Parse $\mathsf{crs} = \mathsf{hk}$, $f(X,y) = \big(f(x_1,y),\ldots,f(x_L,y)\big)$, $X = (x_1,\ldots,x_L)$.
- Sample $\mathsf{rnd} \xleftarrow{R} (\{0,1\}^{\lambda})^L$ and $r_1,\ldots,r_L \xleftarrow{R} \{0,1\}^m$.
- Set $X^{\mathsf{aug}} := \big((x_1, r_1 \oplus f(x_1,y)),\ldots,(x_L, r_L \oplus f(x_L,y))\big)$.
- $(\mathsf{pe}_A, \mathsf{st}_A) := \mathsf{Encode}_A(\mathsf{crs}, 1^{|\mathcal{F}|}, X^{\mathsf{aug}}; \mathsf{rnd})$. ▷ Run Encode_A from Figure 6.5 with coins rnd.
- Parse $\mathsf{pe}_A = c_{\widehat{X}}$.
- Sample $K \leftarrow \mathsf{CPRF.KeyGen}(1^{\lambda})$ and $K_0 \leftarrow \mathsf{CPRF.KeyGen}(1^{\lambda})$.
- Compute $\widetilde{P^{\mathsf{sim}}} \leftarrow \mathsf{iO}(1^{\lambda}, P^{\mathsf{sim}})$, where P^{sim} is as described in Program 3 and has hard-coded inputs $(\mathsf{hk}, K_0, \mathsf{csk}, c_{\widehat{X}}, f)$.
- Define the set $S := \big\{c_{\widehat{X}} \| f \| i \mid i \in [L]\big\}$.
- Compute $\mathsf{csk} \leftarrow \mathsf{CPRF.Constrain}(K, S)$.
- Set $\mathsf{pe}_B := \widetilde{P^{\mathsf{sim}}}$ and $\tilde{z}_B := (c_{\widehat{X}}, f, \mathsf{csk})$.
- Set $\mathsf{coins} := (\mathsf{rnd}, r_1 \oplus f(x_1,y),\ldots,r_L \oplus f(x_L,y))$.
- Output $(\mathsf{coins}, \mathsf{pe}_B, \tilde{z}_B)$.

Program 3: The Simulated Program

Hardcoded: $(\mathsf{hk}, K_0, \mathsf{csk}, c_{\widehat{X}}, f)$.
Input: $(c'_{\widehat{X}}, (x_i, r_i), (\widehat{x}_i, r'_i), i, \pi_i, f')$.
Procedure:

1: **if** $\widehat{x}_i = \mathsf{Commit}((x_i, r_i); r'_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c'_{\widehat{X}}, \widehat{x}_i, i, \pi_i) = 1$ **then**

if $c'_{\widehat{X}} = c_{\widehat{X}} \wedge f' = f$ **then**	**else**
1: $R_i := \mathsf{PPRF.CEval}(\mathsf{csk}, c_{\widehat{X}} \| f \| i)$	1: $R_i := \mathsf{PPRF.Eval}(K_0, c'_{\widehat{X}} \| f' \| i)$
2: **return** $R_i \oplus r_i$	2: **return** $R_i \oplus r_i$

2: **else return** $\perp$

We now prove security via a hybrid argument.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of $(\mathsf{crs}, \mathsf{coins}, \mathsf{pe}_B, \tilde{z}_B)$, where pe_B is an obfuscation of Program 1.

- *Hybrid* $\mathcal{H}_1$. In this hybrid, we set coins to be as computed by $\mathcal{S}$.

  Claim. $\mathcal{H}_1 \approx_s \mathcal{H}_0$.

  Proof. $\mathcal{H}_1$ is perfectly indistinguishable from $\mathcal{H}_0$ since $r_i \oplus f(x_i, y)$ is distributed identically to a uniformly random value when r_i is sampled uniformly. □

- *Hybrid* $\mathcal{H}_2$. In this hybrid, we replace pe_B with an obfuscation of Program 3.

  Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$ *under the same assumptions as required for Lemma 6.12.1.*

  Proof (sketch). The proof follows as a corollary of the proof of Lemma 6.12.1. The sequence of hybrids in the proof of Proposition 6.6.3 are used to prove that a program that just outputs a PRF evaluation on all inputs is computationally indistinguishable from a program that outputs a PRF evaluation under a key K_0 for all inputs smaller than the j-th canonical input and output a secret masked by a PRF evaluation under a key K on all other inputs.

  Without loss of generality, we can reorder the hybrids such that all inputs to the PRF that are prefixed by $c_{\widehat{X}} \| f$ are evaluated under PRF key K and all other inputs are evaluated using the independent PRF key K_0. In doing so, we use the CPRF (rather than the puncturable PRF) to go from one hybrid to the next.

  Eventually, K_0 is only used on inputs that are not prefixed by $c_{\widehat{X}} \| f$, and the key K is used on all other inputs. Then, because of the fact that csk is the constrained key derived from K, and the evaluation using csk is equivalent to the evaluation under K for all inputs prefixed by $c_{\widehat{X}} \| f$, we can replace K with csk while keeping the programs functionally equivalent. □

At this point, $\mathcal{H}_2$ is identical to the output of $\mathcal{S}$, which concludes the proof. ■

6.7.3 Minimizing computation for Bob

We show an additional optimization allowing us reduce the computational complexity for Bob in certain cases. In particular, we consider the case where the output of the batch function is summed together, using Remark 38 (post-composition with a linear function). In this case, while the intermediate result held by Alice and Bob is of length $l \cdot L$, the final output is just one block of length l. Having Bob compute the full intermediate shares in this scenario results in "wasteful" computation, since the final output computed by Bob is just a pseudorandom share of length l rather than $l \cdot L$. As we will show, Bob's computation can be reduced to just $\mathsf{poly}(\lambda, \log L)$ as opposed to $\mathsf{poly}(\lambda, L)$ by resorting to an aggregatable PRF [CGV15]. In a nutshell, an aggregatable PRF allows the party with the PRF key K to compute $\bigoplus_{i=a}^{b} F_K(i)$,

for any range $[a, b]$ in the domain, in the same time it takes to evaluate the PRF F_K on a single input in the domain.

As was observed in Section 6.7.2, Bob's output consists of the PRF evaluated on input $(c_{\widehat{X}} \| f \| i)$, for i ranging from 1 to L. In particular, this share is computed by evaluating the PRF on a finite range of consecutive inputs in the domain, which are then summed together to derive the final share. Therefore, using an aggregatable PRF, the computation can be performed in $\mathsf{poly}(\lambda, \log L)$ time. However, because we require a *puncturable* PRF for the security proof, we must first construct what we call a puncturable aggregatable PRF (PAPRF) to be able to apply this optimization.

6.7.3.1 Puncturable and aggregatable PRF

A PRF that is both puncturable and aggregatable allows aggregating the PRF over a range $[a, b]$ using the master key, while given the punctured key, it should not be possible to evaluate the PRF on the punctured input $c \in [a, b]$ (or aggregate over a range that includes the punctured input).

Before explaining how we construct a puncturable aggregatable PRF (PAPRF), we first recall the black-box construction of Cohen, Goldwasser, and Vaikuntanathan [CGV15] for aggregatable PRFs supporting summation over a finite range.[7] Let G be any PRF. The aggregatable PRF F is constructed as:

$$G_K(x) = \begin{cases} F_K(0) & : x = 0, \\ F_K(x) \oplus F_K(x-1) & : x \neq 0. \end{cases}$$

Note that the PRF F_K is efficiently aggregatable on any range $[a, b]$ in the domain:

$$\bigoplus_{x \in [a,b]} G_K(x) = F_K(b) \oplus F_K(a-1).$$

If we want to puncture the above PRF on some input $c \in [a, b]$, then we need to puncture F_K on *two* points in the range, namely c and $c+1$, using a "t-puncturable" PRF (cf. Remark 43). In particular, if the punctured key only prevents evaluating the aggregatable PRF on input $c \in [a, b]$ but still enables aggregation elsewhere, then the punctured key can be used to recover $F_K(c)$, breaking puncturing security. To see this, note that it is possible to compute the aggregate over three ranges: $[a, c-1]$, $[c+1, b]$ and $[a, b]$ to then recover the PRF value on c by computing:

$$\bigoplus_{x \in [a,b]} F_K(x) \oplus \sum_{x \in [a,c-1]} F_K(x) \oplus \bigoplus_{x \in [c+1,b]} F_K(x) = F_K(c).$$

To get around this issue, we therefore require puncturing two consecutive inputs at a time.

We explain our construction next.

[7] We will not formally define the notion of aggregatable PRFs since we only focus on the simple case of summation over a range, which has a simple black-box construction.

PAPRF construction. We present the black-box construction of PAPRF in Figure 6.9. Our construction simply instantiates the aggregatable PRF using a 2-puncturable PRF.

Puncturable Aggregatable PRF

Parameters. Let F be a 2-puncturable PRF constructed (cf. Remark 43).

$\mathsf{PAPRF.KeyGen}(1^\lambda)$

1 : $K \leftarrow F.\mathsf{KeyGen}(1^\lambda)$
2 : **return** K

$\mathsf{PAPRF.Puncture}(K, x)$

1 : $S := \{x, x+1\}$
2 : $K^* \leftarrow F.\mathsf{Puncture}(K, S)$
3 : **return** K^*

$\mathsf{PAPRF.Eval}(K, x)$

1 : **if** $x = 0$: $y := F_K(x)$
2 : **else** $y := F_K(x) \oplus F_K(x-1)$
3 : **return** y

Figure 6.9: Puncturable Aggregatable PRF.

Lemma 6.7.2. *The PAPRF construction in Figure 6.9 is puncturable on any pair of consecutive inputs in the domain.*

Proof. Suppose, towards contradiction, that there exists an efficient adversary $\mathcal{A}$ that wins the puncturable PRF security game (extended to the 2-puncturing case in the natural way) with non-negligible advantage $\nu(\lambda)$ against the PAPRF construction.

Now, consider the following sequence of hybrid games.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the 2-puncturable PRF game.
- *Hybrid* $\mathcal{H}_1$. In this hybrid game, the 2-puncturing PRF challenger outputs a real-or-random challenge on one of the two punctured inputs (the other input is now always pseudorandom). It follows that $\mathcal{A}$'s advantage in $\mathcal{H}_1$ is at least $\nu/2$. Specifically, recall that we are considering F to be a 2-puncturable PRF as constructed in Remark 43 by taking the bit-wise XOR of the output of two independent 1-puncturable PRF evaluations.

We can now construct an efficient $\mathcal{B}$ breaking the 1-puncturing security of an underlying puncturable PRF instance used to realize the 2-puncturing PRF via Remark 43. $\mathcal{B}$ proceeds as follows:

1. Receive input x^* as the 1-punctured PRF challenge from $\mathcal{A}$.
2. Forward x^* to the challenger and receive the 1-punctured PRF key K_0^*.
3. Generate a fresh 1-puncturable PRF master key K_1 along with the corresponding punctured key K_1^*, punctured on input $x^* + 1$.
4. Define the 2-punctured key $K^* = (K_0^*, K_1^*)$.
5. Respond to $\mathcal{A}$ with K^*.
6. For each query x issued by $\mathcal{A}$:

- Query the challenger on x to get response $y_0^{(x)}$ and compute $y_1^{(x)} := F_{K_1}(x)$.
- Compute $y_0^{(x-1)} := F_{K_0^*}(x-1)$ and $y_1^{(x-1)} := F_{K_1}(x-1)$.
- Respond with $(y_0^{(x)} \oplus y_1^{(x)}) \oplus (y_0^{(x-1)} \oplus y_1^{(x-1)})$.

7. Output as $\mathcal{A}$ does.

First, an admissible $\mathcal{A}$ never queries $\mathcal{B}$ on input $x^* + 1$, so $\mathcal{B}$ answers all queries correctly and the responses are distributed identically to either $\mathcal{H}_0$ or $\mathcal{H}_1$, depending on the challenge. Therefore, $\mathcal{B}$ has advantage $\nu/2$ in breaking the 1-puncturing security game against F, which contradicts the puncturing security of F. ■

Lemma 6.7.3. *The PAPRF construction in Figure 6.9 is aggregatable over interval subsets of the domain.*

Proof. The construction is the same as the one of Cohen et al. [CGV15]. It follows that:

$$\bigoplus_{x \in [a,b]} \mathsf{PAPRF.Eval}(K, x) = F_K(b) \oplus F_K(a-1).$$

■

Remark 45 (Using a PAPRF with Figure 6.5)**.** *By using a PAPRF instead of a PPRF in Figure 6.5, Bob's computation time in the case where Alice and Bob compute an aggregation over their intermediate shares is reduced to* $\mathsf{poly}(\lambda, \log L)$*. Moreover, the security proof from Proposition 6.6.3 remains nearly identical with the only exception being that the PPRF is replaced with a 2-puncturable PRF. This requires puncturing two consecutive inputs at a time in the relevant hybrids but does not otherwise change the proof and analysis.*

6.8 Trapdoor Hashing from SMS

In this section, we show that an SMS scheme implies a trapdoor hashing scheme for all predicates that can be computed by the class of functions supported by the SMS scheme. In particular, this results in a TDH scheme for all predicates represented by polynomial-depth circuits under the LWE assumption (or all polynomial-size predicates if we additionally assume circular security of LWE).

6.8.1 Background on TDH and relation to SMS

We first recall the definition of trapdoor hashing (TDH) [DGI+19]. We adapt the definition of Döttling et al. [DGI+19] to only consider rate-1 trapdoor hash schemes (which is our SMS scheme will imply).

Definition 6.8.1 (Trapdoor Hash Scheme [DGI+19])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter and let $\mathcal{F} = \{\mathcal{F}_L\}_{L \in \mathbb{N}}$ be a class of predicates, where each $\mathcal{F}_L$ is a set of predicates defined over $\{0,1\}^L$. A trapdoor hash (TDH) scheme for $\mathcal{F}$ consists of efficient algorithms* $\mathsf{TDH} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Encode}, \mathsf{Decode})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda, 1^L) \to \mathsf{hk}$. *The randomized setup algorithm takes as input a security parameter and an input length L. It outputs a public hash key* hk.
- $\mathsf{KeyGen}(\mathsf{hk}, f) \to (\mathsf{ek}, \mathsf{td})$. *The randomized generation algorithm takes as input a hash key* hk *and a predicate $f \in \mathcal{F}_L$. It outputs an encoding key* ek *and a trapdoor* td.
- $\mathsf{Hash}(\mathsf{hk}, X; \rho) \to \mathsf{d}$. *The deterministic hashing algorithm takes as input a hash key* hk*, a string $X \in \{0,1\}^L$ and random coins $\rho \in \{0,1\}^*$. It outputs a digest $\mathsf{d} \in \{0,1\}^\ell$.*
- $\mathsf{Encode}(\mathsf{ek}, X; \rho) \to \mathsf{e}$. *The deterministic encoding algorithm takes as input an encoding key* ek*, a string $X \in \{0,1\}^L$ and random coins $\rho \in \{0,1\}^*$. It outputs a bit* e.
- $\mathsf{Decode}(\mathsf{td}, \mathsf{d}) \to (\mathsf{e}_0, \mathsf{e}_1)$. *The deterministic decoding algorithm takes as input a trapdoor* td*, a digest $\mathsf{d} \in \{0,1\}^\ell$, and outputs a pair of bits $(\mathsf{e}_0, \mathsf{e}_1)$.*

Correctness. *A TDH scheme is correct if for all security parameters $\lambda \in \mathbb{N}$, all $X \in \{0,1\}^L$, and all predicates $f \in \mathcal{F}_L$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\Pr\left[\mathsf{e} = \mathsf{e}_{f(X)} \wedge \mathsf{e} \neq \mathsf{e}_{1-f(X)} \quad : \quad \begin{array}{r} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^L) \\ (\mathsf{ek}, \mathsf{td}) \leftarrow \mathsf{KeyGen}(\mathsf{hk}, f) \\ \rho \xleftarrow{R} \{0,1\}^\star \\ \mathsf{d} := \mathsf{Hash}(\mathsf{hk}, X; \rho) \\ \mathsf{e} := \mathsf{Encode}(\mathsf{ek}, X; \rho) \\ (\mathsf{e}_0, \mathsf{e}_1) := \mathsf{Decode}(\mathsf{td}, \mathsf{d}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Function Privacy. *A TDH scheme is function-private if for all $L = L(\lambda) \in \mathsf{poly}(\lambda)$, and all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\Pr\left[b = b' \quad : \quad \begin{array}{r} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^L) \\ (f_0, f_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{hk}) \\ b \xleftarrow{R} \{0,1\} \\ (\mathsf{ek}, \mathsf{td}) \leftarrow \mathsf{KeyGen}(\mathsf{hk}, f_b) \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{ek}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $f_0, f_1 \in \mathcal{F}_L$.

Input Privacy. *A TDH scheme is input-private if for all $L = L(\lambda) \in \mathsf{poly}(\lambda)$, and all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\Pr\left[b = b' \quad : \quad \begin{array}{r} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^L) \\ (X_0, X_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{hk}) \\ \rho \xleftarrow{R} \{0,1\}^\star,\ b \xleftarrow{R} \{0,1\} \\ \mathsf{d} := \mathsf{Hash}(\mathsf{hk}, X_b; \rho) \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{d}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $X_0, X_1 \in \{0,1\}^L$.

Compactness. *A TDH scheme is compact if the digest length* $\ell = \ell(\lambda) \in \mathsf{poly}(\lambda)$ *is independent of the input length* L.

6.8.2 Construction from SMS

We construct TDH from SMS in Figure 6.10. We briefly provide some intuition for the relationship between TDH and SMS, which helps understand the construction.

Relationship between TDH and SMS. A trapdoor hash (TDH) scheme defines a publicly parameterized hash function $\mathsf{Hash}_{\mathsf{hk}} : \{0,1\}^L \to \{0,1\}^\ell$ which allows Alice and Bob to execute the following functionality, described by Döttling et al. [DGI+19]. Here, we explain how it relates to SMS, which gives some intuition for our construction.

- *Step 1: Generate a key and encoding.* Bob with a private predicate $f \in \mathcal{F}$ over $\{0,1\}^L$, for some class of predicates $\mathcal{F}$, generates an encoding key ek and a trapdoor td. The encoding key ek can be made public and hides the function f, thanks to the function-privacy property. This step can be emulated by using Bob's SMS encoding algorithm $\mathsf{SMS.Encode}_B$ with input f.

- *Step 2: Hashing.* Alice, who has a (long) private input $X \in \{0,1\}^L$, can use the public hash key hk to compute a short digest $\mathsf{d} := \mathsf{Hash}_{\mathsf{hk}}(X)$ that does not reveal X thanks to the input-privacy property, and send it to Bob. This step can be emulated by using Alice's SMS encoding algorithm $\mathsf{SMS.Encode}_A$ with input X.

- *Step 3: Encoding.* Using the encoding key ek, anyone (including Alice), can compute an encoding $\mathsf{e} := \mathsf{Encode}(\mathsf{ek}, X)$ for an input $X \in \{0,1\}^L$. This step can be emulated using $\mathsf{SMS.Decode}_A$ with input X.

- *Step 4: Decoding.* Bob, who has the secret trapdoor td, can decode the encoding e to recover $f(X)$, given only the digest d. This step can be emulated using $\mathsf{SMS.Decode}_B$ with input f.

Observe that the above functionality is similar to SMS yet is strictly weaker in several respects. First, TDH does not require output-succinctness, since it is defined around predicates (which output a single bit) rather than computing shares of a function. As such, SMS (satisfying output succinctness, cf. Definition 6.4.2) is stronger. Similarly, TDH does not require succinctness for Bob's encoding key ek (which may grow, for example, with the size of X). In contrast, SMS requires input succinctness for both parties' encodings, which makes our TDH construction achieve this extra feature "for free."

Trapdoor Hashing from SMS

Public Parameters. Let $\mathsf{SMS} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma\in\{A,B\}})$ be an SMS scheme, and let C be a circuit that takes as input a function f and input X and outputs $f(X)$.

$\mathsf{TDH.Setup}(1^\lambda, 1^L)$:

1 : $\mathsf{crs} \leftarrow \mathsf{SMS.Setup}(1^\lambda)$
2 : **return** $\mathsf{hk} := \mathsf{crs}$

$\mathsf{TDH.KeyGen}(\mathsf{hk}, f; \rho)$:

1 : **parse** $\mathsf{hk} = \mathsf{crs}$
2 : $(\mathsf{pe}_B, \mathsf{st}_B) := \mathsf{SMS.Encode}_B(\mathsf{crs}, C, f; \rho)$
3 : $\mathsf{td} := (\mathsf{crs}, f, \mathsf{st}_B)$
4 : $\mathsf{ek} := \mathsf{pe}_B$
5 : **return** $(\mathsf{ek}, \mathsf{td})$

$\mathsf{TDH.Hash}(\mathsf{hk}, X; \rho)$:

1 : **parse** $\mathsf{hk} := \mathsf{crs}$
2 : $(\mathsf{pe}_A, \_) := \mathsf{SMS.Encode}_A(\mathsf{crs}, C, X; \rho)$
3 : **return** $\mathsf{d} := \mathsf{pe}_A$

$\mathsf{TDH.Encode}(\mathsf{ek}, X; \rho)$:

1 : **parse** $\mathsf{ek} := \mathsf{pe}_B$
2 : $(\mathsf{pe}_A, \mathsf{st}_A) := \mathsf{SMS.Encode}_A(\mathsf{crs}, C, X; \rho)$
3 : $z_A := \mathsf{SMS.Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A)$
4 : **return** $\mathsf{e} := z_A$

$\mathsf{TDH.Decode}(\mathsf{td}, \mathsf{d})$:

1 : **parse** $\mathsf{td} := (\mathsf{crs}, f, \mathsf{st}_B)$
2 : **parse** $\mathsf{d} = \mathsf{pe}_A$
3 : $z_B := \mathsf{SMS.Decode}_B(\mathsf{crs}, f, \mathsf{pe}_A, \mathsf{st}_B)$
4 : $\mathsf{e}_0 := z_B$
5 : $\mathsf{e}_1 := z_B \oplus 1$
6 : **return** $(\mathsf{e}_0, \mathsf{e}_1)$

Figure 6.10: Trapdoor Hashing from SMS.

Proposition 6.8.1. *Assume the existence of an SMS scheme for a family of functions $\mathcal{F}$ satisfying full succinctness (cf. Definition 6.4.3) where, additionally, Alice's public encoding is of size* $\mathsf{poly}(\lambda)$. *The* TDH *construction from Figure 6.10 is a trapdoor hashing scheme (cf. Definition 6.8.1) for any class of predicates computable via $\mathcal{F}$.*

Proof. We prove each property in turn, we note that all the properties of TDH are are almost immediately implied by SMS.

Correctness. To see correctness, observe that when the predicate outputs zero, i.e., $f(X) = 0$, then by inspection and by the correctness of SMS, it holds that

$$\Pr[\mathsf{e} \oplus \mathsf{e}_0 = f(X)] \geq 1 - \mathsf{negl}(\lambda)$$

and so it holds that $\Pr[\mathsf{e} = \mathsf{e}_0] \geq 1 - \mathsf{negl}(\lambda)$, when $f(X) = 0$. Similarly, by noting that $\mathsf{e}_1 = (\mathsf{e}_0 \oplus 1)$, we have that

$$\Pr[\mathsf{e} \oplus \mathsf{e}_1 = f(X) \oplus 1] \geq 1 - \mathsf{negl}(\lambda),$$

and so we have that $\Pr[\mathsf{e} \neq \mathsf{e}_1] \geq 1 - \mathsf{negl}(\lambda)$.

The case where $f(X) = 1$ follows by symmetry. Thus, correctness holds.

Function and Input Privacy. Function and input privacy follow directly from the SMS privacy for Bob and Alice, respectively. Specifically, the hashing key hk consists only of the CRS.

To see function privacy, it suffices to note that the encoding key ek is simply the SMS public encoding of Bob with input $y := f$ (i.e., the description of the function, which is private to Bob). As such, SMS guarantees privacy of the function in the resulting TDH scheme.

To see input privacy, it suffices to note that the digest d is simply the public encoding of Alice computed over her input X. As such, SMS guarantees privacy of the input in the TDH scheme.

Compactness. Since the digest consists solely of Alice's public encoding in the SMS scheme, the size of the digest is $\mathsf{poly}(\lambda)$ when using a fully input-succinct SMS scheme satisfying the theorem statement (i.e., where Alice's public encoding is independent of $|X|$). ■

Remark 46. *We note that Bob's encoding in Figure 6.10 may grow with the size of $|X|$ in the case where the predicate does not admit a succinct description, since SMS only guarantees succinctness in Alice's input but not Bob's input. However, this is admissible in the context of TDH where there are no restrictions on the size of Bob's encoding key* ek.

Using the fact that our LWE construction of SMS from Section 6.5 supports polynomial-depth circuits (or all circuits under circular-security of LWE) and, in addition, Alice's public encoding in our construction is of size $\mathsf{poly}(\lambda)$, we obtain the following corollary:

Corollary 6.8.1. *Under the LWE assumption (with a superpolynomial modulus-to-noise ratio), there exists a TDH scheme for all polynomial-depth predicates. By additionally assuming the circular security of LWE, there exists a TDH scheme for all polynomial-size predicates.*

6.9 Rate-1 FHE from SMS

In this section, we show that SMS can be used to compile any fully homomorphic encryption scheme into a rate-1 FHE scheme. We recall the definition of a rate-1 FHE [BDGM19].

At a high level, a rate-1 FHE scheme is an FHE scheme (cf. Definition 6.3.2) adorned with additional algorithms to compress a vector of ciphertexts into a compact representation that approaches the message length, asymptotically.

Definition 6.9.1 (Rate-1 Fully Homomorphic Encryption [BDGM19])**.** *A* rate-1 *FHE scheme consists of an FHE scheme* $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *adorned with additional algorithms* $(\mathsf{Compress}, \mathsf{CompressDec})$ *with the following syntax:*

- $\mathsf{Compress}(\mathsf{pk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_m)) \to \mathsf{ct}^*$. *The deterministic compression algorithm takes as input the public key* pk *and* m *(possibly evaluated) ciphertexts* $(\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$. *It outputs a compressed ciphertext* ct^*.
- $\mathsf{CompressDec}(\mathsf{sk}, \mathsf{ct}^*) \to x$. *The deterministic compressed decryption algorithm takes as input a compressed ciphertext, the secret key* sk, *and a compressed ciphertext* ct^*. *It outputs the message* x.

The above algorithms must satisfy the following properties:

Compressed Correctness. *There exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all messages* $x_1, \ldots, x_\ell \in \mathcal{M}$ *and all* ℓ*-argument,* m*-output functions* f *that can be represented by polynomial-size circuits, we have that:*

$$\Pr\left[\begin{array}{c} \mathsf{CompressDec}(\mathsf{sk}, \mathsf{ct}^*) \\ = f(x_1, \ldots, x_\ell) \end{array} : \begin{array}{r} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x_i), \forall i \in [\ell] \\ \mathsf{ct}'_j \leftarrow \mathsf{Eval}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)), \forall j \in [m] \\ \mathsf{ct}^* \leftarrow \mathsf{Compress}(\mathsf{pk}, (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_m)) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

where the probability is over the randomness of KeyGen *and* Enc.

Rate-1. *For any* $(\mathsf{pk}, \mathsf{sk})$ *in the support of* KeyGen *and for all ciphertexts* $\mathsf{ct}_1, \ldots, \mathsf{ct}_m$ *in the support of either* $\mathsf{Enc}(\mathsf{pk}, \cdot)$ *or* $\mathsf{Eval}(\mathsf{pk}, \cdot)$, *it holds that:*

$$\lim_{m \to \infty} \frac{m \cdot |\mathcal{M}|}{|\mathsf{Compress}(\mathsf{pk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_m))|} = 1.$$

That is, the compressed ciphertext output by $\mathsf{Compress}$ *is asymptotically of the same length as the tuple of* m *messages it encrypts.*

6.9.1 Generic construction from SMS

In Figure 6.11, we present the construction of rate-1 FHE using a sufficiently powerful SMS scheme. Our construction closely follows the overview from Section 6.1.1.

Remark 47. *We briefly remark that, if the underlying (non-compact) FHE scheme satisfies near-linear decryption (cf. Theorem 6.5.1), then we can obtain a rate-1 FHE using the transformation described in Figure 6.11 and SMS for just degree-2 functions (e.g., succinct NIVOLE, for instance). This is because the decryption can be described as a linear function and we can replace* $\mathsf{SMS.Decode}$ *with a rounding operation to round-away the error, resulting in additive shares. In other words, we can use the same trick we exploit in our LWE-based construction in Figure 6.2 and round-away the error from the noisy shares of the decryption using Lemma 6.2.1. However, we stress that Figure 6.11 is* generic *and thus works using any black-box FHE scheme (which may not necessarily have a near-linear decryption).*

Rate-1 FHE from SMS

Public Parameters. Let $\mathsf{SMS} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma\in\{A,B\}})$ be an SMS scheme, let $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be an FHE scheme (cf. Definition 6.3.2), and f be the function that takes as input any FHE secret key $\widetilde{\mathsf{sk}}$ and any L FHE ciphertexts $\widetilde{\mathsf{ct}}_1, \ldots, \widetilde{\mathsf{ct}}_L$, and outputs $\mathsf{FHE.Dec}(\widetilde{\mathsf{sk}}, (\widetilde{\mathsf{ct}}_1, \ldots, \widetilde{\mathsf{ct}}_L))$.

$\mathsf{FHE1.KeyGen}(1^\lambda)$

1 : $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
2 : $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$
3 : $(\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, f, \mathsf{sk}')$
4 : $\mathsf{pk} := (\mathsf{crs}, \mathsf{pk}', \mathsf{pe}_B)$
5 : $\mathsf{sk} := (\mathsf{crs}, \mathsf{sk}', \mathsf{st}_B)$
6 : **return** $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{FHE1.Compress}(\mathsf{pk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_L))$

1 : **parse** $\mathsf{pk} = (\mathsf{crs}, \mathsf{pk}', \mathsf{pe}_B)$
2 : $X := (\mathsf{ct}_1, \ldots, \mathsf{ct}_L)$
3 : $(\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, f, X)$
4 : $z_A := \mathsf{Decode}_A(\mathsf{crs}, f, \mathsf{pe}_B, \mathsf{st}_A)$
5 : $\mathsf{ct}^* := (\mathsf{pe}_A, z_A)$
6 : **return** ct^*

$\mathsf{FHE1.CompressDec}(\mathsf{sk}, \mathsf{ct}^*)$

1 : **parse** $\mathsf{sk} = (\mathsf{crs}, \mathsf{sk}', \mathsf{st}_B)$
2 : **parse** $\mathsf{ct}^* = (\mathsf{pe}_A, z_A)$
3 : $z_B := \mathsf{Decode}_B(\mathsf{crs}, f, \mathsf{pe}_A, \mathsf{st}_B)$
4 : $x := z_A \oplus z_B$
5 : **return** x

$\mathsf{FHE1.Enc}(\mathsf{pk}, x)$

1 : **parse** $\mathsf{pk} = (\mathsf{crs}, \mathsf{pk}', \mathsf{pe}_B)$
2 : **return** $\mathsf{FHE.Enc}(\mathsf{pk}', x)$

$\mathsf{FHE1.Eval}(\mathsf{pk}, f, \mathbf{ct})$

1 : **parse** $\mathsf{pk} = (\mathsf{crs}, \mathsf{pk}', \mathsf{pe}_B)$
2 : **return** $\mathsf{FHE.Eval}(\mathsf{pk}', \mathbf{ct})$

$\mathsf{FHE1.Dec}(\mathsf{sk}, \mathsf{ct})$

1 : **parse** $\mathsf{sk} = (\mathsf{crs}, \mathsf{sk}', \mathsf{st}_B)$
2 : **return** $\mathsf{FHE.Dec}(\mathsf{sk}', \mathsf{ct})$

Figure 6.11: Rate-1 FHE from SMS.

6.9.2 Security analysis

We now prove security of our construction.

Proposition 6.9.1. *Assume that* FHE *is an FHE scheme satisfying Definition 6.3.2. The construction of* FHE1 *from Figure 6.11 is a rate-1 FHE scheme satisfying Definition 6.9.1.*

Proof. We prove each required property in turn.

Correctness. Correctness of $\mathsf{FHE1.Enc}$, $\mathsf{FHE1.Eval}$, and $\mathsf{FHE1.Dec}$ follow immediately from the correctness of FHE. In fact, our transformation does not modify these algorithms.

Compressed Correctness. We now examine the correctness of $\mathsf{FHE1.CompressDec}$. Notice that from the correctness of SMS, we have that:

$$\Pr\Big[\ z_A \oplus z_B = \mathsf{FHE.Dec}(\mathsf{sk}', (\mathsf{ct}_1, \ldots, \mathsf{ct}_L))\ \Big] \geq 1 - \mathsf{negl}(\lambda).$$

Separately, by the correctness of FHE and a simple union bound over all $L = L(\lambda) \in \mathsf{poly}(\lambda)$ decryptions, we have that:

$$\Pr\Big[\ \mathsf{FHE.Dec}(\mathsf{sk}', (\mathsf{ct}_1, \ldots, \mathsf{ct}_L) = (x_1, \ldots, x_L)\ \Big] \geq 1 - \mathsf{negl}(\lambda).$$

Therefore, we have that:

$$\Pr\Big[\ z_A \oplus z_B = (x_1, \ldots, x_L)\ \Big] \geq 1 - \mathsf{negl}(\lambda).$$

This concludes the proof of the compressed correctness property.

Compactness. Similarly to correctness, the compactness follows directly from the compactness property of FHE.

Rate-1. We recall that the compressed ciphertext ct^* output by $\mathsf{FHE1.Compress}$ is of the form (pe_A, z_A). Because of the additive reconstruction property of SMS (cf. Definition 6.4.1), we have that:

$$|z_A| = |(x_1, \ldots, x_L)| = |\mathcal{M}| \cdot L.$$

Moreover, from the ϵ-succinctness property of SMS (cf. Definition 6.4.3), it follows that

$$|\mathsf{pe}_A| \leq |\mathcal{M}| \cdot L^{\epsilon} \cdot \mathsf{poly}(\lambda).$$

Therefore, we have that $|\mathsf{ct}^*| \leq |\mathcal{M}| \cdot L + |\mathcal{M}| \cdot L^{\epsilon} \cdot \mathsf{poly}(\lambda)$, which asymptotically approaches $|\mathcal{M}| \cdot L$.

Security. To prove security, consider the following sequence of hybrids.

- *Hybrid* $\mathcal{H}_0$. This hybrid corresponds to the distribution where x_0 is encrypted. That is, $\mathcal{H}_0$ is defined as:

$$\left\{ (\mathsf{pk}, \mathsf{ct}_0) \;\middle|\; \begin{array}{r} (\mathsf{pk}, \_) \leftarrow \mathsf{FHE1.KeyGen}(1^\lambda) \\ \mathsf{ct}_0 \leftarrow \mathsf{FHE1.Enc}(\mathsf{pk}, x_0) \end{array} \right\}.$$

- *Hybrid* $\mathcal{H}_1$. In this hybrid, we change $\mathsf{FHE1.KeyGen}$ to output $\mathsf{pk} = (\mathsf{crs}, \mathsf{pk}', \widetilde{\mathsf{pe}}_B)$, where crs and pk are distributed identically to $\mathcal{H}_0$ but where $\widetilde{\mathsf{pe}} \leftarrow \mathsf{Encode}_A(\mathsf{crs}, f, 0)$.

  It follows that $\mathcal{H}_1 \approx_c \mathcal{H}_0$ by the security of SMS.

- *Hybrid* $\mathcal{H}_2$. In this hybrid, we encrypt x_1. That is,

$$\mathcal{H}_2 := \left\{ (\mathsf{pk}, \mathsf{ct}_1) \;\middle|\; \begin{array}{r} (\mathsf{pk}, \_) \leftarrow \mathsf{FHE1.KeyGen}(1^\lambda) \\ \mathsf{ct}_0 \leftarrow \mathsf{FHE1.Enc}(\mathsf{pk}, x_1) \end{array} \right\}.$$

  It follows that $\mathcal{H}_2 \approx_c \mathcal{H}_1$ by the security of FHE, given that Enc simply outputs $\mathsf{FHE.Enc}$.

- *Hybrid* $\mathcal{H}_3$. In this hybrid, we revert the changes made in $\mathcal{H}_1$ and output pk distributed identically to KeyGen in $\mathcal{H}_0$.

  It follows that $\mathcal{H}_3 \approx_c \mathcal{H}_2$ by the security of SMS. Note that $\mathcal{H}_3$ is distributed identically to:

$$\left\{ (\mathsf{pk}, \mathsf{ct}_1) \;\middle|\; \begin{array}{r} (\mathsf{pk}, \_) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{pk}, x_1) \end{array} \right\}.$$

It follows that no efficient adversary can distinguish between an encryption of x_0 and x_1 with better than negligible advantage. This concludes the proof of security. ■

6.10 Correlation-Intractable Hashing from SMS

In this section, we show that an SMS scheme implies a correlation-intractable (CI) hashing for all efficiently searchable relations.

Definition 6.10.1 (Searchable Relations [PS19])**.** *We say that a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ is searchable in size S if there exists a function f computable by a size S circuit, such that for any $(x, y) \in R$, it holds that $f(x) = y$. This, in particular, means that for every x, there is at most one witness y for its membership in the relation.*

Definition 6.10.2 (Correlation-Intractable Hash Function)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter and let $S(\lambda) \in \mathsf{poly}(\lambda)$ be a circuit size parameter. Let $n = n(\lambda)$ be the input length parameter and $m = m(\lambda)$ be the output length parameter. Let $\mathcal{R} = \{\mathcal{R}_\lambda\}_\lambda$ be a class of relations that is searchable by circuits of size $S(\lambda)$ via functions $\mathcal{F} = \{f_\lambda \colon \{0,1\}^n \to \{0,1\}^m\}_\lambda$. A correlation-intractable (CI) hash function is given by a tuple of algorithms $\mathsf{CI} = (\mathsf{KeyGen}, \mathsf{Hash})$ with the following syntax:*

- $\mathsf{KeyGen}(1^\lambda, f) \to \mathsf{hk}$. *The randomized key generation algorithm takes as input the security parameter λ and a function description $f \in \mathcal{F}$. It outputs a hash key hk.*
- $\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{d}$. *The deterministic hashing algorithm takes as input the hash key and an input $x \in \{0,1\}^n$. It outputs a digest $\mathsf{d} \in \{0,1\}^m$.*

The above algorithms must satisfy the following properties:

Indistinguishability. *For all security parameters $\lambda \in \mathbb{N}$ and all functions $f \in \mathcal{F}$, it holds that:*

$$\left\{ \mathsf{hk} \;\middle|\; \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, f) \right\} \approx_c \left\{ \mathsf{hk} \;\middle|\; \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathbf{0}) \right\},$$

where $\mathbf{0}$ is the all-zeroes function padded to size $S(\lambda)$.

Correlation-Intractability. *A hash function family* $(\mathsf{KeyGen}, \mathsf{Hash})$ *is said to be correlation-intractable for the relation* $\mathcal{R}$ *if for all efficient adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\mathsf{d} = \mathsf{Hash}(\mathsf{hk}, x) \wedge (x, y) \in R_\lambda \quad : \quad \begin{array}{r} \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, f_\lambda) \\ (x, \mathsf{d}) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the probability is over the randomness of KeyGen *and* $\mathcal{A}$*.*

6.10.1 Generic construction from SMS

The construction from SMS is nearly identical to the one described by Brakerski et al. [BKM20] using trapdoor hash functions. We note that the digest of the CI hash need not be succinct (and the transformation does not output a succinct digest). Instead, the succinctness property of SMS is used to argue correlation intractability.

Correlation-Intractable Hashing from SMS

Let $\mathsf{SMS} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma \in \{A,B\}})$ be an SMS scheme, $\mathcal{F}$ be a family of functions (represented by circuits of size $S = S(\lambda)$) that map $n = n(\lambda)$ bits to $m = m(\lambda)$ bits. Let $\mathcal{U}$ be the a universal circuit that takes as input a function $f \in \mathcal{F}$ and an input x and outputs $f(x)$.

$\mathsf{CI.KeyGen}(1^\lambda, f)$

1 : $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
2 : $(\mathsf{pe}_B, \_) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, \mathcal{U}, f)$
3 : $z \xleftarrow{R} \{0,1\}^m$
4 : $r \xleftarrow{R} \{0,1\}^\lambda$
5 : $\mathsf{hk} := (\mathsf{crs}, \mathsf{pe}_B, z, r)$
6 : **return** hk

$\mathsf{CI.Hash}(\mathsf{hk}, x)$

1 : **parse** $\mathsf{hk} = (\mathsf{crs}, \mathsf{st}_B, z, r)$
2 : $(\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, \mathcal{U}, x; r)$
▷ Encode_A uses fixed randomness r.
3 : $z_A := \mathsf{Decode}_A(\mathsf{crs}, \mathcal{U}, \mathsf{pe}_B, \mathsf{st}_A)$
4 : $\mathsf{d} := z \oplus z_A$
5 : **return** d

Figure 6.12: Correlation-Intractable Hashing from SMS.

Proposition 6.10.1. *Assume the existence of an SMS scheme satisfying full succinctness (cf. Definition 6.4.3). The* CI *construction from Figure 6.12 is a correlation-intractable hash function family satisfying Definition 6.10.2.*

Proof. We prove each property in turn.

Indistinguishability. The indistinguishability of the hashing keys follows directly from the SMS security of Bob. In particular, the choice of function f is equivalent to Bob's private input in SMS, and indistinguishability follows trivially from Definition 6.4.1.

Correlation-Intractability. Suppose, towards contradiction, that there exists an efficient adversary $\mathcal{A}$ that breaks the correlation-intractability property with non-negligible probability $\nu(\lambda)$. That is,

$$\Pr\left[\mathsf{d} = \mathsf{Hash}(\mathsf{hk}, x) \wedge (x, y) \in R_\lambda \quad : \quad \begin{array}{r} \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, f_\lambda) \\ (x, \mathsf{d}) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array} \right] \geq \nu(\lambda).$$

Then, because $(x, y) \in R_\lambda$, we have that $\mathsf{d} = f_\lambda(x)$, and so we can equivalently write $\mathsf{d} = z \oplus z_A$, as defined in Hash. That is, we have that:

$$\Pr\left[f_\lambda(x) = z \oplus z_A \quad : \quad \begin{array}{r} \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, f_\lambda) \\ (x, \mathsf{d}) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array} \right] \geq \nu(\lambda).$$

Next, by the correctness of SMS, we have that $f_\lambda(x) = z_A \oplus \mathsf{Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B)$, and so we have that:

$$\Pr\left[z_A \oplus z_B = z \oplus z_A \quad : \quad \begin{array}{r} \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda, f_\lambda) \\ (x, \mathsf{d}) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array} \right] \geq \nu(\lambda) - \mathsf{negl}(\lambda),$$

where $z_B = \mathsf{Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B)$.

Therefore, we have that, with probability at least $\nu(\lambda) - \mathsf{negl}(\lambda)$ over the randomness of KeyGen and $\mathcal{A}$, it holds that $z = z_B$. We will show that this raises a contradiction.

Let $m = m(\lambda) \in \mathsf{poly}(\lambda)$ be the length of the output of the hash. Note that fixing both crs and st_B, the set of all possible z_B is bounded by the set of all possible pe_B, given that Decode_B is deterministic.

From the ϵ-output succinctness of the SMS scheme, we have $|\mathsf{pe}_A| \leq m^\epsilon \cdot \mathsf{poly}(\lambda)$. Thus, the set of all possible z_B values has size at most $2^{m^\epsilon \cdot \mathsf{poly}(\lambda)}$. Recall that z is a randomly and independently chosen string of size m. Therefore, the probability that z belongs to this set is at most $\frac{2^{m^\epsilon \cdot \mathsf{poly}(\lambda)}}{2^m}$, which is negligible when m is a large enough polynomial in λ. This contradicts the inequality we derived above, assuming that $\nu(\cdot)$ is a non-negligible function.

This concludes the proof of correlation-intractability and the proof of Proposition 6.10.1. ■

6.11 Generic Upgrade to a Simulation-Based Definition

In this section, we show that Definition 6.4.1 can be generically upgraded to satisfy a simulation-based definition for secure computation. Specifically, we show that any SMS scheme can be generically made to satisfy the (corruptible) ideal functionality presented in Functionality 1.

Transformation. To transform any SMS scheme satisfying Definition 6.4.1 into one that instantiates the ideal functionality presented in Functionality 1, we need to "re-randomize" the output shares (otherwise the simulator would be unable to properly simulate the output in the case where both parties are honest). This randomization can be achieved with the help

Functionality $\mathcal{F}_{\mathsf{sms}}$

Procedure. The functionality is instantiated between parties Alice and Bob and an adversary $\mathcal{A}$ playing the role of Charlie and possibly corrupting either Alice or Bob. The functionality aborts if it receives any incorrectly formatted messages.

- If both parties are honest:
  1: Wait for input (X, f) from Alice and (y, f) from Bob.
  2: Sample $R_i \xleftarrow{R} \{0,1\}^{|f(\cdot,\cdot)|}$.
  3: Output $f(X, y) \oplus R_i$ to Alice, R_i to Bob, and $(f(X, y) \oplus R_i, R_i)$ to $\mathcal{A}$.
- If Alice is corrupted:
  1: Wait for input (X, f, out) from $\mathcal{A}$ and (y, f) from Bob.
  2: Output $(f(X, y) \oplus \mathsf{out},\ \mathsf{out})$ to $\mathcal{A}$, $f(X, y) \oplus \mathsf{out}$ to Bob.
- If Bob is corrupted:
  1: Wait for input (y, out) from $\mathcal{A}$ and X from Alice.
  2: Output $(f(X, y) \oplus \mathsf{out},\ \mathsf{out})$ to $\mathcal{A}$, $f(X, y) \oplus \mathsf{out}$ to Alice.

Functionality 1: Corruptible ideal functionality for SMS.

of a pseudorandom function that the parties use to generate pseudorandom shares of zero, and does not require introducing any new assumptions. The transformation is described in Figure 6.13 and uses a non-interactive key exchange (NIKE), which we recall is implied by Definition 6.4.1 using the reduction of Boyle et al. [BGI+18]. We formally define NIKE in Definition 6.11.1.

Definition 6.11.1 (Non-Interactive Key Exchange [DH76, CKS08, FHKP13])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. A non-interactive key exchange (NIKE) scheme consists of algorithms $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer})$ with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$*. The randomized setup algorithm takes as input the security parameter λ and outputs a common reference string* crs*.*
- $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$*. The randomized key generation algorithm takes as input the CRS* crs*. It outputs a public key* pk *and secret key* sk*.*
- $\mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_i, \mathsf{sk}_j) \to K$*. The deterministic key derivation algorithm takes as input the CRS* crs*, a public key* pk_i*, and a secret key* sk_j*. It outputs a key $K \in \{0,1\}^\lambda$.*

The above algorithms must satisfy the following properties:

Correctness. *For all security parameters $\lambda \in \mathbb{N}$, it holds that:*

$$\Pr\left[\begin{array}{rcl} & & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & & (\mathsf{pk}_A, \mathsf{sk}_A) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ K_A = K_B & : & (\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ & & K_A \leftarrow \mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_B, \mathsf{sk}_A) \\ & & K_B \leftarrow \mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_A, \mathsf{sk}_B) \end{array}\right] = 1.$$

Security. *For all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\Pr\left[\begin{array}{rcl} & & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ & & (\mathsf{pk}_A, \mathsf{sk}_A) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ & & (\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ b = b' & : & K_0 \leftarrow \mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_A, \mathsf{sk}_B) \\ & & K_1 \xleftarrow{R} \{0,1\}^\lambda \\ & & b \xleftarrow{R} \{0,1\} \\ & & b' \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_A, \mathsf{pk}_B, K_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

In particular, this security definition for NIKE is known as "CKS-light" security [FHKP13], which is known to be polynomially equivalent to stronger notions of NIKE.

Simulation-Secure SMS Transformation

Let $\mathsf{SMS} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma\in\{A,B\}})$ be an SMS scheme, $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer})$ be a NIKE scheme, and let $F\colon \{0,1\}^\lambda \times \{0,1\}^\star \to \{0,1\}^m$ be a PRF. We let $\overline{\sigma} := \{A, B\} \setminus \{\sigma\}$.

$\mathsf{SMS}^*.\mathsf{Setup}(1^\lambda)$:

1 : $\mathsf{crs}_{\mathsf{sms}} \leftarrow \mathsf{SMS.Setup}(1^\lambda)$
2 : $\mathsf{crs}_{\mathsf{nike}} \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$
3 : **return** $\mathsf{crs} := (\mathsf{crs}_{\mathsf{sms}}, \mathsf{crs}_{\mathsf{nike}})$

$\mathsf{SMS}^*.\mathsf{Encode}_\sigma(\mathsf{crs}, f, x_\sigma)$:

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{sms}}, \mathsf{crs}_{\mathsf{nike}})$
2 : $(\mathsf{pe}'_\sigma, \mathsf{st}'_\sigma) \leftarrow \mathsf{SMS.Encode}_\sigma(\mathsf{crs}, f, x_\sigma)$
3 : $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{crs}_{\mathsf{nike}})$
4 : $\mathsf{pe}_\sigma := (\mathsf{pe}'_\sigma, \mathsf{pk}_\sigma)$
5 : **return** $(\mathsf{pe}_\sigma, \mathsf{st}_\sigma)$

$\mathsf{SMS}^*.\mathsf{Decode}_\sigma(\mathsf{crs}, f, \mathsf{pe}_{\overline{\sigma}}, \mathsf{st}_\sigma)$:

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{sms}}, \mathsf{crs}_{\mathsf{nike}})$
2 : **parse** $\mathsf{pe}_{\overline{\sigma}} = (\mathsf{pe}'_{\overline{\sigma}}, \mathsf{pk}_{\overline{\sigma}})$
3 : $z'_\sigma := \mathsf{SMS.Decode}_\sigma(\mathsf{crs}, \mathsf{pe}'_{\overline{\sigma}}, \mathsf{st}_\sigma)$
4 : $K := \mathsf{NIKE.KeyDer}(\mathsf{pk}_{\overline{\sigma}}, \mathsf{sk}_\sigma)$
5 : $z_\sigma := z'_\sigma \oplus F_K(f)$

Figure 6.13: Generic transformation to simulation-security.

Claim. *The transformed SMS scheme* SMS^* *described in Figure 6.13, when viewed as an interactive protocol between Alice, Bob, and Charlie, securely instantiates* $\mathcal{F}_{\mathsf{sms}}$.

Proof. We consider the three possible cases:

Case 1: Both parties are honest. On input the CRS crs and $f(X, y)$, the simulator generates z_A uniformly at random and defines $z_B := z_A \oplus f(X, y)$ and outputs (z_A, z_B) as the simulated view of the adversary, which matches the output of $\mathcal{F}_{\mathsf{sms}}$. We prove that this simulated view is computationally indistinguishable to the real view of the adversary via a hybrid argument:

- *Hybrid* $\mathcal{H}_0$. This hybrid corresponds to the output (z_A, z_B) in the real view.
- *Hybrid* $\mathcal{H}_1$. In this hybrid, we define $z_B := z_A \oplus f(X, y)$. This hybrid is statistically indistinguishable to the previous one by the correctness property of the underlying SMS scheme.
- *Hybrid* $\mathcal{H}_2$. In this hybrid, the key K is sampled uniformly at random. This hybrid is computationally indistinguishable to the previous one by the security of the NIKE scheme.
- *Hybrid* $\mathcal{H}_3$. In this hybrid, z_A is sampled uniformly at random from $\{0,1\}^m$. This hybrid is computationally indistinguishable to the previous one by the pseudorandomness of the PRF.

At this point, it suffices to note that the distribution in $\mathcal{H}_3$ is identical to the simulated distribution, concluding the proof.

Case 2: Party A is corrupted. Suppose that Alice is corrupted by the adversary $\mathcal{A}$. We construct an efficient simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathsf{sms}}$ to simulate the view of the adversary $\mathcal{A}$, which matches the output of $\mathcal{F}_{\mathsf{sms}}$. We start with the description of $\mathcal{S}$.

$\mathcal{S}$: On input the CRS crs, Alice's input (X, f), and the random coins of $\mathcal{A}$,

- Compute $(\mathsf{pe}_B, \_) \leftarrow \mathsf{SMS.Encode}_B(\mathsf{crs}, 0)$.
- Use the private state st_A of Alice (which is can be obtained from the random coins of $\mathcal{A}$ in the semi-honest setting) to compute $z_A := \mathsf{Decode}_A(\mathsf{crs}, f, \mathsf{pe}_B, \mathsf{st}_A)$.
- Send (X, f, z_A) to $\mathcal{F}_{\mathsf{sms}}$ on behalf of $\mathcal{A}$, and receive $(f(X, y) \oplus z_A, z_A)$ from $\mathcal{F}_{\mathsf{sms}}$.
- Define $z_B := f(X, y) \oplus z_A$.
- Output $(X, f, z_A, \mathsf{pe}_B, z_B)$.

We now show that the view generated by $\mathcal{S}$ is computationally indistinguishable to the view of $\mathcal{A}$ in the real protocol execution via the following sequence of hybrids.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the view $(X, f, z_A, \mathsf{pe}_B, z_B)$ of the adversary $\mathcal{A}$ in the real execution of the protocol, where $\mathcal{A}$ corrupts Alice and plays the role of Charlie.
- *Hybrid* $\mathcal{H}_1$. In this hybrid, we define z_B as $z_B := z_A \oplus f(X, y)$. This hybrid is statistically close to the previous one by the correctness property of the SMS scheme.
- *Hybrid* $\mathcal{H}_2$. In this hybrid, we generate pe_B as the output of $\mathsf{SMS.Encode}_B(\mathsf{crs}, 0)$ and derive z_A using $\mathsf{SMS.Decode}_A$. This hybrid is indistinguishable to the previous one from security the property of the SMS scheme.

At this point, it suffices to note that $\mathcal{H}_2$ is distributed identically to the view generated by $\mathcal{S}$ in the ideal world, which concludes the proof.

Case 3: Party B is corrupted. This case follows by symmetry.

■

6.12 Deferred Proofs

6.12.1 Proof of Proposition 6.6.3

We prove that the real view of the adversary is computationally indistinguishable to a simulated view. First, we describe the simulator $\mathcal{S}$ for pe_B.

$\mathcal{S}$: On input crs,

- Parse $\mathsf{crs} = \mathsf{hk}$.
- Sample $K \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda)$
- Compute $\widetilde{P^{\mathsf{sim}}} \leftarrow \mathsf{iO}(1^\lambda, P^{\mathsf{sim}})$, where P^{sim} is as described in Program 4 with hardcoded inputs (hk, K). Notice that the program does not contain y.
- Output $\mathsf{pe}_B := \widetilde{P^{\mathsf{sim}}}$

Program 4: The Simulated Program

Hardcoded: (hk, K).
Input: $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$.
Procedure:

1: **if** $\widehat{x}_i = \mathsf{Commit}(x_i; r_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}_i, i, \pi_i) = 1$ **then**
2: $\quad R_i := \mathsf{PPRF.Eval}(K, c_{\widehat{X}} \| f \| i)$
3: $\quad$ **return** R_i
4: **else return** $\perp$

We now turn to proving that the output of $\mathcal{S}$ is computationally indistinguishable to pe_B as computed by Encode_B in Figure 6.5.

Notation. Let n denote the domain length (in bits) of the puncturable PRF PPRF and consider the 2^n possible inputs to PPRF. Let $(c_j \| f_j \| i_j)$, parsed as a binary string of length n, denote the j-th canonical input in the domain of the PPRF.

Circuit padding. We assume, without loss of generality, that all obfuscated programs (including Bob's Program 1 and the simulated Program 4) have a polynomial amount of padding added to the circuit so as to make all the obfuscations used in the security proof have the same circuit size as Program 1.

Consider the following sequence of hybrids.

- *Hybrid* $\mathcal{H}_0$. This hybrid consists of the pe_B computed exactly according to Encode_B in Figure 6.5. In particular, pe_B consists of an obfuscation of program P described in Program 1.
- *Hybrid* $\mathcal{H}_{1,j}$. We define $\mathcal{H}_{1,j}$ to be the hybrid distribution where we replace the obfuscation of the program P with an obfuscation of the program $P^{(1,j)}_{\mathsf{hyb}}$. The program $P^{(1,j)}_{\mathsf{hyb}}$ is described in Hybrid 1. Moreover, in this hybrid, we set $\mathsf{hk} \leftarrow \mathsf{SSB.KeyGen}(1^\lambda, L, i_j)$. That is, we make the SSB hash statistically binding on index i_j as parsed from the j-th canonical input $(c_j \| f_j \| i_j)$.

  $P^{(1,j)}_{\mathsf{hyb}}$ has a PPRF master key K_0, along with the j-th canonical input in the PPRF domain (denoted $(c_j \| f_j \| i_j)$), as additional harcoded inputs. Additionally, it uses the output mask R_i, computed as $R_i := \mathsf{PPRF.Eval}(K_0, c_{\widehat{X}} \| f \| i)$, for all inputs $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$ where $(c_{\widehat{X}} \| f \| i)$ is smaller than $(c_j \| f_j \| i_j)$ (the comparison is performed with respect to some arbitrary total ordering assigned to all the inputs in the PPRF domain).

Hybrid 1: (Parameterized by $j \in \{0, 1, \ldots, 2^n\}$ and a universal circuit $\mathcal{U}$)

Hardcoded: $(\mathsf{hk}, K_0, K, y, (c_j \| f_j \| i_j))$.
Input: $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$.
Procedure:

1: **if** $\widehat{x}_i = \mathsf{Commit}(x_i; r_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}_i, i, \pi_i) = 1$ **then**

if $(c_{\widehat{X}} \| f \| i) < (c_j \| f_j \| i_j)$	**if** $(c_{\widehat{X}} \| f \| i) \geq (c_j \| f_j \| i_j)$
1: $R_i := \mathsf{PPRF.Eval}(K_0, c_{\widehat{X}}, f, i)$	1: $R_i := \mathsf{PPRF.Eval}(K, c_{\widehat{X}} \| f \| i)$
2: **return** R	2: $d := \mathcal{U}(f, x_i, y)$
	3: **return** $d \oplus R_i$

2: **else return** $\perp$

Claim. $\mathcal{H}_{1,0} \approx_c \mathcal{H}_0$ *assuming the security of* $i\mathcal{O}$.

Proof. The only difference between $\mathcal{H}_{1,0}$ and $\mathcal{H}_0$ is the inclusion of additional hardcoded inputs since the PPRF key K_0 is not used when $j = 0$. In particular, $P^{(1,0)}_{\mathsf{hyb}}$ and P are functionally equivalent (and of equivalent size due to padding). Indistinguishability thus follows directly from the security of $i\mathcal{O}$. □

- *Hybrid* $\mathcal{H}_{2,j}$. We define $\mathcal{H}_{2,j}$ to be the hybrid distribution where we replace the obfuscation of the program $P^{(1,j)}_{\mathsf{hyb}}$ with an obfuscation of the program $P^{(2,j)}_{\mathsf{hyb}}$. The program $P^{(2,j)}_{\mathsf{hyb}}$ is described in Hybrid 2 and has the hardcoded master PPRF key K replaced with a punctured PPRF key $K^* \leftarrow \mathsf{PPRF.Puncture}(K, (c_j \| f_j \| i_j))$, and additionally has the value $R^* := \mathsf{PPRF.Eval}(K, c_j \| f_j \| i_j)$ as a hardcoded input.

Hybrid 2: (Parameterized by $j \in \{0, 1, \ldots, 2^n\}$ and a universal circuit $\mathcal{U}$)

Hardcoded: $(\mathsf{hk}, K_0, K^*, R^*, y, (c_j \| f_j \| i_j))$.
Input: $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$.
Procedure:

1: **if** $\widehat{x}_i = \mathsf{Commit}(x_i; r_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}_i, i, \pi_i) = 1$ **then**

if $(c_{\widehat{X}} \| f \| i) < (c_j \| f_j \| i_j)$	**if** $(c_{\widehat{X}} \| f \| i) = (c_j \| f_j \| i_j)$	**if** $(c_{\widehat{X}} \| f \| i) > (c_j \| f_j \| i_j)$
1: $R_i := \mathsf{PPRF.Eval}(K_0, c_{\widehat{X}} \| f \| i)$	1: $d := \mathcal{U}(f, x_i, y)$	1: $R_i := \mathsf{PPRF.Eval}(K^*, c_{\widehat{X}} \| f \| i)$
2: **return** R_i	2: **return** $d \oplus R^*$	2: $d := \mathcal{U}(f, x_i, y)$
		3: **return** $d \oplus R_i$

2: **else return** $\perp$

Claim. $\mathcal{H}_{2,j} \approx_c \mathcal{H}_{1,j}$ *assuming the security of* $i\mathcal{O}$.

Proof. Note that the program $P^{(2,j)}_{\mathsf{hyb}}$ outputs the same mask value as the program $P^{(1,j)}_{\mathsf{hyb}}$ on the punctured PPRF input $(c_j \| f_j \| i_j)$, given that $R^* = \mathsf{PPRF.Eval}(K, c_j \| f_j \| i_j)$.

Furthermore, since $\mathsf{PPRF.Eval}(K, \cdot)$ and $\mathsf{PPRF.Eval}(K^*, \cdot)$ agree on all other inputs, the two programs are functionally equivalent. The claim then follows directly from the security of $i\mathcal{O}$. □

- *Hybrid* $\mathcal{H}_{3,j}$. We define $\mathcal{H}_{3,j}$ to be the hybrid distribution where we replace the obfuscation of the program $P^{(2,j)}_{\mathsf{hyb}}$ with an obfuscation of the program $P^{(3,j)}_{\mathsf{hyb}}$. The program $P^{(3,j)}_{\mathsf{hyb}}$ is described in Hybrid 3 and has the hardcoded mask R^* replaced with a uniformly random output R sampled from the support of the PPRF.

Hybrid 3: (Parameterized by $j \in \{0, 1, \ldots, 2^n\}$ and a universal circuit $\mathcal{U}$)

Hardcoded: $(\mathsf{hk}, K_0, K^*, R, y, (c_j \| f_j \| i_j))$.
Input: $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$.
Procedure:

1: **if** $\widehat{x}_i = \mathsf{Commit}(x_i; r_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}_i, i, \pi_i) = 1$ **then**

if $(c_{\widehat{X}} \| f \| i) < (c_j \| f_j \| i_j)$
1: $R_i := \mathsf{PPRF.Eval}(K_0, c_{\widehat{X}} \| f \| i)$
2: **return** R_i

if $(c_{\widehat{X}} \| f \| i) = (c_j \| f_j \| i_j)$
1: $d := \mathcal{U}(f, x_i, y)$
2: **return** $d \oplus R$

if $(c_{\widehat{X}} \| f \| i) > (c_j \| f_j \| i_j)$
1: $R_i := \mathsf{PPRF.Eval}(K^*, c_{\widehat{X}} \| f \| i)$
2: $d := \mathcal{U}(f, x_i, y)$
3: **return** $d \oplus R_i$

2: **else return** $\perp$

Claim. $\mathcal{H}_{3,j} \approx_c \mathcal{H}_{2,j}$ *assuming the security of the PPRF.*

Proof. Notice that any distinguisher between $\mathcal{H}_{3,j} \approx_c \mathcal{H}_{2,j}$ is also a distinguisher for the PPRF security game, given that R^* is distributed identically to the case where the challenger outputs the PPRF evaluation and R is distributed identically to the case where the challenger outputs a uniformly random output. The claim then follows from the security of the PPRF. □

- *Hybrid* $\mathcal{H}_{4,j}$. This hybrid depends on a preprocessing phase.

  1. In the preprocessing phase, the value x_{i_j} is computed by finding any tuple of values $(\widehat{x}_{i_j}, x_{i_j}, \pi_i, r_i)$ such that:

  $$\widehat{x}_{c_j} = \mathsf{Commit}(x_{i_j}; r_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c_j, \widehat{x}_{i_j}, \pi_{i_j}) = 1.$$

  Then, the value d_{i_j} is computed by evaluating $d_{i_j} := \mathcal{U}(f_j, x_{i_j}, y)$.

  2. We then define $\mathcal{H}_{4,j}$ to be the hybrid distribution where we replace the obfuscation of the program $P^{(3,j)}_{\mathsf{hyb}}$ with an obfuscation of the program $P^{(4,j)}_{\mathsf{hyb}}$. The program $P^{(4,j)}_{\mathsf{hyb}}$ is described in Hybrid 4 and has the value $d_j \oplus R$ hardcoded as an input, where R is as defined in $\mathcal{H}_{3,j}$.

Hybrid 4: (Parameterized by $j \in \{0, 1, \ldots, 2^n\}$ and a universal circuit $\mathcal{U}$)

Hardcoded: $(\mathsf{hk}, K_0, K^*, d_j \oplus R, y, (c_j \| f_j \| i_j))$.
Input: $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$.
Procedure:

1: **if** $\widehat{x}_i = \mathsf{Commit}(x_i; r_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}_i, i, \pi_i) = 1$ **then**

if $(c_{\widehat{X}}, f, i) < (c_j \| f_j \| i_j)$
1: $R_i := \mathsf{PPRF.Eval}(K_0, c_{\widehat{X}} \| f \| i)$
2: **return** R_i

if $(c_{\widehat{X}}, f, i) = (c_j \| f_j \| i_j)$
1: **return** $d_j \oplus R$

if $(c_{\widehat{X}} \| f \| i) > (c_j \| f_j \| i_j)$
1: $R_i := \mathsf{PPRF.Eval}(K^*, c_{\widehat{X}} \| f \| i)$
2: $d := \mathcal{U}(f, x_i, y)$
3: **return** $d \oplus R_i$

2: **else return** $\perp$

Claim. $\mathcal{H}_{4,j} \approx_c \mathcal{H}_{3,j}$ *assuming the security of* $i\mathcal{O}$*, the somewhere* perfect *binding of the SSB hash function, and the perfect binding of the commitment scheme.*

Proof. At a high level, we prove that there is only one input to the program $P^{(3,j)}_{\mathsf{hyb}}$ that makes it output $d \oplus R$ at the punctured input. Then, because $d_j \oplus R$ is hardcoded in $P^{(3,j)}_{\mathsf{hyb}}$, and is identical to the output on the punctured input in $P^{(4,j)}_{\mathsf{hyb}}$, we can invoke the security of $i\mathcal{O}$ to finish proving the claim.

Formally, suppose, towards contradiction, that there exists a pair of inputs on which $P^{(3,j)}_{\mathsf{hyb}}$ outputs $d \oplus R$ and $d' \oplus R$, respectively, using the same hardcoded value of R and some $d \neq d'$.[8] Let this pair of inputs be:

$$(c_{\widehat{X}}, x, (\widehat{x}, r), i, \pi, f) \neq (c'_{\widehat{X}}, x', (\widehat{x}', r'), i', \pi', f').$$

By inspection of $P^{(3,j)}_{\mathsf{hyb}}$ (described in Hybrid 3), it is clear that if these inputs produce outputs $d \oplus R$ and $d' \oplus R$, respectively, then the following three conditions must hold simultaneously:

(1) $(c_{\widehat{X}}, f, i) = (c_j, f_j, i_j) = (c'_{\widehat{X}}, f', i')$, ▷ Otherwise, R is not used.
(2) $\widehat{x} = \mathsf{Commit}(x; r) \wedge \widehat{x}' = \mathsf{Commit}(x'; r')$, and ▷ Otherwise, the output is $\perp$.
(3) $\mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}, i, \pi) = \mathsf{SSB.Verify}(\mathsf{hk}, c'_{\widehat{X}}, \widehat{x}', i', \pi') = 1$.

By (1) we have that $f = f'$, and so it must be the case that $x \neq x'$ given that $d = f(x, y) \neq f(x', y) = d'$. Moreover, because Commit is perfectly binding, we also have that $\widehat{x} \neq \widehat{x}'$. Then, using (1), we can rewrite (3) as:

$$\mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}, i, \pi) = \mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}', i, \pi') = 1.$$

This implies that there exist at least two openings $\widehat{x}' \neq \widehat{x}$ for the same index i, such that

[8] While there may be other inputs that produce collisions in the PPRF outputs such that $R_i = R$, we only need to examine the case where the *hardcoded* value R is used twice. For all other inputs, the two programs behave identically.

$\mathsf{SSB.Verify}$ accepts under the same hash key hk and hash $c_{\widehat{X}}$. However, in $P^{(3,j)}_{\mathsf{hyb}}$, the SSB hash hk was set to be perfectly binding on index $i_j = i$, which raises a contradiction. Intuitively, the PPRF evaluation forces $c_{\widehat{X}}$ and i to be consistent with $c'_{\widehat{X}}$ and i'.

At this point, we conclude that the hardcoded value R is only used to mask a single output value d in $P^{(3,j)}_{\mathsf{hyb}}$. Moreover, by the analysis above, the output value d_j (as computed in the preprocessing phase of $P^{(4,j)}_{\mathsf{hyb}}$) must be equal to d (as output by $P^{(3,j)}_{\mathsf{hyb}}$). To see this, first note that $c_{\widehat{X}} = c_j$ since otherwise d is not output. Then, by our analysis above, we have that the inputs $x_{i_j} = x$ and $f_j = f$ are uniquely determined by $c_j = c_{\widehat{X}}$. So the preprocessing outputs $d_j = d$, as output by $P^{(3,j)}_{\mathsf{hyb}}$ on the punctured input.

It follows that $P^{(4,j)}_{\mathsf{hyb}}$ is functionally equivalent to $P^{(3,j)}_{\mathsf{hyb}}$ on the punctured input. Since the two programs also agree on all other inputs, the two programs are therefore functionally equivalent and the claim follows from the security of $i\mathcal{O}$ against a non-uniform distinguishing adversary that is given the preprocessing as non-uniform advice. □

- *Hybrid* $\mathcal{H}_{5,j}$. We define $\mathcal{H}_{5,j}$ to be the hybrid distribution where we replace the obfuscation of the program $P^{(4,j)}_{\mathsf{hyb}}$ with an obfuscation of the program $P^{(5,j)}_{\mathsf{hyb}}$. The program $P^{(5,j)}_{\mathsf{hyb}}$ is described in Hybrid 5 and has the hardcoded master PPRF key K_0 replaced with a punctured PPRF key $K^*_0 \leftarrow \mathsf{PPRF.Puncture}(K_0, (c_j \| f_j \| i_j))$.

Hybrid 5: (Parameterized by $j \in \{0, 1, \ldots, 2^n\}$ and a universal circuit $\mathcal{U}$)

Hardcoded: $(\mathsf{hk}, K^*_0, K^*, d_j \oplus R, y, (c_j \| f_j \| i_j))$.
Input: $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$.
Procedure:

1: **if** $\widehat{x}_i = \mathsf{Commit}(x_i; r_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}_i, i, \pi_i) = 1$ **then**

if $(c_{\widehat{X}}, f, i) < (c_j \| f_j \| i_j)$
1: $R_i := \mathsf{PPRF.Eval}(K^*_0, c_{\widehat{X}} \| f \| i)$
2: **return** R_i

if $(c_{\widehat{X}}, f, i) = (c_j \| f_j \| i_j)$
1: **return** $d_j \oplus R$

if $(c_{\widehat{X}} \| f \| i) > (c_j \| f_j \| i_j)$
1: $R_i := \mathsf{PPRF.Eval}(K^*, c_{\widehat{X}} \| f \| i)$
2: $d := \mathcal{U}(f, x_i, y)$
3: **return** $d \oplus R_i$

2: **else return** $\perp$

Claim. $\mathcal{H}_{5,j} \approx_c \mathcal{H}_{4,j}$ *assuming the security of* $i\mathcal{O}$.

Proof. The master PPRF key K_0 is never used to evaluate the punctured input in $\mathcal{H}_{4,j}$ and so we can conclude the two programs are functionally equivalent. The claim follows from the security of $i\mathcal{O}$ against non-uniform distinguishing adversaries (we still require the preprocessing to compute $d_j \oplus R$ as non-uniform advice). □

- *Hybrid* $\mathcal{H}_{6,j}$. We define $\mathcal{H}_{6,j}$ to be the hybrid distribution where we replace the obfuscation of the program $P^{(5,j)}_{\mathsf{hyb}}$ with an obfuscation of the program $P^{(6,j)}_{\mathsf{hyb}}$. The program $P^{(6,j)}_{\mathsf{hyb}}$ is described in Hybrid 6 and has the hardcoded output $d_j \oplus R$ replaced with the value $R^* := \mathsf{PPRF.Eval}(K_0, c_j \| f_j \| i_j)$, computed using the PPRF master key K_0.

Hybrid 6: (Parameterized by $j \in \{0, 1, \ldots, 2^n\}$ and a universal circuit $\mathcal{U}$)

Hardcoded: $(\mathsf{hk}, K_0^*, K^*, R^*, y, (c_j \| f_j \| i_j))$.
Input: $(c_{\widehat{X}}, x_i, (\widehat{x}_i, r_i), i, \pi_i, f)$.
Procedure:

1: **if** $\widehat{x}_i = \mathsf{Commit}(x_i; r_i) \wedge \mathsf{SSB.Verify}(\mathsf{hk}, c_{\widehat{X}}, \widehat{x}_i, i, \pi_i) = 1$ **then**

if $(c_{\widehat{X}}, f, i) < (c_j \| f_j \| i_j)$
1: $R_i := \mathsf{PPRF.Eval}(K_0^*, c_{\widehat{X}}, f, i)$
2: **return** R_i

if $(c_{\widehat{X}}, f, i) = (c_j \| f_j \| i_j)$
1: **return** R^*

if $(c_{\widehat{X}} \| f \| i) > (c_j \| f_j \| i_j)$
1: $R_i := \mathsf{PPRF.Eval}(K^*, c_{\widehat{X}} \| f \| i)$
2: $d := \mathcal{U}(f, x_i, y)$
3: **return** $d \oplus R_i$

2: **else return** $\perp$

Claim. $\mathcal{H}_{6,j} \approx_c \mathcal{H}_{5,j}$ *assuming the security of the PPRF.*

Proof. The proof is almost identical to the proof for $\mathcal{H}_{3,j} \approx_c \mathcal{H}_{2,j}$. In particular, observe that $d_j \oplus R$ is distributed as a uniformly random string, by the fact that R is uniformly random. As such, indistinguishability is implied by the puncturing security of the PPRF. □

- *Hybrid* $\mathcal{H}_{7,j}$. We define $\mathcal{H}_{7,j}$ to be the hybrid distribution where we set

$$\mathsf{hk} \leftarrow \mathsf{SSB.KeyGen}(1^\lambda, L, i_{j+1}).$$

That is, we make the SSB hash statistically binding on index i_j, as parsed from the $(j+1)$-st canonical input $(c_{j+1} \| f_{j+1} \| i_{j+1})$.

Claim. $\mathcal{H}_{7,j} \approx_c \mathcal{H}_{6,j}$ *assuming the index-hiding of the SSB hash function.*

Proof. On the one hand, if the switch from $\mathcal{H}_{6,j}$ to $\mathcal{H}_{7,j}$ has $i_j = i_{j+1}$ (recall that we are switching over from one canonical *input* to the next, which may not change the value of i_j and i_{j+1}), the claim follows trivially. On the other hand, if the switch from $\mathcal{H}_{6,j}$ to $\mathcal{H}_{7,j}$ has $i_j \neq i_{j+1}$, then the claim follows directly from the index-hiding property of the SSB hash. □

Claim. $\mathcal{H}_{1,j+1} \approx_c \mathcal{H}_{7,j}$ *assuming the security of* $i\mathcal{O}$.

Proof. The claim follows immediately by noticing that the two programs compute identical values at the punctured input, making them functionally equivalent. The claim then follows directly from the security of $i\mathcal{O}$. □

Lemma 6.12.1. $\mathcal{H}_{1,2^n} \approx_c \mathcal{H}_{1,0}$ *assuming the sub-exponential security of* $i\mathcal{O}$*, the existence of sub-exponentially secure one-way functions, the existence of injective one-way functions (for perfectly-binding commitments), and the security of somewhere statistical binding hash functions with (perfect) binding.*

Proof. First, following Section 6.6.3, we can complexity leverage by assuming sub-exponential security of $i\mathcal{O}$ and one-way functions. Then, we have that $\mathcal{H}_{1,j+1} \approx_c \mathcal{H}_{7,j}$ and $\mathcal{H}_{7,j} \approx_c \mathcal{H}_{1,j}$, which implies that $\mathcal{H}_{1,j} \approx_c \mathcal{H}_{1,j+1}$. It then follows from Section 6.6.3 that we can set parameters such that there does not exist an efficient distinguisher $\mathcal{D}$ with a sub-exponential distinguishing advantage between hybrids $\mathcal{H}_{1,0}$ and $\mathcal{H}_{1,2^n}$.

Finally, we note that injective one-way functions give us perfectly-binding commitment schemes (cf. Definition 6.6.4). This concludes the proof of the lemma. □

Corollary 6.12.1. $\mathcal{S}(\mathsf{crs}) \approx_c \mathcal{H}_{1,2^n}$ *assuming the security of* $i\mathcal{O}$*.*

Proof. Indistinguishability follows from the fact that the program $P^{(1,2^n)}_{\mathsf{hyb}}$ (cf. Hybrid 1) does not use the hardcoded input y at all which, by the security of $i\mathcal{O}$, makes the obfuscation of $P^{(1,2^n)}_{\mathsf{hyb}}$ computationally indistinguishable to the obfuscation of P^{sim}. This concludes the proof of the corollary. □

This concludes the proof of Proposition 6.6.3. ■

Remark 48 (On the security of the SSB hash)**.** *Note that though the above proof relies on* 2^n *hybrids, the number of times that index hiding security of SSB hashing is invoked is only* L *(which is the batch size). Therefore, it is sufficient to rely on polynomially-secure SSB hashing rather than a sub-exponential secure version.*

Bibliography

[ABC+24] G. Alagic, M. Bros, P. Ciadoux, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, H. Silberg, D. Smith-Tone, and N. Waller. Status report on the first round of the additional digital signature schemes for the NIST post-quantum cryptography standardization process. NIST Internal Report NIST IR 8528, National Institute of Standards and Technology, Gaithersburg, MD, October 2024.

[ABPP14] M. Abdalla, F. Benhamouda, A. Passelègue, and K. G. Paterson. Related-key security for pseudorandom functions beyond the linear barrier. In *CRYPTO 2014, Part I*, *LNCS* 8616, pages 77–94. Springer, Berlin, Heidelberg, August 2014.

[ABPP18] M. Abdalla, F. Benhamouda, A. Passelègue, and K. G. Paterson. Related-key security for pseudorandom functions beyond the linear barrier. *Journal of Cryptology*, 31(4):917–964, October 2018.

[ACC+18] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, 2018.

[ADI+17] B. Applebaum, I. Damgård, Y. Ishai, M. Nielsen, and L. Zichron. Secure arithmetic computation with constant computational overhead. In *CRYPTO 2017, Part I*, *LNCS* 10401, pages 223–254. Springer, Cham, August 2017.

[ADOS22] D. Abram, I. Damgård, C. Orlandi, and P. Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In *CRYPTO 2022, Part IV*, *LNCS* 13510, pages 421–452. Springer, Cham, August 2022.

[AHI11] B. Applebaum, D. Harnik, and Y. Ishai. Semantic security under related-key attacks and applications. In *ICS 2011*, pages 45–60. Tsinghua University Press, January 2011.

[AJJM20] P. Ananth, A. Jain, Z. Jin, and G. Malavolta. Multi-key fully-homomorphic encryption in the plain model. In *TCC 2020, Part I*, *LNCS* 12550, pages 28–57. Springer, Cham, November 2020.

[AJJM21] P. Ananth, A. Jain, Z. Jin, and G. Malavolta. Unbounded multi-party computation from learning with errors. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 754–781. Springer, Cham, October 2021.

[AJL+12] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT 2012*, *LNCS* 7237, pages 483–501. Springer, Berlin, Heidelberg, April 2012.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996.

[AL16] B. Applebaum and S. Lovett. Algebraic attacks against random local functions and their countermeasures. In *48th ACM STOC*, pages 1087–1100. ACM Press, June 2016.

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS 2013*, pages 535–548. ACM Press, November 2013.

[ALSZ15] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *EUROCRYPT 2015, Part I*, *LNCS* 9056, pages 673–701. Springer, Berlin, Heidelberg, April 2015.

[AMN+18] N. Attrapadung, T. Matsuda, R. Nishimaki, S. Yamada, and T. Yamakawa. Constrained PRFs for NC^1 in traditional groups. In *CRYPTO 2018, Part II*, *LNCS* 10992, pages 543–574. Springer, Cham, August 2018.

[AMN+19] N. Attrapadung, T. Matsuda, R. Nishimaki, S. Yamada, and T. Yamakawa. Adaptively single-key secure constrained PRFs for NC^1. In *PKC 2019, Part II*, *LNCS* 11443, pages 223–253. Springer, Cham, April 2019.

[AR16] B. Applebaum and P. Raykov. Fast pseudorandom functions based on expander graphs. In *TCC 2016-B, Part I*, *LNCS* 9985, pages 27–56. Springer, Berlin, Heidelberg, October / November 2016.

[ARS24] D. Abram, L. Roy, and P. Scholl. Succinct homomorphic secret sharing. In *EUROCRYPT 2024, Part VI*, *LNCS* 14656, pages 301–330. Springer, Cham, May 2024.

[AS22] D. Abram and P. Scholl. Low-communication multiparty triple generation for SPDZ from ring-LPN. In *PKC 2022, Part I*, *LNCS* 13177, pages 221–251. Springer, Cham, March 2022.

[AW14] B. Applebaum and E. Widder. Related-key secure pseudorandom functions: The case of additive attacks. Cryptology ePrint Archive, Report 2014/478, 2014.

[BBB+22] A. A. Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R. V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Report 2022/915, 2022.

[BBC+21] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776. IEEE Computer Society Press, May 2021.

[BBC+24] M. Bombar, D. Bui, G. Couteau, A. Couvreur, C. Ducros, and S. Servan-Schreiber. FOLEAGE: $\mathbb{F}_4$OLE-based multi-party computation for boolean circuits. In *ASIACRYPT 2024, Part VI, LNCS 15489*, pages 69–101. Springer, Singapore, 2024.

[BBD+23] C. Baum, L. Braun, C. Delpech de Saint Guilhem, M. Klooß, E. Orsini, L. Roy, and P. Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In *CRYPTO 2023, Part V*, *LNCS* 14085, pages 581–615. Springer, Cham, August 2023.

[BBMHS22] C. Baum, L. Braun, A. Munch-Hansen, and P. Scholl. Moz$\mathbb{Z}_{2^k}$arella: Efficient vector-OLE and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. In *CRYPTO 2022, Part IV*, *LNCS* 13510, pages 329–358. Springer, Cham, August 2022.

[BC10] M. Bellare and D. Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In *CRYPTO 2010*, *LNCS* 6223, pages 666–684. Springer, Berlin, Heidelberg, August 2010.

[BCC+23] G. Banegas, K. Carrier, A. Chailloux, A. Couvreur, T. Debris-Alazard, P. Gaborit, P. Karpman, J. Loyer, R. Niederhagen, N. Sendrier, B. Smith, and J.-P. Tilich. WAVE: Round 1 submission. https://wave-sign.org/wave_documentation.pdf, 2023. Accessed on January 24, 2025.

[BCCD23] M. Bombar, G. Couteau, A. Couvreur, and C. Ducros. Correlated pseudorandomness from the hardness of quasi-abelian decoding. In *CRYPTO 2023, Part IV*, *LNCS* 14084, pages 567–601. Springer, Cham, August 2023.

[BCCS24] A. Bondarchuk, O. Chakraborty, G. Couteau, and R. Sirdey. Downlink (t)FHE ciphertexts compression. Cryptology ePrint Archive, Paper 2024/1921, 2024.

[BCE+24] C. Brzuska, G. Couteau, C. Egger, P. Karanko, and P. Meyer. Instantiating the hash-then-evaluate paradigm: Strengthening PRFs, PCFs, and OPRFs. In *SCN 24, Part II*, *LNCS* 14974, pages 97–116. Springer, Cham, September 2024.

[BCG+14] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[BCG+17] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, and M. Orrù. Homomorphic secret sharing: Optimizations and applications. In *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017.

[BCG+19a] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[BCG+19b] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 489–518. Springer, Cham, August 2019.

[BCG+20a] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.

[BCG+20b] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators from ring-LPN. In *CRYPTO 2020, Part II*, *LNCS* 12171, pages 387–416. Springer, Cham, August 2020.

[BCG+21] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 871–900. Springer, Cham, October 2021.

[BCG+22] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, N. Resch, and P. Scholl. Correlated pseudorandomness from expand-accumulate codes. In *CRYPTO 2022, Part II*, *LNCS* 13508, pages 603–633. Springer, Cham, August 2022.

[BCGI18] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. Compressing vector OLE. In *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BCM23] E. Boyle, G. Couteau, and P. Meyer. Sublinear-communication secure multiparty computation does not require FHE. In *EUROCRYPT 2023, Part II*, *LNCS* 14005, pages 159–189. Springer, Cham, April 2023.

[BCM+24] D. Bui, G. Couteau, P. Meyer, A. Passelègue, and M. Riahinia. Fast public-key silent OT and more from constrained Naor-Reingold. In *EUROCRYPT 2024, Part VI*, *LNCS* 14656, pages 88–118. Springer, Cham, May 2024.

[BCP03] E. Bresson, D. Catalano, and D. Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *ASIACRYPT 2003*, *LNCS* 2894, pages 37–54. Springer, Berlin, Heidelberg, November / December 2003.

[BCPW15] F. Benhamouda, G. Couteau, D. Pointcheval, and H. Wee. Implicit zero-knowledge arguments and applications to the malicious setting. In *CRYPTO 2015, Part II*, *LNCS* 9216, pages 107–129. Springer, Berlin, Heidelberg, August 2015.

[BDGM19] Z. Brakerski, N. Döttling, S. Garg, and G. Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In *TCC 2019, Part II*, *LNCS* 11892, pages 407–437. Springer, Cham, December 2019.

[BDK+11] B. Barak, Y. Dodis, H. Krawczyk, O. Pereira, K. Pietrzak, F.-X. Standaert, and Y. Yu. Leftover hash lemma, revisited. In *CRYPTO 2011*, *LNCS* 6841, pages 1–20. Springer, Berlin, Heidelberg, August 2011.

[BDSS25] E. Boyle, L. Devadas, and S. Servan-Schreiber. Non-interactive distributed point functions. Cryptology ePrint Archive, Paper 2025/095, 2025.

[BDSZ24] P. Branco, N. Döttling, A. Srinivasan, and R. Zanotto. Rate-1 fully local somewhere extractable hashing from DDH. In *PKC 2024, Part II*, *LNCS* 14603, pages 356–386. Springer, Cham, April 2024.

[Bea95] D. Beaver. Precomputing oblivious transfer. In *CRYPTO'95*, *LNCS* 963, pages 97–109. Springer, Berlin, Heidelberg, August 1995.

[Bea96] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

[BGG+14] D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT 2014*, *LNCS* 8441, pages 533–556. Springer, Berlin, Heidelberg, May 2014.

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO 2001*, *LNCS* 2139, pages 1–18. Springer, Berlin, Heidelberg, August 2001.

[BGI14] E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *PKC 2014*, *LNCS* 8383, pages 501–519. Springer, Berlin, Heidelberg, March 2014.

[BGI15] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In *EUROCRYPT 2015, Part II*, *LNCS* 9057, pages 337–367. Springer, Berlin, Heidelberg, April 2015.

[BGI16] E. Boyle, N. Gilboa, and Y. Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO 2016, Part I*, *LNCS* 9814, pages 509–539. Springer, Berlin, Heidelberg, August 2016.

[BGI17] E. Boyle, N. Gilboa, and Y. Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In *EUROCRYPT 2017, Part II*, *LNCS* 10211, pages 163–193. Springer, Cham, April / May 2017.

[BGI+18] E. Boyle, N. Gilboa, Y. Ishai, H. Lin, and S. Tessaro. Foundations of homomorphic secret sharing. In *ITCS 2018*, pages 21:1–21:21. LIPIcs, January 2018.

[BGI19] E. Boyle, N. Gilboa, and Y. Ishai. Secure computation with preprocessing via function secret sharing. In *TCC 2019, Part I*, *LNCS* 11891, pages 341–371. Springer, Cham, December 2019.

[BGIK22] E. Boyle, N. Gilboa, Y. Ishai, and V. I. Kolobov. Programmable distributed point functions. In *CRYPTO 2022, Part IV*, *LNCS* 13510, pages 121–151. Springer, Cham, August 2022.

[BGMM20] J. Bartusek, S. Garg, D. Masny, and P. Mukherjee. Reusable two-round MPC from DDH. In *TCC 2020, Part II*, *LNCS* 12551, pages 320–348. Springer, Cham, November 2020.

[BGV12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, pages 309–325. ACM, January 2012.

[Bih94] E. Biham. New types of cryptanalytic attacks using related keys. *Journal of Cryptology*, 7(4):229–246, December 1994.

[BIP+18] D. Boneh, Y. Ishai, A. Passelègue, A. Sahai, and D. J. Wu. Exploring crypto dark matter: New simple PRF candidates and their applications. In *TCC 2018, Part II*, *LNCS* 11240, pages 699–729. Springer, Cham, November 2018.

[BJKL21] F. Benhamouda, A. Jain, I. Komargodski, and H. Lin. Multiparty reusable non-interactive secure computation from LWE. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 724–753. Springer, Cham, October 2021.

[BJSSS25] E. Boyle, A. Jain, S. Servan-Schreiber, and A. Srinivasan. Simultaneous-message and succinct secure computation. Cryptology ePrint Archive, Paper 2025/096, 2025.

[BK03] M. Bellare and T. Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *EUROCRYPT 2003*, *LNCS* 2656, pages 491–506. Springer, Berlin, Heidelberg, May 2003.

[BKM17] D. Boneh, S. Kim, and H. W. Montgomery. Private puncturable PRFs from standard lattice assumptions. In *EUROCRYPT 2017, Part I*, *LNCS* 10210, pages 415–445. Springer, Cham, April / May 2017.

[BKM20] Z. Brakerski, V. Koppula, and T. Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In *CRYPTO 2020, Part III*, *LNCS* 12172, pages 738–767. Springer, Cham, August 2020.

[BKS19] E. Boyle, L. Kohl, and P. Scholl. Homomorphic secret sharing from lattices without FHE. In *EUROCRYPT 2019, Part II*, *LNCS* 11477, pages 3–33. Springer, Cham, May 2019.

[BL17] D. J. Bernstein and T. Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, 2017.

[BL18] F. Benhamouda and H. Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *EUROCRYPT 2018, Part II*, *LNCS* 10821, pages 500–532. Springer, Cham, April / May 2018.

[BL20] F. Benhamouda and H. Lin. Mr NISC: Multiparty reusable non-interactive secure computation. In *TCC 2020, Part II*, *LNCS* 12551, pages 349–378. Springer, Cham, November 2020.

[BLMR13] D. Boneh, K. Lewi, H. W. Montgomery, and A. Raghunathan. Key homomorphic PRFs and their applications. In *CRYPTO 2013, Part I*, *LNCS* 8042, pages 410–428. Springer, Berlin, Heidelberg, August 2013.

[BLW17] D. Boneh, K. Lewi, and D. J. Wu. Constraining pseudorandom functions privately. In *PKC 2017, Part II*, *LNCS* 10175, pages 494–524. Springer, Berlin, Heidelberg, March 2017.

[BM90] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *CRYPTO'89*, *LNCS* 435, pages 547–557. Springer, New York, August 1990.

[BMO17] R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *ACM CCS 2017*, pages 1465–1482. ACM Press, October / November 2017.

[BOV03] B. Barak, S. J. Ong, and S. P. Vadhan. Derandomization in cryptography. In *CRYPTO 2003*, *LNCS* 2729, pages 299–315. Springer, Berlin, Heidelberg, August 2003.

[BP12] O. Blazy and D. Pointcheval. Traceable signature with stepping capabilities. In *Cryptography and Security: From Theory to Applications: Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, pages 108–131. Springer, 2012.

[BPR12] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *EUROCRYPT 2012*, *LNCS* 7237, pages 719–737. Springer, Berlin, Heidelberg, April 2012.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[BTVW17] Z. Brakerski, R. Tsabary, V. Vaikuntanathan, and H. Wee. Private constrained PRFs (and more) from LWE. In *TCC 2017, Part I*, *LNCS* 10677, pages 264–302. Springer, Cham, November 2017.

[BV11] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[BV15] Z. Brakerski and V. Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *TCC 2015, Part II*, *LNCS* 9015, pages 1–30. Springer, Berlin, Heidelberg, March 2015.

[BW13] D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT 2013, Part II*, *LNCS* 8270, pages 280–300. Springer, Berlin, Heidelberg, December 2013.

[BZ14] D. Boneh and M. Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO 2014, Part I*, *LNCS* 8616, pages 480–499. Springer, Berlin, Heidelberg, August 2014.

[CBM15] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society Press, May 2015.

[CC17] R. Canetti and Y. Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In *EUROCRYPT 2017, Part I*, *LNCS* 10210, pages 446–476. Springer, Cham, April / May 2017.

[CCH+19] R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, R. D. Rothblum, and D. Wichs. Fiat-Shamir: from practice to theory. In *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019.

[CD23] G. Couteau and C. Ducros. Pseudorandom correlation functions from variable-density LPN, revisited. In *PKC 2023, Part II*, *LNCS* 13941, pages 221–250. Springer, Cham, May 2023.

[CDD+24] G. Couteau, L. Devadas, S. Devadas, A. Koch, and S. Servan-Schreiber. QuietOT: Lightweight oblivious transfer with a public-key setup. In *ASIACRYPT 2024, Part II, LNCS 15485*, pages 197–231. Springer, Singapore, 2024.

[CDG+17] C. Cho, N. Döttling, S. Garg, D. Gupta, P. Miao, and A. Polychroniadou. Laconic oblivious transfer and its applications. In *CRYPTO 2017, Part II*, *LNCS* 10402, pages 33–65. Springer, Cham, August 2017.

[CDH+25] G. Couteau, L. Devadas, A. Hegde, A. Jain, and S. Servan-Schreiber. Multi-key homomorphic secret sharing. Cryptology ePrint Archive, Paper 2025/094, 2025.

[CDM+18] G. Couteau, A. Dupin, P. Méaux, M. Rossi, and Y. Rotella. On the concrete security of Goldreich's pseudorandom generator. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 96–124. Springer, Cham, December 2018.

[CF01] R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO 2001*, *LNCS* 2139, pages 19–40. Springer, Berlin, Heidelberg, August 2001.

[CGB17] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 259–282, Boston, MA, March 2017. USENIX Association.

[CGH98] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

[CGV15] A. Cohen, S. Goldwasser, and V. Vaikuntanathan. Aggregate pseudorandom functions and connections to learning. In *TCC 2015, Part II*, *LNCS* 9015, pages 61–89. Springer, Berlin, Heidelberg, March 2015.

[CHK+19] A. R. Choudhuri, P. Hubácek, C. Kamath, K. Pietrzak, A. Rosen, and G. N. Rothblum. Finding a nash equilibrium is no easier than breaking Fiat-Shamir. In *51st ACM STOC*, pages 1103–1114. ACM Press, June 2019.

[CJJ21] A. R. Choudhuri, A. Jain, and Z. Jin. Non-interactive batch arguments for NP from standard assumptions. In *CRYPTO 2021, Part IV*, *LNCS* 12828, pages 394–423, Virtual Event, August 2021. Springer, Cham.

[CJJ22] A. R. Choudhuri, A. Jain, and Z. Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, February 2022.

[CK24] G. Couteau and N. Kumar. 10-party sublinear secure computation from standard assumptions. In *CRYPTO 2024, Part IX*, *LNCS* 14928, pages 39–73. Springer, Cham, August 2024.

[CKS08] D. Cash, E. Kiltz, and V. Shoup. The twin Diffie-Hellman problem and applications. In *EUROCRYPT 2008*, *LNCS* 4965, pages 127–145. Springer, Berlin, Heidelberg, April 2008.

[Cle90] R. Cleve. Towards optimal simulations of formulas by bounded-width programs. In *22nd ACM STOC*, pages 271–277. ACM Press, May 1990.

[CLP17] H. Chen, K. Laine, and R. Player. Simple encrypted arithmetic library - SEAL v2.1. In *FC 2017 Workshops*, *LNCS* 10323, pages 3–18. Springer, Cham, April 2017.

[CLWG19] N. Cui, S. Liu, Y. Wen, and D. Gu. Pseudorandom functions from LWE: RKA security and application. In *ACISP 19*, *LNCS* 11547, pages 229–250. Springer, Cham, July 2019.

[CM19] J. Chen and S. Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[CM21] G. Couteau and P. Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 842–870. Springer, Cham, October 2021.

[CMPR23] G. Couteau, P. Meyer, A. Passelègue, and M. Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In *EUROCRYPT 2023, Part III*, *LNCS* 14006, pages 194–224. Springer, Cham, April 2023.

[COSW23] M. Ciampi, R. Ostrovsky, L. Siniscalchi, and H. Waldner. List oblivious transfer and applications to round-optimal black-box multiparty coin tossing. In *CRYPTO 2023*, number 14081 in LNCS, pages 459–488. Springer, 2023.

[Cou19] G. Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In *EUROCRYPT 2019, Part II*, *LNCS* 11477, pages 473–503. Springer, Cham, May 2019.

[CRR21] G. Couteau, P. Rindal, and S. Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *CRYPTO 2021, Part III*, *LNCS* 12827, pages 502–534, Virtual Event, August 2021. Springer, Cham.

[CRV16] N. Chandran, S. Raghuraman, and D. Vinayagamurthy. Reducing depth in constrained PRFs: From bit-fixing to $\mathbf{NC}^1$. In *PKC 2016, Part II*, *LNCS* 9615, pages 359–385. Springer, Berlin, Heidelberg, March 2016.

[CS02] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT 2002*, *LNCS* 2332, pages 45–64. Springer, Berlin, Heidelberg, April / May 2002.

[CVW18] Y. Chen, V. Vaikuntanathan, and H. Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In *CRYPTO 2018, Part II*, *LNCS* 10992, pages 577–607. Springer, Cham, August 2018.

[CZ22] G. Couteau and M. Zarezadeh. Non-interactive secure computation of inner-product from LPN and LWE. In *ASIACRYPT 2022, Part I*, *LNCS* 13791, pages 474–503. Springer, Cham, December 2022.

[Dam88] I. Damgård. Collision free hash functions and public key signature schemes. In *EUROCRYPT'87*, *LNCS* 304, pages 203–216. Springer, Berlin, Heidelberg, April 1988.

[dCJV21] L. de Castro, C. Juvekar, and V. Vaikuntanathan. Fast vector oblivious linear evaluation from ring learning with errors. In *WAHC 2021: Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 29–41, 2021.

[DFL+20] E. Dauterman, E. Feng, E. Luo, R. A. Popa, and I. Stoica. DORY: An encrypted search system with distributed trust. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1101–1119, 2020.

[DGI+19] N. Döttling, S. Garg, Y. Ishai, G. Malavolta, T. Mour, and R. Ostrovsky. Trapdoor hash functions and their applications. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 3–32. Springer, Cham, August 2019.

[DGKV22] L. Devadas, R. Goyal, Y. Kalai, and V. Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *63rd FOCS*, pages 1057–1068. IEEE Computer Society Press, October / November 2022.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DHRW16] Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. Spooky encryption and its applications. In *CRYPTO 2016, Part III*, *LNCS* 9816, pages 93–122. Springer, Berlin, Heidelberg, August 2016.

[DIJL23] Q. Dao, Y. Ishai, A. Jain, and H. Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In *CRYPTO 2023, Part II*, *LNCS* 14082, pages 315–348. Springer, Cham, August 2023.

[DJ03] I. Damgård and M. Jurik. A length-flexible threshold cryptosystem with applications. In *ACISP 03*, *LNCS* 2727, pages 350–364. Springer, Berlin, Heidelberg, July 2003.

[DKK20] I. Dinur, N. Keller, and O. Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. *Journal of Cryptology*, 33(3):824–873, July 2020.

[DKN+20] A. Davidson, S. Katsumata, R. Nishimaki, S. Yamada, and T. Yamakawa. Adaptively secure constrained pseudorandom functions in the standard model. In *CRYPTO 2020, Part I*, *LNCS* 12170, pages 559–589. Springer, Cham, August 2020.

[DPSZ12] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*, *LNCS* 7417, pages 643–662. Springer, Berlin, Heidelberg, August 2012.

[DRPS22] E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica. Waldo: A private time-series database from function secret sharing. In *2022 IEEE Symposium on Security and Privacy*, pages 2450–2468. IEEE Computer Society Press, May 2022.

[DRWY12] Z. Dvir, A. Rao, A. Wigderson, and A. Yehudayoff. Restriction access. In *ITCS 2012*, pages 19–33. ACM, January 2012.

[Ds17] J. Doerner and a. shelat. Scaling ORAM for secure computation. In *ACM CCS 2017*, pages 523–535. ACM Press, October / November 2017.

[Eli91] P. Elias. Error-correcting codes for list decoding. *IEEE Transactions on Information Theory*, 37(1):5–12, 1991.

[Fen23] T. Feneuil. *Post-Quantum Signatures from Secure Multiparty Computation*. PhD thesis, Sorbonne Université, 2023.

[FHKP13] E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson. Non-interactive key exchange. In *PKC 2013*, *LNCS* 7778, pages 254–271. Springer, Berlin, Heidelberg, February / March 2013.

[FKN94] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994.

[FLLL24] X. Fu, M. Li, S. Lyu, and C. Liu. Bit-fixing correlation attacks on goldreich's pseudorandom generators. Cryptology ePrint Archive, Paper 2024/1594, 2024.

[FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, *LNCS* 263, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

[Gen09] C. Gentry. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GGH+13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GH19] C. Gentry and S. Halevi. Compressible FHE with applications to PIR. In *TCC 2019, Part II*, *LNCS* 11892, pages 438–464. Springer, Cham, December 2019.

[GHO20] S. Garg, M. Hajiabadi, and R. Ostrovsky. Efficient range-trapdoor functions and applications: Rate-1 OT and more. In *TCC 2020, Part I*, *LNCS* 12550, pages 88–116. Springer, Cham, November 2020.

[GI14] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *EUROCRYPT 2014*, *LNCS* 8441, pages 640–658. Springer, Berlin, Heidelberg, May 2014.

[Gil99] N. Gilboa. Two party RSA key generation. In *CRYPTO'99*, *LNCS* 1666, pages 116–129. Springer, Berlin, Heidelberg, August 1999.

[GKM+00] Y. Gertner, S. Kannan, T. Malkin, O. Reingold, and M. Viswanathan. The relationship between public key encryption and oblivious transfer. In *41st FOCS*, pages 325–335. IEEE Computer Society Press, November 2000.

[GKPV10] S. Goldwasser, Y. T. Kalai, C. Peikert, and V. Vaikuntanathan. Robustness of the learning with errors assumption. In *ICS 2010*, pages 230–240. Tsinghua University Press, January 2010.

[GKWY20] C. Guo, J. Katz, X. Wang, and Y. Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy*, pages 825–841. IEEE Computer Society Press, May 2020.

[GL10] D. Goldenberg and M. Liskov. On related-secret pseudorandomness. In *TCC 2010*, *LNCS* 5978, pages 255–272. Springer, Berlin, Heidelberg, February 2010.

[GMMM18] S. Garg, M. Mahmoody, D. Masny, and I. Meckler. On the round complexity of OT extension. In *CRYPTO 2018, Part III*, *LNCS* 10993, pages 545–574. Springer, Cham, August 2018.

[GMPP16] S. Garg, P. Mukherjee, O. Pandey, and A. Polychroniadou. The exact round complexity of secure computation. In *EUROCRYPT 2016, Part II*, *LNCS* 9666, pages 448–476. Springer, Berlin, Heidelberg, May 2016.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[Gol06] O. Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, USA, 2006.

[Gol11] O. Goldreich. Candidate one-way functions based on expander graphs. In *Studies in Complexity and Cryptography*, number 6650 in LNCS, pages 76–87. Springer, 2011.

[GOR11] V. Goyal, A. O'Neill, and V. Rao. Correlated-input secure hash functions. In *TCC 2011*, *LNCS* 6597, pages 182–200. Springer, Berlin, Heidelberg, March 2011.

[GPP+24] T. Geoghegan, C. Patton, B. Pitman, E. Rescorla, and C. A. Wood. Distributed aggregation protocol for privacy preserving measurement. Internet-Draft draft-ietf-ppm-dap, Internet Engineering Task Force, December 2024. Work in Progress.

[GS18] S. Garg and A. Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *EUROCRYPT 2018, Part II*, *LNCS* 10821, pages 468–499. Springer, Cham, April / May 2018.

[GSW13] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO 2013, Part I*, *LNCS* 8042, pages 75–92. Springer, Berlin, Heidelberg, August 2013.

[GVW15] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from LWE. In *CRYPTO 2015, Part II*, *LNCS* 9216, pages 503–523. Springer, Berlin, Heidelberg, August 2015.

[GY08] R. Gradwohl and A. Yehudayoff. t-Wise independence with local dependencies. *Information processing letters*, 106(5):208–212, 2008.

[HDCGZ23] A. Henzinger, E. Dauterman, H. Corrigan-Gibbs, and N. Zeldovich. Private web search with tiptoe. In *Proceedings of the 29th symposium on operating systems principles*, pages 396–416, 2023.

[HHC+23] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *USENIX Security 2023*, pages 3889–3905. USENIX Association, August 2023.

[HILL99] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HJKS22] J. Hulett, R. Jawale, D. Khurana, and A. Srinivasan. SNARGs for P from sub-exponential DDH and QR. In *EUROCRYPT 2022, Part II*, *LNCS* 13276, pages 520–549. Springer, Cham, May / June 2022.

[HK21] D. Heath and V. Kolesnikov. One hot garbling. In *ACM CCS 2021*, pages 574–593. ACM Press, November 2021.

[HKKW19] D. Hofheinz, A. Kamath, V. Koppula, and B. Waters. Adaptively secure constrained pseudorandom functions. In *FC 2019*, *LNCS* 11598, pages 357–376. Springer, Cham, February 2019.

[HKW15] S. Hohenberger, V. Koppula, and B. Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In *ASIACRYPT 2015, Part I*, *LNCS* 9452, pages 79–102. Springer, Berlin, Heidelberg, November / December 2015.

[HLL23] Y.-C. Hsieh, H. Lin, and J. Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th FOCS*, pages 415–434. IEEE Computer Society Press, November 2023.

[HLP11] S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO 2011*, *LNCS* 6841, pages 132–150. Springer, Berlin, Heidelberg, August 2011.

[HS20] S. Halevi and V. Shoup. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481, 2020.

[HW15] P. Hubacek and D. Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS 2015*, pages 163–172. ACM, January 2015.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, *LNCS* 2729, pages 145–161. Springer, Berlin, Heidelberg, August 2003.

[IKOS07] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[Imp95] R. Impagliazzo. A personal view of average-case complexity. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pages 134–147. IEEE, 1995.

[IPS09] Y. Ishai, M. Prabhakaran, and A. Sahai. Secure arithmetic computation with no honest majority. In *TCC 2009*, *LNCS* 5444, pages 294–314. Springer, Berlin, Heidelberg, March 2009.

[IR89] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *21st ACM STOC*, pages 44–61. ACM Press, May 1989.

[IS22] J. Indigo and D. Smart. Page Weight: 2022: The web almanac by HTTP archive. https://almanac.httparchive.org/en/2022/page-weight, 2022. Accessed on January 24, 2025.

[ISW03] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO 2003*, *LNCS* 2729, pages 463–481. Springer, Berlin, Heidelberg, August 2003.

[JGB+24] N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. Sharma. Orca: FSS-based secure training and inference with GPUs. In *2024 IEEE Symposium on Security and Privacy*, pages 597–616. IEEE Computer Society Press, May 2024.

[JJ21] A. Jain and Z. Jin. Non-interactive zero knowledge from sub-exponential DDH. In *EUROCRYPT 2021, Part I*, *LNCS* 12696, pages 3–32. Springer, Cham, October 2021.

[JK20] R. Jawale and D. Khurana. Lossy correlation intractability and PPAD hardness from sub-exponential LWE. Cryptology ePrint Archive, Report 2020/911, 2020.

[JKKZ21] R. Jawale, Y. T. Kalai, D. Khurana, and R. Y. Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In *53rd ACM STOC*, pages 708–721. ACM Press, June 2021.

[JLS21] A. Jain, H. Lin, and A. Sahai. Indistinguishability obfuscation from well-founded assumptions. In *53rd ACM STOC*, pages 60–73. ACM Press, June 2021.

[JRX24] J. Januzelli, L. Roy, and J. Xu. Under what conditions is encrypted key exchange actually secure? Cryptology ePrint Archive, Report 2024/324, 2024.

[Kil88] J. Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988.

[KKL+16] V. Kolesnikov, H. Krawczyk, Y. Lindell, A. J. Malozemoff, and T. Rabin. Attribute-based key exchange with general policies. In *ACM CCS 2016*, pages 1451–1463. ACM Press, October 2016.

[KL07] J. Katz and Y. Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and Hall/CRC, 1 edition, 2007.

[KLVW23] Y. Kalai, A. Lombardi, V. Vaikuntanathan, and D. Wichs. Boosting batch arguments and RAM delegation. In *55th ACM STOC*, pages 1545–1552. ACM Press, June 2023.

[KO04] J. Katz and R. Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO 2004*, *LNCS* 3152, pages 335–354. Springer, Berlin, Heidelberg, August 2004.

[KPTZ13] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[KPW13] H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In *CRYPTO 2013, Part I*, *LNCS* 8042, pages 429–448. Springer, Berlin, Heidelberg, August 2013.

[KRA+18] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero. Coming of age: A longitudinal study of TLS deployment. In *Proceedings of the Internet Measurement Conference 2018*, pages 415–428, 2018.

[KS08] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP 2008, Part II*, *LNCS* 5126, pages 486–498. Springer, Berlin, Heidelberg, July 2008.

[KSS13] M. Keller, P. Scholl, and N. P. Smart. An architecture for practical actively secure MPC with dishonest majority. In *ACM CCS 2013*, pages 549–560. ACM Press, November 2013.

[LMR14] K. Lewi, H. W. Montgomery, and A. Raghunathan. Improved constructions of PRFs secure against related-key attacks. In *ACNS 14International Conference on Applied Cryptography and Network Security*, *LNCS* 8479, pages 44–61. Springer, Cham, June 2014.

[Lom22] A. Lombardi. *Provable Instantiations of Correlation Intractability and the Fiat-Shamir Heuristic*. PhD thesis, Massachusetts Institute of Technology, 2022.

[LP23] A. Lazzaretti and C. Papamanthou. TreePIR: Sublinear-time and polylog-bandwidth private information retrieval from DDH. In *CRYPTO 2023, Part II*, *LNCS* 14082, pages 284–314. Springer, Cham, August 2023.

[LPR10] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT 2010*, *LNCS* 6110, pages 1–23. Springer, Berlin, Heidelberg, May / June 2010.

[LPR13] V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for ring-LWE cryptography. In *EUROCRYPT 2013*, *LNCS* 7881, pages 35–54. Springer, Berlin, Heidelberg, May 2013.

[LTV12] A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012.

[Mar98] J. Markoff. Profile: Giving birth to a web business. *The New York Times*, October 1998. Accessed on January 24, 2025.

[Mel22] K. Melissaris. *Witness-Authenticated Key Exchange*. PhD thesis, City University of New York, 2022.

[Mey23] P. Meyer. *Sublinear-communication secure multiparty computation*. PhD thesis, Université Paris Cité, 2023.

[MP12] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT 2012*, *LNCS* 7237, pages 700–718. Springer, Berlin, Heidelberg, April 2012.

[MPD+24] D. Mouris, C. Patton, H. Davis, P. Sarkar, and N. G. Tsoutsos. Mastic: Private weighted heavy-hitters and attribute-based metrics. Cryptology ePrint Archive, Report 2024/221, 2024.

[MST24] D. Mouris, P. Sarkar, and N. G. Tsoutsos. PLASMA: Private, lightweight aggregated statistics against malicious adversaries. *PoPETs*, 2024(3):4–24, July 2024.

[MW16] P. Mukherjee and D. Wichs. Two round multiparty computation via multi-key FHE. In *EUROCRYPT 2016, Part II*, *LNCS* 9666, pages 735–763. Springer, Berlin, Heidelberg, May 2016.

[MW22] S. J. Menon and D. J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy*, pages 930–947. IEEE Computer Society Press, May 2022.

[MZRA22] Y. Ma, K. Zhong, T. Rabin, and S. Angel. Incremental offline/online PIR. In *USENIX Security 2022*, pages 1741–1758. USENIX Association, August 2022.

[NP01] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[NP06] M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM Journal on Computing*, 35(5):1254–1281, 2006.

[NR97] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997.

[Ope24] OpenSSL Project. OpenSSL: Cryptography and SSL/TLS toolkit. https://www.openssl.org/, 2024. Accessed on January 24, 2025.

[OPWW15] T. Okamoto, K. Pietrzak, B. Waters, and D. Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *ASIACRYPT 2015, Part I*, *LNCS* 9452, pages 121–145. Springer, Berlin, Heidelberg, November / December 2015.

[OSY21] C. Orlandi, P. Scholl, and S. Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In *EUROCRYPT 2021, Part I*, *LNCS* 12696, pages 678–708. Springer, Cham, October 2021.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, *LNCS* 1592, pages 223–238. Springer, Berlin, Heidelberg, May 1999.

[Pei15] C. Peikert. A decade of lattice cryptography. Cryptology ePrint Archive, Report 2015/939, 2015.

[Pet24] O. Peters. Polymur universal hash function. https://github.com/orlp/polymur-hash, 2024. Accessed on January 24, 2025.

[Pra62] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[PS08] K. Pietrzak and J. Sjödin. Weak pseudorandom functions in minicrypt. In *ICALP 2008, Part II*, *LNCS* 5126, pages 423–436. Springer, Berlin, Heidelberg, July 2008.

[PS18] C. Peikert and S. Shiehian. Privately constraining and programming PRFs, the LWE way. In *PKC 2018, Part II*, *LNCS* 10770, pages 675–701. Springer, Cham, March 2018.

[PS19] C. Peikert and S. Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In *CRYPTO 2019, Part I*, *LNCS* 11692, pages 89–114. Springer, Cham, August 2019.

[PSZ18] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on OT extension. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):1–35, 2018.

[PTW20] N. Peter, R. Tsabary, and H. Wee. One-one constrained pseudorandom functions. In *ITC 2020*, *LIPIcs* 163, pages 13:1–13:22. Schloss Dagstuhl, June 2020.

[QWW18] W. Quach, H. Wee, and D. Wichs. Laconic function evaluation and applications. In *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018.

[Rab81] M. O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981.

[Reg05] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Ria24] M. Riahinia. *Constrained Pseudorandom Functions: New Constructions and Connections with Secure Computation*. PhD thesis, Ecole normale supérieure de lyon-ENS LYON, 2024.

[Roy22] L. Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In *CRYPTO 2022, Part I*, *LNCS* 13507, pages 657–687. Springer, Cham, August 2022.

[RR20] P. Rindal and L. Roy. libOTe: an efficient, portable, and easy to use oblivious transfer library. https://github.com/osu-crypto/libOTe, 2020. Accessed on January 24, 2025.

[RRT23] S. Raghuraman, P. Rindal, and T. Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In *CRYPTO 2023, Part IV*, *LNCS* 14084, pages 602–632. Springer, Cham, August 2023.

[RS21] L. Roy and J. Singh. Large message homomorphic secret sharing from DCR and applications. In *CRYPTO 2021, Part III*, *LNCS* 12827, pages 687–717, Virtual Event, August 2021. Springer, Cham.

[RTPB22] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. R. Bach. AriaNN: Low-interaction privacy-preserving deep learning via function secret sharing. *PoPETs*, 2022(1):291–316, January 2022.

[RW14] K. Ramchen and B. Waters. Fully secure and fast signing from obfuscation. In *ACM CCS 2014*, pages 659–673. ACM Press, November 2014.

[RZCGP24] M. Rathee, Y. Zhang, H. Corrigan-Gibbs, and R. A. Popa. Private analytics via streaming, sketching, and silently verifiable proofs. *Cryptology ePrint Archive*, 2024.

[SGRR19] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova. Distributed vector-OLE: Improved constructions and implementation. In *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.

[SLC+24] M. Sporny, D. Longley, D. Chadwick, D. Reed, and O. Steele. Verifiable credentials data model v2.0. W3C candidate recommendation snapshot, W3C, January 2024.

[SS24a] S. Servan-Schreiber. Constrained pseudorandom functions for inner-product predicates from weaker assumptions. In *ASIACRYPT 2024, Part II, LNCS 15485*, pages 232–265. Springer, Singapore, December 2024.

[SS24b] S. Servan-Schreiber. Open-source implementation of constrained PRFs for inner product predicates. https://github.com/sachaservan/cprf, 2024.

[SS24c] S. Servan-Schreiber. Open-source implementation of efficient oblivious transfer from QuietOT. https://github.com/sachaservan/quietot, 2024.

[SSLD22] S. Servan-Schreiber, S. Langowski, and S. Devadas. Private approximate nearest neighbor search with sublinear communication. In *2022 IEEE Symposium on Security and Privacy*, pages 911–929. IEEE Computer Society Press, May 2022.

[SW14] A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[SYL+18] S. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *ACM CCS 2018*, pages 763–780. ACM Press, October 2018.

[Ünal23] A. Ünal. New baselines for local pseudorandom number generators by field extensions. Cryptology ePrint Archive, Report 2023/550, 2023.

[Val84] L. G. Valiant. A theory of the learnable. In *16th ACM STOC*, pages 436–445. ACM Press, 1984.

[vDGHV10] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT 2010*, *LNCS* 6110, pages 24–43. Springer, Berlin, Heidelberg, May / June 2010.

[VHG23] A. Vadapalli, R. Henry, and I. Goldberg. Duoram: A bandwidth-efficient distributed ORAM for 2- and 3-party computation. In *USENIX Security 2023*, pages 3907–3924. USENIX Association, August 2023.

[VSH22] A. Vadapalli, K. Storrier, and R. Henry. Sabre: Sender-anonymous messaging with fast audits. In *2022 IEEE Symposium on Security and Privacy*, pages 1953–1970. IEEE Computer Society Press, May 2022.

[W3C21] W3C. Private advertising technology community group. https://www.w3.org/community/patcg/, 2021. W3C Community Group. Accessed: January 12, 2025.

[WC81] M. N. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22:265–279, 1981.

[WYG+17] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia. Splinter: Practical private queries on public data. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 299–313, 2017.

[XW23] P. Xu and L.-P. Wang. Multi-key homomorphic secret sharing from LWE without multi-key HE. In *ACISP 23*, *LNCS* 13915, pages 248–269. Springer, Cham, July 2023.

[Yao86] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[YGJL21] J. Yang, Q. Guo, T. Johansson, and M. Lentmaier. Revisiting the concrete security of Goldreich's pseudorandom generator. *IEEE Transactions on Information Theory*, 68(2):1329–1354, 2021.

[YJG+23] P. Yang, Z. L. Jiang, S. Gao, J. Zhuang, H. Wang, J. Fang, S. Yiu, and Y. Wu. FssNN: Communication-efficient secure neural network training via function secret sharing. Cryptology ePrint Archive, Report 2023/073, 2023.

[YWL+20] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM CCS 2020*, pages 1607–1626. ACM Press, November 2020.

[ZPZS24] M. Zhou, A. Park, W. Zheng, and E. Shi. Piano: Extremely simple, single-server PIR with sublinear server computation. In *2024 IEEE Symposium on Security and Privacy*, pages 4296–4314. IEEE Computer Society Press, May 2024.