

Cryptographically Certified Hypothesis Testing

by

Sacha Servan-Schreiber



Submitted to the Department of Computer Science
in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science with Honors

at

BROWN UNIVERSITY

December 2018

Thesis Advisor

Anna Lysyanskaya
Professor of Computer Science
Brown University

Thesis Reader

Tim Kraska
Associate Professor of Computer Science
MIT

Cryptographically Certified Hypothesis Testing

by

Sacha Servan-Schreiber

Submitted to the Department of Computer Science
on December 1, 2018, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Computer Science with Honors

Abstract

We present CUSTODES: a new approach to solving the complex issue of preventing “p-hacking” in scientific studies. This novel protocol provides a concrete and publicly auditable method for controlling false discoveries, and eliminates any potential for data dredging on the part of researchers during the data analysis phase. CUSTODES provides provable guarantees for the validity of each hypothesis test performed on a dataset by using cryptographic techniques to certify outcomes of statistical tests. CUSTODES achieves this using a decentralized authority and a tamper-proof ledger which enables the auditing of the hypothesis testing process. We present a construction of CUSTODES which we implement and evaluate using both real and synthetic datasets on common statistical tests, demonstrating the effectiveness and practicality of CUSTODES in the real world. Furthermore, we propose an extension to CUSTODES for rendering the results of statistical tests computed through CUSTODES differentially private without introducing the need for a trusted party, and provide upper bounds on the sensitivity of each test statistic.

Thesis Advisor: Anna Lysyanskaya
Title: Professor of Computer Science

Thesis Reader: Tim Kraska
Title: Associate Professor of Computer Science

Acknowledgments

First and foremost, I would like to thank Anna Lysyanskaya and Tim Kraska for their incredible pedagogy and guidance in advising me for this thesis. I am forever grateful for the myriad opportunities you’ve both created in my path.

To Olya Ohrimenko: Thank you for your collaboration on this project, for always being available to answer my many questions, and for discussing all the new and interesting problems we’ve encountered along the way. Many aspects of this work were only made possible thanks to your patience and outstanding guidance.

To Emanuel Zgraggen: Thank you for providing the brilliant seed idea for this project, for collaborating with me throughout my research endeavors, and for always believing in my ability to see this project through from start to finish. I will always have fond memories of our many brainstorming sessions and countless coffee breaks.

To Andy van Dam: Thank you for generously supporting part of this research and for being an outstanding mentor and teacher throughout my time at Brown. Though I’m neither the first nor the last to say this, taking and TAing *Introduction to Computer Graphics* truly remains among my fondest memories of my time here.

To my family Olga and Edouard (and my little brother Roman). I am so grateful for all the support I’ve received over the years (not to mention the multitude of care packages delivered to my doorstep). Thank you for always believing in me—in rain or shine—and giving me hope and courage when I needed it most. Mom, thank you for raising me, supporting me in *all* my endeavors, and most of all for always understanding my perspective. Edouard, thank you for your help and support in my efforts, and for being there for me on no small number of occasions.

To the Brown community: I am deeply touched when I think of all the people I was fortunate enough to cross paths with during my time as an undergraduate here. It was truly incredible to be surrounded by such vibrant and creative energy. To John Savage, thank you for instructing me on everything from cybersecurity to circuits, and for many tangential conversation related to these topics. To Matteo Riondato, thank you for your collaboration, mentorship, and for providing me with outstanding advice on everything from research to life. To Aja, mahalo for your love, for your energy, and most of all for always being open to board a flight with no destination. To Ben, thank you for your friendship, advice, humor — you were a tremendous inspiration on many occasions. To Christopher, thank you for being a great friend and an awesome flatmate (alas, not everyone is lucky enough to experience live readings of Greek poetry every morning). To Leah, thank you for inspiring me on many topics and for introducing me to many beautiful aspects of computer science, math, and most of all cryptography. To Sergio, thank you for bringing into existence interesting conversations and inspiring ideas; your understanding and analysis of everything from philosophy to policy has greatly influenced my perspective.

I dedicate this thesis to my father, David Servan-Schreiber
(*April 21, 1961 – July 24, 2011*).

Papa, your courage, your humility, your strength, will forever remain in my memories and in my heart. Though I know that you are here in spirit, it brings tears to my eyes, on this occasion as on many others, to not have you present.

Tu me manques.

Contents

1	Introduction	11
1.1	Introduction	11
1.2	Contributions	13
1.3	System Overview	14
1.3.1	Design	15
1.3.2	Security Goals	15
1.3.3	Security Model	16
1.3.4	Controlling False Discoveries	18
1.3.5	Certification of Hypotheses	19
1.4	Contributions	20
1.5	Related Work	20
2	Preliminaries & Building Blocks	23
2.1	Tamper-proof Ledger	23
2.2	Threshold Homomorphic Encryption	24
2.2.1	Threshold-Paillier Scheme	25
2.3	Performing Arithmetic Operations	27
2.3.1	Fixed-Point Encoding	27
2.3.2	Signed Encoding	28
2.4	Secure Multi-party Computation (MPC)	29
2.4.1	Universal Composability (UC)	29
2.4.2	Statistical Security	30
2.4.3	Evaluating Non-linear Arithmetic Gates	31

2.5	MPC Protocols	31
2.5.1	Generating Random Bits	33
2.5.2	Sign Extraction	34
2.5.3	Computing Fixed-Point Square Roots	35
2.6	Practical Considerations	38
2.7	Relevant Notation	41
3	System Construction	42
3.1	Framework	42
3.2	Implementation	44
3.3	Security Evaluation	47
3.4	Evaluating Statistical Tests	50
3.4.1	Dataset Characteristics	51
3.5	Student's t-test	51
3.6	Pearson's Correlation Test	54
3.7	Chi-Squared Test	57
4	Providing Differential Privacy	60
4.1	Motivation	60
4.2	Definitions	61
4.2.1	Design of Differentially Private Mechanisms	62
4.2.2	Impelentation Challenges	62
4.3	Sensitivity of Statistical Tests	63
4.3.1	Sensitivity of Student's t-test	63
4.3.2	Sensitivity of Pearson Correlation	65
4.3.3	Sensitivity of Chi-Squared	67
4.4	Secure Noise Generation	68
4.5	Discussion	70
4.6	Future Work (Related to this Chapter)	71

5	Experimental Evaluation	73
5.1	Goals	73
5.2	Experiment Setup	74
5.2.1	Implementation and Environment	74
5.2.2	Datasets	74
5.2.3	Evaluation	75
5.2.4	Parameters	76
5.3	Results	77
5.3.1	A Note on Scalability	81
6	Discussion	83
6.1	Alternative Instantiations	83
6.1.1	FHE-based Instantiation	83
6.1.2	LSS-based Instantiation	85
6.2	Conclusion	87

List of Figures

1-1	High level overview of CUSTODES.	13
3-1	Overview of Paillier-based instantiation of CUSTODES.	43
3-2	Use of Tamper-proof ledger in CUSTODES.	45
5-1	Runtime comparisons for the Student's t-test.	77
5-2	Runtime comparisons for the Pearson Correlation test.	77
5-3	Runtime comparisons for the Chi-Squared test.	78
5-4	Runtime comparisons for computing a single multiplication interactively as a function of the number of parties, using the threshold Paillier and linear secret sharing as the underlying method.	80
6-1	High-level overview of an FHE-based implementation of CUSTODES. .	84
6-2	High-level overview of an FHE-based implementation of CUSTODES. .	86
6-3	XKCD Cartoon on p-hacking [58].	94

List of Tables

2.1	Summary of cryptographic building blocks and their role in realizing the main properties of CUSTODES.	24
2.2	Summary of notation used throughout this work.	41
5.1	Dataset characteristics	75
5.2	Absolute error of statistical tests computed through CUSTODES compared to the equivalent test computed using the SciPy package. . . .	76
5.3	Number of multiplications required to compute a Student's t-test and Pearson Correlation test for each dataset.	81
5.4	Number of multiplication required to compute a Chi-Squared test for each dataset.	81

Chapter 1

Introduction

1.1 Introduction

“Data is the new oil” and as such, it is mined (i.e., gathered), shared, analyzed, re-analyzed, and re-re-analyzed until it yields to more and more interesting insights. With every exploration to find yet another insight, the chance of encountering a random correlation increases. This phenomenon is formally known as the multiple comparisons problem (MCP) and, if done in a systematic fashion, is often referred to as “HARKing” [51], “p-hacking” [40] or “data dredging”.

While a variety of statistical techniques exist to control the False Discovery Rate (FDR) introduced by the MCP [24, 4], there is surprisingly almost no support to ensure that analysts actually use them. Rather individual research groups rely on frequently varying data analysis guidelines and trust in their group members to follow them. Things get even worse when the same data is analyzed by several institutions or teams. It is currently close to impossible to reliably employ statistical procedures, such as the Bonferroni [24] method, that guard against p-hacking across collaborators. It only requires one member to “misuse” the data (whether intentionally or not) and detecting, let alone recovering, from such incidents is next to impossible. This problem is perhaps amplified by three factors: 1) the pressure on PhD students and PIs to publish [60], 2) “publication bias” [22] as papers with significant results are more likely to be published, and 3) the increasing trend to share and make datasets

publicly available for any researchers to use. It is therefore unsurprising that the MCP is among the leading reasons why the scientific community is plagued by false discoveries [3, 42, 45, 43].

To illustrate this problem further consider a publicly available dataset such as MIMIC III [47]. This dataset contains de-identified health data associated with $\approx 40,000$ critical care patients. MIMIC III has already been used in various studies [57, 36, 41] and it is probably one of the most (over)analyzed clinical datasets. As such, any discovery made on MIMIC runs the risk of being a false discovery. Even if a particular group of researchers follow a proper FDR protocol, there is no control over what happens across different groups and tracking hypotheses at a global scale poses many challenges. It is therefore hard to judge the validity of any insight derived from such a dataset.

A solution to guarantee validity of insights commonly used in clinical trials - preregistration of hypotheses [14] - falls short in these scenarios. The data is collected upfront without knowing what kind of analysis will be done later on. Perhaps more promising is the use of a hold-out dataset. The MIMIC authors can release only 30K patient records as an exploration dataset (EDS) and hold back 10K as a validation dataset (VDS). The EDS can then be used in arbitrary ways to find interesting hypothesis. However, before any publication is made by a research group using the dataset, all hypotheses must be tested for its statistical significance over the VDS. Unfortunately, in order to use the VDS more than once, the same requirement as before holds true: every hypothesis over the VDS has to be tracked and controlled for. Furthermore, the data owner, the MIMIC authors in this case, need to provide this hypothesis validation service. This is both a burden for the data owners as well as a potential risk. Researchers need to trust the data owners to apply FDR control procedures correctly and to objectively evaluate their hypotheses.

The above example illustrates the motivation behind CUSTODES. Our goal is to create a system that guarantees the validity of statistical test outcomes and allows readers (and/or reviewers) of publications to audit them for correctness. Using proven cryptographic techniques to certify outcomes of statistical tests by a decentralized au-

thority, we eliminate the risk of data-dredging (intentional and otherwise) on the part of researchers or data owners. CUSTODES can be used in various settings, including cases where the data is public and only the hold-out data is fed into CUSTODES (as in the example above), in smaller settings where a few research groups collaborate on combined data, or even within single teams where lab managers can opt to encrypt all of their data and use CUSTODES as a way to prevent unintentional false-discoveries, assign accountability and foster reproducibility.

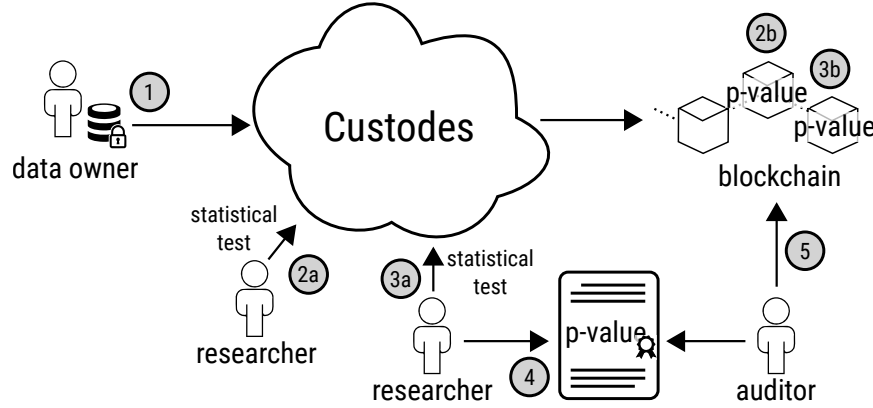


Figure 1-1: High level overview of CUSTODES.

1.2 Contributions

The primary contribution of this work is the CUSTODES framework. Figure 1-1 shows a high-level overview of the proposed system. (1) A data owner encrypts their dataset (either the full dataset or just a hold-out) and submits it to CUSTODES: a decentralized platform consisting of multiple nodes that are run by different entities (e.g., universities or research groups). (2a and 3a) Researchers can post requests for statistical tests to CUSTODES. (2b and 3b) CUSTODES securely computes these tests using several cryptographic techniques, and stores the results (and transcript of the computation) sequentially on a tamper-proof ledger (e.g., a Blockchain [59]). (4) One of the researchers decides to publish their finding and includes a “certified p-value” in their paper. (5) A reader or auditor of this publication can query the ledger, retrieve *all* p-values of *all* the tests that have been run on this particular dataset and apply an

incremental FDR control procedure *a posteriori* [69, 31] to validate the publication’s finding.

CUSTODES prevents p-hacking by using encrypted data and by guaranteeing that every statistical test is accounted for. It is, to the best of our knowledge, the first cryptographically based solution to the problem of p-hacking with provable security guarantees. CUSTODES is a framework which we instantiate based on additively homomorphic encryption and multi-party computation protocols. Using those building blocks, we implement three widely used hypothesis tests (Student’s t-test, Pearson Correlation, and Chi-Squared, see § 3.4) in CUSTODES and evaluate its performance with various configurations and dataset sizes.

1.3 System Overview

We model the scenario considered in the introduction as follows: a data owner wants to release a dataset \mathcal{D} , to a set of researchers, denoted $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$, for the purpose of executing statistical tests. Note: \mathcal{D} can be either a full dataset or a hold-out. We aim to build a system that guarantees that all statistical tests over \mathcal{D} are accounted for and are executed and reported in a truthful manner. In other words, an FDR control procedure is correctly applied for *all* statistical tests computed on \mathcal{D} . If all p-values resulting from tests computed on \mathcal{D} are accounted for, then it is possible to apply an FDR control procedure to determine which null-hypothesis should be accepted (resp. rejected) to ensure the probability of a false rejection remains below a threshold [69, 31, 4] (detailed in § 1.3). Moreover, the entire process must be auditable: a publication can be vetted for having followed the correct FDR-control procedure. At a high level, CUSTODES achieves these requirements by restricting the exposure of the dataset \mathcal{D} such that researchers are only able to test hypotheses on \mathcal{D} via the CUSTODES interface which stores all hypotheses and results in a tamper-proof, auditable trace.

1.3.1 Design

Our design of CUSTODES is motivated by the following observations. The data owner cannot release a dataset \mathcal{D} to the researchers directly since it creates a possibility for p-hacking and other forms of bias (e.g., parties can run tests privately and report only favorable results without controlling for the FDR). Hence, \mathcal{D} has to be released such that only the output of the statistical tests performed on \mathcal{D} is made available to researchers. A naïve solution is to provide access to \mathcal{D} through a stringent interface which *only* returns the results of statistical tests. However, even such a solution is not sufficient for enforcing correctness as it can be abused by parties querying the interface until a favorable (i.e., significant) result is obtained while avoiding to report the intermediate queries or adequately controlling for the FDR. Moreover, such a solution has no way to audit the test computation making it impossible to “certify” the validity of discoveries.

CUSTODES addresses this problem by using several cryptographic methods, specifically, *secure computation* and a *tamper-proof ledger* which are discussed in detail in the following chapters. At a high level, CUSTODES requires that a dataset \mathcal{D} be encrypted by the data owner *prior* to being made available to researchers (or publicly released). Researchers use the encrypted dataset, denoted $\llbracket \mathcal{D} \rrbracket$, to compute statistical tests using secure computations over the encrypted data with a mechanism for revealing (decrypting) only the test statistic and simultaneously ensuring that the FDR is controlled for. CUSTODES ensures that each revealed statistic is recorded on a publicly accessible and secure ledger making each result of a statistical test public and verifiable. This latter requirement ensures that 1) all statistical tests executed on $\llbracket \mathcal{D} \rrbracket$ are recorded *in sequence* and 2) makes it possible to certify that FDR control procedures are applied correctly.

1.3.2 Security Goals

At a high level, CUSTODES aims to achieve the following security goals. We formalize these properties in § 3.3.

- **Correctness.** Statistical tests computed on $\llbracket \mathcal{D} \rrbracket$ are correct, i.e, the p-value of a test computed in CUSTODES is equivalent to the p-value computed through standard statistical packages (e.g., SciPy [48]).
- **Confidentiality.** The only information revealed about \mathcal{D} is the results of statistical tests executed through CUSTODES which ensures that the FDR is fully controlled for and no hypothesis can be tested outside of CUSTODES’s interface from “leaked” information about the contents of \mathcal{D} .
- **Access control.** Statistical tests on \mathcal{D} can be executed only by a set of approved researchers which ensures accountability in the hypothesis testing procedure.
- **Verifiability.** Every hypothesis test executed through CUSTODES has a corresponding certificate ψ that is publicly accessible and verifiable, even by a third-party (e.g., reviewers, readers, organizations, etc.).
- **Auditability.** Given a certificate ψ , anyone can verify the correctness of the hypothesis test computation associated with the certificate (i.e., determine whether the null hypothesis should be accepted or rejected).

1.3.3 Security Model

CUSTODES provides provable guarantees to the five properties outlined in § 1.3.2 under the following threat model. The assumptions in the model are with respect to the data owner, participants engaged in the protocol, and the network over which the system is instantiated.

Data Owner. We assume that the data owner does not collude with the researchers who run the statistical tests. This assumption is necessary given that a dishonest owner would have unfettered access to the (unencrypted) dataset \mathcal{D} and can thereby trivially bypass any FDR control procedure set in place by CUSTODES.

Participants. We assume that all parties engaged in interactive computations are

honest-but-curious, that is, they individually adhere to the correct protocol but are interested in learning more about the underlying dataset so that they may circumvent FDR control procedures, and may collude among themselves in attempt to achieve this (i.e., to engage in p-hacking). To this end, we assume that at most $t-1$ out of v parties may collude with each other in order to obtain more information about the dataset or manipulate a test certificate. However, we assume that the set of colluding parties is static (i.e., does not change during protocol execution). Researchers evaluating statistical tests (which potentially includes one of the parties in the network), however, are assumed to be malicious and may request malformed evaluations of statistical tests circuits in an attempt to learn more about the underlying dataset. In other words, while evaluations of arithmetic operations and other interactive computations in a statistical test circuit are assumed to be secure (in the honest-but-curious setting), we make no assumptions on the *validity* of the circuit itself which may deviate from a valid statistical test. To this end, we assume that the evaluation of the circuit is performed honestly but the circuit itself may be corrupted. However, malicious behavior is exposed in the auditing phase where a malformed circuit makes the certification fail.

Finally, we require a secure public key infrastructure in place which allows to identify individual parties (and researchers) by their public key and assume that all messages exchanged during protocol execution are digitally signed with a party's identity. This requirement is easily satisfied using standard cryptographic primitives, and as such, we omit details on the implementation of such a scheme.

Network. CUSTODES does not rely on a secure communication channel between participants in the network and therefore tolerate the presence of a *non-blocking* adversary controlling the network. However, we do not assume the presence of an *active* network adversary (one that can actively block or drop packets) as that would disrupt the availability of participants and the tamper-proof ledger. As such, this model allows for messages exchanged between parties to be made publicly available (e.g., by recording them on a public ledger) without compromising security of the overall system.

1.3.4 Controlling False Discoveries

An important step in realizing the construction of CUSTODES is understanding how FDR control procedures are used in the data analysis phase.

In order to control for false discoveries it is necessary to know *all* p-values computed over a dataset up to and including the current test. While standard procedures such as Bonferroni [24] must be applied *a posteriori* of the data analysis (once all hypotheses have been tested), in CUSTODES, we desire a method for controlling the FDR in a streaming fashion, ideally without knowledge or restriction on future tests. We use a common control procedure known as α -investing [31]. The α -investing procedure is a standard choice for controlling false discoveries in practice when the total number of hypotheses is unknown beforehand [69, 70]. We briefly describe the procedure below and we refer the reader to [31, 69] for additional details to the ones we provide as they are outside the scope of this work.

Formally, α -investing controls the *marginal False Discovery Rate* (mFDR) which is defined as:

$$mFDR(i) = \frac{E[V(i)]}{E[R(i)] + 1}$$

where i denotes the total number of tests which have been executed so far, $V(i)$ denotes the number of false discoveries, and $R(i)$ denotes the total number of discoveries (up to and including the i th hypotheses test controlled with the α -investing procedure).

The testing procedure is said to control the $mFDR$ at level α if $mFDR(i) \leq \alpha$ [69]. Intuitively the α -investing procedure works by assigning to each hypothesis test a budget α_i from the initial “ α -wealth”, $w_0 = \alpha$. If the p-value of the null hypothesis being considered is above α_i the null hypothesis is accepted and some budget is lost, otherwise it is rejected and some testing budget is gained.

Consider the i th hypothesis, H_i , being tested. H_i is assigned an α -budget $0 < \alpha_i < w_i$. If the p-value, p_i , is below the threshold, i.e., $p_i < \alpha_i$, then the null hypothesis H_i is rejected and the testing procedure gains some $\omega < \alpha$ increase in wealth (e.g., “*return on investment*”). On the other hand, if the null hypothesis H_i is accepted,

then $\alpha_i/(1 - \alpha_i)$ alpha wealth is deducted from the overall wealth. Formally, let w_i be the alpha wealth available for the i th hypothesis. Then, for the $i + 1$ st hypothesis,

$$w_{i+1} = \begin{cases} w_i + \omega & \text{if } p_i \leq \alpha_i, \\ w_i - \frac{\alpha_i}{1-\alpha_i} & \text{if } p_i > \alpha_i \end{cases} \quad (1.1)$$

In CUSTODES, the initial α -wealth and budget allocated per test can be determined by the data owner or set as a static constant. The importance of this procedure, when it comes to this work, is that given all p-values computed on \mathcal{D} up to the i th test, it is easy to determine whether the hypothesis of the $i + 1$ st test should be accepted (resp. rejected) based on the resulting p-value.

1.3.5 Certification of Hypotheses

The overarching goal of CUSTODES is to *certify* insights gained from data as valid. Therefore, CUSTODES must provide a mechanism for certifying the outcomes of hypotheses testing procedures. Taking a birds-eye view, for every test that CUSTODES executes, it produces a publicly available certificate ψ which is verifiable and unforgeable. Formally, we define a certificate of a statistical test as a tuple $(\tau, \mathcal{P}^{\text{pk}}, \mathcal{T}, (\mathbf{t}, \mathbf{p}))$ where τ is the *test index* (i.e., its order among all the tests that have been executed so far on \mathcal{D}), \mathcal{P}^{pk} is the identifier of the researcher that requested the test (e.g., a public key), \mathcal{T} is the code of the statistical test function, and (\mathbf{t}, \mathbf{p}) is the result of executing $\mathcal{T}(\mathcal{D})$ where \mathbf{t} is the test-statistic and \mathbf{p} is the p-value corresponding to \mathbf{t} . Given that each certificate contains both the test index τ and resulting p-value, it is trivial to verify whether an FDR control procedure was applied when accepting (resp. rejecting) a null-hypothesis using the procedure in § 1.3.4. Note that the certificate by itself does not ensure that $(\mathbf{t}, \mathbf{p}) = \mathcal{T}(\mathcal{D})$, however, CUSTODES records sufficient intermediate information for each test on a tamper-proof ledger that anyone can verify whether it is indeed the case. We formalize and elaborate on these claims in § 3.4.

1.4 Contributions

- We present the first cryptographically based solution to the problem of p-hacking. Our system provides provable certification for the validity of hypothesis tests computed through CUSTODES.
- We show how CUSTODES achieves third-party auditable statistical insights through a careful combination of tamper-proof ledgers, homomorphic encryption, and multi-party computation to solve the complex issue of false-discoveries as a result of multiple-hypotheses testing in data analysis, especially in situations where researchers cannot be trusted to apply the FDR control procedures correctly.
- We describe a construction for compiling three common hypothesis tests (Student’s t-test, Pearson Correlation test, and Chi-Squared test) in CUSTODES and formally prove the security guarantees of our system under the specified threat model.
- We present an extension to CUSTODES for evaluating statistical tests in a differentially private manner without requiring a central authority and derive sensitivity bounds on the three statistical tests described in this work.
- We implement and extensively evaluate CUSTODES in regard to its performance and accuracy on both real-world and synthetic data to demonstrate the practicality of the solution for addressing p-hacking in the real world.

1.5 Related Work

Much of the related work surrounding CUSTODES either focuses on private computations using homomorphic encryption or non-cryptographic methods for preventing p-hacking such as methods for pre-registration of hypotheses. To our knowledge, there has been no prior work using cryptographic techniques for certifying the validity of statistical tests in an auditable way.

Lauter *et al.* [54] and Zhang *et al.* [68] propose the use of homomorphic encryption to compute statistical tests on encrypted genomic data. However, the constructions are not intended for validating statistical testing procedures but rather for guarding the privacy of patient data. Furthermore, Lauter *et al.* make several simplifications such as not performing encrypted division (rather performing arithmetic division in the clear), imposing assumptions on the data, etc., making their solution less general compared to methods for computing statistical tests in CUSTODES.

Homomorphic encryption and multi-party computation techniques have been used for outsourcing machine learning tasks of private data [65, 8, 66, 39]. Our work, however, crucially relies on the decryption functionality being decentralized in order to control and keep track of data exposure.

Several *non-cryptographic* techniques exist to guard against false discoveries in statistical analysis. For example, there are several procedures that adjust p-values in scenarios where multiple hypotheses are examined at once. These range from conservative protocols that bound the family-wise error rate [24] to more relaxed procedures that bound the ratio of false rejections among the rejected tests (false discovery rate, FDR) [4] or the marginal FDR [69, 31]. When applied properly these techniques prevent p-hacking in an idealized scenario. However, there is no guarantee that they are applied correctly. A researcher can misuse these procedures, intentionally or otherwise, and only apply them over a subset of all statistical tests being computed on a given dataset. Furthermore, there is no way for external auditors to verify how these methods were applied. Our work, on the other hand, leverage these techniques by operating on encrypted data and tracking statistical tests in a tamper-proof ledger which certifies the correctness of results in an auditable way.

Dwork *et al.* [27, 26] propose a method that constrains analyst’s access to a hold-out dataset as follows. A trusted party keeps a hold-out dataset and answers up to m hypotheses using a differentially private algorithm. Here, m depends on the size of the hold-out data and the generalization error that one is willing to tolerate. Hence, compared to our decentralized approach, this method assumes *trust into a single party* that (1) does not release the dataset to the researchers and (2) does not

answer more than m hypotheses queries.

Frankle *et al.* [32] propose a system for making governments accountable for secret processes which uses a combination of a tamper-proof ledger and MPC to create a publicly auditable ledger of surveillance requests issued to companies. While the purpose of the system is unrelated to hypothesis testing, the techniques used in achieving auditability are similar to ours and therefore of related interest.

Bogdanov *et al.* [6, 7] describe RMind which is a system for secure statistical analysis using secret sharing and multi-party computation. While the computation techniques are similar, the work focuses on computing statistical analysis using private data and thus differs from our goal of validating hypotheses.

Finally, we note that preserving privacy of dataset values when releasing results of statistical tests [33, 46] is orthogonal to the primary contribution of this work. However, we do touch on techniques for achieving statistical test output privacy in chapter 4.

Chapter 2

Preliminaries & Building Blocks

In this chapter we describe the necessary preliminaries and cryptographic building-blocks for instantiating CUSTODES. The chapter covers existing cryptographic primitives as well as new variants and extensions to existing work. The reader should keep in mind that the purpose of these building blocks is to construct a system where researchers can compute over an encrypted dataset in an auditable way. Table 2.1 summarizes the building blocks introduced in this chapter and their usage in the system instantiation.

2.1 Tamper-proof Ledger

CUSTODES relies on a tamper-proof append-only ledger that we model as a Blockchain abstraction, \mathcal{B} . It records test runs against the dataset, each test result, order of test execution, and the identity of the researcher requesting test executions. The ledger can be maintained by a central authority (e.g., the data owner), by the parties $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ themselves using a consensus protocol (e.g., Paxos [53]), or by independent parties (e.g., using a public Blockchain based on a decentralized consensus protocol such as Nakamoto consensus [59]). Similar to the ledger abstraction used in [12], our main requirement is that the above abstraction be correct and always available.

Property	Building block(s)	Role in CUSTODES
<i>Access Control</i>	Digital signatures & threshold encryption.	Only approved researchers can request test computation and a threshold majority of approved parties can decrypt the data.
<i>Hypothesis Tracking</i>	Threshold encryption, multi-party computation, tamper-proof ledger.	Computation results can only be revealed through a consensus of parties in the system which record every tested hypothesis at decryption time on a ledger.
<i>Auditability</i>	Tamper-proof ledger	A transcript of every statistical test computation is recorded on the ledger making them verifiable and auditable by researchers and third-parties.
<i>Certification</i>	Tamper-proof ledger	Sequence of statistical tests recorded on the ledger make it possible to apply the α -investing procedure <i>a posteriori</i> which serves as a certificate.

Table 2.1: Summary of cryptographic building blocks and their role in realizing the main properties of CUSTODES.

2.2 Threshold Homomorphic Encryption

Homomorphic Encryption (HE) is a special form of encryption which enables the evaluation of certain arithmetic operations over encrypted values without knowledge of the secret key used to encrypt. Thus, HE schemes make it possible to evaluate a subset of functions without revealing information on the inputs nor the output. While several such schemes exist, in this work we use the additively homomorphic encryption scheme known as Paillier [61]. The Paillier scheme enables the evaluation linear arithmetic gates (i.e., addition, subtraction) but does not support non-linear operations such as multiplication, out-of-the-box. We describe methods for extending the functionality of Paillier later in this section. CUSTODES aims to distribute the power of the data owner to a set of computing parties $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$. Thus, it is necessary to distribute the power of decrypting the dataset (i.e., the secret key) to the computing parties in such a way that no individual party can decrypt without

consensus from a threshold number of parties. A threshold encryption scheme allows the decryption of ciphertexts only when a threshold t of parties cooperate in the decryption procedure. Specifically, no subset of parties *smaller* than the specified threshold can learn any information, even if colluding among themselves [18]. In essence, this allows CUSTODES to distribute the ability of the data owner to decrypt the dataset to a set of parties provided that no more than $t-1$ of parties are corrupted by the adversary.

2.2.1 Threshold-Paillier Scheme

We use the threshold variant of the Paillier scheme from [18]. The threshold variant enables delegation of decryption to a set of parties $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ provided no more than t out of v of them are malicious [18]. We make use of this fact later on when describing the decryption of computed test statistics as well as in our security analysis.

Let $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ be a set of v parties where a threshold number t of them are required to participate in order to decrypt (reveal) a ciphertext.

Paillier.KeyGen(k). Pick two k -bit primes, p and q such that $p = 2p' + 1$ and $q = 2q' + 1$ where p', q' are Sophie Germain primes. Set $n = pq$, $m = p'q'$, $\lambda = \text{lcm}(p-1, q-1)$.

Pick an integer d such that $d \equiv 1 \pmod{n^2}$ and $d \equiv 0 \pmod{m}$.¹ Set t such that $t \leq v$ and construct the polynomial $f(X) = \sum_{i=0}^{t-1} a_i X^i \pmod{n^2 m}$ where a_i for $i \geq 1$ is picked at random from the set $\{0, 1, \dots, n^2 m - 1\}$.

Set $a_0 = d$ and for $i = 1 \dots v$, $\mathbf{sk}_i = f(i)$ corresponding to the i th share of the secret key $\mathbf{sk} = \lambda$ such that any subset of t secret key shares can be used to decrypt a ciphertext. Set the public key $\mathbf{pk} = n$.

Output $(\mathbf{pk}, \mathbf{sk}_1, \dots, \mathbf{sk}_v)$.²

¹[18] warns the reader that using $d = \lambda$ (as per the original Paillier scheme) is not secure in the threshold setting. Therefore, d must be chosen independently of λ .

²After running **Paillier.KeyGen** the dealer distributes the secret key shares such that only \mathcal{P}_i receives the share \mathbf{sk}_i .

Paillier.Enc(pk, $z \in \mathbb{Z}_n$). To encrypt a message $x \in \mathbb{Z}_n$, pick a random $r \in \mathbb{Z}_{n^2}$ and compute the ciphertext $c = (n+1)^x r^{n^2} \pmod{n^2}$. Output c .

Paillier.ThresDec($c \in \mathbb{Z}_{n^2}, \{\mathcal{P}_1, \dots, \mathcal{P}_v\}$). To decrypt a ciphertext c , each player $\mathcal{P}_i \in \{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ uses the private secret key share \mathbf{sk}_i to compute $c_i = c^{2\Delta \mathbf{sk}_i}$ where $\Delta = v!$. With the required t unique shares (c_1, \dots, c_t) , the decryption can be computed by taking the set $C = \{c_1, \dots, c_t\}$ of t shares and combining them using Lagrange interpolation as follows:

$$c' = \prod_{i \in C} c_i^{2\lambda_{0,i}^C} \text{ where } \lambda_{0,i}^C = \Delta \prod_{i' \in C \setminus \{i\}} \frac{i}{i' - i} \in \mathbb{Z}$$

Note that $c' = c^{4\Delta^2 f(0)} = c^{4\Delta^2 d}$. Since $4\Delta^2 d \equiv 0 \pmod{\lambda}$ and $4\Delta^2 d \equiv 4\Delta^2 x \pmod{n^2}$, we get that $c' = (n+1)^{4\Delta^2 x}$ where x is the plaintext we wish to extract. Therefore, $4\Delta^2 x$ can be (efficiently) obtained as described in the original Paillier which allows us to recover x as follows:

$$m = L(c' \bmod n^2)(4\Delta^2)^{-1} \pmod{n}$$

where $L(u) = \frac{u-1}{n}$ as specified in [61, 18]. Output m .

Paillier.EvalAdd($c_1, c_2 \in \mathbb{Z}_{n^2}$). Addition of two encrypted values $c_1, c_2 \in \mathbb{Z}_{n^2}$ is computed as follows: pick a random $r \in \mathbb{Z}_{n^2}$ and compute $c' = (c_1 c_2) r^{n^2} \pmod{n^2}$. Observe that $c_1 c_2 = (n+1)^{x_1+x_2} (r')^{n^2}$ for some $r' \in \mathbb{Z}_{n^2}$ hence the result is equivalent to encrypting $x_1 + x_2$ using r' for randomness and making the result a correct encryption of $x_1 + x_2$. We note that the randomization step may be omitted when deterministic computations are acceptable. Output c' .

Paillier.EvalMult($c \in \mathbb{Z}_{n^2}, b \in \mathbb{Z}_n$). Multiplication of a ciphertext $c \in \mathbb{Z}_{n^2}$ by a public constant $b \in \mathbb{Z}_n$ is computed as follows: pick a random $r \in \mathbb{Z}_{n^2}$ and compute

$c' = c^b r^{n^2} \pmod{n^2}$. Observe that $c^b = (n+1)^{bx} (r')^{n^2}$ for some $r' \in \mathbb{Z}_{n^2}$, hence the result is a correct encryption of bx . Again, we emphasize that the randomization step may be omitted when deterministic computations are acceptable. Output c' .

2.3 Performing Arithmetic Operations

Computing statistical tests requires the ability to evaluate arithmetic gates using floating-point integers. However, given that the ciphertext space of Paillier is the ring \mathbb{Z}_n , several non-trivial considerations must be made to smoothly port the statistical test computations, as they would be done on plaintext, to the equivalent computation in \mathbb{Z}_n . This requires thought on two fronts: (1) representing floating point values in \mathbb{Z}_n and (2) dealing with negative values when encoding and computing in \mathbb{Z}_n .

2.3.1 Fixed-Point Encoding

Representation of real numbers in both Paillier and secret sharing schemes is a common problem given the message space consists of elements from \mathbb{Z}_n . Performing arithmetic over real numbers is therefore non-trivial and requires either floating-point or fixed-point encoding. While floating-point representation provides better precision, it is also far less efficient compared to fixed-point representation, at least in the Secure Multi Party Computation (MPC) context [71].

Encoding. Given a public fixed-point precision parameter f denoting bits of precision required per computation, we can approximate any real number $a \in \mathbb{R}$ as $a \approx \lfloor a2^f \rfloor 2^{-f}$. We define $\mathbf{fp}_f(a) = \lfloor a2^f \rfloor$ to be the function encoding real numbers into the corresponding fixed-point representation.

Linear Arithmetic. Addition (and subtraction) of two fixed-point numbers is trivial and can be computed without interaction. Consider real numbers $a, b \in \mathbb{R}$. Let x, y be the fixed-point representations of a and b , respectively. Then x and y each have f -bits of precision which implies $(x + y)/2^f \approx_f (a + b)$ and hence addition of encoded reals has f -bits of precision, as required.

Multiplication. Multiplication of fixed-point numbers is more involved as we need to *scale down* the result to have the correct f bits of precision. Consider real numbers $a, b \in \mathbb{R}$. Let x, y be the fixed-point representations of a and b , respectively. Observe that $xy2^{-2f} \approx_f ab$ which has $2f$ -bits of precision. Hence, we need to *scale down* the product xy by a factor of 2^f in order to obtain the correct precision for the approximation of the product. This can be achieved interactively using the **TruncPR** protocol (described in § 2.4) to obtain $\hat{x}y = (xy)/2^f$ so that the resulting value $\hat{x}y$ has f -bits of precision in the approximation of the product ab . It then follows that $\hat{x}y2^{-f} \approx_f ab$ as desired.

Remark 2.3.1. In subsequent sections, we assume all encrypted and secret shared values are fixed-point encoded reals represented as integers with a corresponding scaling factor 2^f . Hence, whenever we refer to “integers” or “values” the reader may assume we are referring to fixed-point approximations of real numbers unless explicitly stated otherwise.

2.3.2 Signed Encoding

We designate a range of length ω (e.g., $\omega = n$) within \mathbb{Z}_n for the representation of negative integers. More concretely we let the range $[0, \lceil \frac{\omega}{2} \rceil)$ represent the *positive* integers in the range $[0, \lceil \frac{\omega}{2} \rceil)$ and elements in the range $[\lceil \frac{\omega}{2} \rceil, \omega)$ encode *negative* integers in the range $[-\lceil \frac{\omega}{2} \rceil, 0)$. Note that the correctness of both addition and multiplication is preserved with such encoding allowing for efficient representation and arithmetic of signed integer values in \mathbb{Z}_n . Consider integers $a, b \in (-\frac{\omega}{2}, \frac{\omega}{2})$. We prove this in the following theorem.

Theorem 2.3.1. *Signed encoding is correct and preserves arithmetic operations in \mathbb{Z}_n provided arithmetic operations do not overflow.*

Proof. Clearly if $a, b \in [0, \frac{\omega}{2})$ then $a \pmod{n} + b \pmod{n} = a + b \pmod{n}$, likewise for multiplication.

- If $a, b \in (-\frac{\omega}{2}, 0)$ (i.e., a and b are both negative integers), then $a \equiv n - |a| \pmod{n}$ and $b \equiv n - |b| \pmod{n}$. Hence, $a + b \equiv (n - |a|) + (n - |b|) \pmod{n} \equiv$

$n - (|a| + |b|) \pmod n$. Likewise we get $ab \equiv (n - |a|)(n - |b|) \pmod n \equiv |ab| \pmod n \equiv ab \pmod n$ for multiplication.

- If $a \in [0, \frac{\omega}{2})$ and $b \in (-\frac{\omega}{2}, 0)$ (i.e., a positive, b negative) then we get that $a + b \equiv a + (n - |b|) \pmod n \equiv a - |b| \pmod n$ which is correct given that if $a \geq b$ we get $a - |b| \pmod n$ and if $a \leq b$ then we get $n - (|b| - a) \pmod n$. For multiplication we obtain $ab \equiv a(n - |b|) \pmod n \equiv n - a|b| \pmod n$ as needed. The case of a negative and b positive follows by symmetry.

□

2.4 Secure Multi-party Computation (MPC)

Computing statistical tests such as Student's t-test, Pearson Correlation, and Chi-Squared in CUSTODES requires the ability to evaluate addition, multiplication and division gates on encrypted inputs without leaking information. While threshold-Paillier can be used to evaluate linear arithmetic gates, it is necessary to invoke *interactive protocols* to compute non-linear arithmetic gates (i.e., multiplication and division). In this section we introduce the necessary definitions of security for MPC protocols and cover the protocols used in efficiently computing statistical tests.

2.4.1 Universal Composability (UC)

UC is a framework for proving the security of MPC protocols [9, 55]. The UC model formalizes an MPC environment where parties (modeled as interactive Turing-machines) interact to perform a computation. The security of a protocol π is proven by showing that any adversarial environment \mathcal{Z} interacting with honest parties in the execution of π is indistinguishable from an ideal functionality executing π . The importance of this definition is that any protocol π which is UC-secure can be composed with other UC-secure protocols without compromising the security of the composed protocol. More formally, UC-security guarantees that an ideal execution of π in a

malicious (in our case honest-but-curious) environment \mathcal{Z} is indistinguishable to the real execution of π in \mathcal{Z} .

Remark 2.4.1. The UC composability framework is of importance to this work for two reasons: (1) it guarantees that the protocol transcript posted to the tamper-proof ledger does not reveal information about the encrypted inputs (i.e., the dataset \mathcal{D}) and (2) makes it possible to instantiate new and more complex protocol using secure UC-secure sub-protocols without jeopardizing security. We utilize the latter in instantiating the statistical test protocols which are composed exclusively of UC-secure sub-protocols.

2.4.2 Statistical Security

We make extensive use of statistically secure MPC protocols introduced in [11, 10]. The advantage of using statistical security (as opposed to perfect) is that many existing protocols can be rendered far more efficient with only slightly more relaxed notions of security.

Definition 2.4.1 (Statistical Security [11]). For any two random variables X and Y with finite sample spaces U and V , respectively. The statistical distance between X and Y is said to be *statistically indistinguishable* in the security parameter κ if:

$$\Delta(X, Y) = \frac{1}{2} \sum_{w \in U \cup V} |\Pr[X = w] - \Pr[Y = w]| \leq \text{negl}(\kappa)$$

In the special case that $\Delta(X, Y) = 0$ we say the distributions variables are *perfectly indistinguishable* or *information-theoretically secure*.

Remark 2.4.2. Since we use the Paillier encryption scheme as a building-block for instantiating these protocols, the security of *all* MPC protocols (both statistically secure and perfectly secure) is reduced to computational security since the hardness of the Paillier cryptosystem is based on factorization.

2.4.3 Evaluating Non-linear Arithmetic Gates

We used methods described in [11, 10, 18] to compute non-linear arithmetic gates using interactive MPC protocols. We emphasize that while some of the protocols used in this work were originally described for the secret-sharing schemes (i.e., [62]), all protocols apply to a threshold-Paillier setting, even when active security is required [16].

Following notation in [17, 72], we denote a Paillier ciphertext $c \in \mathbb{Z}_{n^2}$ as $[c]$. We leave arithmetic additions, subtractions, and scalar multiplications (local computations) implicit since they can easily be inferred from context, i.e., $[a] + [b]$, denotes the addition of two Paillier encryptions and $[a]c$ denotes the multiplication of an encrypted value by a public scalar. If a value is public, we let $a + [b]$ represent the addition of a “dummy” encryption, $[a]$, of the publicly known quantity a with the (secret) encryption $[b]$.

Remark 2.4.3. In the complexity analysis of MPC protocols, the term *invocation* is often used to describe the number of times a *constant-round* interactive sub-protocol (e.g., `Mult`) is invoked during the execution of a protocol, this convention follows existing MPC literature for providing an upper bound on the round complexity of a given protocol. Round complexity almost entirely determines the run-time of MPC protocols in practice [7] and therefore it is often desirable to invoke protocols which require a constant number of interactive round.

2.5 MPC Protocols

We begin this section by describing existing protocols for evaluating certain arithmetic operations efficiently using MPC. We then describe two derived protocols which we require for computing statistical tests.

Existing Protocols

Reveal $([a])$. Given an encryption $[a]$, obtain a . This protocol requires 1 round and no invocations and is realized by simply invoking the threshold-decryption protocol described in § 2.2. Security of the protocol under the UC-framework is proven in [16].

Mult $([a], [b])$. Given encryptions $[a]$ and $[b]$, obtain the encryption $[ab]$ in 1 round and 1 invocation. Details for instantiating this protocol are found in [16]. As we shall see, multiplication is the fundamental protocol used as a building block for all non-linear gate evaluation (e.g., division can be approximated by repeated multiplication).

PrefixOR $([a_1], \dots, [a_\ell])$. Given a vector of encrypted bits $([a_1], \dots, [a_\ell])$ obtain encrypted vector $([b_1], \dots, [b_\ell])$ of the prefix-OR operation, i.e., $b_i = \bigvee_{j=1}^i a_j$. This protocol is described in [17, 72] and has a total complexity of 17 rounds and 20ℓ invocations.

TruncPR $([a], \ell, m)$. Given encryption $[a]$ and an integer m , obtain the encryption $[\tilde{a}] = [a/2^m]$. The protocol is fundamental to achieving secure fixed-point multiplication in \mathbb{Z}_n as it allows successive multiplications to be performed without “overflow” modulo n . Details for instantiating the protocol are described in [11, 71]. The protocol is statistically secure and has a total complexity of 2 rounds and $2m$ sub-protocol invocations.

BitDec $([a])$. Given encryption $[a]$, obtain the encrypted bit vector $([a_1], \dots, [a_\ell])$ representing the bit decomposition of a . We assume the convention of having the most significant bit be $[a_1]$, in other words, a is decomposed in big-endian format. The description of the protocol is provided in [17, 71] and has a total complexity of 114 rounds and $110\ell \log_2(\ell) + 118\ell$ invocations.

BitLT $(([a_1], \dots, [a_\ell]), ([b_1], \dots, [b_\ell]))$. Given encrypted bit vector representations of a and b , respectively, obtain the encrypted result of the comparison where $c = a \stackrel{?}{<} b$ such that $c \in \{0, 1\}$. In combination with the **BitDec** protocol, this protocol can be used to compare arbitrary encrypted values securely. The instantiation of the protocol is describe in [17, 71] and requires 19 rounds and 22ℓ invocations.

FPDiv($[a], [b], \ell, f$). Given encryptions $[a]$ and $[b]$ obtain the encryption $[a/b]$ with f -bits of fixed-point precision. This protocol is based on the Goldschmidt method for achieving integer division presented in [37]. The idea behind Goldschmidt’s method is to obtain initial approximations to both the divisor and dividend and iteratively compute *quadratically converging* approximations up to a desired precision. The MPC version of the protocol is described in [11, 72], is statistically secure, and has a total complexity of $3 \log_2(\ell) + 2$ rounds and $1.5\ell \log_2(\ell) + 4\ell$ sub-protocol invocations.

Symmetric Boolean Functions. We note that any symmetric boolean function $F : \{0, 1\}^k \rightarrow \{0, 1\}$ can be computed in constant rounds using Lagrange interpolation as described in [17]. While the protocol in [17] assumes a finite field of prime order, the protocol remains correct and secure when used with Paillier [16]. The protocol requires 5 rounds and $6k$ invocations.

2.5.1 Generating Random Bits

Generating random bits is necessary for securely instantiating **TruncPR**, however, most existing protocols for generating shared random bits require secret sharing in a field of prime order since the protocol assumes that computing the square root of an element is feasible. As such, the protocols do not apply to the Paillier setting since computing the square root of elements in the ring \mathbb{Z}_n requires knowing the factorization of n (which would break the security of the scheme). To this extent, we propose Protocol 1 as an alternative to generating encrypted random bits securely in \mathbb{Z}_n by exploiting the ability to compute symmetric Boolean functions and compute $\bigoplus_{i=1}^k [a_i]$ on encrypted inputs $([a_1], \dots, [a_k])$.

Correctness. Correctness follows from the fact that each bit b_j is the xor of the bits b_{ij} provided by party \mathcal{P}_i for $i = 1, \dots, v$. Therefore, by assumption the input bits b_{ij} are random and unknown to all parties, the final bit array will likewise be random and unknown to any party.

Security. Security (in the honest-but-curious model) follows from the use of secure

Protocol 1: $([b_1], \dots, [b_m]) \leftarrow \text{RandBits}(m)$

Public Parameters: $\llbracket \mathcal{D} \rrbracket, N, M, \mathcal{B}, \text{pk}, v, t, \kappa, \ell, f$

Result: Tuple of m encrypted random bits.

```

1 foreach  $i \leftarrow 1, 2, \dots, v$  do parallel
2   |  $([b_{i,1}], \dots, [b_{i,m}]) \leftarrow \text{BitVector}_{\mathcal{P}_i}(m);$     //  $\mathcal{P}_i$  sends  $m$  encrypted bits.
3 foreach  $j \leftarrow 1, 2, \dots, m$  do parallel
4   |  $[b_j] \leftarrow \bigoplus_{i=1}^v [b_{i,j}];$                         // symmetric boolean function eval
5 return  $([b_1], \dots, [b_m]);$ 

```

sub-protocols and the fact that no values are revealed. Moreover, since each b_{ij} is random (and only known to one of the parties), the final result b_j will be a random bit unknown to all parties. This latter property follows directly from the secure XOR evaluation.

Complexity. The protocol requires 2 rounds and m invocations of the symmetric Boolean function evaluation to compute the xor.

2.5.2 Sign Extraction

We require the ability to extract the sign from an encrypted value. While intuitively simple, this task is non-trivial given the nature of signed-encoding in \mathbb{Z}_n , discussed in § 2.3.2. In order to securely extract the sign of an encrypted value, it is necessary to perform an expensive bit decomposition operation followed by a comparison, both of which are round intensive protocols. Fortunately, this protocol is only required once per statistical test and can be avoided if the sign is deemed of no importance for a given dataset. We present protocol 2 to extract the sign of an encrypted value.

Correctness. Observe that line 1 shifts a to the range $[0, 2^{\ell+1}]$. Hence, if $a \leq 0$ then $b \geq 0$. Line 2 converts b into a bitwise representation which is then used in line 4 to compare with the threshold of 2^ℓ . If $a \leq 0$ then $b \leq 2^\ell$ hence the protocol is correct.

Security. Security follows from the use of secure and composable sub-protocols and the fact that no values are revealed during protocol execution.

Protocol 2: $[s] \leftarrow \text{SignBit}([a])$

Public Parameters: $\llbracket \mathcal{D} \rrbracket, N, M, \mathcal{B}, \text{pk}, v, t, \kappa, \ell, f$

Result: Obtain the sign bit of $[a]$. $s \in \{0, 1\}$, $s = 1$ iff $a < 0$.

```

1  $[b] \leftarrow [a] + 2^\ell;$ 
2  $([b_1], \dots, [b_\ell]) \leftarrow \text{BitDec}([b]);$ 
3  $c \leftarrow \text{BigEndian}(2^\ell);$            // big-endian representation of  $2^\ell$ 
4  $[s] \leftarrow \text{BitLT}([b_1], \dots, [b_\ell], c);$ 
5 return  $[s];$ 

```

Complexity. The protocol requires 2 rounds and 2 invocations. Note, however, that the two invocations are round-intensive protocols which have significant (albeit constant-round) overhead.

2.5.3 Computing Fixed-Point Square Roots

Finding efficient methods for computing fixed-point reciprocals and square roots in MPC is still an open problem. The most efficient solutions found so far are based on iterative methods with quadratic convergence (requiring a logarithmic number of iterations in the precision parameter) but require an expensive normalization step to bring the secret value into a specific range. The two most commonly used methods are Newton-Raphson which are based on root-finding algorithms and Goldschmidt's method based on series approximations[30, 56]. In practice, Goldschmidt's method is more efficient compared to Newton-Raphson by requiring fewer sequential multiplications per round thus reducing the total round complexity of the protocol.

Goldschmidt's method applies to computing division, reciprocals, square roots, and square-root reciprocals and uses similar iterative techniques to achieve each one. As we shall see in subsequent chapters, the statistical tests we describe require dividing by the square root of an encrypted real number (approximated using fixed-point representation, see § 2.3.1). While several works provide details for computing division using Goldschmidt's algorithm in MPC [10, 71], we are not aware of a body of work describing computations of square-root and square-root reciprocals in the MPC

context. To this end, we describe the necessary protocols for computing the reciprocal of the square-root in MPC which we later use in the computations of statistical tests.

To achieve quadratic convergence, Goldschmidt's method requires an initial approximation to the reciprocal of the square-root of the value a , i.e., $Y_0 \approx 1/\sqrt{a}$ such that $1/2 \leq aY_0^2 \leq 3/2$. In standard uses of this algorithm (e.g., non-MPC contexts), this initial approximation is achieved using look up tables [56]. However, given that in our case a must be kept secret, computing the initial approximation requires more finesse (and, unfortunately, more rounds).

We achieve this initial approximation by first decomposing a into its bit-wise encrypted representation using the **BitDec** protocol and then isolating the MSB of a in order to securely obtain the following initial approximation to the reciprocal of the square root:

$$Y_0 = 1.0/\sqrt{2^{\text{msb}(a)+1}} \approx 1/\sqrt{a}$$

This initial approximation is achieved in MPC using Protocol 3. Once the approximation is obtained (which is the most expensive step in computing the reciprocal of the square root), we can easily compute quadratically converging iterative approximations using the following procedure [56]. Let $b_0 = a$ and $z_0 = Y_0$ then for each subsequent iteration we set:

$$b_{i+1} = b_i Y_i^2$$

$$Y_{i+1} = (3 - b_{i+1})/2$$

$$z_{i+1} = z_i Y_{i+1}$$

Following these rules we obtain the converging approximation z_{i+1} of the reciprocal square root, i.e.,

$$\lim_{i \rightarrow \infty} aY_0^2 Y_1^2 \dots Y_i^2 = 1 \implies Y_0 Y_1 \dots Y_i = \frac{1}{\sqrt{a}}$$

If we set the number of iterations to θ , then we obtain the following approximation

with a relative error $\epsilon_\theta = \epsilon_0^{2^\theta}$, where ϵ_0 is the error in the initial approximation.

$$\frac{1}{\sqrt{a}} \approx Y_0 Y_1 Y_2 \dots Y_{\theta-1}$$

Since for our applications we require f bits of precision, we set $\theta = \lceil \log_2(f) \rceil$. Protocol 4 computes the reciprocal of the square root and closely follows the above description.

Protocol 3: $[\hat{a}] \leftarrow \text{InitSqrt}([a])$

Public Parameters: $\llbracket \mathcal{D} \rrbracket, N, M, \mathcal{B}, \text{pk}, v, t, \kappa, \ell, f$

Result: Initial approximation to the reciprocal of the square root of a .

```

1  $([b_1], \dots, [b_\ell]) \leftarrow \text{BitDec}([a], \ell);$ 
2  $([c_1], \dots, [c_\ell]) \leftarrow \text{PrefixOR}([b_1], \dots, [b_\ell]);$ 
3  $[d_\ell] \leftarrow [c_\ell] - [c_{\ell-1}];$ 
4 foreach  $i \leftarrow 1, 2, \dots, \ell - 1$  do parallel
5    $[d_{\ell-i}] \leftarrow [c_{\ell-i}] - [c_{\ell-i-1}];$ 
6  $[e_\ell] \leftarrow [d_\ell];$ 
7 foreach  $i \leftarrow 1, 2, \dots, \ell - 1$  do
8    $[e_{\ell-i}] \leftarrow [d_{\ell-i}] \text{fp}_f(1.0/\sqrt{2^{i+1}});$ 
9  $[\hat{a}] \leftarrow \sum_{i=1}^{\ell} [e_i]$ 
10 return  $[\hat{a}];$ 

```

Correctness. Lines 3 to 5 isolate the MSB of a by computing the vector $([d_1], \dots, [d_\ell])$ containing 0's everywhere except at the MSB index which is an encryption of 1. Finally, lines 5 to 8 compute the approximation using the isolated MSB.

Security. Security follows from the use of secure sub-protocols and the fact that no values are revealed during the execution of the protocol.

Complexity. The protocol requires 2 rounds and 2 invocations. Notice that the complexity of the protocol is dominated by the `BitDec` protocol making it highly inefficient in practice, even if the round complexity is constant.

Protocol 4: $[z] \leftarrow \text{FPSqrtRcpr}([a])$

Public Parameters: $\llbracket \mathcal{D} \rrbracket, N, M, \mathcal{B}, \text{pk}, v, t, \kappa, \ell, f$

Result: Square root reciprocal $1.0/\sqrt{a}$ of a .

```

1  $\theta \leftarrow \lceil \log_2(f) \rceil;$ 
2  $[y] \leftarrow \text{InitSqrt}([a]);$ 
3  $[b] \leftarrow [a];$ 
4  $[z] \leftarrow [y]$ 
5 foreach  $i \leftarrow 1, 2 \dots \theta$  do
6    $[y^2] \leftarrow \text{Mult}([y][y]);$ 
7    $[y^2] \leftarrow \text{TruncPR}([y^2], 2\ell, f);$ 
8    $[b] \leftarrow \text{Mult}([b], [y^2]);$ 
9    $[b] \leftarrow \text{TruncPR}([b], 2\ell, f);$ 
10   $[y] \leftarrow (3 - b)\text{fp}_f(0.5);$ 
11   $[z] \leftarrow \text{Mult}([z], [y]);$ 
12   $[z] \leftarrow \text{TruncPR}([z], 2\ell, f);$ 
13 return  $[z]$ 

```

Correctness. Line 1 computes the number of iterations necessary to achieve f -bits of precision in the final result. Line 2 computes the initial approximation to the reciprocal of the square root. Correctness follows from the description of the algorithm.

Security. Security follows from the use of secure sub-protocols and that no secret values are revealed during the execution of the protocol.

Complexity. The protocol requires $O(\theta)$ rounds and $O(\theta)$ invocations.

2.6 Practical Considerations

In practice, it may be desirable to perform more computationally intensive interactive protocols such as **SignBit**, **FPDiv**, and **FPSqrtRcpr** using techniques for multi-party computation over a linear secret sharing scheme (LSS) such as Shamir [62]. It has been shown that general multi-party computation can be instantiated over any LSS scheme, even when active security is required [15]. By using LSS as the underlying

primitive, we can perform computations over a finite field F_p of prime order which comes with several *computational* efficiency improvements (note: the number of interactive rounds remains the same). The downside of this approach, however, is that shares of secret values must be distributed to all v parties. Therefore, we only use LSS for computing a fixed number of gates (e.g., division gates) which only require two values (the inputs to the gate) to be shared with all parties rather than the full dataset. Furthermore, this approach does not change the overall design of CUSTODES but in some cases can improve the overall runtime of computations. Specifically, the performance improvement is noticeable for protocols requiring a large number of multiplications since Paillier requires one modular exponentiation for every *interactive* multiplication, while multiplication with LSS do not [15]. The question, of course, becomes how to integrate secret sharing protocols with Paillier based computations as it necessitates conversion between Paillier encryptions in the ring of integers \mathbb{Z}_n to secret shares in a finite field \mathbb{F}_p where shares of the secret values are distributed among the parties. Moreover, this conversion must be achieved *on the fly*, without a trusted dealer to distribute the shares prior to protocol execution.

We extend the method described in [21] for converting shares between fields. First, we describe a fairly simple protocol for generating a secret shared random value along with the Paillier encryption of the same value. This can be achieved using a similar technique as for generating shared random values described in [17, 16]. Let $\langle s \rangle$ denote a secret share of the (secret) value s in the finite field \mathbb{F}_p . Each party $\mathcal{P}_i \in \{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ generates a random $r_i \in [0, 2^{\ell+\kappa})$ (using methods for generating shared random values [20]) and sends a tuple $([r_i], \langle r_i \rangle)$ to all other parties. Observe that the tuple contains the equivalent random value in both the ring \mathbb{Z}_n and field \mathbb{F}_p (assuming that $p > n$), where the first random value is encrypted and the other secret shared among the parties. The parties then locally compute $[r] = \sum_{i=1}^v [r_i]$ and $\langle r \rangle = \sum_{i=1}^v \langle r_i \rangle$ to obtain a random value in \mathbb{Z}_n and in secret shared form such that the final value remains random and unknown to all parties (in the honest-but-curious security model though it can be made secure against active adversaries as well [21]).

We briefly summarize the two fundamental protocols for LSS based multi-party

computation. We refer the reader to [5] for a full summary of LSS-based multi-party computation techniques.

CreateShares(s). Generates a polynomial g with random coefficients from the field \mathbb{F}_p such that $g(0) = s$. Each party \mathcal{P}_i obtains a share $g(i)$ for $i = 1 \dots v$. Denote the secret shares of s distributed among the parties $\mathcal{P}_1 \dots \mathcal{P}_v$ as $\langle s \rangle = g(0) = s$.

Reveal($\langle s \rangle$). Receive a set of t unique shares from t (or more) parties. Reconstruct the polynomial g by interpolating the secret shares and output the secret $s = g(0)$.

Protocol 5 converts shares from \mathbb{Z}_n to the field \mathbb{F}_p where p is some prime determined by either the parties or by the data owner at setup time.

Protocol 5: $\langle a \rangle \leftarrow \text{ConvertToShare}([a])$

Public Parameters: $\llbracket \mathcal{D} \rrbracket, N, M, \mathcal{B}, \text{pk}, v, t, \kappa, \ell, f$

Result: Secret shared value of $\langle a \rangle$.

```

1  $([r], \langle r \rangle) \leftarrow \text{RandIntFields}(2\ell + \kappa, n, p);$ 
2  $[\hat{a}] \leftarrow [a] + 2^\ell;$  // shift to positive range
3  $b \leftarrow \text{Reveal}([\hat{a}] + [r]);$ 
4  $\langle c \rangle \leftarrow \text{CreateShares}(b);$  // distribute shares to all parties
5  $\langle \hat{a} \rangle \leftarrow \langle c \rangle - \langle r \rangle;$ 
6  $\langle a \rangle \leftarrow \langle \hat{a} \rangle - 2^\ell;$  // restore sign
7 return  $\langle a \rangle;$ 
```

Correctness. Line 1 generates a random value r in both the ring \mathbb{Z}_n and field \mathbb{F}_p as described above such that $r \pmod{p} \equiv r \pmod{n}$. Line 3 reveals the statistically hidden value $a + 2^\ell + r$. The share $\langle a + 2^\ell + r \rangle$ is then created and distributed to all parties. Finally, the parties locally subtract 2^ℓ and the share $\langle r \rangle$ to obtain $\langle a \rangle$ restored to the correct signed encoding \pmod{p} .

Security. Observe that $[r]$ and $\langle r \rangle$ are random integers in the range $[0, 2^{\ell+\kappa+v}]$ unbeknownst to all parties, therefore, $\Delta(a, r) \leq 2^\kappa$ providing κ -bits of statistical security.

Complexity. The total complexity of the protocol is 3 rounds and 3 invocations.

2.7 Relevant Notation

A summary of notation appears in Table 2.2.

\mathcal{D}	A dataset containing rows and columns with real values.
$\llbracket \mathcal{D} \rrbracket$	Encrypted form of the dataset \mathcal{D} . Each cell of $\llbracket \mathcal{D} \rrbracket$ is an encryption of the fixed-point approximation to the real value found in \mathcal{D} .
v	Number of computing parties.
t	Decryption threshold.
τ	Counter of statistical tests executed on \mathcal{D} .
(t, p)	Tuple containing the test-statistic t and corresponding p-value p .
\mathcal{T}	Arithmetic circuit of a statistical test.
ψ	Certificate associated with a test result (p-value).
\mathcal{R}	Researcher performing tests on $\llbracket \mathcal{D} \rrbracket$ where \mathcal{R} may be in the set of computing parties $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$.
\mathcal{P}^{pk}	The public key identifier of party \mathcal{P} .
\mathcal{B}	Tamper-proof ledger (e.g., Blockchain).
n	The Paillier scheme modulus.
\mathbb{Z}_n	The Paillier scheme's message space.
$[a]$	Paillier encrypted value (ciphertext)
N	Number of rows in the dataset \mathcal{D} .
M	Number of attributes (columns) in the dataset \mathcal{D} .
ℓ	Number of bits required to represent the largest integer of an arithmetic computation (e.g., $\ell = 64$ for most applications).
f	Bits of precision in fixed-point arithmetic computation.
κ	Statistical security parameter for interactive protocols.
$\text{fp}_f(\cdot)$	Function encoding reals to fixed-point representation with f -bits of resulting precision.

Table 2.2: Summary of notation used throughout this work.

Chapter 3

System Construction

3.1 Framework

With the necessary preliminaries behind us, we are now ready to describe the CUSTODES framework. The functionality of CUSTODES can be split into three phases: *Setup*, *Compute*, and *Audit*. We first present the general framework followed by a concrete instantiation using the Paillier encryption scheme and methods for multi-party computations covered in the previous chapter.

Setup. A data owner publishes an (encrypted) dataset $\llbracket \mathcal{D} \rrbracket$ under the public key \mathbf{pk} . The owner proceeds to distribute the secret key shares to parties $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ such that any threshold t of them can collectively compute on the encrypted data and decrypt $\llbracket \mathcal{D} \rrbracket$ (see § 2.2). The owner then initializes a Blockchain \mathcal{B} by posting to it a signed message $(0, \mathbf{pk}, \perp, \perp)$ that corresponds to the counter of the tests to be executed on \mathcal{D} set to zero. In addition to releasing the encrypted dataset, the data owner releases metadata pertaining to \mathcal{D} deemed sufficient for researchers to form hypotheses on the dataset. We formalize the metadata requirements in § 3.3.

Compute. A researcher, say \mathcal{R} , specifies the test \mathcal{T} corresponding to a statistical test that she wants to execute on \mathcal{D} . We note that \mathcal{R} can either be one of the parties in the set $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ approved by the data owner during the setup of CUSTODES or an independent entity granted computing access by the owner and/or parties. \mathcal{R}

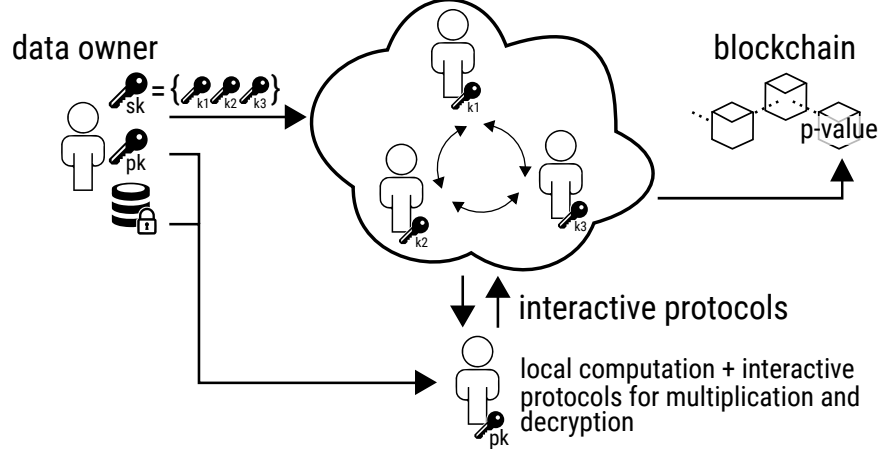


Figure 3-1: Overview of Paillier-based instantiation of CUSTODES.

posts a message $(\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, \perp)$ to \mathcal{B} marking the start of a statistical test computation \mathcal{T} (described as an arithmetic circuit). For simplicity, we assume that only one test is executed at a time though we note that this is not a necessary requirement since the dataset is static and \mathcal{B} ensures that every test is assigned a sequential test index τ . Once the test is computed, the result of the test is made available only in an encrypted form as a result of *secure computation*. Recall that the test result cannot be recovered by *any* of the parties individually and requires a threshold number of the parties to reach consensus in order to reveal it. To this end, \mathcal{R} requests help from $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ by posting a signed message to \mathcal{B} along with the (encrypted) result. Each party $\mathcal{P}_i \in \{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ verifies that the message indeed came from one of the researchers allowed to run the tests and posts a message to \mathcal{B} with a partially decrypted share of the final result $([\mathbf{t}], [\mathbf{p}])$ consisting of the test statistic \mathbf{t} and p-value \mathbf{p} . When combined, the shares reveal the decrypted result (\mathbf{t}, \mathbf{p}) . This specification ensures that each test result is made publicly available (to all parties). The final entry recorded on \mathcal{B} is the test certificate consisting of the tuple $(\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, (\mathbf{t}, \mathbf{p}))$ where \mathcal{R}^{pk} is the identifier of the researcher \mathcal{R} , \mathcal{T} is the arithmetic test circuit. This final tuple is signed by all parties engaged in the reveal computation (i.e., the parties revealing the result) since after the shares are revealed, each party may individually reconstruct and ensure that the shares indeed correspond to the correct decryption.

Audit. A test with certificate $(\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, (\mathbf{t}, \mathbf{p}))$ can be audited for correctness by any

entity with access to \mathcal{B} , $\llbracket \mathcal{D} \rrbracket$ and the public key \mathbf{pk} used to encrypt \mathcal{D} . An auditor \mathcal{V} retrieves all messages related to τ from \mathcal{B} and verifies the signature of each message, including the certificate. \mathcal{V} then proceeds to use this information to ensure the test code was computed correctly and indeed produces the result (\mathbf{t}, \mathbf{p}) . We require that \mathcal{B} contains sufficient information to verify whether $(\mathbf{t}, \mathbf{p}) = \mathcal{T}(\mathcal{D})$, even if all parties are offline, while simultaneously ensuring \mathcal{V} gains no information about \mathcal{D} (except for the result of the statistical test). In particular, any entity with access to \mathcal{B} (e.g., a researcher, an auditor, a data owner, or a third party) must have the ability to verify the computed statistical tests that have been run using CUSTODES: for every tuple $(\tau, \mathcal{R}^{\mathbf{pk}}, \mathcal{T}, (\mathbf{t}, \mathbf{p}))$ anyone can verify that (\mathbf{t}, \mathbf{p}) is the correct output. Hence, if $\mathcal{T}(\mathcal{D}) \neq (\mathbf{t}, \mathbf{p})$, \mathcal{R} 's malicious behaviour is exposed and the result (\mathbf{t}, \mathbf{p}) is deemed invalid.

3.2 Implementation

We are now ready to describe the detailed construction of CUSTODES. By using a threshold additively-homomorphic encryption scheme for encrypting the dataset, and distributing the key among a set of (possibly untrusted) parties, we can construct CUSTODES with minimal overhead on the researchers and parties engaged in the system. Indeed, we achieve a construction which only requires the active participation of parties in the network in two cases: 1) when computing a non-linear arithmetic gate in the statistical test circuit evaluated over the encrypted dataset $\llbracket \mathcal{D} \rrbracket$ (as explained above) and 2) for decryption of the result of the computation (i.e., to reveal the test statistic).

Recall that we divide the overall protocol into three distinct phases (Setup, Compute and Audit).

Setup. During the setup phase, the data owner executes algorithm $\text{Paillier.KeyGen}(1^k)$ ¹ that outputs a public key \mathbf{pk} and secret key shares $\mathbf{sk}_1, \dots, \mathbf{sk}_v$. The shares $\mathbf{sk}_1, \dots, \mathbf{sk}_v$ are distributed to parties $\mathcal{P}_1, \dots, \mathcal{P}_v$, respectively.

¹In practice $k \geq 1024$ to ensure security against computationally bounded adversaries.

The data owner then runs $\text{Paillier}.\text{Encrypt}(\mathbf{pk}, \mathcal{D})$ to obtain the encrypted dataset $\llbracket \mathcal{D} \rrbracket$, where every value of \mathcal{D} is individually encrypted. The data owner sends the tuple $(\mathbf{pk}, \mathbf{sk}_i)$ to $\mathcal{P}_i \in \{\mathcal{P}_1, \dots, \mathcal{P}_v\}$, $\forall i = 0 \dots v$ and publishes the public key \mathbf{pk} and $\llbracket \mathcal{D} \rrbracket$. The data owner then initializes a Blockchain \mathcal{B} and records the digital identity of every party $\mathcal{P}_i \in \{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ and researcher \mathcal{R} that can run the protocol (e.g., it can be the verification key of a digital signature that will be used to verify \mathcal{P}_i 's signatures) on \mathcal{B} . The owner then posts the signed message $(0, \mathbf{pk}, \perp, \perp)$ corresponding to the counter of the tests to be executed on \mathcal{D} set to zero. In addition to releasing the dataset, the data owner makes available metadata pertaining to the dataset (e.g., size of the dataset, attributes, etc). With the exception of dataset size, the choice of metadata to be released is left up to the data owner.

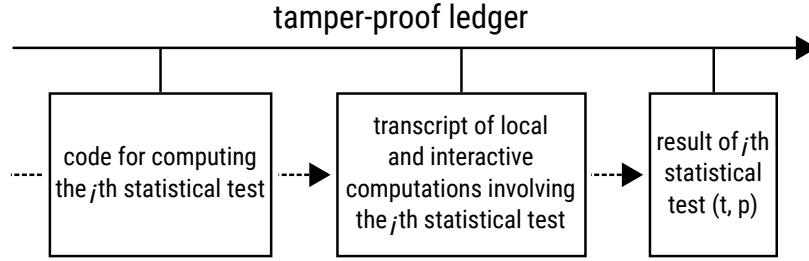


Figure 3-2: Use of Tamper-proof ledger in CUSTODES.

Compute. Let \mathcal{P} be the researcher who wishes to run a statistical test \mathcal{T} represented as an arithmetic circuit with input $\llbracket \mathcal{D} \rrbracket$. \mathcal{R} posts $(\tau, \mathcal{R}^{\mathbf{pk}}, \mathcal{T}, \perp)$ to \mathcal{B} and obtains counter τ . \mathcal{R} then proceeds to deterministically evaluate the arithmetic test circuit $\mathcal{T}(\llbracket \mathcal{D} \rrbracket)$ locally by using the homomorphic properties of the Paillier scheme. Assuming the arithmetic circuit contains a set of non-linear gates, \mathcal{P} is only able to evaluate the first $w - 1 \geq 0$ gates locally before reaching some gate w which requires an interactive protocol. Let $\text{op}_w(\mathcal{C}_w)$ denote this computation where op_w is the arithmetic gate and \mathcal{C}_w is the tuple of ciphertexts that \mathcal{R} obtained from local computations up to level $w - 1$. Suppose, op_w is a multiplication gate and \mathcal{C}_w contains ciphertexts $[a]$ and $[b]$. \mathcal{R} posts to \mathcal{B} the message tuple $(\tau, \mathcal{R}^{\mathbf{pk}}, \text{op}_w, \mathcal{C}_w)$. All parties receive this message and proceed to engage in the interactive protocol **Mult** to compute $\text{op}_w(\mathcal{C}_w)$ and obtain the encrypted result of this computation $[c] = [ab]$. Every party \mathcal{P}_i then

posts a signed message tuple $(\tau, \mathcal{P}_i^{\text{pk}}, \text{op}_w, [c])$ to \mathcal{B} . \mathcal{R} then uses $[c]$ and proceeds with the local computation(s). In general, for the j th interactive protocol required during the evaluation of \mathcal{T} , \mathcal{R} posts a message $(\tau, \mathcal{R}^{\text{pk}}, \text{op}_j, \mathcal{C}_j)$ and the parties engage in the protocol, posting the result of op_j to \mathcal{B} . Finally, once the evaluation of $\mathcal{T}(\llbracket \mathcal{D} \rrbracket)$ is completed, \mathcal{R} requests the decryption of the result $([t], [p]) = \mathcal{T}(\llbracket \mathcal{D} \rrbracket)$ by posting the tuple $(\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, ([t], [p]))$ to \mathcal{B} . Once the result is obtained, the certificate $\psi = (\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, (t, p))$, which, in conjunction with the information posted to \mathcal{B} , certifies the result of the hypothesis test (t, p) . Figure 3-2 illustrates the use of the tamper-proof ledger in the Compute process.

Audit. Suppose an auditor \mathcal{V} wishes to verify that the certificate $\psi = (\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, (t, p))$ indeed certifies the result (t, p) . \mathcal{V} retrieves $\llbracket \mathcal{D} \rrbracket$ and proceeds to evaluate the test circuit $\mathcal{T}(\llbracket \mathcal{D} \rrbracket)$ until the first non-linear operation (which required an interactive protocol during the Compute phase). Let op'_w denote the first such operation and \mathcal{C}'_w denote its ciphertext list. \mathcal{V} retrieves $(\tau, \mathcal{R}^{\text{pk}}, \text{op}'_w, \mathcal{C}'_w)$ from \mathcal{B} and verifies that $\text{op}' = \text{op}$ and $\mathcal{C}'_w = \mathcal{C}_w$. \mathcal{V} then gathers the transcript resulting from the computation of op_w from \mathcal{B} and reconstructs $[c]$. Note that \mathcal{V} does *not* engage in an interactive protocol to obtain $[c]$. \mathcal{V} then proceeds with the evaluation of \mathcal{T} using $[c]$ as input and continues these verification steps for every gate in \mathcal{T} until obtaining a result $([t'], [p']) = ([t], [p])$ at which point \mathcal{V} accepts the certificate ψ as valid. Conversely, if $([t'], [p']) \neq ([t], [p])$, \mathcal{V} rejects the certificate. We omit the verification of ciphertexts and other verification since we assume all parties adhere to protocol. However, we note that there exist numerous ways to augment the audit procedure by requiring various cryptographic proofs from the computing parties as described in [16].

Finally, once the certificates have been verified, \mathcal{V} can verify whether the FDR control procedure described in § 1.3.4 were applied correctly when accepting (resp. rejecting) the τ th hypothesis by ensuring $mFDR_\eta(\tau) \leq \alpha$ for the specified η and α parameters.

3.3 Security Evaluation

We now argue that CUSTODES satisfies properties outlined in § 1.3.

Correctness. CUSTODES follows the plaintext execution of statistical tests by using a secure version of the sub-protocols as presented in § 3.4. The main difference with plaintext execution of these tests is in the accuracy of the results since CUSTODES guarantees f -bits of precision.

Confidentiality. The aim of CUSTODES is to enforce p-value calculations in a truthful manner. It achieves this goal by hiding the content of \mathcal{D} and revealing only the metadata information about the dataset \mathcal{D} sufficient to carry out statistical tests in addition to revealing the results of the statistical tests themselves. Specifically, the data owner is only required to reveal the following metadata:

- **The size of the dataset:** both the number of columns (attributes) and number of rows in each attribute, N and M , respectively. These values are necessary for both computing statistical tests and determining the associated p-values.
- **Attribute metadata:** information on the contents of \mathcal{D} such as the characteristics of each attribute, the domain size, and independence from other attributes. For example, metadata for an “age” attribute may be the set $\{\text{AttrType: Age, NumericRange: 0-110}\}$. We stress, however, that the metadata can be made general and independent of the values in \mathcal{D} when deemed of no influence on computation correctness.

The above information can be expressed as a function $\mathcal{L}_s : \mathbb{D}_{N \times M} \rightarrow \{0, 1\}^*$, that takes as input the dataset and returns N , M and other metadata encoded as a binary strings

We capture the confidentiality property of CUSTODES using a common experiment used in cryptography where an adversary is required to distinguish whether it is set in the real or an ideal world. In the real world, the adversary (i.e., a entity modeling the collusion of malicious parties) interacts with CUSTODES as it would in the real setting. In the ideal world, the adversary interacts with a simulator who has access only to

the output of $\mathcal{L}_s(\mathcal{D})$ and to a test result oracle $\mathcal{O}(\mathcal{D}, \cdot)$. Observe that the simulator does not have access to the dataset itself. The oracle $\mathcal{O}(\mathcal{D}, \cdot)$ knows the database \mathcal{D} and takes as input a statistical test circuit which it evaluates on \mathcal{D} and returns its result. We note that \mathcal{O} is a concept that is used only as part of the confidentiality definition and the proof thereof. In particular, it is used to capture the fact that the simulator is not given \mathcal{D} but only the results of the tests. If one can show that an adversary cannot distinguish the real from the ideal world then CUSTODES does not reveal more about \mathcal{D} than what the simulator knows about \mathcal{D} . Otherwise, the adversary could use this leaked information to distinguish between the two worlds. We formally capture this experiment below.

Definition 3.3.1. Let $\text{CUSTODES} = (\text{Setup}, \text{Compute}, \text{Audit})$, let \mathcal{L}_s be a stateful metadata function, $\mathcal{O}(\mathcal{D}, \cdot)$ a test result oracle, v is the number of parties and t is the threshold of parties required for decryption. Consider the following probabilistic experiments where \mathcal{A} is an honest-but-curious, stateful adversary and \mathcal{S} is a stateful simulator:

RealCUSTODES $_{\mathcal{A}}(1^k, v, t)$: An adversary \mathcal{A} chooses \mathcal{D} and a challenger \mathcal{C} runs $\text{Setup}(1^k, \mathcal{D}, v, t)$ and obtains the encrypted dataset $\llbracket \mathcal{D} \rrbracket$ and keys $(\text{pk}, \{\text{sk}_1, \text{sk}_2, \dots, \text{sk}_v\})$. *WLOG* the parties are split into two disjoint sets $\{\mathcal{P}_1, \dots, \mathcal{P}_{t-1}\}$ and $\{\mathcal{P}_t, \dots, \mathcal{P}_v\}$ where the first set is controlled by \mathcal{A} and the second by \mathcal{C} who acts on behalf of honest parties. \mathcal{C} sends $\llbracket \mathcal{D} \rrbracket$, pk , and corresponding secret keys to all the parties $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$. \mathcal{A} then adaptively chooses q test queries $\{\mathcal{T}_1, \dots, \mathcal{T}_q\}$ where each \mathcal{T}_i corresponds to a valid statistical test circuit. For each \mathcal{T}_i , \mathcal{A} and \mathcal{C} engage in the **Compute** protocol that returns result \mathbf{t}_i . In the end, \mathcal{A} outputs a bit b .

IdealCUSTODES $_{\mathcal{A}, \mathcal{S}}(1^k, v, t)$: An adversary \mathcal{A} chooses a dataset \mathcal{D} and \mathcal{S} is given the output of $\mathcal{L}_s(\mathcal{D})$. \mathcal{S} runs $\text{Setup}_{\mathcal{S}}(1^k, \mathcal{L}_s(\mathcal{D}), v, t)$ that returns an encryption of a random dataset $\llbracket \mathcal{D}_R \rrbracket$ and $(\text{pk}, \{\text{sk}_1, \text{sk}_2, \dots, \text{sk}_v\})$. \mathcal{S} also updates its state. *WLOG* the parties are split into two disjoint sets $\{\mathcal{P}_1, \dots, \mathcal{P}_{t-1}\}$ and $\{\mathcal{P}_t, \dots, \mathcal{P}_v\}$ where the first set is controlled by \mathcal{A} and the second by \mathcal{S} . \mathcal{A} then adaptively chooses q

test queries $\{\mathcal{T}_1, \dots, \mathcal{T}_q\}$ where each \mathcal{T}_i corresponds to a valid statistical test circuit. For each \mathcal{T}_i , \mathcal{S} calls $\mathcal{O}(\mathcal{D}, \mathcal{T}_i)$ and gets the result \mathbf{t}_i . \mathcal{S} then calls $\text{Compute}_{\mathcal{S}}(\mathbf{t}_i)$ where $\text{Compute}_{\mathcal{S}}$ is an interactive protocol that simulates the interaction of the honest parties in the **Compute** phase with parties controlled by the adversary. In the end, \mathcal{A} outputs a bit b .

We say that **CUSTODES** is $(\mathcal{L}_s, \mathbf{t}_1, \dots, \mathbf{t}_q)$ -secure if \exists PPT simulator \mathcal{S} such that for all PPT adversaries \mathcal{A} ,

$$\left| \Pr[\text{RealCUSTODES}_{\mathcal{A}}(1^k, v, t) = 1] - \Pr[\text{IdealCUSTODES}_{\mathcal{A}, \mathcal{S}}(1^k, v, t) = 1] \right| \leq \text{negl}(k)$$

Theorem 3.3.1. **CUSTODES** is $(\mathcal{L}_s, \mathbf{t}_1, \dots, \mathbf{t}_q)$ -secure, i.e., no information beyond the metadata and the result of q statistical tests on \mathcal{D} is revealed.

Proof (Sketch). Since the statistical test evaluation function is secure under protocol composition (see § 2.4.1), there exists a simulator \mathcal{S}' which given input $(\text{pk}, \{\text{sk}_t, \dots, \text{sk}_v\}, [\mathbf{t}_i])$ indistinguishably simulates the view of parties $\{\mathcal{P}_t, \dots, \mathcal{P}_v\}$ while interacting with \mathcal{A} to produce the secure output $[\mathbf{t}_i]$. Using \mathcal{S}' construct simulator \mathcal{S} which reveals each test statistic \mathbf{t}_i . Given input, $(\text{pk}, \{\text{sk}_t, \dots, \text{sk}_v\}, \mathcal{L}_s(\mathcal{D}, v))$, \mathcal{S} runs \mathcal{S}' with input $(\text{pk}, \{\text{sk}_t, \dots, \text{sk}_v\}, [\mathbf{t}_i])$ where \mathcal{S} obtains $[\mathbf{t}_i]$ from the oracle \mathcal{O} and \mathcal{S}' indistinguishably simulates the execution of the test statistic computation. \mathcal{S} proceeds to engage with \mathcal{A} in the threshold-decryption protocol from § 2.2 using keys from the set $\{\text{sk}_t, \dots, \text{sk}_v\}$ to simulate parties $\mathcal{P}_t, \dots, \mathcal{P}_v$ and obtain the test result \mathbf{t}_i . \square

Access Control. Though anyone can homomorphically compute on $\llbracket \mathcal{D} \rrbracket$, the obtained encrypted result cannot be decrypted unless a threshold number of $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ collude. Moreover, each party performs a partial decryption of a result only if the decryption request message was posted on \mathcal{B} and signed by one of the approved parties. Hence, decryption of a ciphertext can only occur if an approved party or researcher requested the decryption.

Verifiability. This property is guaranteed again by the fact that a threshold number of the parties are required to decrypt a result. That is, even if someone computes on

$\llbracket D \rrbracket$, the obtained result is encrypted and will not be decrypted unless more than t of the parties collude. However, every test whose result (\mathbf{t}, \mathbf{p}) is decrypted has partial decryption shares of (\mathbf{t}, \mathbf{p}) recorded on \mathcal{B} . Hence, even if the party \mathcal{P} that initiated the test goes offline after requesting a decryption of (\mathbf{t}, \mathbf{p}) , the certificate on (\mathbf{t}, \mathbf{p}) can be reconstructed from shares stored on \mathcal{B} .

Auditability. Since all *local* computations are deterministic and the transcripts of *interactive* protocols are recorded on \mathcal{B} , given a certificate $\psi = (\tau, \mathcal{P}^{\mathbf{pk}}, \mathcal{T}, (\mathbf{t}, \mathbf{p}))$ an auditor \mathcal{V} can evaluate the arithmetic circuit $\mathcal{T}(\llbracket D \rrbracket)$ *without interaction* and ensure that each input to a non-linear gate corresponds to a local computation of $\mathcal{T}(\llbracket D \rrbracket)$. Note that it suffices to ensure the input is correct given that the output is signed by all parties during the computation. We stress that an audit does not require any MPC evaluations given that the transcript and (encrypted) result is available publicly on \mathcal{B} . Thus, an audit of a certificate ψ is essentially a linear scan of records obtained from \mathcal{B} and local homomorphic ciphertext evaluations which are highly efficient.

3.4 Evaluating Statistical Tests

We showed how a dataset can be encrypted to ensure that every tested hypothesis is locked (i.e., by being encrypted) such that the FDR control procedure can be enforced in an auditable way. However, one important piece remains: How do researchers execute statistical tests over the encrypted dataset in an efficient way. In this chapter we present pseudo code for computing the three common statistical tests: Student’s t-test, Pearson Correlation and Chi-Squared. While not exhaustive, this trio of statistical tests forms a basis for quantitative analysis and covers many use cases: from reasoning about population means to analyzing differences between sets of categorical data. We stress that CUSTODES can be also easily be extended to other statistical tests using the same techniques we describe here.

The pseudo-code in this section computes the test statistics of Student’s t-test, Pearson Correlation and Chi-Squared, denoted by t , r and χ^2 , respectively. We note that using these statistics computing p-values can be done trivially for each respective

statistic using the *degrees of freedom* [23, 50] associated with the data and we therefore omit the full details of this process.

3.4.1 Dataset Characteristics

Let $\mathbb{D}_{N \times M}(\mathbb{R})$ be the set of all real-valued N -by- M matrices. Formally, a dataset $\mathcal{D} \in \mathbb{D}_{N \times M}(\mathbb{R})$ is a matrix with N rows and M columns (attributes). Recall that the encrypted form of \mathcal{D} is denoted by $\llbracket \mathcal{D} \rrbracket$ where each entry is encrypted. Then $\llbracket \mathcal{D} \rrbracket \in \mathbb{D}_{N \times M}(\mathbb{Z}_n^2)$ (note: n and N have nothing to do with each other. See § 2.7).

Therefore, \mathcal{D} can be seen as a $N \times M$ matrix containing real values and $\llbracket \mathcal{D} \rrbracket$ as a $N \times M$ matrix containing encrypted fixed-point approximations to the real values of each entry. In describing the tests, we assume, *WLOG*, that the tests are computed over the first M (where $M \geq 2$) attributes π_1, \dots, π_M of \mathcal{D} . As such, we denote the value of the i th row of attribute j as \mathcal{D}_{i,π_j} , equivalently denoted as $\llbracket \mathcal{D}_{i,\pi_j} \rrbracket$ in the encrypted dataset.

Recall, that in order to allow researchers to form hypotheses on \mathcal{D} , we require the data owner to release *attribute metadata* (e.g., number of attributes, their type and domain size, independence from other attributes, etc) and assume that the size of the dataset is known. Again, we define $\mathcal{L}_s : \mathbb{D}_{N \times M} \rightarrow \{0, 1\}^*$ to be a function from datasets to binary strings encoding attribute metadata. Given $\mathcal{L}_s(\mathcal{D})$ it should be possible to 1) define a hypothesis on \mathcal{D} and 2) perform a statistical test in CUSTODES. At minimum, we require that $\mathcal{L}_s(\mathcal{D})$ provides N , M , a set of independent attributes and a set of bounds on attribute domains.

We are now ready to describe the three statistical tests we implement in CUSTODES, Student's t-test, Pearson Correlation test, and the Chi-Squared test.

3.5 Student's t-test

Student's t-test is used to compare means of two independent samples where the null hypothesis stipulates that there is no statistically significant difference between the two distributions [64]. Student's t-test, for example, can be used to compare outcomes

of clinical trials to determine whether a significant difference is observed between the experiment and control group.

Let π_1 and π_2 be two independent attributes (columns) in \mathcal{D} . Let x and y represent the column values of π_1 and π_2 in \mathcal{D} , respectively. That is, x is $\mathcal{D}_{1,\pi_1}, \dots, \mathcal{D}_{N,\pi_1}$ and y is $\mathcal{D}_{1,\pi_2}, \dots, \mathcal{D}_{N,\pi_2}$. Denote the mean of x and y as \bar{x} and \bar{y} , respectively. Let the empirical standard deviations of the samples x and y be denoted as $\hat{\sigma}_x$ and $\hat{\sigma}_y$.

The t statistic is computed according to:

$$t = \frac{\bar{x} - \bar{y}}{\hat{\sigma}_p \sqrt{\frac{2}{N}}} \quad (3.1)$$

where $\hat{\sigma}_p = \sqrt{\frac{\hat{\sigma}_x^2 + \hat{\sigma}_y^2}{2}}$ is an estimator of the empirical pooled standard deviation of the two samples x and y [50].

Student's t-test in CUSTODES. Pseudo-code for the test is presented in Protocol 6. The protocol closely follows equation 3.1 while adjusting the precision of fixed-point computations using **TruncPR** as explained in § 2.3.1.

We briefly describe the steps of the protocol: Lines 2 to 4 compute the sample mean of the two samples x and y . Lines 10 to 12 compute the variance (square of the standard deviation) of the two variables. Line 13 computes the pooled variance of the sample. Line 21 computes the square of the denominator in equation 3.1. Line 17 extracts the sign of the test statistic. Line 21 computes the reciprocal square-root of the denominator. Line 23 reveals the test statistic. Finally, line 27 computes the p-value in the clear by using the revealed test statistic t and the degrees of freedom, which, in the case of the Student's t-test is one less than the number of rows [50].

Protocol 6: $(t, p) \leftarrow \text{StudentTTest}(\pi_1, \pi_2)$

Public Parameters: $\llbracket \mathcal{D} \rrbracket, N, M, \mathcal{B}, \text{pk}, v, t, \kappa, \ell, f$

Result: Test statistic and p-value (t, p) resulting from computing the Student's t-test over the selected attributes π_1, π_2 .

```

1   $\text{df} \leftarrow N - 1;$                                      // t-test degrees of freedom
2   $([x], [y]) \leftarrow (\sum_{i=1}^N \llbracket \mathcal{D}_{i, \pi_1} \rrbracket, \sum_{i=1}^N \llbracket \mathcal{D}_{i, \pi_2} \rrbracket);$ 
3   $[\bar{x}] \leftarrow \text{TruncPR}([x] \text{fp}_f(1/N), 2\ell, f);$       // compute the sample mean of  $x$ 
4   $[\bar{y}] \leftarrow \text{TruncPR}([y] \text{fp}_f(1/N), 2\ell, f);$       // compute the sample mean of  $y$ 
5  foreach  $i \leftarrow 1, 2, \dots, N$  do parallel
6  |    $[d_{x_i}] \leftarrow \llbracket \mathcal{D}_{i, \pi_1} \rrbracket - [\bar{x}];$ 
7  |    $[d_{y_i}] \leftarrow \llbracket \mathcal{D}_{i, \pi_2} \rrbracket - [\bar{y}];$ 
8  |    $[h_{x_i}] \leftarrow \text{Mult}([d_{x_i}], [d_{x_i}]);$ 
9  |    $[h_{y_i}] \leftarrow \text{Mult}([d_{y_i}], [d_{y_i}]);$ 
10  $([h_x], [h_y]) \leftarrow (\sum_{i=1}^N [h_{x_i}], \sum_{i=1}^N [h_{y_i}]);$ 
11  $[\hat{\sigma}_x^2] \leftarrow \text{TruncPR}([h_x], 2\ell, f);$           // compute the sample variance of  $x$ 
12  $[\hat{\sigma}_y^2] \leftarrow \text{TruncPR}([h_y], 2\ell, f);$           // compute the sample variance of  $y$ 
13  $[u] \leftarrow [\hat{\sigma}_x] + [\hat{\sigma}_y];$ 
14  $[u'] \leftarrow [u] \text{fp}_f(1/(N^2 - N));$ 
15  $[v] \leftarrow \text{TruncPR}([u'], 2\ell, f);$ 
16  $[w] \leftarrow [\bar{x}] - [\bar{y}];$ 
17  $[s] \leftarrow \text{SignBit}([w]);$                           // extract the sign of the test statistic
18  $s \leftarrow \text{Reveal}([s])$ 
19 if  $s = 1$  then
20 |    $[w] \leftarrow [w](-1);$                              // make the numerator positive
21  $[q] \leftarrow \text{FPSqrtRcpr}([v]);$                         // compute the denominator in equation 3.1
22  $[t] \leftarrow \text{Mult}([w], [q]);$ 
23  $\tilde{t} \leftarrow \text{Reveal}([t]);$ 
24  $t \leftarrow \tilde{t}/2^{2f};$                                 // obtain the real value approximation
25 if  $s = 1$  then
26 |    $t \leftarrow -t;$                                     // restore the sign of the test statistic
27  $p \leftarrow \text{computePValue}(t, \text{df});$ 
28 return  $(t, p);$ 

```

Requirements. Let $\eta \in \mathbb{N}$ be an upper bound on the largest absolute value in \mathcal{D} (i.e., given as part of attributes' domain size), to evaluate a Student's t-test over any two attributes \mathcal{D} , the following constraints must hold to ensure no “overflow” occurs during computation: $\ell > \log_2(N) + 2\log_2(\eta) + f$ and $n > 2^{\ell+\kappa+v+1}$. The calculation is trivial to verify by examining the arithmetic circuit.

Complexity. All sub-protocols invoked are constant-rounds with the exception of `FPSqrtRcpr` which requires a number of rounds on the order of $\log_2(\ell)$. Therefore, the total round complexity hinges on the `FPSqrtRcpr` invocation making the final complexity $O(\log_2(\ell))$ rounds and $O(2N)$ invocations of constant-round sub-protocols.

3.6 Pearson's Correlation Test

Pearson Correlation test is used to compare the linear correlation between two continuous independent variables and is frequently used to evaluate the effect that two variables have on each other. The test statistic, r , lies in the range $[-1, 1]$ where $r = 1$ corresponds to a positive correlation level between the variables and $r = -1$ corresponds to a negative correlation [64].

Let π_1 and π_2 be two continuous attributes and let variables x and y represent the values of π_1 and π_2 , respectively. We let x_1, \dots, x_N and y_1, \dots, y_N denote the observed values for x and y and denote the empirical mean of x and y by \bar{x} and \bar{y} . Pearson Correlation's correlation coefficient r is computed according to the following equation [50]:

$$r = \frac{\sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^N (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^N (y_i - \bar{y})^2}} \quad (3.2)$$

Pearson Correlation in CUSTODES. Pseudo-code for computing Pearson Correlation coefficient is presented in Protocol 7. The protocol closely follows equation 3.2 to compute the statistic and uses `TruncPR` to adjust the precision of fixed-point values as explained in § 2.4. Lines 2 to 4 compute the mean of the two samples. Lines 11 to 13 compute the sum of the variance of two samples. Line 14 computes the covariance

of the two samples (the numerator of equation 3.2). Line 21 computes the square of the denominator in equation 3.2. Line 16 extracts the sign of the test statistic. Line 21 computes the reciprocal square-root of the denominator. Line 23 reveals the test statistic. Finally, line 27 computes the p-value in the clear using the value of the test statistic and the degrees of freedom, which, in the case of the Pearson Correlation test is one less than the number of rows [50].

Protocol 7: $(t, p) \leftarrow \text{PearsonTest}(\pi_1, \pi_2)$

Public Parameters: $\llbracket \mathcal{D} \rrbracket, N, M, \mathcal{B}, \text{pk}, v, t, \kappa, \ell, f$

Result: Test statistic and p-value (t, p) resulting from computing the Pearson Correlation test over the selected attributes π_1, π_2 .

```

1   $\text{df} \leftarrow N - 2;$                                 // correlation test degrees of freedom
2   $([x], [y]) \leftarrow (\sum_{i=1}^N \llbracket \mathcal{D}_{i, \pi_1} \rrbracket, \sum_{i=1}^N \llbracket \mathcal{D}_{i, \pi_2} \rrbracket);$ 
3   $[\bar{x}] \leftarrow \text{TruncPR}([x] \text{fp}_f(1/N), 2\ell, f);$     // compute the sample mean of  $x$ 
4   $[\bar{y}] \leftarrow \text{TruncPR}([y] \text{fp}_f(1/N), 2\ell, f);$     // compute the sample mean of  $y$ 
5  foreach  $i \leftarrow 1, 2, \dots, N$  do parallel
6  |    $[d_{x_i}] \leftarrow \llbracket \mathcal{D}_{i, \pi_1} \rrbracket - [\bar{x}];$ 
7  |    $[d_{y_i}] \leftarrow \llbracket \mathcal{D}_{i, \pi_2} \rrbracket - [\bar{y}];$ 
8  |    $[h_{x_i}] \leftarrow \text{Mult}([d_{x_i}], [d_{x_i}]);$ 
9  |    $[h_{y_i}] \leftarrow \text{Mult}([d_{y_i}], [d_{y_i}]);$ 
10 |    $[q_i] \leftarrow \text{Mult}([d_{x_i}], [d_{y_i}]);$ 
11  $([h_x], [h_y]) \leftarrow (\sum_{i=1}^N [h_{x_i}], \sum_{i=1}^N [h_{y_i}]);$ 
12  $[\hat{\sigma}_x^2] \leftarrow \text{TruncPR}([h_x], 2\ell, f);$     // compute the sample variance of  $x$ 
13  $[\hat{\sigma}_y^2] \leftarrow \text{TruncPR}([h_y], 2\ell, f);$     // compute the sample variance of  $y$ 
14  $[q] \leftarrow \sum_{i=1}^N [q_i];$     // compute the sample covariance of  $x$  and  $y$ 
15  $[w] \leftarrow \text{TruncPR}([q], 2\ell, f);$ 
16  $[s] \leftarrow \text{SignBit}([w]);$     // extract the sign of the test statistic
17  $s \leftarrow \text{Reveal}([s])$ 
18 if  $s = 1$  then
19 |    $[w] \leftarrow [w](-1);$     // make the numerator positive
20  $[v] \leftarrow \text{Mult}([\hat{\sigma}_x^2], [\hat{\sigma}_y^2]);$ 
21  $[x] \leftarrow \text{FPSqrtRcpr}([v]);$     // compute the denominator in equation 3.1
22  $[r] \leftarrow \text{Mult}([w], [x]);$ 
23  $r \leftarrow \text{Reveal}([r]);$ 
24  $r \leftarrow r/2^{2f};$     // obtain real value approximation
25 if  $s = 1$  then
26 |    $r \leftarrow -r;$     // restore the sign of the test statistic
27  $p \leftarrow \text{computePValue}(r, \text{df});$ 
28 return  $(r, p);$ 

```

Requirements. Let $\eta \in \mathbb{N}$ be an upper bound to the largest absolute value in \mathcal{D} , to evaluate a Pearson Correlation test over any two attributes in \mathcal{D} , the following requirements must hold for the user-set parameters to ensure no “overflow” occurs during computation: $\ell > 2\log_2(\eta) + \log_2(N) + f$ and $n > 2^{\ell+\kappa+v+1}$. Again, the calculation follows from the description of the circuit in the protocol.

Complexity. All sub-protocols invoked are constant-rounds with the exception of `SqrtRcpr` which requires a number of rounds proportional to $\log_2(\ell)$. We therefore conclude that the protocol requires $O(\log_2(\ell))$ rounds and 17 invocations.

3.7 Chi-Squared Test

The Chi-Squared test determines whether the sampling distribution of the test statistic follows a χ^2 distribution when the null hypothesis is true [13, 64]. The Chi-Squared test is useful in determining whether there is a significant difference between the expected frequencies and the observed frequencies in a set of observations from *mutually exclusive* categories. In other words, Chi-Squared evaluates the “goodness of fit” between a set of expected values and observed values, the test result is deemed significant if the expected frequencies are different from the observed frequencies.

For a collection of N observations classified into M mutually exclusive categories where each observed value is denoted by x_i for $i = 1, 2, \dots, M$, denote the expected probability that a value falls into the i th category by α_i such that $\sum_{i=1}^M \alpha_i = 1$. Note that the expected value for each category is $E_i = n\alpha_i$. The Chi-Squared statistic is computed according to the following equation:

$$\chi^2 = \sum_{i=1}^M \frac{(x_i - E_i)^2}{E_i} \quad (3.3)$$

Let \mathcal{D} contain M mutually exclusive attributes (categories) π_1, \dots, π_M such that $\mathcal{D}_i, \pi_j \in \{0, 1\}$ for all $i = 1 \dots N$ and $j = 1 \dots M$. In other words, each category is a Boolean flag representing whether a row i in \mathcal{D} is in the category j . Given such a “raw” dataset \mathcal{D} , we need a way to convert \mathcal{D} into histogram form $\mathcal{H} = (h_1, h_2, \dots, h_M)$,

filtered based on the selected categories, so as to compute the Chi-Squared statistic over \mathcal{H} . To achieve this in a private and secure manner, we must first “pre-process” $\llbracket \mathcal{D} \rrbracket$ into a histogram \mathcal{H} containing the summations of the M categories selected by the user.

Remark 3.7.1. Note that α_i may or may not be known by the researcher. In some applications of the test, the expected probability of a given category given for the null hypothesis may be computed from the dataset and may therefore be deemed private. In the case that α_i is known (e.g., it may simply be N/M in the case of an expected uniform distribution) then the evaluation of the test does not require evaluating interactive gates. However, in the case that each α_i is computed from the data, then a linear (in the number of evaluated categories) number of division evaluations are necessary. Protocol 8 presents the latter variant for the sake of generality.

Remark 3.7.2. We use \mathcal{H} for the purpose of providing a general solution and to remain consistent with the descriptions of the previous two tests (i.e., \mathcal{D} has the same format across all tests with the exception of being categorical). If \mathcal{D} is already in histogram form, then the Chi-Squared test may be applied directly on \mathcal{D} . Furthermore, we note that it is possible to compute a histogram from an arbitrary dataset using the basic computational building blocks presented in § 2.4, though it could potentially require $O(N)$ multiplications in order to perform a **SELECT** operation over attributes in $\llbracket \mathcal{D} \rrbracket$.

Chi-Squared in CUSTODES. Pseudo-code for computing the Chi-Squared test is presented in Protocol 8 and closely follows equation 4.3.3. The first for-loop (line 2) computes the histogram $\llbracket \mathcal{H} \rrbracket = ([h_1], \dots, [h_M])$ from $\llbracket \mathcal{D} \rrbracket$. The correctness of the computed histogram \mathcal{H} follows from the fact that each attribute in the set is mutually exclusive, i.e., if $\mathcal{D}_{j,\pi_i} = 1$ then $\mathcal{D}_{j,\pi_M} = 0$ for all $j = 1 \dots N$ and $i \neq M$. Line 4 computes the sample total. The second for-loop (line 5) computes the *expected values* of each entry in the $\llbracket \mathcal{H} \rrbracket$ as well as the $(x_i - E_i)$ term of equation 4.3.3. The third for-loop (line 8) computes each term in the summation. Finally, line 12 computes the Chi-Squared statistic by summing over all the individual terms.

Protocol 8: $(t, p) \leftarrow \text{ChiSq}(\{\pi_1, \dots, \pi_w\}, \{\alpha_1, \dots, \alpha_w\})$

Public Parameters: $\llbracket \mathcal{D} \rrbracket, N, M, \mathcal{B}, \text{pk}, v, t, \kappa, \ell, f$

Result: Test statistic and p-value (t, p) of computing the Chi-Squared test over the selected attributes $\{\pi_1, \dots, \pi_w\}$ and expectations $\{\alpha_1, \dots, \alpha_w\}$.

```

1  $\text{df} \leftarrow w - 1;$                                      //  $\chi^2$  test degrees of freedom
2 foreach  $i \leftarrow 1, 2, \dots, w$  do parallel
3    $[h_i] \leftarrow \sum_{j=1}^N \llbracket \mathcal{D}_{j, \pi_i} \rrbracket;$            // count observations in each category
4  $[s] \leftarrow \sum_{i=1}^w [h_i];$  // compute total number of observations for selection
5 foreach  $i \leftarrow 1, 2, \dots, w$  do parallel
6    $[e_i] \leftarrow [s] \alpha_i;$                          // compute the expected observations
7    $[d_i] \leftarrow [h_i] - [e_i];$ 
8 foreach  $i \leftarrow 1, 2, \dots, w$  do parallel
9    $[w_i] \leftarrow \text{Mult}([d_i], [d_i]);$ 
10   $[w_i] \leftarrow \text{TruncPR}([w_i], 2\ell, f);$ 
11   $[u_i] \leftarrow \text{FPDiv}([w_i], [e_i])$ 
12  $[q] \leftarrow \sum_{i=1}^w [u_i];$ 
13  $q \leftarrow \text{Reveal}([q]);$ 
14  $\chi^2 \leftarrow q/2^f;$                                    // obtain real value approximation
15  $p \leftarrow \text{computePValue}(\chi^2, \text{df});$ 
16 return  $(\chi^2, p)$ 

```

Requirements. Let $\eta \in \mathbb{N}$ be an upper bound to the largest absolute value in \mathcal{D} . Let \mathcal{H} be a histogram with w attributes ($2 \leq w \leq M$). To correctly perform the Chi-Squared test over the w selected attributes in \mathcal{D} , the following requirements must hold for these user-set parameters to ensure no “overflow” occurs during computation: $\ell > \log_2(\eta) + 2 \log_2(M) + f$ and $n > 2^{\ell + \kappa + v + 1}$. The requirement follows from the test circuit description.

Complexity. All sub-protocols invoked are constant-rounds with the exception of FPDiv which requires a number of rounds proportional to $\log_2(\ell)$. We therefore conclude that the protocol requires $O(M \log_2(\ell))$ rounds and $3M + 1$ invocations.

Chapter 4

Providing Differential Privacy

4.1 Motivation

A desirable extension to CUSTODES is the ability to evaluate hypotheses in a *differentially private* manner. Roughly speaking, differential privacy [28] is a definition for ensuring the privacy of any given *individual* in the database. Intuitively, differential privacy guarantees that the presence or absence of any individual’s contribution to the dataset does not affect the output of the computation (from the point of view of an adversary seeing the computation results). By making the *outputs*¹ of the statistical tests differentially private, sensitive datasets (e.g., medical records) can be analyzed through CUSTODES without fear of leaking personal, classified or otherwise secret information while still gaining insights on the data. As there is no such thing as a free lunch, ensuring differential privacy may significantly reduce the utility of the output for some class of functions with high variability in the output evaluated on differing inputs. However, as we shall see, under certain reasonable assumptions we may achieve differentially private outputs for statistical tests while still preserving utility and correctness of the FDR control procedure.

The reader is advised to view the techniques presented in this chapter as a modular extension to the main CUSTODES construction. The usefulness of differentially private outputs is solely based on the context in which the statistical tests are performed

¹Not to be confused with dataset perturbation.

and, as we shall see, depends heavily on attributes of the dataset. It is therefore an extension which should only be applied in situations where the trade-offs in privacy and utility are carefully considered and deemed reasonable. We note, however, that this is not a requirement unique to CUSTODES as many applications of differential privacy require careful trade-off considerations [25].

4.2 Definitions

Formally, if \mathcal{D}_1 and \mathcal{D}_2 are datasets differing in at most one entry, then \mathcal{D}_1 and \mathcal{D}_2 are said to be *adjacent* or *neighboring*. Let \mathcal{M} be a randomized mechanism (algorithm) which takes a dataset \mathcal{D} as input and produces a randomized response as output. We denote the range of \mathcal{M} by $\text{Range}(\mathcal{M})$.

Definition 4.2.1 (Differential Privacy [28]). A randomized mechanism \mathcal{M} is said to be (ε, δ) -differentially private for non-negative ε, δ if for all adjacent datasets $\mathcal{D}_1, \mathcal{D}_2$ and all outputs $S \subset \text{range}(\mathcal{M})$

$$\Pr[\mathcal{M}(\mathcal{D}_1) \in S] \leq e^\varepsilon \Pr[\mathcal{M}(\mathcal{D}_2) \in S] + \delta$$

where the probability is over the randomness of the mechanism \mathcal{M} . When $\delta = 0$, the mechanism is said to achieve *pure* differential privacy.

Definition 4.2.2 (Global Sensitivity [28]). The Global Sensitivity (GS) for any function $F : \mathbb{D} \rightarrow \mathbb{R}$ is defined to be

$$\text{GS}_F \triangleq \max_{\mathcal{D}_1, \mathcal{D}_2 \in \mathbb{D}} |F(\mathcal{D}_1) - F(\mathcal{D}_2)|$$

Where \mathcal{D}_1 and \mathcal{D}_2 are neighboring datasets, i.e., differ in at most one entry. Note that GS depends only on the function itself and *not* the dataset (or contents thereof). Specifically, GS must hold for all possible datasets in the domain and not just a specific instance.

4.2.1 Design of Differentially Private Mechanisms

Two common mechanisms used in practice to satisfy differential privacy are the Laplace and Gaussian mechanisms which perturb a sensitive output using a random sample from the Laplace and Gaussian distributions, respectively, where the parameters of the distributions are set based on the sensitivity of the evaluated function [25]. Again, it is important to remember that a differentially private mechanism works independently of the data over which the function is evaluated and only takes into consideration the sensitivity of the function producing the sensitive output. For the sake of brevity, we provide an informal overview of the Laplace mechanism but omit the Gaussian mechanism which follows a similar vein.

Definition 4.2.3 (Laplace Mechanism [28]). Let randomized mechanism \mathcal{M}_F output $F(\mathcal{D}) + y$ where $y \sim \mathcal{L}(\lambda)$ is a random variable sampled independently from the Laplace distribution with variance $\sqrt{2}\lambda$ [28]. Formally,

$$\mathcal{M}_F(\mathcal{D}) = F(\mathcal{D}) + y$$

Theorem 4.2.1. *The Laplace Mechanism satisfies $(\varepsilon, 0)$ -differential privacy.*

Proof. We refer the reader to [28] for the proof of the theorem. □

4.2.2 Impelentation Challenges

Several challenges arise when attempting to make the outputs of statistical tests differentially private. The first challenge is deriving the sensitivities of the test statistics. The second challenge is securely generating random samples from the Laplace distribution in a distributed manner, since none of the computing parties (or researchers) can know the noise term added to the final result. Finally, the third subtlety which needs to be dealt with is ensuring that the added noise does not inhibit the FDR control procedure set in place. Specifically, we must guarantee that adding noise does not lead to false-rejections (e.g., by making the p-value smaller; more significant). Fortunately, the latter requirement is satisfied immediately by the definition

of α -investing FDR control procedure which bounds the *expected* number of false discoveries. Since the noise added to the results is sampled from the Laplace distribution, centered at zero, the expected value of the noise added is likewise *zero* due to symmetry. Therefore, the *overall* number of expected false discoveries remains unaffected by the differentially private mechanism. We leave it as future work to analyze the interplay between differential privacy, statistical tests, and FDR control procedures but note that a similar line of work was recently initiated in [29, 26, 27].

4.3 Sensitivity of Statistical Tests

A crucial component required by the differentially private mechanism the global sensitivity of the evaluated function (i.e., the statistical test). This is not always trivial to achieve in practice with complex function such as statistical tests and, to our knowledge, has not been attempted prior to this work (with the exception of the Chi-Squared test [34]). In what follows, we require several assumptions be placed on the data in order to effectively upper bound the sensitivity of each statistical test. Furthermore, we omit discussion of how the parameter ε should be set for each of these tests. As is generally the case with differentially private mechanisms, the value for ε is ultimately a social question and application specific [28].

4.3.1 Sensitivity of Student's t-test

Let samples $x = (x_1, \dots, x_N)$, $y = (y_1, \dots, y_N)$ contain values in the range $[\alpha, \beta]$ and have equal size $|x| = |y| = N$. To analyze the sensitivity of the Student's t-test statistic, we maximize the difference in test statistics $|t - t'|$, where t and t' are the Student's t-test statistics computed over the samples *differing in at most one value*. Hence, to upper bound the sensitivity, we must, WLOG, *maximize* t and *minimize* t' . Recall that the t -statistic is computed according to the following equation (see § 3.4):

$$t = \frac{\bar{x} - \bar{y}}{\hat{\sigma}_p \sqrt{\frac{2}{N}}}$$

Let the samples x, y (resp. x', y') correspond to the samples used in computing the test statistic t (resp. t') such that x, x' (resp. y, y') differ in at most a single element. It suffices to maximize the sample mean \bar{x} (resp. minimize \bar{x}') by changing only a single element in the sample x and likewise minimize \bar{y} (resp. maximize \bar{y}') with the constraint that y, y' differ in at most one element. We note that by the definition of global sensitivity, it would suffice to change either the sample x or y (not necessarily both) but for ease of analysis, we derive an upper bound to the global sensitivity by assuming that both x and y differ in a single entry.

Theorem 4.3.1. *Assuming samples x, x' (resp. y, y') contain values in the range $[\alpha, \beta]$ such that they differ in at most one value, and assuming the empirical pooled standard deviations of the differing samples are such that $\hat{\sigma}_p = \hat{\sigma}'_p \geq \gamma$ for some lower bound γ , the sensitivity of the Student's t -test is bounded by,*

$$|t - t'| \leq \frac{|\beta - \alpha|}{\gamma \sqrt{\frac{N}{2}}}$$

Proof.

$$\begin{aligned} |t - t'| &= \left| \frac{\bar{x} - \bar{y}}{\hat{\sigma}_{p1} \sqrt{\frac{2}{N}}} - \frac{\bar{x}' - \bar{y}'}{\hat{\sigma}_{p2} \sqrt{\frac{2}{N}}} \right| = \left| \frac{\bar{x} - \bar{y}}{\sqrt{\frac{\hat{\sigma}_x^2 + \hat{\sigma}_y^2}{2}} \sqrt{\frac{2}{N}}} - \frac{\bar{x}' - \bar{y}'}{\sqrt{\frac{\hat{\sigma}_{x'}^2 + \hat{\sigma}_{y'}^2}{2}} \sqrt{\frac{2}{N}}} \right| \\ &= \sqrt{N} \left| \frac{\bar{x} - \bar{y}}{\sqrt{\hat{\sigma}_x^2 + \hat{\sigma}_y^2}} - \frac{\bar{x}' - \bar{y}'}{\sqrt{\hat{\sigma}_{x'}^2 + \hat{\sigma}_{y'}^2}} \right| \\ &\leq \sqrt{N} \left| \frac{\bar{x} - \bar{y}}{\sqrt{\hat{\sigma}_x^2 + \hat{\sigma}_y^2}} - \frac{\bar{x} - \frac{\beta - \alpha}{N} - \bar{y} - \frac{\beta - \alpha}{N}}{\sqrt{\hat{\sigma}_x^2 + \hat{\sigma}_y^2}} \right| \\ &\leq \sqrt{N} \left| \frac{\bar{x} - \bar{y} - \bar{x} + \bar{y} + 2\left(\frac{\beta - \alpha}{N}\right)}{\sqrt{2\gamma^2}} \right| \end{aligned}$$

$$= \frac{|2(\beta - \alpha)|}{\sqrt{N}\sqrt{2}\gamma} = \frac{|\beta - \alpha|}{\gamma\sqrt{\frac{N}{2}}}$$

The first inequality holds on the assumption that $\hat{\sigma}_x^2 = \hat{\sigma}_y^2$ and the second inequality holds on the assumption that $\hat{\sigma}_x = \hat{\sigma}_y \geq \gamma$. \square

Remark 4.3.1 (Standard Deviation Equality). In computing the sensitivities, we make the assumption that the empirical standard deviation of the two samples differing in at most one value are equal. While at first glance this may seem as a strong assumption, given that the Student's t-test assumes the variables in the sample follow a normal distribution this is a reasonable assumption to make in practice [50]. Furthermore, by the central limit theorem, the resulting samples will follow a normal distribution in the limit (regardless of their original distribution), and as such, changing two of the variables to either extreme (i.e., setting the one of the variables to α and the other to β) changes the variances by an equal amount due to symmetry.

4.3.2 Sensitivity of Pearson Correlation

Let samples $x = (x_1, \dots, x_N)$, $y = (y_1, \dots, y_N)$ contain values in the range $[\alpha, \beta]$ and have equal size $|x| = |y| = N$. To maximize $|r - r'|$, where r and r' are the Pearson Correlation correlation statistics computed over samples *differing in at most one value*, we must, WLOG, *maximize* r and *minimize* r' . Recall that the Pearson Correlation statistic is computed according to the following equation (see § 3.4):

$$r = \frac{\sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^N (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^N (y_i - \bar{y})^2}}$$

and can be re-written as,

$$r = \frac{\sum_{i=1}^N x_i y_i - N \bar{x} \bar{y}}{(N-1) \hat{\sigma}_x \hat{\sigma}_y}$$

for the purpose of simplifying the subsequent analysis, where $\hat{\sigma}_x, \hat{\sigma}_y$ are the empirical standard deviations of the two samples, respectively.

Let the samples x, y (resp. x', y') correspond to the samples used in computing

the Pearson Correlation test statistic r (resp. r') such that they differ in at most a single entry. Again, for ease of analysis, we upper bound the global sensitivity by having both samples x, x' and y, y' differ in one value each.

Theorem 4.3.2. *Assuming samples x, x' (resp. y, y') contain values in the range $[\alpha, \beta]$ such that they differ in at most one value, and the empirical standard deviations are such that $\hat{\sigma}_x = \hat{\sigma}_y \geq \gamma$, the sensitivity of the Pearson Correlation test is given by,*

$$|r - r'| \leq \frac{|\beta^2 - \alpha^2|}{N\gamma^2}$$

Proof.

$$\begin{aligned} |r - r'| &= \left| \frac{\sum_{i=1}^N x_i y_i - N\bar{x}\bar{y}}{(N-1)\hat{\sigma}_x \hat{\sigma}_y} - \frac{\sum_{i=1}^N x'_i y'_i - N\bar{x}'\bar{y}'}{(N-1)\hat{\sigma}_{x'} \hat{\sigma}_{y'}} \right| \\ &= \left| \frac{\sum_{i=1}^{N-1} x_i y_i + \beta^2 - N\bar{x}\bar{y} - \sum_{i=1}^{N-1} x_i y_i - \alpha^2 + N\bar{x}'\bar{y}'}{(N-1)\hat{\sigma}_x \hat{\sigma}_y} \right| \\ &= \left| \frac{\beta^2 - \alpha^2 - N(\bar{x}\bar{y} - \bar{x}'\bar{y}')}{(N-1)\hat{\sigma}_x \hat{\sigma}_y} \right| \\ &= \left| \frac{\beta^2 - \alpha^2 - N(\frac{\beta^2 - \alpha^2}{N^2})}{(N-1)\hat{\sigma}_x \hat{\sigma}_y} \right| \\ &\leq \left| \frac{\beta^2 - \alpha^2}{(N-1)\gamma^2} - \frac{\beta^2 - \alpha^2}{N(N-1)\gamma^2} \right| \leq \frac{|\beta^2 - \alpha^2|}{N\gamma^2} \end{aligned}$$

The first inequality holds on the assumption that $\hat{\sigma}_x = \hat{\sigma}_y \geq \gamma$. The second inequality follows from the factor of N^2 in the denominator of the term being subtracted.

□

4.3.3 Sensitivity of Chi-Squared

Given a histogram consisting of N observations across M categories, we wish to measure the sensitivity of the χ^2 statistic on samples differing in at most one element which, in the case of the Chi-Squared test consists of a single observation in one category shifting to an observation in a different category (e.g., an observation in the i th category moves to an observation in the k th category). Recall that the Chi-Squared statistic is computed according to,

$$\chi^2 = \sum_{i=1}^M \frac{(x_i - E_i)^2}{E_i}$$

Hence, we can bound the sensitivity by examining the change in the Chi-Squared statistic when a single observation is swapped from one category to another (i.e., moves from x_i to x_j for some $i, j \leq M$). Furthermore, the expected number of observations, E_i , can either be a function of N or a function of N and M and denotes the number of *expected* observations in each category. Let the expected observations per category follow the probability distribution $(\eta_1, \eta_2, \dots, \eta_M)$ and let $\eta_{\min} = \min(\eta_1, \eta_2, \dots, \eta_M)$. Then, each $E_i = N\eta_i$ (note: this is simply a reformulation of the definition of the Chi-Squared test [50]).

Theorem 4.3.3. *Given samples x, x' of size $N \geq 2$ differing in at most one observation and assuming the expected observations across categories follow a distribution $(\eta_1, \eta_2, \dots, \eta_M)$ proportional to the sample size N , where $\eta_{\min} = \min(\eta_1, \eta_2, \dots, \eta_M)$, the sensitivity of Chi-Squared test for N observations in M categories is bounded by,*

$$|\chi^2 - \chi^{2'}| \leq \frac{2}{\eta_{\min}},$$

Proof. WLOG let the histogram differ in the last two categories (observation switches from category M to category $M - 1$). Then, the difference in the empirical χ^2 test

statistics is given by,

$$\begin{aligned}
|\chi^2 - \chi^{2'}| &= \left| \sum_{i=1}^M \frac{(x_i - E_i)^2}{E_i} - \sum_{i=1}^{M-2} \frac{(x_i - E_i)^2}{E_i} - \frac{(x_{M-1} + 1 - E_{M-1})^2}{E_{M-1}} - \frac{(x_M - 1 - E_M)^2}{E_M} \right| \\
&= \left| \frac{(x_{M-1} - E_{M-1})^2}{E_{M-1}} + \frac{(x_M - E_M)^2}{E_M} - \frac{(x_{M-1} + 1 - E_{M-1})^2}{E_{M-1}} - \frac{(x_M - 1 - E_M)^2}{E_M} \right| \\
&= \left| \frac{(x_{M-1} - E_{M-1})^2 - (x_{M-1} + 1 - E_{M-1})^2}{E_{M-1}} + \frac{(x_M - E_M)^2 - (x_M - 1 - E_M)^2}{E_M} \right| \\
&= \left| \frac{-2x_{M-1} + 2E_{M-1} - 1}{E_{M-1}} + \frac{2x_M - 2E_M - 1}{E_M} \right| \\
&= \left| \frac{-2x_{M-1} - 1}{E_{M-1}} + \frac{2x_M - 1}{E_M} \right| \\
&\leq \left| \frac{2(N-1) - 1}{E_N} - \frac{2+1}{E_{N-1}} \right| \quad (\text{by letting } x_M = N-1 \text{ and } x_{M-1} = 1) \\
&\leq \left| \frac{2N-3}{N\eta_{min}} \right| \leq \frac{2}{\eta_{min}}
\end{aligned}$$

The first equality is simply the difference in the Chi-Squared equations where one observation in category x_M is changed to category x_{M-1} . The second and third inequalities hold on the assumption that $E_N = N\eta_{min}$ (in the worst case) where we have that $\eta_{min} \in (0, 1]$.

□

4.4 Secure Noise Generation

To achieve differential privacy in CUSTODES, it is necessary to generate a random sample from the Laplace distribution without the parties learning the resulting value

of the sample. To achieve this, we exploit the infinite divisibility of the Laplace distribution [52, 38] which makes it possible to generate a random sample from the Laplace distribution using four samples from the Gaussian distribution as follows,

$$\mathcal{L}(\mu, \lambda) = \mathcal{N}(\mu, \hat{\sigma})^2 + \mathcal{N}(\mu, \hat{\sigma})^2 + \mathcal{N}(\mu, \hat{\sigma})^2 + \mathcal{N}(\mu, \hat{\sigma})^2 \quad (4.1)$$

where $\hat{\sigma} = \lambda/2$.

Since the Gaussian distribution is stable [38], random samples of the Gaussian distribution can be generated in a distributed fashion,

$$\mathcal{N}(\mu, \hat{\sigma}) = \sum_{i=1}^v \mathcal{N}(\mu, \hat{\sigma}/v) \quad (4.2)$$

Protocols 9 and 10 are used in conjunction to generate noise for the Laplace mechanism in a secure manner.

Protocol 9: $[g] \leftarrow \text{SampleNormal}(\mu, \hat{\sigma})$

Public Parameters: $\llbracket \mathcal{D} \rrbracket, N, M, \mathcal{B}, \text{pk}, v, t, \kappa, \ell, f$

Result: Encrypted i.i.d. value sampled from the normal distribution.

```

1 foreach  $i \leftarrow 1, 2, \dots, v$  do parallel
2   |  $[g_i] \leftarrow \text{RandNormal}_{\mathcal{P}_i}(\mu, \hat{\sigma}/v)$  ; //  $\mathcal{P}_i$  samples  $\mathcal{N}(\mu, \hat{\sigma}/v)$ 
3  $[g] \leftarrow \sum_{i=1}^v [g_i]$ ;
4 return  $[g]$ ;

```

Correctness. Correctness follows from the fact that the normal distribution is infinitely divisible and that the sum of random normal variables is normal [50]. Each party generates a sample from the normal distribution with standard deviation $\hat{\sigma}/v$ making the sum have standard deviation $\hat{\sigma}$ as required.

Security. Security follows from the fact that each input chosen by the party is sampled independently and at random from the other inputs making the sum a random sample from the normal distribution unknown to the other parties. It is important to note, however, that while the variable is random, it follows a normal distribution

which is not uniform.

Complexity. The protocol has total complexity of one round and no invocations.

Protocol 10: $[c] \leftarrow \text{SampleLaplace}(\lambda)$

Public Parameters: $\llbracket \mathcal{D} \rrbracket, N, M, \mathcal{B}, \text{pk}, v, t, \kappa, \ell, f$

Result: Encrypted i.i.d. value sampled from the Laplace distribution.

```

1 foreach  $i \leftarrow 1, 2, \dots, 4$  do parallel
2    $[a_i] \leftarrow \text{SampleNormal}(\mu, \beta/2);$ 
3    $[b_i] \leftarrow \text{Mult}([a_i], [a_i]);$ 
4  $[b] \leftarrow [b_1] + [b_2] + [b_3] + [b_4];$ 
5  $[c] \leftarrow \text{TruncPR}([b], 2\ell, f);$ 
6 return  $[c];$ 

```

Correctness. Correctness follows from Equation 4.1. The protocol generates four random samples from the normal distribution which are then squared and summed to generate a random sample from the Laplace distribution.

Security. Security follows from the use of secure sub-protocols and the fact that no inputs are revealed.

Complexity. The protocol has total complexity of 2 rounds and 9 invocations.

4.5 Discussion

We can use the derived sensitivities and protocols for securely generating random samples from the Laplace distribution to make the revealed tests statistics described in § 3.4 differentially private. The obvious question, however, is whether the injected noise perturbs the test statistic beyond any utility. Consider, for example, the r -statistic computed using the Pearson Correlation test. Assuming the sample size is 100 and the range of values $|\beta^2 - \alpha^2| = 100$ (e.g., $[\alpha, \beta] = [0, 10]$) then, even for a relatively high epsilon (e.g., $\epsilon \geq \ln 3$), the noisy statistic can deviate from the true value by as much as ± 0.3 (with high probability) which, though seemingly small, may

drastically affect the perceived correlation in the sample. This example illustrates why applying the methods discussed in this chapter to CUSTODES must be done on a per-dataset bases where the added noise is deemed sufficiently small (in the worst case) so as to not hinder the utility of the test. We note, however, that the issue of utility is a universal question when it comes to differential privacy, one that is not confined to the practicality of CUSTODES.

We envision these techniques being effective when the dataset is relatively large (e.g., $N > 1000$) and the range of values contained within is small. As mentioned at the beginning of this chapter, it is also important to consider how the perturbed statistics will affect the FDR control procedure presented in § 1.3. While the expectation of the injected noise is zero, the expected number of false discoveries equals the expected number of false discoveries without injected noise (by linearity of expectation) and therefore does not affect the number of false discoveries in the average case.

4.6 Future Work (Related to this Chapter)

Future work should explore whether tighter bounds on the sensitivity of test statistics can be derived for these specific statistical test. Furthermore, there are interesting questions (many of which are outside the scope of this work) surrounding the interplay between differential privacy and FDR control procedures. Recent work by Dwork *et al.* [29] describes a method for achieving differentially private false discovery rate control using the Benjamini-Hochberg FDR control procedure [4] and proves sensitivity bounds on the p-values (rather than the test statistics). Furthermore, in another work by Dwork *et al.* [27], the authors demonstrate how differentially private outputs can be harnessed to prevent over-fitting in machine learning models, a similar problem to the one of false discovery control. While [27], unlike CUSTODES, the proposed system requires a central (and trusted) authority, future work could explore ways in which techniques from this chapter, and the CUSTODES framework in general, can be combined do distribute the trust to a set of computing parties

(e.g., universities). Other directions for future work include exploring the relationship between FDR control procedures and differentially private mechanisms. Are test statistics above (or below) a certain threshold worth making differentially private? While all these questions are tangential to this work, we believe they contain fruitful ideas for future research.

Chapter 5

Experimental Evaluation

5.1 Goals

We now turn to demonstrating the applicability of CUSTODES to the real world. The purpose of this chapter is to understand the feasibility of using CUSTODES for certifying valid hypothesis testing. The goals of our experimental evaluations are as follows:

- Thoroughly evaluate CUSTODES on a variety of datasets in terms of overall runtime and dataset characteristics.
- Ensure the correctness of statistical tests computed through CUSTODES matches those of the theoretical guarantees and compare the error (if any) to the equivalent computation performed outside of CUSTODES using the SciPy [48] scientific computing library in Python.
- Measure the impact of the number of parties in the system on the overall computation time required for each statistical test on a combination of datasets, parties, and other parameters.
- Evaluate the auditing process for statistical tests computed through CUSTODES.

5.2 Experiment Setup

5.2.1 Implementation and Environment

We implemented CUSTODES in Go 1.10.1. The code is open-source and available at <https://github.com/sachaservan/custodes>. The implementation of statistical tests closely follows the pseudo-code described in § 3.4. We implement the threshold-variant of the Paillier encryption scheme for performing the computation required for evaluating statistical tests but deviate slightly when computing division gates as we found that using linear-secret sharing methods for computing division was more efficient in practice (see § 2.6). As such, we implement `FPDiv` protocol using Shamir secret sharing [62] as the underlying primitive for MPC and convert from Paillier ciphertext to shares using the protocol described in § 2.6. We stress, however, that this is only done for *two ciphertexts* (numerator and denominator of the division or square-root reciprocal gate) and not the entire computation.

All experiments were conducted on a single machine with Intel Xeon E5 v3 (Haswell) @ 2.30GHz processors (40 cores in total) and 36GB of RAM, running Ubuntu 16.04 LTS. Unless otherwise stated, each result is the average over five separate runs (for each combination of parameters). While in most cases the variance in runtime across runs was minimal, we nonetheless report 95%-confidence markers (under a normal approximation assumption).

We emphasize that our implementation is unoptimized and does not make use of pre-computation and parallelization techniques to reduce interactive protocol execution overhead. Therefore, the results of our experiments should be viewed as an upper-bound on the runtime required for a real-world, optimized, implementation of CUSTODES.

5.2.2 Datasets

We evaluate each statistical test on synthetic and real-world datasets. Notice, however, that CUSTODES’s performance is not impacted by the distribution of the un-

derlying data since the data itself is encrypted. We nonetheless evaluate CUSTODES on two real-world datasets to illustrate this fact.

The real-world datasets are obtained from the UC Irvine Machine Learning Repository [2]. For experiments involving Student’s t-test and Pearson Correlation test we use the Abalone dataset containing weights and heights of 4177 Tasmanian abalones. For Chi-Squared we use the Pittsburgh Bridges dataset which contains categorical data on 108 bridges in the city of Pittsburgh.

In addition to these datasets, we generate three continuous synthetic datasets containing 1,000, 5,000 and 10,000 rows, respectively, with random real value entries ranging between 0 and 100. We use these datasets to evaluate the Student’s t-test and the Pearson Correlation test. For evaluating the Chi-Squared test, we generate three categorical datasets containing 20 mutually exclusive categories and compute the Chi-Squared statistic on a subset of 5, 10 and 20 categories to demonstrate the impact of varying the number of selected attributes. The characteristics of the datasets used in our evaluation are summarized in Table 5.1.

Dataset	Size	# Attributes	Type	Range
Abalone	4177	2	Continuous	[0, 1.37]
Pittsburgh	108	4	Categorical	{0, 1}
cat_1k	1000	5, 10, 20	Categorical	{0, 1}
cat_5k	5000	5, 10, 20	Categorical	{0, 1}
cat_10k	10000	5, 10, 20	Categorical	{0, 1}
rand_1k	1000	2	Continuous	[0, 100.0]
rand_5k	5000	2	Continuous	[0, 100.0]
rand_10k	10000	2	Continuous	[0, 100.0]

Table 5.1: Dataset characteristics

5.2.3 Evaluation

We benchmark all statistical tests with 3, 5 and 11 parties, where a threshold majority is required to decrypt in each case ($t = 2$, $t = 3$ and $t = 6$, respectively). Each party is run as a separate process and with access to two cores on the computing machine. In other words, each party is simulated as a dual-core machine, separate from other

parties¹. This setup allows us to simulate a multi-party computing environment while simultaneously controlling for all external variables, (e.g., network latency) in the system. We set the simulated network latency (delay between communication rounds) to 1ms which simulates average latencies observed on a local area network (LAN); a standard setup used for bench-marking multi-party computations [67, 7].

P-value Absolute Errors				
Statistical Test	Mean Error	Std.	Min	Max
Student's t-test	5.86×10^{-5}	1.00×10^{-4}	0.0	2.32×10^{-4}
Pearson Correlation	4.43×10^{-11}	5.21×10^{-11}	0.0	2.05×10^{-10}
Chi-Squared	2.35×10^{-9}	1.22×10^{-9}	1.37×10^{-20}	4.48×10^{-9}

Table 5.2: Absolute error of statistical tests computed through CUSTODES compared to the equivalent test computed using the SciPy package.

5.2.4 Parameters

To ensure all tests are evaluated in a way that enables comparisons between results, we fix the parameters of CUSTODES ahead of time to satisfy the requirements imposed by all three statistical tests (and datasets) and do not change the parameters between experiments. Specifically, we set $f = 30$ which provides approximately 9 decimal places of precision. We let the statistical security parameter $\kappa = 40$ which provides 40-bits of statistical security; a common default value [7]. The largest value found in all the datasets is 100.0 so we set $\eta_{max} = 100.0$ and the number of entries in the largest dataset is $N_{max} = 10,000$. Hence, we fix $\ell = 100$ which satisfies the parameter requirements for all three statistical tests. Finally, we set N (the Paillier modulus) to be a 1024-bit composite which both ensures security of ciphertexts and guarantees that \mathbb{Z}_N (the plaintext message space) is large enough to support statistical test computations with the given parameters.

¹The machine on which the experiments are conducted has 40 cores which allows such a setup without CPU swapping.

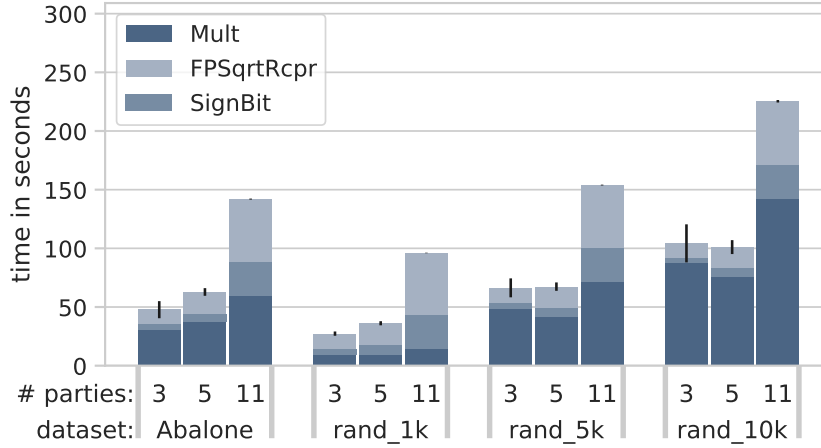


Figure 5-1: Runtime comparisons for the Student's t-test.

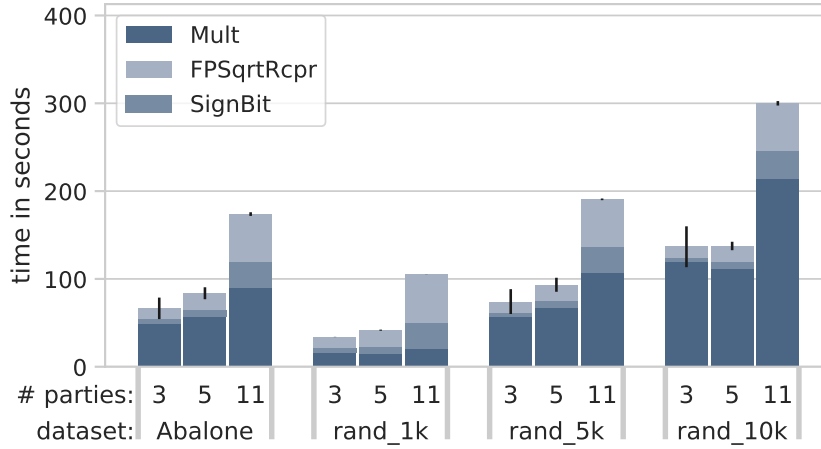


Figure 5-2: Runtime comparisons for the Pearson Correlation test.

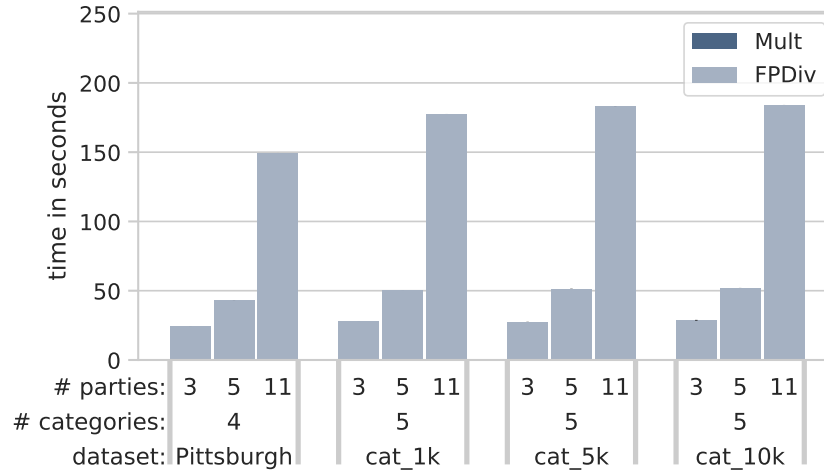
5.3 Results

We break down the results into four sections each of which roughly corresponds to a goal that we outline at the beginning of this section.

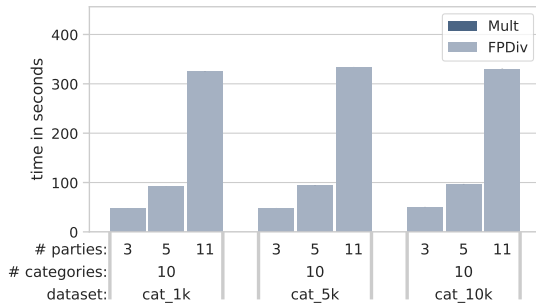
System Setup. Across all experiments, the amount of time required to setup CUSTODES (i.e., generating keys, encrypting the dataset, etc) remained below 45 seconds with mean 16 seconds, std. 14. This affirms that the overhead placed on the data owner is minimal in addition to being a one-time cost incurred at system setup time.

Correctness. We compare the resulting precision of each statistical test computed

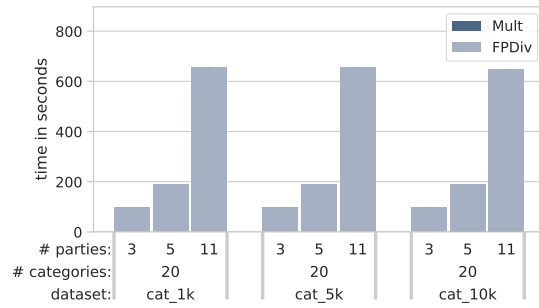
in CUSTODES with results obtained from computing the same test over the same data using the Python SciPy Library [48]. The absolute error for all tests was between 0.0 and 0.00023. The mean absolute error over all tests was 1.9×10^{-5} . We observe that the empirical error obtained in CUSTODES matches the precision expected based on the parameter selection, even with small variation in precision due to the probabilistic correctness of the TruncPR protocol.



(a) Chi-Squared test with 5 selected categories.



(b) 10 selected categories.



(c) 20 selected categories.

Figure 5-3: Runtime comparisons for the Chi-Squared test.

Runtime Evaluation. We run each statistical test on each configuration of parameters five times and report the average runtime in Figures 5-1, 5-2, and 5-3. The Chi-Squared test had the highest maximum runtime at 11m18s, and mean runtime of 2m54s across all combinations of parties, datasets, and category selections. Both Student's t-test and Pearson Correlation had a maximum runtime of approximately

4m50s with a mean of approximately 2 minutes across combinations of parties and datasets.

In all experiments, computing division gates (or reciprocals) required the most computation time relative to the other protocols invoked. This observation is not unexpected considering that it is the most computationally expensive protocol, along with the sign bit extraction protocol, invoked by each test due to the need for decomposing encrypted values into their bit-wise representations. Furthermore, the results demonstrate that for larger datasets, interactive multiplication (which is required by all interactive protocols) is a dominant factor in the overall computations, and moreover, dependent on the size of the dataset (with the exception of the Chi-Squared test). Perhaps surprisingly, the number of parties involved has a larger impact on performance than the size of the dataset. This is due to the high overhead incurred by interactive protocols which require many interactions between parties. Both Protocol 6 and 7, require only a single division operation making the computation of the division (resp. reciprocal) gate independent of the dataset size. When computing the Chi-Squared test, division is overwhelmingly the dominant contributor to the overall runtime (see Figure 5-3). This is due to the nature of the Chi-Squared statistic which *requires one division per category* (in the general case) which equates to a total of 20 calls to `FPDiv` for the largest selection of categories used in the Chi-Squared test experiments. In practice, under certain assumptions (e.g., when the expectation is uniform across categories or the totals for each category are made public) it would be possible to evaluate the Chi-Squared test with only a single division operation (for the purpose of obtaining the reciprocal) and would thus considerably reduce the interactive overhead.

Finally, we evaluate the runtime of computing a single multiplication gate with both Paillier and linear secret sharing instantiation of the interactive protocols to demonstrate the practical motivation of using the latter for computing highly-interactive gates such as division. Figure 5-4 shows the total time required for computing a single multiplication (over LAN) as a function of parties.

Again, we stress that we make no attempt to optimize the interactivity or complex-

ity of the protocols. With many recent developments in multi-party computations, overall runtime can be significantly improved. Nonetheless, our results demonstrate practical feasibility as-is and we believe that with the right optimizations, we could see a two to three fold reduction in the online computation time, even in upgraded security settings, based on benchmarks provided by general multi-party protocols frameworks such as SPDZ [19, 49].

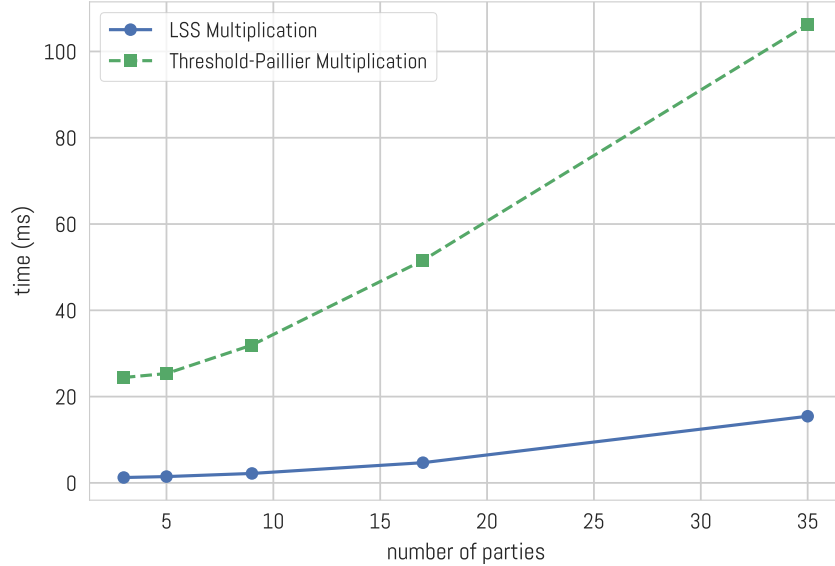


Figure 5-4: Runtime comparisons for computing a single multiplication interactively as a function of the number of parties, using the threshold Paillier and linear secret sharing as the underlying method.

Interactive Protocol Complexity. We measure the total number of multiplications required per statistical test and combination of parameters. Since `Mult` is the fundamental building block of *all* interactive protocols, this measurement provides an estimate for the total number of interactive rounds required between parties for computing a given test. We report the results in Table 5.3 and 5.4).

For both Student’s t-test and Pearson Correlation test the number of multiplications ranged between 35,969 and 57,616 and was dependent on the total number of rows in the dataset. We stress that this also includes the computation of the division gate and therefore the number of multiplications does not scale linearly with the number of rows for this pair of tests (given there is only a single division operation).

Student’s t-test and Pearson Correlation Complexity		
Dataset	Student’s t-test	Pearson Correlation
Abalone	35,969	40,147
rand_1k	29,615	30,616
rand_5k	37,615	42,616
rand_10k	47,615	57,616

Table 5.3: Number of multiplications required to compute a Student’s t-test and Pearson Correlation test for each dataset.

Chi-Squared Test Interactive Complexity			
Dataset	5 categories	10 categories	20 categories
Pittsburgh	N/A	N/A	N/A
cat_1k	138,105	276,180	552,330
cat_5k	138,105	276,180	552,330
cat_10k	138,105	276,180	552,330

Table 5.4: Number of multiplication required to compute a Chi-Squared test for each dataset.

For Chi-Squared test, the number of multiplications is independent of the number of rows. This is expected given that the protocol for computing Chi-Squared is dependent only on the number of categories selected and not on the number of observations in the dataset.

Auditing. For each test computed in our evaluation, we store the locally computed ciphertext trace in addition to the results of interactive protocols which we then use to evaluate the audit process for each computed statistical test. As expected, the audit verified successfully across all computed tests. The total audit time was consistently below 25 seconds with mean 12s, std. 9s. This demonstrates that while computing statistical tests in CUSTODES incurs a computational overhead, the auditing process remains relatively efficient.

5.3.1 A Note on Scalability

The results of the runtime evaluation begs the question: Can CUSTODES scale to a setting with more than a handful of computing parties? We believe the answer is yes. We note that in the experimental evaluation we set the threshold of parties

necessary for decryption and computations to be a *majority* of the total number of parties in the system. In practice, however, this is too stringent a requirement. From a technical standpoint, a system with n parties need only have t of them present in the computation phase and t need not scale with n . Therefore, even in a setting with thousands of researchers, similar performance can be achieved as demonstrated in our evaluation. Furthermore, as mentioned, our implementation does not exploit many parallelization techniques and efficiency improvements made possible with recent advances in multi-party computation protocols. With this in mind, we believe the results of our evaluation provides convincing evidence for the practical nature of CUSTODES in the real-world application.

Chapter 6

Discussion

6.1 Alternative Instantiations

In conclusion, we briefly mention alternative instantiations of CUSTODES. While these instantiations have several drawbacks compared to the Paillier-based implementation, we believe they may be of theoretical and practical interest in certain contexts where CUSTODES might be used.

6.1.1 FHE-based Instantiation

An alternative instantiation of CUSTODES can be achieved using *threshold fully-homomorphic encryption* (TFHE). Fully homomorphic encryption (FHE) allows one to evaluate arbitrary functions on encrypted data without decrypting the data [63, 35]. Unlike additively-homomorphic schemes, FHE enables the computation of both linear and non-linear arithmetic gates. Recent advances have led to threshold fully homomorphic encryption [44, 1] which enables computing with a single public key. The *Setup*, *Compute* and *Audit* phases of this instantiation are similar to the main instantiation with the exception of the *interactive protocols* which are no longer required for the evaluation of the statistical test circuit except for revealing the final result. While at first glance this may seem appealing, there are several drawbacks to this approach. Due to many inefficiencies surrounding FHE such an instantiation is likely

to be far less practical compared to the instantiation we present [35], at least for the foreseeable future. Furthermore, while in our instantiation the auditing is fairly efficient because the auditing entity only needs to evaluate *linear* gates (see § 3.2), an FHE-based approach would require the auditor to recompute the entire circuit from scratch.

Nonetheless, we believe that with sufficient improvements to the practicality of FHE schemes, such an instantiation may be of interests in certain contexts. Figure 6-1 displays a high-level overview of CUSTODES instantiated with FHE.

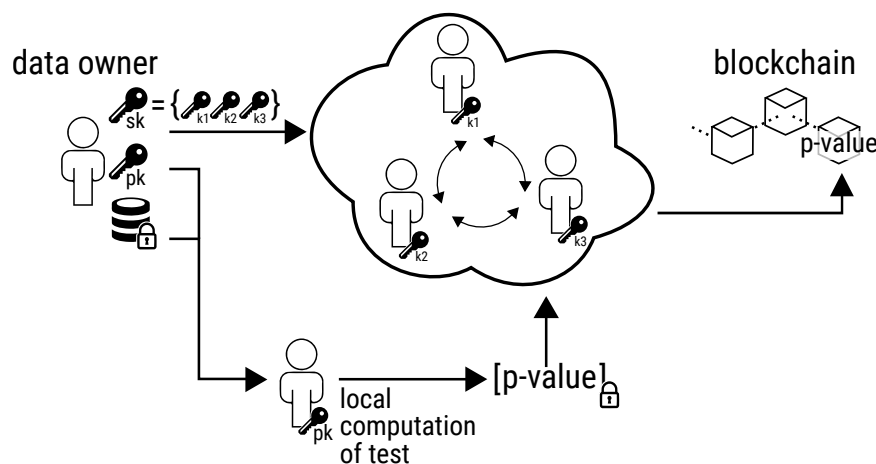


Figure 6-1: High-level overview of an FHE-based implementation of CUSTODES.

We now briefly outline how the TFHE-based version of CUSTODES can be instantiated. Formally, TFHE is a collection of algorithms TFHE.Setup , TFHE.Encrypt , TFHE.Eval , TFHE.PartDec , and TFHE.FinDec [44]. Using TFHE as a building-block, we can instantiate CUSTODES as follows.

TFHE CUSTODES.

Setup. During the setup phase, the trusted data owner executes algorithm Setup that outputs a public key pk and secret key shares $\text{sk}_1, \dots, \text{sk}_v$. The data owner then runs $\text{TFHE.Encrypt}(\text{pk}, \mathcal{D})$ on dataset \mathcal{D} that returns encoded dataset $\llbracket \mathcal{D} \rrbracket$, where every value of \mathcal{D} is individually encrypted. The data owner sends $\text{pk}, \text{sk}_i, \llbracket \mathcal{D} \rrbracket$ to \mathcal{P} and publishes the encrypted dataset $\llbracket \mathcal{D} \rrbracket$.

Compute. Let \mathcal{R} be the researcher that wishes to run a statistical test \mathcal{T} (rep-

resented as a Boolean or arithmetic circuit). \mathcal{R} posts $(\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, \perp)$ to \mathcal{B} . \mathcal{R} then runs the deterministic algorithm $\text{TFHE.Eval}(\mathcal{T}, \llbracket \mathcal{D} \rrbracket)$ to evaluate the circuit using \mathcal{D} as input. The output of this algorithm is the encrypted result of the statistical test $([t], [p])$. \mathcal{R} posts the message $(\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, ([t], [p]))$ to \mathcal{B} . Every party $\mathcal{P}_j^{\text{pk}}$, upon receiving $(\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, ([t], [p]))$ verifies that it was signed by a valid researcher and/or party. If the verification succeeds, \mathcal{P}_j executes $\text{TFHE.PartDec}([t], \text{sk}_j)$ and $\text{TFHE.PartDec}([p], \text{sk}_j)$ and obtains the partial decryption (t_j, p_j) of the result. \mathcal{P}_j then posts $(\tau, \mathcal{P}_j^{\text{pk}}, \mathcal{T}, (t_j, p_j))$ to \mathcal{B} . With at least t partial decryptions posted to \mathcal{B} , the result can be decrypted by running $\text{TFHE.FinDec}(T)$ where T is a sequence of all partial decryptions that correspond to τ th test.

Audit. Auditing follows an almost identical procedure as *Compute* where the auditor replaces the role of the researcher in computing the circuit. A test with certificate $(\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, (t, p))$ can be audited for correctness by any entity with access to \mathcal{B} , $\llbracket \mathcal{D} \rrbracket$ and the public key pk used to encrypt \mathcal{D} . The auditor then checks whether the output of $\text{TFHE.Eval}(\mathcal{T}', \llbracket \mathcal{D} \rrbracket)$ matches the output of $\text{TFHE.Eval}(\mathcal{T}, \llbracket \mathcal{D} \rrbracket)$ where \mathcal{T}' corresponds to a valid statistical test circuit. If the evaluation matches then the computation is deemed valid.

6.1.2 LSS-based Instantiation

Yet another alternative instantiation is one based solely on secret shared data. The downside of this approach is that the owner must share the entire database among the parties which also increases the number of interaction.

Compared to the threshold-Paillier based approach, parties are required to interact during *every* non-linear operation of a statistical test increasing the overhead required of the network. An advantage, however, is that coupled with zero-knowledge proofs and an assumption of an honest majority of parties, such an instantiation can guarantee the correctness of computations without the need for an audit. Figure 6-2 presents a high-level overview of such an instantiation. Note that researcher no longer perform any of the computations locally and offload the storage and computations to

the set of parties in the network.

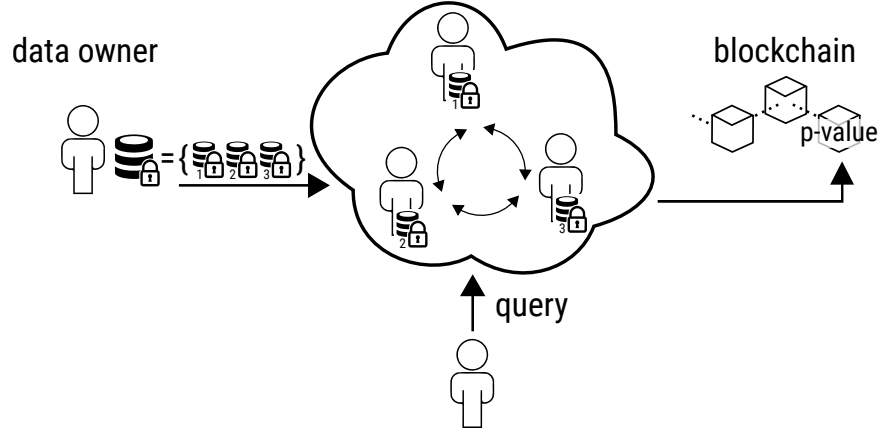


Figure 6-2: High-level overview of an FHE-based implementation of CUSTODES.

Using a secret sharing scheme such as Shamir [62], the owner would secret share the dataset \mathcal{D} with all parties in the network such that a threshold t of the parties can reconstruct \mathcal{D} . Researchers interested in computing a statistical test would issue a query to the network (or post a message to \mathcal{B}) requesting that parties compute a statistical test over their shares of the data. Note that researchers take no part in the computations.

LSS CUSTODES. We omit the details of secret sharing schemes which are described at a high-level in § 2.6 and in-depth in [5].

Setup. During the setup phase the owner runs `CreateShares()` on each value of the dataset \mathcal{D} to distribute \mathcal{D} among the set of computing parties $\{\mathcal{P}_1, \dots, \mathcal{P}_v\}$ such that any t (or more) of the parties can collectively recover the shared values. As in the previous instantiation, after sharing the dataset and establishing access rights, the data owner goes offline.

Compute. For every statistical test a researcher \mathcal{R} wants to execute, \mathcal{R} posts a corresponding arithmetic circuit of this test, \mathcal{T} (or some specified test identifier) to \mathcal{B} . Let τ be the index of this test in \mathcal{B} . All the parties download the circuit, store the index of the test and engage in the computation. Once the parties have obtained their share of the final result $(\langle \mathbf{t} \rangle_i, \langle \mathbf{p} \rangle_i)$, they post message $(\tau, \mathcal{R}^{\text{pk}}, \mathcal{T}, (\langle \mathbf{t} \rangle_i, \langle \mathbf{p} \rangle_i))$ to

\mathcal{B} . Using these public shares, \mathcal{R} can obtain test result from \mathcal{B} .

Audit. If an honest-majority of parties is assumed and the researchers simply issue statistical test queries, then no auditing of the *computation* is necessary and it suffices to check that the FDR control procedure was applied correctly given the transcript of tests performed on the shared dataset.

On the other hand, if researchers supply the circuit to be evaluated or the parties are deemed malicious, then an auditor can verify the computed circuit matches a valid statistical test and ensure that the parties evaluated the circuit correctly (e.g., using zero-knowledge proofs or other techniques for ensuring correctness when the parties are adversarial [19, 16]).

6.2 Conclusion

In this thesis we present CUSTODES, a system that certifies hypothesis testing using proven cryptographic techniques and a decentralized certifying authority. CUSTODES computes statistical test over encrypted data and uses a handful of existing cryptographic building-blocks to provide auditability of those results to third parties. The protocol we describe is a novel approach to the problem of ensuring validity in the scientific process.

In addition, we present several novel extensions to efficient MPC computation techniques when it comes to evaluating complex arithmetic circuits required by statistical tests with minimal interaction between parties and researchers computing over the data. We present an extension for making the computations differentially private without imposing additional requirements on the data owner which we believe is a contribution which may be of independent interest to the field.

Based on the results of our implementation and evaluation, we believe that CUSTODES is a viable solution to prevent p-hacking and controlling false-discoveries in statistical testing procedures. Through enforcing accountability, CUSTODES ensures reproducibility in scientific studies and inhibits the possibility of p-hacking.

Bibliography

- [1] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 483–501. Springer, 2012.
- [2] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.
- [3] C Glenn Begley and Lee M Ellis. Drug development: Raise standards for pre-clinical cancer research. *Nature*, 483(7391):531, 2012.
- [4] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the royal statistical society. Series B (Methodological)*, pages 289–300, 1995.
- [5] Dan Bogdanov. Foundations and properties of shamirs secret sharing scheme research seminar in cryptography. *University of Tartu, Institute of Computer Science*, 1, 2007.
- [6] Dan Bogdanov, Liina Kamm, Sven Laur, and Ville Sokk. Rmind: a tool for cryptographically secure statistical analysis. *IEEE Transactions on Dependable and Secure Computing*, 15(3):481–495, 2018.
- [7] Dan Bogdanov, Sven Laur, and Jan Willemsen. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.
- [8] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.
- [9] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.
- [10] Octavian Catrina and Sebastiaan De Hoogh. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, pages 182–199. Springer, 2010.

- [11] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *International Conference on Financial Cryptography and Data Security*, pages 35–50. Springer, 2010.
- [12] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi. Solidus: Confidential distributed ledger transactions via PVORM. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 701–717, 2017.
- [13] William G Cochran. The χ^2 test of goodness of fit. *The Annals of Mathematical Statistics*, pages 315–345, 1952.
- [14] Andy Cockburn, Carl Gutwin, and Alan Dix. Hark no more: on the preregistration of chi experiments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 141. ACM, 2018.
- [15] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 316–334. Springer, 2000.
- [16] Ronald Cramer, Ivan Damgård, and Jesper B Nielsen. Multiparty computation from threshold homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 280–300. Springer, 2001.
- [17] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pages 285–304. Springer, 2006.
- [18] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of Pailliers public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.
- [19] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority—or: breaking the spdz limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.
- [20] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Annual International Cryptology Conference*, pages 247–264. Springer, 2003.
- [21] Ivan Damgård and Rune Thorbek. Efficient conversion of secret-shared values between different fields. *IACR Cryptology ePrint Archive*, 2008:221, 2008.
- [22] Kay Dickersin, SS Chan, TC Chalmersx, HS Sacks, and H Smith Jr. Publication bias and clinical trials. *Controlled clinical trials*, 8(4):343–353, 1987.

- [23] Yadolah Dodge. *The concise encyclopedia of statistics*. Springer Science & Business Media, 2008.
- [24] Olive Jean Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.
- [25] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.
- [26] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. Guilt-free data reuse. *Communications of the ACM*, 60(4):86–93, 2017.
- [27] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. Preserving statistical validity in adaptive data analysis. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 117–126. ACM, 2015.
- [28] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.
- [29] Cynthia Dwork, Weijie J Su, and Li Zhang. Differentially private false discovery rate control. *arXiv preprint arXiv:1807.04209*, 2018.
- [30] Michael J Flynn. On division by functional iteration. *IEEE Transactions on Computers*, 100(8):702–706, 1970.
- [31] Dean P Foster and Robert A Stine. α -investing: a procedure for sequential control of expected false discoveries. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(2):429–444, 2008.
- [32] Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel Weitzner. Practical accountability of secret processes. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 657–674. USENIX Association, 2018.
- [33] Marco Gaboardi, Hyun Lim, Ryan Rogers, and Salil Vadhan. Differentially private chi-squared hypothesis testing: Goodness of fit and independence testing. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 2111–2120, 2016.
- [34] Marco Gaboardi, Hyun-Woo Lim, Ryan M Rogers, and Salil P Vadhan. Differentially private chi-squared hypothesis testing: Goodness of fit and independence testing. In *ICML’16 Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*. JMLR, 2016.

- [35] Craig Gentry and Shai Halevi. Implementing Gentry’s fully-homomorphic encryption scheme. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 129–148. Springer, 2011.
- [36] Mohammad M Ghassemi, Stefan E Richter, Ifeoma M Eche, Tszyi W Chen, John Danziger, and Leo A Celi. A data-driven approach to optimized medication dosing: a focus on heparin. *Intensive care medicine*, 40(9):1332–1339, 2014.
- [37] Robert E Goldschmidt. *Applications of division by convergence*. PhD thesis, Massachusetts Institute of Technology, 1964.
- [38] Slawomir Goryczka, Li Xiong, and Vaidy Sunderam. Secure multiparty aggregation with differential privacy: A comparative study. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 155–163. ACM, 2013.
- [39] Thore Graepel, Kristin Lauter, and Michael Naehrig. ML confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology (ICISC)*, 2013.
- [40] Megan L Head, Luke Holman, Rob Lanfear, Andrew T Kahn, and Michael D Jennions. The extent and consequences of p-hacking in science. *PLoS biology*, 13(3):e1002106, 2015.
- [41] Katharine E Henry, David N Hager, Peter J Pronovost, and Suchi Saria. A targeted real-time early warning score (trewscore) for septic shock. *Science translational medicine*, 7(299):299ra122–299ra122, 2015.
- [42] John PA Ioannidis. Contradicted and initially stronger effects in highly cited clinical research. *Jama*, 294(2):218–228, 2005.
- [43] John PA Ioannidis. Why most published research findings are false. *PLoS medicine*, 2(8):e124, 2005.
- [44] Aayush Jain, Peter MR Rasmussen, and Amit Sahai. Threshold fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2017:257, 2017.
- [45] Leslie K John, George Loewenstein, and Drazen Prelec. Measuring the prevalence of questionable research practices with incentives for truth telling. *Psychological science*, 23(5):524–532, 2012.
- [46] Aaron Johnson and Vitaly Shmatikov. Privacy-preserving data exploration in genome-wide association studies. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’13, pages 1079–1087, 2013.
- [47] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

- [48] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2018-07-01].
- [49] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 158–189. Springer, 2018.
- [50] Maurice George Kendall et al. The advanced theory of statistics. *The advanced theory of statistics.*, (2nd Ed), 1946.
- [51] Norbert L Kerr. Harking: Hypothesizing after the results are known. *Personality and Social Psychology Review*, 2(3):196–217, 1998.
- [52] Samuel Kotz, Tomasz Kozubowski, and Krzysztof Podgorski. *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*. Springer Science & Business Media, 2012.
- [53] Leslie Lamport. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.
- [54] Kristin Lauter, Adriana López-Alt, and Michael Naehrig. Private computation on encrypted genomic data. In *International Conference on Cryptology and Information Security in Latin America*, pages 3–27. Springer, 2014.
- [55] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 394–403. IEEE, 2003.
- [56] Peter Markstein. Software division and square root using goldschmidts algorithms. In *Proceedings of the 6th Conference on Real Numbers and Computers (RNC6)*, volume 123, pages 146–157, 2004.
- [57] Louis Mayaud, Peggy S Lai, Gari D Clifford, Lionel Tarassenko, Leo Anthony G Celi, and Djillali Annane. Dynamic data during hypotensive episode improves mortality predictions among patients with sepsis and hypotension. *Critical care medicine*, 41(4):954, 2013.
- [58] R Munroe. xkcd: Significant, 1995.
- [59] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. <http://www.bitcoin.org/bitcoin.pdf>.
- [60] Ushma S Neill. Publish or perish, but at what cost? *The Journal of clinical investigation*, 118(7):2368–2368, 2008.
- [61] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.
- [62] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

- [63] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010.
- [64] Thomas H Wonnacott and Ronald J Wonnacott. *Introductory statistics*, volume 5. Wiley New York, 1990.
- [65] David Wu and Jacob Haven. Using homomorphic encryption for large scale statistical analysis. Technical report, Technical Report: cs. stanford.edu/people/dwu4/papers/FHESI Report. pdf, 2012.
- [66] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin E. Lauter, and Michael Naehrig. Crypto-nets: Neural networks over encrypted data. *CoRR*, abs/1412.6181, 2014.
- [67] Yihua Zhang, Aaron Steele, and Marina Blanton. Picco: a general-purpose compiler for private distributed computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 813–826. ACM, 2013.
- [68] Yuchen Zhang, Wenrui Dai, Xiaoqian Jiang, Hongkai Xiong, and Shuang Wang. Foresee: Fully outsourced secure genome study based on homomorphic encryption. 15:S5, 12 2015.
- [69] Zheguang Zhao, Lorenzo De Stefani, Emanuel Zgraggen, Carsten Binnig, Eli Upfal, and Tim Kraska. Controlling false discoveries during interactive data exploration. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 527–540. ACM, 2017.
- [70] Jing Zhou, Dean Foster, Robert Stine, and Lyle Ungar. Streaming feature selection using alpha-investing. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 384–393. ACM, 2005.
- [71] Guy Zyskind et al. *Efficient secure computation enabled by blockchain technology*. PhD thesis, Massachusetts Institute of Technology, 2016.
- [72] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 180–184. IEEE, 2015.

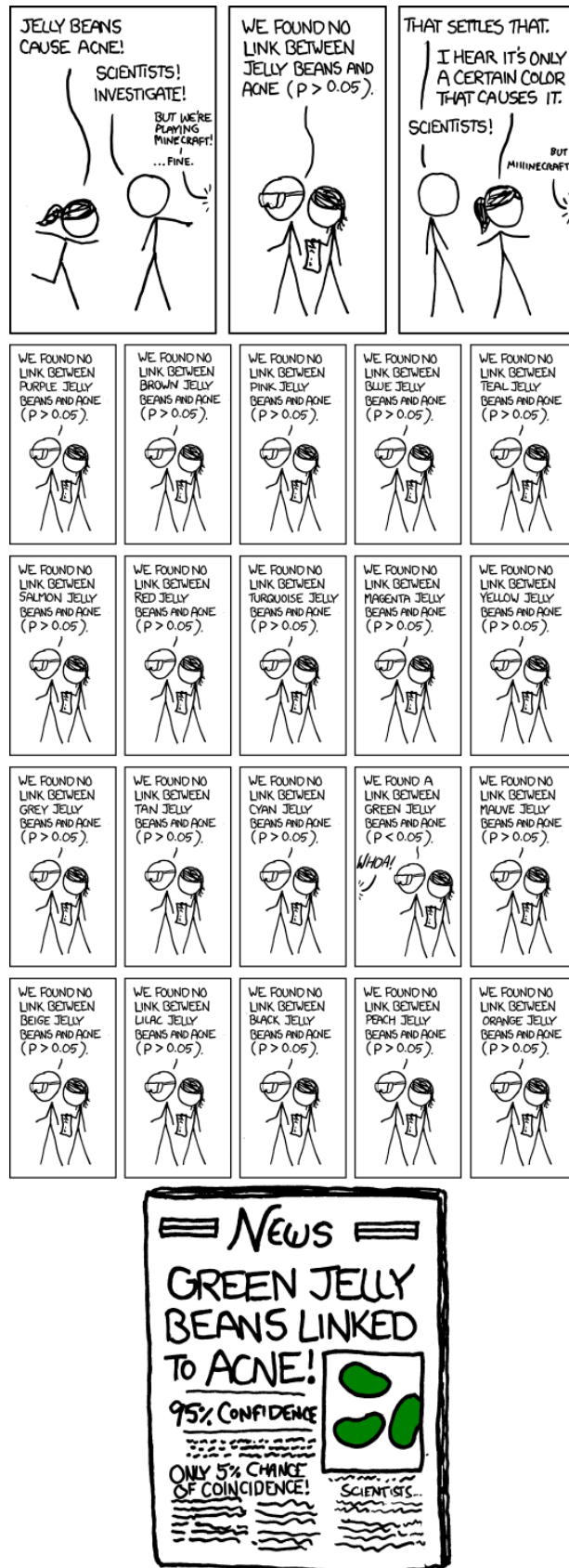


Figure 6-3: XKCD Cartoon on p-hacking [58].