Private Access Control for Function Secret Sharing

Sacha Servan-Schreiber[1] Simon Beyzerov[2,3] Eli Yablon[2,4] Hyojae Park[2,4]

[1]MIT CSAIL
[2]MIT PRIMES
[3]Massachusetts Academy of Math and Science at WPI
[4]Sharon High School

Outline

- Background & Preliminaries
- Private Access Control Lists
- Applications
- Constructions
- Evaluation

Background: Secret Sharing

Secret sharing is a way of splitting a value into multiple *shares*, such that:

1. the shares can be recombined to reveal the secret value and
2. strict subsets of shares reveal nothing about the secret value.

- We will use bracket notation $[v]_i$ to indicate the ith secret share of the value v.
- We will use $+$ for recombining: $[v]_1 + [v]_2 = v$.

Background: Function Secret Sharing (FSS)

Function secret sharing has the additional requirement that shares of f can be *evaluated* on an input x to obtain shares of $f(x)$.

- Given $[f]_i$, it is possible to efficiently compute $[f(x)]_i$ for any x.
- The secret shares $[f]_i$ are *succinct* (smaller than the truth table for f).

- We will use $[f(x)]_i$ to denote an evaluation of the *share* $[f]_i$ on input x.
- We will use $+$ for recombining: $[f(x)]_1 + [f(x)]_2 = f(x)$.

Background: Function Secret Sharing (FSS)

Function Secret Sharing

For a function $f : \{0,1\}^n \to \{0,1\}^*$ and $p \geq 2$ evaluators, FSS is described by the following (possibly randomized) algorithms:

$\text{Gen}(1^\lambda, f) \to ([f]_1, \cdots, [f]_p)$

- Splits the function into a set of compact secret shares $([f]_1, \cdots, [f]_p)$.

$\text{Eval}([f]_i, x) \to [f(x)]_i$

- Uses secret share $[f]_i$ and input x to output a secret share of $f(x)$.

$\text{Recover}([f(x)]_1, \cdots, [f(x)]_p) \to f(x)$

- Recovers $f(x)$ from the p secret shares.

See Boyle, Gilboa, and Ishai [1, 2].

Why is Function Secret Sharing (FSS) useful?

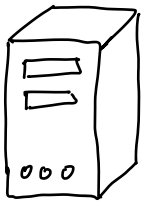
Suppose a client wants to run a function on data stored in the cloud but doesn't want to reveal the function to the cloud servers?

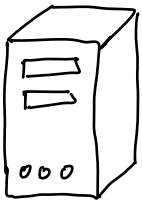
Solution

Assume non-collusion between servers and use FSS to **hide** the function.

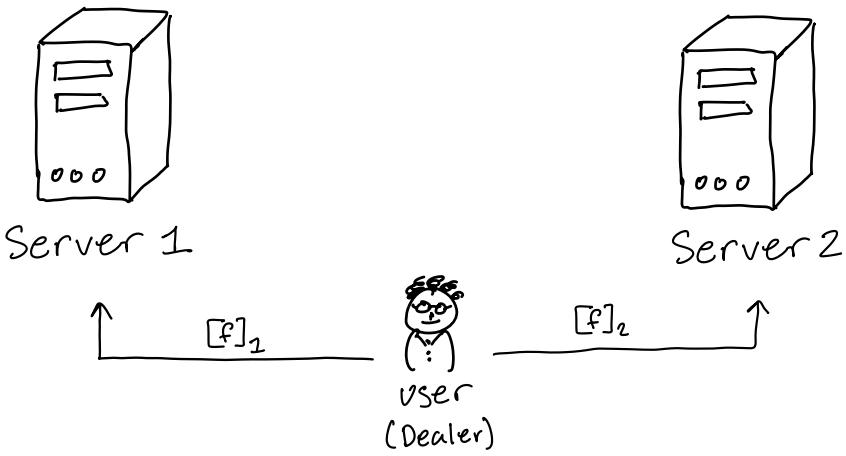
1. The client uses FSS to secret share the function f with the cloud servers.
2. The cloud servers evaluate the secret-shared function f and send the (secret-shared) result $f(x)$ back to the client.
3. The client locally recombines the shares to get $f(x)$.

Why is Function Secret Sharing (FSS) useful?

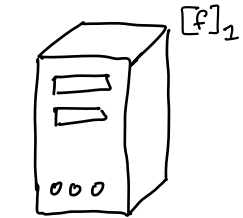


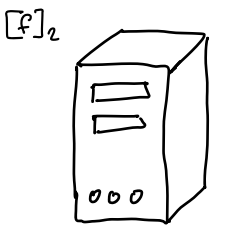


Why is Function Secret Sharing (FSS) useful?

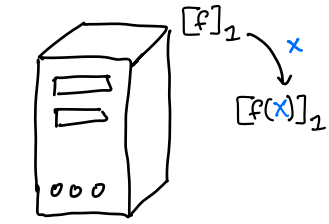


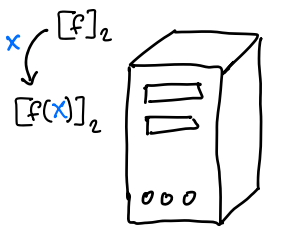
Why is Function Secret Sharing (FSS) useful?



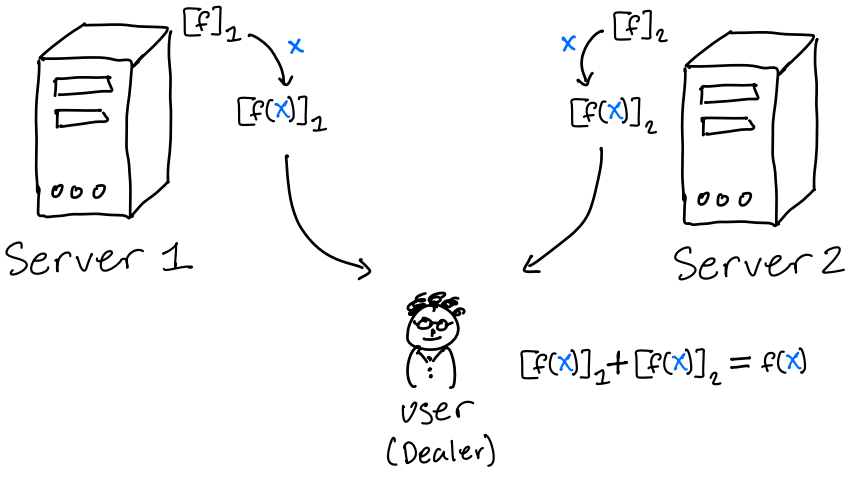


Why is Function Secret Sharing (FSS) useful?





Why is Function Secret Sharing (FSS) useful?



Applications of Function Secret Sharing (FSS)

FSS is useful in building privacy-preserving systems:

- Privately reading from a distributed database (private information retrieval)
  - e.g., private keyword search on a remote database [1, 3].
- Privately writing to a distributed database (private information writing)
  - e.g., anonymous communication [4, 5, 6, 7].
- Multi-party computation
  - e.g., generating pre-processing for multi-party computations [8].

Why is **Access Control** for FSS useful?

- FSS is used in systems with many users (some of which may be malicious).
- Users have different access rights for different functions.

We develop access control for FSS

Function evaluators want to make sure that only users who have "access rights" to a function f can secret share it.

For example: some users might only have access to a function foo$(\cdot)$ whereas other users might have access to a function bar$(\cdot)$.

Challenge

The **privacy** of FSS must be maintained (hiding which function is secret shared). However, privacy is often in conflict with access control [9].

Applications of Access Control for FSS

1. Private Information Retrieval with Access Control.
   For example: prevent users from accessing records in the database that they don't have permission to access (e.g., records belonging to other users, paid digital content downloading) [9, 10, 11].

2. Private Information Writing with Access Control.
   For example: prevent malicious users from "spamming" communication channels in anonymous communication or writing to records belonging to other users [7, 5, 6].

3. Anonymous authentication: proving that a user has a valid account without revealing which account belongs to the user.
   For example: students can access campus buildings without revealing their identity.

Modeling Access Control for Function Secret Sharing (FSS)

We introduce the notion of Private Access Control Lists (PACLs).

Goal: "The access key α gives access to the function f."

- Authenticating with access key α does not reveal anything about f.
- Don't know the access key α? Can't secret-share f with the evaluators!

We consider some fixed family of functions $\mathcal{F} = \{f_1, \ldots, f_N\}$ and access control list $\Lambda = \{\mathsf{vk}_1, \ldots, \mathsf{vk}_N\}$ such that vk_i is the verification key for f_i.

Modeling Access Control for Function Secret Sharing (FSS)

access verification keys

$\Lambda = (VK_1 \ \ldots \ VK_N)$

Server 1

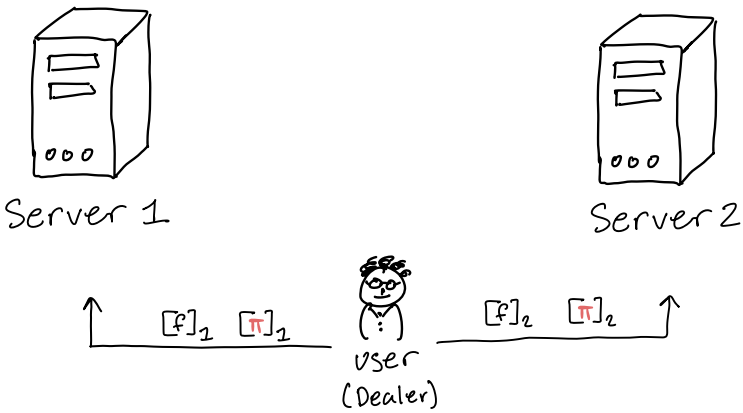
Server 2

User
(Dealer)

Modeling Access Control for Function Secret Sharing (FSS)

access verification keys

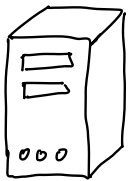
$\Lambda = (VK_1 \ \ldots \ VK_N)$



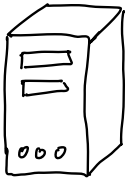
Modeling Access Control for Function Secret Sharing (FSS)

access verification keys

$\Lambda = (VK_1 \ \ldots \ VK_N)$



Server 1



Server 2

$[f]_1 \ \ [\pi]_1$

user
(Dealer)

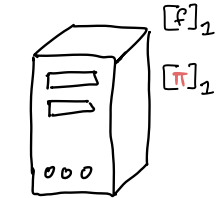
$[f]_2 \ \ [\pi]_2$

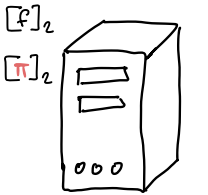
π = proof that user knows the access key for f.

Modeling Access Control for Function Secret Sharing (FSS)

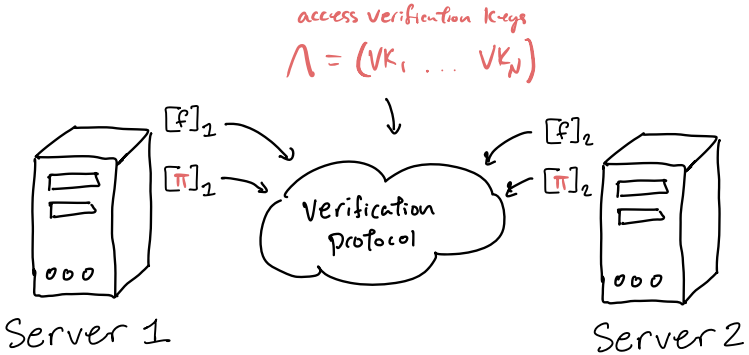
access verification keys

$\Lambda = (VK_1 \ \ldots \ VK_N)$

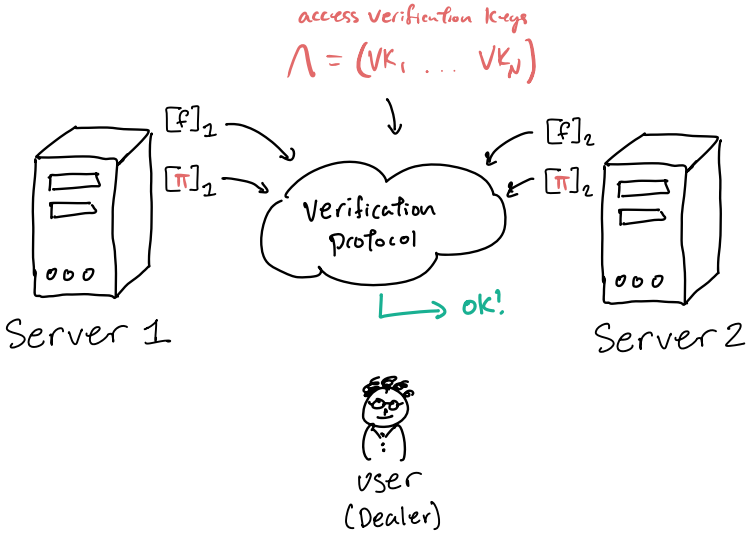




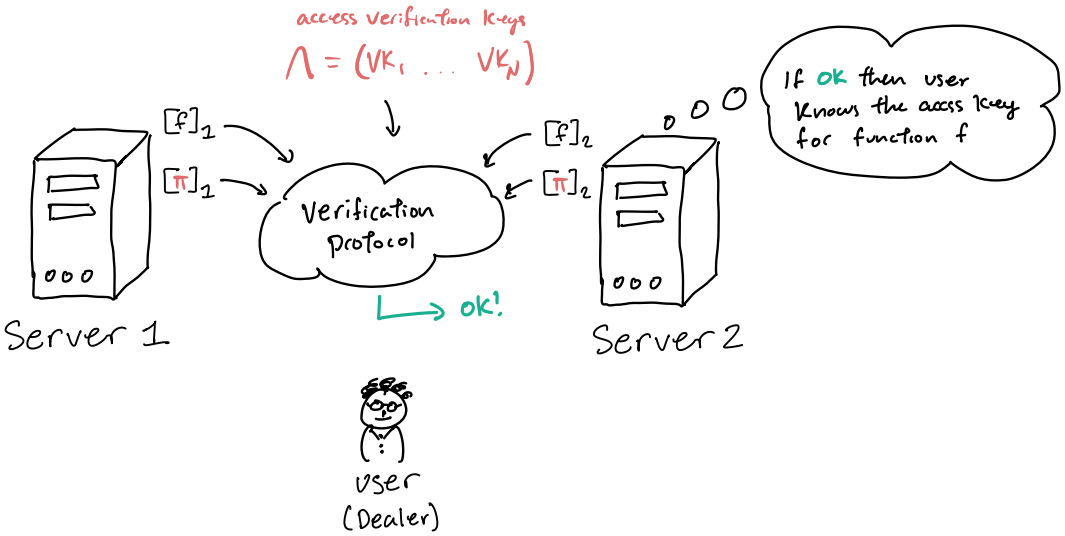
Modeling Access Control for Function Secret Sharing (FSS)



Modeling Access Control for Function Secret Sharing (FSS)



Modeling Access Control for Function Secret Sharing (FSS)



Building block: Access Control for Distributed Point Functions (DPFs)

Point functions:

- $P_j(x) = \begin{cases} 1 & \text{for } x = j, \\ 0 & \text{for } x \neq j. \end{cases}$
- Compact encoding of a length-N one-hot vector $\big(0, \cdots, 0, \underbrace{1}_{j\text{th index}}, 0, \cdots, 0\big)$.

DPFs (Distributed Point Functions) are FSS for point functions

DPFs can be used to construct FSS for more complex functions [1].

Access control for more complex functions follows (see paper).

FSS for Point Functions

DPFs can be used to instantiate FSS for a larger class of functions [1, 2]:

- Interval functions: like point functions but evaluate to 1 on a range of indices.
- NC^0 functions: the set of all functions that can be represented by a constant-depth boolean circuit (with two inputs per gate).
- Small function families (e.g., with a logarithmic domain) with a canonical ordering.
- Decision trees: higher-dimensional generalization of interval functions.

See [1, Section 3.2] for details on these transformations.

Modeling Access Control for Function Secret Sharing (FSS)

We introduce the notion of Private Access Control Lists (PACLs).

Goal: "The access key α gives access to the function f."

- Authenticating with access key α does not reveal anything about f.
- Don't know the access key α? Can't secret-share f with the evaluators!

We focus on constructing PACLs for DPFs.

PACLs for more complex functions follow (see paper for details).

Modeling Access Control for FSS

Private Access Control Lists

Fix $p \geq 2$ evaluators, let (Gen, Eval, Recover) be the algorithms describing the FSS scheme for a function family $\mathcal{F}$. A PACL scheme is described by:

$\mathsf{KeyGen}(1^\lambda, f) \to (\mathsf{vk}, \mathsf{sk})$ `// Used by the dealer or a trusted setup`

- Generates a new verification key vk and access key sk for function $f \in \mathcal{F}$.

$\mathsf{Prove}(f, \mathsf{sk}) \to ([\pi]_1, \ldots, [\pi]_p)$ `// Used by the prover (i.e., the dealer)`

- Uses access key sk to generate access proof shares $([\pi]_1, \ldots, [\pi]_p)$.

$\mathsf{Audit}(\Lambda, [f]_i, [\pi]_i) \to \tau_i$ `// Used by each verifier (i.e., evaluator)`

- Uses the access policy Λ, share $[f]_i$ (output of Gen), and proof share $[\pi]_i$ to generate an audit token τ_i.

$\mathsf{Verify}(\tau_1, \ldots, \tau_p) \to \text{yes/no}$ `// Used by each verifier (i.e., evaluator)`

- Use the audit tokens to determine access rights.

Applications of Access Control for FSS

Private Information Retrieval (PIR) with Access Control

- PIR is used to *privately* retrieve a record from a remote database.

  For example:

  - Retrieve a song from Spotify without revealing which song was downloaded.
  - Search Google without revealing the query to Google.

- PIR with Access Control can be used to prevent unauthorized users from accessing a particular record in the database (e.g., personal records, premium content).

Access control for FSS immediately enables access control for PIR, allowing the database to restrict access to certain records, without compromising on privacy.

Applications of Access Control for FSS

Private Information *Writing* (PIW) with Access Control

- PIW is used to *privately* write to a record in a remote database.

  For example:

  - Anonymous communication protocols, where users write to other users' mailboxes, use PIW to hide which mailbox is written to by a user [7, 5, 6, 4].
  - Privacy-preserving aggregate statistics, where users privately write their personal data into aggregate counters, use PIW to hide which counters are updated by a user [12, 2].

- PIW with Access Control can be used to prevent malicious users from corrupting communication channels or "spamming" records belonging to other users [7, 5, 6].

Access control for FSS immediately enables access control for PIW, allowing the databse to restrict which users are allowed to write to which records.

Applications of Access Control for FSS

Anonymous Authentication

Anonymous authentication allows a person to prove they are part of an authorized group, without revealing anything else about themselves.

For example:

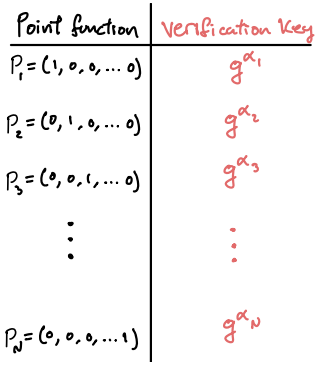
- A university student can prove that they are in the set of all students in a class, without revealing *who* they are.
- An employee can prove they have access the company's office building, without revealing their name or other identifying information.

Access control for FSS enables anonymous authentication (in a setting with two non-colluding servers) that integrates well with existing approaches to authentication.

Constructing PACLs for Distributed Point Functions (DPFs)

Setup and Assumptions

- Let $g \in \mathbb{G}$ be generator of $\mathbb{G}$.
- Assign DPF f_i access key $\mathsf{sk}_i := \alpha_i$ and $\mathsf{vk}_i := g^{\alpha_i}$
  - By the Discrete Logarithm Assumption, if α_i is random, then it is hard to find α_i given g^{α_i} [13]
- Evaluators store all of the verification keys: $\Lambda := (g^{\alpha_1}, \ldots, g^{\alpha_N})$.
- Dealer has the access key $\mathsf{sk}_i = \alpha_i$.



Constructing PACLs for Distributed Point Functions (DPFs)

Step 1: The dealer sends secret share $[\pi]$ as the proof, where $\pi := -\alpha_j$.

Step 2: The evaluators use $[f_j]$ to retrieve multiplicative shares of g^{α_j}:

$$g^{[\alpha_j]} := \prod_{k=1}^{N} (g^{\alpha_k})^{[f_j](k)} = g^{\langle(\alpha_1,\ldots,\alpha_N),([0],\ldots,[1],\ldots,[0])\rangle}.$$

Step 3: The evaluators compute the audit token:

$$\tau_i := g^{[\alpha_j]} \cdot g^{[\pi]} = g^{[\alpha_j - \alpha_j]} = g^{[0]}.$$

Step 4: Each evaluator broadcasts their audit token τ_i,
and all evaluators check that: $\prod_{i=1}^{p} \tau_i \stackrel{?}{=} g^0$.

Constructing PACLs for Distributed Point Functions (DPFs)

Note that the DPF f_j can be seen as efficiently encoding the (point-wise secret-shared) one-hot vector: $[\mathbf{e}_j] = ([f_j](1), \cdots, [f_j](j), \cdots [f_j](N)) = ([0], \cdots, [1], \cdots, [0])$.

Therefore, we can "select" *multiplicative* secret shares of g^{α_j} with the DPF f_j as:

$$g^{\langle(\alpha_1,\cdots,\alpha_N),[\mathbf{e_j}]\rangle} = \prod_{k=1}^{N} (g^{\alpha_k})^{[f_j](k)}$$

$$= g^{[0]\alpha_1} \cdots g^{[1]\alpha_j} \cdots g^{[0]\alpha_N}$$

$$= g^{[\alpha_j]}$$

Multiplication of a secret share by a constant is equivalent to "multiplying inside the brackets"

Security analysis of PACLs for DPFs

- **Completeness**

  If $\pi = -\alpha_j$, then $\prod_{i=1}^{p} \tau_i = \prod_{i=1}^{p} g^{[\alpha_j + \pi]_i} = g^{\alpha_j - \alpha_j} = g^0$ ✓

- **Proof-of-knowledge**

  The knowledge extractor recovers $\alpha_j = -\sum_{i=1}^{p} [\pi_j]_i$, the DL of $\mathsf{vk}_j := g^{\alpha_j}$ ✓

- **Zero-knowledge**

  Simulator generates *random shares* for $[\pi]_i$ and τ_i. In the real view, $[\pi]_i$ is secret-shared by the honest prover and therefore uniformly distributed. Each τ_i is also uniformly distributed (and computed independently of malicious verifiers) ✓

Optimized PACLs for Verifiable DPFs

- Operations on group elements in $\mathbb{G}$ are slow (require exponentiation) when $\mathbb{G}$ is a Diffie-Hellman group.
- Operations in a field $\mathbb{F}$ are much faster (require only addition mod a prime q).

Main idea: perform inner-product in the field over which $\mathbb{G}$ is defined to obtain an additive secret share $[g^{\alpha_j}]$ instead of a multiplicative share $g^{[\alpha_j]}$.

Limitation: This optimization only works for *verifiable* DPFs [14] (also known as *extractable* DPFs [15]) due to technical reasons in the security proof.

Optimized PACLs for Verifiable DPF

A more efficient PACL construction in a field

To select additive shares, move the inner-product "out of the exponent":

Group operations:

exponentiation + multiply

$$g^{\langle(\alpha_1,\cdots,\alpha_N),[\mathrm{e_j}]\rangle} = \prod_{k=1}^{N} \left(g^{\alpha_k}\right)^{[f_j(k)]}$$

$$= g^{[0]\alpha_1} \cdots g^{[1]\alpha_j} \cdots g^{[0]\alpha_N}$$

$$= g^{[\alpha_j]}$$

Field operations:

multiply + add

$$\langle(g_1^{\alpha},\cdots,g_N^{\alpha}),[\mathrm{e_j}]\rangle = \sum_{k=1}^{N} g^{\alpha_k}[f_j(k)]$$

$$= [0]g^{\alpha_1} + \cdots [1]g^{\alpha_j} + \cdots [0]g^{\alpha_N}$$

$$= [g^{\alpha_j}]$$

Optimized PACLs for Verifiable DPFs

How can the dealer prove knowledge of α over $[g^\alpha]$?

- Schnorr proof [16]: Prove knowledge of α for g^α, without revealing α ✗
- SPoSS proof (new): Prove knowledge of α for $[g^\alpha]$, without revealing α or g^α ✓

Schnorr Proof over Secret Shares

Schnorr Proof over Secret Shares (SPoSS)

Fix values $\mathbb{F}, g, [g^x]_1, \ldots, [g^x]_p$, where $\mathbb{F}$ is a finite field, and g is the generator for a cyclic group $\mathbb{G}$. SPoSS allows a prover to convince verifiers that it knows the value of x, *without* revealing to either verifier the value of x or g^x.

Prove$(x) \rightarrow ([\pi]_1, \ldots, [\pi]_p)$

- Given x, the prover outputs a proof share for each verifier.

Audit$([\pi]_i, [g^x]_i) \rightarrow \tau_i$

- Takes as input a proof share and share $[g^x]_i$, and outputs an audit token τ_i.

Verify$(\tau_1, \ldots, \tau_p) \rightarrow$ yes/no

- Accepts or rejects the proof based on the audit tokens from all verifiers.

Schnorr Proof over Secret Shares

Construction for $\mathbb{Z}_p^*$ in the random oracle model

Setup: Verifiers hold $[y] = [g^x]$ and the prover has x.

Step 1: The dealer sends secret shares $[x]$ as the proof to all verifiers.

Step 2: Each verifier computes *multiplicative* share of g^x by computing $g^{[x]_i}$.

Step 3: Verifiers derive *additive* secret shares $[g^{[x]}]$ of their *multiplicative* share $g^{[x]}$.

Step 4: The verifiers do a lightweight MPC to compute:
$[\prod_{i=1}^{p} g^{[x]_i}]_i = [g^{\sum_{i=1}^{p}[x]_i}] = [g^x]$ (additive secret shares of g^x).

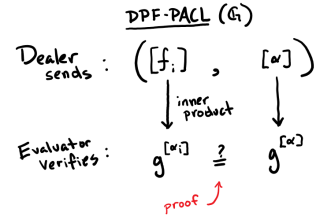
Step 5: The verifiers check that $\sum_{i=1}^{p}([g^x]_i - [y]_i) = 0$.

To fit the SPoSS definition, Step 4 is replaced with a "prover-assisted" computation which eliminates all interaction between verifiers (using a random oracle) using [17].

Optimized PACLs for Verifiable DPFs

Sketch of the VDPF-PACL construction:

1. Dealer prove knowledge of α over the additive shares $[g^{\alpha}]$ using SPoSS.
2. Using $[f_j]$, evaluators "select" additive secret share $[g^{\alpha}]$ by computing the inner product.
3. Evaluators check the SPoSS proof using $[g^{\alpha}]$.



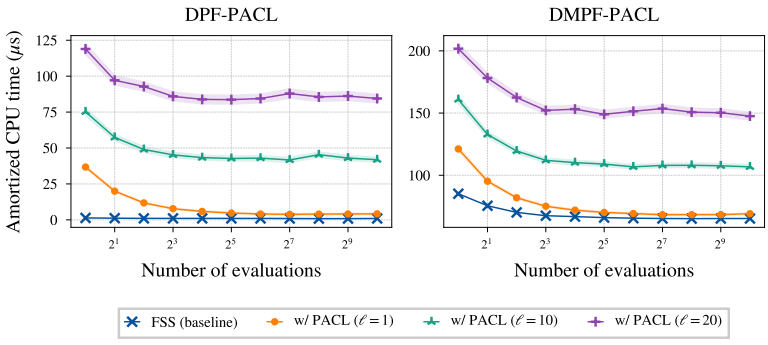
More details in the paper!

- We model access control requirements under different threat models
  - e.g., when $\mathbb{G} = \mathbb{Z}_p$, we get PACLs with secret access policies.
  - e.g., when $\mathbb{G}$ is a Diffie-Hellman group, we get PACLs with public access policies.
- We construct more efficient PACLs and discuss several concrete optimizations.
- We construct "generic" PACLs for *any* FSS scheme from zero-knowledge proofs on distributed data [12].

Implementation

- Implement DPF-PACLs and VDPF-PACLs:
  - DPF-PACL instantiated $\mathbb{G}$ as an 256-bit elliptical curve (P256 from `go/elliptic`).
  - VDPF-PACL for public-key: instantiated $\mathbb{G}$ as $\mathbb{Z}_p^*$ with a 3072-bit safe prime p.
- 4,500 lines of code written in Go and C (open source).
- Evaluated on Amazon Elastic Cloud Compute with 16 vCPUs and 32 GiB of RAM.
- **All experiments evaluated on a single core.**
- 10 trials per experiment, with averages reported.

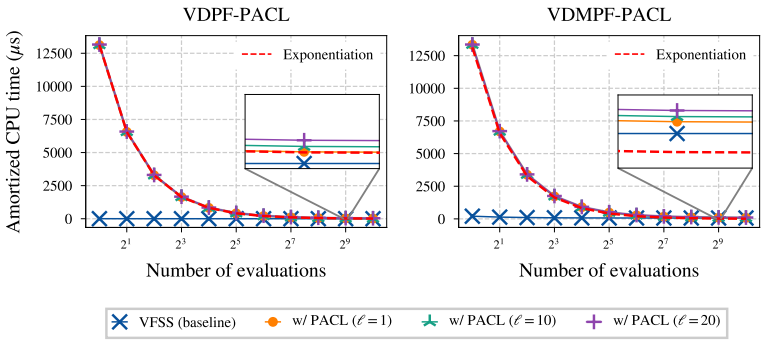
Evaluation of DPF-PACLs



Note: ℓ represents the number of access keys associated with each function.

Takeaway: DPF-PACLs add a noticeable concrete overhead when amortized over many evaluations of f (i.e., when f is evaluated on > 128 different inputs).

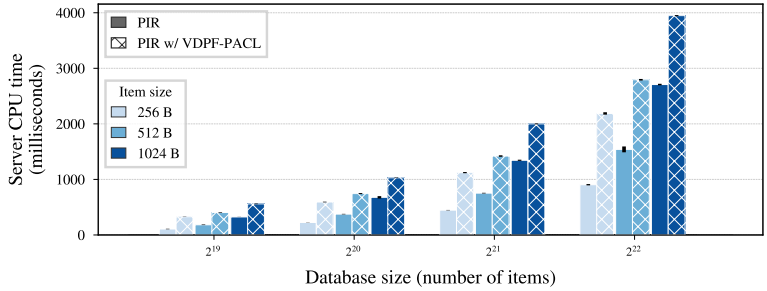
Evaluation of VDPF-PACLs



Note: ℓ represents the number of access keys associated with each function.

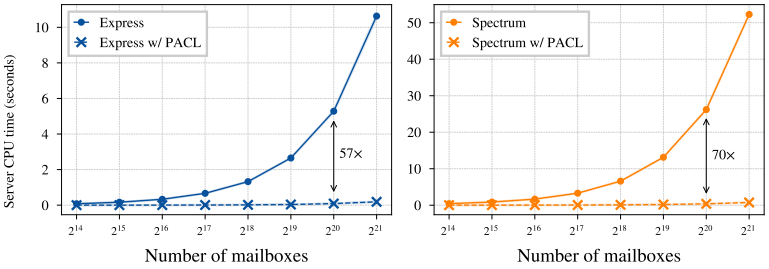
Takeaway: VDPF-PACLs add an asymptotically small overhead when amortized over many evaluations of f (i.e., when f is evaluated on > 128 different inputs).

Application I: PIR with Access Control



- 1.5–3× overhead to introduce access control for PIR (up to 1 KiB records).
- Overhead diminishes as the *size of the records* in the database increases.
- Overhead stays the same change when the number of records increases.

Application II: Anonymous Communication



- Anonymous communication systems (Express [5] and Spectrum [7]) can replace ad-hoc access control with PACLs for better performance with many mailboxes.
- PACLs provide 50–70× reduction in computational overhead for the verifiers.[1]

[1] When the number of mailboxes is approximately 1 million.

Application III: Anonymous Authentication

Anonymous authentication

- Verifiers determine if a given user has a valid account without learning which account was used for authentication.
- One-to-one mapping between DPF index j and user's account: $j = \texttt{user\_id}$ and access key $\alpha = \texttt{password}$.
- Authentication time grows linearly in the total number of accounts.

Number of accounts:	250K	500K	1M	2M
Authentication time:	95 ms	190 ms	385 ms	770 ms

Questions?

Implementation: github.com/sachaservan/pacl.

ePrint: ia.cr/2022/1707.

Contact me: 3s@mit.edu.

[1] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function secret sharing". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2015, pp. 337–367.

[2] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function secret sharing: Improvements and extensions". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 1292–1303.

[3] Emma Dauterman et al. "DORY: An encrypted search system with distributed trust". In: *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*. 2020, pp. 1101–1119.

[4] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. "Riposte: An anonymous messaging system handling millions of users". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 321–338.

[5] Saba Eskandarian et al. "Express: Lowering the cost of metadata-hiding communication with cryptographic privacy". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 1775–1792.

[6] Adithya Vadapalli, Kyle Storrier, and Ryan Henry. "Sabre: Sender-anonymous messaging with fast audits". In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2022, pp. 1953–1970.

[7] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. "Spectrum: High-bandwidth Anonymous Broadcast". In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 2022, pp. 229–248.

[8] Elette Boyle et al. "Efficient pseudorandom correlation generators: Silent OT extension and more". In: *Annual International Cryptology Conference*. Springer. 2019, pp. 489–518.

[9] Trinabh Gupta et al. "Scalable and private media consumption with Popcorn". In: *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. 2016, pp. 91–107.

[10] Kurt Thomas et al. "Protecting accounts from credential stuffing with password breach alerting.". In: *USENIX Security Symposium*. 2019, pp. 1556–1571.

[11] Ryan Henry, Femi Olumofin, and Ian Goldberg. "Practical PIR for electronic commerce". In: *Proceedings of the 18th ACM conference on Computer and communications security*. 2011, pp. 677–690.

[12] Henry Corrigan-Gibbs and Dan Boneh. "Prio: Private, robust, and scalable computation of aggregate statistics". In: *14th USENIX symposium on networked systems design and implementation (NSDI 17)*. 2017, pp. 259–282.

[13] Whitfield Diffie and Martin Hellman. "New Directions in Cryptography". In: *Ideas That Created the Future*. The MIT Press, Feb. 1976, pp. 421–440. DOI: 10.7551/mitpress/12274.003.0044. URL: https://doi.org/10.7551/mitpress/12274.003.0044.

[14] Leo de Castro and Anitgoni Polychroniadou. "Lightweight, Maliciously Secure Verifiable Function Secret Sharing". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2022, pp. 150–179.

[15] Dan Boneh et al. "Lightweight techniques for private heavy hitters". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 762–776.

[16] Claus-Peter Schnorr. "Efficient identification and signatures for smart cards". In: *Conference on the Theory and Application of Cryptology*. Springer. 1989, pp. 239–252.

[17] Dan Boneh et al. "Zero-knowledge proofs on secret-shared data via fully linear PCPs". In: *Annual International Cryptology Conference*. Springer. 2019, pp. 67–97.