NTNU | Norwegian University of Science and Technology

PRIO: PRIVATE, ROBUST, AND SCALABLE COMPUTATION OF AGGREGATE STATISTICS

Authors: Henry Corrigan-Gibbs and Dan Boneh

Presented by: Caroline Sandsbråten

December 7, 2023

Contents

Contents

Key Contributions

Key Contributions

- Introduction of Secret-Shared Non-Interactive Proofs (SNIPs).

Key Contributions

- Introduction of Secret-Shared Non-Interactive Proofs (SNIPs).
- Presentation of affine-aggregatable encodings, unifying many data-encoding techniques for private aggregation.

Key Contributions

- Introduction of Secret-Shared Non-Interactive Proofs (SNIPs).
- Presentation of affine-aggregatable encodings, unifying many data-encoding techniques for private aggregation.
- Demonstration of combining these encodings with SNIPs to ensure robustness and privacy in large-scale data collection.

Contents

The Problem

The Problem

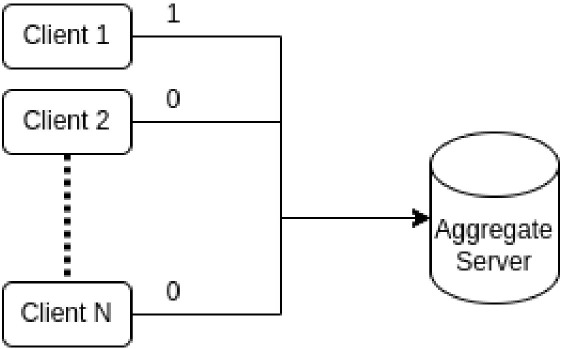
- Collecting aggregate statistics pose privacy risks

The Problem

- Collecting aggregate statistics pose privacy risks
- This collection are vulnerable to data manipulation by malicious clients

The Problem

- Collecting aggregate statistics pose privacy risks
- This collection are vulnerable to data manipulation by malicious clients



Prio’s Solution

Overview

Prio’s Solution

Overview

- Manages to achieve: privacy and robustness against faulty and malicious clients

Prio's Solution

Overview

- Manages to achieve: privacy and robustness against faulty and malicious clients
- Prio is also scalable

Prio's Solution

Overview

- Manages to achieve: privacy and robustness against faulty and malicious clients
- Prio is also scalable
- Achieves this with the use of a novel technique: secret-shared non-interactive proofs (SNIPs)

Applications for Prio

Applications for Prio

- Machine Learning

Applications for Prio

- Machine Learning
- Health/fitness tracking

Applications for Prio

- Machine Learning
- Health/fitness tracking
- Web browsing data collection

Applications for Prio

- Machine Learning
- Health/fitness tracking
- Web browsing data collection
- Essentially in every scenario where aggregate data is valuable, but user privacy is just as important

Efficiency of Prio

Efficiency of Prio

- Prio has a minimal slowdown compared to non-private systems

Efficiency of Prio

- Prio has a minimal slowdown compared to non-private systems
- It also has a significant performance advantage over systems using conventional ZK approaches

Efficiency of Prio

Comparison

Type	Prio	Prio (non-robust)	NIZK
Client-side cost	50x slowdown	N/A	50-100x vs. (compared to Prio)
Server-side cost	1-2x slowdown	5-15x slowdown	267x slowdown
Overall system	5.7x slowdown	N/A	N/A

Efficiency of Prio

Comparison

Type	Prio	Prio (non-robust)	NIZK
Client-side cost	50x slowdown	N/A	50-100x vs. (compared to Prio)
Server-side cost	1-2x slowdown	5-15x slowdown	267x slowdown
Overall system	5.7x slowdown	N/A	N/A

Experiment

Case: Privately collect responses to a survey with 434 true/false questions.

Efficiency of Prio

Comparison

Type	Prio	Prio (non-robust)	NIZK
Client-side cost	50x slowdown	N/A	50-100x vs. (compared to Prio)
Server-side cost	1-2x slowdown	5-15x slowdown	267x slowdown
Overall system	5.7x slowdown	N/A	N/A

Experiment

Case: Privately collect responses to a survey with 434 true/false questions.

Results:

Efficiency of Prio

Comparison

Type	Prio	Prio (non-robust)	NIZK
Client-side cost	50x slowdown	N/A	50-100x vs. (compared to Prio)
Server-side cost	1-2x slowdown	5-15x slowdown	267x slowdown
Overall system	5.7x slowdown	N/A	N/A

Experiment

Case: Privately collect responses to a survey with 434 true/false questions.

Results:

- Client: 26ms computation

Efficiency of Prio

Comparison

Type	Prio	Prio (non-robust)	NIZK
Client-side cost	50x slowdown	N/A	50-100x vs. (compared to Prio)
Server-side cost	1-2x slowdown	5-15x slowdown	267x slowdown
Overall system	5.7x slowdown	N/A	N/A

Experiment

Case: Privately collect responses to a survey with 434 true/false questions.

Results:

- Client: 26ms computation
- Servers: 2ms computation per submission

Contents

What are SNIPs?

What are SNIPs?

- Cryptographic tools that allow a client to prove to a set of servers that a submitted value is correct and within expected parameters, without revealing the actual value.

What are SNIPs?

- Cryptographic tools that allow a client to prove to a set of servers that a submitted value is correct and within expected parameters, without revealing the actual value.
- Designed to work in a distributed setting where multiple servers collaboratively verify the correctness of client submissions.

This sounds very similar to NIZKs

This sounds very similar to NIZKs

So what is the difference?

This sounds very similar to NIZKs

So what is the difference?

- SNIPs are tailored for efficiency in client/server settings.

This sounds very similar to NIZKs

So what is the difference?

- SNIPs are tailored for efficiency in client/server settings.
- SNIPs are specifically designed for data aggregation settings, while NIZKs have a broader range of applications.

This sounds very similar to NIZKs

So what is the difference?

- SNIPs are tailored for efficiency in client/server settings.
- SNIPs are specifically designed for data aggregation settings, while NIZKs have a broader range of applications.
- SNIPs generally use a combination of polynomial identity tests and secret sharing, and are usualyl focused on information theoretic security.

SNIPs in Prio (simplified)

SNIPs in Prio (simplified)

Setup

SNIPs in Prio (simplified)

Setup

- Let M be the number of multiplication gates for the circuit $Valid$

SNIPs in Prio (simplified)

Setup

- Let M be the number of multiplication gates for the circuit $Valid$
- $2M << |\mathbb{F}|$

SNIPs in Prio (simplified)

Setup

- Let M be the number of multiplication gates for the circuit $Valid$
- $2M << |\mathbb{F}|$

Client Evaluation

SNIPs in Prio (simplified)

Setup

- Let M be the number of multiplication gates for the circuit $Valid$
- $2M << |\mathbb{F}|$

Client Evaluation

- Client evaluates $Valid(x)$ on input x to know the value of every wire in the circuit

SNIPs in Prio (simplified)

Setup

- Let M be the number of multiplication gates for the circuit $Valid$
- $2M << |\mathbb{F}|$

Client Evaluation

- Client evaluates $Valid(x)$ on input x to know the value of every wire in the circuit
- Client uses wires to construct polynomials f, g, h which encodes values on input and output wires of the M gates.

SNIPs in Prio (simplified)

Polynomial Construction

SNIPs in Prio (simplified)

Polynomial Construction

- Let u_t, v_t be the input wires for the t-th multiplication gate

SNIPs in Prio (simplified)

Polynomial Construction

- Let u_t, v_t be the input wires for the t-th multiplication gate
- define f, g as the lowest degree possible polynomials s.t. $f(t) = u_t, g(t) = v_t$

SNIPs in Prio (simplified)

Polynomial Construction

- Let u_t, v_t be the input wires for the t-th multiplication gate
- define f, g as the lowest degree possible polynomials s.t. $f(t) = u_t, g(t) = v_t$
- Define $h = f \cdot g$

SNIPs in Prio (simplified)

Polynomial Construction

- Let u_t, v_t be the input wires for the t-th multiplication gate
- define f, g as the lowest degree possible polynomials s.t. $f(t) = u_t, g(t) = v_t$
- Define $h = f \cdot g$
- Then $\deg(f) \leq M - 1, \deg(f) \leq M - 1, \deg(h) \leq 2M - 2$

SNIPs in Prio (simplified)

Polynomial Construction

- Let u_t, v_t be the input wires for the t-th multiplication gate
- define f, g as the lowest degree possible polynomials s.t. $f(t) = u_t, g(t) = v_t$
- Define $h = f \cdot g$
- Then $\deg(f) \leq M - 1, \deg(f) \leq M - 1, \deg(h) \leq 2M - 2$
- Since $h(t) = f(t) \cdot g(t)$, then $h(t)$ equals the output wire of the t-th gate.

SNIPs in Prio (simplified)

Client's Computation

SNIPs in Prio (simplified)

Client's Computation

- Polynomial interpolation and multiplication to compute $Valid(x)$

SNIPs in Prio (simplified)

Client's Computation

- Polynomial interpolation and multiplication to compute $Valid(x)$
- The client then splits the coefficients of h in s parts and sends the i-th share to the i-th server

SNIPs in Prio (simplified)

Client's Computation

- Polynomial interpolation and multiplication to compute $Valid(x)$
- The client then splits the coefficients of h in s parts and sends the i-th share to the i-th server
- This way, only one honest server is needed to achieve information theoretic security (they each also only get x_i)

SNIPs in Prio (simplified)

Consistency Checking

- Each server holds share x_i and h_i.

SNIPs in Prio (simplified)

Consistency Checking

- Each server holds share x_i and h_i.
- From this, the servers produce shares f_i, g_i without communicating with each other.

SNIPs in Prio (simplified)

Consistency Checking

- Each server holds share x_i and h_i.
- From this, the servers produce shares f_i, g_i without communicating with each other.
- If clients and servers all act honestly, then correctness is obvious

SNIPs in Prio (simplified)

Consistency Checking

- Assume a malicious client sends $\hat{h}$ s.t. for some $t \in [M], \hat{h}(t) \neq h(t)$

SNIPs in Prio (simplified)

Consistency Checking

- Assume a malicious client sends $\hat{h}$ s.t. for some $t \in [M], \hat{h}(t) \neq h(t)$
- Then the servers reconstructs shares of $\hat{f}, \hat{g}$ that might not equal f, g.

SNIPs in Prio (simplified)

Consistency Checking

- Assume a malicious client sends $\hat{h}$ s.t. for some $t \in [M], \hat{h}(t) \neq h(t)$
- Then the servers reconstructs shares of $\hat{f}, \hat{g}$ that might not equal f, g.
- Then, with certainty: $\hat{h} \neq \hat{f} \cdot \hat{g}$

SNIPs in Prio (simplified)

Consistency Checking

- Assume a malicious client sends $\hat{h}$ s.t. for some $t \in [M], \hat{h}(t) \neq h(t)$
- Then the servers reconstructs shares of $\hat{f}, \hat{g}$ that might not equal f, g.
- Then, with certainty: $\hat{h} \neq \hat{f} \cdot \hat{g}$
- Since $\hat{h}(t_0) \neq h(t_0) = f(t_0) \cdot g(t_0) = \hat{f}(t_0) \cdot \hat{g}(t_0)$, then $\hat{h} \neq \hat{f} \cdot \hat{g}$ for the least t_0 s.t. $\hat{h}(t_0) \neq h(t_0)$

SNIPs in Prio (simplified)

Polynomial Identity Test

SNIPs in Prio (simplified)

Polynomial Identity Test

- The servers to check whether $f \cdot g = h$ holds by executing the Schwartz-Zippel randomized polynomial identity test. The principle of this test is:

SNIPs in Prio (simplified)

Polynomial Identity Test

- The servers to check whether $f \cdot g = h$ holds by executing the Schwartz-Zippel randomized polynomial identity test. The principle of this test is:
- If $f \cdot g \neq h$ then $f \cdot g - h$ is a non-zero polynomial with $\deg \leq 2M - 2$.

SNIPs in Prio (simplified)

Polynomial Identity Test

- The servers to check whether $f \cdot g = h$ holds by executing the Schwartz-Zippel randomized polynomial identity test. The principle of this test is:
- If $f \cdot g \neq h$ then $f \cdot g - h$ is a non-zero polynomial with $\deg \leq 2M - 2$.
- One server chose a random $r \in \mathbb{F}$

SNIPs in Prio (simplified)

Polynomial Identity Test

- The servers to check whether $f \cdot g = h$ holds by executing the Schwartz-Zippel randomized polynomial identity test. The principle of this test is:
- If $f \cdot g \neq h$ then $f \cdot g - h$ is a non-zero polynomial with $\deg \leq 2M - 2$.
- One server chose a random $r \in \mathbb{F}$
- Each server evaluates their share by calculating $\sigma_i = f_i(r) \cdot g_i(r) - h_i(r)$

SNIPs in Prio (simplified)

Polynomial Identity Test

- The servers to check whether $f \cdot g = h$ holds by executing the Schwartz-Zippel randomized polynomial identity test. The principle of this test is:
- If $f \cdot g \neq h$ then $f \cdot g - h$ is a non-zero polynomial with $\deg \leq 2M - 2$.
- One server chose a random $r \in \mathbb{F}$
- Each server evaluates their share by calculating $\sigma_i = f_i(r) \cdot g_i(r) - h_i(r)$
- Servers publish σ_i and ensure $\sum_i \sigma_i = 0$, if not reject

SNIPs in Prio (simplified)

Multiplication of Shares

SNIPs in Prio (simplified)

Multiplication of Shares

- Without leaking information to each other, multiply $f_i \cdot g_i$'s

SNIPs in Prio (simplified)

Multiplication of Shares

- Without leaking information to each other, multiply $f_i \cdot g_i$'s
- From a trusted dealer, each server receives one-time-use shares $(a_i, b_i, c_i) \in \mathbb{F}^3$ s.t. $a \cdot b = c \in \mathbb{F}$, then using the Beaver MPC multiplication protocol.

SNIPs in Prio (simplified)

Multiplication of Shares

- Without leaking information to each other, multiply $f_i \cdot g_i$'s
- From a trusted dealer, each server receives one-time-use shares $(a_i, b_i, c_i) \in \mathbb{F}^3$ s.t. $a \cdot b = c \in \mathbb{F}$, then using the Beaver MPC multiplication protocol.
- This is fast (each server needs to broadcast a single message).

SNIPs in Prio (simplified)

Multiplication of Shares

- Without leaking information to each other, multiply $f_i \cdot g_i$'s
- From a trusted dealer, each server receives one-time-use shares $(a_i, b_i, c_i) \in \mathbb{F}^3$ s.t. $a \cdot b = c \in \mathbb{F}$, then using the Beaver MPC multiplication protocol.
- This is fast (each server needs to broadcast a single message).
- In this setting, the client generates a, b, c and splits into shares a_i, b_i, c_i for each of the servers.

SNIPs in Prio (simplified)

Multiplication of Shares

- Without leaking information to each other, multiply $f_i \cdot g_i$'s
- From a trusted dealer, each server receives one-time-use shares $(a_i, b_i, c_i) \in \mathbb{F}^3$ s.t. $a \cdot b = c \in \mathbb{F}$, then using the Beaver MPC multiplication protocol.
- This is fast (each server needs to broadcast a single message).
- In this setting, the client generates a, b, c and splits into shares a_i, b_i, c_i for each of the servers.
- This saves computation time/resources.

SNIPs in Prio (simplified)

Beavers MPC Protocol

SNIPs in Prio (simplified)

Beavers MPC Protocol

- Each server holds share x_i of input vector x.

SNIPs in Prio (simplified)

Beavers MPC Protocol

- Each server holds share x_i of input vector x.
- Servers wants to compute $C(x)$ for some arithmetic circuit C.

SNIPs in Prio (simplified)

Beavers MPC Protocol

- Each server holds share x_i of input vector x.
- Servers wants to compute $C(x)$ for some arithmetic circuit C.
- For each step, the servers wants to compute $f \cdot g$, each holding f_i, g_i

SNIPs in Prio (simplified)

Beavers MPC Protocol

- Each server holds share x_i of input vector x.
- Servers wants to compute $C(x)$ for some arithmetic circuit C.
- For each step, the servers wants to compute $f \cdot g$, each holding f_i, g_i
- Using the triples (a_i, b_i, c_i) and f_i, g_i, to compute:

SNIPs in Prio (simplified)

Beavers MPC Protocol

- Each server holds share x_i of input vector x.
- Servers wants to compute $C(x)$ for some arithmetic circuit C.
- For each step, the servers wants to compute $f \cdot g$, each holding f_i, g_i
- Using the triples (a_i, b_i, c_i) and f_i, g_i, to compute:
- $d_i = f_i(\tau) - a_i, e_i = g_i(\tau) - b_i$ where τ is the last multiplication gate of the circuit C.

SNIPs in Prio (simplified)

Beavers MPC Protocol

- Each server holds share x_i of input vector x.
- Servers wants to compute $C(x)$ for some arithmetic circuit C.
- For each step, the servers wants to compute $f \cdot g$, each holding f_i, g_i
- Using the triples (a_i, b_i, c_i) and f_i, g_i, to compute:
- $d_i = f_i(\tau) - a_i, e_i = g_i(\tau) - b_i$ where τ is the last multiplication gate of the circuit C.
- Each server broadcasts d_i, e_i

SNIPs in Prio (simplified)

Beavers MPC Protocol

- Each server holds share x_i of input vector x.
- Servers wants to compute $C(x)$ for some arithmetic circuit C.
- For each step, the servers wants to compute $f \cdot g$, each holding f_i, g_i
- Using the triples (a_i, b_i, c_i) and f_i, g_i, to compute:
- $d_i = f_i(\tau) - a_i, e_i = g_i(\tau) - b_i$ where τ is the last multiplication gate of the circuit C.
- Each server broadcasts d_i, e_i
- Each server calculates $\rho_i = de/s + db_i + ea_i + c_i$.

SNIPs in Prio (simplified)

Beaver MPC Protocol Correctness

$$\begin{aligned}
\sum_i \rho_i &= \sum_i (de/s + db_i + ea_i + c_i) \\
&= de + db + ea + c \\
&= (f(\tau) - a)(g(\tau) - b) + (f(\tau) - a)b + (g(\tau) - b)a + c \\
&= f(\tau)g(\tau) - ag(\tau) + ag(\tau) - ab + c \\
&= f(\tau)g(\tau) - ab + c \\
&= f(\tau)g(\tau) && = h(\tau)
\end{aligned}$$

SNIPs in Prio (simplified)

Output Verification

SNIPs in Prio (simplified)

Output Verification

- Servers publish output shares after the circuit

SNIPs in Prio (simplified)

Output Verification

- Servers publish output shares after the circuit
- Sum up shares to confirm $Valid(x) = 1$

SNIPs in Prio (simplified)

Security

SNIPs in Prio (simplified)

Security

- Correctness follows construction.

SNIPs in Prio (simplified)

Security

- Correctness follows construction.
- A malicious client must cheat the polynomial identity test with probability $(2M-2)/|\mathbb{F}|$.

SNIPs in Prio (simplified)

Security

- Correctness follows construction.
- A malicious client must cheat the polynomial identity test with probability $(2M - 2)/|\mathbb{F}|$.
- Completeness nor soundness holds in the presence of malicious servers.

SNIPs in Prio (simplified)

Security

- Correctness follows construction.
- A malicious client must cheat the polynomial identity test with probability $(2M-2)/|\mathbb{F}|$.
- Completeness nor soundness holds in the presence of malicious servers.
- Malicious servers can mount selective DoS attacks against clients

SNIPs in Prio (simplified)

Security

- Correctness follows construction.
- A malicious client must cheat the polynomial identity test with probability $(2M-2)/|\mathbb{F}|$.
- Completeness nor soundness holds in the presence of malicious servers.
- Malicious servers can mount selective DoS attacks against clients
- As long as at least one server is honest, dishonest servers learn nothing about the clients data.

SNIPs in Prio (simplified)

Efficiency

SNIPs in Prio (simplified)

Efficiency

- Server-to-server communication cost grows neither with complexity of the verification circuit nor with the size of x.

SNIPs in Prio (simplified)

Efficiency

- Server-to-server communication cost grows neither with complexity of the verification circuit nor with the size of x.

- Computation cost for each server is not much more than to evaluate the $Valid$ circuit.

SNIPs in Prio (simplified)

Efficiency

- Server-to-server communication cost grows neither with complexity of the verification circuit nor with the size of x.
- Computation cost for each server is not much more than to evaluate the $Valid$ circuit.
- Client-to-server communication cost grows linearly with the size of the $Valid$ circuit.

SNIPs in Prio (simplified)

Efficiency

- Server-to-server communication cost grows neither with complexity of the verification circuit nor with the size of x.
- Computation cost for each server is not much more than to evaluate the $Valid$ circuit.
- Client-to-server communication cost grows linearly with the size of the $Valid$ circuit.
- The authors note an interesting challenge to try to reduce the communication cost without needing expensive asymm. cryptography.

Contents

Simple Prio

Simple Prio

Setup

Simple Prio

Setup

- Each client holds a one-bit integer x_i.

Simple Prio

Setup

- Each client holds a one-bit integer x_i.
- The servers wants to compute $\sum_i x_i$.

Simple Prio

Setup

- Each client holds a one-bit integer x_i.
- The servers wants to compute $\sum_i x_i$.
- We have s servers.

Simple Prio

Upload

Simple Prio

Upload

- Each client i splits its private value x_i into s shares

Simple Prio

Upload

- Each client i splits its private value x_i into s shares
- Then sends this share $[x_i]_j$, $j \in [s]$ to each corresponding server j.

Simple Prio

Aggregate

Simple Prio

Aggregate

- Each server j holds an accumulator value $A_j \in \mathbb{F}_p$

Simple Prio

Aggregate

- Each server j holds an accumulator value $A_j \in \mathbb{F}_p$
- And updates this $A_j \leftarrow A_j + [x_i]_j \in \mathbb{F}_p$ each time it receives a new value.

Simple Prio

Publish

Simple Prio

Publish

- Once the servers have received all clients shares, they publish A_j.

Simple Prio

Publish

- Once the servers have received all clients shares, they publish A_j.
- Computing $\sum_j A_j \in \mathbb{F}_p$ yields $\sum_i x_i$.

Affine-Aggregatable Encodings (AFEs)

Affine-Aggregatable Encodings (AFEs)

Our Setting

Affine-Aggregatable Encodings (AFEs)

Our Setting

- Each client i holds a value $x_i \in D$, where D is some set of data values.

Affine-Aggregatable Encodings (AFEs)

Our Setting

- Each client i holds a value $x_i \in D$, where D is some set of data values.
- The servers holds an aggregation function $f : D^n \to A$.

Affine-Aggregatable Encodings (AFEs)

Our Setting

- Each client i holds a value $x_i \in D$, where D is some set of data values.
- The servers holds an aggregation function $f : D^n \to A$.
- The servers goal is to evaluate $f(x_1, \ldots, x_n)$ without learning $x_i \forall i$.

Affine-Aggregatable Encodings (AFEs)

What do AFEs do?

Affine-Aggregatable Encodings (AFEs)

What do AFEs do?

- Gives an efficient way to encode data values x_i s.t. it is possible to compute $f(x_1, \ldots, x_n)$ given only the sum of the encodings of $x_1, \ldots, x_n$.

Affine-Aggregatable Encodings (AFEs)

What do AFEs do?

- Gives an efficient way to encode data values x_i s.t. it is possible to compute $f(x_1, \ldots, x_n)$ given only the sum of the encodings of $x_1, \ldots, x_n$.

- An AFE have 3 (efficient) algorithms:

Affine-Aggregatable Encodings (AFEs)

What do AFEs do?

- Gives an efficient way to encode data values x_i s.t. it is possible to compute $f(x_1, \ldots, x_n)$ given only the sum of the encodings of $x_1, \ldots, x_n$.

- An AFE have 3 (efficient) algorithms:

- $\mathsf{Encode}(x)$: maps an input $x \in D$ to its encoding in $\mathbb{F}^k$

Affine-Aggregatable Encodings (AFEs)

What do AFEs do?

- Gives an efficient way to encode data values x_i s.t. it is possible to compute $f(x_1, \ldots, x_n)$ given only the sum of the encodings of $x_1, \ldots, x_n$.

- An AFE have 3 (efficient) algorithms:

- $\mathsf{Encode}(x)$: maps an input $x \in D$ to its encoding in $\mathbb{F}^k$

- $\mathsf{Valid}(y)$: returns true iff $y \in \mathbb{F}^k$ is a valid encoding of some data item in D.

Affine-Aggregatable Encodings (AFEs)

What do AFEs do?

- Gives an efficient way to encode data values x_i s.t. it is possible to compute $f(x_1, \ldots, x_n)$ given only the sum of the encodings of $x_1, \ldots, x_n$.
- An AFE have 3 (efficient) algorithms:
- $\mathsf{Encode}(x)$: maps an input $x \in D$ to its encoding in $\mathbb{F}^k$
- $\mathsf{Valid}(y)$: returns true iff $y \in \mathbb{F}^k$ is a valid encoding of some data item in D.
- $\mathsf{Decode}(\sigma)$: Takes $\sigma = \sum_{i=1}^{n} Trunc_{k'}(\mathsf{Encode}(x_i)) \in \mathbb{F}^k$ and outputs $f(x_1, \ldots, x_n)$

Contents

Contents

Small Fun Facts

Small Fun Facts

Implementation

Small Fun Facts

Implementation

- The prototype is only 5700 lines of Go and 620 lines of C (for FLINT)

Small Fun Facts

Implementation

- The prototype is only 5700 lines of Go and 620 lines of C (for FLINT)
- Code is available on `https://crypto.stanford.edu/prio/`.

Questions?