COM-506 Report - Prio: Private, Robust, and Scalable Computation of Aggregate Statistics

February 2025

Charrin Octave
SCIPER - 404884

Stentella Gabriele
SCIPER - 404913

Villa Federico
SCIPER - 386986

Abstract

The paper [2] introduces Prio, a system designed to privately and robustly compute aggregate statistics from client data without sacrificing scalability; it utilizes "Secret-shared Non-Interactive Proofs (SNIPs)" and "Affine-aggregatable Encodings (AFEs)" to achieve its goals. Prio can be practically deployed in a variety of scenarios, including gathering browser statistics and training machine learning models.

1 Introduction

Nowadays, it is common for electronics of everyday use to be connected to cloud services which aim to store telemetry data from the users of the devices or services, and learn aggregate statistics. Since a centralized storage of private data represents a risk for both privacy and security, many of those services have implemented a "randomized response" mechanism. This approach allows a client that has to communicate a bit representing a private information, to flip it with probability $p < 0.5$, and over a large number of collected bits it is still possible to compute useful statistics. However, with p too small every bit still leaks information, while with p too high the accuracy decreases so much that the aggregation becomes useless. If the clients instead send encrypted values, the privacy guarantees are stronger, but the clients acquire the capability of influencing the aggregation by an arbitrary value. Even if it is possible to use zero-knowledge proofs as a protection, scaling this solution becomes impractical for end users. Prio makes use of SNIPs to allow the clients to submit a proof of correctness together with its private data, so that the server can validate the submission without seeing data in the clear.

2 System goals

Since Prio needs authenticated and encrypted channels between each pair of protocol participants, the authors assume the existence of a PKI, but they don't assume a synchronous network and handle the case where an attacker controls a large number of clients, has full access to the network and controls all servers except for one. Prio aims to achieve three security properties: Anonymity, privacy, and robustness.

- Anonymity: Prio ensures that an adversary cannot link specific honest clients to their submitted data, even if the adversary controls all but one server and all other clients. The adversary may see the set of honest clients' inputs but cannot determine which client submitted which value.
- Privacy: Prio guarantees that an adversary controlling all but one server and any number of clients learns nothing beyond the output of the aggregation function. The adversary's view of the protocol can be simulated using only this output.
- Robustness: Prio ensures that malicious clients can only affect the system's output by misreporting their own data, not by otherwise corrupting the result. However, it does not protect against adversarial servers.

Providing robustness against faulty servers weakens privacy because if the system tolerates k faulty servers, then $s - k$ dishonest servers can still compute individual client submissions, compromising confidentiality. Specifically, dishonest servers could evaluate the aggregation function over a single submission. Additionally, enhancing robustness incurs performance costs, as it would require abandoning leader-based optimizations, where a designated "leader" server coordinates the processing of client submissions to improve efficiency.

3 A simple scheme

A simplified version of Prio enables privacy-preserving computation of the sum of clients' one-bit private values using secret-sharing. The scheme consists of three steps. Initially, each client splits its input value into shares, and submits them to the servers. Then, all the servers add the shares they receive to their internal accumulator. Finally, the servers publish their internal values, and every participant can compute the sum of the accumulators. While this scheme ensures privacy by preventing servers from learning individual inputs, it lacks robustness since a single malicious client can corrupt the output; Prio improves upon this by adding robustness against malicious clients and extending functionality to support various aggregation functions beyond sums.

4 Protecting correctness with SNIPs

A SNIP is a cryptographic tool that allows the server to check the inputs they receive are well-formed, without learning nothing about single inputs. The construction of these tools relies on a linear additive secret-sharing scheme over a finite

field $\mathbb{F}$. Moreover, it uses an arithmetic circuit Valid, composed of gates representing operations in $\mathbb{F}$. Given the number M of multiplication gates in Valid, the protocol works in a field such that $2M+2 \ll |\mathbb{F}|$. The client evaluates the circuit over its private data, so that she knows the input values u_t and v_t for each multiplication gate from 1 to M, while she samples i.i.d. from $\mathbb{F}$ the values of u_0 and v_0. Then, she defines the lowest-degree polynomials for which holds $f(t) = u_t, g(t) = v_t \forall t \in \{0,\ldots,M\}$, and $h = f \cdot g$. The client then submits the shares of the random values $[u_0]_i, [v_0]_i$[1], and the shares of the coefficients of h, sending only the ith share to server i. The servers, thanks to the linearity of the secret sharing scheme, can locally compute the shares of f and g. Each server indeed knows a share of the private input x, and the shares of all outputs of multiplication gates, so it is possible to derive all the other values. Then, the servers need a method to detect with high probability a cheating client: They use a randomized polynomial identity test, based on the fact that if some clients submit malicious values for which $\hat{h}(t) \neq \hat{f}(t) \cdot \hat{g}(t)$ for some t, then the servers can choose a random point r and evaluate $r \cdot (\hat{f}(r) \cdot \hat{g}(r) - \hat{h}(r))$. This polynomial has at most $2M+1$ zeros in the field, so the servers detect the cheating client with probability $\geq 1 - ((2M+1)/|\mathbb{F}|)$. To compute locally the shares of $r \cdot \hat{f}(r) \cdot \hat{g}(r)$, the servers use an adaptation of the MPC technique of Beaver [1]. For this computation, the clients generate a triple $a, b, c \in \mathbb{F}^3$ for which holds $a \cdot b = c$ and send shares of these values to the servers, hence the final SNIP proof is a tuple $\pi = (f(0), g(0), h, a, b, c)$. Thanks to the multiplication of the polynomial by a random value r, even if the malicious clients choose adversarially the triple of values $(a \cdot b = c + \alpha)$, the polynomial on which the test is performed remains a non-zero polynomial. Finally, each server publishes her share of $\sigma = r \cdot (\hat{f}(r) \cdot \hat{g}(r) - \hat{h}(r))$ and they accept the input data only if the sum of all the shares of σ is 0. In the final step of the protocol the servers only need to publish the shares of the output values: This, with the input shares $[x]_i$, allow to verify that $\mathsf{Valid}(x) = 1$.

5 Security Analysis of the SNIP

In this section, we analyze the security of the SNIP construction. For a SNIP to be useful in this context, it must satisfy three key properties: **Correctness**, **Soundness** and **Zero-knowledge**. Correctness ensures that if all parties behave honestly, the servers will accept a valid input x. This property holds by construction. Soundness and Zero-knowledge are more nuanced and are analyzed in the following subsections.

5.1 Soundness

In the context of this paper, soundness ensures that if all servers are honest and $\mathsf{Valid}(x) \neq 1$, then for any malicious clients, even ones running in super-polynomial time, the servers will reject x, with overwhelming probability.

[1]We use $[x]_i$ to denote the ith share of x: $x = \sum_i [x]_i$

We now prove that the SNIP construction is sound. This property is formalized through the following experiment, in which the adversary attempts to generate an input x and a SNIP proof $\pi = (f(0), g(0), h, a, b, c)$, such that $\mathsf{Valid}(x) \neq 1$ and yet the servers accept x as valid. Since the computation between multiplication gates in the circuits are affine transformations, we can, without loss of generality, assume that the output of the Valid circuit corresponds to the final multiplication gate's output wire. Let M denote the number of multiplication gates in Valid. The adversarial game proceeds as follows:

1. Run the adversary $\mathcal{A}$. For each server i, the adversary outputs a set of values: $[x]_i \in \mathbb{F}^L$, $([f(0)]_i, [g(0)]_i) \in \mathbb{F}^2$, $[h]_i \in \mathbb{F}_{2M}[X]$, and $([a]_i, [b]_i, [c]_i) \in \mathbb{F}^3$.
2. Choose r at random: $r \leftarrow^{\$} \in \mathbb{F}$. Each server computes their shares $[f]_i$ and $[g]_i$ as in the real protocol, and evaluates $[f(r)]_i$, $[r \cdot g(r)]_i$, $[r \cdot h(r)]_i$, and $[h(M)]_i$.
3. The servers compute $h(M) = \sum_i [h(M)]_i$, and $\sigma = r \cdot (f(r)g(r) - h(r)) + (c - ab)$, where we use additive secret sharing notation: $x = \sum_i [x]_i$. This computation follows Beaver's MPC protocol [1], assuming a supposedly valid multiplication triple (a, b, c). Note that since $\mathcal{A}$ is malicious, there is no guarantee that $ab = c$.
4. We say that the adversary wins the game if $h(M) = 1$, $\sigma = 0$, and $\mathsf{Valid}(x) \neq 1$.

The SNIP protocol is sound if, for all adversaries $\mathcal{A}$, including computationally unbounded ones, the probability of winning the game, over the uniform choice of r in $\mathbb{F}$, is bounded by: $\Pr[W] \leq (2M+1)/|\mathbb{F}|$.

First, in the case where $fg \neq h$, we can express σ as the evaluation of the polynomial $P(t) = t \cdot Q(t) + (c - ab)$, where $Q = fg - h$. Since Q is nonzero, P is also a nonzero polynomial (no matter how Q and (a, b, c) are related) of degree at most $2M+1$. Because the adversary $\mathcal{A}$ must generate (a, b, c) and Q before r is chosen, r and P are independent. Because of its degree, P has at most $2M+1$ zeros in $\mathbb{F}$. Thus, the probability that $P(r) = 0$, i.e., $\sigma = 0$ is at most $(2M+1)/|\mathbb{F}|$, which implies soundness.

Now consider the case where $fg = h$. We can prove by induction that $h(M) = \mathsf{Valid}(x)$ (Appendix D.1 of [2]), meaning the servers can compute the true value of Valid and detect when the adversary is trying to submit an invalid input value. Therefore, $\Pr[h(M) = 1 \text{ and } \mathsf{Valid}(x) \neq 1] = 0$, which also implies soundness in this case.

Since soundness requires bounding the probability of the adversary successfully forging an invalid proof, and we have shown that $\Pr[W] \leq (2M+1)/|\mathbb{F}|$, the SNIP presented in the paper is sound.

5.2 Zero-knowledge

For conciseness, we omit the formal proof of zero-knowledge and present only the core arguments (see Appendix D.2 of [2]

for the full proof).

To formally define zero-knowledge, we consider two experiments: 1) A "real-world" experiment, where the adversary is given data as in the actual protocol; 2) An "ideal-world" experiment, where the adversary interacts with data generated by a simulator. The SNIP protocol satisfies the zero-knowledge property if, even when the adversary is given x as input, she has no advantage in distinguishing these two experiments. Since all servers play symmetric roles and all exchanged values are additively secret-shared, it suffices to prove security in the case of a single adversarial server interacting with a single honest server.

The core idea of the proof is to construct an explicit simulator that generates the adversary's view in the ideal world. The simulator proceeds as follows: It generates the adversary's view $([a]_{\mathcal{A}}, [b]_{\mathcal{A}}, [c]_{\mathcal{A}}, [x]_{\mathcal{A}}, [f(0)]_{\mathcal{A}}, [g(0)]_{\mathcal{A}}, [h]_{\mathcal{A}})$ uniformly at random in $\mathbb{F}$, and computes the honest server's corresponding values $([h(M)]_{\mathbf{h}}, [d]_{\mathbf{h}}, [e]_{\mathbf{h}})$ accordingly before sharing them with $\mathcal{A}$.

Since $f(0)$ is sampled uniformly at random in $\mathbb{F}$ in the real world, a well-chosen r ensures that $f(r)$ is also uniformly distributed and independent of the adversary's initial view, even though she has access to x and can reconstruct $f(1), \ldots, f(M)$ (and similarly for g). Moreover, the perfect secrecy of the secret-sharing scheme ensures that the values $a, b, c, x, f(0), g(0), h$ are perfectly hidden. As a result, the adversary's view in the ideal world (where these values are generated uniformly at random) remains statistically indistinguishable from the real-world distribution.

Thus, the adversary's view at the start of both experiments is identically distributed, providing strong intuition as to why she cannot distinguish between the real and ideal worlds. With a more detailed formalization of the simulator, this argument extends to a full proof that the SNIP protocol satisfies the zero-knowledge property.

6 AFEs

So far, the protocol presented allows computing private sums over client-provided data (Section 3) and to check arbitrary validation predicate against secret-shared data (Section 4). Now we want to compute more sophisticated statistics, for example $f(x_1, \ldots, x_n)$, over the private client data $x_1, \ldots, x_n$. This is where **Affine-aggregatable encodings (AFEs)** come into play.

Formally, an AFE consists of three efficient algorithms defined with respect to a field $\mathbb{F}$, and two integers k and k', where $k' \leq k$:

- $\mathsf{Encode}(x)$: Maps an input x to its encoding in $\mathbb{F}^k$,
- $\mathsf{Valid}(y)$: Return true if and only if $y \in \mathbb{F}^k$ is a valid encoding of some valid input x.
- $\mathsf{Decode}(\sigma)$: Takes $\sigma = \sum_i \mathsf{Trunc}_{k'}(\mathsf{Encode}(x_i)) \in \mathbb{F}^{k'}$ as input, and outputs $f(x_1, \ldots, x_n)$. The $\mathsf{Trunc}(\cdot)$ function outputs the first k' components of its input.

As an example, let's look at the AFE that allows computing the variance of the set of b-bit integers $(x_1, \ldots, x_n)$, and see how it integrates into the Prio protocol. First, each client encodes its integer x as $(x, x^2, \beta_0, \ldots, \beta_{b-1})$ where $(\beta_0, \ldots, \beta_{b-1}) \in \{O,1\}^b$ is the binary representation of x. The clients then share those encoding according to the additive secret-sharing mechanism described in Section 3. To check this encoding, the Valid algorithm checks that the following equalities hold over $\mathbb{F}$: $\mathsf{Valid}(\mathsf{Encode}(x)) = \left(x = \sum_{i=0}^{b-1} 2^i \beta_i\right) \wedge (x \cdot x = x^2) \wedge \bigwedge_{i=0}^{b-1} [(\beta_i - 1) \cdot \beta_i = 0]$. This verification is performed using the SNIP construct described in Section 4. Then, the servers aggregate their shares by summing the first $k' = 2$ components of each encoding, leading to $(\sigma_0, \sigma_1) = \sum_{i=1}^{n} \mathsf{Trunc}_{k'}(\mathsf{Encode}(x_i)) = \sum_{i=1}^{n} (x_i, x_i^2) = (\sum x_i, \sum x_i^2)$. Finally, the variance can be computed using the identity $\mathrm{Var}(\mathrm{X}) = \mathrm{E}[\mathrm{X}^2] - \mathrm{E}[\mathrm{X}]^2$: $\mathsf{Decode}(\sigma) = (\sigma_1 - \sigma_0^2)/n$.

This approach not only enables the computation of complex statistics like variance but also extends to more advanced tasks, such as training machine learning models on private client data, while preserving data privacy and ensuring correctness.

7 Prio Protocol Overlook

Consider a setting with n clients, where m of them are malicious and $n-m$ are honest. Each honest client holds a private input $x_i \in \mathbb{D}$ (for $i \in \{1, ..., n-m\}$) and there are s servers that wish to compute an aggregate statistic via a function $f : \mathbb{D}^{n-m} \to \mathbb{A}$. The Prio protocol proceeds in four main steps:

1. **Upload phase**: Each client i encodes its private input $x_i \in \mathbb{D}$ into a vector $y_i \in \mathbb{F}^k$ using an affine-aggregatable encoding (*AFE*) algorithm: $y_i \leftarrow Encode(x_i)$. The client then splits y_i in s shares, $[y_i]_1, \ldots, [y_i]_s$, such that $y_i = [y_i]_1 + \ldots + [y_i]_s$. This sharing ensures that each server receives only a portion of the data, sufficient for computing the aggregation function but insufficient to reconstruct the original information. To prove that the encoded input is well-formed ($x_i \in \mathbb{D}$), the client generates a *secret-shared non-interactive proof* (*SNIP*). Finally the client sends each server j the corresponding input share $[y_i]_j$ and share of the SNIP proof over an authenticated and encrypted channel.
2. **Validation phase**: After receiving client submissions, the servers jointly verify the SNIP confirming that each encoded input corresponds to a valid $x_i \in D$.
3. **Aggregation phase**: Each server j maintains an accumulator $A_j \in F^{k'}$ (initialized to zero) and for each verified client submission, the server updates its accumulator by adding a truncated version of the client share (only the the first k' elements carry the necessary information): $A_j \leftarrow A_j + Trunc_{k'}([y_i]_j) \in \mathbb{F}^{k'}$.

4. **Publish phase**: Once all client shares have been processed, the servers publish their accumulator values $A_1, A_2, \ldots, A_s$. The final aggregate is computed by summing the accumulators $\sigma = \sum_{j=1}^{s} A_j = \sum_{i=1}^{n} Trunc_{k'} y_i$ and the final aggregate statistic is then recovered by applying the AFE decoding algorithm: $Decode(\sigma) \in \mathbb{A}$.

Security protocol properties: *Robustness against malicious clients* - guaranteed by the soundness of the SNIP construction; *f-privacy* - holds as the only information leaked is the final aggregate value (as long as at least one server is honest, the AFE is f-private and the SNIP is zero-knowledge); *Anonymity* - holds if the function f is *symmetric*, i.e. if a permutation $(x'_1, \ldots, x'_{n-m})$ of the inputs $(x_1, \ldots, x_{n-m})$ does not affect the output, so if $f(x_1, \ldots, x_{n-m}) = f(x'_1, \ldots, x'_{n-m})$.

8 Evaluation

Protocol evaluation was done by comparing a protocol implementation (using SNIPs) with a private aggregation scheme that uses non-interactive zero-knowledge proofs (NIZKs).

Client Performance: Encoding a data submission of 100 4-bit integers (of 87-bit finite field) takes ~ 0.024 seconds on a workstation (using a 2010 *Intel Xeon E5620*) and ~ 0.112 seconds on a smartphone (using a 2012 *Samsung S3*). For a 265-bit field, the encoding times increase slightly: ~ 0.036 seconds on a workstation and ~ 0.17 on a smartphone.

Server Throughput (*submission processed/s*): For the Prio protocol it is roughly one-fifth of a no-privacy scheme and Prio outperforms NIZK-based schemes by $10\times$. Adding more servers does not significantly affect throughput because the workload for checking each client submission is efficiently distributed. A server is selected as leader and coordinates verification while the non-leader servers send only a constant amount of data per submission, independent of the submission size and of the complexity of the validity check.

Computational efficiency: Compared to schemes based on public key cryptography, Prio outperforms them as it only requires the client to perform a single public key cryptography operation. For a validation circuit with M multiplication gates, a NIZK based scheme requires the client to perform $2M$ exponentiations; in contrast Prio requires $O(M \cdot \log M)$ multiplications in $\mathbb{F}$. Although SNARK-based protocols produce succinct proofs, their verification time grows with the statement size: A client has to compute $s \cdot L$ hashes (where s is the number of servers and L is the submission length).

Using Prio protocol to privately train a d-dimensional *least-squares regression model* on private client-submitted data (e.g. a vector of 14-bit integers with health information) it leads to a $50\times$ slowdown at the client over a *no-privacy* scheme (due to the computation of crafting SNIPs); the server incurs in only a 1-$2\times$ slowdown compared to a *no robust* scheme and a 5-$15\times$ slowdown over a *no privacy* scheme. In comparison a NIZK-based scheme is around 100-$200\times$ worse than the *no privacy* scheme.

9 Discussion

9.1 Limitations

The protocol suffer some limitations and attacks that can lower the privacy target:

- **Selective Denial-of-Service Attack**: An attacker could prevent all honest clients except one from contacting the servers. In this case the protocol computes $f(x_{honest}, x_{evil_1}, \ldots, x_{evil_m})$ and the adversary could infer information about x_{honest}. A standard defense involves servers keeping a list of registered client keys.
- **Intersection Attack**: An adversary observing the output of $f(x_1, \ldots, x_n)$, can block an honest client (e.g. the n-th) in a subsequent run and retrieve: $f(x'_1, \ldots, x'_{n-1})$. If the client inputs remain unchanged, then the adversary can deduce information about x_n. A standard defense involves servers adding differential privacy noise to the result before publishing it.
- Prio provides *robustness if all servers are honest*. Extending it to faulty servers is useful but it weakens the privacy guarantees of the system: a system that protects robustness in the presence of k faulty servers, can only protect privacy against at most $s - k - 1$ malicious servers.

9.2 Real-world deployments

Since its publication nearly 8 years ago, Prio has seen large-scale deployments. During the *COVID-19 pandemic*, *Apple* and *Google* introduced *exposure notification systems* to alert users about potential contact with virus positive individuals. To monitor viral transmission patterns aggregated insight were requested by public health agencies. To address this need, Apple and Google implemented a privacy-preserving data aggregation protocol similar to the one introduced by *Boneh* and *Corrigan-Gibbs* with *Prio*. Mozilla uses *Oblivious HTTP* and the Prio-based *Distributed Aggregation Protocol* for collecting aggregate browser statistics.

10 Conclusion

Prio protocol provides the bases for a real-world system, based on advanced cryptographic techniques, that combines the benefits of privacy, robustness and performance and balances their trade-offs; this protocol not only protects individual user data, but also opens up new avenues for private large-scale data analytics, which is now more than ever important.

References

[1] BEAVER, D. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology — CRYPTO '91* (Berlin, Heidelberg, 1992), J. Feigenbaum, Ed., Springer Berlin Heidelberg, pp. 420–432.

[2] CORRIGAN-GIBBS, H., AND BONEH, D. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, Mar. 2017), USENIX Association, pp. 259–282.