Prio: Private, Robust and Scalable Computation of Aggregate Statistics

By: Henry Corrigan - Gibbs and Dan Boneh (Stanford University)

Presented By: Avirudh Kaushik
CMPE 253

Smartphones, Cars, Wearable Electronics and many other form of devices which we use today are transmitting telemetry data and sensor reading to cloud services.

With such data in hand the cloud services can gather useful aggregate statistics over the entire population of devices.

Example:

Navigation app providers collect real-time location data from their users to identify areas of trac congestion in a city and route drivers along the least-crowded roads.

Fitness tracking services collect information on their users' physical activity so that each user can see how her fitness regimen compares to the average

BUT ! !

Motivated Attackers may steal and disclose client's sensitive information.

Cloud Services may misuse the client's information for profit.

Intelligence agencies may gather data for targeting or for mass surveillance purposes.

How Do We Protect our Data from these Attackers ?

Apple and Google fight such attacks by deploying privacy-preserving techniques which use 'randomized response' to achieve differential-privacy.

For example, a mobile phone vendor may want to learn how many of its phones have a particular uncommon but sensitive app installed (e.g., the AIDSinfo app). In the simplest variant of this approach, each phone sends the vendor a bit indicating whether it has the app installed, except that the phone flips its bit with a fixed probability $p < 0.5$. By summing a large number of these noisy bits, the vendor can get a good estimate of the true number of phones that are running the sensitive app.

Using the ‘Randomized Response’ makes the system robust but it has some downfalls to it. It provides relatively weak privacy guarantees.

Every bit that each phone transmits leaks some private user information to the vendor. In particular, when p = 0.1 the vendor has a good chance of seeing the correct (unflipped) user response. Increasing the noise level p decreases this leakage, but adding more noise also decreases the accuracy of the vendor’s final estimate.

An alternative approach to the data-collection problem is to have the phones send encryptions of their bits to a set of servers. The servers can sum up the encrypted bits and decrypt only the final sum.

However, in gaining this type of privacy, many secret sharing- based systems sacrifice robustness: a malicious client can send the servers an encryption of a large integer value v instead of a zero/one bit

Introducing Prio

Prio is a system for Private aggregation that resolves tension between privacy, robustness and scalability.

The authors claim that Prio provides privacy in a way such that no server can observe the actual data which is transmitted by a client. The servers can only access the aggregate values of the data being transmitted by the users.

The authors also claim that Prio can maintain robustness even in the presence of unbounded number of malicious clients as long as one server is not colluded.

Prio provides scalability as well !

When the system is deployed in 5 servers spread across different parts of the world it imposes a mere 5.7 X slowdown as compared to a regular non privacy providing system.

While a state of the art Non Interactive Zero Knowledge (NIZK) system provides a 267 X slowdown as compared to the regular non privacy providing system.

(The tests conducted were to compute private sums over vectors of private client data)

System Goals

A Prio deployment consists of a small number of infrastructure servers and a very large number of clients. In each time epoch, every client i in the system holds aprivate value Xi. The goal of the system is to allow the servers to compute f (x1;.....; xn), for some aggregation function f , in a way that leaks as little as possible about each client's private Xi values to the servers.

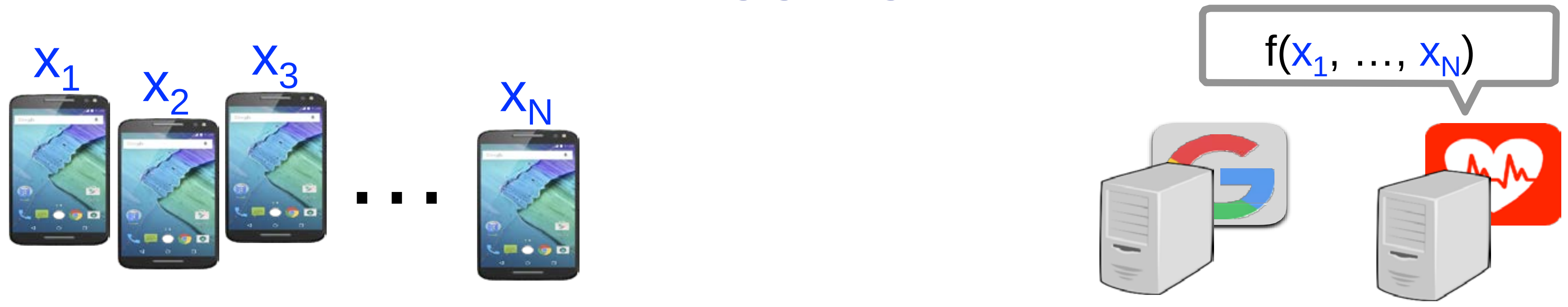
Encryption: The parties to a Prio deployment must establish pairwise authenticated and encrypted channels.Towards this end, we assume the existence of a public key infrastructure and the basic cryptographic primitives (CCA-secure public-key encryption, digital signatures, etc.) that make secure channels possible.

Anonymity: A data-collection scheme maintains client anonymity if the adversary cannot tell which honest client submitted which data value through the system, even if the adversary chooses the honest clients' data values, controls all other clients, and controls all but one server. Prio always protects client anonymity.

Privacy: Prio provides f -privacy, for an aggregation function f , if an adversary, who controls any number of clients and all but one server, learns nothing about the honest clients' values Xi, except what she can learn from the value f (X1;... ; Xn) itself. More precisely, given f (X1; ... ; Xn), every adversary controlling a proper subset of the servers, along with any number of clients, can simulate its view of the protocol run.

Secret Sharing Scheme

Private aggregation



1. **Exact correctness**	If all servers are honest, servers learn f(·)
2. **Privacy**	If one server is honest, servers learn only* f(·)
3. **Robustness**	Malicious clients have bounded influence
4. **Efficiency**	No public-key crypto (apart from TLS) 1000s of submissions per second

Prio is the first system to achieve all four.

Warm-up: Computing private sums

- Every device i holds a value x_i
- We want to compute

$$f(x_1, \ldots, x_N) = x_1 + \ldots + x_N$$

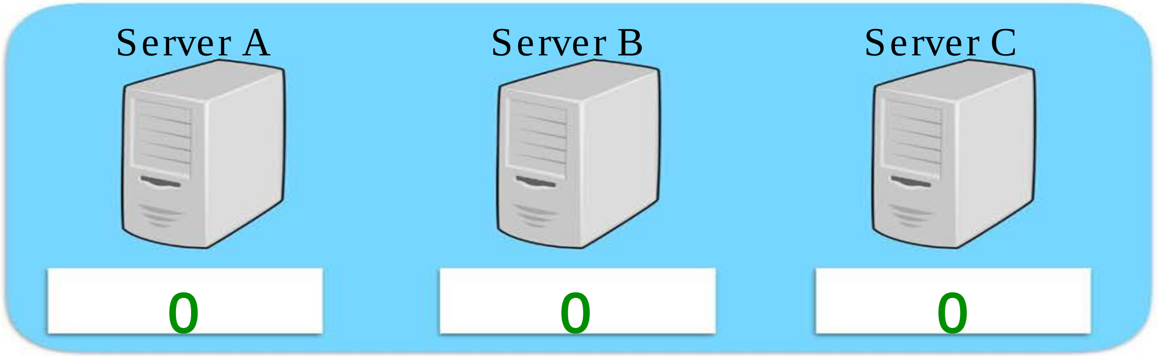
without learning any users' private value x_i.

Example: Privately measuring traffic congestion.

$x_i = 1$ if user i is on the Bay Bridge
$= 0$ otherwise

The sum $x_1 + \ldots + x_N$ yields the number of app users on the Bay Bridge.

Private sums:
A "straw-man" scheme



1

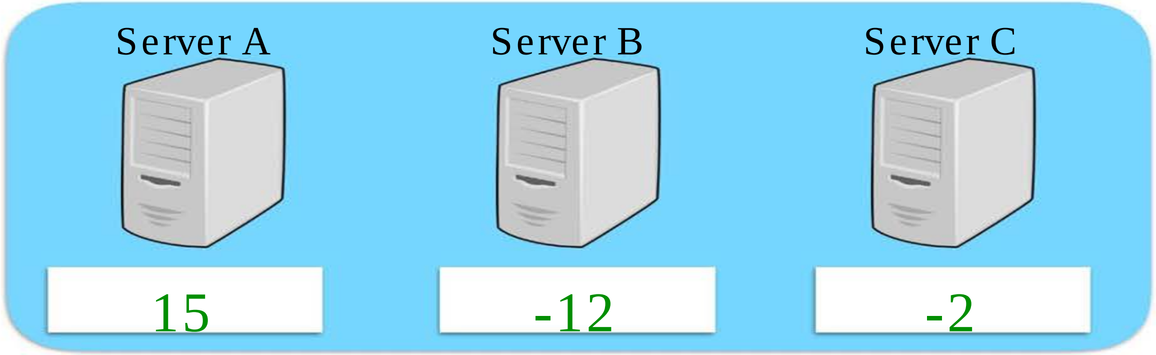
Secret sharing

Pick three random "shares" that sum to 1.

$1 = 15 + (-12) + (-2) \pmod{31}$

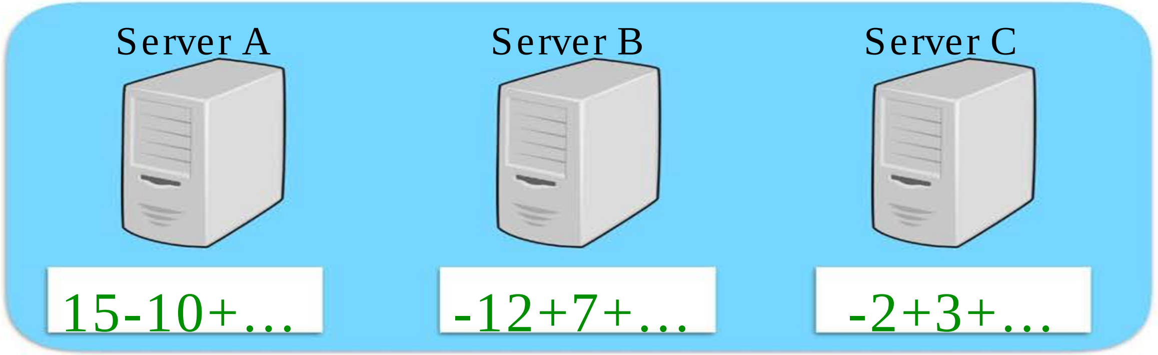
Need all three shares to recover the shared value.

Private sums: A “straw-man” scheme

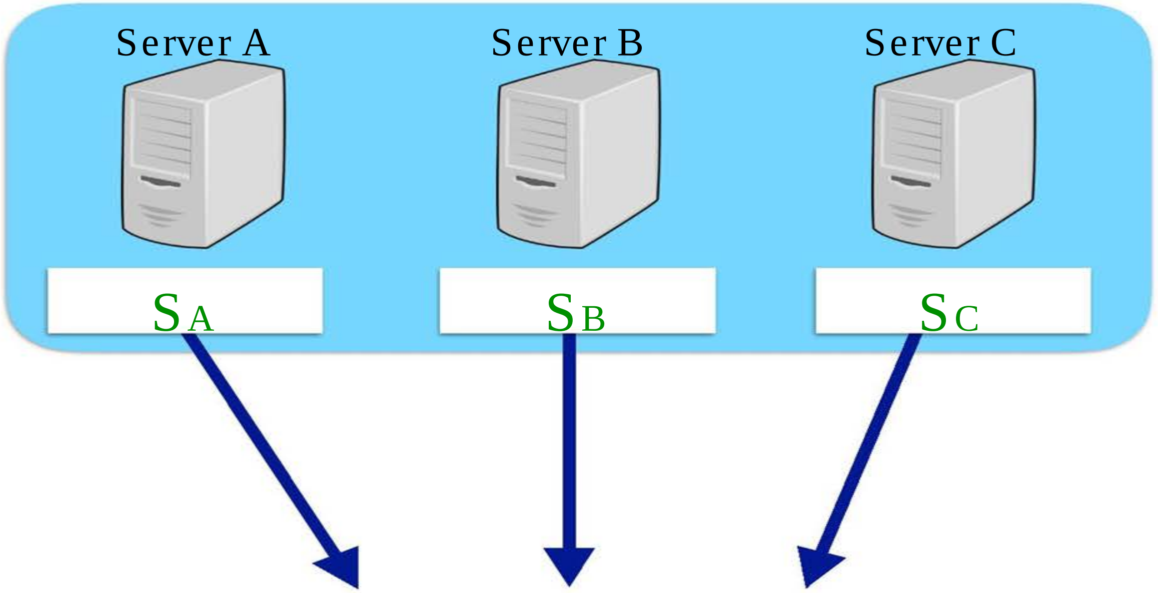


0 = (-10) + 7 + 3

Private sums:
A “straw-man” scheme

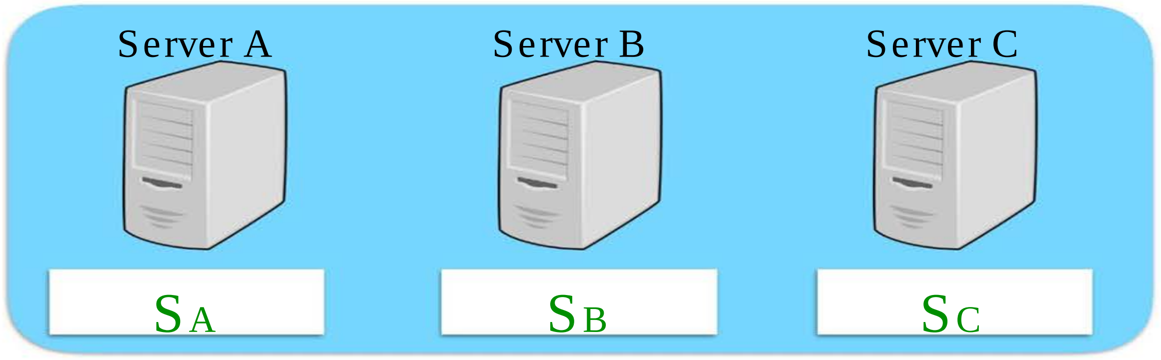


Private sums: A “straw-man” scheme



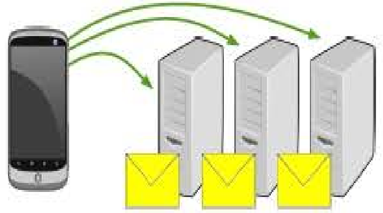
$S_A + S_B + S_C = 15 + -10 + \ldots$

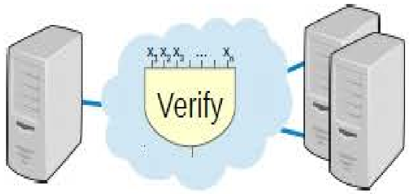
Private sums: A “straw-man” scheme



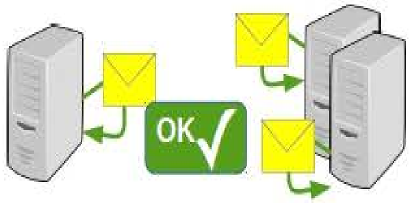
$$S_A + S_B + S_C = 15 + -10 + \ldots$$
$$= 1 + 0 + \ldots + 1$$

Servers learn the sum of client values and learn nothing else.

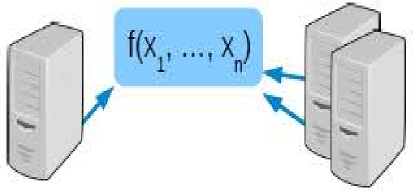

(a) The client sends a share of its encoded submission and SNIP proof to each server.



(b) The servers validate the client's SNIP proof to ensure that the submission is valid.



(c) If the checks pass, the servers update their local accumulators with the client-provided data.



(d) After accumulating many packets, the servers publish their accumulators to reveal the aggregate.

There are two observations we can make about this scheme. First, even this simple scheme provides privacy: the servers learn the sum $\sum_i x_i$ but they learn nothing else about the client's private inputs.

Second, the scheme does not provide robustness. A single malicious client can completely corrupt the protocol output by submitting (for example), a random integer $r \in \mathbb{F}_p$ to each server.

So How do we ensure that a client cannot submit a random integer to a server ?

SNIP - Secret Sharing Non Interactive Proofs

Prio servers need a way to check if the client-submitted value is well formed. For example, in the simplified protocol to calculate sums, every client is supposed to send the servers the share of a value x such that $0 < x < 1$.

But since each server receives a random number it cannot check if the sent bit is valid or not. For the servers to know that the bit is valid they have to communicate within themselves to ensure that the sent bits are valid.

The servers hold a validation predicate Valid(), and should only accept the client's data submission if Valid(x) = 1.

A secret-shared non-interactive proof (SNIP) protocol consists of an interaction between a client (the prover) and multiple servers (the verifiers). At the start of the protocol:

– each server i holds a vector $[x]_i$

– the client holds the vector $x = \sum_i [x]_i$

– all parties hold an arithmetic circuit representing a predicate Valid.

(An arithmetic circuit is like a boolean circuit except that it uses finite-field multiplication, addition, and multiplication by- constant gates, instead of boolean and, or, and not gates)

Conditions for SNIP:

The client’s goal is to convince the servers that Valid(x) =1, without leaking anything else about x to the servers. To do so, the client sends a proof string to each server. After receiving these proof strings, the servers gossip amongst themselves and then conclude either that Valid(x) = 1 (the servers “accept x”) or not (the servers “reject x”).

Correctness: If all parties are honest, the servers will accept x.

Soundness: If all servers are honest, and if Valid(x) , 1,then for all malicious clients, even ones running in super-polynomial time, the servers will reject x with overwhelming probability

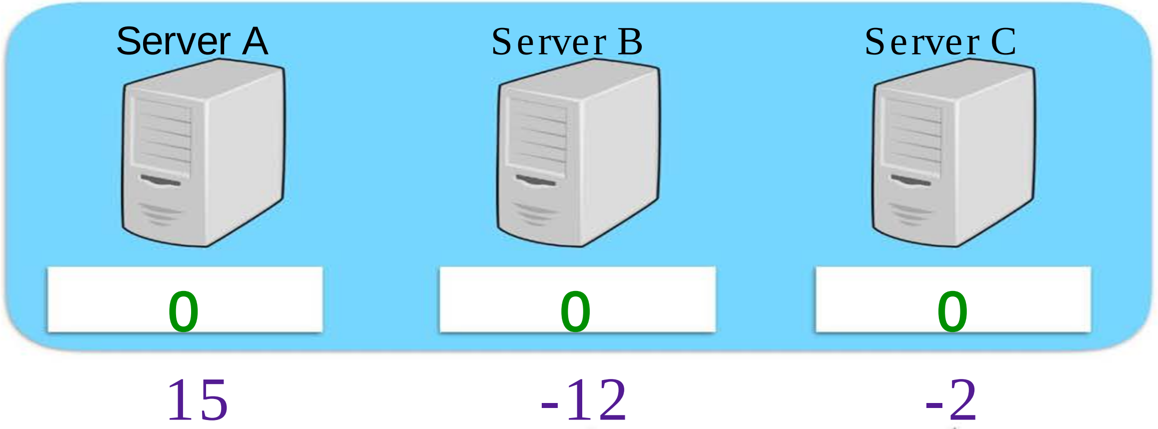
Zero - Knowledge: If the client and at least one server are honest, then the servers learn nothing about x, except that Valid(x) = 1

The author’s have used SNIP in the Prio System because SNIP reduces the number of Server - Server Communications.

To build the SNIP, the author’s first generalize a “batch verification” technique of Ben-Sasson et al. And then show how a set of servers can use it to verify an entire circuit computation by exchanging a only few field elements. They implement this last step with a new adaptation of Beaver’s multi-party computation (MPC) protocol to the client/server setting.

Contribution 1

Secret-shared non-interactive proofs (SNIPs)



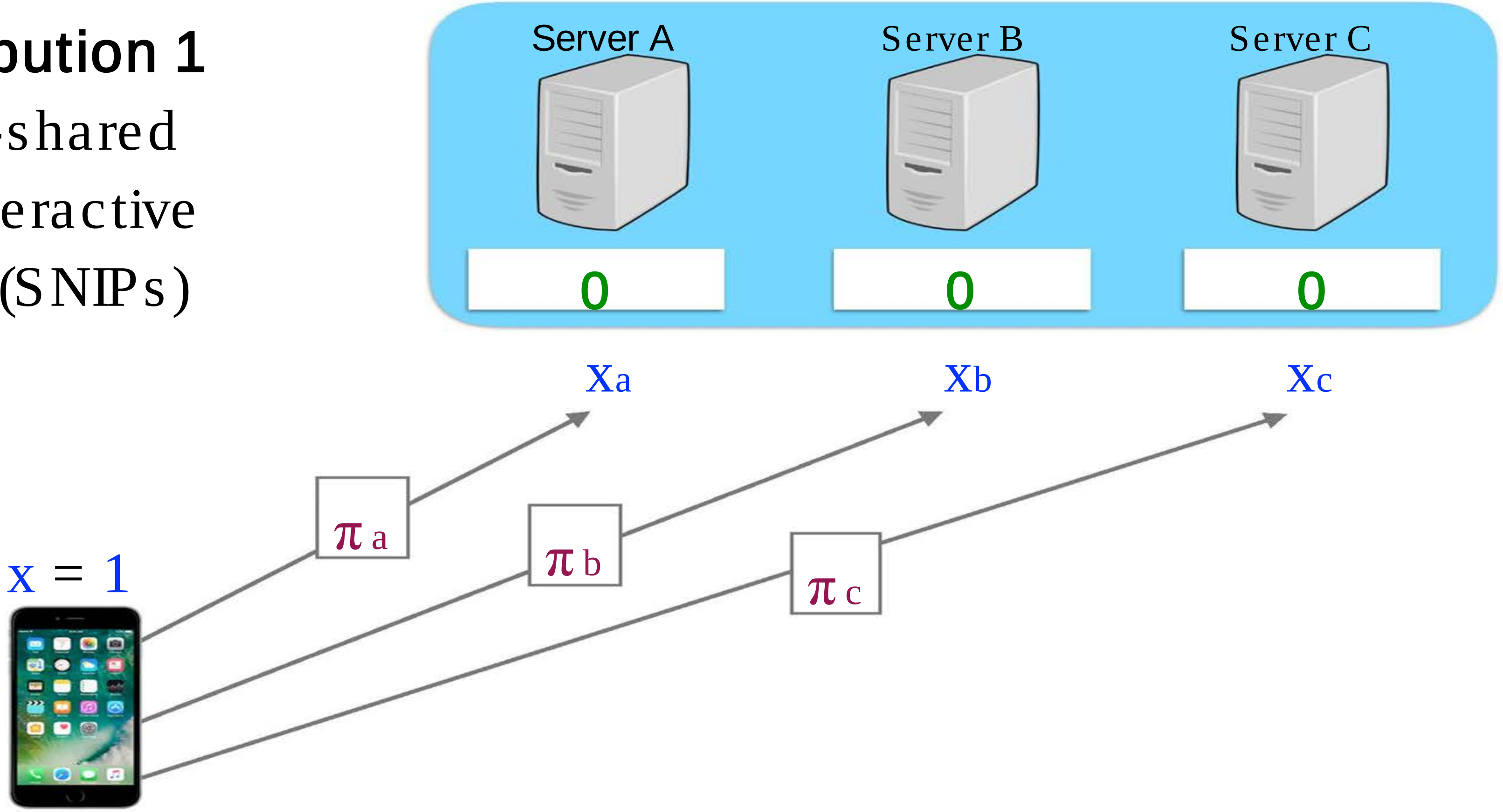
The servers want to ensure that their shares sum to 0 or 1

…without learning x.

x = 1

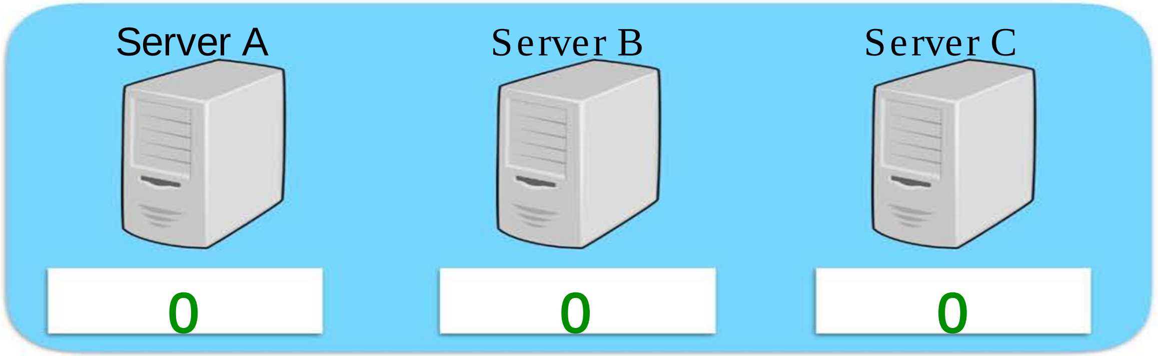
Contribution 1

Secret-shared non-interactive proofs (SNIPs)



Contribution 1

Secret-shared non-interactive proofs (SNIPs)

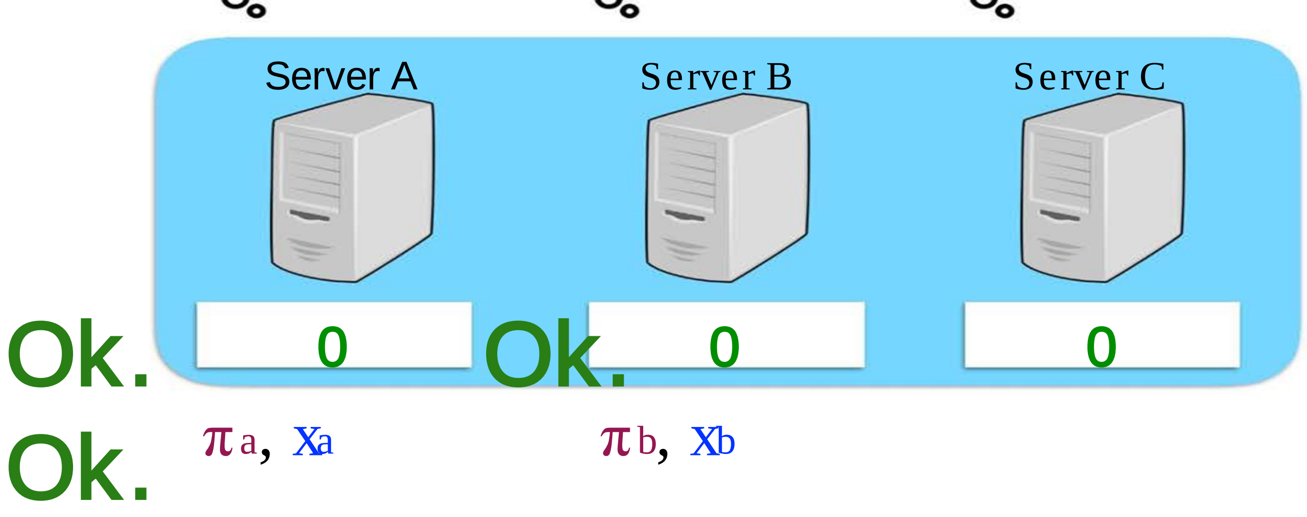


π_a , x_a π_b, x_b Π_c, x_c

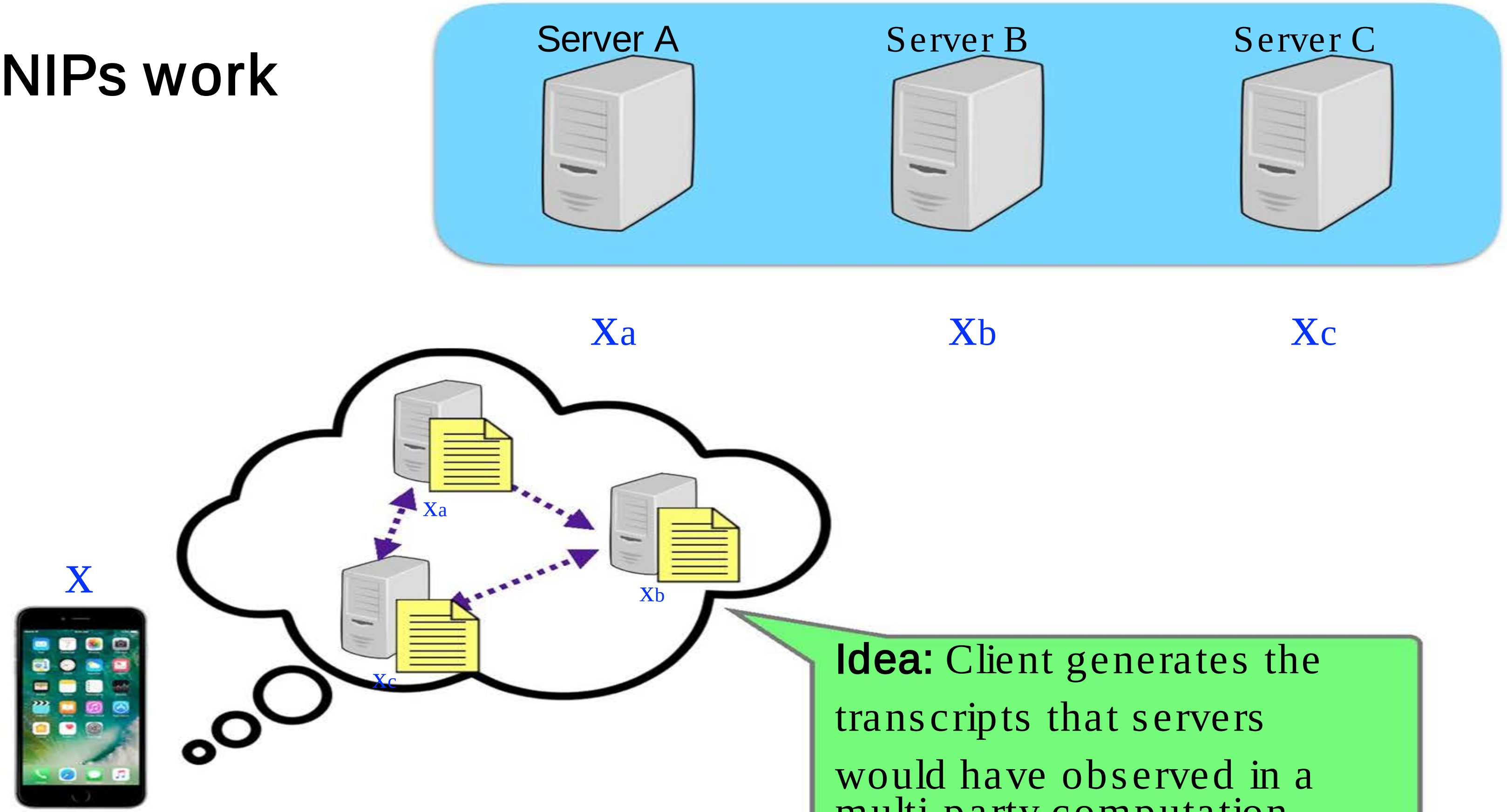
$x = 1$

Contribution 1

Secret-shared non-interactive proofs (SNIPs)



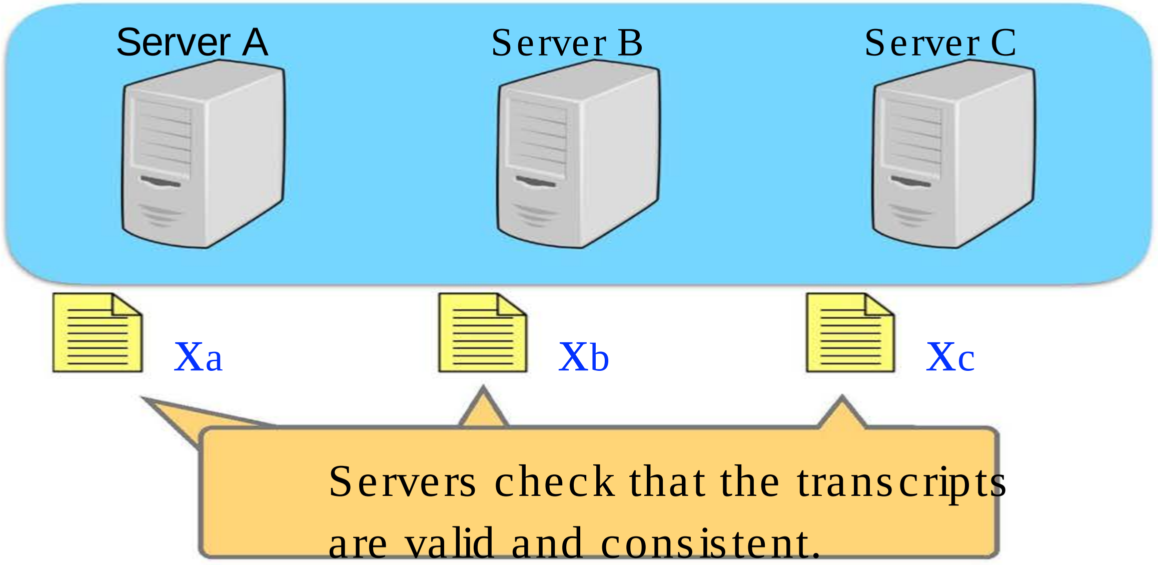
How SNIPs work



Idea: Client generates the transcripts that servers would have observed in a multi-party computation

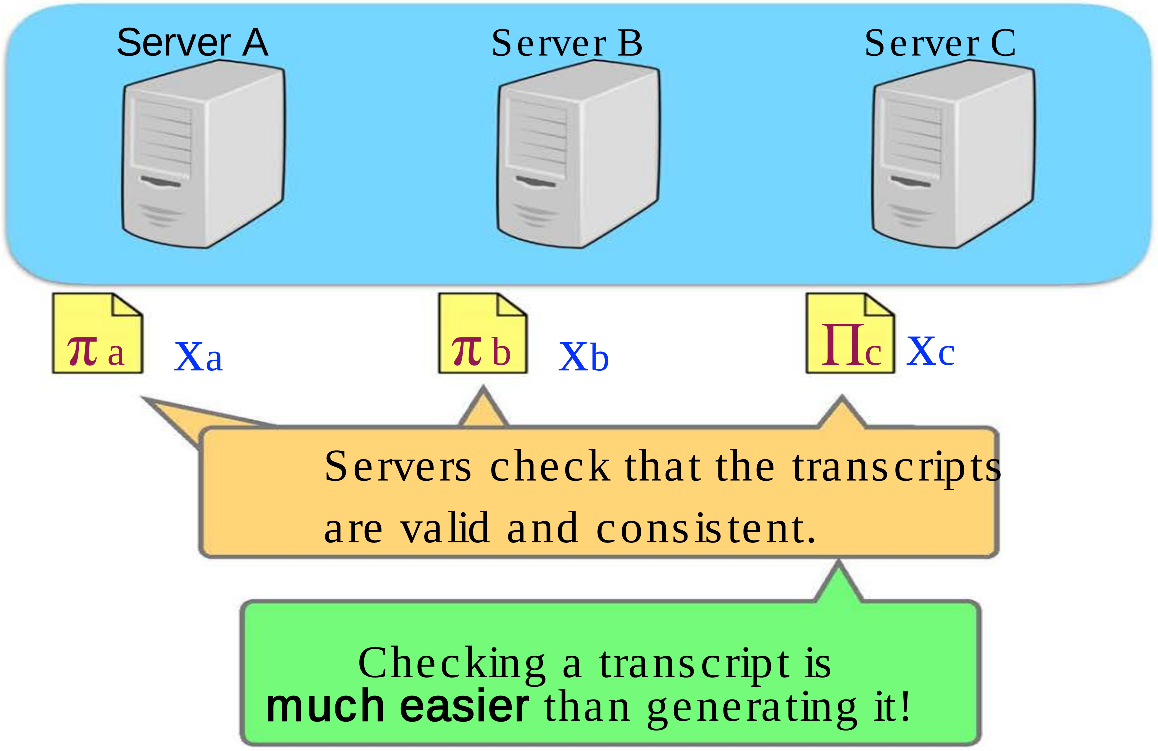
See also [IKOS07]

How SNIPs work



Servers check that the transcripts are valid and consistent.

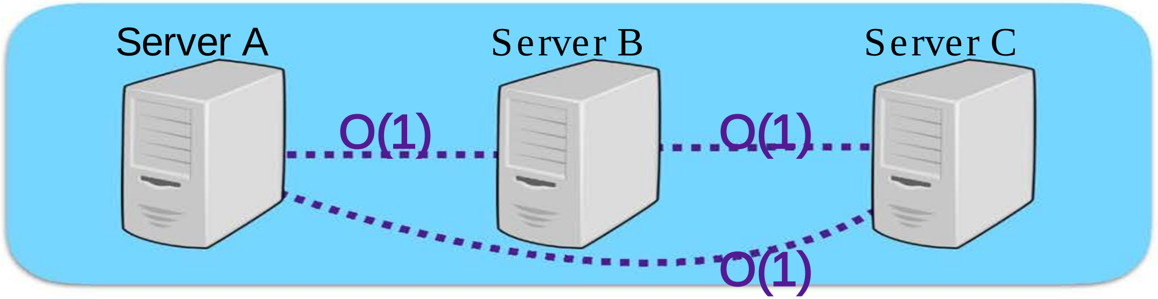
How SNIPs work



Servers check that the transcripts are valid and consistent.

Checking a transcript is **much easier** than generating it!

How SNIPs work



D_a D_b D_c

- If x is valid, $D_a + D_b + D_c = 0$
- If x is invalid, $D_a + D_b + D_c \neq 0$ with high probability

Servers run lightweight multi-party computation to check that $D_a + D_b + D_c = 0$

If so, servers accept x is valid.

[BFO12]

Efficiency of SNIP:

The remarkable property of this SNIP construction is that the server-to-server communication cost grows neither with the complexity of the verification circuit nor with the size of the value of number of bits being used to store the data. The computation cost at the servers is essentially the same as the cost for each server to evaluate the Valid circuit locally. That said, the client-to-server communication cost does grow linearly with the size of the Valid circuit.

M = # of multiplication gates in Valid$(\cdot)$ circuit	Public-key ops.		Communication		Slow-down
	Client	Server	C-to-S	S-to-S	
Dishonest-maj. MPC [CLOS02], [DPSZ12], …	0	$\Theta(M)$	0	$\Theta(M)$	5,000x at server
Commits + NIZKs [FS86], [CP92], [CS97], …	$\Theta(M)$	$\Theta(M)$	$\Theta(M)$	$\Theta(M)$	50x at server
Commits + SNARKs [GGPR13], [BCGTV13], …	$\Theta(M)$	$O(1)$	$O(1)$	$O(1)$	500x
This work: SNIPs	0	0	$\Theta(M)$	$O(1)$	at client 1x

Gathering Complex Statistics

So far, we have developed the means to compute private sums over client-provided data and to check an arbitrary validation predicate against secret-shared data. Combining these two ideas with careful data encodings, which introduced now, allows Prio to compute more sophisticated statistics over private client data.

At a high level, each client first encodes its private data value in a prescribed way, and the servers then privately compute the sum of the encodings. Finally, the servers can decode the summed encodings to recover the statistic of interest. The participants perform this encoding and decoding via a mechanism we call affine-aggregatable encodings (“AFEs”).

Affine Aggregatable Encodings (AFE)

Each Client i holds a value Xi. The servers hold an aggregate function which gives a range of set of aggregates.

An AFE gives an efficient way to encode the data values Xi such that it is possible to compute the value f(x1, …., xn) given only the sum of the encodings.

AFE consists of 3 efficient algorithms:

1> Encode: Maps an input into its encoding

2> Valid (x): Returns True if and only if y is a valid encoding

3> Decode (σ): Takes σ = ∑(Encoding(x))

AFE in Prio System:

The full Prio system computes f (x1;....; xn) privately as follows. Each client encodes its data value x using the AFE Encode routine for the aggregation function f . Then, as in the simple scheme of calculating sum, every client splits its encoding into s shares and sends one share to each of the s servers. The client uses a SNIP proof to convince the servers that its encoding satisfies the AFE Valid predicate.

Upon receiving a client's submission, the servers verify the SNIP to ensure that the encoding is well-formed. If the servers conclude that the encoding is valid, every server adds the first k components of the encoding share to its local running accumulator. Finally, after collecting valid submissions from many clients, every server publishes its local accumulator, enabling anyone to run the AFE Decode routine to compute the final statistic in the clear.

Evaluation

The Authors have implemented a Prio prototype in 5,700 lines of Go and 620 lines of C (for FFT-based polynomial operations, built on the FLINT library). Unless noted otherwise, the evaluations use an FFT-friendly 87-bit field. Our servers communicate with each other using Go’s TLS implementation. Clients encrypt and sign their messages to servers using NaCl’s “box” primitive, which obviates the need for client-to-server TLS connections.

We compare Prio against a private aggregation scheme that uses non-interactive zero-knowledge proofs (NIZKs) to provide robustness. This protocol is similar to the “cryptographically verifiable” interactive protocol of Kursawe et al. and has roughly the same cost, in terms of exponentiations per client request, as the “distributed decryption” variant of PrivEx [56]. We implement the NIZK scheme using a Go wrapper of OpenSSL’s NIST P256 code

To investigate the load that Prio places on the servers, we configured five Amazon EC2 servers (eight-core c3.2xlarge machines, Intel Xeon E5-2680 CPUs) in five Amazon data centers (N. Va., N. Ca., Oregon, Ireland, and Frankfurt) and had them run the Prio protocols. To maximize the load on the servers, we had each client send a stream of pre-generated Prio data packets to the servers over a single TCP connection. There is no need to use TLS on the client-to-server Prio connection because Prio packets are encrypted and authenticated at the application layer and can be replay-protected at the servers.

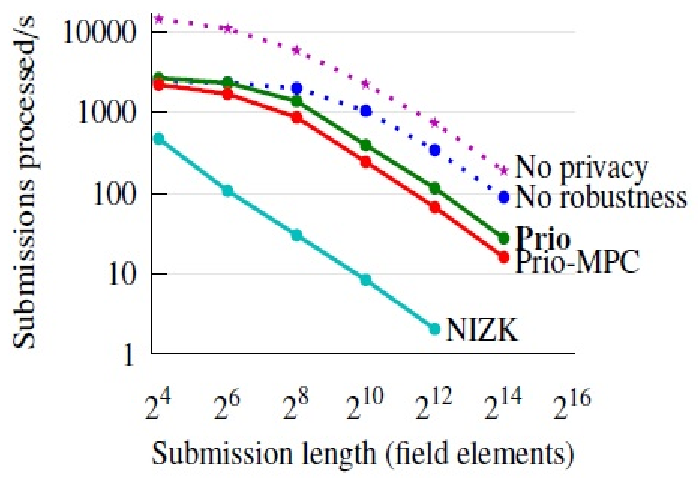


Figure 4: Prio provides the robustness guarantees of zero-knowledge proofs but at 20-50× less cost.

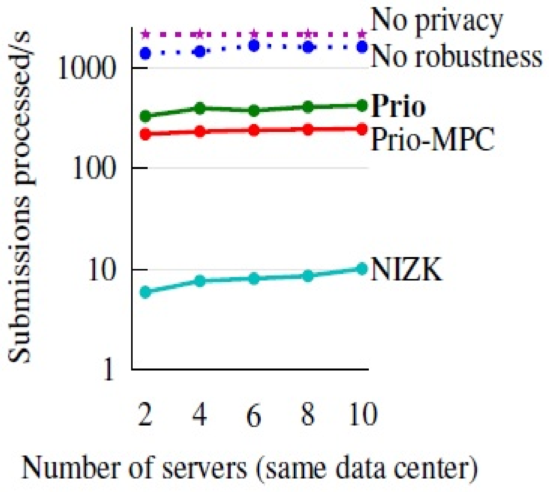


Figure 5: Prio is insensitive to the number of aggregation servers.

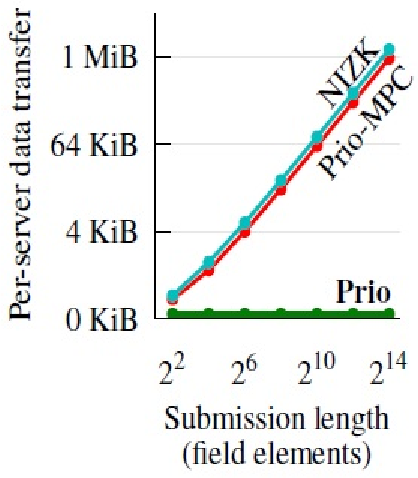


Figure 6: Prio's use of SNIPs (§4) reduces bandwidth consumption.

Application Scenarios:

Cell Signal Strength: A collection of Prio servers can collect the average mobile signal strength in each grid cell in a city without leaking the user's location history to the aggregator. We can divide the geographic area into a km_2 grid—the number of grid cells depends on the city's size and we encode the signal strength at the user's present location as a four-bit integer.

Browser Statistics: The Chromium browser uses the RAPPOR system to gather private information about its users. We can implement a Prio instance for gathering a subset of these statistics: average CPU and memory usage, along with the frequency counts of 16 URL roots.

Health Data Modelling: We implement the AFE for training a regression model on private client data. We use the features from a pre-existing heart disease data set (13 features of varying types: age, sex, cholesterol level, etc.) [78] and a breast cancer diagnosis data set.

Using Machine Learning

We can perform an end-to-end evaluation of Prio when the system is configured to train a d-dimensional least squares regression model on private client-submitted data, in which each training example consists of a vector of 14-bit integers. These integers are large enough to represent vital health information, for example.

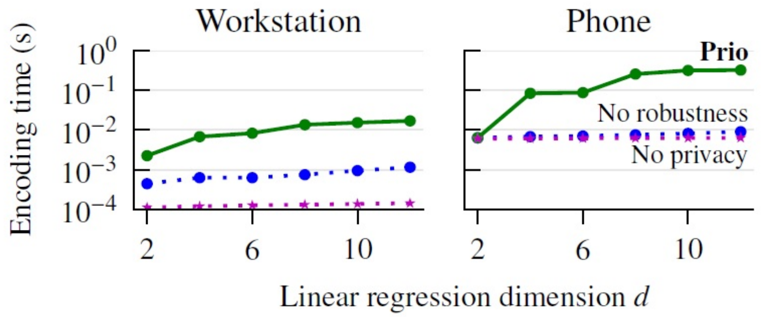


Figure 8: Time for a client to encode a submission consisting of d-dimensional training example of 14-bit values for computing a private least-squares regression.

Deployable Scenarios:

Deployment in Organizations: Prio lets an organization compute aggregate data about its clients without ever storing client data in a single vulnerable location. The organization could run all s Prio servers itself, which would ensures data privacy against an attacker who compromises up to s - 1 servers

App Store (Google Playstore or Ios): Mobile application platform (e.g., Apple's App Store or Google's Play) can run one Prio server, and the developer of a mobile app can run the second Prio server. This allows the app developer to collect aggregate user data without having to bear the risks of holding these data in the clear.

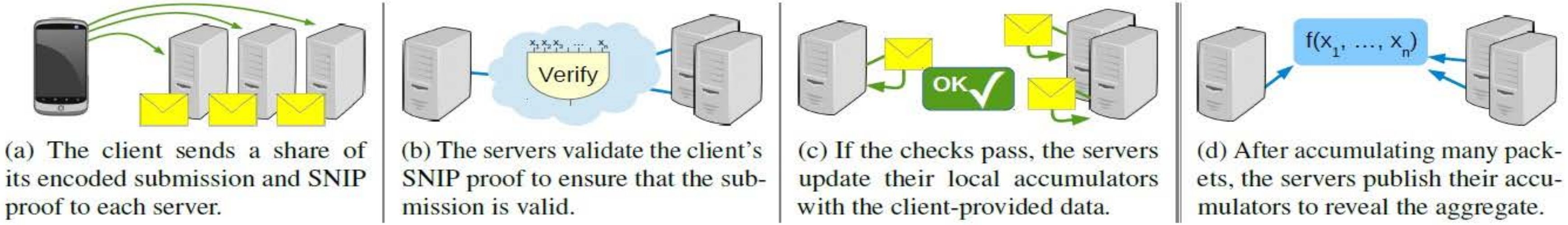
Private Computing Services: A large enterprise can contract with an external auditor or a non-profit (e.g., the Electronic Frontier Foundation) to jointly compute aggregate statistics over sensitive customer data using Prio.

Conclusions

Prio allows a set of servers to compute aggregate statistics over client-provided data while maintaining client privacy, defending against client misbehavior, and performing nearly as well as data-collection platforms that exhibit neither of these security properties.

One question for future work is whether it is possible to efficiently extend Prio to support combining client encodings using a more general function than summation, and what more powerful aggregation functions this would enable. Another task is to investigate the possiblity of shorter SNIP proofs: This implementation grows linearly in the size of the Valid circuit, but sublinear - size information-theoretic SNIPs may be feasible.

Another area of future research is to make the system safe from colluding servers. Right now the system works well if at least one server is not compromised but in a real world scenario there is a high probability that almost all the servers will be compromised. The robustness of the Prio system should be strengthened.



(a) The client sends a share of its encoded submission and SNIP proof to each server.

(b) The servers validate the client's SNIP proof to ensure that the submission is valid.

(c) If the checks pass, the servers update their local accumulators with the client-provided data.

(d) After accumulating many packets, the servers publish their accumulators to reveal the aggregate.

THANK YOU !!