Privacy Enhancing Technologies
Lecture 7 – Collecting Private Statistics

Florian Tramèr

AGENDA

1. Recap
2. Private Aggregation and Applications
3. ZK proofs on secret-shared data

1 Recap

This lecture will conclude the first half of the course where we focus on cryptographic tools for privacy. Today, we will cover a really nice, simple and practical protocol for a specific problem that appears everywhere: computing private sums.

The protocol we'll see combines ideas from many of our previous lectures: secret sharing, non-colluding servers, SNARGs, (some variants also use DPFs), etc. It is also a nice recent "success story" for cryptographic PETs, which had led to large-scale real-world deployments by Mozilla, Apple, Google, and others.

Finally, I think this result is also a super neat marriage of theory and practice. An early version of the protocol we'll discuss was proposed in [CGB17], with a focus on practical efficiency. But it turned out that the protocol could be significantly improved (both asymptotically and in practice) by drawing connections to the broader theory of SNARGs [BBCG+19],

2 Private Aggregation

Let's consider a very simple but ubiquitous task. There is a large number n of parties, each holding a vector x_i of d values:

$$x_1, x_2, \dots, x_n \in \mathbb{F}_p^d .$$

We want to collect the aggregate value

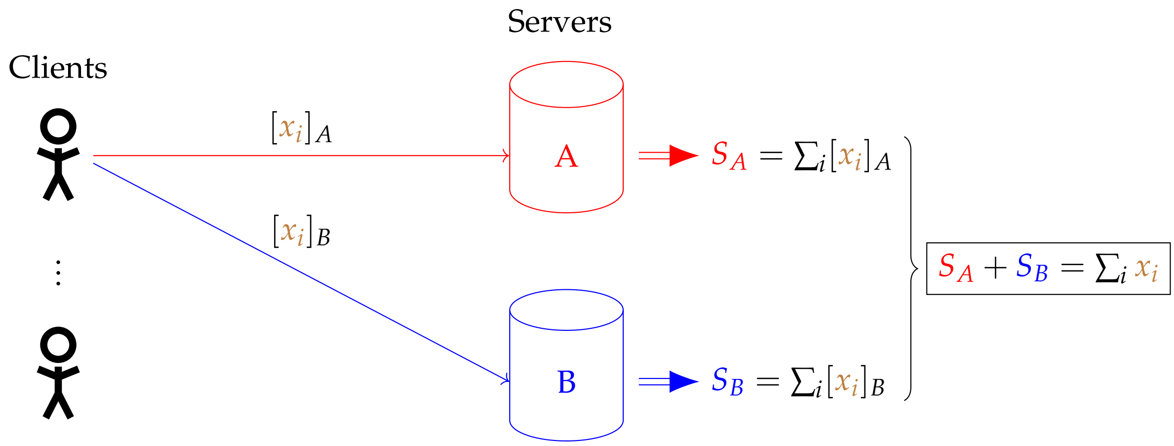
$$\sum_{i=1}^{n} x_i \in \mathbb{F}_p^d .$$

We can view this as a "degenerate" form of MPC where we only want to compute a linear function of the users' values. We could thus solve this with a full-blown (maliciously secure) MPC protocol, but this is rather inefficient, both in computation and communication.

Instead, we will consider a protocol where the data is aggregated by a server.

2.1 A Simple Protocol

To make the protocol as efficient as possible, we'll assume we have two *non-colluding* servers. A simple protocol would then look like this:



That is, each party additively secret-shares their input between both servers, and then the servers add all their shares together before revealing their share of the sum.

Properties of this simple protocol:

- **Correctness:** If all parties act honestly, we get the correct answer.
- **Privacy:** If one server is honest, the protocol leaks nothing beyond the sum $\sum_i x_i$.
- **Efficiency:** Clients send a single message to both servers. The servers exchange a single field element.

2.2 Applications

Such a protocol is useful for many applications. As we will see, it is also possible to build highly-efficient schemes to compute such statistics in a private manner. The main difficulty in deploying these schemes is in coming up with two servers that (1) have an incentive to run the protocol, and (2) can be trusted not to collude.

- **Telemetry:** Mozilla uses the protocol we'll describe below to collect certain types of telemetry data, e.g., the number of times that a certain tracking-protection rule was triggered in Firefox.[1] Mozilla partnered with ISRG (the Internet Security Research Group) to deploy a second non-colluding server.
- **Public health monitoring:** During the COVID-19 pandemic, Apple and Google deployed exposure notification apps, which notify users if they have been in contact with an infected person. Due to privacy concerns, these systems were designed so that Apple, Google, and other third parties don't learn if users were shown an exposure notification, or who they came in contact with. This information is kept strictly on device. Yet, to track the spread of the virus, it would be useful for health authorities to obtain some global statistics, e.g., the number of people who came in contact with an infected person in a given region. Google and Apple thus deployed a private aggregation protocol to collect such statistics, with data secret-shared between the two companies.[2]

[1] https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry

[2] https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf

2.3 The Integrity Problem

The above protocol provides perfect privacy! But this comes at a significant cost: any user can completely disrupt the *integrity* of the protocol's computation.

Indeed, in the applications above, the inputs from each party are not *arbitrary* field elements. (if they were, would learning their sum really be useful?) Rather, they are typically bounded in some range. For example, for the telemetry use-case, each client's data should be a vector of bits. Or, for Federated Learning, each client's data should be a gradient vector, which is typically bounded in norm.

But our simple protocol from before imposes no limit on a client's inputs, as long as they are in $\mathbb{F}_p$. This means that a single malicious user can arbitrarily disrupt the protocol's result...

That is, an evil user can just set $[x_i]_A + [x_i]_B$ to be any arbitrary vector in $\mathbb{F}_p^d$. Then, the protocol outputs

$$\sum_i x_i + \Delta \,,$$

where $\Delta \in \mathbb{F}_p^d$ is an arbitrary shift.

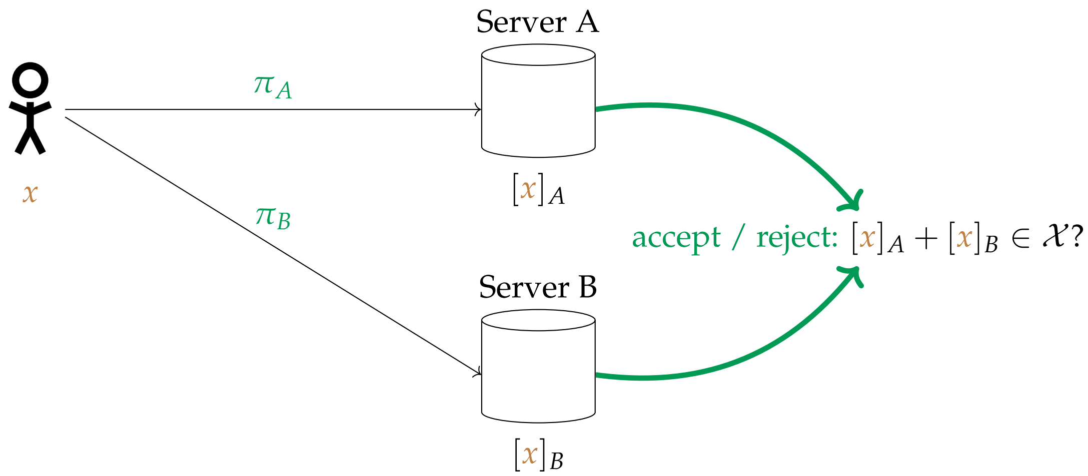
Instead, we might want to add constraints to user inputs, such that we compute

$$\sum_i x_i \quad \text{for } x_i \in \mathcal{X} \subset \mathbb{F}_p^d \,.$$

For example, for telemetry, we might have $\mathcal{X} = \{0,1\}^d$, and for Federated Learning we might have $\mathcal{X} = \{x \in \mathbb{F}_p^d : \|x\| \leq C\}$.[3]

3 ZK Proofs on Secret-Shared Data

We're going to fix this issue with ZK proofs! The client will prove to both servers that their input is well-formed. If the servers trust the proof, they will add the client's shares to their sum, and otherwise reject them.



This proof system differs slightly from the ones we've seen before, in that the statement x is not directly visible to the servers. Instead, the servers can only access the statement

[3] In this case, a malicious client can still send an arbitrary vector in $\mathcal{X}$, but the influence of this input on the final output is bounded.

in secret-shared form. Such proof systems are called *zero-knowledge proofs over secret-shared data* [BBCG+19].

The properties we'll want are analagous to the ones we've seen before:

- **Completeness:** If the client is honest, the servers accept the proof.
- **Soundness:** A malicious client can convince an honest server when $x \notin \mathcal{X}$ with negligible probability.
- **Zero-knowledge:** The servers learn nothing about x, except that it is in $\mathcal{X}$ (unless they collude of course, in which case all privacy is lost).

3.1 Polynomials (Again)

We'll now describe a very cute ZK proof system for secret-shared data, due to [CGB17] and [BBCG+19]. The protocol can be seen as an (extremely) simplified version of the PLONK IOP we saw before. The reason we will get something much simpler than PLONK is due to two factors: (1) there are two non-colluding verifiers, which means we can secret-share data instead of using commitments; (2) the proofs won't actually be succinct.

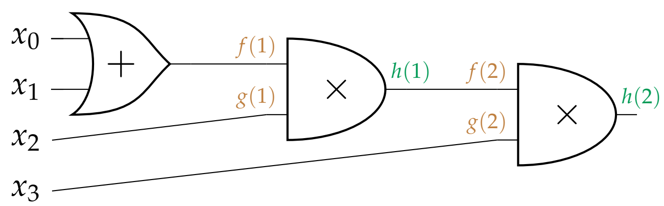
Let $\mathcal{C} : \mathbb{F}_p^d \to \{0, 1\}$ be an arithmetic circuit for testing the validity of an input. That is, $\mathcal{C}$ outputs 0 if and only if $x \in \mathcal{X}$. We thus want a proof that $\mathcal{C}(x) = 0$.

The client (the prover) will start by running the circuit themselves. For each of the T multiplication gates in the circuit, let (u_t, v_t) be the inputs to that gate. Sample u_0 and v_0 randomly from $\mathbb{F}_p$ and define two degree-T polynomials f and g such that

$$f(t) = u_t \quad \text{and} \quad g(t) = v_t \quad \text{for } t \in \{0, \ldots, T\}\,.$$

Then, define the polynomial h as $h := f \cdot g$. Note that h has degree at most $2T$, and that h encodes the *outputs* of each multiplication gate in the circuit, i.e.,

$$h(t) = f(t) \cdot g(t) = u_t \cdot v_t \quad \text{for } t \in \{1, \ldots, T\}\,.$$

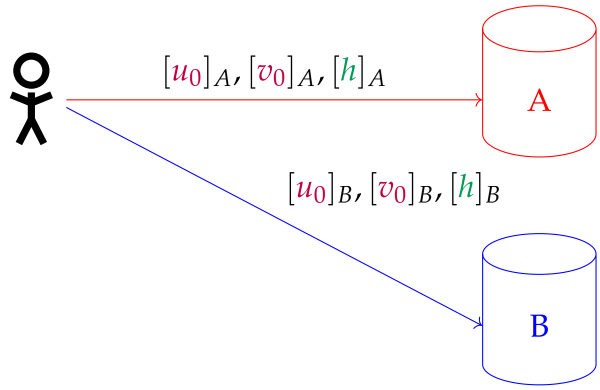


The client then splits u_0, v_0 and h into shares:[4]

$$u_0 = [u_0]_A + [u_0]_B \quad \text{and} \quad v_0 = [v_0]_A + [v_0]_B \quad \text{and} \quad h = [h]_A + [h]_B$$

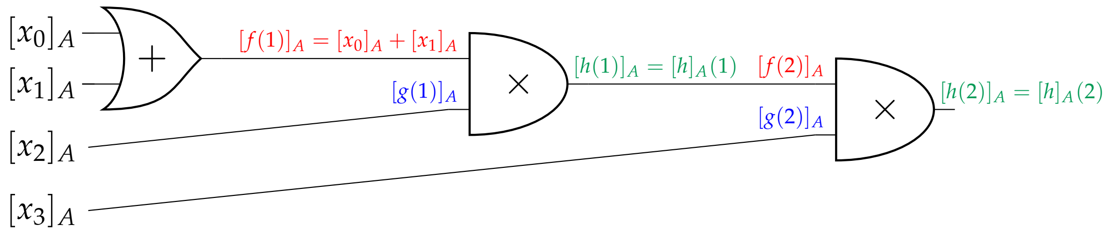
Finally, the client sends the corresponding shares to the two servers.

[4]To secret share a polynomial, we write it out as $h(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots$ and then secret-share the vector of coefficients $[\beta_0, \beta_1, \beta_2, \ldots]$.



The nice observation here is that the servers can reconstruct shares of the polynomials f and g from all the shares they have received, without any communication. First, let's see that the servers can reconstruct shares of the values of all wires in the circuit:

- Server i has a share of each of the input values $[x]_i$
- Server i has a share of each output of a multiplication gate (i.e., $[h]_i(t)$ for $t \in \{1, \ldots, T\}$)
- The servers can calculate shares of all other wires with linear operations on these shares.



Then, given shares of $[f(i)]_A$ and $[g(i)]_A$ for $i \in \{0, 1, \ldots, T\}$, the servers can interpolate polynomial shares $[f]_A, [f]_B$ and $[g]_A, [g]_B$. If all parties acted honestly, the servers end up with shares of polynomials f , g, and h such that $f \cdot g = h$.

It is not too hard to see that if the malicious client sends shares of a polynomial $\hat{h}$ that does *not* interpolate the outputs of all multiplication gates in $C(x)$, then the servers will reconstruct shares of polynomials $\hat{f}$ and $\hat{g}$ for which $\hat{f} \cdot \hat{g} \neq \hat{h}$.

So what's left to do for the servers? They need to check that:

1. They hold shares of polynomials f, g and h such that $f \cdot g = h$.
2. $h(T) = 0$.

Let's start with point (2). Here, each server just evaluates $[h]_i(T)$ locally and shares the value with the other server. Then, they check that $[h]_A(T) + [h]_B(T) = h(T) \stackrel{?}{=} 0$.

For (1), recall the polynomial equality test we used for PLONK: the servers agree on a random value $r \xleftarrow{\$} \mathbb{F}_p$ and then check that $f(r) \cdot g(r) \stackrel{?}{=} h(r)$. Again, the servers can compute shares of $f(r)$, $g(r)$ and $h(r)$ locally, and then reveal the shares to check the equality.

Zero-knowledge. If at least one server is honest, the protocol provides *perfect* zero-knowledge! That is, even a malicious server with unbounded compute power learns nothing about the input x, except that it is in $\mathcal{X}$.

Informally, because we sampled the y-intercept of the polynomials f and g (i.e., u_0 and v_0) at random, the values $f(r)$ and $g(r)$ observed by the servers are uniform in $\mathbb{F}_p$ and independent of the adversary's view. The other values seen by the adversary are additive secret shares (which are information-theoretically hiding), and the value $h(T) = 0$ which reveals nothing more than the fact that $x \in \mathcal{X}$.

Formalizing this argument takes a bit more work. One subtlety is that zero-knowledge only holds if the challenge $r \notin \{1, \ldots, T\}$ (otherwise, releasing $f(r)$ and $g(r)$ leaks information about the input x). If at least one server is honest, we can ensure this doesn't happen.

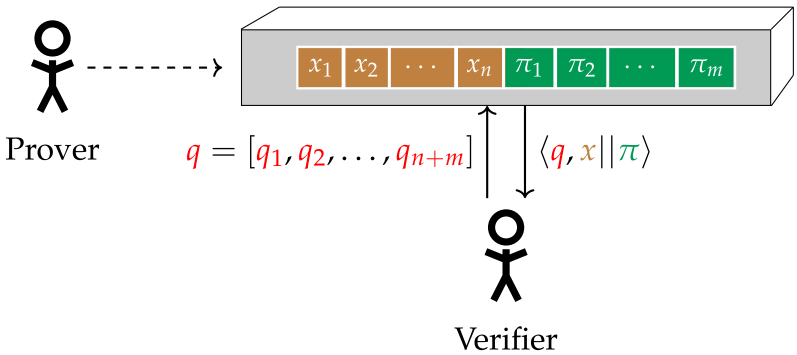
Complexity. What's the complexity of this proof system? The proof π is of size $O(T)$, which is not succinct in general (i.e., we can have $T = O(|\mathcal{C}|)$). Getting around this is possible with more involved (multi-round) protocols.

The server computation complexity to verify the proof is also $O(T)$. But what's really nice is that the servers only need to exchange $O(1)$ field elements per proof!

3.2 Generalization: (Fully)Linear PCPs

The protocol above is a case of a more general class of proof systems: fully linear PCPs [BCI+13, BBCG+19]. Recall that in the standard PCP theorem, we show that given a statement x and proof π, a verifier can check the proof by querying it at three random positions.

There are other PCPs where the verifier instead makes only *linear* queries to the proof string, concatenated with the statement. That is, given a statement $x \in \mathbb{F}_p^n$ and proof $\pi \in \mathbb{F}_p^m$, the verifier can ask queries of the form $q \in \mathbb{F}_p^{n+m}$ and receive the inner product $\langle q, x||\pi \rangle \in \mathbb{F}_p$. They then accept or reject the proof based solely on such inner products.



Linear PCPs are neat for a number of reasons. First, they can be implemented much more efficiently in practice compared to regular PCPs. In fact, our polynomial check above is a linear PCP in disguise! The statement is the input $x \in \mathbb{F}_p^d$, and the proof π consists of the coefficients of the polynomials f, g and h (we actually only need the first coefficients of f and g as we saw above). Then, the linear PCP verifier checks that $f \cdot g = h$ and $h(T) = 0$, with requires only linear operations over x and the proof. Overall, we can represent the verifier checks as four linear queries q_i to a proof of size O(T) where T is the number of multiplication gates in the validation circuit.

The second reason why linear PCPs are neat is that, in cryptography, there are many ways to "hide" data such that linear operations over the hidden data are possible (e.g., linearly homomorphic commitments or encryption, and additive secret sharing)!

So, given any (zero-knowledge) linear PCP, we immediately get a (zero-knowledge) proof over secret-shared data! The prover sends additive shares of x and π to the verifiers, who evaluate the same random linear PCP queries locally.[5] They then reveal their shares of the outputs to reconstruct the inner products $\langle q_i, x||\pi\rangle$ and run the linear PCP verifier.

This view is actually very useful, because there are many different linear PCPs in the literature. By adding these to our private aggregation protocol, we can get protocols with different tradeoffs in communication and computation, for different types of circuits $\mathcal{C}$.

For example, if the verification circuit $\mathcal{C}$ has some special structure (e.g., a sub-circuit that is replicated many times), then it is possible to build linear PCPs (and thus a private statistics protocol) with better communication complexity.

Moreover, if we allow for interaction between provers and verifiers (i.e., a linear *IOP* [BCI+13]), we can get succinct proofs for arbitrary circuits.

References

[BBCG+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In *Annual International Cryptology Conference*, pages 67–97. Springer, 2019.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 315–333. Springer, 2013.

[CGB17] 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)*, pages 259–282, 2017.

[5]We assume here that the servers have some shared randomness, e.g., a shared key for a pseudorandom number generator (PRG).