Secret Sharing and its Applications

Notes by Henry Corrigan-Gibbs

MIT - 6.5610
Lecture 18 (April 19, 2023)

> ***Warning:*** This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Outline

- Definition of Secret Sharing
- Additive secret sharing
  - Application: Private information retrieval
- *Stretch break*
- Shamir's secret-sharing scheme
  - Application: Multiparty computation

Secret sharing

Note: This section is drawn in large part from Dima Kogan's notes from Stanford's CS355.

Imagine that you own a jewelry store with a gigantic safe in the back room. You want to give the code to the safe to your employees so that if something happens to you, they can open up the safe and keep the store running.

You have two conflicting goals:

- You want to make sure that your employees really are able to open the safe, even if some of them don't show up to work, forget whatever secret they were supposed to remember, drop their laptop in the lake, etc.
- You want to make sure that no single rogue employee can open the safe on their own and take off with the jewels.

Equally important: You want to make sure that no one can coerce a single employee into opening the safe. People who handle cryptocurrency secret keys worry about that threat.

To ensure availability, it seems like you should give the code to each of your employees. But that is problematic for security, since any employee can make off with the jewels. Or, you could encrypt the code in such a way that *all* of your employees need to participate to recover it, but that is bad for availability. (If one of them disappears, the rest cannot open the safe.)

Secret-sharing schemes give a very elegant solution to this type of problem. The goal of a secret-sharing scheme is to split a secret value k into many *shares* with two properties:

- **Correctness.** Anyone who holds at least t shares can recover the secret k.
- **Security.** An attacker who sees fewer than some threshold number of shares t learns no information about the secret k, and

In the cryptocurrency, we are typically protecting signing keys, rather than encryption keys. For that application, there are special-purpose *threshold signing* schemes that require multiple signers to agree to sign a message before the group can produce a signature that validates under the group's public key.

We use k to denote the secret, since in practice often the secret is the secret key for another cryptosystem.

Definition

For integers t, n with $t \leq n$, a t-out-of-n secret-sharing scheme over a set Σ is a pair of efficient algorithms $(\mathsf{Gen}, \mathsf{Recover})$ with the following syntax:

- $\mathsf{Gen}(k) \to (s_1, \ldots, s_n)$ is a randomized algorithm that takes as input a secret $k \in \Sigma$ and outputs n shares.
- $\mathsf{Recover}(s_{i_1}, \ldots, s_{i_t}) \to k$ takes as input t shares and outputs the secret k.

Without loss of generality, we can assume that each share includes the "identity" or index of the share, since we can append a unique identifier to each share with only an additional $\lceil \log_2 n \rceil$ bits.

Shamir gave one of the first secret-sharing schemes in his 1979 paper [5].

The pair $(\mathsf{Gen}, \mathsf{Recover})$ must satisfy the following two properties:

Definition 1 (Secret sharing, Correctness)**.** For all $k \in \Sigma$, for all $(s_1, \ldots, s_n) \leftarrow \mathsf{Gen}(k)$, and for all size-$t$ subsets $\{s_{i_1}, \ldots, s_{i_t}\}$ of the shares, $k = \mathsf{Recover}(s_{i_1}, \ldots, s_{i_t})$.

A stronger correctness property would require that the Recover algorithm outputs the correct secret, even when some of the shares it is given are corrupted. Standard secret-sharing schemes can satisfy this stronger notion of correctness; this stronger security property is useful in many applications. We will not discuss those extensions here.

The security definition is similar to our notion of one-time security for a secret-key encryption scheme, or semantic security for a public-key encryption scheme. The idea is that any size-$(t-1)$ subset of the shares should "look the same," whether they are shares of secret k or a different secret k':

Definition 2 (Secret sharing, Security)**.** For all $k, k' \in \Sigma$, for all subsets $T \subset \{1, \ldots, n\}$ of size $t-1$,

$$\{\mathsf{Gen}(k)[T]\} \approx_c \{\mathsf{Gen}(k')[T]\},$$

where $\mathsf{Gen}(k)[T] := \{s_i \colon (s_1, \ldots, s_n) \leftarrow \mathsf{Gen}(k) \mid i \in T\}$.

If the two distributions here are identical, we say that the secret-sharing scheme has *perfect* or *information theoretic* security.

Additive secret sharing

One of the simplest (and surprisingly useful!) secret-sharing scheme is additive secret sharing.

Construction

This is an n-out-of-n secret sharing scheme over any finite group, such as the integers modulo p with addition.

- $\mathsf{Gen}(k) \to (s_1, \ldots, s_n)$. Sample n independent random values $s_1, \ldots, s_n \leftarrow \mathbb{Z}_p$ that are uniform subject to $k = \sum_{i=1}^{n} s_i \bmod p$.
- $\mathsf{Recover}(s_1, \ldots, s_n) \to k$. Return $k \leftarrow \sum_{i=1}^{n} s_i \bmod p$.

It is often useful to secret-share vectors or matrices of elements modulo p.

The practical way to implement this sampling procedure is to first pick $n-1$ of the shares $(s_1, \ldots, s_{n-1})$ independently and uniformly at random and then set $s_n \leftarrow (k - \sum_{i=1}^{n-1} s_i) \bmod p$.

Correctness holds by construction.

Security holds since any $n-1$ shares are independently distributed uniform random values modulo p. Additive secret sharing is perfectly secure—it is secure even without making computational assumptions.

Feature: Applying functions to secret-shared data. One neat property of the additive secret-sharing scheme is that it is possible to apply linear

functions to secret-shared data. That is, if parties hold shares of a secret k, they can compute shares of $f(k)$ for any linear function $f(\cdot)$.

Recall that a linear function is one that can just add and scale elements by constants.

Let us look at this in more detail.

Let $k \in \mathbb{Z}_p^d$ be a vector of dimension d modulo a fixed prime p. Then additive shares of k are vectors $s_1, s_2 \in \mathbb{Z}_p^d$ such that

$$k = s_1 + s_2 \quad \in \mathbb{Z}_p^d.$$

We can represent any linear function $f\colon \mathbb{Z}_p^d \to \mathbb{Z}_p^d$ as a matrix $M \in \mathbb{Z}_p^{d \times d}$.

Then observe that if we reconstruct the values of $M \cdot s_1$ and $M \cdot s_2$, they reconstruct to $M \cdot s$:

$$\begin{aligned} \mathsf{Recover}(M \cdot s_1, M \cdot s_2) &= M \cdot s_1 + M \cdot s_2 \\ &= M \cdot (s_1 + s_2) \\ &= M \cdot s \qquad\qquad\qquad \in \mathbb{Z}_p^d. \end{aligned}$$

So if Alice and Bob each hold shares of s, they can locally "convert" them into shares of $M \cdot s$, where M is any public matrix.

Here, we relied on the fact that the Recover algorithm for additive secret sharing is *linear*. Not all secret-sharing schemes have this nice linearity property, but many do, including Shamir's scheme that we will see in a moment.

Application: Private information retrieval

We will now show one simple (and yet powerful!) application of additive secret sharing.

A two-server *private information protocol* (PIR) takes place between:

- two *servers*, each holding an B-bit database $D = (D_1, \ldots, D_B)$, and
- a *client*, holding an index $i \in \{1, \ldots, B\}$.

The client's goal is to fetch the ith bit of the database from the servers, without revealing which bit it fetched. The client could of course download the entire B-bit data from one of the servers, but this is costly in communication. Can the client privately fetch the ith database bit while using $o(B)$ bits of communication?

In a two-server PIR scheme, we want privacy to hold against an adversary that controls at most one of the two servers. The security goal is similar to that of CPA security:

Definition 3 (PIR Security, informal)**.** A two-server PIR scheme for B-bit databases is secure if, for all $i_0, i_1 \in \{1, \ldots, B\}$ and all $\sigma \in \{1, 2\}$:

$$\left\{ \begin{array}{c} \text{Client's query to server } \sigma \\ \text{on input } i_0 \end{array} \right\} \equiv \left\{ \begin{array}{c} \text{Client's query to server } \sigma \\ \text{on input } i_1 \end{array} \right\}.$$

Here, we assume that the protocol consists of the client sending one message to each server. Essentially all existing PIR schemes have this format.

PIR from secret sharing. The client and servers write the database as a matrix $D \in \mathbb{Z}_2^{\sqrt{B} \times \sqrt{B}}$. We work over $\mathbb{Z}_2$, though the choice of group does not matter much here. The client wants the database bit in entry (i, j) of the matrix.

Assume that B is a perfect square, or round it up to one if not.

- The client constructs a query vector $q \in \mathbb{Z}_2^d$ that is zero everywhere except with a "1" in position j—the index of the column in which the client's desired database bit lies.
- The client splits its vector q into additive shares $q_1, q_2 \in \mathbb{Z}_2^d$ and sends one share to each server. This is the only message that the client sends to the servers, so security follows immediately from the security of additive secret sharing.
- Each server applies the matrix D to their shares and returns the result to the client.
- The client reconstructs $Dq_1 + Dq_2 = Dq$. Since q is zero everywhere except at the jth index, the client recovers the entire jth column of the database matrix D.

Finally, the total communication cost of the protocol is $4\sqrt{B}$ bits: $\sqrt{B}$ up and down to each server. To me, this seems like magic! The client communicated $\sqrt{B} \ll B$ bits with the servers and yet it managed to "fish out" an entire database column without revealing which one it received.

Even if B is not a perfect square, the communication cost is still $O(\sqrt{B})$.

The best known upper bound on the communication complexity of two-server PIR schemes is $B^{O(\sqrt{\frac{\log \log B}{\log B}})}$, due to Dvir and Gopi [4]. For comparison, the scheme we just gave has communication complexity $O(B^{1/2})$ The Dvir-Gopi scheme, asymptotically at least, has much much much smaller communication complexity than the scheme that we just gave. In their scheme, the exponent on B is goes to zero as $B \to \infty$, while ours is a constant $1/2$.

One of the great open problems in this area is to construct a two-server information-theoretic PIR scheme with better communication complexity. We know that all schemes require $\Omega(\log B)$ bits—you need $\log B$ bits just to express which index in the database you want—but are there schemes that match this lower bound?

If one-way functions exist, there are two-server PIR protocols with communication complexity $O(\log B)$ (hiding polynomials in the security parameter) [2]. And under a variety of public-key assumptions [3], *single*-server PIR schemes with polylog(B) communication complexity exist.

Shamir's secret-sharing scheme

The additive secret-sharing scheme does not solve the jeweler's problem from the start of lecture. The problem is that additive secret sharing is an n-out-of-n secret-sharing scheme, where the jeweler needs a t-out-of-n secret-sharing scheme for $t \approx n/2$.

A combinatorial solution with exponentially many shares. One naïve way to build a t-of-n secret sharing scheme is to additively secret share the secret key to each size-t subset of the n parties. That scheme can have an *exponential* number of shares, since

$$\binom{n}{t} \geq \left(\frac{n}{t}\right)^t,$$

which is superpolynomial for many interesting choices of t, such as $t = \Theta(n)$.

Shamir gave a secret-sharing scheme that has polynomial share size for any threshold t. The key idea is to use polynomials: Two points determine a line, but one point does not. Three points determine a parabola, but two do not. More generally, any t points completely determine a polynomial of degree at most $t-$, but with only $t-1$ points, you cannot recover the entire polynomial.

This holds not only over the reals and complex numbers (as you may have seen before), but also modulo any prime p. More generally, this statement holds over any *field*.

We will work modulo a prime $p > n$.

- $\mathsf{Gen}(k) \to (s_1, \ldots, s_n)$
  - Choose a random polynomial f of degree $\leq t-1$ with coefficients modulo p, subject to the constraint that $f(0) = k \bmod p$.
  - For $i = 1, \ldots, n$, let $s_i \leftarrow (i, f(i) \bmod p)$.

We can sample the polynomial by setting its constant term to k and then choosing independent uniform random values modulo p for the other $t-1$ coefficients.

- $\mathsf{Recover}(s_{i_1}, \ldots, s_{i_t}) \to k$.
  - Parse the shares as $(x_1, y_1), \ldots, (x_t, y_t)$.
  - Interpolate the unique polynomial f' (modulo p) that passes through the points that the shares define.
  - Output $f'(0) \bmod p$.

Correctness. We need to show that show that interpolation succeeds and that the polynomial f' that we recover is equal to the original polynomial f that the sharer used to generate the shares.

To interpolate, we just need to solve the following linear system, where the indeterminates are the coefficients of the polynomial f':

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{t-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & x_t^2 & \cdots & x_t^{t-1} \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{t-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_t \end{pmatrix}$$

That the polynomial f' is unique boils down to arguing that the matrix on the left is invertible. (Equivalently, that there is a unique solution to the linear system.) When $(x_1, \ldots, x_t)$ are distinct in $\mathbb{Z}_p$, a matrix of this form is called a Vandermonde matrix over $\mathbb{Z}_p$. A fact of life is that, for prime p, such a matrix is invertible.

We can instantiate the PIR scheme we described earlier with Shamir's secret-sharing scheme. This gives a PIR scheme for n servers where the client only needs responses from t servers (so it is fault tolerant in that sense) and that can tolerate the compromise of up to $t-1$ of the servers.

The polynomial f' that we recover is equal to the original polynomial f at t points. Since these polynomials have degree at most $t-1$, they must then be equal everywhere. It follows that $f'(0) = k$.

Security. Say that the adversary gets her hands on $t-1$ of the shares. Then for every possible value of the secret k, there is exactly one polynomial f' that passes through the shares such that $f'(k)$. So the adversary has no information on the secret.

Application: Secure multiparty computation

In a *secure multiparty computation*, there are a number of parties, each with a private input. The parties want to jointly compute some (public) function of their private inputs. For example, think of ten people who all want to compute the average of their salaries without leaking "anything else" about their salaries, except the average.

This is actually not such a hypothetical example! The Boston Women's Workforce Council has apparently used this type of multiparty computation to compute the average gender pay gap across businesses in Boston. See: `https://thebwwc.org/mpc`.

We do not have time to define secure multiparty computation formally, and the definitions get hairy pretty quickly. So we will stick with the informal one for now.

To start out, say that we have n parties, where the ith party holds x_i. The parties want to compute a *linear* function $f(x_1, \ldots, x_n)$ of their inputs. The protocol is:

- Each party secret-shares their input using t-of-n secret sharing.
- Each party sends one share to every other party.
- Each party locally applies the function f to their shares.
- Each party publishes their share of the output.

The linearity of the secret-sharing scheme means that the parties reconstruct $f(x_1, \ldots, x_n)$ as desired.

You might wonder the parties do if they want to compute a *nonlinear* function of their secret inputs. The basic idea is to write out the function as a circuit, made up of addition and multiplication codes (modulo p). Using the linearity of the secret-sharing scheme, the parties can compute, in a distributed sense, the shares of the output of any addition gate given shares of the inputs. For multiplication gates, things are more complicated [1]. I may sketch the idea if we have time.

If there is time, we will also discuss secure aggregation.

References

[1] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Symposium on the Theory of Computing (STOC)*, pages 1–10, 1998.

[2] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[3] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, 1999.

[4] Zeev Dvir and Sivakanth Gopi. 2-server pir with subpolynomial communication. *Journal of the ACM*, 63(4):1–15, 2016.

[5] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.