COS 330: Great Ideas in Theoretical Computer Science

Fall 2025

Lecture 21

Polynomials and Cryptography

Module: Information and Codes

1 Introduction

How can a group of people jointly compute a function of their private inputs without revealing those inputs to each other? For example, suppose n students each hold a private value s_i (say, their salary), and they wish to compute the class average

$$\frac{1}{n}\sum_{i=1}^{n} s_i$$

without anyone revealing s_i to anyone else—not even to the instructor. At first, this sounds impossible: how can you compute a sum without knowing the individual terms?

The answer comes from an elegant interplay between algebra and cryptography. We will build up the necessary tools step by step:

- First, we introduce *finite fields*, a mathematical structure that provides exact arithmetic in a finite setting, avoiding the pitfalls of floating-point computation.
- Next, we study *polynomials over finite fields* and establish key properties about their degrees and roots. These properties mirror those of polynomials over the reals but work in our finite setting.
- We then apply these polynomial tools to construct *Shamir's secret sharing scheme*, which splits a secret into shares distributed among multiple parties such that only sufficiently large coalitions can reconstruct the secret.
- Finally, we combine these ideas to design a simple protocol for *secure multiparty computation* (MPC), allowing multiple parties to jointly compute sums and averages of their private inputs without revealing individual values.

This lecture demonstrates how abstract mathematical structures can solve concrete security problems. The techniques we develop here form the foundation for modern cryptographic protocols used in secure auctions, private data analysis, and distributed trust systems.

2 Finite Fields

Why can't we just work with real numbers? After all, when we compute an average like $(s_1 + \cdots + s_n)/n$, we naturally think of the s_i's as real numbers. The problem is that real-number arithmetic on computers leads to rounding and precision errors (for instance, $1/3 \approx 0.33333\ldots$), and computers have finite memory, so we cannot represent arbitrary real numbers exactly. For cryptographic constructions where security depends on algebraic properties, such imprecision is unacceptable.

We need a number system that is:

- **finite** (so every element fits in a fixed number of bits),
- **exact** (no rounding errors), and
- **algebraically complete** (we can always divide by nonzero elements).

This leads us to work in a special kind of mathematical structure called a *field*. Informally, a field is a set equipped with two operations (addition and multiplication) where you can add, subtract, multiply, and divide (by nonzero elements), and all the familiar algebraic rules hold: commutativity, associativity, distributivity, existence of identities (0 and 1), and existence of additive and multiplicative inverses.

Remark 2.1. The real numbers $\mathbb{R}$ form a field: you can add, subtract, multiply, and divide any real numbers (except dividing by zero), and the usual algebraic rules apply. However, $\mathbb{R}$ is not suitable for computation because it is infinite and elements cannot be represented exactly.

Why do we care about working in a field? Fields have many nice properties that make them useful. In particular, fields support the full machinery of linear algebra: you can solve systems of equations, perform matrix operations, and work with vector spaces. All the linear algebra tools you know from working over $\mathbb{R}$ continue to work over any field. As we saw in our lectures on error-correcting codes, *finite fields* (fields with finitely many elements) with these properties can be extremely useful.

We now construct a specific family of finite fields that will serve as our "sandbox" for cryptography.

Definition 2.1 (The integers modulo p)**.** Let p be a prime number. The set of integers modulo p, denoted $\mathbb{Z}_p$, is

$$\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}.$$

We define two operations on this set:

- **Addition modulo p:** $(a + b) \bmod p$ is the remainder when $a + b$ is divided by p.
- **Multiplication modulo p:** $(a \times b) \bmod p$ is the remainder when $a \times b$ is divided by p.

Remark 2.2 (Example: arithmetic in $\mathbb{Z}_7$). Let's see how addition and multiplication work in $\mathbb{Z}_7 = \{0, 1, 2, 3, 4, 5, 6\}$:

$$5 + 4 \equiv 9 \equiv 2 \pmod 7, \qquad 3 \times 5 \equiv 15 \equiv 1 \pmod 7.$$

Here we will *always* assume the modulus p is **prime**. This is crucial to make division work.

The key fact that makes $\mathbb{Z}_p$ into a field is that when p is prime, every nonzero element has a multiplicative inverse.

Theorem 2.1 (Existence and uniqueness of inverses)**.** If p is prime, then for every nonzero $a \in \mathbb{Z}_p$ there exists a unique $b \in \mathbb{Z}_p$ such that

$$a \cdot b \equiv 1 \pmod{p}.$$

We call b the *multiplicative inverse* of a and denote it by a^{-1}. We interpret "division by a" as multiplication by a^{-1}.

The proof of this theorem uses some basic number theory (specifically, Bézout's identity and the fact that $\gcd(a, p) = 1$ when p is prime and $a \neq 0$), which is beyond the scope of this course. The key takeaway is that division always works in $\mathbb{Z}_p$ when p is prime.

Remark 2.3 (Example: division in $\mathbb{Z}_7$). In the real numbers, $1/4 = 0.25$, which is not an integer. In $\mathbb{Z}_7$, division is cleaner: we look for x such that $4x \equiv 1 \pmod 7$. Checking small values,

$$4 \cdot 1 = 4, \quad 4 \cdot 2 = 8 \equiv 1 \pmod{7}.$$

So in $\mathbb{Z}_7$, the inverse of 4 is 2, and we can think of

$$\frac{1}{4} \equiv 2 \pmod{7}.$$

Similarly, to compute $3/4$ in $\mathbb{Z}_7$, we compute $3 \times 4^{-1} = 3 \times 2 = 6$.

Because every nonzero element of $\mathbb{Z}_p$ has a multiplicative inverse (by Theorem 2.1), and addition and multiplication satisfy all the usual algebraic rules, we conclude that $\mathbb{Z}_p$ with these two operations is a field. We call this field the *prime field of order* p and denote it by $\mathbb{F}_p$.

Remark 2.4 (Finite fields beyond primes). There exist finite fields with p^m elements for any prime p and positive integer m. However, when $m > 1$, the field $\mathbb{F}_{p^m}$ is *not* simply $\mathbb{Z}_{p^m}$ with addition and multiplication modulo p^m. (For instance, $\mathbb{Z}_4$ is not a field because 2 has no multiplicative inverse: $2 \times 2 \equiv 0 \pmod 4$.) Constructing $\mathbb{F}_{p^m}$ for $m > 1$ requires polynomial arithmetic and other tools from abstract algebra that are beyond the scope of this course.

For all our applications in this lecture, we will only need prime fields $\mathbb{F}_p$ where p is prime.

3 Polynomials over Finite Fields

Now that we have a clean number system, we can talk about polynomials whose coefficients and variable both live in $\mathbb{F}_p$.

Definition 3.1 (Polynomial over $\mathbb{F}_p$)**.** A (univariate) polynomial of degree d over $\mathbb{F}_p$ is an expression

$$P(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0,$$

where each coefficient $a_i \in \mathbb{F}_p$, and $a_d \neq 0$. The degree of P is d. If all coefficients are 0, we call P the *zero polynomial*.

We can evaluate such a polynomial at any $x \in \mathbb{F}_p$, using only operations in $\mathbb{F}_p$ (so everything is done modulo p).

Definition 3.2 (Root)**.** An element $r \in \mathbb{F}_p$ is called a *root* (or *zero*) of P if

$$P(r) = 0 \quad \text{in } \mathbb{F}_p.$$

Surprisingly (or maybe not, if you remember high school algebra), polynomials over $\mathbb{F}_p$ satisfy the same basic "degree vs. roots" relationship that polynomials over $\mathbb{R}$ do.

Theorem 3.1 (Roots are bounded by degree)**.** Let P be a nonzero polynomial over $\mathbb{F}_p$ of degree $d \geq 0$. Then P has at most d distinct roots in $\mathbb{F}_p$.

Proof. We prove this by induction on the degree d.

Intuition. A line ($d = 1$) can cross the x-axis at most once. For higher degrees, whenever you find a root r, you can factor out $(x - r)$ and reduce the degree by 1. You cannot do this more than d times.

Base case ($d = 0$). If $d = 0$, then $P(x) = c$ for some $c \neq 0$, which clearly has no roots. So the statement holds: P has $0 \leq 0$ roots.

Inductive step. Assume the statement holds for all polynomials of degree $d - 1$, and let P be a polynomial of degree $d \geq 1$.

If P has no roots, then we are done (it has $0 \leq d$ roots). So assume P has at least one root r with $P(r) = 0$.

Because $\mathbb{F}_p$ is a field, polynomial long division works: there exists a polynomial Q over $\mathbb{F}_p$ of degree $d - 1$ such that

$$P(x) = (x - r)Q(x).$$

Now let s be any root of P. Then

$$0 = P(s) = (s - r)Q(s).$$

Since $\mathbb{F}_p$ is a field, it has no zero divisors: a product is 0 only if one of the factors is 0. Thus either $s = r$ or $Q(s) = 0$.

This shows that every root of P is either r itself, or a root of Q. By the inductive hypothesis, Q has at most $d - 1$ distinct roots. Therefore, P has at most $1 + (d - 1) = d$ distinct roots. ■

A very important consequence is that a low-degree polynomial is completely determined by its values at a small number of points.

Theorem 3.2 (Lagrange interpolation)**.** Let $x_0, \ldots, x_d$ be $d + 1$ distinct elements of $\mathbb{F}_p$ and let $y_0, \ldots, y_d$ be arbitrary elements of $\mathbb{F}_p$. Then there exists a *unique* polynomial P over $\mathbb{F}_p$ of degree at most d such that

$$P(x_i) = y_i \quad \text{for all } i = 0, \ldots, d.$$

Proof. **Idea.** We want a polynomial that hits y_i at each x_i. We build "basis" polynomials L_i that are 1 at x_i and 0 at all other x_j, then mix them with the y_i's.

Step 1: Build basis polynomials. For each $i \in \{0, \ldots, d\}$ define

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{d} \frac{x - x_j}{x_i - x_j},$$

where the division by $(x_i - x_j)$ is allowed because $x_i \neq x_j$ and we are in a field (so every nonzero element has an inverse).

Each L_i is a polynomial of degree at most d.

Step 2: Check the special values of L_i. If we plug in $x = x_i$, each factor becomes

$$\frac{x_i - x_j}{x_i - x_j} = 1,$$

so $L_i(x_i) = 1$.

If we plug in $x = x_k$ with $k \neq i$, one of the factors in the product becomes

$$\frac{x_k - x_k}{x_i - x_k} = 0,$$

so $L_i(x_k) = 0$.

Step 3: Combine them with the right weights. Now define

$$P(x) = \sum_{i=0}^{d} y_i \cdot L_i(x).$$

This is still a polynomial of degree at most d (sum of degree-$\leq d$ polynomials).

For each k,

$$P(x_k) = \sum_{i=0}^{d} y_i \cdot L_i(x_k) = y_k \cdot 1 + \sum_{i \neq k} y_i \cdot 0 = y_k.$$

So P has the desired values.

Step 4: Uniqueness. Suppose Q is another degree-$\leq d$ polynomial with $Q(x_i) = y_i$ for all i. Let $R(x) = P(x) - Q(x)$. Then R has degree at most d, and $R(x_i) = 0$ for each i, so R has $d+1$ distinct roots. By Theorem 3.1, a nonzero polynomial of degree $\leq d$ can have at most d roots, so R must be the zero polynomial and $P = Q$. ■

Remark 3.1 (Computational efficiency). The Lagrange interpolation method described in the proof above requires computing $d + 1$ basis polynomials $L_i(x)$, each involving d multiplications and divisions. Evaluating the resulting polynomial $P(x)$ at any point takes an additional d operations. Overall, this direct approach has time complexity $O(d^2)$.

For applications requiring frequent interpolation (such as error correction or large-scale cryptographic protocols), this quadratic cost can be significant. Fortunately, there exist faster algorithms based on the Fast Fourier Transform (FFT) that can perform polynomial interpolation and evaluation in $O(d \log d)$ time, but they are a bit more involved and beyond the scope of this course.

4 Shamir's Secret Sharing

We have built up a toolkit of polynomial operations over finite fields. Now we put these tools to work. Our first application is *secret sharing*: a way to split a secret value S into n pieces (called *shares*) and distribute them to n users, such that:

- any sufficiently large group (say, k or more users) can reconstruct S, but
- any smaller group (fewer than k users) learns *nothing* about S.

Why is this useful? In many settings, we want to avoid single points of failure: if a cryptographic key is stored in one location, that location's compromise or unavailability breaks the entire system. But simply replicating the key in n locations is also dangerous: now any single compromise leaks everything. Secret sharing resolves this by ensuring that compromising fewer than k locations reveals nothing, while any k locations suffice to recover the key.

Remarkably, unlike most cryptographic schemes (which rely on computational assumptions like the hardness of factoring or discrete logarithms), Shamir's secret sharing achieves *information-theoretic security*: even an adversary with unlimited computational power learns nothing from fewer than k shares. The security comes purely from the mathematics of polynomials and randomness, not from any unproven complexity assumptions.

Definition 4.1 ((k, n)-threshold secret sharing scheme)**.** A (k, n)-*threshold secret sharing scheme* consists of a sharing algorithm and a reconstruction algorithm with the following properties:

There is a secret $S \in \mathbb{F}_p$ to be shared among n users, labeled $1, 2, \ldots, n$.

- **Correctness:** Any set of k or more users can reconstruct S from their shares.
- **Perfect secrecy:** Any set of fewer than k users learns *no information* about S beyond what they knew beforehand.

The key insight, due to Adi Shamir (1979), is to leverage the polynomial interpolation theorem (Theorem 3.2). The secret S will be encoded as the constant term of a random degree-$(k-1)$ polynomial $P(x)$ over $\mathbb{F}_p$. Each user i receives the share $\sigma_i = P(i)$. By Theorem 3.2, any k users can interpolate P from their shares and recover $S = P(0)$. The perfect secrecy property—that fewer than k shares reveal nothing—will follow from the randomness of the polynomial.

Definition 4.2 (Shamir's (k, n)-threshold scheme)**.** Fix a prime p with $p > n$, and work in $\mathbb{F}_p$. Let k be an integer with $1 \leq k \leq n$.

- **Sharing (dealer phase).**
  1. The dealer chooses coefficients $a_1, \ldots, a_{k-1}$ independently and uniformly at random from $\mathbb{F}_p$.
  2. The dealer sets $a_0 = S$ (the secret), and defines the polynomial
  $$P(x) = a_{k-1}x^{k-1} + \cdots + a_1 x + a_0.$$
  3. For each user $i \in \{1, \ldots, n\}$, the dealer computes the share
  $$\sigma_i = P(i) \in \mathbb{F}_p,$$
  and privately gives σ_i to user i.
- **Reconstruction.** Any set of k users, say with indices $i_1, \ldots, i_k$, pool their shares. They know the k pairs
$$(i_1, \sigma_{i_1}), \ldots, (i_k, \sigma_{i_k}),$$
and use Lagrange interpolation (Theorem 3.2 with $d = k-1$) to recover the unique degree-$(k-1)$ polynomial P that passes through these points. They then output
$$S = P(0).$$

Theorem 4.1 (Correctness of Shamir's scheme)**.** In Shamir's (k, n)-threshold scheme, any set of k or more users can always reconstruct the secret S.

Proof. **Idea.** The secret is $P(0)$, where P is a polynomial of degree at most $k-1$. Any k distinct points on the graph of such a polynomial determine P uniquely.

Step 1: P has degree $\leq k-1$. By construction,

$$P(x) = a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$$

has degree at most $k-1$.

Step 2: k users know k distinct points. Any k users with indices $i_1, \ldots, i_k$ collectively know the k points

$$(i_1, P(i_1)), \ldots, (i_k, P(i_k)).$$

The x-coordinates $i_1, \ldots, i_k$ are distinct integers in $\{1, \ldots, n\}$.

Step 3: Use interpolation. By Theorem 3.2 with $d = k-1$, there is a unique degree-$\leq k-1$ polynomial passing through these k points. That polynomial must be exactly the original $P(x)$, because P also has degree $\leq k-1$ and matches all these values.

Thus, the users can reconstruct $P(x)$ and evaluate $P(0) = S$. ■

The more interesting part is privacy: fewer than k shares should tell you *nothing* about S.

Theorem 4.2 (Perfect secrecy)**.** In Shamir's (k, n)-threshold scheme, any set of fewer than k users learns no information about the secret S.

More precisely: for any set of $t < k$ users and any shares they might observe, every possible secret $S' \in \mathbb{F}_p$ is equally likely given their shares.

Proof sketch. Consider $t < k$ users who observe shares $\sigma_{i_1}, \ldots, \sigma_{i_t}$. These shares tell them that the secret polynomial P passes through the t points $(i_1, \sigma_{i_1}), \ldots, (i_t, \sigma_{i_t})$.

Now fix any candidate secret $S' \in \mathbb{F}_p$. We need to show that there exists a valid polynomial (one that the dealer might have chosen) that:

- produces exactly the shares $\sigma_{i_1}, \ldots, \sigma_{i_t}$, and
- has secret $P(0) = S'$.

By Theorem 3.2, there exists a unique polynomial of degree at most t passing through the $t+1$ points

$$(0, S'), \quad (i_1, \sigma_{i_1}), \ldots, (i_t, \sigma_{i_t}).$$

Since $t < k$, we have $t \leq k-1$, so this polynomial has degree at most $k-1$. Thus it is a valid choice the dealer might have made (a random degree-$(k-1)$ polynomial with constant term S').

Because the dealer chooses the coefficients $a_1, \ldots, a_{k-1}$ uniformly at random, each such polynomial (for each possible secret S') is equally likely. Therefore, from the users' perspective, every secret S' is equally likely, and they learn nothing about S. ■

Remark 4.1 (Geometric picture for $k = 2$). When $k = 2$, the secret is the y-intercept of a random line in the plane over $\mathbb{F}_p$. Each user receives one point on this line.

- Any *two* users know two points, which determine a unique line, and hence a unique intercept S.

- Any *one* user knows only one point. There are p different lines of slope $m \in \mathbb{F}_p$ passing through that point, one for each possible slope. These lines have p different intercepts S' and all are equally likely. So a single user learns nothing about S.

This picture generalizes to higher k: fewer than k points do not determine a unique degree-$(k-1)$ curve.

5 Secure Multiparty Computation

Shamir's secret sharing shows how to protect a single secret among multiple parties. But what if we want to go further: can we *compute* a function of many private inputs without revealing the individual inputs themselves?

Consider the following scenario: n players each hold a private input $s_i \in \mathbb{F}_p$, and they wish to jointly compute the sum

$$\text{Sum} = s_1 + s_2 + \cdots + s_n \in \mathbb{F}_p$$

(and perhaps also the average Sum/n) without revealing their individual values. A naive approach—having each player announce their input—clearly fails to preserve privacy. Secret sharing alone doesn't solve this either: it only protects a *single* secret that one party knows.

The key insight, due to Ben-Or, Goldwasser, and Wigderson (1988), is that Shamir's scheme has a powerful algebraic property: it is *linear*. This linearity allows us to perform computations directly on shared values without ever reconstructing the individual inputs. We will illustrate this idea by designing a simple protocol to compute sums securely.

Lemma 5.1 (Linearity of shares)**.** Let $P_A(x)$ and $P_B(x)$ be polynomials over $\mathbb{F}_p$ of degree at most t such that

$$P_A(0) = A, \qquad P_B(0) = B.$$

Define

$$Q(x) = P_A(x) + P_B(x).$$

Then:

1. $Q(x)$ has degree at most t,
2. $Q(0) = A + B$,
3. for every i, $Q(i) = P_A(i) + P_B(i)$.

Proof. Since each of P_A and P_B has degree at most t, their sum Q also has degree at most t.

Evaluating at $x = 0$ gives

$$Q(0) = P_A(0) + P_B(0) = A + B.$$

Finally, for each i, we have

$$Q(i) = P_A(i) + P_B(i)$$

simply by the definition of Q. ∎

Remark 5.1. Suppose P_A and P_B are the polynomials from $(t+1, n)$-threshold Shamir secret sharings of secrets A and B, respectively. Then $Q(x) = P_A(x) + P_B(x)$ is a $(t+1, n)$-threshold Shamir secret sharing of $A + B$. The share of user i in this sharing of $A + B$ is

$$Q(i) = P_A(i) + P_B(i),$$

the sum of user i's shares in the individual sharings of A and B.

We now describe a protocol that securely computes the sum $s_1 + \cdots + s_n$ with threshold k. The protocol remains private against any coalition of fewer than k players, exactly as in Shamir's scheme.

Definition 5.1 (Secure sum protocol via Shamir sharing)**.** Let n players hold private inputs $s_1, \ldots, s_n \in \mathbb{F}_p$. Fix a threshold k with $1 \leq k \leq n$. The (k, n)-threshold secure sum protocol proceeds as follows:

1. **Input sharing.** Each player i treats their input s_i as a secret and runs Shamir's (k, n) sharing scheme:

   - They choose a random degree-$(k-1)$ polynomial $P_i(x)$ with $P_i(0) = s_i$.
   - For each player j, they privately send the share $P_i(j)$ to player j.

   After this step, player j holds one share of each input:

   $$P_1(j), P_2(j), \ldots, P_n(j).$$

2. **Local computation (sum of shares).** Each player j computes

   $$Q(j) = \sum_{i=1}^{n} P_i(j).$$

   This is just a sum in $\mathbb{F}_p$, done locally. No one learns anything new at this stage; they only combine numbers they already have.

3. **Reconstruction of the sum.** Consider the polynomial

   $$Q(x) = \sum_{i=1}^{n} P_i(x).$$

   By linearity, Q has degree at most $k-1$, and

   $$Q(0) = \sum_{i=1}^{n} P_i(0) = \sum_{i=1}^{n} s_i.$$

   The values $Q(1), \ldots, Q(n)$ are exactly the shares of the Shamir sharing of the sum.

   Any set of k or more players now pool their values $Q(j)$ and use Lagrange interpolation to reconstruct $Q(x)$ and compute $Q(0)$.

Theorem 5.1 (Correctness of the secure sum protocol)**.** If all players follow the protocol, the reconstructed value $Q(0)$ is equal to the true sum $s_1 + \cdots + s_n$.

Proof. Each player i chose a polynomial P_i with $P_i(0) = s_i$. Consider the polynomial $Q(x) = \sum_{i=1}^{n} P_i(x)$ from step 3 of the protocol. Player j computes $Q(j) = \sum_{i=1}^{n} P_i(j)$ in the local computation step. Since Q is the sum of degree-$(k-1)$ polynomials, it has degree at most $k-1$. Evaluating at $x = 0$ gives

$$Q(0) = \sum_{i=1}^{n} P_i(0) = \sum_{i=1}^{n} s_i,$$

so Q encodes the desired sum.

The values $Q(1), \ldots, Q(n)$ form a (k, n)-threshold Shamir sharing of $Q(0)$. By the correctness of Shamir's scheme, any k players can use Lagrange interpolation to reconstruct $Q(x)$ from their values and recover $Q(0)$, which equals the sum $s_1 + \cdots + s_n$. ∎

Remark 5.2 (Privacy intuition). Each polynomial P_i is a Shamir sharing of s_i with threshold k. So any coalition of fewer than k players sees fewer than k shares of each P_i and, by the perfect secrecy theorem for Shamir, learns nothing about s_i.

The values $Q(j)$ are just sums of these shares. From the point of view of any coalition of fewer than k players, the distribution of the $Q(j)$'s they see is exactly what they would see if there were a single secret equal to the sum $\sum_i s_i$ being Shamir-shared to them.

Thus, when the sum is reconstructed, they learn only the final sum (which is the goal), and nothing more about individual inputs.

Theorem 5.2 (Honest-majority secure sum)**.** Suppose $k = \lceil n/2 \rceil$ and at least k players are honest (they share their true values $Q(i)$ during reconstruction and do not reveal any of the individual shares $P_j(i)$ for $j \neq i$). Then the secure sum protocol allows all players to learn $\sum_{i=1}^{n} s_i$, while any coalition of fewer than k players learns nothing about the individual inputs beyond what can be inferred from the final sum.

Proof. Since $k = \lceil n/2 \rceil$, any coalition of fewer than k players has size at most $k - 1 < n/2$. By the privacy intuition argument above, such a coalition sees fewer than k shares of each polynomial P_i, so by the perfect secrecy theorem for Shamir sharing, they learn nothing about any individual input s_i. The at least k honest players can pool their correct values $Q(j)$ and use Lagrange interpolation to reconstruct $Q(0) = \sum_{i=1}^{n} s_i$, giving all players the sum. ■

By choosing $k = \lceil n/2 \rceil$, we create a powerful property: everyone can publicly share their value $Q(i)$, allowing the entire group to reconstruct the sum $Q(0)$ using Lagrange interpolation. However, as long as at least half the players are honest, no coalition can recover any of the individual polynomials P_j (and hence the individual secrets s_j). This is because each P_j was shared with threshold k, so any coalition of fewer than k players sees fewer than k shares of P_j and learns nothing about s_j. The publicly revealed values $Q(i)$ are just sums of shares and provide no additional information beyond what the final sum reveals.

Remark 5.3 (Inherent information leakage). Proofs of security for these protocols are delicate and require careful analysis. For example, if $n = 2$ and $k = 2$, both players must participate in the reconstruction, so when the sum $S = s_1 + s_2$ is revealed, player 1 can compute $s_2 = S - s_1$. This is unavoidable: any protocol that reveals the sum to both players necessarily leaks the other player's input.

Remark 5.4 (Active adversaries). Another subtlety is that malicious players might deviate from the protocol by sharing false values of $Q(j)$ during reconstruction - actually this is also an issue for Shamir's secret sharing! This allows them to manipulate the final output while potentially learning information from honest players' responses. A complete security analysis must account for such *active adversaries* who can send arbitrary messages, not just *passive adversaries* who follow the protocol but try to learn extra information. Defending against active adversaries requires additional mechanisms like verifiable secret sharing, which is beyond the scope of this lecture. Interestingly one of the tools that is used to construct verifiable secret sharing is (Reed-Solomon) error-correcting codes!

If you are interested in learning more about this multiparty computation model, we recommend reading this paper by Gilad Asharov and Yehuda Lindell.