Data Protection

Jorge L. Villar

MCYBERS, UPC, Fall 2025

Outline

Definition of Secret Sharing

A tool to share some cryptographic task among many users.

Definition of Secret Sharing

A tool to share some cryptographic task among many users.

n users share a secret s (e.g., a secret key), so that:

- Every user P_i receives a piece of secret information s_i, called the **share**.
- The computation and distribution of shares is performed by a special party D called the **dealer**.
- Only some specific sets of users are able to recover the secret s from their shares. The set is called **authorized**.
- The dealer is no longer needed after the distribution phase.

Definition of Secret Sharing

A tool to share some cryptographic task among many users.

n users share a secret s (e.g., a secret key), so that:

- Every user P_i receives a piece of secret information s_i, called the **share**.
- The computation and distribution of shares is performed by a special party D called the **dealer**.
- Only some specific sets of users are able to recover the secret s from their shares. The set is called **authorized**.
- The dealer is no longer needed after the distribution phase.

A secret sharing is **perfect** if unauthorized sets of users can recover **no information** about s.

Shamir Secret Sharing

Let q be a prime, n the number of users and $1 \leq t \leq n$.

Every user P_i is associated to a different public value $\xi_i \in \mathbb{Z}_q^{\times}$.

Shamir Secret Sharing

Let q be a prime, n the number of users and $1 \leq t \leq n$.

Every user P_i is associated to a different public value $\xi_i \in \mathbb{Z}_q^{\times}$.

Distribution of shares:

D takes $s \in \mathbb{Z}_q$ and a random polynomial

$P(x) = s + a_1 x + \ldots + a_{t-1} x^{t-1}, \quad a_1, \ldots, a_{t-1} \in \mathbb{Z}_q$.

D computes and sends $s_i = P(\xi_i) \bmod q$ privately to P_i.

Shamir Secret Sharing

Let q be a prime, n the number of users and $1 \leq t \leq n$.

Every user P_i is associated to a different public value $\xi_i \in \mathbb{Z}_q^\times$.

Distribution of shares:

D takes $s \in \mathbb{Z}_q$ and a random polynomial

$$P(x) = s + a_1 x + \ldots + a_{t-1} x^{t-1}, \quad a_1, \ldots, a_{t-1} \in \mathbb{Z}_q.$$

D computes and sends $s_i = P(\xi_i) \bmod q$ privately to P_i.

Reconstruction of the secret:

Any set A of size $\geq t$ of users compute the unique polynomial P of degree $< t$ s.t. $P(\xi_i) = s_i \bmod q$, $\forall P_i \in A$.

(They can use Lagrange polynomial interpolation.)

Then, they recover $s = P(0) \bmod q$.

Shamir Secret Sharing

Perfectness:

Given $s_A = (s_i)_{P_i \in A}$ for any set A with $\leq t-1$ users, all values of $s \in \mathbb{Z}_q$ can occur with equal probabilities.

There exists $Z_A(x)$ of degree $< t$ such that $Z_A(\xi_i) = 0$, $\forall P_i \in A$, and $Z_A(0) = 1$.

Then, all polynomials $Q_\epsilon(x) = P(x) + \epsilon Z_A(x)$, $\epsilon \in \mathbb{Z}_q$, produce the same shares s_i for $P_i \in A$, but the recovered secret $Q_\epsilon(0) = s + \epsilon$ covers all the set $\mathbb{Z}_q$.

Shamir Secret Sharing

Perfectness:

Given $s_A = (s_i)_{P_i \in A}$ for any set A with $\leq t-1$ users, all values of $s \in \mathbb{Z}_q$ can occur with equal probabilities.

There exists $Z_A(x)$ of degree $< t$ such that $Z_A(\xi_i) = 0$, $\forall P_i \in A$, and $Z_A(0) = 1$.

Then, all polynomials $Q_\epsilon(x) = P(x) + \epsilon Z_A(x)$, $\epsilon \in \mathbb{Z}_q$, produce the same shares s_i for $P_i \in A$, but the recovered secret $Q_\epsilon(0) = s + \epsilon$ covers all the set $\mathbb{Z}_q$.

The **access structure** Γ of a secret sharing scheme is the collection of authorized sets.

Shamir scheme has a (t, n)-**threshold** access structure, where a set A is in Γ if is has $\geq t$ users.

Weighted Secret Sharing

What if not all users must have the same power?

Example of non-threshold secret sharing: every user P_i has an associated integer weight w_i.

Weighted Secret Sharing

What if not all users must have the same power?

Example of non-threshold secret sharing: every user P_i has an associated integer weight w_i.

- A set A is authorized if $\sum_{P_i \in A} w_i \geq t$.

Weighted Secret Sharing

What if not all users must have the same power?

Example of non-threshold secret sharing: every user P_i has an associated integer weight w_i.

- A set A is authorized if $\sum_{P_i \in A} w_i \geq t$.
- The access structure can be realized with Shamir secret sharing by giving w_i shares to user P_i.
- The real user P_i corresponds to a set of w_i different users in the underlying Shamir scheme.

Weighted Secret Sharing

What if not all users must have the same power?

Example of non-threshold secret sharing: every user P_i has an associated integer weight w_i.

- A set A is authorized if $\sum_{P_i \in A} w_i \geq t$.
- The access structure can be realized with Shamir secret sharing by giving w_i shares to user P_i.
- The real user P_i corresponds to a set of w_i different users in the underlying Shamir scheme.

It can be very inefficient due to the amount of secret information given to a single user.

Linear Secret Sharing

A variant (generalization) of Shamir secret sharing.

Linear Secret Sharing

A variant (generalization) of Shamir secret sharing.

Secret distribution:

The values ξ_i are replaced by vectors $\boldsymbol{v}_i \in \mathbb{Z}_q^d$, for some dimension d, such that $\boldsymbol{v}_i \neq \boldsymbol{e}_1 = (1, 0, \ldots, 0)$.

D takes a random vector $\boldsymbol{w} \in \mathbb{Z}_q^d$ such that $w_1 = \boldsymbol{w} \cdot \boldsymbol{e}_1 = s$, and computes the shares $s_i = \boldsymbol{w} \cdot \boldsymbol{v}_i$.

Linear Secret Sharing

A variant (generalization) of Shamir secret sharing.

Secret distribution:

The values ξ_i are replaced by vectors $\boldsymbol{v}_i \in \mathbb{Z}_q^d$, for some dimension d, such that $\boldsymbol{v}_i \neq \boldsymbol{e}_1 = (1, 0, \ldots, 0)$.

D takes a random vector $\boldsymbol{w} \in \mathbb{Z}_q^d$ such that $w_1 = \boldsymbol{w} \cdot \boldsymbol{e}_1 = s$, and computes the shares $s_i = \boldsymbol{w} \cdot \boldsymbol{v}_i$.

Secret reconstruction:

A set A is authorized if $\boldsymbol{e}_1$ is in the span of the vectors of the users in A. In this case,

$\boldsymbol{e}_1 = \sum_{P_i \in A} \lambda_i \boldsymbol{v}_i$, for some coefficients λ_i.

Then, $s = \boldsymbol{w} \cdot \boldsymbol{e}_1 = \sum_{P_i \in A} \lambda_i \boldsymbol{w} \cdot \boldsymbol{v}_i = \sum_{P_i \in A} \lambda_i s_i$.

Linear Secret Sharing

Perfectness:

The scheme is again perfect, since if $\boldsymbol{e}_1$ is not in the span then there exists a vector $\boldsymbol{z}_A$ orthogonal to all $\boldsymbol{v}_i$, $P_i \in A$, such that $\boldsymbol{z}_A \cdot \boldsymbol{e}_1 = 1$.

$\boldsymbol{z}_A$ plays the same role as $Z_A(x)$ in Shamir scheme.

Linear Secret Sharing

Perfectness:

The scheme is again perfect, since if $\boldsymbol{e}_1$ is not in the span then there exists a vector $\boldsymbol{z}_A$ orthogonal to all $\boldsymbol{v}_i$, $P_i \in A$, such that $\boldsymbol{z}_A \cdot \boldsymbol{e}_1 = 1$.

$\boldsymbol{z}_A$ plays the same role as $Z_A(x)$ in Shamir scheme.

Generalization:

In a more general setting, every user can be associated to more than one public vector, and receive the corresponding shares.

The access structure depends on the geometric configuration of the public vectors.

Outline

Dealer's Verifiability

In real applications the actions by a possibly dishonest dealer must be verifiable.

Dealer's Verifiability

In real applications the actions by a possibly dishonest dealer must be verifiable.

- Enough honest users must be able to detect a dishonest dealer.
- If the dealer passes the verification, then all authorized sets of honest users will recover the same secret value.
- If the dealer does not pass the verification, then all honest users exit the protocol.

Dealer's Verifiability

In real applications the actions by a possibly dishonest dealer must be verifiable.

- Enough honest users must be able to detect a dishonest dealer.
- If the dealer passes the verification, then all authorized sets of honest users will recover the same secret value.
- If the dealer does not pass the verification, then all honest users exit the protocol.

In Shamir scheme the dealer can corrupt any particular share without being detected, unless more than t users pool their shares.

Feldman Secret Sharing

It adds a layer of verifiability on top of a Shamir scheme:

Let (G, q, g) define a group with a hard discrete logarithm problem, and consider the (t, n) Shamir secret sharing.

Feldman Secret Sharing

It adds a layer of verifiability on top of a Shamir scheme:

Let (G, q, g) define a group with a hard discrete logarithm problem, and consider the (t, n) Shamir secret sharing.

Distribution of shares:

D takes $s \in \mathbb{Z}_q$ and a random $P(x) = s + a_1 x + \ldots + a_{t-1} x^{t-1}$, $a_1, \ldots, a_{t-1} \in \mathbb{Z}_q$, and publishes $A_0 = g^s$ and $A_j = g^{a_j}$ for all j.

D computes and sends $s_i = P(\xi_i) \bmod q$ privately to P_i.

Feldman Secret Sharing

It adds a layer of verifiability on top of a Shamir scheme:

Let (G, q, g) define a group with a hard discrete logarithm problem, and consider the (t, n) Shamir secret sharing.

Distribution of shares:

D takes $s \in \mathbb{Z}_q$ and a random $P(x) = s + a_1 x + \ldots + a_{t-1} x^{t-1}$, $a_1, \ldots, a_{t-1} \in \mathbb{Z}_q$, and publishes $A_0 = g^s$ and $A_j = g^{a_j}$ for all j.

D computes and sends $s_i = P(\xi_i) \bmod q$ privately to P_i.

Dealer's Verification:

P_i checks whether $g^{s_i} = \prod_{j=0}^{t-1} A_j^{\xi_i^j}$, and complains if not.
If there are $\geq t$ complaints, then the protocol is aborted.
Otherwise, complaining users are removed from the protocol.

Feldman Secret Sharing

We assume that the adversary *corrupts* at most $t-1$ users.
Otherwise, it can obtain the secret from t users' shares.

Analysis of Dealer's Verification:

- Any honest P_i accepting the share has received $s_i = P(\xi_i)$, where $P(\cdot)$ is implicitly defined by the $A_j = g^{a_j}$:

$$g^{s_i} = \prod_{j=0}^{t-1} A_j^{\xi_i^j} = g^s \prod_{j=1}^{t-1} g^{a_j \xi_i^j} = g^{s + a_1 \xi_i + \ldots + a_{t-1} \xi_i^{t-1}} = g^{P(\xi_i)}.$$

- If D is honest, only dishonest P_i can complain, and then $t-1$ complaints are not enough to force protocol abortion.
- An accepted dishonest D can cheat to $< t$ honest users.

Feldman Secret Sharing

- In the reconstruction phase every user can verify the others' shares from the A_j's.
- In the worst case, every set of at least $3t - 2$ users has at least t correct shares, and then they can recover s.

  There could be:
  - up to $t - 1$ corrupted users not providing the correct share
  - up to $t - 1$ honest users removed from the protocol
  - at least t remaining users providing valid shares

Feldman Secret Sharing

- In the reconstruction phase every user can verify the others' shares from the A_j's.
- In the worst case, every set of at least $3t-2$ users has at least t correct shares, and then they can recover s.

  There could be:
  - up to $t-1$ corrupted users not providing the correct share
  - up to $t-1$ honest users removed from the protocol
  - at least t remaining users providing valid shares

It requires $n > 3t-2$, i.e. D can corrupt less than $n/3$ users.

The actual reconstruction threshold is not t but $3t-2$.

The secret value is not actually hidden ($A_0 = g^s$ is public and then no IND-like security notion can be achieved).

Pedersen Secret Sharing

Two independent instances of Shamir's scheme are combined to produce the public information:

Now we take (G, q, g, h) where h is a random element in G.

Pedersen Secret Sharing

Two independent instances of Shamir's scheme are combined to produce the public information:

Now we take (G, q, g, h) where h is a random element in G.

Distribution of shares:

D takes $s, s' \in \mathbb{Z}_q$ and random $P(x) = s + a_1 x + \ldots + a_{t-1} x^{t-1}$ and $Q(x) = s' + a'_1 x + \ldots + a'_{t-1} x^{t-1}$, and publishes $A_0 = g^s h^{s'}$ and $A_j = g^{a_j} h^{a'_j}$ for all j.

D computes and sends $(s_i = P(\xi_i), s'_i = Q(\xi_i))$ privately to P_i.

Pedersen Secret Sharing

Two independent instances of Shamir's scheme are combined to produce the public information:

Now we take (G, q, g, h) where h is a random element in G.

Distribution of shares:

D takes $s, s' \in \mathbb{Z}_q$ and random $P(x) = s + a_1 x + \ldots + a_{t-1} x^{t-1}$ and $Q(x) = s' + a'_1 x + \ldots + a'_{t-1} x^{t-1}$, and publishes $A_0 = g^s h^{s'}$ and $A_j = g^{a_j} h^{a'_j}$ for all j.

D computes and sends $(s_i = P(\xi_i), s'_i = Q(\xi_i))$ privately to P_i.

Dealer's Verification:

P_i checks whether $g^{s_i} h^{s'_i} = \prod_{j=0}^{t-1} A_j^{\xi_i^j}$, and complains if not. If there are at least t complaints, then the protocol is aborted.

Pedersen Secret Sharing

Analysis of Dealer's Verification:

- No deterministic function of s is published.
- The secret sharing scheme is perfect (i.e. $< t-1$ users have no information about s even if they can solve the Discrete Log problem).

  The polynomials $P(x) - \lambda\epsilon Z_A(x)$ and $Q(x) + \epsilon Z_A(x)$, where $h = g^{\lambda}$, produce the same A_0, A_j's and the same $t-1$ shares, but the arbitrary secret value $s - \lambda\epsilon$.

Pedersen Secret Sharing

Analysis of Dealer's Verification:

- A dishonest D capable to produce corrupted values $(\tilde{s}_i, \tilde{s}'_i)$ fulfilling the verification equation can be used to solve the Discrete Logarithm problem in G.

  In fact, $DLOG_g(h) = -(\tilde{s}'_i - s'_i)(\tilde{s}_i - s_i)^{-1} \bmod q$.

Pedersen Secret Sharing

Analysis of Dealer's Verification:

- A dishonest D capable to produce corrupted values $(\tilde{s}_i, \tilde{s}'_i)$ fulfilling the verification equation can be used to solve the Discrete Logarithm problem in G.

  In fact, $DLOG_g(h) = -(\tilde{s}'_i - s'_i)(\tilde{s}_i - s_i)^{-1} \bmod q$.

The scheme requires a trusted generation of g and h in order to guarantee that D does not know λ such that $h = g^\lambda$.

For instance, they can use a hash function $H(\cdot)$, and define $h = H(g, u)$, where u is the first nonzero integer that makes h a generator of G.

Outline

Distributed ElGamal PKE

In a distributed version of ElGamal PKE:

- The secret key $x \in \mathbb{Z}_q$ in ElGamal PKE scheme, can be secret shared among n users $P_1, \ldots, P_n$.
- The public key $y = g^x$ is the same as in the original scheme (and the sender cannot see any difference between the two versions).
- There is a way to decrypt a ciphertext without reconstructing the secret key.

  (Hence, the secret key remains distributed among the users in the long term.)

Distributed ElGamal PKE

Simple case: Using a $t = n$ threshold access structure (all users are required to collaborate in the decryption process).

It is implemented with a random splitting of the secret key:

$$x = x_1 + x_2 + \ldots + x_n.$$

(No Lagrange interpolation is needed in the reconstruction).

The public key is the product of the "partial public keys"

$$y = g^x = g^{x_1 + \cdots + x_n} = g^{x_1} \cdots g^{x_n} = y_1 \cdots y_n.$$

Then, given a ciphertext $(c_1, c_2) = (g^r, y^r m)$, we have

$$y^r = c_1^x = c_1^{x_1 + \ldots + x_n} = c_1^{x_1} \cdots c_1^{x_n}.$$

Distributed ElGamal PKE

Decryption procedure:

- Every P_i uses his secret key share x_i to compute the partial decryption $c_1^{x_i}$.
- All partial decryptions are sent to a combiner that multiplies them to recover y^r.
- Finally, $m = c_2(y^r)^{-1}$.

Distributed ElGamal PKE

Decryption procedure:

- Every P_i uses his secret key share x_i to compute the partial decryption $c_1^{x_i}$.
- All partial decryptions are sent to a combiner that multiplies them to recover y^r.
- Finally, $m = c_2(y^r)^{-1}$.

A similar procedure can be applied with a more general threshold structure (i.e. $t < n$).

ZK-proofs (equality of discrete logarithms) can be used to show that every partial decryption is computed honestly.

Distributed RSA-FDH Signature

Using again the additive random splitting of the secret key among k users: $d = d_1 + \cdots + d_k$

The signature of m is computed as

$$s = H(m)^d \bmod n = (H(m)^{d_1} \bmod n) \cdots (H(m)^{d_k} \bmod n),$$

where each "partial signature" $H(m)^{d_i} \bmod n$ is computed by the corresponding P_i.

Distributed RSA-FDH Signature

Using again the additive random splitting of the secret key among k users: $d = d_1 + \cdots + d_k$

The signature of m is computed as

$$s = H(m)^d \bmod n = (H(m)^{d_1} \bmod n) \cdots (H(m)^{d_k} \bmod n),$$

where each "partial signature" $H(m)^{d_i} \bmod n$ is computed by the corresponding P_i.

- The partial secret keys are not revealed in the signature generation
- It can be generalized to smaller thresholds (but it is not so easy!).
- The factorization of the modulus n is not known to any party, and therefore, a trusted key generation is required.

Multiparty Computation

Secret sharing, distributed decryption or joint signature generation are examples of **multiparty computation protocols**.

Multiparty Computation

Secret sharing, distributed decryption or joint signature generation are examples of **multiparty computation protocols**.

In a multiparty computation protocol run by $P_1, \ldots, P_n$:

- Every P_i has some private input x_i.
- There is some public input y known to everybody.
- All the parties interact according to the protocol description.
- In the end every P_i only learns a function $z_i = f_i(x_1, \ldots, x_n, y)$.

Multiparty Computation

In some simple cases all f_i are the same, or all but one are trivial (e.g. $z_i = x_i$).

Examples:

- **Reconstruction of a shared secret.** Every input x_i is P_i's share, and all outputs z_i are equal to the shared secret.
- **Distributed decryption.** x_i are the secret key shares and all z_i are equal to the plaintext.
- **Joint signature generation.** x_i are the secret key shares and all z_i are equal to the signature.

Multiparty Computation

In some simple cases all f_i are the same, or all but one are trivial (e.g. $z_i = x_i$).

Examples:

- **Reconstruction of a shared secret.** Every input x_i is P_i's share, and all outputs z_i are equal to the shared secret.
- **Distributed decryption.** x_i are the secret key shares and all z_i are equal to the plaintext.
- **Joint signature generation.** x_i are the secret key shares and all z_i are equal to the signature.

The homomorphic properties of secret sharing can be used to implement general constructions of multiparty protocols for any arithmetic circuit (i.e. polynomial).

Addition and Multiplication of Secrets

Shamir secret sharing has additive homomorphic properties:

$[s_1, \ldots, s_n] \to s$ denotes that s_i are shares of the secret s.

$$\left.\begin{array}{l}[s_1, \ldots, s_n] \to s \\ [s'_1, \ldots, s'_n] \to s'\end{array}\right\} \Rightarrow \quad [s_1 + s'_1, \ldots, s_n + s'_n] \to s + s',$$

assuming the same threshold is used for both s and s'.

Just add the two sharing polynomials $P(x)$ and $P'(x)$.

Addition and Multiplication of Secrets

Shamir secret sharing has additive homomorphic properties:

$[s_1, \ldots, s_n] \to s$ denotes that s_i are shares of the secret s.

$$\left.\begin{array}{l}[s_1, \ldots, s_n] \to s \\ [s'_1, \ldots, s'_n] \to s'\end{array}\right\} \Rightarrow \quad [s_1 + s'_1, \ldots, s_n + s'_n] \to s + s',$$

assuming the same threshold is used for both s and s'.

Just add the two sharing polynomials $P(x)$ and $P'(x)$.

Multiplication is not so trivial, but there exist some ways to compute the shares of ss' by using interactive protocols.

Just multiplying $P(x)$ and $P'(x)$ is not enough, because the threshold is increased and also the resulting polynomial is not uniformly distributed.

General Multiparty Computation

- Addition and multiplication of shared secrets, allow computing any arbitrary polynomial function on shared secrets.
- There also exist more efficient non-generic multiparty protocol constructions for specific functions.
- An example is the joint generation of Diffie-Hellman keys and RSA keys, that can safely remove the need of the trusted key generation in previous protocols.

General Multiparty Computation

- Addition and multiplication of shared secrets, allow computing any arbitrary polynomial function on shared secrets.
- There also exist more efficient non-generic multiparty protocol constructions for specific functions.
- An example is the joint generation of Diffie-Hellman keys and RSA keys, that can safely remove the need of the trusted key generation in previous protocols.

Multiparty computation protocols can be used to protect private or critical data, even when data comes from different users.

(E.g. statistical analysis of different critical data sources, from the health system or from users' preferences, etc.)

Data Protection

Jorge L. Villar

MCYBERS, UPC, Fall 2025

END