Outline

1. Secret Sharing
   - Introduction
   - Shamir Secret Sharing
   - Verifiable Secret Sharing

2. Distributed Cryptography
   - Introduction
   - Distributed Decryption
   - Distributed Signature
   - Distributed Key Generation

Key Management

In case of a critical private key (decryption or signing key)

- **Abuse**: one user can use the secret key alone
- **Loss**: in case of loss of the key (destruction)

⇒ share the secret key between several users
Secret Sharing Schemes

Let $S \in \{0, 1\}^\ell$ be a secret bit-string to be shared between two people (Alice and Bob):
- one chooses a random $S_1 \in \{0, 1\}^\ell$, and sends it to Alice
- one computes $S_2 = S \oplus S_1$, and sends it to Bob

Security:
- Alice knows a random value
- Bob knows a value masked by a random value: a random value!

$\implies$ individually, they have no information on $S$

Together, they can recover $S = S_1 \oplus S_2$

Unconditional Security

Any subgroup of $(n - 1)$ people has no information on $S$!
$\implies$ if one people does not want / is not able to cooperate:

$S$ is lost forever!

Threshold Secret Sharing

$(n, k)$-Threshold Secret Sharing

A secret $S$ is shared among $n$ users:
- any subgroup of $k$ people (or more) can recover $S$
- any subgroup of less than $k$ people has no information about $S$
Lagrange Interpolation of Polynomials

Let us be given \( k \) points \((x_1, y_1), \ldots, (x_k, y_k)\), with distinct abscissa. There exists a unique polynomial \( P \) of degree \( k - 1 \) such that \( P(x_i) = y_i \) for \( i = 1, \ldots, k \).

\[
L_j(X) = \prod_{i \neq j} \frac{X - x_i}{x_j - x_i}
\]

As a consequence:

\[
P(X) = \sum_{j=1}^{k} y_j L_j(X)
\]

satisfies

\[
\begin{align*}
\deg(P) &= k - 1 \\
P(x_i) &= y_i \quad \forall i = 1, \ldots, k
\end{align*}
\]

Verifiable Secret Sharing

If Eve claims she shared her decryption key: how can we trust her?

- we try to recover the key?
- how to do without revealing additional information?

\[\implies\text{Verifiable Secret Sharing}\]

For DL Keys

Eve’s keys are, in a group \( \mathbb{G} = \langle g \rangle \) of prime order \( q \),

\[
sk = x \quad pk = y = g^x
\]

\((n, k)\)-Secret sharing: \( x = P(0) \) for \( P(X) = \sum_{i=0}^{k-1} a_i X^i \)

\[\implies S_i = P(i) \text{ for } i = 1, \ldots, n\]

For any subset \( S \) of \( k \) indices:

- \( x = \sum_{j \in S} S_j \lambda_{S,j} \)
- \( y = g^x = g^{\sum_{j \in S} S_j \lambda_{S,j}} = \prod_{j \in S} (g^{S_j})^{\lambda_{S,j}} = \prod_{j \in S} v_j^{\lambda_{S,j}} \) for \( v_j = g^{S_j} \)
Verifiable Secret Sharing for DL Keys

For DL Keys [Feldman – FOCS ’87]

Eve’s keys are, in a group \( G = \langle g \rangle \) of prime order \( q \),

\[
sk = x \quad pk = y = g^x
\]

\((n, k)\)-Secret sharing: \( x = P(0) \) for \( P(X) = \sum_{i=0}^{k-1} a_i X^i \)

- Eve computes \( S_i = P(i) \) for \( i = 1 \ldots n \) and \( v_i = g^{S_i} \)
- Eve sends each \( S_i \) privately to each \( U_i \)
- Eve publishes \( v_i = g^{S_i} \) for \( i = 1, \ldots, n \)
- Each \( U_i \) can then check its own \( v_i \) w.r.t. to its \( S_i \)
- Anybody can check

\[
y = \prod_{j \in S} v_j^{\lambda_{S,j}}
\]

for any subset \( S \) of size \( k \)

Outline

1 Secret Sharing

2 Distributed Cryptography

- Introduction
  - Distributed Decryption
  - Distributed Signature
  - Distributed Key Generation

Secret Sharing vs. Distributed Cryptography

If Eve shares her decryption key \( sk \),
the \( (U_i) \) will have to cooperate to recover the key \( sk \)
and then decrypt the ciphertext

But then, they all know the decryption key \( sk \)!

How can the \( (U_i) \) use their shares \( (S_i) \) to decrypt (or sign),
without leaking any additional information about \( sk \)?

\( \Rightarrow \) Multi-party computation

Let us try on ElGamal decryption (with shared DL keys)
ElGamal Encryption

In a group $\mathbb{G} = \langle g \rangle$ of order $q$

- $K(\mathbb{G}, g, q): x \overset{R}{\leftarrow} \mathbb{Z}_q$, and $sk \leftarrow x$ and $pk \leftarrow y = g^x$
- $E_{pk}(m): r \overset{R}{\leftarrow} \mathbb{Z}_q$, $c_1 \leftarrow g^r$ and $c_2 \leftarrow y^r \times m$.
  Then, the ciphertext is $c = (c_1, c_2)$
- $D_{sk}(c)$ outputs $c_2/c_1^x$

We assume an $(n, k)$-secret sharing of $x$ and a qualified set $S$: $x = \sum_{j \in S} S_j \lambda_{S,j}$

$D_{sk}(c) = c_2/c_1^x$: one needs to compute $c_1^x = \prod_{j \in S} (c_1^s)^{\lambda_{S,j}}$

Each user computes $C_j = c_1^{S_j}$, and then $c_1^x = \prod_{j \in S} C_j^{\lambda_{S,j}}$

Fraud Detection

Each user computes $C_j = c_1^{S_j}$, and then $c_1^x = \prod_{j \in S} C_j^{\lambda_{S,j}}$

But $U_1$, sends a random $C_1$: instead of $c_1^{S_1}$, knowing also $v_1 = g^{S_1}$

$\Longrightarrow$ Decide a DDH tuple $(g, c_1, v_1, C_1)$

Robustness

A defrauder can be detected

$\Longrightarrow$ Proof of DDH membership for the tuple $(g, c_1, v_1, C_1)$, without leakage of any information about $S_1$

NIZK Diffie-Hellman Language

In a group $\mathbb{G} = \langle g \rangle$ of order $q$,

- $K(\mathbb{G}, g, q): x \overset{R}{\leftarrow} \mathbb{Z}_q$, and $sk \leftarrow x$ and $pk \leftarrow y = g^x$
- $E_{pk}(m): r \overset{R}{\leftarrow} \mathbb{Z}_q$, $c_1 \leftarrow g^r$ and $c_2 \leftarrow y^r \times m$.
  Then, the ciphertext is $c = (c_1, c_2)$
- $D_{sk}(c)$ outputs $c_2/c_1^x$

Given each qualified set $S$: $x = \sum_{j \in S} S_j \lambda_{S,j}$

Each user computes $C_j = c_1^{S_j}$, and then $c_1^x = \prod_{j \in S} C_j^{\lambda_{S,j}}$

Assume Charlie a.k.a. $U_1$, sends a random $C_1$:

- the others will compute a wrong decryption
- Charlie will be able to extract the plaintext!
1 Secret Sharing

2 Distributed Cryptography
   - Introduction
   - Distributed Decryption
   - Distributed Signature
   - Distributed Key Generation

Schnorr Signature

- \( G = \langle g \rangle \) of order \( q \) and \( H: \{0, 1\}^* \rightarrow \mathbb{Z}_q \)
- Key Generation \( \rightarrow (y, x): x \in \mathbb{Z}_q^* \) and \( y = g^{-x} \)
- Signature of \( m \rightarrow (r, h, s) \)
  \[ k \overset{R}{\leftarrow} \mathbb{Z}_q^* \quad r = g^k \quad h = H(m, r) \quad s = k + xh \mod q \]
- Verification of \( (m, r, s) \)
  compute \( h = H(m, r) \) and check \( r = g^s y^h \)

We assume an \((n, k)\)-secret sharing of \( x \) (with the \( v_i = g^{S_i} \)) and a qualified set \( S: x = \sum_{j \in S} S_j \lambda_{S,j} \)

The users generate a common \( r \) and then sign \((m, r)\) with a partial signature \( s_i \) under \( v_i \):

\[ \implies \text{the linearity leads to a global signature} \]

Distributed Schnorr Signature

- \( G = \langle g \rangle \) of order \( q \) and \( H: \{0, 1\}^* \rightarrow \mathbb{Z}_q \)
- Key Generation \( \rightarrow (y, x): x \in \mathbb{Z}_q^* \) and \( y = g^{-x} \)

We assume an \((n, k)\)-secret sharing of \( x \) (with the \( v_i = g^{S_i} \)) and a qualified set \( S: x = \sum_{j \in S} S_j \lambda_{S,j} \)

The users generate a common \( r \) and then sign \((m, r)\) with a partial signature \( s_i \) under \( v_i \):

\[ \implies \text{the linearity leads to a global signature} \]

Each partial signature \((m, r_i, s_i)\) can be checked: \( r_i = g^{s_i} v_i^h \)
Distributed Key Generation

In the previous schemes (ElGamal encryption and Schnorr signature) the keys are generated in a centralized way: someone knows the secret key!

Distributed cryptography should include a distributed key generation: the secret key should never exist in one place.

\((n, n)\)-Threshold DL Key Generation

- \(G = \langle g \rangle\) of order \(q\)
- Key Generation \(\rightarrow (y, x)\):
  - each \(U_i\) chooses \(x_i \in \mathbb{Z}_q^*\) and publishes \(y_i = g^{x_i}\)
  - anybody can compute \(y = \prod y_i = g^{\sum x_i}\)

The public key \(y\) corresponds to the “virtual” secret key \(x = \sum x_i \mod q\)

\((n, k)\)-Threshold DL Key Generation

- \(G = \langle g \rangle\) of order \(q\)
- Key Generation \(\rightarrow (y, x)\):
  - each \(U_i\) chooses a polynomial \(P_i\) of degree \(k - 1\), and sends \(S_{ij} = P_i(j)\) to \(U_j\)
  - each \(U_j\) can then compute \(S_j = \sum_i S_{ij} = \sum_i P_i(j) = P(j)\), where \(P = \sum_i P_i\)
  - each \(U_j\) computes and publishes \(v_j = g^{S_j}\)

The \((S_j)\) are an \((n, k)\)-secret sharing of the “virtual” secret key \(x\), corresponding to the public key \(y\), that anybody can compute:

For any qualified set \(S\):
- Secretly: \(x = \sum_{j \in S} S_j \lambda_{S,j} \mod q\)
- Publicly: \(y = \prod_{j \in S} v_j^{\lambda_{S,j}}\)