Outline

Schnorr Signatures

Commitments
Schnorr Signatures
Schnorr Signatures

• Signatures from groups
  • Gen outputs \((v_k = g^x, \text{sign key} = x)\)

• Sign \((m, \text{sign key})\) :

• Verify \((\sigma, v_k, m)\) :
Schnorr Signatures

\[ \text{sign key} = x \]

\[ \text{I know } x! \]

\[ \text{vk} = g^x \]

\[ x = g^x, \quad R = g^r, \quad h \]

\[ R = g^r \]

\[ h \]

\[ S = r + hx \]

\[ g^s = (g^r)(g^x)^h = R \cdot x^h \]

"Alice must know \( x \)"
Schnorr Signatures

\[ \text{sign key} = x \]

\[ \text{vk} = g^x \]

I know x!

\[ \text{Sign}(m, sk) = ?? \]

\[ R = g^r, \quad h = H(m || R), \quad S = (r + h \cdot x) \]
Schnorr Signatures

• Signatures from groups
  • Gen outputs \((vk = g^x, \text{ sign key } = x)\)
  • Sign \((m, \text{ sign key}) = R = g^r, h = H(m, R), s = r + hx\). Output \((h, s)\)
  • Verify \((\sigma, vk, m) : \text{ Check if } h = H(m, g^sX^{-h})\)

• Is this secure?
Schnorr Signatures

• Signatures from groups
  • Gen outputs \( (vk = g^x, \text{sign key} = x) \)
  
  • Sign \( (m, \text{sign key}) = R = g^r, \ h = H(m,R), \ s = r + hx. \ Output \ (R,s) \)
  
  • Verify \( (\sigma, vk, m) : \) Check if \( g^s = RX^h \) for \( h = H(m,R) \)

• Is this secure?

  A forger can be used to get distinct signatures \((h_1,s_1), (h_2,s_2)\) with same \((m,R)\) (different \( h \), by programming the RO), and that lets us solve for \( x \)
Schnorr Signatures

\[ vk = g^x \]

\[ (m, R) \]

\[ h \]

\[ m \]

\[ (h, s, R) \]

\[ D_h = H(m, R) \]

s.t. verification passes.

Game 1:

\[ A \]

\[ \frac{m, R}{h_1} \]

Game 2:

\[ A \]

\[ \frac{mR}{s_2} \]

\[ s_2 = r + h_2 \]

(s, s_2, h_1, h_2 are known)
Schnorr Signatures

\[ \text{vk} = g^x \]

m
Multi-Signatures

• Multiple signers signing the same message

• Each signer has an (SK,VK) pair

• Resulting signature must be “compact”: size independent of #signers
Multi-Signatures

- Multiple signers signing the same message
  \( x_i \)
  \( y_i \)

- Each signer has an (SK,VK) pair

- Resulting signature must be “compact”: size independent of #signers

- Security requirement: Unforgeability (chosen message security)

- Adversary can collude with all but one signer
Multi-Signatures

• Schnorr: $sk = x$, $vk = X = g^x$
  - Sign $(m, x): R = g^r$, $h = H(m, R)$, $s = r + hx$. Output $(R, s)$
  - Verify $(\sigma, X, m)$: Check if $g^s = RX^h$ for $h = H(m, R)$
Multi-Signatures

• **Schnorr**: $sk = x$, $vk = X = g^x$
  
  • Sign $(m, x) : R = g^r$, $h = H(m, R)$, $s = r + hx$. Output $(R, s)$
  
  • Verify $(\sigma, X, m) :$ Check if $g^s = RX^h$ for $h = H(m, R)$

• Multi-signatures:
  
  • Multiple signers with signing keys $x_1,..,x_n$ and verification keys $X_1,..,X_n$
  
  can create “aggregated” signature $(R, s)$ such that $g^s = R.X_1^{h_1}...X_n^{h_n}$

$$(s, R)$$

$$g^s = R.X_1^{h_1}X_2^{h_2}...X_n^{h_n}$$
Multi-Signatures

- **Schnorr:** \( sk = x, \; vk = X = g^x \)
  
  - Sign \((m, \; x) : R = g^r, \; h = H(m, R), \; s = r + hx. \) Output \((R, s)\)
  
  - Verify \((\sigma, \; X, \; m) : \) Check if \( g^s = RX^h \) for \( h = H(m, R) \)

- **Multi-signatures:**
  
  - Multiple signers with signing keys \( x_1, \ldots, x_n \) and verification keys \( X_1, \ldots, X_n \)
    can create “aggregated” signature \((R, s)\) such that \( g^s = R.X_1^{-1}X_n^{-n} \)
  
  - Each party picks \( r_i \) and publishes \( g^{r_i} \). Set \( R = g^{r_1 + \ldots + r_n} \).
  
  - Set \( h_i = H(m, R, X_i, L) \), where \( L = (X_1, \ldots, X_n) \)

\[
\text{Verify:} \quad s = r_i + (x_i \cdot h_i) + (\text{other terms})
\]

\[
\text{Sign:} \quad h_i = H(m, R, X_i, X_1, \ldots, X_n)
\]
Multi-Signatures

• Schnorr: \( sk = x, \; vk = X = g^x \)
  
  • Sign \((m, x)\): \( R = g^r, \; h = H(m, R), \; s = r + hx. \) Output \((R, s)\)
  
  • Verify \((\sigma, X, m)\): Check if \( g^s = RX^h \) for \( h = H(m, R) \)

• Multi-signatures:
  
  • Multiple signers with signing keys \( x_1, ..., x_n \) and verification keys \( X_1, ..., X_n \)
    can create “aggregated” signature \((R, s)\) such that \( g^s = R.X_1^{h_1}...X_n^{h_n} \)
  
  • Each party picks \( r_i \) and publishes \( g^{r_i} \). Set \( R = g^{r_1 + ... + r_n} \).
  
  • Set \( h_i = H(m, R, X_i, L) \), where \( L = (X_1, ..., X_n) \)
  
  • Then, sequentially \( s_i = s_{i-1} + r_i + h_i x_i \) (starting with \( s_0 = 0 \)).
  
  • So that final signature \( s_n = r + h_1 x_1 + ... + h_n x_n \) where \( R = g^r \).
Commitments
Commitments
Commitments
Commitments

"Commit":

"Decommit":

"Secret":

Com. string
Commitments

- Hiding

\[ \forall m_0, m_1, \quad \text{com}(m_0; r) \approx \text{com}(m_1; r) \]

- Binding

\[ \forall \text{ string } c \in \{0, 1\}^*, \quad \Pr \left[ \exists (k_1, m_1) \left( \text{Decommit}(c, k_1, m_1) = 1 \right) \land \left( \text{Decommit}(c, k_2, m_2) = 1 \right) \right] = \text{negl} \]

\[ \Pr(\text{Adv can find } (k_1, m_1, k_2, m_2) \text{ s.t. } (\text{Decom}(c, k_1, m_1) = 1) \land (\text{Decom}(c, k_2, m_2) = 1)) = \text{negl}. \]
Examples

1. Is \((g, g^x)\) a commitment to \(x\)?

   *Not*

2. \(Ct = E(k, m)\) for a symmetric key encryption \(E\)

   *Not commitment*

   because not binding

   (Recall: one-time pad \(E(k, m) = k \oplus m\)).

Decommit:

\[ C, m, y = E(k, m) \]

\[ m', k' \]

\[ m' = m \oplus 1, \quad k' = k \oplus 1. \]
Examples

In practice, we use:

- To commit to message M, choose random, fixed-length r, send $H(r || M)$
- To open commitment, send r, M
- Receiver cannot fully recover M.

- Sender cannot find another $M'$ to open.

To commit to M, pick large r. $\text{com}(M, r)^2$ SHA$(M || r)$ (computational binding).
Pedersen Commitments

• Public parameters: (p,g,h)
  • p: large prime (1024 bit)
  • g: generator
  • h: $g^a$ for hidden a

• Protocol
  • To commit to x, C chooses random r and sends $(g^x h^r)$ to R.
  • To open, C sends x and r to R.

• Benefits:
  • One can prove many things about the committed value without opening it
Pedersen Commitments

• Unconditionally hiding
  • Given a commitment c, every value x is equally likely to be the value committed in c.
  • For example, given x, r, and any x’, there exists r’ such that \( g^x h^r = g^{x’} h^{r’} \), in fact \( r = (x - x’) a^{-1} + r \mod q \).
Pedersen Commitments

• Computationally binding
  • Suppose committer sent \( g^x h^r \mod p \) for some \((x, r)\)
  • Now it finds \( x' \neq x \) and \( r' \) such that \( c = g^{x'} h^{r'} \).

  • This means that the sender ‘‘knows’’ \( \log_g(h) = (x' - x) \cdot (r - r')^{-1} \).

  • This means: assuming DL is hard, the sender cannot open the commitment with another value.
Prove Knowledge of Discrete Log without revealing it?

\[ vk = g^x \]

sign key = x

PROVE: I know x!
Prove Knowledge of Discrete Log without revealing it?

PROVE: I know x!

$vk = g^x$

sign key = x