Outline

OR Composition, continued

Pairing-based cryptography
OR Composition, Continued
Encrypting bits

**Prover**

NP language = \{ (g,h,v,w) : \exists (b,c) such that b \in \{0,1\} and v = g^c, w = h^c . g^b \}

\[
\text{Enc : } \quad \text{pp} = (g, h) \quad h = g^x \\
ct = \text{Enc}(b, g, h, c) \rightarrow (h^c . g^b, g^c)
\]

p.t. \( ct \) = Enc(0, ...) or Enc(1, ...)

\[
g, h, (h^c . g^b = h^c) = (g, g^x, g^c, g^{xc'})
\]
OR Composition

**Prover**

Goal: Prove that \( \exists w \) such that \((x_0, w) \in R_0 \) or \((x_1, w) \in R_1\)

Suppose we have a protocol \((P_0, V_0)\) for \(R_0\), and a protocol \((P_1, V_1)\) for \(R_1\)

Can we combine them to obtain a protocol for \(R_0 \ OR \ R_1\)?

**Verifier**

What about letting the prover simulate exactly one of them?
OR Composition

**Prover**

Goal: Prove that $\exists w$ such that $(x_0, w)$ in $R_0$ or $(x_1, w)$ in $R_1$

---

**Verifier**

- TRUE
- FALSE

- Sim: guess $b_3$
- output

- $b_0$

- $b_1$
- $c$

$V$ checks each proof also checks that $c = b_0 \oplus b_3$. 
OR Composition

Prover Simulator

Goal: Prove that $w$ such that $(x_0, w)$ in $R_0$ or $(x_1, w)$ in $R_1$

Guess $(b_0, b_1)$

$\text{Sim}(b_0) \quad \text{Sim}(b_1)$

If $c = b_0 \oplus b_1$,

$\text{c}$
Application: Encrypting bits

**Prover**

NP language = \{ (g,h,v,w) : \exists (b,c) such that b \in \{0,1\} and v = g^c, w = h^c.g^b \}

\((g,h,u,v)\) is a DH tuple

**Verifier**

\((g,h,u,v) is a DH tuple\)

\((g,h,u,v) is a DH tuple\)

\((g,h,u,v) is a DH tuple\)
Non-Interactive Zero-Knowledge
Non-Interactive Zero-Knowledge: Fiat-Shamir

Prover

Verifier

com₁, com₂, ..., comₙ
ch₁, ch₂, ..., chₙ

RO

cum₁, cum₂, ..., cumₙ
ch₁, ..., chₙ
open₁, open₂, ..., openₙ
Non-Interactive Zero-Knowledge: Fiat-Shamir

Prover

Verifier

if \( ch \neq b \)

\[ \text{Sim}(b), b, \text{opening} \]
Pairings
Pairing-based cryptography

- So far, we’ve looked at hard problems like discrete log, CDH, HDH, DDH in groups
- Certain groups have an additional structure

- Let $G_0, G_1, G_T$ be 3 cyclic groups of prime order
  - where $g_0 \in G_0$ and $g_1 \in G_1$ are generators
  - base groups

- A pairing is an efficiently computable function $e: G_0 \times G_1 \to G_T$ such that:
  1. $g_T = e(g_0, g_1)$ is a generator of $G_T$
  2. For all $(u, u') \in G_0$ and $(v, v') \in G_1$,
     
        $e(u,u', v) = e(u,v).e(u',v)$ and $e(u, v'.v') = e(u,v).e(u,v')$

\[
\begin{align*}
e(u,v) \cdot e(u',v) &= e(u.u',v) \\
e(u,v) \cdot e(u,v') &= e(u,v.v')
\end{align*}
\]
Pairing-based cryptography

- A pairing is an efficiently computable function $e: G_0 \times G_1 \rightarrow G_T$ such that:

  1. $g_T = e(g_0, g_1)$ is a generator of $G_T$

  2. For all $(u, u') \in G_0$ and $(v, v') \in G_1$, $e(u, u', v) = e(u, v) . e(u', v)$ and $e(u, v, v') = e(u, v) . e(u, v')$

- Consequences: $e(g_0^a, g_1^b) = (e(g_0, g_1))^{ab} = e(g_0^b, g_1^a)$

  $e(g_0^2, g_1^b) = e(g_0, g_1^b) . e(g_0, g_1^b)$  

  Generalized: $e(g_0^a, g_1^b) = (e(g_0, g_1^b))^a$

  Similarly, $e(g_0, g_1^b) = (e(g_0, g_1^b))^{ab}$
**Normal Groups**

\[ (g^a)^b = g^{ab} \]

**Pairing Groups:**

\[ g_0^a g_1^b : \text{Easy to compute} \quad g_T^{ab} \]

**Pairing-based cryptography**

- A pairing is an efficiently computable function \( e : G_0 \times G_1 \rightarrow G_T \) such that:
  1. \( g_T = e(g_0, g_1) \) is a generator of \( G_T \)
  2. For all \( (u, u') \in G_0 \) and \( (v, v') \in G_1 \),
     \[ e(u, u') \cdot v = e(u, v) \cdot e(u', v) \quad \text{and} \quad e(u, v' \cdot v) = e(u, v) \cdot e(u, v') \]

- Consequences: when \( G_0 = G_1 \), then DDH in \( G_0 \) is easy.

**DDH assumption**

\[ (u, v, w) \approx (g_0^a, g_0^b, g_0^c) \] for random \( c \).

**Recall:**

\[ e(g_0^a, g_0^b) \approx (e(g_0, g_0))^{ab} \cdot e(g_0^c, g_0)^c \]
A Useful Hardness Assumption

- **Co-CDH assumption:**
  - Sample random \((a, b)\) in \(\mathbb{Z}_q\)
  - \(u_0 = g_0^a, u_1 = g_1^a, v_0 = g_0^b, z\)
  - Send \((u_0, u_1, v_0)\) to \(A\)
  - \(A\) outputs \(z' \in G_0\)
  - \(A\) wins if \(z' = g_0^{ab}\)
Warmup: BLS Signatures

• Constructed as:

1. **KeyGen** \((1^k)\): Sample random \(a\), output \((sk = a), (pk = g_1^a)\).

2. **Sign** \((sk = a, m)\): \(\sigma = (H(m))^a \in G_0\)

3. **Verify** \((pk, m, \sigma)\): Output 1 iff \(e(H(m), pk) = e(\sigma, g_1)\).
Warmup: BLS Signatures

1. KeyGen ($1^k$): Sample random $a$, output $(sk = a), (pk = g_1^a)$.

2. Sign $(sk = a, m)$: $\sigma = (H(m))^a \in G_0$

3. Verify $(pk, m, \sigma)$: Output 1 iff $e(H(m), pk) = e(\sigma, g_1)$. 
BLS Signatures - Aggregation

- Constructed as:
  1. KeyGen ($1^k$): Sample random $a$, output (sk = a), (pk = $g_1^a \in G_1$)
  2. Sign (sk = a, m): $\sigma = (H(m))^a \in G_0$
  3. Verify (pk, m, $\sigma$): Output 1 iff $e(H(m), pk) = e(\sigma, g_1)$.

  4. SignAgg (pk$_1$, ...pk$_n$, $\sigma_1$, ... $\sigma_n$): $\sigma_{agg} = \sigma_1 \cdot \sigma_2 \cdot ... \cdot \sigma_n \in G_0$

  5. VerifyAgg (pk$_1$, ...pk$_n$, m$_1$, ...m$_n$, $\sigma_1$, ... $\sigma_n$):
BLS Signatures - Aggregation

1. SignAgg \((pk_1, ... pk_n, \sigma_1, ... \sigma_n)\): \(\sigma_{agg} = \sigma_1 \cdot \sigma_2 \cdot ... \cdot \sigma_n \in G_0\)

2. VerifyAgg \((pk_1, ... pk_n, m_1, ... m_n, \sigma_1, ... \sigma_n)\):
Next time:
Zero-Knowledge from Pairings