Signatures
Signatures

Signatures with various functionality/properties
Signatures

- Signatures with various functionality/properties
- Constructions come in different flavors (we’ll sample each flavor):
Signatures

- Signatures with various functionality/properties
- Constructions come in different flavors (we’ll sample each flavor):
  - Simple and efficient ones in the Random Oracle Model
Signatures

Signatures with various functionality/properties

Constructions come in different flavors (we’ll sample each flavor):

- Simple and efficient ones in the Random Oracle Model
- Relatively efficient ones under specific assumptions (often relatively strong/new assumptions)
Signatures

Signatures with various functionality/properties

Constructions come in different flavors (we’ll sample each flavor):

Simple and efficient ones in the Random Oracle Model

Relatively efficient ones under specific assumptions (often relatively strong/new assumptions)

Using minimal/general assumptions, often simple, but not very efficient (e.g., involving NIZK for general NP statements)
Signatures

Signatures with various functionality/properties

Constructions come in different flavors (we’ll sample each flavor):

- Simple and efficient ones in the Random Oracle Model
- Relatively efficient ones under specific assumptions (often relatively strong/new assumptions)
- Using minimal/general assumptions, often simple, but not very efficient (e.g., involving NIZK for general NP statements)
- Definitions sometimes have subtleties (not all of them have ideal functionality specifications)
Multi-Signatures
Multi-Signatures

- Multiple signers signing the same message
Multi-Signatures

- Multiple signers signing the same message
- Each signer has an (SK,VK) pair
Multi-Signatures

- Multiple signers signing the same message
- Each signer has an (SK,VK) pair
- Resulting signature must be “compact”: size independent of the number of signers
Multi-Signatures

- Multiple signers signing the same message
- Each signer has an (SK, VK) pair
- Resulting signature must be “compact”: size independent of the number of signers
- Security requirement: Unforgeability (chosen message security)
Multi-Signatures

- Multiple signers signing the same message
- Each signer has an (SK,VK) pair
- Resulting signature must be “compact”: size independent of the number of signers
- Security requirement: Unforgeability (chosen message security)
- Adversary can collude with all but one signer
Schnorr Signature
Schnorr Signature

A regular (i.e., non-multi) digital signature scheme secure in the Random Oracle model under the discrete log assumption
Schnorr Signature

A regular (i.e., non-multi) digital signature scheme secure in the Random Oracle model under the discrete log assumption

**KeyGen:** Signing key is $x$ and Verification key is $X = g^x$
Schnorr Signature

A regular (i.e., non-multi) digital signature scheme secure in the Random Oracle model under the discrete log assumption

KeyGen: Signing key is $x$ and Verification key is $X = g^x$

Sign($m;x$): compute $R = g^r$, $h = H(m,R)$, $s = r + hx$. Output $(h,s)$
Schnorr Signature

A regular (i.e., non-multi) digital signature scheme secure in the Random Oracle model under the discrete log assumption

- **KeyGen**: Signing key is $x$ and Verification key is $X = g^x$
- **Sign($m$;$x$)**: compute $R = g^r$, $h = H(m, R)$, $s = r + hx$. Output $(h,s)$
- **Verify($m$,$(h,s)$;$X$)**: check if $h = H(m, g^s X^{-h})$
Schnorr Signature

A regular (i.e., non-multi) digital signature scheme secure in the Random Oracle model under the discrete log assumption

- **KeyGen**: Signing key is $x$ and Verification key is $X = g^x$
- **Sign($m;x$)**: compute $R = g^r$, $h = H(m,R)$, $s = r + hx$. Output $(h,s)$
- **Verify($m,(h,s);X$)**: check if $h = H(m,g^sX^{-h})$

Alternately Sign($m;x$) outputs $(R,s)$. Verify($m,(R,s);X$) computes $h = H(m,R)$ and checks if $g^s = RX^h$
Schnorr Signature

A regular (i.e., non-multi) digital signature scheme secure in the Random Oracle model under the discrete log assumption

**KeyGen:** Signing key is $x$ and Verification key is $X = g^x$

**Sign($m$; $x$):** compute $R = g^r$, $h = H(m, R)$, $s = r + hx$. Output $(h, s)$

**Verify($m$,$(h, s)$; $X$):** check if $h = H(m, g^sX^{-h})$

Alternately Sign($m$; $x$) outputs $(R, s)$. Verify($m$, $(R, s)$; $X$) computes $h = H(m, R)$ and checks if $g^s = RX^h$

**Security:** Mimics a (concurrent) ZK PoK of $x$
Schnorr Signature

A regular (i.e., non-multi) digital signature scheme secure in the Random Oracle model under the discrete log assumption

**KeyGen:** Signing key is $x$ and Verification key is $X = g^x$

**Sign**($m;x$): compute $R = g^r$, $h = H(m, R)$, $s = r + hx$. Output $(h, s)$

**Verify**($m,(h,s);X$): check if $h = H(m, g^sX^{-h})$

Alternately **Sign**($m;x$) outputs $(R, s)$. **Verify**($m,(R,s);X$) computes $h = H(m, R)$ and checks if $g^s = RX^h$

**Security:** Mimics a (concurrent) ZK PoK of $x$

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 Signature

A regular (i.e., non-multi) digital signature scheme secure in the Random Oracle model under the discrete log assumption

**KeyGen:** Signing key is $x$ and Verification key is $X = g^x$

**Sign(m;x):** compute $R = g^r$, $h = H(m, R)$, $s = r + hx$. Output $(h, s)$

**Verify(m,(h,s);X):** check if $h = H(m, g^sX^{-h})$

Alternately $\text{Sign}(m;x)$ outputs $(R,s)$. $\text{Verify}(m,(R,s);X)$ computes $h = H(m, R)$ and checks if $g^s = RX^h$

**Security:** Mimics a (concurrent) ZK PoK of $x$

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 $x$

Extended to a multi-signature scheme [BN’06] →
A Multi-Signature Scheme
A Multi-Signature Scheme

Schnorr: $\text{Sign}(m;x) = (R,s)$ where $R=g^r$, $s = r + hx$ for $h=H(m,R)$. 
$\text{Verify}(m,(R,s);X)$ checks if $g^s = RX^h$ for $h = H(m,R)$
A Multi-Signature Scheme

Schnorr: Sign(m;x) = (R,s) where R=g^r, s = r + hx for h=H(m,R). Verify(m,(R,s);X) checks if g^s = RX^h for h = H(m,R)

For multiple signers with keys X_1,...,X_n can create an “aggregated” signature (R,s) such that g^s = R.X_1^{h_1}...X_n^{h_n}, where:
A Multi-Signature Scheme

Schnorr: \( \text{Sign}(m;x) = (R,s) \) where \( R = g^r, \ s = r + hx \) for \( h = H(m,R) \). Verify\((m,(R,s);X)\) checks if \( g^s = RX^h \) for \( h = H(m,R) \)

For multiple signers with keys \( X_1,\ldots,X_n \) can create an “aggregated” signature \((R,s)\) such that \( g^s = R.X_1^{h_1}\ldots X_n^{h_n} \), where:

- Pick \( R \): each party picks \( r_i \) and publishes \( g^{r_i} \). Set \( R = g^{r_1+\ldots+r_n} \)
A Multi-Signature Scheme

Schnorr: Sign(m;x) = (R,s) where R=g^r, s = r + hx for h=H(m,R). Verify(m,(R,s);X) checks if g^s = RX^h for h = H(m,R)

For multiple signers with keys X_1,...,X_n can create an “aggregated” signature (R,s) such that g^s = R.X_1^{h_1}...X_n^{h_n}, where:

- Pick R: each party picks r_i and publishes g^{r_i}. Set R = g^{r_1+...+r_n}
- Ensure simultaneous announcement of g^{r_i}. (Commit & reveal.)
A Multi-Signature Scheme

Schnorr: Sign(m;x) = (R,s) where R=g^r, s = r + hx for h=H(m,R). Verify(m,(R,s);X) checks if g^s = RX^h for h = H(m,R)

For multiple signers with keys X_1,...,X_n can create an “aggregated” signature (R,s) such that g^s = R.X_1^{h_1}...X_n^{h_n}, where:

- Pick R: each party picks r_i and publishes g^{ri}. Set R = g^{r_1+...+r_n}
- Ensure simultaneous announcement of g^{ri}. (Commit & reveal.)
- h_i = H(m,R,X_i,L), where L = <X_1,...,X_n>
A Multi-Signature Scheme

Schnorr: Sign(m;x) = (R,s) where R=g^r, s = r + hx for h=H(m,R). Verify(m,(R,s);X) checks if g^s = RX^h for h = H(m,R)

For multiple signers with keys X_1,...,X_n can create an “aggregated” signature (R,s) such that g^s = R.X_1^{h_1}...X_n^{h_n}, where:

- Pick R: each party picks r_i and publishes g^{r_i}. Set R = g^{r_1+...+r_n}
- Ensure simultaneous announcement of g^{r_i}. (Commit & reveal.)
- h_i = H(m,R,X_i,L), where L = <X_1,...,X_n>
- Then, sequentially s_i = s_{i-1} + r_i + h_ix_i (starting with s_0 = 0)
A Multi-Signature Scheme

Schnorr: Sign(m;x) = (R,s) where R=g^r, s = r + hx for h=H(m,R). Verify(m,(R,s);X) checks if g^s = RX^h for h = H(m,R).

For multiple signers with keys X_1,...,X_n can create an “aggregated” signature (R,s) such that g^s = R.X_1^{h_1}...X_n^{h_n}, where:

- Pick R: each party picks r_i and publishes g^{r_i}. Set R = g^{r_1+...+r_n}
- Ensure simultaneous announcement of g^{r_i}. (Commit & reveal.)
- h_i = H(m,R,X_i,L), where L = <X_1,...,X_n>
- Then, sequentially s_i = s_{i-1} + r_i + h_ix_i (starting with s_0 = 0)
- So that final signature s_n = r + h_1x_1 + ... + h_nx_n where R= g^r
Aggregate Signatures
Aggregate Signatures

Generalization of multi-signatures where multiple signers may have different messages
Aggregate Signatures

- Generalization of multi-signatures where multiple signers may have different messages
- Sequential aggregation: each signer gets the aggregated signature so far and adds her signature into it
Aggregate Signatures

- Generalization of multi-signatures where multiple signers may have different messages.
- Sequential aggregation: each signer gets the aggregated signature so far and adds her signature into it.
- General aggregation: signatures can be created independently and then aggregated in arbitrary order.
Waters Signature
Waters Signature

A regular (non-aggregate) signature scheme that is secure if the Computational Diffie-Hellman assumption holds in a group with bilinear pairings (no RO)
Waters Signature

A regular (non-aggregate) signature scheme that is secure if the Computational Diffie-Hellman assumption holds in a group with bilinear pairings (no RO)

**Keys:** Signing key is $x$ and verification key is $X := e(g,g)^x$, and generators $u_0, u_1, ..., u_k$ (for $k$ bit long messages)
Waters Signature

A regular (non-aggregate) signature scheme that is secure if the Computational Diffie-Hellman assumption holds in a group with bilinear pairings (no RO)

**Keys:** Signing key is $x$ and verification key is $X := e(g,g)^x$, and generators $u_0,u_1,\ldots,u_k$ (for $k$ bit long messages)

**Sign($m;x$) = ($R,S$)** where $R = g^r$ and $S = g^x H^r$, where $H = \pi(m) = u_0.u_1^{m_1}\ldots u_k^{m_k}$
Waters Signature

A regular (non-aggregate) signature scheme that is secure if the Computational Diffie-Hellman assumption holds in a group with bilinear pairings (no RO)

**Keys:** Signing key is $x$ and verification key is $X := e(g,g)^x$, and generators $u_0, u_1, \ldots, u_k$ (for $k$ bit long messages)

**Sign**($m; x$) = ($R, S$) where $R = g^r$ and $S = g^x H^r$, where $H = \pi(m) = u_0 u_1^{m_1} \ldots u_k^{m_k}$

**Verify**($m, (R, S); X, u, u_1, \ldots, u_k$): check $e(S, g) = e(R, H).X$
Waters Signature

A regular (non-aggregate) signature scheme that is secure if the Computational Diffie-Hellman assumption holds in a group with bilinear pairings (no RO)

**Keys:** Signing key is \( x \) and verification key is \( X := e(g,g)^x \), and generators \( u_0, u_1, \ldots, u_k \) (for \( k \) bit long messages)

**Sign** \((m; x) = (R, S)\) where \( R = g^r \) and \( S = g^x H^r \), where 
\[ H = \pi(m) = u_0 u_1 m_1 \ldots u_k m_k \]

**Verify** \((m, (R, S); X, u, u_1, \ldots, u_k)\): check \( e(S, g) = e(R, H).X \)

Extended to a sequential aggregate scheme [LOSSW’06]
A Sequential Aggregate Signature Scheme
A Sequential Aggregate Signature Scheme

**Keys:** For user $i$ verification key is $X_i := e(g,g)^{x_i}$, and $u_{i0}, u_{i1}, \ldots, u_{ik}$. Signing key is $x_i$ and $y_{i0}, y_{i1}, \ldots, y_{ik}$ where $u_{ij} = g^{y_{ij}}$
A Sequential Aggregate Signature Scheme

**Keys:** For user i verification key is $X_i := e(g,g)^{x_i}$, and $u_{i0}, u_{i1},..., u_{ik}$.
Signing key is $x_i$ and $y_{i0}, y_{i1},..., y_{ik}$ where $u_{ij} = g^{y_{ij}}$

**Signature** = $(R,S)$, where $R=g^{r_1+...+r_n}$, $S = g^{x_1+...+x_n} (H_1 ... H_n)^{r_1+...+r_n}$
where $H_i = u_{i0}.(u_{i1})^{m_1}...(u_{ik})^{m_k}$
A Sequential Aggregate Signature Scheme

**Keys:** For user i verification key is $X_i := e(g,g)^{x_i}$, and $u_{i0}, u_{i1}, \ldots, u_{ik}$. Signing key is $x_i$ and $y_{i0}, y_{i1}, \ldots, y_{ik}$ where $u_{ij} = g^{y_{ij}}$

**Signature** = $(R, S)$, where $R = g^{r_1+\ldots+r_n}$, $S = g^{x_1+\ldots+x_n} (H_1 \ldots H_n)^{r_1+\ldots+r_n}$ where $H_i = u_{i0}.(u_{i1})^{m_1} \ldots (u_{ik})^{m_k}$

**Verification of signature** $(R, S)$ for messages $(m_1, \ldots, m^n)$: check if $e(S, g) = e(R, H_1)X_1 \ldots e(R, H_n)X_n$
A Sequential Aggregate Signature Scheme

**Keys:** For user $i$ verification key is $X_i := e(g,g)^{x_i}$, and $u_{i0}, u_{i1}, \ldots, u_{ik}$. Signing key is $x_i$ and $y_{i0}, y_{i1}, \ldots, y_{ik}$ where $u_{ij} = g^{y_{ij}}$

**Signature** = $(R, S)$, where $R = g^{r_1+\ldots+r_n}$, $S = g^{x_1+\ldots+x_n} (H_1 \ldots H_n)^{r_1+\ldots+r_n}$ where $H_i = u_{i0}(u_{i1})^{m_1} \ldots (u_{ik})^{m_k}$

**Verification of signature** $(R, S)$ for messages $(m_1, \ldots, m_n)$: check if $e(S,g) = e(R,H_1)X_1 \ldots e(R,H_n)X_n$

Signing done sequentially by individual signers. Initially set $R=1$ and $S = 1$ (identity in the group). Then:
A Sequential Aggregate Signature Scheme

**Keys:** For user i verification key is $X_i := e(g,g)^{x_i}$, and $u_{i0}, u_{i1}, ..., u_{ik}$. Signing key is $x_i$ and $y_{i0}, y_{i1}, ..., y_{ik}$ where $u_{ij} = g^{y_{ij}}$

**Signature** = $(R,S)$, where $R = g^{r_1 + ... + r_n}$, $S = g^{x_1 + ... + x_n} (H_1 ... H_n)^{r_1 + ... + r_n}$ where $H_i = u_{i0} (u_{i1})^{m_1} ... (u_{ik})^{m_k}$

**Verification of signature** $(R,S)$ for messages $(m_1, ..., m_n)$: check if $e(S,g) = e(R,H_1)X_1 ... e(R,H_n)X_n$

Signing done sequentially by individual signers. Initially set $R=1$ and $S = 1$ (identity in the group). Then:

**AddSign**($m_i, (R’, S’)$; $x_i, y_{i0}, y_{i1}, ..., y_{ik}$) = ReRand($R”, S”$), where $R” = R’$ and $S” = S’g^{x_i}(R’)^{h_i}$ where $h_i$ s.t. $g^{h_i} = H_i$
A Sequential Aggregate Signature Scheme

**Keys:** For user i verification key is $X_i := e(g,g)^{x_i}$, and $u_{i0}, u_{i1}, ..., u_{ik}$. Signing key is $x_i$ and $y_{i0}, y_{i1}, ..., y_{ik}$ where $u_{ij} = g^{y_{ij}}$

**Signature** = $(R,S)$, where $R = g^{r_{1+...+rn}}$, $S = g^{x_{1+...+xn}} (H_1 ... H_n)^{r_{1+...+rn}}$

where $H_i = u_{i0}.(u_{i1})^{m_1}...(u_{ik})^{m_k}$

**Verification of signature** $(R,S)$ for messages $(m_1, ..., m^n)$: check if $e(S,g) = e(R,H_1)X_1 ... e(R,H_n)X_n$

Signing done sequentially by individual signers. Initially set $R=1$ and $S = 1$ (identity in the group). Then:

**AddSign**$(m_i,(R',S')); x_i, y_{i0}, y_{i1}, ..., y_{ik}) = \text{ReRand}(R'',S''),$ where $R'' = R'$ and $S'' = S'.g^{x_i}.(R')^{h_i}$ where $h_i$ s.t. $g^{h_i} = H_i$

**ReRand**$(R'',S'') = (R,S)$, where $R = R''g^\dagger$ and $S = S'' (H_1..H_i)^\dagger$
Batch Verification
Batch Verification

To speed up verification of a collection of signatures
Batch Verification

- To speed up verification of a collection of signatures
- Batching done by the verifier
Batch Verification

- To speed up verification of a collection of signatures
- Batching done by the verifier
- Incomparable to aggregate signatures
Batch Verification

- To speed up verification of a collection of signatures
  - Batching done by the verifier
  - Incomparable to aggregate signatures
  - Batch verifiable signature scheme reduces verification time, but does not reduce the total size of signatures that verifier gets. No co-ordination among signers.
Batch Verification

To speed up verification of a collection of signatures

Batching done by the verifier

Incomparable to aggregate signatures

Batch verifiable signature scheme reduces verification time, but does not reduce the total size of signatures that verifier gets. No co-ordination among signers.

Aggregate signatures saves on bandwidth and verification time, but needs coordination among signers and does not allow un-aggregating the signatures
Batch Verification
Batch Verification

Idea: to verify several equations of the form $Z_i = g^{z_i}$, pick random weights $w_i$ and check $\prod_i Z_i^{w_i} = g^{\sum z_i w_i}$
Batch Verification

Idea: to verify several equations of the form $Z_i = g^{z_i}$, pick random weights $w_i$ and check $\prod_i Z_i^{w_i} = g^{\sum z_i w_i}$

If one (or more) equation is wrong, probability of verifying is at most $1/q$, where $q$ is the size of the domain of $w_i$
Batch Verification

Idea: to verify several equations of the form $Z_i = g^{z_i}$, pick random weights $w_i$ and check $\prod_i Z_i^{w_i} = g^{\sum z_i w_i}$

If one (or more) equation is wrong, probability of verifying is at most $1/q$, where $q$ is the size of the domain of $w_i$

Efficiency by using a small domain for $w_i$. e.g., use $w_i \in \{0,1\}$, and repeat $k$ times (independent of number of signatures)
Batch Verification

Idea: to verify several equations of the form $Z_i = g^{z_i}$, pick random weights $w_i$ and check $\prod_i Z_i^{w_i} = g^{\sum z_i w_i}$

If one (or more) equation is wrong, probability of verifying is at most $1/q$, where $q$ is the size of the domain of $w_i$

Efficiency by using a small domain for $w_i$. e.g., use $w_i \in \{0,1\}$, and repeat $k$ times (independent of number of signatures)

Similarly for pairing equations, but with further optimizations
**Batch Verification**

Idea: to verify several equations of the form $Z_i = g^{z_i}$, pick random weights $w_i$ and check $\prod_i Z_i^{w_i} = g^{\sum z_i w_i}$

- If one (or more) equation is wrong, probability of verifying is at most $1/q$, where $q$ is the size of the domain of $w_i$

- Efficiency by using a small domain for $w_i$. e.g., use $w_i \in \{0,1\}$, and repeat $k$ times (independent of number of signatures)

- Similarly for pairing equations, but with further optimizations

- e.g. Waters’ signature: $e(S,g)=e(R,H).X$ (g same for all signers)
Batch Verification

Idea: to verify several equations of the form $Z_i = g^{z_i}$, pick random weights $w_i$ and check $\prod_i Z_i^{w_i} = g^{\sum z_i w_i}$

• If one (or more) equation is wrong, probability of verifying is at most $1/q$, where $q$ is the size of the domain of $w_i$

• Efficiency by using a small domain for $w_i$. e.g., use $w_i \in \{0,1\}$, and repeat $k$ times (independent of number of signatures)

• Similarly for pairing equations, but with further optimizations

  e.g. Waters' signature: $e(S, g) = e(R, H).X$ (g same for all signers)

  Can save on number of pairing operations using
  $\prod_i e(S_i, g)^{w_i} = \prod_i e(S_i^{w_i}, g) = e(\prod_i S_i^{w_i}, g)$
Group Signatures
Group Signatures

To sign a message “anonymously” [CvH’91]
Group Signatures

To sign a message “anonymously” [CvH’91]

Signature shows that message was signed by some member of a group
Group Signatures

To sign a message “anonymously” [CvH’91]

Signature shows that message was signed by some member of a group

But a group manager can “trace” the signer
Group Signatures

To sign a message “anonymously” [CvH’91]

- Signature shows that message was signed by some member of a group
- But a group manager can “trace” the signer
- However, the group manager or other group members “cannot frame” a member
Group Signatures
Group Signatures

**Full-Anonymity**: Adversary gives \((m, ID_0, ID_1)\) and gets back \(\text{Sign}(m; ID_b)\) for a random bit \(b\). Advantage of the adversary in finding \(b\) should be negligible.
Group Signatures

**Full-Anonymity**: Adversary gives \((m, ID_0, ID_1)\) and gets back \(\text{Sign}(m; ID_b)\) for a random bit \(b\). Advantage of the adversary in finding \(b\) should be negligible.

Adversary knows secret keys of all group-members, and has oracle access to the “tracing algorithm” (but not allowed to query it on the challenge)
Group Signatures

**Full-Anonymity**: Adversary gives \((m, ID_0, ID_1)\) and gets back \(\text{Sign}(m; ID_b)\) for a random bit \(b\). Advantage of the adversary in finding \(b\) should be negligible.

Adversary knows secret keys of all group-members, and has oracle access to the “tracing algorithm” (but not allowed to query it on the challenge)

Implies unlinkability (can’t link signatures from same user)
Group Signatures

**Full-Anonymity**: Adversary gives \((m, \text{ID}_0, \text{ID}_1)\) and gets back \(\text{Sign}(m; \text{ID}_b)\) for a random bit \(b\). Advantage of the adversary in finding \(b\) should be negligible.

Adversary knows secret keys of all group-members, and has oracle access to the “tracing algorithm” (but not allowed to query it on the challenge)

Implies unlinkability (can’t link signatures from same user)

**Full-Traceability**: If a set of group members collude and create a valid signature, the tracing algorithm will trace at least one member of the set. This holds even if the group manager is passively corrupt.
Group Signatures

**Full-Anonymity**: Adversary gives \((m, ID_0, ID_1)\) and gets back \(\text{Sign}(m; ID_b)\) for a random bit \(b\). Advantage of the adversary in finding \(b\) should be negligible.

- Adversary knows secret keys of all group-members, and has oracle access to the “tracing algorithm” (but not allowed to query it on the challenge)
- **Implies unlinkability** (can’t link signatures from same user)

**Full-Traceability**: If a set of group members collude and create a valid signature, the tracing algorithm will trace at least one member of the set. This holds even if the group manager is passively corrupt.

- **Implies unforgeability** (i.e., with no group members colluding with it, adversary cannot produce a valid signature) and **framing-resistance** (even colluding with the group manager)
Group Signatures
Group Signatures

A general construction: using a digital signature scheme, a CCA secure encryption scheme, and a “simulation-sound” NIZK [BMW’03]
Group Signatures

A general construction: using a digital signature scheme, a CCA secure encryption scheme, and a “simulation-sound” NIZK [BMW’03]

Each member’s signing key $SK^*_i = (SK_i, VK_i, ID_i, \sigma)$ where $(SK_i, VK_i)$ are signing/verification keys, $PK_i$ is an encryption key and $\sigma$ is a signature (w.r.t. $VK_{\text{group}}$) in from the group-manager on $(VK_i, ID_i)$
Group Signatures

A general construction: using a digital signature scheme, a CCA secure encryption scheme, and a “simulation-sound” NIZK [BMW’03]

Each member’s signing key $SK^*_i = (SK_i, VK_i, ID_i, \sigma)$ where $(SK_i, VK_i)$ are signing/verification keys, $PK_i$ is an encryption key and $\sigma$ is a signature (w.r.t. $VK_{group}$) in from the group-manager on $(VK_i, ID_i)$

Group signature’s verification key = $(VK_{group}, PK_{group}, CRS_{group})$
Group Signatures

A general construction: using a digital signature scheme, a CCA secure encryption scheme, and a “simulation-sound” NIZK [BMW’03]

Each member’s signing key $SK^*_i = (SK_i, VK_i, ID_i, \sigma)$ where $(SK_i, VK_i)$ are signing/verification keys, $PK_i$ is an encryption key and $\sigma$ is a signature (w.r.t. $VK_{\text{group}}$) in from the group-manager on $(VK_i, ID_i)$

Group signature's verification key = $(VK_{\text{group}}, PK_{\text{group}}, CRS_{\text{group}})$

Signature is $(C, \pi)$, where:
Group Signatures

A general construction: using a digital signature scheme, a CCA secure encryption scheme, and a “simulation-sound” NIZK [BMW’03]

Each member’s signing key \( \text{SK}^*_i = (\text{SK}_i, \text{VK}_i, \text{ID}_i, \sigma) \) where \((\text{SK}_i, \text{VK}_i)\) are signing/verification keys, \(\text{PK}_i\) is an encryption key and \(\sigma\) is a signature (w.r.t. \(\text{VK}_{\text{group}}\)) in from the group-manager on \((\text{VK}_i, \text{ID}_i)\)

Group signature’s verification key = \((\text{VK}_{\text{group}}, \text{PK}_{\text{group}}, \text{CRS}_{\text{group}})\)

Signature is \((C, \pi)\), where:

\[ s = \text{Sign}(\text{message}; \text{SK}_i) \]
Group Signatures

A general construction: using a digital signature scheme, a CCA secure encryption scheme, and a “simulation-sound” NIZK [BMW’03]

Each member’s signing key $SK^*_i = (SK_i, VK_i, ID_i, \sigma)$ where $(SK_i, VK_i)$ are signing/verification keys, $PK_i$ is an encryption key and $\sigma$ is a signature (w.r.t. $VK_{group}$) in from the group-manager on $(VK_i, ID_i)$

Group signature’s verification key = $(VK_{group}, PK_{group}, CRS_{group})$

Signature is $(C, \pi)$, where:

$s = \text{Sign(message; } SK_i)$

$C = \text{Encrypt}_{PK_{group}}(s, SK^*_i)$
Group Signatures

A general construction: using a digital signature scheme, a CCA secure encryption scheme, and a “simulation-sound” NIZK [BMW’03]

Each member’s signing key $SK^*_i = (SK_i, VK_i, ID_i, \sigma)$ where $(SK_i, VK_i)$ are signing/verification keys, $PK_i$ is an encryption key and $\sigma$ is a signature (w.r.t. $VK_{\text{group}}$) in from the group-manager on $(VK_i, ID_i)$

Group signature’s verification key = $(VK_{\text{group}}, PK_{\text{group}}, CRS_{\text{group}})$

Signature is $(C, \pi)$, where:

- $s = \text{Sign}(\text{message}; SK_i)$
- $C = \text{Encrypt}_{PK_{\text{group}}}(s, SK^*_i)$
- $\pi = \text{a proof (w.r.t CRS}_{\text{group}})$ that $C$ is correct
Group Signatures

A general construction: using a digital signature scheme, a CCA secure encryption scheme, and a “simulation-sound” NIZK [BMW’03]

Each member’s signing key $SK^*_i = (SK_i, VK_i, ID_i, \sigma)$ where $(SK_i, VK_i)$ are signing/verification keys, $PK_i$ is an encryption key and $\sigma$ is a signature (w.r.t. $VK_{\text{group}}$) in from the group-manager on $(VK_i, ID_i)$

Group signature’s verification key = $(VK_{\text{group}}, PK_{\text{group}}, CRS_{\text{group}})$

Signature is $(C, \pi)$, where:

- $s = \text{Sign(message; } SK_i)$
- $C = \text{Encrypt}_{PK_{\text{group}}}(s, SK^*_i)$
- $\pi = \text{a proof } (\text{w.r.t CR}_S_{\text{group}}) \text{ that } C \text{ is correct}$

Tracing algorithm decrypts $C$ to find $SK^*_i$ and hence $ID_i$
Ring Signatures
Ring Signatures

For “leaking secrets”
Ring Signatures

- For “leaking secrets”
- Similar to group signatures, but with unwitting collaborators
Ring Signatures

For “leaking secrets”

Similar to group signatures, but with unwitting collaborators

i.e. the “ring” is not a priori fixed
Ring Signatures

For “leaking secrets”

Similar to group signatures, but with unwitting collaborators

i.e. the “ring” is not a priori fixed

And no manager who can trace the signer
Ring Signatures
Ring Signatures

Recall T-OWP/RO based signature
Ring Signatures

Recall T-OWP/RO based signature

\((SK, VK) = (F^{-1}, F)\)
Ring Signatures

Recall T-OWP/RO based signature

\((SK, VK) = (F^{-1}, F)\)

\(\text{Sign}(m; F^{-1}) = F^{-1}(H(m))\)
Ring Signatures

Recall T-OWP/RO based signature

- $(SK,VK) = (F^{-1}, F)$
- $\text{Sign}(m; F^{-1}) = F^{-1}(H(m))$
- $\text{Verify}(S; F):$ check if $H(m) = F(S)$
Recall T-OWP/RO based signature

\((SK, VK) = (F^{-1}, F)\)

\(\text{Sign}(m; F^{-1}) = F^{-1}(H(m))\)

\(\text{Verify}(S; F): \text{check if } H(m) = F(S)\)

Extended to a ring signature [RST’01]
Ring Signatures

Recall T-OWP/RO based signature

\[(SK, VK) = (F^{-1}, F)\]

\[\text{Sign}(m; F^{-1}) = F^{-1}(H(m))\]

\[\text{Verify}(S; F): \text{check if } H(m) = F(S)\]

Extended to a ring signature [RST’01]

\[\text{Verify}(m, (S_1, ..., S_n); (F_1, ..., F_n)) : \text{check } H(m) = F_1(S_1) + ... + F_n(S_n)\]
Ring Signatures

Recall T-OWP/RO based signature

$(SK, VK) = (F^{-1}, F)$

$Sign(m; F^{-1}) = F^{-1}(H(m))$

$Verify(S; F)$: check if $H(m) = F(S)$

Extended to a ring signature [RST’01]

$Verify(m, (S_1, ..., S_n); (F_1, ..., F_n))$ : check $H(m) = F_1(S_1) + ... + F_n(S_n)$

$Sign(m; F_1^{-1}, F_2, ..., F_n) = (S_1, ..., S_n)$ where $S_2, ..., S_n$ are random and $S_1 = F_1^{-1}(H(m) - F_2(S_2) - ... - F_n(S_n))$
Ring Signatures

Recall T-OWP/RO based signature

\((SK, VK) = (F^{-1}, F)\)

\(\text{Sign}(m; F^{-1}) = F^{-1}(H(m))\)

\(\text{Verify}(S; F): \text{check if } H(m) = F(S)\)

Extended to a ring signature [RST’01]

\(\text{Verify}(m, (S_1, \ldots, S_n); (F_1, \ldots, F_n)): \text{check } H(m) = F_1(S_1) + \ldots + F_n(S_n)\)

\(\text{Sign} (m; F_1^{-1}, F_2, \ldots, F_n) = (S_1, \ldots, S_n) \text{ where } S_2, \ldots, S_n \text{ are random and } S_1 = F_1^{-1} (H(m) - F_2(S_2) - \ldots - F_n(S_n))\)

Unwitting collaborators: \(F_i\)'s could be the verification keys for a standard signature scheme
Mesh Signatures
Mesh Signatures

Ring signature allows statements of the form
$(P_1 \text{ signed } m) \text{ or } (P_2 \text{ signed } m) \text{ or } \ldots \text{ or } (P_n \text{ signed } m)$
Mesh Signatures

- Ring signature allows statements of the form 
  \((P_1 \text{ signed } m) \text{ or } (P_2 \text{ signed } m) \text{ or } \ldots \text{ or } (P_n \text{ signed } m)\)

- Mesh signatures extend this to more complex statements
Mesh Signatures

Ring signature allows statements of the form
\((P_1 \text{ signed } m) \text{ or } (P_2 \text{ signed } m) \text{ or } \ldots \text{ or } (P_n \text{ signed } m)\)

Mesh signatures extend this to more complex statements

\(\text{e.g., } (P_1 \text{ signed } m_1) \text{ or } (P_2 \text{ signed } m_2) \text{ and } (P_3 \text{ signed } m_3)\)
Mesh Signatures

- Ring signature allows statements of the form
  \((P_1 \text{ signed } m) \text{ or } (P_2 \text{ signed } m) \text{ or } \ldots \text{ or } (P_n \text{ signed } m)\)

- Mesh signatures extend this to more complex statements

  - e.g., \((P_1 \text{ signed } m_1) \text{ or } ((P_2 \text{ signed } m_2) \text{ and } (P_3 \text{ signed } m_3))\)

  - e.g., some two out of the three statements \((P_1 \text{ signed } m_1), \text{ (}P_2 \text{ signed } m_2), \text{ (}P_3 \text{ signed } m_3)\) hold
Mesh Signatures

- Ring signature allows statements of the form $(P_1 \text{ signed } m)$ or $(P_2 \text{ signed } m)$ or .... or $(P_n \text{ signed } m)$

- Mesh signatures extend this to more complex statements

  - e.g., $(P_1 \text{ signed } m_1)$ or $(P_2 \text{ signed } m_2)$ and $(P_3 \text{ signed } m_3)$

  - e.g., some two out of the three statements $(P_1 \text{ signed } m_1)$, $(P_2 \text{ signed } m_2)$, $(P_3 \text{ signed } m_3)$ hold

- Signature is produced by the relevant parties collaborating
Mesh Signatures

- Ring signature allows statements of the form 
  \((P_1 \text{ signed } m) \text{ or } (P_2 \text{ signed } m) \text{ or } ... \text{ or } (P_n \text{ signed } m)\)

- Mesh signatures extend this to more complex statements
  
  e.g., \((P_1 \text{ signed } m_1) \text{ or } ((P_2 \text{ signed } m_2) \text{ and } (P_3 \text{ signed } m_3))\)

  e.g., some two out of the three statements \((P_1 \text{ signed } m_1), (P_2 \text{ signed } m_2), (P_3 \text{ signed } m_3)\) hold

- Signature is produced by the relevant parties collaborating

- Security requirements: Unforgeability and Hiding
Attribute-Based Signatures
Attribute-Based Signatures

“Claim-and-endorse”: Claim to have attributes satisfying a certain policy, and sign a message
Attribute-Based Signatures

“Claim-and-endorse”: Claim to have attributes satisfying a certain policy, and sign a message

Soundness: can’t forge, even by colluding
Attribute-Based Signatures

“Claim-and-endorse”: Claim to have attributes satisfying a certain policy, and sign a message

Soundness: can’t forge, even by colluding

Hiding: Verification without learning how the policy was satisfied
Attribute-Based Signatures

“Claim-and-endorse”: Claim to have attributes satisfying a certain policy, and sign a message

Soundness: can’t forge, even by colluding

Hiding: Verification without learning how the policy was satisfied

Also unlinkable: cannot link multiple signatures as originating from the same signer
Attribute-Based Signatures

“Claim-and-endorse”: Claim to have attributes satisfying a certain policy, and sign a message

- **Soundness**: can’t forge, even by colluding

- **Hiding**: Verification without learning how the policy was satisfied

  - Also unlinkable: cannot link multiple signatures as originating from the same signer

- **c.f. Mesh signatures**: here, instead of multiple parties signing a message, a single party with multiple attributes
Undeniable Signatures
Undeniable Signatures

Suppose Signer wants to control when/how often the signature can be verified, but signature is a commitment to a message.
Undeniable Signatures

Suppose Signer wants to control when/how often the signature can be verified, but signature is a commitment to a message

Verification is via an interactive protocol
Undeniable Signatures

Suppose Signer wants to control when/how often the signature can be verified, but signature is a commitment to a message.

Verification is via an interactive protocol.

It lets the signer verifiably accept or deny endorsing the message.
Undeniable Signatures

Suppose Signer wants to control when/how often the signature can be verified, but signature is a commitment to a message.

- Verification is via an interactive protocol.
- It lets the signer verifiably accept or deny endorsing the message.
- Signer refusing to deny can be taken as accepting.
Undeniable Signatures

Suppose Signer wants to control when/how often the signature can be verified, but signature is a commitment to a message.

Verification is via an interactive protocol.

It lets the signer verifiably accept or deny endorsing the message.

Signer refusing to deny can be taken as accepting.

Zero-knowledge verification: A verifier cannot transfer a signature that it verified.
Undeniable Signatures

Suppose Signer wants to control when/how often the signature can be verified, but signature is a commitment to a message.

Verification is via an interactive protocol.

It lets the signer verifiably accept or deny endorsing the message.

Signer refusing to deny can be taken as accepting.

Zero-knowledge verification: A verifier cannot transfer a signature that it verified.

Note: Still allows multiple (mutually distrusting) verifiers to be convinced if they run a secure MPC protocol to implement a virtual verifier.
Designated Verifier
Signatures
Designated Verifier
Signatures

Signature addressed to a single designated verifier
Designated Verifier Signatures

- Signature addressed to a single designated verifier
- Verifier cannot convince others of the validity of the signature
Designated Verifier Signatures

- Signature addressed to a single designated verifier
- Verifier cannot convince others of the validity of the signature
- e.g. a ring signature with a ring of size 2, containing the signer and the designated verifier
Today
Today

Signatures
Today

- Signatures
- Multi-signatures
Today

- Signatures
- Multi-signatures
- Aggregate Signatures
Today

- Signatures
  - Multi-signatures
  - Aggregate Signatures
  - Signatures with Batch verification
Today

- Signatures
- Multi-signatures
- Aggregate Signatures
- Signatures with Batch verification
- Group signatures
Today

- Signatures
  - Multi-signatures
  - Aggregate Signatures
  - Signatures with Batch verification
  - Group signatures
  - Ring and Mesh signatures
Today

- Signatures
  - Multi-signatures
  - Aggregate Signatures
  - Signatures with Batch verification
  - Group signatures
  - Ring and Mesh signatures
  - Attribute-Based signatures
Today

- Signatures
  - Multi-signatures
  - Aggregate Signatures
  - Signatures with Batch verification
  - Group signatures
  - Ring and Mesh signatures
  - Attribute-Based signatures
  - Undeniable signatures
Today

- Signatures
  - Multi-signatures
  - Aggregate Signatures
  - Signatures with Batch verification
  - Group signatures
  - Ring and Mesh signatures
  - Attribute-Based signatures
  - Undeniable signatures
  - Designated verifier signatures
Today

- Signatures
  - Multi-signatures
  - Aggregate Signatures
  - Signatures with Batch verification
  - Group signatures
  - Ring and Mesh signatures
  - Attribute-Based signatures
  - Undeniable signatures
  - Designated verifier signatures

- Next up: digital cash
Today

- Signatures
  - Multi-signatures
  - Aggregate Signatures
  - Signatures with Batch verification
  - Group signatures
  - Ring and Mesh signatures
  - Attribute-Based signatures
  - Undeniable signatures
  - Designated verifier signatures

Next up: digital cash

- Using Blind signatures and P-signatures