Foundations of Adaptor Signatures

Paul Gerhart, Dominique Schröder, Pratik Soni, Sri AravindaKrishnan Thyagarajan
Once Upon A Time

Alice
(wants to buy a witness for a statement $Y$)

$Y$

Bob
(knows a witness for $Y$ and wants to sell it)

$y$

wants to rely on minimum trust
does not like the ROM
Once Upon A Time

Alice
(wants to buy a witness for a statement $Y$)

Bob
(knows a witness for $Y$ and wants to sell it)

$Y$

$y$

wants to rely on minimum trust
does not like the ROM
Once Upon A Time

Alice
(wants to buy a witness for a statement $Y$)

$Y$

Bob
(knows a witness for $Y$ and wants to sell it)

$y$

wants to rely on minimum trust
does not like the ROM
Once Upon A Time

Alice
(wants to buy a witness for a statement \( Y \))

\( Y \)

Bob
(knows a witness for \( Y \) and wants to sell it)

\( y \)

wants to rely on minimum trust

does not like the ROM

Paul Gerhart
Adaptor Sigantures
Adaptor Signature Interfaces

### Signature Scheme

- Signature Scheme: $(pk, sk) \leftarrow \text{KGen}(\lambda)$
- Sign: $\sigma \leftarrow \text{Sign}(sk, m)$
- Vrfy: $b \leftarrow \text{Vrfy}(pk, m, \sigma)$

### Hard Relation

- Hard Relation: $(Y, y) \leftarrow \text{GenR}(\lambda)$
- pSign: $\tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y)$
- pVrfy: $b \leftarrow \text{pVrfy}(pk, m, \tilde{\sigma}, Y)$
- Adapt: $\sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y)$
- Extract: $y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y)$
## Adaptor Signature Interfaces

<table>
<thead>
<tr>
<th>Signature Scheme</th>
<th>$$(pk, sk) \leftarrow \text{KGen}(\lambda)$$</th>
<th>$$\sigma \leftarrow \text{Sign}(sk, m)$$</th>
<th>$$b \leftarrow \text{Vrfy}(pk, m, \sigma)$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard Relation</td>
<td>$$(Y, y) \leftarrow \text{GenR}(\lambda)$$</td>
<td>$$\tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y)$$</td>
<td>$$b \leftarrow \text{pVrfy}(pk, m, \tilde{\sigma}, Y)$$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$$\sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y)$$</td>
<td>$$y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y)$$</td>
</tr>
</tbody>
</table>
Adaptor Signature Interfaces

<table>
<thead>
<tr>
<th>Signature Scheme</th>
<th>((pk, sk) \leftarrow \text{KGen}(\lambda))</th>
<th>(\sigma \leftarrow \text{Sign}(sk, m))</th>
<th>(b \leftarrow \text{Vrfy}(pk, m, \sigma))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard Relation</td>
<td>((Y, y) \leftarrow \text{GenR}(\lambda))</td>
<td>(\tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y))</td>
<td>(b \leftarrow \text{pVrfy}(pk, m, \tilde{\sigma}, Y))</td>
</tr>
<tr>
<td></td>
<td>(\sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y))</td>
<td>(y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y))</td>
<td></td>
</tr>
</tbody>
</table>
Adaptor Signature Interfaces

<table>
<thead>
<tr>
<th>Signature Scheme</th>
<th>(pk, sk) ← KGen(λ)</th>
<th>σ ← Sign(sk, m)</th>
<th>b ← Vrfy(pk, m, σ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard Relation</td>
<td>(Y, y) ← GenR(λ)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>~σ ← pSign(sk, m, Y)</td>
<td>b ← pVrfy(pk, m, ~σ, Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>σ ← Adapt(pk, ~σ, y)</td>
<td></td>
<td>y ← Extract(~σ, σ, Y)</td>
</tr>
</tbody>
</table>
Adaptor Signature Interfaces

<table>
<thead>
<tr>
<th>Signature Scheme</th>
<th>((pk, sk) \leftarrow \text{KGen}(\lambda))</th>
<th>(\sigma \leftarrow \text{Sign}(sk, m))</th>
<th>(b \leftarrow \text{Vrfy}(pk, m, \sigma))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard Relation</td>
<td>((Y, y) \leftarrow \text{GenR}(\lambda))</td>
<td>(\tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y))</td>
<td>(b \leftarrow \text{pVrfy}(pk, m, \tilde{\sigma}, Y))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y))</td>
<td>(y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y))</td>
</tr>
</tbody>
</table>
### Adaptor Signature Interfaces

<table>
<thead>
<tr>
<th>Signature Scheme</th>
<th>( (pk, sk) \leftarrow \text{KGen}(\lambda) )</th>
<th>( \sigma \leftarrow \text{Sign}(sk, m) )</th>
<th>( b \leftarrow \text{Vrfy}(pk, m, \sigma) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard Relation</td>
<td>( (Y, y) \leftarrow \text{GenR}(\lambda) )</td>
<td>( \tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y) )</td>
<td>( b \leftarrow \text{pVrfy}(pk, m, \tilde{\sigma}, Y) )</td>
</tr>
<tr>
<td></td>
<td>( \sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y) )</td>
<td>( y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y) )</td>
<td></td>
</tr>
</tbody>
</table>
Fair Exchange using Adaptor Signatures

Alice
(wants to learn a witness)

\( \tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y) \)

\( y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y) \)

Bob
(wants to learn a signature)

\( (Y, y) \leftarrow \text{GenR}(\lambda) \)

\( \sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y) \)
Fair Exchange using Adaptor Signatures

Alice
(wants to learn a witness)

\( \tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y) \)

\( y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y) \)

Bob
(wants to learn a signature)

\( (Y, y) \leftarrow \text{GenR}(\lambda) \)

\( \sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y) \)
Alice
(wants to learn a witness)

\(\tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y)\)

\(y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y)\)

Bob
(wants to learn a signature)

\((Y, m)\)

\((Y, y) \leftarrow \text{GenR}(\lambda)\)

\(\tilde{\sigma} \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y)\)

\(\sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y)\)
Alice
(wants to learn a witness)

\( \tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y) \)

\( y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y) \)

Bob
(wants to learn a signature)

\( (Y, m) \leftarrow \text{GenR}(\lambda) \)

\( (Y, y) \leftarrow \text{GenR}(\lambda) \)

\( \sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y) \)

Fair Exchange using Adaptor Signatures
Fair Exchange using Adaptor Signatures

**Alice**
(wants to learn a witness)

\[
\tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y)
\]

\[
y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y)
\]

**Bob**
(wants to learn a signature)

\[
(Y, y) \leftarrow \text{GenR}(\lambda)
\]

\[
\sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y)
\]
Fair Exchange using Adaptor Signatures

Alice
(wants to learn a witness)

\(\tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y)\)

\(y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y)\)

Bob
(wants to learn a signature)

\((Y, m)\)

\((Y, y) \leftarrow \text{GenR}(\lambda)\)

\(\sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y)\)
Fair Exchange using Adaptor Signatures

Alice
(wants to learn a witness)

\[\tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y)\]

\[y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y)\]

Bob
(wants to learn a signature)

\[(Y, m) \leftarrow \text{GenR}(\lambda)\]

\[(Y, y) \leftarrow \text{GenR}(\lambda)\]

\[\sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y)\]
Fair Exchange using Adaptor Signatures

Alice
(wants to learn a witness)

\[ \tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y) \]

\[ y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y) \]

Bob
(wants to learn a signature)

\[ (Y, m) \]

\[ (Y, y) \leftarrow \text{GenR}(\lambda) \]

\[ \tilde{\sigma} \]

\[ \sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y) \]

5 | Foundations of Adaptor Signatures | Paul Gerhart
Fair Exchange using Adaptor Signatures

Alice
(wants to learn a witness)

\( \tilde{\sigma} \leftarrow \text{pSign}(sk, m, Y) \)

\( y \leftarrow \text{Extract}(\tilde{\sigma}, \sigma, Y) \)

Bob
(wants to learn a signature)

\( (Y, m) \leftarrow \text{GenR}(\lambda) \)

\( (Y, y) \leftarrow \text{GenR}(\lambda) \)

\( \tilde{\sigma} \leftarrow \text{Adapt}(pk, \tilde{\sigma}, y) \)

(\( \tilde{\sigma}, \sigma, Y \))
Adaptor Signatures in the Literature

- Introduced by Andrew Poelstra 2017
- Formally defined by Aumayr et al. [AEEFHMMR’21]

- Applications:
  - (Generalized) Payment Channels [AEEFHMMR’21]
  - (Blind) Coin Mixing [GMMMTT’22, QPMSESELYY’23]
  - Oracle-Based Payments [MTVFMM’23]

- Theory:
  - PQ Adaptors [TMM’20]
  - Stronger Definitions [DOY’22]
Adaptor Signatures in the Literature

• Introduced by Andrew Poelstra 2017
• Formally defined by Aumayr et al. [AEEFHMMR’21]
• Applications:
  – (Generalized) Payment Channels [AEEFHMMR’21]
  – (Blind) Coin Mixing [GMMMTT’22, QPMSESELYY’23]
  – Oracle-Based Payments [MTVFMM’23]
• Theory:
  – PQ Adaptors [TMM’20]
  – Stronger Definitions [DOY’22]
Adaptor Signatures in the Literature

- Introduced by Andrew Poelstra 2017
- Formally defined by Aumayr et al. [AEEFHMMR’21]
- Applications:
  - (Generalized) Payment Channels [AEEFHMMR’21]
  - (Blind) Coin Mixing [GMMMTT’22, QPMSESELYY’23]
  - Oracle-Based Payments [MTVFMM’23]
- Theory:
  - PQ Adaptors [TMM’20]
  - Stronger Definitions [DOY’22]
Theoretical Challenges

Given a signature scheme, building a secure adaptor signature is hard.

There is no secure adaptor signature in the standard model.
Theoretical Challenges

Given a signature scheme, building a secure adaptor signature is hard.

There is no secure adaptor signature in the standard model.
Adaptor signatures were formalized to build payment channels.

This formalization does not match the most recent applications.

Practical Challenges
Practical Challenges

Adaptor signatures were formalized to build payment channels.

This formalization does not match the most recent applications.
Adaptor signatures were formalized to build payment channels.

This formalization does not match the most recent applications.

Payment Channels

Oracle-Based Payments

Coin Mixing
Our Contribution

Gaps

Definitions

Constructions

Transparent Reductions
Our Contribution

- Gaps
- Definitions
- Constructions
- Transparent Reductions
Adaptor Signature Formalization

- The definition is a **one-shot experiment**
  - The adversary can only learn a single challenge pre-signature
- Adaptor signatures achieve only **existential unforgeability**, even if the signature scheme is strongly unforgeable
- The pre-signer **cannot influence** the statement
Adaptor Signature Formalization

- The definition is a one-shot experiment
  - The adversary can only learn a single challenge pre-signature

- Adaptor signatures achieve only existential unforgeability, even if the signature scheme is strongly unforgeable

- The pre-signer cannot influence the statement
Adaptor Signature Formalization

- The definition is a **one-shot experiment**
  - The adversary can only learn a single challenge pre-signature
- Adaptor signatures achieve only **existential unforgeability**, even if the signature scheme is strongly unforgeable
- The pre-signer **cannot influence** the statement
Adaptor Signature Formalization

- The definition is a **one-shot experiment**
  - The adversary can only learn a single challenge pre-signature
- Adaptor signatures achieve only **existential unforgeability**, even if the signature scheme is strongly unforgeable
- The pre-signer **cannot influence** the statement
Alice sends a payment when the oracle testifies for an event.

∀i ∈ {1, ..., M}:

\((Y_i, y_i) \leftarrow \text{RGen}(1^\lambda)\)

∀i ∈ {1, ..., M}:

\(\tilde{\sigma}_i \leftarrow \text{pSign}(sk, m, Y_i)\)

(y₁, ..., y₇) ➔

Oracles testify for events

Bob obtains pre-signatures from Alice and requests the oracle for testimony.

\(\tilde{\sigma}_{1 \leq i \leq M} \rightarrow \sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}_i, y_i)\)
Oracle-Based Conditional Payments

Alice sends a payment when the oracle testifies for an event

∀i ∈ {1, . . . , M} :
(Y_i, y_i) ← RGen(1^λ)

∀i ∈ {1, . . . , M} :
σ_i ← pSign(sk, m, Y_i)

Oracles testify for events

Bob obtains pre-signatures from Alice and requests the oracle for testimony

σ ← Adapt(pk, σ_i, y_i)
Alice sends a payment when the oracle testifies for an event

\[
\forall i \in \{1, \ldots, M\} : \\
(Y_i, y_i) \leftarrow \text{RGen}(1^\lambda)
\]

\[
\forall i \in \{1, \ldots, M\} : \\
\tilde{\sigma}_i \leftarrow \text{pSign}(\text{sk}, m, Y_i)
\]

Bob obtains pre-signatures from Alice and requests the oracle for testimony

\[
\sigma \leftarrow \text{Adapt}(\text{pk}, \tilde{\sigma}_i, y_i)
\]
Oracle-Based Conditional Payments [MTVFMS'22]

Alice sends a payment when the oracle testifies for an event

\[ \forall i \in \{1, \ldots, M\} : \]
\[ (Y_i, y_i) \leftarrow \text{RGen}(1^\lambda) \]

\[ \forall i \in \{1, \ldots, M\} : \]
\[ \tilde{\sigma}_i \leftarrow \text{pSign}(sk, m, Y_i) \]

Oracles testify for events

Bob obtains pre-signatures from Alice and requests the oracle for testimony

\[ \sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}_i, y_i) \]
Alice sends a payment when the oracle testifies for an event

\[ \forall i \in \{1, \ldots, M\} : (Y_i, y_i) \leftarrow \text{RGen}(1^\lambda) \]

\[ \forall i \in \{1, \ldots, M\} : \tilde{\sigma}_i \leftarrow \text{pSign}(\text{sk}, m, Y_i) \]

Oracles testify for events

Bob obtains pre-signatures from Alice and requests the oracle for testimony

\[ \sigma \leftarrow \text{Adapt}(\text{pk}, \tilde{\sigma}_i, y_i) \]
Alice sends a payment when the oracle testifies for an event

\[ \forall i \in \{1, \ldots, M\} : (Y_i, y_i) \leftarrow \text{RGen}(1^\lambda) \]

\[ \forall i \in \{1, \ldots, M\} : \tilde{\sigma}_i \leftarrow \text{pSign}(sk, m, Y_i) \]

Bob obtains pre-signatures from Alice and requests the oracle for testimony

\[ \tilde{\sigma}_{1 \leq i \leq M} \]

\[ \sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}_i, y_i) \]
Alice sends a payment when the oracle testifies for an event

\[
\forall i \in \{1, \ldots, M\} : (Y_i, y_i) \leftarrow \text{RGen}(1^\lambda)
\]

\[
\forall i \in \{1, \ldots, M\} : \tilde{\sigma}_i \leftarrow \text{pSign}(sk, m, Y_i)
\]

Oracles testify for events

Bob obtains pre-signatures from Alice and requests the oracle for testimony

\[
\sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}_i, y_i)
\]
Oracle-Based Conditional Payments

**Alice**
sends a payment when the oracle testifies for an event

\[
\forall i \in \{1, \ldots, M\} : \\
(Y_i, y_i) \leftarrow \text{RGem}(1^\lambda)
\]

**Oracles**
testify for events

\[
(y_1, \ldots, y_N)
\]

**Bob**
obeats pre-signatures from Alice and requests the oracle for testimony

\[
\sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}_i, y_i)
\]
Oracle-Based Conditional Payments

Alice sends a payment when the oracle testifies for an event

\[ \forall i \in \{1, \ldots, M\} : (Y_i, y_i) \leftarrow \text{RGen}(1^\lambda) \]

\[ \forall i \in \{1, \ldots, M\} : \tilde{\sigma}_i \leftarrow \text{pSign}(sk, m, Y_i) \]

Bob obtains pre-signatures from Alice and requests the oracle for testimony

\[ \tilde{\sigma}_{1 \leq i \leq M} \]

\[ \sigma \leftarrow \text{Adapt}(pk, \tilde{\sigma}_i, y_i) \]

Oracles testify for events

\[ (y_1, \ldots, y_N) \]

[MTFMS'22]
Alice sends a payment when the oracle testifies for an event

∀i ∈ {1, . . . , M} :
(Y_i, y_i) ← RGen(1^λ)

∀i ∈ {1, . . . , M} :
\tilde{σ}_i ← pSign(sk, m, Y_i)

(y_1, . . . , y_N)

Oracles

testify for events

Bob

obtains pre-signatures from Alice and requests the oracle for testimony

σ ← Adapt(pk, \tilde{σ}_i, y_i)

σ ← \tilde{σ}_1 ⊕ \tilde{σ}_2
Overview

- Gaps
- Definitions
- Constructions
- Transparent Reductions
Theoretical Challenges

Can we generically transform signatures into adaptor signatures?

Can we find an adaptor signature scheme in the standard model?
Theoretical Challenges

Can we generically transform signatures into adaptor signatures?

Can we find an adaptor signature scheme in the standard model?
Dichotomic Signatures: Pre-Signing

\[\text{pSign}(sk, m, Y)\]

1: \( r \leftarrow \mathbb{Z}_p; R \leftarrow g^r \)
2: \( h \leftarrow H(pk, R \cdot Y, m) \)
3: \textbf{return} \((R \cdot Y, sk \cdot h + r)\)

- The signature consists of two parts \(\sigma = (\sigma_1, \sigma_2)\)
- The signature uses a homomorphic one-way function
  \(R = \text{OWF}(r); Y = \text{OWF}(y); r, y \in \mathbb{Z}_p\)
- One part can be computed using
  \(\sigma_1 = \Sigma_1(sk, m; \text{OWF}(r) \cdot \text{OWF}(y))\)
- The other part can be computed using
  \(\sigma_2 = \Sigma_2(sk, m; r)\)
Dichotomic Signatures: Pre-Signing

\[ \text{pSign}(sk, m, Y) \]

1: \( r \leftarrow \mathbb{Z}_p; R \leftarrow g^r \)
2: \( h \leftarrow H(pk, R \cdot Y, m) \)
3: \( \text{return } (R \cdot Y, sk \cdot h + r) \)

- The signature consists of two parts
  \[ \sigma = (\sigma_1, \sigma_2) \]

- The signature uses a homomorphic one-way function
  \[ R = \text{OWF}(r); Y = \text{OWF}(y); r, y \in \mathbb{Z}_p \]

- One part can be computed using
  \[ \sigma_1 = \Sigma_1(sk, m; \text{OWF}(r) \cdot \text{OWF}(y)) \]

- The other part can be computed using
  \[ \sigma_2 = \Sigma_2(sk, m; r) \]

• The signature consists of two parts
  \[ \sigma = (\sigma_1, \sigma_2) \]

• The signature uses a homomorphic one-way function
  \[ R = \text{OWF}(r); Y = \text{OWF}(y); r, y \in \mathbb{Z}_p \]

• One part can be computed using
  \[ \sigma_1 = \Sigma_1(sk, m; \text{OWF}(r) \cdot \text{OWF}(y)) \]

• The other part can be computed using
  \[ \sigma_2 = \Sigma_2(sk, m; r) \]
Dichotomic Signatures: Pre-Signing

\[ \text{pSign}(sk, m, Y) = (R \cdot Y, sk \cdot h + r) \]

1: \( r \leftarrow \mathbb{Z}_p; R \leftarrow g^r \)
2: \( h \leftarrow H(pk, R \cdot Y, m) \)
3: return \((R \cdot Y, sk \cdot h + r)\)

- The signature consists of two parts \( \sigma = (\sigma_1, \sigma_2) \)
- The signature uses a homomorphic one-way function
  \( R = \text{OWF}(r); Y = \text{OWF}(y); r, y \in \mathbb{Z}_p \)
- One part can be computed using
  \[ \sigma_1 = \Sigma_1(sk, m; \text{OWF}(r) \cdot \text{OWF}(y)) \]
- The other part can be computed using
  \[ \sigma_2 = \Sigma_2(sk, m; r) \]
Dichotomic Signatures: Pre-Signing

\( \text{pSign}(sk, m, Y) \)

1. \( r \leftarrow Z_p; R \leftarrow g^r \)
2. \( h \leftarrow H(pk, R \cdot Y, m) \)
3. \text{return} \((R \cdot Y, sk \cdot h + r)\)

- The signature consists of two parts
  \( \sigma = (\sigma_1, \sigma_2) \)

- The signature uses a homomorphic one-way function
  \( R = \text{OWF}(r); Y = \text{OWF}(y); r, y \in Z_p \)

- One part can be computed using
  \( \sigma_1 = \Sigma_1(sk, m; \text{OWF}(r) \cdot \text{OWF}(y)) \)

- The other part can be computed using
  \( \sigma_2 = \Sigma_2(sk, m; r) \)
Dichotomic Signatures: Adapt/Extract

Adapt(pk, \tilde{\sigma}, y)

1: parse \tilde{\sigma} as (\tilde{\sigma}_1, \tilde{\sigma}_2)
2: return (\tilde{\sigma}_1, \tilde{\sigma}_2 + y)

- The second part of the signature is homomorphic in the randomness

Extract(Y, \tilde{\sigma}, \sigma)

1: parse \tilde{\sigma} as (\tilde{\sigma}_1, \tilde{\sigma}_2)
2: parse \sigma as (\sigma_1, \sigma_2)
3: return \sigma_2 - \tilde{\sigma}_2

\[ \Sigma_2(sk, m; r) + y = \Sigma_2(sk, m; r + y) \]
Dichotomic Signatures: A Definition

A signature scheme w.r.t. a homomorphic one-way function $\text{OWF}$ is dichotomic; if

- It is decomposable

\[ \sigma = (\sigma_1, \sigma_2) = (\Sigma_1(\text{sk}, m; \text{OWF}(r)), \Sigma_2(\text{sk}, m; r)) \]

- It is homomorphic in the randomness

\[ \Sigma_2(\text{sk}, m; r) + y = \Sigma_2(\text{sk}, m; r + y) \]
Proving Security

We need to simulate pre-signatures to the adversary.

We cannot use the random oracle.

Converting a signature into a pre-signature seems impossible.

We cannot reduce to the strong unforgeability directly.
Proving Security

- We need to simulate pre-signatures to the adversary.
- We cannot use the random oracle.
- Converting a signature into a pre-signature seems impossible.
- We cannot reduce to the strong unforgeability directly.
We need to simulate pre-signatures to the adversary.

We cannot use the random oracle.

Converting a signature into a pre-signature seems impossible.

We cannot reduce to the strong unforgeability directly.
Proving Security

• We need to simulate pre-signatures to the adversary
• We cannot use the random oracle

Converting a signature into a pre-signature seems impossible

• We cannot reduce to the strong unforgeability directly
Transparent Reductions

- **SimKG**: Simulates keys \((\text{simSK}, \text{simPK})\)
- **SimSign**: Simulates signatures using \(\text{simSK}\)
- **Break**: Solve problem instance using valid forgery

\(\mathcal{R}\)

\((m^*, \sigma^*)\)

-\(\text{inst} \rightarrow \text{SimKG} \rightarrow \text{SimSign} \rightarrow \text{Break} \rightarrow \text{sol}\)
Transparent Reductions

- **SimKG:**
  - Simulates keys (simSK, simPK)

- **SimSign:**
  - Simulates signatures using simSK

- **Break:**
  - Solve problem instance using valid forgery
Transparent Reductions

- **SimKG**: Simulates keys \((\text{simSK}, \text{simPK})\)

- **SimSign**: Simulates signatures using \(\text{simSK}\)

- **Break**: Solve problem instance using valid forgery

\(\mathcal{R}\)
Simulating Pre Signatures

SimKG  SimSign

Break

\( (m^*, \sigma^*) \)

\( \mathcal{R} \)

- So far, we **can**:
  - Simulate keys
  - Provide a signature oracle
  - Break the problem instance using a forgery

- So far, we **cannot**:
  - Provide a pre-signature oracle
Simulating Pre Signatures

So far, we can:
- Simulate keys
- Provide a signature oracle
- Break the problem instance using a forgery

So far, we cannot:
- Provide a pre-signature oracle
Simulatable Transparent Reductions

\[ (m^*, \sigma^*) \]

\[ \begin{array}{ccc}
\text{SimKG} & \xrightarrow{\text{inst}} & \text{SimSign} \\
\downarrow & & \downarrow \\
\text{Break} & \xrightarrow{\text{sol}} & \mathcal{A}
\end{array} \]

\[ \mathcal{R} \]
A secure adaptor signature scheme requires the following three checks:

• The signature scheme is **dichotomic**

• There is a **transparent reduction** from the strong unforgeability to an underlying hard problem

• We can simulate a pre-signature oracle (**simulatability**)
Conclusion

Gaps

Definitions

Constructions

Transparent Reductions