Boosting the Security of Blind Signature Schemes

Jonathan Katz
University of Maryland, College Park
jkatz2@gmail.com

Julian Loss*
CISPA Helmholtz Center for Information Security
lossjulian@gmail.com

Michael Rosenberg**
University of Maryland, College Park
micro@cs.umd.edu

*Work done while at the University of Maryland, College Park
**Supported by the National Defense Science and Engineering Graduate (NDSEG) fellowship
Blind Signatures
Digital signatures where the signer learns nothing

\[(sk, pk) \rightarrow (pk, m) \rightarrow \text{output } \sigma\]
Blind Signatures

*Digital signatures where the signer learns nothing*

**Blindness** The signer can't link any msg-sig pair to a particular execution.

\[(\text{sk, pk}) \rightarrow (\text{pk, m}) \rightarrow \text{output } \sigma\]
Blind Signatures

Digital signatures where the signer learns nothing

**Blindness** The signer can't link any msg-sig pair to a particular execution

**Unforgeability** Given access to the Signer, adversary cannot produce a fresh msg-sig pair

\[(sk, pk) \rightarrow (pk, m) \rightarrow \text{output } \sigma\]
Blind Signatures

Digital signatures where the signer learns nothing

**Blindness** The signer can't link any msg-sig pair to a particular execution

**Unforgeability** Given access to the Signer, adversary cannot produce a fresh msg-sig pair

Problem: *Every pair looks fresh to a blind signer*
Blind Signatures
Digital signatures where the signer learns nothing

**Blindness** The signer can't link any msg-sig pair to a particular execution

**Unforgeability** Given access to the Signer, adversary cannot produce a fresh msg-sig pair

*Problem: Every pair looks fresh to a blind signer*

**One-more-unforgeability (OMUF)** Given \( \ell \) msg-sig pairs, cannot produce \( \ell + 1 \) distinct pairs
Blind Signatures
Digital signatures where the signer learns nothing

**Blindness** The signer can't link any msg-sig pair to a particular execution

**Unforgeability** Given access to the Signer, adversary cannot produce a fresh msg-sig pair

*Problem: Every pair looks fresh to a blind signer*

**One-more-unforgeability (OMUF)** Given $\ell$ msg-sig pairs, cannot produce $\ell+1$ distinct pairs

---

Digital signatures where the signer learns nothing

**Blindness** The signer can't link any msg-sig pair to a particular execution

**Unforgeability** Given access to the Signer, adversary cannot produce a fresh msg-sig pair

*Problem: Every pair looks fresh to a blind signer*

**One-more-unforgeability (OMUF)** Given $\ell$ msg-sig pairs, cannot produce $\ell+1$ distinct pairs
Blind Signatures
Digital signatures where the signer learns nothing

Blindness The signer can't link any msg-sig pair to a particular execution.

Unforgeability Given access to the Signer, adversary cannot produce a fresh msg-sig pair.

Problem: Every pair looks fresh to a blind signer.

One-more-unforgeability (OMUF) Given $\ell$ msg-sig pairs, cannot produce $\ell+1$ distinct pairs.

Digital signatures where the signer learns nothing.

Blindness (sk, pk) (pk, m)

Unforgeability

(Ecash [Chaum83]

Bitcoin Tumbling [HBG16]

Ecash [Chaum83]
Overview of Existing Efficient Blind Signature Schemes
Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for log-many signatures [HKL19]
Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for log-many signatures [HKL19]
  - Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]
Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for log-many signatures [HKL19]
  - Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]
- Nonstandard assumptions:
Overview of Existing Efficient Blind Signature Schemes

- Only proven secure for \textit{log}-many signatures [HKL19]
  - Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]
- Nonstandard assumptions:
  - Co-DH-inversion [FHS15]
Overview of Existing Efficient Blind Signature Schemes

• Only proven secure for log-many signatures [HKL19]
  • Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]

• Nonstandard assumptions:
  • Co-DH-inversion [FHS15]
  • One-more-RSA inversion [BNPS01]
Overview of Existing Efficient Blind Signature Schemes

• Only proven secure for \textit{log-many signatures} [HKL19]
  • Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]

• \textbf{Nonstandard assumptions:}
  • Co-DH-inversion [FHS15]
  • One-more-RSA inversion [BNPS01]
  • Algebraic group model [KLX20]
Overview of Existing Efficient Blind Signature Schemes

• Only proven secure for **log-many signatures** [HKL19]
  • Provably insecure for more than $O(\log(\kappa))$ concurrent sessions [BLLOR20]

• **Nonstandard assumptions:**
  • Co-DH-inversion [FHS15]
  • One-more-RSA inversion [BNPS01]
  • Algebraic group model [KLX20]

• **No concurrency** [BL12, JLO97, KLX20]
Goal

Construct a blind signature scheme in the ROM, using standard number-theoretic assumptions, that is provably secure after:

1. Poly-many signatures
2. With arbitrary concurrency
Our Contribution
Our Contribution

• A **generic** transformation for blind signature schemes:
  log-many sigs $\rightarrow$ poly-many sigs
Our Contribution

• A **generic** transformation for blind signature schemes: log-many sigs → poly-many sigs

• Resulting blind signature schemes rely only on **DLOG, RSA**, or **factoring** in the ROM
Our Contribution

• A generic transformation for blind signature schemes:
  log-many sigs → poly-many sigs

• Resulting blind signature schemes rely only on DLOG, RSA,
  or factoring in the ROM

• Arbitrary concurrency
Our Contribution

• A generic transformation for blind signature schemes:
  log-many sigs $\rightarrow$ poly-many sigs

• Resulting blind signature schemes rely only on **DLOG**, **RSA**, or **factoring** in the ROM

• Arbitrary concurrency
Our Contribution

• A **generic** transformation for blind signature schemes: log-many sigs $\rightarrow$ poly-many sigs

• Resulting blind signature schemes rely only on **DLOG**, **RSA**, or **factoring** in the ROM

• Arbitrary concurrency

Caveats

• Communication linear in #sessions
Our Contribution

• A generic transformation for blind signature schemes: log-many sigs → poly-many sigs

• Resulting blind signature schemes rely only on DLOG, RSA, or factoring in the ROM

• Arbitrary concurrency

Caveats

• Communication linear in #sessions

• Can be improved to #dishonest users who were caught
Our Contribution

- A **generic** transformation for blind signature schemes: log-many sigs → poly-many sigs
- Resulting blind signature schemes rely only on **DLOG**, **RSA**, or **factoring** in the ROM
- Arbitrary concurrency

Caveats

- Communication linear in #sessions
- Can be improved to #dishonest users who were caught
- ~5000 bit groups
Beginning: Pointcheval's Transform

\[(sk, pk) \quad (pk, m)\]

- **Init**
- **Blind**
- **Sign**
- **Unblind**
- **Output \( \sigma \)**

Blind Okamoto-Schnorr. \( O(\log(\kappa)) \)-OMUF
Beginning: Pointcheval's Transform

\[(sk, pk) \rightarrow (pk, m)\]

- **Init**
- **Cut and Choose**
- **Blind**
- **Sign**
- **Unblind**
- **output \( \sigma \)**

Blind Okamoto-Schnorr. \( O(\log(\kappa)) \)-OMUF
Beginning: Pointcheval's Transform

Blind Okamoto-Schnorr. $O(\log(\kappa))$-OMUF

Transformed scheme via [Pointcheval98], poly($\kappa$)-OMUF
Beginning: Pointcheval's Transform

(Transformed scheme via [Pointcheval98], poly(κ)-OMUF)

Init → Blind → Choose → Open

if wrong: Teardown else: Sign

output σ

commit, com2

(sk, pk) (pk, m)
User commits to 2 sets of random coins

**Beginning: Pointcheval's Transform**

User commits to 2 sets of random coins

1. **Commit**
   - com₁, com₂

2. **Init**

3. **Blind**

4. **Choose**
   - b

5. **Open**
   - o⁻ᵇ

6. **Teardown** if wrong:
7. **Sign** else:

**Output** σ

Transformed scheme via [Pointcheval98], poly(κ)-OMUF
**Beginning: Pointcheval's Transform**

User commits to 2 sets of random coins

Signer chooses $b$

**Diagram:**

- **Commit**
  - $(sk, pk)$
  - $(pk, m)$

- **Init**
  - $com_1, com_2$

- **Blind**
  - $b$

- **Choose**
  - $b$

- **Open**
  - $o_{-b}$

- **If wrong:**
  - **Teardown**

- **Else:**
  - **Sign**

- **Unblind**
  - **Output $\sigma$**

Transformed scheme via [Pointcheval98], poly($\kappa$)-OMUF

User commits to 2 sets of random coins. Signer chooses $b$. If wrong, Teardown; else, Sign. Output $\sigma$.
Beginning: Pointcheval's Transform

User commits to 2 sets of random coins

Signer chooses $b$

User opens the com not chosen

Transformed scheme via [Pointcheval98], poly($\kappa$)-OMUF
**Beginning: Pointcheval's Transform**

User commits to 2 sets of random coins

Signer chooses $b$

User opens the com not chosen

Signer checks the opening
  - **Success**: sign
  - **Failure**: abort and **refuse to sign** anything in the future

Transformed scheme via [Pointcheval98], poly($\kappa$)-OMUF
Beginning: Pointcheval's Transform

Intuition

Transformed scheme via [Pointcheval98], poly(κ)-OMUF
Beginning: Pointcheval's Transform

Intuition

Proof strategy: show that poly-one-more-forgery $\rightarrow$ log-one-more-forgery in the underlying scheme

Transformed scheme via [Pointcheval98], poly($\kappa$)-OMUF
Beginning: Pointcheval's Transform

Intuition

Proof strategy: show that poly-one-more-forgery $\rightarrow$ log-one-more-forgery in the underlying scheme

Limit calls by **simulating the signer** when user acts honestly

Transformed scheme via [Pointcheval98], poly($\kappa$)-OMUF
Beginning: Pointcheval's Transform

Intuition

Proof strategy: show that poly-one-more-forgery $\rightarrow$ log-one-more-forgery in the underlying scheme

Limit calls by **simulating the signer** when user acts honestly

How do you force the user to not cheat?

Transformed scheme via [Pointcheval98], poly($\kappa$)-OMUF
Beginning: Pointcheval's Transform

Intuition

Proof strategy: show that poly-one-more-forgery → log-one-more-forgery in the underlying scheme

Limit calls by simulating the signer when user acts honestly

How do you force the user to not cheat?

**Cut and choose**: make two commitments, and later open one at random: \( \Pr[\text{cheat}] = 1/2 \)
Beginning: Pointcheval's Transform

The Upshot

\[(sk, pk) \quad \rightarrow \quad (pk, m)\]

1. **Commit**
   - \(\text{com}_1, \text{com}_2\)

2. **Init**

3. **Blind**

4. **Choose**
   - \(b\)

   - if wrong:
     - **Teardown**
   - else:
     - **Sign**

5. **Open**
   - \(o_{-b}\)

6. **Unblind**
   - **output** \(\sigma\)

Transformed scheme via [Pointcheval98], poly(\(\kappa\))-OMUF
Beginning: Pointcheval's Transform

The Upshot

Init

Blind

Choose

Open

if wrong: Teardown
else: Sign

Unblind

output $\sigma$

Transformed scheme via [Pointcheval98], poly($\kappa$)-OMUF
Beginning: Pointcheval's Transform

The Upshot

Since Okamoto-Schnorr is $\log(\kappa)$-OMUF, the **transformed scheme is** $\text{poly}(\kappa)$-OMUF

Transformed scheme via [Pointcheval98], $\text{poly}(\kappa)$-OMUF
Beginning: Pointcheval's Transform

The Upshot

Since Okamoto-Schnorr is log(κ)-OMUF, the **transformed scheme is** poly(κ)-OMUF

Caveats:
Since Okamoto-Schnorr is $\log(\kappa)$-OMUF, the transformed scheme is $\text{poly}(\kappa)$-OMUF.

Caveats:
- Parallel, not arbitrarily concurrent
Since Okamoto-Schnorr is $\log(\kappa)$-OMUF, the *transformed scheme is $\text{poly}(\kappa)$-OMUF*

**Caveats:**
- Parallel, not arbitrarily concurrent
- Teardown on cheat detection

**The Upshot**

Beginning: Pointcheval's Transform

Transformed scheme via [Pointcheval98]. $\text{poly}(\kappa)$-OMUF
Beginning: Pointcheval's Transform

The Upshot

Since Okamoto-Schnorr is $\log(\kappa)$-OMUF, the **transformed scheme is** $\text{poly}(\kappa)$-OMUF

Caveats:
- Parallel, not arbitrarily concurrent
- Teardown on cheat detection
- Not generic

Transformed scheme via [Pointcheval98], $\text{poly}(\kappa)$-OMUF
Our Transform

Construction

Choose $b$

if wrong: Teardown
else: Sign

Open

Unblind output $\sigma$

Commit

Init

$(sk, pk)$

$(pk, m)$

$\text{com}_1, \text{com}_2$

$b$

$o_{-b}$
Our Transform

Construction

Protocol depends on how many sessions have begun

\[(sk, pk)\]

\[(pk, m)\]

\[\text{Choose} \quad b\]

\[o_{\neg b}\]

\[\text{if wrong: Teardown}\]

\[\text{else: Sign}\]

\[\text{Commit}\]

\[\text{Blind}\]

\[\text{Open}\]

\[\text{Unblind}\]

\[\text{output } \sigma\]
Our Transform

Construction
Protocol depends on how many sessions have begun

In session $N$:
Protocol depends on how many sessions have begun

In session $N$:

User commits to $N$ sets of random coins

Our Transform

Construction

Protocol depends on how many sessions have begun

In session $N$:

User commits to $N$ sets of random coins
Protocol depends on how many sessions have begun

**In session** $N$:

User commits to $N$ sets of random coins

**Our Transform**

**Construction**

Protocol depends on how many sessions have begun

**In session** $N$:

User commits to $N$ sets of random coins
Our Transform

Construction
Protocol depends on how many sessions have begun

In session $N$:
- User commits to $N$ sets of random coins
- Signer picks an $I$

![Diagram](image-url)
Protocol depends on how many sessions have begun

In session $N$:
- User commits to $N$ sets of random coins
- Signer picks an $I$
Our Transform

Construction
Protocol depends on how many sessions have begun

In session $N$:
- User commits to $N$ sets of random coins
- Signer picks an $I$
- User opens all but the $I$-th com

$(sk, pk)$
$(pk, m)$

$\text{Commit}$
$\text{Init}$
$\text{Blind}$
$\text{Choose}$
$\text{Open}$

if wrong:
Teardown
else: $\text{Sign}$

Unblind
output $\sigma$
**Our Transform**

**Construction**

Protocol depends on how many sessions have begun

**In session $N$:**

- User commits to $N$ sets of random coins
- Signer picks an $I$
- User opens all but the $I$-th com

```
(\text{sk, pk}) \quad \Rightarrow \quad \text{(pk, m)}
```

- **Commit**: $\text{com}_1, \ldots, \text{com}_N$
- **Init**: $I$
- **Blind**: $\{o_j\}_{j \neq I}$
- **Choose**: $I$
- **Open**: $\text{Unblind}$ output $\sigma$
- **Teardown** if wrong
- **Sign** else
Our Transform

Construction
Protocol depends on how many sessions have begun

In session $N$:

User commits to $N$ sets of random coins
Signer picks an $I$
User opens all but the $I$-th com
Signer checks the opening

Success: sign
Failure: abort

Diagram:

1. (sk, pk)
2. $\text{Commit}$
3. $\text{Init}$
4. $\text{Blind}$
5. $\text{Choose}$
6. $I$
7. $\{o_j\}_{j\neq I}$
8. $\text{Open}$
9. if wrong: $\text{Teardown}$
   else: $\text{Sign}$
10. $\text{Unblind}$
11. output $\sigma$
Our Transform

Protocol depends on how many sessions have begun

In session $N$:

User commits to $N$ sets of random coins

Signer picks an $I$

User opens all but the $I$-th com

Signer checks the opening

**Success:** sign

**Failure:** abort

![Diagram](image.png)
Our Transform
Analysis - OMUF

\[(\text{sk, pk}) \rightarrow (\text{pk, m})\]

- **Commit**
- **Init**
- **Blind**
- **Choose**
- **Open**
- **Unblind**

\[\text{com}_1, \ldots, \text{com}_N\]

\[
\text{Init} \
\rightarrow \
\text{Blind} \
\rightarrow \
\text{Choose} \
\rightarrow \
\text{Open} \
\rightarrow \
\text{Unblind} \
\]

- if wrong: **Abort**
- else: **Sign**

\[\{o_j\}_{j \neq I}\]

**output**: \(\sigma\)
Our Transform
Analysis - OMUF

Recall: Underlying scheme is only secure when \#cheats = \( O(\log(\kappa)) \)
Our Transform
Analysis - OMUF

Recall: Underlying scheme is only secure when \( \#\text{cheats} = O(\log(\kappa)) \)

Previously

\[(sk, pk) \quad \Rightarrow \quad (pk, m)\]

\[
\text{Commit} \\
\text{Init} \quad \text{Blind} \quad \text{Choose} \\
\text{if wrong:} \quad \text{Abort} \quad \text{else:} \quad \text{Sign} \\
\text{if wrong} \quad \{o_j\}_{j \neq I} \\
\text{Unblind output } \sigma
\]
Our Transform
Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = $O(\log(\kappa))$

Previously
Pr[cheat] = 1/2 each session
Our Transform Analysis - OMUF

Recall: Underlying scheme is only secure when \#cheats = O(\log(\kappa))

Previously

Pr[cheat] = 1/2 each session

Without teardown, after \(s\) sessions \(\mathbb{E}[\#\text{cheats}] = s/2\)
Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when \#cheats = \(O(\log(\kappa))\)

Previously

\(\Pr[\text{cheat}] = 1/2\) each session

Without teardown, after \(s\) sessions \(\mathbb{E}[\#\text{cheats}] = s/2\)

For \(s = \text{poly}(\kappa)\), this is not secure. Hence, teardowns
Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when \( \#\text{cheats} = O(\log(\kappa)) \)

Previously

\[ \Pr[\text{cheat}] = 1/2 \text{ each session} \]

Without teardown, after \( s \) sessions \( \mathbb{E}[\#\text{cheats}] = s/2 \)

For \( s = \text{poly}(\kappa) \), this is not secure. Hence, teardowns

Now

\[(sk, pk) \rightarrow (pk, m)\]

\begin{align*}
\text{Commit} & \quad \text{com}_1, \ldots, \text{com}_N \\
\text{Init} & \\
\text{Blind} & \\
\text{Choose} & \quad I \\
\text{Open} & \quad \{o_j\}_{j \neq I} \\
\text{Abort} & \quad \text{if wrong:} \\
\text{Sign} & \quad \text{else:} \\
\text{Unblind} & \quad \text{output } \sigma
\end{align*}
Our Transform

Analysis - OMUF

Recall: Underlying scheme is only secure when \( \#\text{cheats} = O(\log(\kappa)) \)

Previously

- \( \Pr[\text{cheat}] = 1/2 \) each session
- Without teardown, after \( s \) sessions \( \mathbb{E}[\#\text{cheats}] = s/2 \)
- For \( s = \text{poly}(\kappa) \), this is not secure. Hence, teardowns

Now

- \( \Pr[\text{cheat}] = 1/N \) in session \( N \)
Our Transform
Analysis - OMUF

Recall: Underlying scheme is only secure when $\#\text{cheats} = O(\log(\kappa))$

Previously
$Pr[\text{cheat}] = 1/2$ each session
Without teardown, after $s$ sessions $E[\#\text{cheats}] = s/2$
For $s = \text{poly}(\kappa)$, this is not secure. Hence, teardowns

Now
$Pr[\text{cheat}] = 1/N$ in session $N$
Without teardown, after $s$ sessions,
Our Transform Analysis - OMUF

Recall: Underlying scheme is only secure when \( \text{#cheats} = O(\log(\kappa)) \)

Previously
- \( \Pr[\text{cheat}] = 1/2 \) each session
- Without teardown, after \( s \) sessions \( \mathbb{E}[\text{#cheats}] = s/2 \)
- For \( s = \text{poly}(\kappa) \), this is not secure. Hence, teardowns

Now
- \( \Pr[\text{cheat}] = 1/N \) in session \( N \)
- Without teardown, after \( s \) sessions,
  - \( \mathbb{E}[\text{#cheats}] = \sum_{N=2}^{s+1} \frac{1}{N} \leq \ln(s + 1) \)

\[ (sk, pk) \xrightarrow{\text{Commit}} \text{com}_1, \ldots, \text{com}_N \]

\[ \xrightarrow{\text{Init}} \]

\[ \xrightarrow{\text{Blind}} \]

\[ \xrightarrow{\text{Choose}} I \]

\[ \xrightarrow{\text{Open}} \{o_j\}_{j \neq I} \]

if wrong: Abort
else: Sign

\[ \xrightarrow{\text{Unblind}} \text{output } \sigma \]
Our Transform
Analysis - OMUF

Recall: Underlying scheme is only secure when #cheats = O(log(κ))

Previously
Pr[cheat] = 1/2 each session
Without teardown, after s sessions E[#cheats] = s/2
For s = poly(κ), this is not secure. Hence, teardowns

Now
Pr[cheat] = 1/N in session N
Without teardown, after s sessions,
E[#cheats] = \sum_{N=2}^{s+1} \frac{1}{N} \leq \ln(s + 1)
Tail bound: within O(log(κ)) w.h.p

Diagram:

(sk, pk) -> (pk, m)

Commit
com₁, ..., comₙ

Init

Blind

Choose
I

Open
{o_j}_{j \neq I}

if wrong: Abort
else: Sign

Unblind
output σ
Our Transform
Analysis - OMUF

**Recall:** Underlying scheme is only secure when $\#\text{cheats} = O(\log(\kappa))$

**Previously**
- $\Pr[\text{cheat}] = 1/2$ each session
- Without teardown, after $s$ sessions $\mathbb{E}[\#\text{cheats}] = s/2$
- For $s = \text{poly}(\kappa)$, this is not secure. Hence, teardowns

**Now**
- $\Pr[\text{cheat}] = 1/N$ in session $N$
- Without teardown, after $s$ sessions,
  \[ \mathbb{E}[\#\text{cheats}] = \sum_{N=2}^{s+1} \frac{1}{N} \leq \ln(s + 1) \]
  Tail bound: within $O(\log(\kappa))$ w.h.p

Thus, our scheme is $\text{poly}(\kappa)$-OMUF without teardowns
Our Transform
Analysis - Concurrency

\[ (sk, pk) \quad (pk, m) \]

\[ \text{Commit} \]
\[ \text{Init} \]
\[ \text{Blind} \]
\[ \text{Choose} \]
\[ I \]
\[ \{o_j\}_{j \neq I} \]
\[ \text{Open} \]

\[ \text{if wrong: } \text{Abort} \]
\[ \text{else: } \text{Sign} \]

Unblind output \( \sigma \)
Our Transform
Analysis - Concurrency

Previously
Our Transform
Analysis - Concurrency

Previously **parallel sessions**: a session cannot advance faster than its predecessors.

\[(sk, pk) \Rightarrow (pk, m)\]

- **Init**
  - \(com_1, \ldots, com_N\)
  - **Commit**

- **Blind**
  - \(I\)

- **Choose**
  - \(\{o_j\}_{j \neq I}\)
  - **Open**

  - if wrong: **Abort**
  - else: **Sign**

- **Unblind**
  - output \(\sigma\)
Our Transform
Analysis - Concurrency

Previously

Supported **parallel sessions**: a session cannot advance faster than its predecessors

Now

![Diagram of the transform process]

- **Init**
  - \( \text{com}_1, \ldots, \text{com}_N \)
  - \( \text{Commit} \)

- **Blind**
  - \( \text{Choose} I \)

- **Open**
  - \( \{o_j\}_{j \neq I} \)
  - \( \text{Unblind output} \ \sigma \)

- **Sign**
  - \( (sk, pk) \)

- **Abort**
Our Transform
Analysis - Concurrency

Previously
  Supported **parallel sessions**: a session cannot advance faster than its predecessors

Now
  If $N$ is incremented atomically every session, **we get arbitrary concurrency**
Our Transform
Analysis - Concurrency

Previously
- Supported **parallel sessions**: a session cannot advance faster than its predecessors

Now
- If $N$ is incremented atomically every session, **we get arbitrary concurrency**
- Can be improved: only increment $N$ every time a user is caught cheating

\[
\begin{align*}
\text{(sk, pk)} & \quad \text{(pk, m)} \\
\text{Commit} & \quad \text{Choose} \\
\text{Init} & \quad \text{Blind} \\
\text{Open} & \quad \text{Sign} \\
\text{Unblind} & \quad \text{output } \sigma
\end{align*}
\]
Our Transform
Analysis - Genericity

\[\sigma \]

\[
\begin{align*}
\text{Init} & \quad \quad \text{Blind} \\
\text{Choose} & \\
\text{Commit} & \quad \quad \text{Open} \quad \quad \text{Unblind} \\
\text{Sign} & \end{align*}
\]

\[
\begin{align*}
\text{Init} & \quad \quad \text{Blind} \\
\text{Commit} & \\
\text{Open} & \quad \quad \text{Unblind} \quad \quad \text{Output} \sigma
\end{align*}
\]

Choose

\[
\{o_j\}_{j \neq I}
\]

if wrong:

Abort

else:

Sign

\[
\text{com}_1, \ldots, \text{com}_N
\]

\[
\text{(sk, pk)} \Rightarrow (pk, m)
\]
Our Transform
Analysis - Genericity

Previously

Commit

Init

Blind

Choose

Open

if wrong: Abort
else: Sign

Unblind

output $\sigma$

$(sk, pk)$

$(pk, m)$

$\sigma = \prod_{i=1}^{N} \left\{ o_j \right\}_{j \neq I}$
Our Transform
Analysis -Genericity

Previously
Not generic. Only defined for Okamoto-Schnorr
Our Transform
Analysis - Genericity

Previously
Not generic. Only defined for Okamoto-Schnorr

Now
Our Transform
Analysis - Genericity

Previously
Not generic. Only defined for Okamoto-Schnorr

Now
Generic for any algebraic hash function with homomorphic properties

\[(sk, pk) \quad (pk, m)\]

- **Init**
- **Blind**
- **Choose**
  - \(\{o_j\}_{j\neq I}\)
- **Open**
- **Unblind**
  - Output \(\sigma\)
Our Transform
Analysis - Genericity

Previously
Not generic. Only defined for Okamoto-Schnorr

Now
Generic for any algebraic hash function with homomorphic properties
• Schnorr

\[
\begin{align*}
\text{Commit} & : com_1, \ldots, com_N \\
\text{Init} & \\
\text{Blind} & \\
\text{Choose} & : I \\
\{o_j\}_{j \neq I} & \\
\text{Open} & \\
\text{Unblind} & \\
\text{Sign} & \\
\text{output } \sigma
\end{align*}
\]
Our Transform
Analysis - Genericity

Previously
Not generic. Only defined for Okamoto-Schnorr

Now
Generic for any algebraic hash function with homomorphic properties
  • Schnorr
  • Okamoto-Schnorr
Our Transform
Analysis - Genericity

Previously
Not generic. Only defined for Okamoto-Schnorr

Now
Generic for any algebraic hash function with homomorphic properties
  • Schnorr
  • Okamoto-Schnorr
  • Fiat-Shamir
Our Transform
Analysis - Genericity

Previously
Not generic. Only defined for Okamoto-Schnorr

Now
Generic for any algebraic hash function with homomorphic properties
• Schnorr
• Okamoto-Schnorr
• Fiat-Shamir
• Okamoto-Guillou-Quisquater
References


[Chaum83] D. Chaum, * Blind Signatures for Untraceable Payments*


[Pointcheval98] D. Pointcheval, *Strengthened security for blind signatures*