(Remarks on) Security Proofs of Certificateless Signature Schemes

R. Castro and R. Dahab
UNICAMP – University of Campinas, SP

Workshop on Formal Methods in Security
STIC-AmSud
November 2007

Outline

Certificateless Signature Schemes

- Certificateless Public-Key Cryptography [Al-Riyami and Paterson, 2003]
- Main design goal: compromise between ID-Based Cryptography and traditional "PKI-Based" Cryptography:
  - Avoid IBC's key escrow
  - Avoid certificates altogether
- CL-PKC is a form of implicit key certification [Girault, 1991]

Certificateless Signature Schemes

- Certificateless Public-Key Cryptography [Al-Riyami and Paterson, 2003]
- Main design goal: compromise between ID-Based Cryptography and traditional "PKI-Based" Cryptography:
  - Avoid IBC's key escrow
  - Avoid certificates altogether
- CL-PKC is a form of implicit key certification [Girault, 1991]

Certificateless Signature Schemes

- The original motivation was certificateless *encryption*:
  - Kept a few of IBC's features, such as "encryption into the future"
  - Certificateless signatures were presented as a "by-product"
- The original definition by Al-Riyami/Paterson had 7 algorithms;
  - The following definition, with 5 algorithms, is enough [Hu et al., 2006].

Certificateless Signature Schemes

- The original motivation was certificateless *encryption*:
  - Kept a few of IBC's features, such as "encryption into the future"
  - Certificateless signatures were presented as a "by-product"
- The original definition by Al-Riyami/Paterson had 7 algorithms;
  - The following definition, with 5 algorithms, is enough [Hu et al., 2006].

Certificateless Signature Schemes - Definition

- **Setup.** Run by the KGC to initialize the system. Given a security parameter 1^k, returns master keys, public and secret (mpk, msk).
- **PartialKeyGen.** Takes as input (mpk, msk) and the identity $ID \in \{0,1\}^*$, and outputs the partial private key D_{ID}.
- **UserKeysGen.** Takes as input mpk and generates the user's public key P_{ID} and corresponding secret value, x_{ID}.
- **CL-Sign.** Takes as input mpk, ID, (D_{ID}, x_{ID}) and a message M. Outputs a signature σ on M.
- **CL-Verify.** Takes as input mpk, ID, P_{ID}, M and the signature σ, and outputs `ACCEPT` if and only if σ is a valid signature by user U_{ID}, under public key P_{ID}, on M.

Certificateless Signature Schemes - Definition

- **Setup.** Run by the KGC to initialize the system. Given a security parameter 1^k, returns master keys, public and secret (*mpk*, *msk*).
- **PartialKeyGen.** Takes as input (*mpk, msk*) and the identity $I\!D \in \{0,1\}^*$, and outputs the partial private key $D_{I\!D}$.
- **UserKeysGen.** Takes as input *mpk* and generates the user's public key $P_{I\!D}$ and corresponding secret value, $x_{I\!D}$.
- **CL-Sign.** Takes as input *mpk*, $I\!D$, $(D_{I\!D}, x_{I\!D})$ and a message M. Outputs a signature σ on M.
- **CL-Verify.** Takes as input *mpk*, $I\!D$, $P_{I\!D}$, M and the signature σ, and outputs ACCEPT if and only if σ is a valid signature by user $U_{I\!D}$, under public key $P_{I\!D}$, on M.

Certificateless Signature Schemes - Definition

- **Setup.** Run by the KGC to initialize the system. Given a security parameter 1^k, returns master keys, public and secret (*mpk*, *msk*).
- **PartialKeyGen.** Takes as input (*mpk, msk*) and the identity $ID \in \{0,1\}^*$, and outputs the partial private key D_{ID}.
- **UserKeysGen.** Takes as input *mpk* and generates the user's public key P_{ID} and corresponding secret value, x_{ID}.
- **CL-Sign.** Takes as input *mpk*, *ID*, (D_{ID}, x_{ID}) and a message M. Outputs a signature σ on M.
- **CL-Verify.** Takes as input *mpk*, *ID*, P_{ID}, M and the signature σ, and outputs `ACCEPT` if and only if σ is a valid signature by user U_{ID}, under public key P_{ID}, on M.

Certificateless Signature Schemes - Definition

- **Setup.** Run by the KGC to initialize the system. Given a security parameter 1^k, returns master keys, public and secret (*mpk*, *msk*).
- **PartialKeyGen.** Takes as input (*mpk, msk*) and the identity $ID \in \{0,1\}^*$, and outputs the partial private key D_{ID}.
- **UserKeysGen.** Takes as input *mpk* and generates the user's public key P_{ID} and corresponding secret value, x_{ID}.
- **CL-Sign.** Takes as input *mpk*, *ID*, (D_{ID}, x_{ID}) and a message M. Outputs a signature σ on M.
- **CL-Verify.** Takes as input *mpk*, *ID*, P_{ID}, M and the signature σ, and outputs `ACCEPT` if and only if σ is a valid signature by user U_{ID}, under public key P_{ID}, on M.

Certificateless Signature Schemes - Definition

- **Setup.** Run by the KGC to initialize the system. Given a security parameter 1^k, returns master keys, public and secret (*mpk*, *msk*).
- **PartialKeyGen.** Takes as input (*mpk, msk*) and the identity $ID \in \{0,1\}^*$, and outputs the partial private key D_{ID}.
- **UserKeysGen.** Takes as input *mpk* and generates the user's public key P_{ID} and corresponding secret value, x_{ID}.
- **CL-Sign.** Takes as input *mpk*, *ID*, (D_{ID}, x_{ID}) and a message M. Outputs a signature σ on M.
- **CL-Verify.** Takes as input *mpk*, *ID*, P_{ID}, M and the signature σ, and outputs `ACCEPT` if and only if σ is a valid signature by user U_{ID}, under public key P_{ID}, on M.

Security of CLS Schemes

- No explicit certification $\rightarrow$ keys can be replaced.
- KGC is assumed not to replace public keys.
- Must take two types of adversaries into consideration:
  - **Type I.** Arbitrary adversaries that are able to replace public keys;
  - **Type II.** the KGC, who has access to the master secret.
- Formalized through two very similar games.

Security of CLS Schemes

- No explicit certification $\rightarrow$ keys can be replaced.
- KGC is assumed not to replace public keys.
- Must take two types of adversaries into consideration:
  - **Type I.** Arbitrary adversaries that are able to replace public keys;
  - **Type II.** the KGC, who has access to the master secret.
- Formalized through two very similar games.

Security of CLS Schemes

- No explicit certification $\rightarrow$ keys can be replaced.
- KGC is assumed not to replace public keys.
- Must take two types of adversaries into consideration:
  - **Type I.** Arbitrary adversaries that are able to replace public keys;
  - **Type II.** the KGC, who has access to the master secret.
- Formalized through two very similar games.

Security of CLS Schemes

- No explicit certification $\rightarrow$ keys can be replaced.
- KGC is assumed not to replace public keys.
- Must take two types of adversaries into consideration:
  - **Type I.** Arbitrary adversaries that are able to replace public keys;
  - **Type II.** the KGC, who has access to the master secret.
- Formalized through two very similar games.

Security of CLS Schemes

Definition

Basic Game: Let $\mathcal{C}$ be the challenger algorithm and k be a security parameter:

1. $\mathcal{C}$ executes `Setup`(1^k) and obtains (mpk, msk);
2. $\mathcal{C}$ runs $\mathcal{A}$ on 1^k and mpk. During its run, $\mathcal{A}$ has access to the following oracles: `RevealPublicKey`, `RevealPartialKey`, `RevealSecretValue`, `ReplacePublicKey`, `QueryHash`, `Sign`;
3. $\mathcal{A}$ outputs (ID^*, M^*, σ^*).

$\mathcal{A}$ wins the game if the verification procedure of the CLS scheme accepts (ID^*, M^*, σ^*).

Security of CLS Schemes

Additional conditions to win the game:

- **Type I Adversaries.** $\mathcal{A}_I$ wins the game if both conditions below hold:
  - `Sign`(ID^*,M^*) was never queried;
  - `RevealPartialKey`(ID^*) was also never queried.
- **Type II Adversaries.** $\mathcal{A}_{II}$ wins the game if all conditions below hold:
  - `Sign`(ID^*,M^*) was never queried;
  - `RevealSecretValue`(ID^*) was never queried;
  - `ReplacePublicKey`(ID^*, .) was never queried.

Security of CLS Schemes

Additional conditions to win the game:

- **Type I Adversaries.** $\mathcal{A}_I$ wins the game if both conditions below hold:
  - `Sign`(ID^*,M^*) was never queried;
  - `RevealPartialKey`(ID^*) was also never queried.
- **Type II Adversaries.** $\mathcal{A}_{II}$ wins the game if all conditions below hold:
  - `Sign`(ID^*,M^*) was never queried;
  - `RevealSecretValue`(ID^*) was never queried;
  - `ReplacePublicKey`(ID^*, .) was never queried.

Remark #1: *How do adversaries replace public keys?*

- Many schemes make the assumption that when $\mathcal{A}$ replaces a public key, it knows the corresponding secret value:
  - [Goya, 2006], [Huang et al., 2005], [Yap et al., 2006], [Du and Wen, 2007], [Choi et al., 2007].
- Deriving the secret value from the public key is hard.
- Therefore, this assumption implies that the only way to compute public keys is the "naive" way:
  1. Choose a secret value;
  2. compute a valid public key from the secret value, using the prescribed procedure for the scheme.
- This does not allow adversaries to pick a public key of their choice.

Remark #1: *How do adversaries replace public keys?*

- Many schemes make the assumption that when $\mathcal{A}$ replaces a public key, it knows the corresponding secret value:
  - [Goya, 2006], [Huang et al., 2005], [Yap et al., 2006], [Du and Wen, 2007], [Choi et al., 2007].
- Deriving the secret value from the public key is hard.
- Therefore, this assumption implies that the only way to compute public keys is the "naive" way:
  1. Choose a secret value;
  2. compute a valid public key from the secret value, using the prescribed procedure for the scheme.
- This does not allow adversaries to pick a public key of their choice.

Remark #1: *How do adversaries replace public keys?*

- Many schemes make the assumption that when $\mathcal{A}$ replaces a public key, it knows the corresponding secret value:
  - [Goya, 2006], [Huang et al., 2005], [Yap et al., 2006], [Du and Wen, 2007], [Choi et al., 2007].
- Deriving the secret value from the public key is hard.
- Therefore, this assumption implies that the only way to compute public keys is the "naive" way:
  1. Choose a secret value;
  2. compute a valid public key from the secret value, using the prescribed procedure for the scheme.
- This does not allow adversaries to pick a public key of their choice.

Remark #1: *How do adversaries replace public keys?*

- Many schemes make the assumption that when $\mathcal{A}$ replaces a public key, it knows the corresponding secret value:
  - [Goya, 2006], [Huang et al., 2005], [Yap et al., 2006], [Du and Wen, 2007], [Choi et al., 2007].
- Deriving the secret value from the public key is hard.
- Therefore, this assumption implies that the only way to compute public keys is the “naive” way:
  1. Choose a secret value;
  2. compute a valid public key from the secret value, using the prescribed procedure for the scheme.
- This does not allow adversaries to pick a public key of their choice.

Remark #1: *How do adversaries replace public keys?*

To illustrate this issue, we use Goya & Terada's scheme:

- **Setup**
  1. Choose $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.
  2. generators $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ such that $P = \psi(Q)$;
  3. compute $g = e(P, Q)$; choose $s \overset{R}{\leftarrow} \mathbb{Z}_p^*$; compute $Q_{pub} = sQ$;
  4. choose hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ and $H_2 : \{0,1\}^* \times \{0,1\}^* \times \mathbb{G}_T \times \mathbb{G}_T \rightarrow \mathbb{Z}_p^*$.
- **PartialKeyGen.** $D_A = \frac{1}{H_1(ID_A)+s} P$.
- **UserKeysGen.** Pick a random $t_A \in \mathbb{Z}_p^*$ and compute $N_A = g^{t_A}$.
- **CL-Sign.** Pick a random $r \in \mathbb{Z}_p^*$; compute $U = g^r \in \mathbb{G}_T$; compute $h = H_2(M, ID_A, N_A, U) \in \mathbb{Z}_p^*$, and $S = (r + ht_A)D_A \in \mathbb{G}_1$. The signature is $\sigma = (S, h)$.
- **CL-Verify.** Compute $U' = e[S, H_1(ID_A)Q + Q_{pub}](N_A)^{-h}$; accept if and only if

$$h = H_2(M, ID_A, N_A, U').$$

Remark #1: *How do adversaries replace public keys?*

To illustrate this issue, we use Goya & Terada's scheme:

- **Setup**
  1. Choose $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.
  2. generators $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ such that $P = \psi(Q)$;
  3. compute $g = e(P, Q)$; choose $s \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$; compute $Q_{pub} = sQ$;
  4. choose hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$ and $H_2 : \{0,1\}^* \times \{0,1\}^* \times \mathbb{G}_T \times \mathbb{G}_T \to \mathbb{Z}_p^*$.
- **PartialKeyGen.** $D_A = \frac{1}{H_1(ID_A)+s} P$.
- **UserKeysGen.** Pick a random $t_A \in \mathbb{Z}_p^*$ and compute $N_A = g^{t_A}$.
- **CL-Sign.** Pick a random $r \in \mathbb{Z}_p^*$; compute $U = g^r \in \mathbb{G}_T$; compute $h = H_2(M, ID_A, N_A, U) \in \mathbb{Z}_p^*$, and $S = (r + ht_A)D_A \in \mathbb{G}_1$. The signature is $\sigma = (S, h)$.
- **CL-Verify.** Compute $U' = e[S, H_1(ID_A)Q + Q_{pub}](N_A)^{-h}$; accept if and only if

$$h = H_2(M, ID_A, N_A, U').$$

Remark #1: *How do adversaries replace public keys?*

To illustrate this issue, we use Goya & Terada's scheme:

- **Setup**
  1. Choose $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.
  2. generators $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ such that $P = \psi(Q)$;
  3. compute $g = e(P, Q)$; choose $s \overset{R}{\leftarrow} \mathbb{Z}_p^*$; compute $Q_{pub} = sQ$;
  4. choose hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$ and $H_2 : \{0,1\}^* \times \{0,1\}^* \times \mathbb{G}_T \times \mathbb{G}_T \to \mathbb{Z}_p^*$.
- **PartialKeyGen.** $D_A = \frac{1}{H_1(ID_A)+s}P$.
- **UserKeysGen.** Pick a random $t_A \in \mathbb{Z}_p^*$ and compute $N_A = g^{t_A}$.
- **CL-Sign.** Pick a random $r \in \mathbb{Z}_p^*$; compute $U = g^r \in \mathbb{G}_T$; compute $h = H_2(M, ID_A, N_A, U) \in \mathbb{Z}_p^*$, and $S = (r + ht_A)D_A \in \mathbb{G}_1$. The signature is $\sigma = (S, h)$.
- **CL-Verify.** Compute $U' = e[S, H_1(ID_A)Q + Q_{pub}](N_A)^{-h}$; accept if and only if

$$h = H_2(M, ID_A, N_A, U').$$

Remark #1: *How do adversaries replace public keys?*

To illustrate this issue, we use Goya & Terada's scheme:

- **Setup**
  1. Choose $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.
  2. generators $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ such that $P = \psi(Q)$;
  3. compute $g = e(P, Q)$; choose $s \overset{R}{\leftarrow} \mathbb{Z}_p^*$; compute $Q_{pub} = sQ$;
  4. choose hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$ and $H_2 : \{0,1\}^* \times \{0,1\}^* \times \mathbb{G}_T \times \mathbb{G}_T \to \mathbb{Z}_p^*$.
- **PartialKeyGen.** $D_A = \frac{1}{H_1(ID_A)+s} P$.
- **UserKeysGen.** Pick a random $t_A \in \mathbb{Z}_p^*$ and compute $N_A = g^{t_A}$.
- **CL-Sign.** Pick a random $r \in \mathbb{Z}_p^*$; compute $U = g^r \in \mathbb{G}_T$; compute $h = H_2(M, ID_A, N_A, U) \in \mathbb{Z}_p^*$, and $S = (r + ht_A)D_A \in \mathbb{G}_1$. The signature is $\sigma = (S, h)$.
- **CL-Verify.** Compute $U' = e[S, H_1(ID_A)Q + Q_{pub}](N_A)^{-h}$; accept if and only if

$$h = H_2(M, ID_A, N_A, U').$$

Remark #1: *How do adversaries replace public keys?*

To illustrate this issue, we use Goya & Terada's scheme:

- **Setup**
  1. Choose $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.
  2. generators $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ such that $P = \psi(Q)$;
  3. compute $g = e(P, Q)$; choose $s \overset{R}{\leftarrow} \mathbb{Z}_p^*$; compute $Q_{pub} = sQ$;
  4. choose hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$ and $H_2 : \{0,1\}^* \times \{0,1\}^* \times \mathbb{G}_T \times \mathbb{G}_T \to \mathbb{Z}_p^*$.
- **PartialKeyGen.** $D_A = \frac{1}{H_1(ID_A)+s} P$.
- **UserKeysGen.** Pick a random $t_A \in \mathbb{Z}_p^*$ and compute $N_A = g^{t_A}$.
- **CL-Sign.** Pick a random $r \in \mathbb{Z}_p^*$; compute $U = g^r \in \mathbb{G}_T$; compute $h = H_2(M, ID_A, N_A, U) \in \mathbb{Z}_p^*$, and $S = (r + ht_A)D_A \in \mathbb{G}_1$. The signature is $\sigma = (S, h)$.
- **CL-Verify.** Compute $U' = e[S, H_1(ID_A)Q + Q_{pub}](N_A)^{-h}$; accept if and only if

$$h = H_2(M, ID_A, N_A, U').$$

Remark #1: *How do adversaries replace public keys?*

- Correctness of the scheme:

$$
\begin{aligned}
U' &= e[S, h_1 Q + Q_{pub}](N_A)^{-h} \\
&= e[(r + ht_A)D_A, h_1 Q + Q_{pub}](N_A)^{-h} \\
&= e[(r + ht_A)(h_1 + s)^{-1} P, (h_1 + s)Q] g^{-t_A h} \\
&= e[P, Q]^{r + ht_A} g^{-t_A h} \\
&= g^r g^{ht_A} g^{-t_A h} = g^r = U
\end{aligned}
$$

- Based on Barreto et al.'s IBS [Barreto et al., 2005]
- Very efficient: only one pairing for verification
- Thought to be provably secure:
  - Actually insecure as the following attack shows

Remark #1: *How do adversaries replace public keys?*

- Correctness of the scheme:

$$\begin{aligned} U' &= e[S, h_1 Q + Q_{pub}](N_A)^{-h} \\ &= e[(r + ht_A)D_A, h_1 Q + Q_{pub}](N_A)^{-h} \\ &= e[(r + ht_A)(h_1 + s)^{-1} P, (h_1 + s)Q]g^{-t_A h} \\ &= e[P, Q]^{r+ht_A} g^{-t_A h} \\ &= g^r g^{ht_A} g^{-t_A h} = g^r = U \end{aligned}$$

- Based on Barreto et al.'s IBS [Barreto et al., 2005]
- Very efficient: only one pairing for verification
- Thought to be provably secure:
  - Actually insecure as the following attack shows

Remark #1: *How do adversaries replace public keys?*

- Correctness of the scheme:

$$\begin{aligned} U' &= e[S, h_1 Q + Q_{pub}](N_A)^{-h} \\ &= e[(r + ht_A)D_A, h_1 Q + Q_{pub}](N_A)^{-h} \\ &= e[(r + ht_A)(h_1 + s)^{-1} P, (h_1 + s)Q]g^{-t_A h} \\ &= e[P, Q]^{r + ht_A} g^{-t_A h} \\ &= g^r g^{ht_A} g^{-t_A h} = g^r = U \end{aligned}$$

- Based on Barreto et al.'s IBS [Barreto et al., 2005]
- Very efficient: only one pairing for verification
- Thought to be provably secure:
  - Actually insecure as the following attack shows

Remark #1: *How do adversaries replace public keys?*

- Correctness of the scheme:

$$
\begin{aligned}
U' &= e[S, h_1 Q + Q_{pub}](N_A)^{-h} \\
&= e[(r + ht_A)D_A, h_1 Q + Q_{pub}](N_A)^{-h} \\
&= e[(r + ht_A)(h_1 + s)^{-1} P, (h_1 + s)Q] g^{-t_A h} \\
&= e[P, Q]^{r + ht_A} g^{-t_A h} \\
&= g^r g^{ht_A} g^{-t_A h} = g^r = U
\end{aligned}
$$

- Based on Barreto et al.'s IBS [Barreto et al., 2005]
- Very efficient: only one pairing for verification
- Thought to be provably secure:
  - Actually insecure as the following attack shows

Remark #1: *How do adversaries replace public keys?*

Forging Goya/Terada Signatures.

- Given the target identity ID_A:
  1. Choose a random $t_A \xleftarrow{r} \mathbb{Z}_q^*$ and compute
     $$N_A = (e(P, Q_{pub})g^{H_1(ID_A)})^{t_A} = (g^s g^{H_1(ID_A)})^{t_A} = g^{t_A(s+H_1(ID_A))};$$
  2. Replace ID_A's public key with N_A
- Now, to a sign message M:
  1. Choose $r \xleftarrow{r} \mathbb{Z}_q^*$; compute $U = N_A^r$; let $h = H(M, ID_A, N_A, U)$
  2. Compute $S = (r+h)t_A P$; output the forgery $\gamma = (S, h)$
- Correctness:

$$\begin{aligned}
U' &= e[S, h_1 Q + Q_{pub}](N_A)^{-h} \\
&= e[(r+h)t_A P, (h_1 + s)Q](g^{t_A(s+h_1)})^{-h} \\
&= e[P, Q]^{(r+h)(h_1+s)t_A}(g^{t_A(s+h_1)})^{-h} \\
&= g^{r(h_1+s)t_A} = N_A^r = U.
\end{aligned}$$

- $\mathcal{A}$ does not know x, $N_A = g^x$

Remark #1: *How do adversaries replace public keys?*

Forging Goya/Terada Signatures.

- Given the target identity ID_A:
  1. Choose a random $t_A \xleftarrow{r} \mathbb{Z}_q^*$ and compute
  $$N_A = (e(P, Q_{pub})g^{H_1(ID_A)})^{t_A} = (g^s g^{H_1(ID_A)})^{t_A} = g^{t_A(s+H_1(ID_A))};$$
  2. Replace ID_A's public key with N_A
- Now, to a sign message M:
  1. Choose $r \xleftarrow{r} \mathbb{Z}_q^*$; compute $U = N_A^r$; let $h = H(M, ID_A, N_A, U)$
  2. Compute $S = (r + h)t_A P$; output the forgery $\gamma = (S, h)$
- Correctness:

$$\begin{aligned} U' &= e[S, h_1 Q + Q_{pub}](N_A)^{-h} \\ &= e[(r+h)t_A P, (h_1 + s)Q](g^{t_A(s+h_1)})^{-h} \\ &= e[P, Q]^{(r+h)(h_1+s)t_A}(g^{t_A(s+h_1)})^{-h} \\ &= g^{r(h_1+s)t_A} = N_A^r = U. \end{aligned}$$

- $\mathcal{A}$ does not know x, $N_A = g^x$

Remark #1: *How do adversaries replace public keys?*

Forging Goya/Terada Signatures.

- Given the target identity ID_A:
  1. Choose a random $t_A \xleftarrow{r} \mathbb{Z}_q^*$ and compute
  $$N_A = (e(P, Q_{pub})g^{H_1(ID_A)})^{t_A} = (g^s g^{H_1(ID_A)})^{t_A} = g^{t_A(s+H_1(ID_A))};$$
  2. Replace ID_A's public key with N_A
- Now, to a sign message M:
  1. Choose $r \xleftarrow{r} \mathbb{Z}_q^*$; compute $U = N_A^r$; let $h = H(M, ID_A, N_A, U)$
  2. Compute $S = (r+h)t_A P$; output the forgery $\gamma = (S, h)$
- Correctness:

$$\begin{aligned} U' &= e[S, h_1 Q + Q_{pub}](N_A)^{-h} \\ &= e[(r+h)t_A P, (h_1 + s)Q](g^{t_A(s+h_1)})^{-h} \\ &= e[P, Q]^{(r+h)(h_1+s)t_A}(g^{t_A(s+h_1)})^{-h} \\ &= g^{r(h_1+s)t_A} = N_A^r = U. \end{aligned}$$

- $\mathcal{A}$ does not know x, $N_A = g^x$

Remark #1: *How do adversaries replace public keys?*

Forging Goya/Terada Signatures.

- Given the target identity $I\!D_A$:
  1. Choose a random $t_A \xleftarrow{r} \mathbb{Z}_q^*$ and compute
     $$N_A = (e(P, Q_{pub})g^{H_1(I\!D_A)})^{t_A} = (g^s g^{H_1(I\!D_A)})^{t_A} = g^{t_A(s+H_1(I\!D_A))};$$
  2. Replace $I\!D_A$'s public key with N_A
- Now, to a sign message M:
  1. Choose $r \xleftarrow{r} \mathbb{Z}_q^*$; compute $U = N_A^r$; let $h = H(M, I\!D_A, N_A, U)$
  2. Compute $S = (r + h)t_A P$; output the forgery $\gamma = (S, h)$
- Correctness:

$$\begin{aligned} U' &= e[S, h_1 Q + Q_{pub}](N_A)^{-h} \\ &= e[(r+h)t_A P, (h_1 + s)Q](g^{t_A(s+h_1)})^{-h} \\ &= e[P, Q]^{(r+h)(h_1+s)t_A}(g^{t_A(s+h_1)})^{-h} \\ &= g^{r(h_1+s)t_A} = N_A^r = U. \end{aligned}$$

- $\mathcal{A}$ does not know x, $N_A = g^x$

Remark #1: *How do adversaries replace public keys?*

Forging Goya/Terada Signatures.

- Given the target identity ID_A:
  1. Choose a random $t_A \xleftarrow{r} \mathbb{Z}_q^*$ and compute
  $$N_A = (e(P, Q_{pub})g^{H_1(ID_A)})^{t_A} = (g^s g^{H_1(ID_A)})^{t_A} = g^{t_A(s+H_1(ID_A))};$$
  2. Replace ID_A's public key with N_A
- Now, to a sign message M:
  1. Choose $r \xleftarrow{r} \mathbb{Z}_q^*$; compute $U = N_A^r$; let $h = H(M, ID_A, N_A, U)$
  2. Compute $S = (r+h)t_A P$; output the forgery $\gamma = (S, h)$
- Correctness:

$$\begin{aligned} U' &= e[S, h_1 Q + Q_{pub}](N_A)^{-h} \\ &= e[(r+h)t_A P, (h_1+s)Q](g^{t_A(s+h_1)})^{-h} \\ &= e[P, Q]^{(r+h)(h_1+s)t_A}(g^{t_A(s+h_1)})^{-h} \\ &= g^{r(h_1+s)t_A} = N_A^r = U. \end{aligned}$$

- $\mathcal{A}$ does not know x, $N_A = g^x$

Remark #1: *How do adversaries replace public keys?*

Forging Goya/Terada Signatures.

- Given the target identity ID_A:
  1. Choose a random $t_A \xleftarrow{r} \mathbb{Z}_q^*$ and compute
     $$N_A = (e(P, Q_{pub})g^{H_1(ID_A)})^{t_A} = (g^s g^{H_1(ID_A)})^{t_A} = g^{t_A(s+H_1(ID_A))};$$
  2. Replace ID_A's public key with N_A
- Now, to a sign message M:
  1. Choose $r \xleftarrow{r} \mathbb{Z}_q^*$; compute $U = N_A^r$; let $h = H(M, ID_A, N_A, U)$
  2. Compute $S = (r + h)t_A P$; output the forgery $\gamma = (S, h)$
- Correctness:

$$\begin{aligned} U' &= e[S, h_1 Q + Q_{pub}](N_A)^{-h} \\ &= e[(r+h)t_A P, (h_1 + s)Q](g^{t_A(s+h_1)})^{-h} \\ &= e[P, Q]^{(r+h)(h_1+s)t_A}(g^{t_A(s+h_1)})^{-h} \\ &= g^{r(h_1+s)t_A} = N_A^r = U. \end{aligned}$$

- $\mathcal{A}$ does not know x, $N_A = g^x$

Remark #1: *How do adversaries replace public keys?*

This puts in doubt the security of two recent proposals:

- [Du and Wen, 2007] and [Choi et al., 2007]
- Both are related and very efficient
- We weren't able to find attacks on any of these schemes
- Their situation is uncertain

Remark #1: *How do adversaries replace public keys?*

This puts in doubt the security of two recent proposals:

- [Du and Wen, 2007] and [Choi et al., 2007]
- Both are related and very efficient
- We weren't able to find attacks on any of these schemes
- Their situation is uncertain

Remark #1: *How do adversaries replace public keys?*

This puts in doubt the security of two recent proposals:

- [Du and Wen, 2007] and [Choi et al., 2007]
- Both are related and very efficient
- We weren't able to find attacks on any of these schemes
- Their situation is uncertain

Remark #1: *How do adversaries replace public keys?*

This puts in doubt the security of two recent proposals:

- [Du and Wen, 2007] and [Choi et al., 2007]
- Both are related and very efficient
- We weren't able to find attacks on any of these schemes
- Their situation is uncertain

Remark #2: *The Oracle Replay Technique and CLS*

- The Oracle Replay Technique was proposed in [Pointcheval and Stern, 2000] to prove the security of *generic* signature schemes:
  - Schnorr, variants of ElGamal, schemes from Fiat-Shamir heuristics.
- A signature scheme $\mathcal{S}$ is said to be generic if, given the input message m, it produces triples (r, h, σ), where:
  - r takes its value randomly within a large set;
  - h is the hash value of m, r;
  - σ depends only on r, m and h.
- **Forking Lemma.** If an adversary $\mathcal{A}$ can forge signatures then it's possible to replay a successful execution and (with non-negligible probability) obtain a pair of related forgeries (r, h, σ) and (r, h', σ') where $h' \neq h$.

Remark #2: *The Oracle Replay Technique and CLS*

- The Oracle Replay Technique was proposed in [Pointcheval and Stern, 2000] to prove the security of *generic* signature schemes:
  - Schnorr, variants of ElGamal, schemes from Fiat-Shamir heuristics.
- A signature scheme $\mathcal{S}$ is said to be generic if, given the input message m, it produces triples (r, h, σ), where:
  - r takes its value randomly within a large set;
  - h is the hash value of m, r;
  - σ depends only on r, m and h.
- **Forking Lemma.** If an adversary $\mathcal{A}$ can forge signatures then it's possible to replay a successful execution and (with non-negligible probability) obtain a pair of related forgeries (r, h, σ) and (r, h', σ') where $h' \neq h$.

Remark #2: *The Oracle Replay Technique and CLS*

- The Oracle Replay Technique was proposed in [Pointcheval and Stern, 2000] to prove the security of *generic* signature schemes:
  - Schnorr, variants of ElGamal, schemes from Fiat-Shamir heuristics.
- A signature scheme $\mathcal{S}$ is said to be generic if, given the input message m, it produces triples (r, h, σ), where:
  - r takes its value randomly within a large set;
  - h is the hash value of m, r;
  - σ depends only on r, m and h.
- **Forking Lemma.** If an adversary $\mathcal{A}$ can forge signatures then it's possible to replay a successful execution and (with non-negligible probability) obtain a pair of related forgeries (r, h, σ) and (r, h', σ') where $h' \neq h$.

Remark #2: *The Oracle Replay Technique and CLS*

- A pair of signatures (r, h, σ) and (r, h', σ') where $h' \neq h$ is usually enough to compute private keys (in generic schemes).
  - A Schnorr signature is $\sigma = k + hx \mod q$, and $r = g^k$.
  - If we also know $\sigma' = k + h'x \mod q$, then:

$$\begin{aligned}(\sigma' - \sigma)(h' - h)^{-1} &= (k + h'x - k - hx)((h' - h)^{-1}) \\ &= x(h' - h)(h' - h)^{-1} \\ &= x \mod q,\end{aligned}$$

  - Thus revealing the secret key.

Remark #2: *The Oracle Replay Technique and CLS*

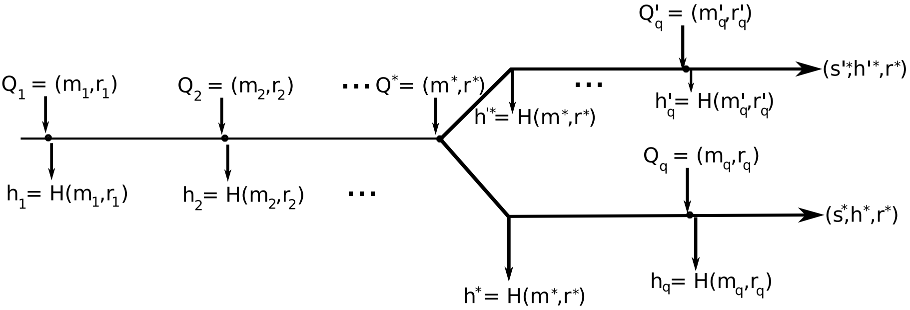
- A pair of signatures (r, h, σ) and (r, h', σ') where $h' \neq h$ is usually enough to compute private keys (in generic schemes).
  - A Schnorr signature is $\sigma = k + hx \mod q$, and $r = g^k$.
  - If we also know $\sigma' = k + h'x \mod q$, then:

$$\begin{aligned}(\sigma' - \sigma)(h' - h)^{-1} &= (k + h'x - k - hx)((h' - h)^{-1}) \\ &= x(h' - h)(h' - h)^{-1} \\ &= x \mod q,\end{aligned}$$

  - Thus revealing the secret key.

Remark #2: *The Oracle Replay Technique and CLS*

The Oracle Replay Technique can be illustrated as follows:



- (s^*, h^*, r^*) is the first forgery, $(s^{*'}, h^{*'}, r^*)$ is the second.

Remark #2: *The Oracle Replay Technique and CLS*

- What guarantees that r will be the same in both forgeries?
  - It's in the $\mathcal{Q}^*$ hash query so it must be chosen before the execution "forks"
- But in CLS keys can be replaced
- So, in a CLS, what guarantees that the public key will be the same in both executions?
  - Nothing, *unless it is also in the hash query*
- Thus, we define the notion of *CL-Generic* Signature scheme:
- A CLS scheme $\mathcal{S}$ is said to be CL-Generic if, given the input message M, it produces triples (r, h, σ), where:
  - r takes its value randomly within a large set;
  - h is the hash value of m, r and the public key PK_{ID};
  - σ depends only on r, m and h.

Remark #2: *The Oracle Replay Technique and CLS*

- What guarantees that r will be the same in both forgeries?
  - It's in the $\mathcal{Q}^*$ hash query so it must be chosen before the execution "forks"
- But in CLS keys can be replaced
- So, in a CLS, what guarantees that the public key will be the same in both executions?
  - Nothing, *unless it is also in the hash query*
- Thus, we define the notion of *CL-Generic* Signature scheme:
- A CLS scheme $\mathcal{S}$ is said to be CL-Generic if, given the input message M, it produces triples (r, h, σ), where:
  - r takes its value randomly within a large set;
  - h is the hash value of m, r and the public key PK_{ID};
  - σ depends only on r, m and h.

Remark #2: *The Oracle Replay Technique and CLS*

- What guarantees that r will be the same in both forgeries?
  - It's in the $\mathcal{Q}^*$ hash query so it must be chosen before the execution "forks"
- But in CLS keys can be replaced
- So, in a CLS, what guarantees that the public key will be the same in both executions?
  - Nothing, *unless it is also in the hash query*
- Thus, we define the notion of *CL-Generic* Signature scheme:
- A CLS scheme $\mathcal{S}$ is said to be CL-Generic if, given the input message M, it produces triples (r, h, σ), where:
  - r takes its value randomly within a large set;
  - h is the hash value of m, r and the public key PK_{ID};
  - σ depends only on r, m and h.

Remark #2: *The Oracle Replay Technique and CLS*

- What guarantees that r will be the same in both forgeries?
  - It's in the $\mathcal{Q}^*$ hash query so it must be chosen before the execution "forks"
- But in CLS keys can be replaced
- So, in a CLS, what guarantees that the public key will be the same in both executions?
  - Nothing, *unless it is also in the hash query*
- Thus, we define the notion of *CL-Generic* Signature scheme:
- A CLS scheme $\mathcal{S}$ is said to be CL-Generic if, given the input message M, it produces triples (r, h, σ), where:
  - r takes its value randomly within a large set;
  - h is the hash value of m, r and the public key PK_{ID};
  - σ depends only on r, m and h.

Remark #2: *The Oracle Replay Technique and CLS*

- What guarantees that r will be the same in both forgeries?
  - It's in the $\mathcal{Q}^*$ hash query so it must be chosen before the execution "forks"
- But in CLS keys can be replaced
- So, in a CLS, what guarantees that the public key will be the same in both executions?
  - Nothing, *unless it is also in the hash query*
- Thus, we define the notion of *CL-Generic* Signature scheme:
- A CLS scheme $\mathcal{S}$ is said to be CL-Generic if, given the input message M, it produces triples (r, h, σ), where:
  - r takes its value randomly within a large set;
  - h is the hash value of m, r and the public key PK_{ID};
  - σ depends only on r, m and h.

Remark #2: *The Oracle Replay Technique and CLS*

- Al-Riyami & Paterson's original CLS: no security proof
  - Later found insecure by Huang et al.
- Signing procedure: $r \xleftarrow{r} \mathbb{Z}_p^*$; $u = e(rP, P)$; $S = H_2(M, u)t_A D_A + rP$.
  - *Insecure.*
- Change to: $r \xleftarrow{r} \mathbb{Z}_p^*$; $u = e(rP, P)$; $S = H_2(M, u, \boxed{PK_{ID}})t_A D_A + rP$.
  - *Secure.* Proof by the Oracle Replay Technique.
  - As efficient as the original version.
- Same can be done for Li, Chen & Sun's version [Li et al., 2005]

Remark #2: *The Oracle Replay Technique and CLS*

- Al-Riyami & Paterson's original CLS: no security proof
  - Later found insecure by Huang et al.
- Signing procedure: $r \xleftarrow{r} \mathbb{Z}_p^*; u = e(rP, P); S = H_2(M, u)t_A D_A + rP$.
  - *Insecure.*
- Change to: $r \xleftarrow{r} \mathbb{Z}_p^*; u = e(rP, P); S = H_2(M, u, \boxed{PK_{ID}})t_A D_A + rP$.
  - *Secure.* Proof by the Oracle Replay Technique.
  - As efficient as the original version.
- Same can be done for Li, Chen & Sun's version [Li et al., 2005]

Remark #2: *The Oracle Replay Technique and CLS*

- Al-Riyami & Paterson's original CLS: no security proof
  - Later found insecure by Huang et al.
- Signing procedure: $r \stackrel{r}{\leftarrow} \mathbb{Z}_p^*$; $u = e(rP, P)$; $S = H_2(M, u)t_A D_A + rP$.
  - *Insecure.*
- Change to: $r \stackrel{r}{\leftarrow} \mathbb{Z}_p^*$; $u = e(rP, P)$; $S = H_2(M, u, \boxed{PK_{ID}})t_A D_A + rP$.
  - *Secure.* Proof by the Oracle Replay Technique.
  - As efficient as the original version.
- Same can be done for Li, Chen & Sun's version [Li et al., 2005]

Remark #2: *The Oracle Replay Technique and CLS*

- Al-Riyami & Paterson's original CLS: no security proof
  - Later found insecure by Huang et al.
- Signing procedure: $r \xleftarrow{r} \mathbb{Z}_p^*; u = e(rP, P); S = H_2(M, u)t_A D_A + rP$.
  - *Insecure.*
- Change to: $r \xleftarrow{r} \mathbb{Z}_p^*; u = e(rP, P); S = H_2(M, u, \boxed{PK_{ID}})t_A D_A + rP$.
  - *Secure.* Proof by the Oracle Replay Technique.
  - As efficient as the original version.
- Same can be done for Li, Chen & Sun's version [Li et al., 2005]

Summary of CLS Schemes

Scheme	Sign	Verify	Status
[Al-Riyami and Paterson, 2003]	1	4	*Broken*
[Huang et al., 2005]	2	5	**OK**
[Castro and Dahab, 2007]	1	4	**OK**
[Li et al., 2005]	0	4	**OK**
[Gorantla and Saxena, 2005]	0	2	*Broken*
[Yap et al., 2006]	0	2	*Broken*
[Zhang et al., 2006]	0	4	**OK**
[Goya, 2006]	0	1	*Broken*
[Liu et al., 2006]	0	6	**OK**
[Choi et al., 2007]	0	1	*Unknown*
[Choi et al., 2007]	0	2	*Unknown*
[Du and Wen, 2007]	0	1	*Unknown*
Castro & Dahab [soon on ePrint]	0	3	**OK**

The cost of signing and verifying is expressed in number of pairings.

Concluding Remarks

We discussed two common pitfalls in the security proofs of CLS schemes:

1. **Knowledge of secret values related to replaced public-keys:**
   - Assumption used in the proofs of too many schemes
   - Leads to attack on Goya/Terada
   - Puts security of [Du and Wen, 2007] and [Choi et al., 2007] in doubt
2. **The use of the Replay Technique:**
   - Efficient, provably secure, correction of Al-Riyami/Paterson
   - Security proofs of a previously unproven scheme [Li et al., 2005]
   - General guideline for constructing CLS schemes

Concluding Remarks

We discussed two common pitfalls in the security proofs of CLS schemes:

1. **Knowledge of secret values related to replaced public-keys:**
   - Assumption used in the proofs of too many schemes
   - Leads to attack on Goya/Terada
   - Puts security of [Du and Wen, 2007] and [Choi et al., 2007] in doubt
2. **The use of the Replay Technique:**
   - Efficient, provably secure, correction of Al-Riyami/Paterson
   - Security proofs of a previously unproven scheme [Li et al., 2005]
   - General guideline for constructing CLS schemes

Bibliography

Al-Riyami, S. S. and Paterson, K. G. (2003).
Certificateless public key cryptography.
In Laih, C.-S., editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 452–473. Springer.

Barreto, P. S. L. M., Libert, B., McCullagh, N., and Quisquater, J.-J. (2005).
Efficient and provably-secure identity-based signatures and signcryption from bilinear maps.
In Roy, B. K., editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 515–532. Springer.

Castro, R. and Dahab, R. (2007).
Two notes on the security of certificateless signatures.
In Susilo, W., Liu, J. K., and Mu, Y., editors, *The 1st International Conference on Provable Security (ProvSec) 2007*, volume 4784 of *Lecture Notes in Computer Science*. Springer.

Choi, K. Y., Park, J. H., Hwang, J. Y., and Lee, D. H. (2007).
Efficient certificateless signature schemes.

In Katz, J. and Yung, M., editors, *ACNS*, volume 4521 of *Lecture Notes in Computer Science*, pages 443–458. Springer.

Du, H. and Wen, Q. (2007).
Efficient and provably-secure certificateless short signature scheme from bilinear pairings.
Cryptology ePrint Archive, Report 2007/250.
http://eprint.iacr.org/.

Girault, M. (1991).
Self-certified public keys.
In Davies, D. W., editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 490–497. Springer.

Gorantla, M. C. and Saxena, A. (2005).
An efficient certificateless signature scheme.
In Hao, Y., Liu, J., Wang, Y., ming Cheung, Y., Yin, H., Jiao, L., Ma, J., and Jiao, Y.-C., editors, *CIS (2)*, volume 3802 of *Lecture Notes in Computer Science*, pages 110–116. Springer.

Goya, D. H. (2006).

Proposta de esquemas de criptografia e de assinatura sob modelo de criptografia de cha pública sem certificado.
Master's thesis, USP.

Hu, B. C., Wong, D. S., Zhang, Z., and Deng, X. (2006).
Key replacement attack against a generic construction of certificateless signature.
In Batten, L. M. and Safavi-Naini, R., editors, *ACISP*, volume 4058 of *Lecture Notes in Computer Science*, pages 235–246. Springer.

Huang, X., Susilo, W., Mu, Y., and Zhang, F. (2005).
On the security of certificateless signature schemes from asiacrypt 2003.
In Desmedt, Y., Wang, H., Mu, Y., and Li, Y., editors, *CANS*, volume 3810 of *Lecture Notes in Computer Science*, pages 13–25. Springer.

Li, X., Chen, K., and Sun, L. (2005).
Certificateless signature and proxy signature schemes from bilinear pairings.

Lithuanian Mathematical Journal, 45(1):76–83.

Liu, J. K., Au, M. H., and Susilo, W. (2006).
Self-generated-certificate public key cryptography and certificateless signature / encryption scheme in the standard model.
Cryptology ePrint Archive, Report 2006/373.
http://eprint.iacr.org/.

Pointcheval, D. and Stern, J. (2000).
Security arguments for digital signatures and blind signatures.
Journal of Cryptology: the journal of the International Association for Cryptologic Research, 13(3):361–396.

Yap, W.-S., Heng, S.-H., and Goi, B.-M. (2006).
An efficient certificateless signature scheme.
In Zhou, X., Sokolsky, O., Yan, L., Jung, E.-S., Shao, Z., Mu, Y., Lee, D. C., Kim, D., Jeong, Y.-S., and Xu, C.-Z., editors, *EUC Workshops*, volume 4097 of *Lecture Notes in Computer Science*, pages 322–331. Springer.

Zhang, Z., Wong, D. S., Xu, J., and Feng, D. (2006).
Certificateless public-key signature: Security model and efficient construction.
In Zhou, J., Yung, M., and Bao, F., editors, *ACNS*, volume 3989 of *Lecture Notes in Computer Science*, pages 293–308.

Crypto at UNICAMP

- R. Dahab and Julio López.
- Efficient implementation of elliptic curve algorithms, pairings, crypto for sensor networks, formal methods.
- National (Barreto, van de Graaf) and international collaboration (Menezes, Hankerson, Scott, Koç).
- Yearly Workshop on Crypto Algorithms and Protocols (WCAP).
- National PKI working groups.