Foundation of Cryptography (0368-4162-01), Lecture 7

MACs and Signatures

Iftach Haitner, Tel Aviv University

December 27, 2011

Section 1

Message Authentication Code (MAC)

Message Authentication Code (MAC)

Definition 1 (MAC)

A trippet of PPT's (Gen, Mac, Vrfy) such that

1. $\mathrm{Gen}(1^n)$ outputs a key $k \in \{0,1\}^*$
2. $\mathrm{Mac}(k, m)$ outputs a "tag" t
3. $\mathrm{Vrfy}(k, m, t)$ output 1 (YES) or 0 (NO)

Message Authentication Code (MAC)

Definition 1 (MAC)

A trippet of PPT's (Gen, Mac, Vrfy) such that

1. $\text{Gen}(1^n)$ outputs a key $k \in \{0,1\}^*$
2. $\text{Mac}(k, m)$ outputs a “tag" t
3. $\text{Vrfy}(k, m, t)$ output 1 (YES) or 0 (NO)

Consistency: $\text{Vrfy}_k(m, t) = 1$ for any $k \in \text{Supp}(\text{Gen}(1^n))$, $m \in \{0,1\}^n$ and $t = \text{Mac}_k(m)$

Message Authentication Code (MAC)

Definition 1 (MAC)

A trippet of PPT's (Gen, Mac, Vrfy) such that

1. $\text{Gen}(1^n)$ outputs a key $k \in \{0,1\}^*$
2. $\text{Mac}(k, m)$ outputs a "tag" t
3. $\text{Vrfy}(k, m, t)$ output 1 (YES) or 0 (NO)

Consistency: $\text{Vrfy}_k(m, t) = 1$ for any $k \in \text{Supp}(\text{Gen}(1^n))$, $m \in \{0,1\}^n$ and $t = \text{Mac}_k(m)$

Definition 2 (Existential unforgability)

A MAC (Gen, Mac, Vrfy) is existential unforgeable (EU), if for any oracle-aided PPT A:

$$\Pr\Big[k \leftarrow \text{Gen}(1^n); (m,t) \leftarrow \mathsf{A}^{\text{Mac}_k, \text{Vrfy}_k}(1^n) : \\ \text{Vrfy}_k(m,t) = 1 \wedge \text{Mac}_k \text{ was not asked on } m\Big] = \text{neg}(n)$$

- “Private key" definition

- “Private key" definition
- Security definition too strong?

- “Private key" definition
- Security definition too strong? Any message? Use of Verifier?

- “Private key" definition
- Security definition too strong? Any message? Use of Verifier?
- “Replay attacks"

- “Private key" definition
- Security definition too strong? Any message? Use of Verifier?
- “Replay attacks"
- strong MACS

Length-restricted MACs

Definition 3 (Length-restricted MAC)

Same as in Definition 1, but for $k \in \text{Supp}(G(1^n))$, Mac_k and Vrfy_k only accept messages of length n.

Bounded-query MACs

Definition 4 (ℓ-time MAC)

A MAC scheme is existential unforgeable against ℓ queries (for short, ℓ-time MAC), if it is existential unforgeable as in Definition 2, but A can only ask for ℓ queries.

Section 2

Constructions

Zero-time, restricted length, MAC

Construction 5 (Zero-time, restricted length, MAC)

- $\text{Gen}(1^n)$: outputs $k \leftarrow \{0,1\}^n$
- $\text{Mac}_k(m) = k$
- $\text{Vrfy}_k(m,t) = 1$, iff $t = k$

Zero-time, restricted length, MAC

Construction 5 (Zero-time, restricted length, MAC)

- $\mathsf{Gen}(1^n)$: outputs $k \leftarrow \{0,1\}^n$
- $\mathsf{Mac}_k(m) = k$
- $\mathsf{Vrfy}_k(m,t) = 1$, iff $t = k$

Claim 6

The above scheme is a length-restricted, zero-time MAC

ℓ-wise independent hash

Definition 7 (ℓ-wise independent)

A function family $\mathcal{H}$ from $\{0,1\}^n$ to $\{0,1\}^m$ is ℓ-wise independent, where $\ell \in \mathbb{N}$, if for every distinct $x_1, \ldots, x_\ell \in \{0,1\}^n$ and every $y_1, \ldots, y_\ell \in \{0,1\}^m$, it holds that $\Pr_{h \leftarrow \mathcal{H}}[h(x_1) = y_1 \wedge \cdots \wedge h(x_\ell) = y_\ell] = 2^{-\ell m}$.

ℓ-times, restricted length, MAC

Construction 8 (ℓ-time MAC)

Let $\mathcal{H} = \{\mathcal{H}_n : \{0,1\}^n \mapsto \{0,1\}^n\}$ be an efficient $(\ell + 1)$-wise independent function family.

- $\text{Gen}(1^n)$: outputs $h \leftarrow \mathcal{H}_n$
- $\text{Mac}(h, m) = h(m)$
- $\text{Vrfy}(h, m, t) = 1$, iff $t = h(m)$

ℓ-times, restricted length, MAC

Construction 8 (ℓ-time MAC)

Let $\mathcal{H} = \{\mathcal{H}_n \colon \{0,1\}^n \mapsto \{0,1\}^n\}$ be an efficient $(\ell + 1)$-wise independent function family.

- $\mathsf{Gen}(1^n)$: outputs $h \leftarrow \mathcal{H}_n$
- $\mathsf{Mac}(h, m) = h(m)$
- $\mathsf{Vrfy}(h, m, t) = 1$, iff $t = h(m)$

Claim 9

The above scheme is a length-restricted, ℓ-time MAC

ℓ-times, restricted length, MAC

Construction 8 (ℓ-time MAC)

Let $\mathcal{H} = \{\mathcal{H}_n \colon \{0,1\}^n \mapsto \{0,1\}^n\}$ be an efficient $(\ell+1)$-wise independent function family.

- $\mathsf{Gen}(1^n)$: outputs $h \leftarrow \mathcal{H}_n$
- $\mathsf{Mac}(h, m) = h(m)$
- $\mathsf{Vrfy}(h, m, t) = 1$, iff $t = h(m)$

Claim 9

The above scheme is a length-restricted, ℓ-time MAC

Proof: HW

OWF $\Longrightarrow$ existential unforgeable MAC

Construction 10

Same as Construction 8, but uses function $\mathcal{F} = \{\mathcal{F}_n \colon \{0,1\}^n \mapsto \{0,1\}^n\}$ instead of $\mathcal{H}$.

Claim 11

Assuming that $\mathcal{F}$ is a PRF, then Construction 10 is an existential unforgeable MAC.

OWF $\Longrightarrow$ existential unforgeable MAC

Construction 10

Same as Construction 8, but uses function $\mathcal{F} = \{\mathcal{F}_n \colon \{0,1\}^n \mapsto \{0,1\}^n\}$ instead of $\mathcal{H}$.

Claim 11

Assuming that $\mathcal{F}$ is a PRF, then Construction 10 is an existential unforgeable MAC.

Proof:

OWF $\Longrightarrow$ existential unforgeable MAC

Construction 10

Same as Construction 8, but uses function $\mathcal{F} = \{\mathcal{F}_n \colon \{0,1\}^n \mapsto \{0,1\}^n\}$ instead of $\mathcal{H}$.

Claim 11

Assuming that $\mathcal{F}$ is a PRF, then Construction 10 is an existential unforgeable MAC.

Proof: Easy to prove if $\mathcal{F}$ is a family of random functions. Hence, also holds in case $\mathcal{F}$ is a PRF. $\square$

Collision Resistant Hash Family

Definition 12 (collision resistant hash family (CRH))

A function family $\mathcal{H} = \{\mathcal{H}_n \colon \{0,1\}^* \mapsto \{0,1\}^n\}$ is collision resistant, if

$$\Pr[h \leftarrow \mathcal{H}_n, (x, x') \leftarrow \mathsf{A}(1^n, h) \colon x \neq x' \in \{0,1\}^* \\ \wedge h(x) = h(x')] = \operatorname{neg}(n)$$

for any PPT A.

Collision Resistant Hash Family

Definition 12 (collision resistant hash family (CRH))

A function family $\mathcal{H} = \{\mathcal{H}_n \colon \{0,1\}^* \mapsto \{0,1\}^n\}$ is collision resistant, if

$$\Pr[h \leftarrow \mathcal{H}_n, (x, x') \leftarrow \mathsf{A}(1^n, h) \colon x \neq x' \in \{0,1\}^* \wedge h(x) = h(x')] = \operatorname{neg}(n)$$

for any PPT A.

- Not known to be implied by OWF

Length restricted MAC $\Longrightarrow$ MAC

Construction 13 (Length restricted MAC $\Longrightarrow$ MAC)

Let $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ be a length-restricted MAC, and let $\mathcal{H} = \{\mathcal{H}_n \colon \{0,1\}^* \mapsto \{0,1\}^n\}$ be an efficient function family.

- $\mathsf{Gen}'(1^n)$: $k \leftarrow \mathsf{Gen}(1^n)$, $h \leftarrow \mathcal{H}_n$. Set $k' = (k, h)$
- $\mathsf{Mac}'_{k,h}(m) = \mathsf{Mac}_k(h(m))$
- $\mathsf{Vrfy}'_{k,h}(t, m) = \mathsf{Vrfy}_k(t, h(m))$

Length restricted MAC $\Longrightarrow$ MAC

Construction 13 (Length restricted MAC $\Longrightarrow$ MAC)

Let (Gen, Mac, Vrfy) be a length-restricted MAC, and let $\mathcal{H} = \{\mathcal{H}_n \colon \{0,1\}^* \mapsto \{0,1\}^n\}$ be an efficient function family.

- $\mathrm{Gen}'(1^n)$: $k \leftarrow \mathrm{Gen}(1^n)$, $h \leftarrow \mathcal{H}_n$. Set $k' = (k, h)$
- $\mathrm{Mac}'_{k,h}(m) = \mathrm{Mac}_k(h(m))$
- $\mathrm{Vrfy}'_{k,h}(t, m) = \mathrm{Vrfy}_k(t, h(m))$

Claim 14

Assume $\mathcal{H}$ is an efficient collision-resistant family and (Gen, Mac, Vrfy) is existential unforgeable, then (Gen′, Mac′, Vrfy′) is existential unforgeable MAC.

Length restricted MAC $\Longrightarrow$ MAC

Construction 13 (Length restricted MAC $\Longrightarrow$ MAC)

Let (Gen, Mac, Vrfy) be a length-restricted MAC, and let $\mathcal{H} = \{\mathcal{H}_n \colon \{0,1\}^* \mapsto \{0,1\}^n\}$ be an efficient function family.

- $\mathrm{Gen}'(1^n)$: $k \leftarrow \mathrm{Gen}(1^n)$, $h \leftarrow \mathcal{H}_n$. Set $k' = (k, h)$
- $\mathrm{Mac}'_{k,h}(m) = \mathrm{Mac}_k(h(m))$
- $\mathrm{Vrfy}'_{k,h}(t, m) = \mathrm{Vrfy}_k(t, h(m))$

Claim 14

Assume $\mathcal{H}$ is an efficient collision-resistant family and (Gen, Mac, Vrfy) is existential unforgeable, then (Gen′, Mac′, Vrfy′) is existential unforgeable MAC.

Proof: ?

Section 3

Signature Schemes

Definition

Definition 15 (Signature schemes)

A trippet of PPT's (Gen, Sign, Vrfy) such that

1. $\text{Gen}(1^n)$ outputs a pair of keys $(s, v) \in \{0,1\}^* \times \{0,1\}^*$
2. $\text{Sign}(s, m)$ outputs a "signature" $\sigma \in \{0,1\}^*$
3. $\text{Vrfy}(v, m, \sigma)$ outputs 1 (YES) or 0 (NO)

Definition

Definition 15 (Signature schemes)

A trippet of PPT's (Gen, Sign, Vrfy) such that

1. $\text{Gen}(1^n)$ outputs a pair of keys $(s, v) \in \{0,1\}^* \times \{0,1\}^*$
2. $\text{Sign}(s, m)$ outputs a "signature" $\sigma \in \{0,1\}^*$
3. $\text{Vrfy}(v, m, \sigma)$ outputs 1 (YES) or 0 (NO)

Consistency: $\text{Vrfy}_v(m, \sigma) = 1$ for any $(s, v) \in \text{Supp}(\text{Gen}(1^n))$, $m \in \{0,1\}^*$ and $\sigma \in \text{Supp}(\text{Sign}_s(m))$

Definition

Definition 15 (Signature schemes)

A trippet of PPT's (Gen, Sign, Vrfy) such that

1. $\text{Gen}(1^n)$ outputs a pair of keys $(s, v) \in \{0,1\}^* \times \{0,1\}^*$
2. $\text{Sign}(s, m)$ outputs a “signature" $\sigma \in \{0,1\}^*$
3. $\text{Vrfy}(v, m, \sigma)$ outputs 1 (YES) or 0 (NO)

Consistency: $\text{Vrfy}_v(m, \sigma) = 1$ for any $(s, v) \in \text{Supp}(\text{Gen}(1^n))$, $m \in \{0,1\}^*$ and $\sigma \in \text{Supp}(\text{Sign}_s(m))$

Definition 16 (Existential unforgability)

A signature scheme is existential unforgeable (EU), if for any oracle-aided PPT A

$$\Pr\Big[(s, v) \leftarrow \text{Gen}(1^n); (m, \sigma) \leftarrow \mathsf{A}^{\text{Sign}_s}(1^n, v) : \text{Vrfy}_v(m, \sigma) = 1 \wedge \text{Sign}_s \text{ was not asked on } m\Big] = \text{neg}(n)$$

- Signature $\implies$ MAC

- Signature $\implies$ MAC
- “Harder" to construct than MACs: (even restricted forms) require OWF

- Signature $\implies$ MAC
- “Harder" to construct than MACs: (even restricted forms) require OWF
- Oracle access to Vrfy is not given

- Signature $\implies$ MAC
- “Harder" to construct than MACs: (even restricted forms) require OWF
- Oracle access to Vrfy is not given
- Strong existential unforgeable signatures (for short, strong signatures): infeasible to generate *any* new valid signatures (even for message for which a signature was asked)

- Signature $\implies$ MAC
- “Harder" to construct than MACs: (even restricted forms) require OWF
- Oracle access to Vrfy is not given
- Strong existential unforgeable signatures (for short, strong signatures): infeasible to generate *any* new valid signatures (even for message for which a signature was asked)

Theorem 17

OWFs imply strong existential unforgeable signatures.

Section 4

OWFs $\implies$ Signatures

Length-restricted Signatures

Definition 18 (Length-restricted Signatures)

Same as in Definition 15, but for $(s, v) \in \text{Supp}(G(1^n))$, Sign_s and Vrfy_v only accept messages of length n.

Bounded-query Signatures

Definition 19 (ℓ-time signatures)

A signature scheme is existential unforgeable against ℓ-query (for short, ℓ-time signature), if it is existential unforgeable as in Definition 16, but A can only ask for ℓ queries.

Bounded-query Signatures

Definition 19 (ℓ-time signatures)

A signature scheme is existential unforgeable against ℓ-query (for short, ℓ-time signature), if it is existential unforgeable as in Definition 16, but A can only ask for ℓ queries.

Claim 20

Assuming CRH exists: length restricted, one-time signatures, imply one-time signatures.

OWF $\implies$ length restricted, One Time Signature

Construction 21 (length restricted, one time signature)

Let $f\colon \{0,1\}^n \mapsto \{0,1\}^n$.

1. $\text{Gen}(1^n)$: $s_1^0, s_1^1, \ldots, s_n^0, s_n^1 \leftarrow \{0,1\}^n$, let $s = (s_1^0, s_1^1, \ldots, s_n^0, s_n^1)$ and $v = (v_1^0 = f(s_1^0), v_1^1 = f(s_1^1), \ldots, v_n^0 = f(s_n^0), v_n^1 = f(s_n^1))$
2. $\text{Sign}(s, m)$: Output $(s_1^{m_1}, \ldots, s_n^{m_n})$
3. $\text{Vrfy}(v, m, \sigma = (\sigma_1, \ldots, \sigma_n))$ check that $f(\sigma_i) = v_{m_i}$ for all $i \in [n]$

OWF $\Longrightarrow$ length restricted, One Time Signature

Construction 21 (length restricted, one time signature)

Let $f\colon \{0,1\}^n \mapsto \{0,1\}^n$.

1. Gen(1^n): $s_1^0, s_1^1, \ldots, s_n^0, s_n^1 \leftarrow \{0,1\}^n$, let $s = (s_1^0, s_1^1, \ldots, s_n^0, s_n^1)$ and $v = (v_1^0 = f(s_1^0), v_1^1 = f(s_1^1), \ldots, v_n^0 = f(s_n^0), v_n^1 = f(s_n^1))$
2. Sign(s, m): Output $(s_1^{m_1}, \ldots, s_n^{m_n})$
3. Vrfy($v, m, \sigma = (\sigma_1, \ldots, \sigma_n)$) check that $f(\sigma_i) = v_{m_i}$ for all $i \in [n]$

Lemma 22

Assume that f is a OWF, then scheme from Construction 21 is a length restricted one-time signature scheme

Proving Lemma 22

Let a PPT A, $\mathcal{I} \subseteq \mathbb{N}$ and $p \in$ poly that break the security of Construction 21, we use A to invert f.

Algorithm 23 (Inv)

Input: $y \in \{0,1\}^n$

1. Choose $(s, v) \leftarrow Gen(1^n)$ and replace $v^{i^*}_{j^*}$ for a random $i^* \in [n]$ and $j^* \in \{0,1\}$, with y.
2. If $A(1^n, v)$ asks to sign message $m \in \{0,1\}^n$ with $m_{i^*} = j^*$ abort, otherwise use s to answer the query.
3. Let (m, σ) be A's output. If σ is not a valid signature for m, or $m_{i^*} \neq j^*$, abort.
   Otherwise, return σ_{i^*}.

Proving Lemma 22

Let a PPT A, $\mathcal{I} \subseteq \mathbb{N}$ and $p \in$ poly that break the security of Construction 21, we use A to invert f.

Algorithm 23 (Inv)

Input: $y \in \{0,1\}^n$

1. Choose $(s, v) \leftarrow Gen(1^n)$ and replace $v_{j^*}^{i^*}$ for a random $i^* \in [n]$ and $j^* \in \{0,1\}$, with y.
2. If $A(1^n, v)$ asks to sign message $m \in \{0,1\}^n$ with $m_{i^*} = j^*$ abort, otherwise use s to answer the query.
3. Let (m, σ) be A's output. If σ is not a valid signature for m, or $m_{i^*} \neq j^*$, abort.
   Otherwise, return σ_{i^*}.

v is distributed as it is in the real “signature game" (ind. of i^* and j^*).

Proving Lemma 22

Let a PPT A, $\mathcal{I} \subseteq \mathbb{N}$ and $p \in$ poly that break the security of Construction 21, we use A to invert f.

Algorithm 23 (Inv)

Input: $y \in \{0,1\}^n$

1. Choose $(s, v) \leftarrow Gen(1^n)$ and replace $v_{j^*}^{i^*}$ for a random $i^* \in [n]$ and $j^* \in \{0,1\}$, with y.
2. If $A(1^n, v)$ asks to sign message $m \in \{0,1\}^n$ with $m_{i^*} = j^*$ abort, otherwise use s to answer the query.
3. Let (m, σ) be A's output. If σ is not a valid signature for m, or $m_{i^*} \neq j^*$, abort.
   Otherwise, return σ_{i^*}.

v is distributed as it is in the real "signature game" (ind. of i^* and j^*). Therefore Inv inverts f w.p. $\frac{1}{2np(n)}$ for any $n \in \mathcal{I}$.

Stateful schemes (also known as, Memory-dependant schemes)

Definition 24 (Stateful scheme)

Same as in Definition 15, but Sign might keep state.

Stateful schemes (also known as, Memory-dependant schemes)

Definition 24 (Stateful scheme)

Same as in Definition 15, but Sign might keep state.

- Make sense in many applications (e.g., , smartcards)

Stateful schemes (also known as, Memory-dependant schemes)

Definition 24 (Stateful scheme)

Same as in Definition 15, but Sign might keep state.

- Make sense in many applications (e.g., , smartcards)
- We'll use it a building block for building a stateless scheme

Naive construction

Let (Gen, Sign, Vrfy) be a one-time signature scheme.

Construction 25 (Naive construction)

1. $\text{Gen}'(1^n)$ outputs $(s_1, v_1) = \text{Gen}(1^n)$.
2. $\text{Sign}'_{s_1}(m_i)$, where m_i is i'th message to sign:
   Let $((m_1, \sigma'_1), \ldots, (m_{i-1}, \sigma'_{i-1}))$ be the previously signed pairs of messages/signatures.
   1. Let $(s_{i+1}, v_{i+1}) \leftarrow \text{Gen}(1^n)$
   2. Let $\sigma_i = \text{Sign}_{s_i}(m_i, v_{i+1})$, and output $\sigma'_i = (\sigma'_{i-1}, m_i, v_{i+1}, \sigma_i)$.[a]
3. $\text{Vrfy}'_{v_1}(m, \sigma' = (m_1, v_2, \sigma_1), \ldots, (m_i, v_{i+1}, \sigma_i))$:
   1. Verify $\text{Vrfy}_{v_j}((m_j, v_{j+1}), \sigma_j) = 1$ for every $j \in [i]$
   2. Verify $m_i = m$

[a] Where σ'_0 is the empty string.

1. State is used for maintaining the private key (e.g., s_i') and to prevent using the same one-time signature twice.
2. Inefficient scheme, thought still polynomial, both running time and signature size are linear in number of signatures

Stateful schemes

1. State is used for maintaining the private key (e.g., s_i') and to prevent using the same one-time signature twice.
2. Inefficient scheme, thought still polynomial, both running time and signature size are linear in number of signatures
3. Critically uses the fact that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is works for any length

Lemma 26

Assume that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is one time signature scheme, then* $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ *is a stateful existential unforgeable signature scheme.*

Lemma 26

Assume that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is one time signature scheme, then* $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ *is a stateful existential unforgeable signature scheme.*

Proof: Let a PPT A', $\mathcal{I} \subseteq \mathbb{N}$ and $p \in \text{poly}$ that breaks the security of $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$, we present a PPT A that breaks the security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$.

Lemma 26

Assume that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is one time signature scheme, then* $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ *is a stateful existential unforgeable signature scheme.*

Proof: Let a PPT A', $\mathcal{I} \subseteq \mathbb{N}$ and $p \in$ poly that breaks the security of $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$, we present a PPT A that breaks the security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$.

- We assume for simplicity that p also bounds the query complexity of A'

Proving Lemma 26 cont.

Let the random variables
$(m, \sigma = (m_1, v_2, \sigma_1), \ldots, (m_q, v_{q+1}, \sigma_q))$ be the pair output by A′

Proving Lemma 26 cont.

Let the random variables
$(m, \sigma = (m_1, v_2, \sigma_1), \ldots, (m_q, v_{q+1}, \sigma_q))$ be the pair output by A′

Claim 27

Whenever A′ succeeds, $\exists \widetilde{i} = \widetilde{i}(m, \sigma) \in [q]$ such that:

1. Sign′ *was not* asked by A′ on $m_{\widetilde{i}}$.
2. Sign′ *was* asked by A′ on m_i, for every $i \in [\widetilde{i} - 1]$

Proving Lemma 26 cont.

Let the random variables
$(m, \sigma = (m_1, v_2, \sigma_1), \ldots, (m_q, v_{q+1}, \sigma_q))$ be the pair output by A′

Claim 27

Whenever A′ succeeds, $\exists \widetilde{i} = \widetilde{i}(m, \sigma) \in [q]$ such that:

1. Sign′ *was not* asked by A′ on $m_{\widetilde{i}}$.
2. Sign′ *was* asked by A′ on m_i, for every $i \in [\widetilde{i} - 1]$

Proof:

Proving Lemma 26 cont.

Let the random variables
$(m, \sigma = (m_1, v_2, \sigma_1), \ldots, (m_q, v_{q+1}, \sigma_q))$ be the pair output by A′

Claim 27

Whenever A′ succeeds, $\exists \widetilde{i} = \widetilde{i}(m, \sigma) \in [q]$ such that:

1. Sign′ *was not* asked by A′ on $m_{\widetilde{i}}$.
2. Sign′ *was* asked by A′ on m_i, for every $i \in [\widetilde{i} - 1]$

Proof: Let $\widetilde{i}$ be the maximal index such that condition (2) holds (cannot be $q + 1$).□

Proving Lemma 26 cont.

Let the random variables $(m, \sigma = (m_1, v_2, \sigma_1), \ldots, (m_q, v_{q+1}, \sigma_q))$ be the pair output by A′

Claim 27

Whenever A′ succeeds, $\exists \widetilde{i} = \widetilde{i}(m, \sigma) \in [q]$ such that:

1. Sign′ *was not* asked by A′ on $m_{\widetilde{i}}$.
2. Sign′ *was* asked by A′ on m_i, for every $i \in [\widetilde{i} - 1]$

Proof: Let $\widetilde{i}$ be the maximal index such that condition (2) holds (cannot be $q + 1$).□

- Let $\widetilde{m} = (m_{\widetilde{i}}, v_{\widetilde{i}+1})$, and let $s_{\widetilde{i}}$ be the signing key generated together with $v_{\widetilde{i}}$.

Proving Lemma 26 cont.

Let the random variables
$(m, \sigma = (m_1, v_2, \sigma_1), \ldots, (m_q, v_{q+1}, \sigma_q))$ be the pair output by A′

Claim 27

Whenever A′ succeeds, $\exists \widetilde{i} = \widetilde{i}(m, \sigma) \in [q]$ such that:

1. Sign′ *was not* asked by A′ on $m_{\widetilde{i}}$.
2. Sign′ *was* asked by A′ on m_i, for every $i \in [\widetilde{i} - 1]$

Proof: Let $\widetilde{i}$ be the maximal index such that condition (2) holds (cannot be $q + 1$).$\square$

- Let $\widetilde{m} = (m_{\widetilde{i}}, v_{\widetilde{i}+1})$, and let $s_{\widetilde{i}}$ be the signing key generated together with $v_{\widetilde{i}}$.
- Hence, $\text{Sign}_{s_{\widetilde{i}}}(\sigma_{\widetilde{i}}, \widetilde{m}) = 1$, and Sign_{s_i} was not queried by Sign'_s on $\widetilde{m}$.

Definition of A

Algorithm 28 (A)

Input: v, 1^n
Oracle: Sign_s

1. Choose $i^* \leftarrow [p = p(n)]$ and $(s', v') \leftarrow \mathrm{Gen}'(1^n)$.
2. Emulate a random execution of $\mathsf{A}'^{\mathrm{Sign}'_{s'}}$ with a single twist:
   - On the i^*'th call to $\mathrm{Sign}'_{s'}$, set $v_{i^*} = v$ (rather then choosing it via Gen)
   - When need to sign using s_{i^*}, use Sign_s.
3. Let $(m, \sigma = (m_1, v_1, \sigma_1), \ldots, (m_q, v_q, \sigma_q)) \leftarrow \mathsf{A}'$
4. Output $((m_{i^*}, v_{i^*}), \sigma_{i^*})$ (abort if $i^* > q$))

Definition of A

Algorithm 28 (A)

Input: v, 1^n
Oracle: Sign_s

1. Choose $i^* \leftarrow [p = p(n)]$ and $(s', v') \leftarrow \mathsf{Gen}'(1^n)$.
2. Emulate a random execution of $\mathsf{A}'^{\mathsf{Sign}'_{s'}}$ with a single twist:
   - On the i^*'th call to $\mathsf{Sign}'_{s'}$, set $v_{i^*} = v$ (rather then choosing it via Gen)
   - When need to sign using s_{i^*}, use Sign_s.
3. Let $(m, \sigma = (m_1, v_1, \sigma_1), \ldots, (m_q, v_q, \sigma_q)) \leftarrow \mathsf{A}'$
4. Output $((m_{i^*}, v_{i^*}), \sigma_{i^*})$ (abort if $i^* > q$))

- Sign_s is called at most once

Definition of A

Algorithm 28 (A)

Input: v, 1^n
Oracle: Sign_s

1. Choose $i^* \leftarrow [p = p(n)]$ and $(s', v') \leftarrow \mathrm{Gen}'(1^n)$.
2. Emulate a random execution of $\mathrm{A}'^{\mathrm{Sign}'_{s'}}$ with a single twist:
   - On the i^*'th call to $\mathrm{Sign}'_{s'}$, set $v_{i^*} = v$ (rather then choosing it via Gen)
   - When need to sign using s_{i^*}, use Sign_s.
3. Let $(m, \sigma = (m_1, v_1, \sigma_1), \ldots, (m_q, v_q, \sigma_q)) \leftarrow \mathrm{A}'$
4. Output $((m_{i^*}, v_{i^*}), \sigma_{i^*})$ (abort if $i^* > q$))

- Sign_s is called at most once
- The emulated game $\mathrm{A}'^{\mathrm{Sign}'_{s'}}$ has the “right" distribution.

Definition of A

Algorithm 28 (A)

Input: v, 1^n
Oracle: Sign_s

1. Choose $i^* \leftarrow [p = p(n)]$ and $(s', v') \leftarrow \mathrm{Gen}'(1^n)$.
2. Emulate a random execution of $\mathsf{A}'^{\mathrm{Sign}'_{s'}}$ with a single twist:
   - On the i^*'th call to $\mathrm{Sign}'_{s'}$, set $v_{i^*} = v$ (rather then choosing it via Gen)
   - When need to sign using s_{i^*}, use Sign_s.
3. Let $(m, \sigma = (m_1, v_1, \sigma_1), \ldots, (m_q, v_q, \sigma_q)) \leftarrow \mathsf{A}'$
4. Output $((m_{i^*}, v_{i^*}), \sigma_{i^*})$ (abort if $i^* > q$))

- Sign_s is called at most once
- The emulated game $\mathsf{A}'^{\mathrm{Sign}'_{s'}}$ has the "right" distribution.
- A breaks (Gen, Sign, Vrfy) whenever $i^* = \widetilde{i} > 1$.

Analysis of A

For any $n \in \mathcal{I}$

$$\begin{aligned}
&\Pr[\mathsf{A}(1^n) \text{ breaks } (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})] \\
&\quad \geq \Pr_{i^* \leftarrow [p=p(n)]}[i = \widetilde{i}] \\
&\quad \geq \frac{1}{p} \cdot \Pr[\mathsf{A}' \text{ breaks } (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')] \geq \frac{1}{p(n)^2}
\end{aligned}$$

"Somewhat"-Stateful Schemes

A one-time scheme (Gen, Sign, Vrfy), and $\ell = \ell(n) \in \omega(\log n)$

Construction 29

- $\mathrm{Gen}'(1^n)$: output $(s_\lambda, v_\lambda) \leftarrow \mathrm{Gen}(1^n)$.
- $\mathrm{Sign}'_s(m)$: choose *unused* $\bar{r} \in \{0,1\}^\ell$
  1. For $i = 0$ to $\ell - 1$: if $a_{\bar{r}_{1,\ldots,i}}$ was not set:
     1. For both $j \in \{0,1\}$, let $(s_{\bar{r}_{1,\ldots,i},j}, v_{\bar{r}_{1,\ldots,i},j}) \leftarrow \mathrm{Gen}(1^n)$
     2. $\sigma_{\bar{r}_{1,\ldots,i}} = \mathrm{Sign}_{s_{\bar{r}_{1,\ldots,i}}}(a_{1,\ldots,i} = (v_{\bar{r}_{1,\ldots,i},0}, v_{\bar{r}_{1,\ldots,i},1}))$
  2. Output $(\bar{r}, a_\lambda, \sigma_\lambda, \ldots, a_{\bar{r}_{1,\ldots,\ell-1}}, \sigma_{\bar{r}_{1,\ldots,\ell-1}}, \sigma_{\bar{r}} = \mathrm{Sign}_{s_{\bar{r}}}(m))$
- $\mathrm{Vrfy}'_v(m, \sigma' = (\bar{r}, a_\lambda, \sigma_\lambda, \ldots, a_{\bar{r}-1}, \sigma_{\bar{r}_{1,\ldots,\ell-1}}, \sigma_{\bar{r}})$
  1. Verify $\mathrm{Vrfy}_{v_{\bar{r}_{1,\ldots,i}}}(a_{\bar{r}_{1,\ldots,i}}, \sigma_{\bar{r}_{1,\ldots,i}}) = 1$ for every $i \in \{0, \ldots, \ell - 1\}$
  2. Verify $\mathrm{Vrfy}_{v_{\bar{r}}}(m, \sigma_{\bar{r}}) = 1$ (where $v_{\bar{r}} = (a_{\bar{r}})_{\bar{r}[\ell]}$)

1. More efficient scheme

1. More efficient scheme
2. Sign$'$ does not keep track of the message history.

1. More efficient scheme
2. Sign$'$ does not keep track of the message history.
3. Each leaf is visited at most once.

1. More efficient scheme
2. Sign' does not keep track of the message history.
3. Each leaf is visited at most once.
4. Each one-time signature is used once.

Lemma 30

Assume that $(\mathrm{Gen}, \mathrm{Sign}, \mathrm{Vrfy})$ *is one time signature scheme, then* $(\mathrm{Gen}', \mathrm{Sign}', \mathrm{Vrfy}')$ *is a stateful existential unforgeable signature scheme.*

Proof:

Lemma 30

Assume that (Gen, Sign, Vrfy) *is one time signature scheme, then* (Gen′, Sign′, Vrfy′) *is a stateful existential unforgeable signature scheme.*

Proof: Let $(m, \sigma' = (\bar{r}, a_\lambda, \sigma_\lambda, \ldots, a_{\bar{r}-1}, \sigma_{\bar{r}_{1,\ldots,\ell-1}}, \sigma_{\bar{r}})$ be the output of a cheating A′ and let $a_{\bar{r}} = m$

Lemma 30

Assume that (Gen, Sign, Vrfy) *is one time signature scheme, then* (Gen′, Sign′, Vrfy′) *is a stateful existential unforgeable signature scheme.*

Proof: Let $(m, \sigma' = (\bar{r}, a_\lambda, \sigma_\lambda, \ldots, a_{\bar{r}-1}, \sigma_{\bar{r}_{1,\ldots,\ell-1}}, \sigma_{\bar{r}})$ be the output of a cheating A′ and let $a_{\bar{r}} = m$

Claim 31

Whenever A′ succeeds, $\exists \widetilde{i} = \widetilde{i}(m, \sigma') \in \{0, \ldots, \ell\}$ such that:

1. Sign'_s queried $\mathrm{Sign}_{s_{\bar{r}_{1,\ldots,i}}}(a_{\bar{r}_{1,\ldots,i}})$ for every $i \in [\widetilde{i} - 1]$, where $s_{\bar{r}_{1,\ldots,i}}$ is the value sampled by Sign′ when sampling $a_{\bar{r}_{1,\ldots,i-1}}$ (or s_λ, if $\widetilde{i} = 0$)
2. Sign'_s did not query $\mathrm{Sign}_{s_{\bar{r}_{1,\ldots,i}}}(a_{\bar{r}_{1,\ldots,i}})$.

Stateless Scheme

Inefficient scheme:

Let $\Pi_{\ell,q}$ be the set of random functions from $\{0,1\}^*$ to $\{0,1\}^q$.

Stateless Scheme

Inefficient scheme:

Let $\Pi_{\ell,q}$ be the set of random functions from $\{0,1\}^*$ to $\{0,1\}^q$.

1. $\mathsf{Gen}'(1^n)$: let $(s,v) \leftarrow \mathsf{Gen}(1^n)$ and $\pi \leftarrow \Pi_{\ell(n),q(n)}$, where $q \in \mathrm{poly}$ is large enough for the application below, and outputs $(s' = (s,\pi), v' = v)$

Stateless Scheme

Inefficient scheme:

Let $\Pi_{\ell,q}$ be the set of random functions from $\{0,1\}^*$ to $\{0,1\}^q$.

1. $\text{Gen}'(1^n)$: let $(s, v) \leftarrow \text{Gen}(1^n)$ and $\pi \leftarrow \Pi_{\ell(n),q(n)}$, where $q \in$ poly is large enough for the application below, and outputs $(s' = (s, \pi), v' = v)$
2. $\text{Sign}'(1^n)$:
    1. choose $\bar{r} = \pi(0^\ell \circ m)_{1,\ldots,\ell}$

Stateless Scheme

Inefficient scheme:

Let $\Pi_{\ell,q}$ be the set of random functions from $\{0,1\}^*$ to $\{0,1\}^q$.

1. $\text{Gen}'(1^n)$: let $(s, v) \leftarrow \text{Gen}(1^n)$ and $\pi \leftarrow \Pi_{\ell(n),q(n)}$, where $q \in \text{poly}$ is large enough for the application below, and outputs $(s' = (s, \pi), v' = v)$
2. $\text{Sign}'(1^n)$:
   1. choose $\bar{r} = \pi(0^\ell \circ m)_{1,\ldots,\ell}$
   2. When setting $(s_{\bar{r}_{1,\ldots,i},j}, v_{\bar{r}_{1,\ldots,i},j}) \leftarrow \text{Gen}(1^n)$, use $\pi(\bar{r}_{1,\ldots,i}, j)$ as the randomness for Gen.

Stateless Scheme

Inefficient scheme:

Let $\Pi_{\ell,q}$ be the set of random functions from $\{0,1\}^*$ to $\{0,1\}^q$.

1. $\text{Gen}'(1^n)$: let $(s,v) \leftarrow \text{Gen}(1^n)$ and $\pi \leftarrow \Pi_{\ell(n),q(n)}$, where $q \in \text{poly}$ is large enough for the application below, and outputs $(s' = (s,\pi), v' = v)$
2. $\text{Sign}'(1^n)$:
   1. choose $\bar{r} = \pi(0^\ell \circ m)_{1,\ldots,\ell}$
   2. When setting $(s_{\bar{r}_{1,\ldots,i},j}, v_{\bar{r}_{1,\ldots,i},j}) \leftarrow \text{Gen}(1^n)$, use $\pi(\bar{r}_{1,\ldots,i},j)$ as the randomness for Gen.

- Sign' keeps no state

Stateless Scheme

Inefficient scheme:

Let $\Pi_{\ell,q}$ be the set of random functions from $\{0,1\}^*$ to $\{0,1\}^q$.

1. $\mathrm{Gen}'(1^n)$: let $(s,v) \leftarrow \mathrm{Gen}(1^n)$ and $\pi \leftarrow \Pi_{\ell(n),q(n)}$, where $q \in \mathrm{poly}$ is large enough for the application below, and outputs $(s' = (s,\pi), v' = v)$
2. $\mathrm{Sign}'(1^n)$:
    1. choose $\bar{r} = \pi(0^\ell \circ m)_{1,\ldots,\ell}$
    2. When setting $(s_{\bar{r}_{1,\ldots,i},j}, v_{\bar{r}_{1,\ldots,i},j}) \leftarrow \mathrm{Gen}(1^n)$, use $\pi(\bar{r}_{1,\ldots,i},j)$ as the randomness for Gen.

- Sign' keeps no state
- A single one-time signature key might be used several times, but always on *the same* message

Stateless Scheme

Inefficient scheme:

Let $\Pi_{\ell,q}$ be the set of random functions from $\{0,1\}^*$ to $\{0,1\}^q$.

1. $\mathsf{Gen}'(1^n)$: let $(s,v) \leftarrow \mathsf{Gen}(1^n)$ and $\pi \leftarrow \Pi_{\ell(n),q(n)}$, where $q \in$ poly is large enough for the application below, and outputs $(s' = (s,\pi), v' = v)$
2. $\mathsf{Sign}'(1^n)$:
   1. choose $\bar{r} = \pi(0^\ell \circ m)_{1,\ldots,\ell}$
   2. When setting $(s_{\bar{r}_{1,\ldots,i},j}, v_{\bar{r}_{1,\ldots,i},j}) \leftarrow \mathsf{Gen}(1^n)$, use $\pi(\bar{r}_{1,\ldots,i},j)$ as the randomness for Gen.

- Sign′ keeps no state
- A single one-time signature key might be used several times, but always on *the same* message

Efficient scheme:

Stateless Scheme

Inefficient scheme:

Let $\Pi_{\ell,q}$ be the set of random functions from $\{0,1\}^*$ to $\{0,1\}^q$.

1. $\text{Gen}'(1^n)$: let $(s,v) \leftarrow \text{Gen}(1^n)$ and $\pi \leftarrow \Pi_{\ell(n),q(n)}$, where $q \in \text{poly}$ is large enough for the application below, and outputs $(s' = (s,\pi), v' = v)$
2. $\text{Sign}'(1^n)$:
   1. choose $\bar{r} = \pi(0^\ell \circ m)_{1,\ldots,\ell}$
   2. When setting $(s_{\bar{r}_{1,\ldots,i},j}, v_{\bar{r}_{1,\ldots,i},j}) \leftarrow \text{Gen}(1^n)$, use $\pi(\bar{r}_{1,\ldots,i},j)$ as the randomness for Gen.

- Sign' keeps no state
- A single one-time signature key might be used several times, but always on *the same* message

Efficient scheme: use PRF

Without CRH

Definition 32 (target collision resistant (TCR))

A function family $\mathcal{H} = \{\mathcal{H}_n\}$ is target collision resistant, if any pair of PPT's $\mathsf{A}_1, \mathsf{A}_2$:

$$\Pr[(x, a) \leftarrow \mathsf{A}_1(1^n); h \leftarrow \mathcal{H}_n; x' \leftarrow \mathsf{A}_2(a, h) : x \neq x' \land h(x) = h(x')] = \operatorname{neg}(n)$$

Without CRH

Definition 32 (target collision resistant (TCR))

A function family $\mathcal{H} = \{\mathcal{H}_n\}$ is target collision resistant, if any pair of PPT's A_1, A_2:

$$\Pr[(x, a) \leftarrow A_1(1^n); h \leftarrow \mathcal{H}_n; x' \leftarrow A_2(a, h) : x \neq x' \wedge h(x) = h(x')] = \operatorname{neg}(n)$$

Theorem 33

OWFs imply efficient compressing TCRs.

Definition 34 (target one-time signatures)

A signature scheme (Gen, Sign, Vrfy) is target one-time existential unforgeable (for short, target one-time signature), if for any pair of PPT's $\mathsf{A}_1, \mathsf{A}_2$

$$\Pr\big[(m,a) \leftarrow \mathsf{A}_1(1^n); (s,v) \leftarrow \mathsf{Gen}(1^n); \\ (m',\sigma) \leftarrow \mathsf{A}(a, \mathsf{Sign}_s(m))\colon m' \neq m \wedge \mathsf{Vrfy}_v(m',\sigma) = 1\big] \\ = \mathrm{neg}(n)$$

Definition 34 (target one-time signatures)

A signature scheme (Gen, Sign, Vrfy) is target one-time existential unforgeable (for short, target one-time signature), if for any pair of PPT's A_1, A_2

$$\Pr\big[(m,a) \leftarrow A_1(1^n); (s,v) \leftarrow \mathrm{Gen}(1^n); (m',\sigma) \leftarrow A(a, \mathrm{Sign}_s(m)) \colon m' \neq m \wedge \mathrm{Vrfy}_v(m',\sigma) = 1\big] = \mathrm{neg}(n)$$

Claim 35

OWFs imply target one-time signatures.

Definition 36 (random one-time signatures)

A signature scheme (Gen, Sign, Vrfy) is random one-time existential unforgeable (for short, random one-time signature), if for any PPT A and any samplable ensemble $\mathcal{M} = \{\mathcal{M}_n\}_{n\in\mathbb{N}}$, it holds that

$$\Pr\big[m \leftarrow \mathcal{M}_n; (s, v) \leftarrow \mathsf{Gen}(1^n); (m', \sigma) \leftarrow \mathsf{A}(m, \mathsf{Sign}_s(m)) : m' \neq m \wedge \mathsf{Vrfy}_v(m', \sigma) = 1\big] = \mathsf{neg}(n)$$

Definition 36 (random one-time signatures)

A signature scheme (Gen, Sign, Vrfy) is random one-time existential unforgeable (for short, random one-time signature), if for any PPT A and any samplable ensemble $\mathcal{M} = \{\mathcal{M}_n\}_{n \in \mathbb{N}}$, it holds that

$$\Pr\left[m \leftarrow \mathcal{M}_n; (s, v) \leftarrow \mathrm{Gen}(1^n); (m', \sigma) \leftarrow \mathsf{A}(m, \mathrm{Sign}_s(m)) : m' \neq m \wedge \mathrm{Vrfy}_v(m', \sigma) = 1\right] = \mathrm{neg}(n)$$

Claim 37

Assume (Gen, Sign, Vrfy) is target one-time existential unforgeable, then it is random one-time existential unforgeable.

Lemma 38

Assume that (Gen, Sign, Vrfy) *is a target one-time signature scheme, then* (Gen′, Sign′, Vrfy′) *from Construction 29 is a stateful existential unforgeable signature scheme.*

Lemma 38

Assume that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is a target one-time signature scheme, then* $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ *from Construction 29 is a stateful existential unforgeable signature scheme.*

Proof: ?