Lecture 10: Signature Schemes (Cont.)

Notes by Yael Kalai

MIT - 6.5620
Lecture 10 (October 6, 2025)

> ***Warning:*** This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Recap

- Defined the notion of a signature scheme, which is the public-key analogue of a MAC.
- Constructed a one-time secure signature scheme (due to Lamport [1]), assuming the existence of one-way functions.
- Defined the notion of a collision resistant hash family.
- Introduced the Hash-then-Sign paradigm.
- The Hash-then-signed applied to Lamport's scheme gives a one-time signature scheme with message space $\{0,1\}^*$ assuming the existence of a collision resistant hash family.

Definition 1. A signature scheme is associated with a message space $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ and with three PPT algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, with the following syntax:

- Gen**:** Takes as input the security parameter 1^λ in unary and outputs a pair $(\mathsf{vk}, \mathsf{sk})$ of a public verification key and a secret signing key.
- Sign: Takes as input a secret signing key sk and a message $m \in \mathcal{M}_\lambda$ and outputs a signature σ.
- Ver**:** Takes as input a verification key vk, a message m and a signature σ and outputs $0/1$, indicating accept or reject.

A signature scheme is required to satisfy the following completeness guarantee: For every $\lambda \in \mathbb{N}$ and every $m \in \mathcal{M}_\lambda$,

$$\Pr[\mathsf{Ver}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1] = 1$$

where the probability is over $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and over the randomness of Sign (if it is randomized).[1]

[1] Ver is always deterministic.

Definition 2. A signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ with message space $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ is said to be existentially unforgeable against adaptive chosen message attacks if for every poly-size $\mathcal{A}$ there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$,

$$\Pr[\mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{vk}) = (m^*, \sigma^*) \text{ s.t. } \mathsf{Ver}(\mathsf{vk}, m^*, \sigma^*) = 1 \ \wedge \ m^* \notin Q] \leq \mu(\lambda)$$

where Q denotes the set of all oracle calls that $\mathcal{A}$ makes to the oracle, and the probability is over $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and over the randomness of Sign (if it is randomized).

It is said to be **one-time secure** if the above holds against any poly-size $\mathcal{A}$ that queries the oracle only once (i.e., $|Q| = 1$).

Today

In this lecture, we show how to take any **one-time secure** signature scheme with message space $\mathcal{M} = \{0,1\}^*$, and convert it into one that is many-time secure (i.e., existentially unforgeable against adaptive chosen message attacks).

This will be done analogously to the way we built a PRF from a PRG. In fact, this whole lecture might give you a deja vu, since we are basically repeating the PRF construction and proof in the context of signature schemes.

In this lecture, we construct a signature scheme with message space $\mathcal{M}_\lambda = \{0,1\}^\lambda$. Using the hash-then-sign paradigm we can convert it into a signature scheme with message space $\mathcal{M}_\lambda = \{0,1\}^*$, by first hashing the message $m \in \{0,1\}^*$ and then signing $H_{\mathsf{hk}}(m) \in \{0,1\}^\lambda$, where H_{hk} is a collision resistant hash function.

From One-Time Security to Many-Time Security

The idea is simple! To sign many messages lets generate many secret keys, and sign each message using a different key! The problem is that Gen is a PPT algorithm, and hence can generate at most polynomially many keys. Therefore, the adversary can query the signing oracle with polynomially many messages where this number is larger than the number of keys, and hence will use at least one key more than once, which may cause a breach in security.

Idea 1: Generate these secret keys on the fly, as many as needed!

A Stateful signature scheme: From one-time to many-time security via a chain of keys

As a warmup, let us first construct a "stateful" signature scheme. The idea is to evolve the keys as we go along. We start with a key pair $(\mathsf{vk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$. Then to sign the i'th message we generate a pair of fresh keys $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ and use sk_{i-1} to sign the (long) message (m_i, vk_i). The verifier's algorithm will verify the signature of the i'th message w.r.t. the $(i-1)$'st verification key vk_{i-1}.

Note that the verifier and the signer need to keep track of the number of messages and verification keys that were signed. We can make the verifier stateless and keep only the signer stateful, at the price of making the signatures grow linearly with the number of messages that were signed. A signature of the i'th message m_i would consist of verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_i)$, messages $(m_1, \ldots, m_{i-1})$ and signatures $(\sigma_1, \ldots, \sigma_i)$. The verifier will accept this signature if and only if for every $j \in [i]$,

$$\mathsf{Ver}(\mathsf{vk}_{j-1}, (m_j, \mathsf{vk}_j), \sigma_j) = 1.$$

The drawback of this approach is that the signer must be stateful and the signatures grow linearly with the number of messages signed.

Remark. This should remind you of our first attempt to construct a stateful PRF via a line (as opposed to using a tree).

We next show how to shrink the signature length.

Idea 2: Shrink the signature length via a tree!

From one-time to many-time security via a tree of keys

So far, we showed how to use each secret key sk_i to sign one new verification key vk_{i+1}. Next, she will use each secret key to sign **two** new verification keys. This gives a tree of keys instead of a list of keys. Specifically, the signer will generate 2^λ key pairs $\{(\mathsf{vk}_x, \mathsf{sk}_x)\}_{x \in \{0,1\}^\lambda}$. To sign message $m \in \{0,1\}^\lambda$ he will use the signing key sk_m. The signature will consist of $(\mathsf{vk}_m, \mathsf{Sign}(\mathsf{sk}_m, m))$ and the authentication of vk_m, which is

$$\left\{(\mathsf{vk}_{m_1,\ldots,m_{i-1},0}, \mathsf{vk}_{m_1,\ldots,m_{i-1},1}), \mathsf{Sign}(\mathsf{vk}_{m_1,\ldots,m_{i-1}}, (\mathsf{vk}_{m_1,\ldots,m_{i-1},0}, \mathsf{vk}_{m_1,\ldots,m_{i-1},1}))\right\}_{i \in [\lambda]}$$

where the original verification key is $\mathsf{vk}_\emptyset$. This makes the verifier stateless and the signature length "short" (i.e. of fix polynomial length that does not grow with the number of messages signed).[2]

However, this comes with a price! The signer is now inefficient since he needs to store this exponential size tree!

[2] It actually grows logarithmically with the number of messages signed, but since we always sign less than 2^λ many messages we can think of the signature as being of fixed size (that grows only with λ.

Idea 3: Generate the tree of keys on the fly! Rather than generating all the 2^λ key pairs a priori, the signer will generate the keys it needs *on the fly*. This will allow the signer to run in polynomial time, but will require the signer to be stateful. Specifically, the signer will keep a set S of all the key pairs generated thus far.

Remark. If the signature scheme was randomized then it would also need to store the signatures of these verification keys. Recall that Lamport's one-time signature scheme has a deterministic signing algorithm. Actually, every signature scheme can be made to have a deterministic signing algorithm by using a PRF to generate the randomness used by the signing algorithm.

To sign a message m the signer needs to generate key pairs

$$\left\{\left(\mathsf{vk}_{m_1,\ldots,m_{i-1},b}, \mathsf{sk}_{m_1,\ldots,m_{i-1},b}\right)\right\}_{i\in[\lambda],b\in\{0,1\}}$$

along with signatures

$$\left\{\sigma_{m_1,\ldots,m_i} = \mathsf{Sign}(\mathsf{sk}_{m_1,\ldots,m_i}, (\mathsf{vk}_{m_1,\ldots,m_{i-1},0}, \mathsf{vk}_{m_1,\ldots,m_{i-1},1}))\right\}_{i\in[\lambda]}$$

Notice that he cannot generate all these key pairs at random since then he will sign many messages with the same $\mathsf{sk}_\emptyset$. Instead, he will generate only those key pairs that were not already generated and stored in S. This ensures that each secret key will be used to sign only one message!
It remains to make the signer stateless.

Idea 4: Generate the keys using a PRF. Rather than storing the key pairs, the signer will store a key to a PRF and will regenerate each key pair $(\mathsf{vk}_x, \mathsf{sk}_x)$ by running $\mathsf{Gen}(1^\lambda)$ with the same randomness $r_x = \mathsf{PRF}(k, x)$.

The Construction

Let $F : \mathcal{K}_\lambda \times \{0,1\}^{\leq\lambda} \to \{0,1\}^\lambda$ be a PRF, and let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ be a one-time signature scheme w.r.t. message space $\{0,1\}^*$. We define a signature scheme $(\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Ver}^*)$ for message space $\mathcal{M}_\lambda = \{0,1\}^\lambda$, as follows.

- $\mathsf{Gen}^*(1^\lambda)$**:**
  1. Sample a PRF key $k \leftarrow \mathcal{K}_\lambda$.
  2. Sample a key pair $(sk_\emptyset, \mathsf{vk}_\emptyset) \leftarrow \mathsf{Gen}(1^\lambda)$.
  3. Let $\mathsf{vk}^* = \mathsf{vk}_\emptyset$ and $\mathsf{sk}^* = (k, \mathsf{sk}_\emptyset)$.
  4. Output $(\mathsf{vk}^*, \mathsf{sk}^*)$.
- $\mathsf{Sign}^*(\mathsf{sk}^*, m)$:

1. Parse $\mathsf{sk}^* = (k, \mathsf{sk}_\emptyset)$.
2. Denote by $m = (m_1, \ldots, m_\lambda)$.
3. For every $i \in [\lambda]$ do the following:
   (a) For every $b \in \{0,1\}$, compute $r_{m_1,\ldots,m_{i-1},b} = F(k, (m_1, \ldots, m_{i-1}, b))$.
   (b) For every $b \in \{0,1\}$, sample $(\mathsf{sk}_{m_1,\ldots,m_{i-1},b}, \mathsf{vk}_{m_1,\ldots,m_{i-1},b}) \leftarrow \mathsf{Gen}(1^\lambda; r_{m_1,\ldots,m_{i-1},b})$.
   (c) Compute $\sigma_{m_1,\ldots,m_{i-1}} = \mathsf{Sign}(\mathsf{sk}_{m_1,\ldots,m_{i-1}}, (\mathsf{vk}_{m_1,\ldots,m_{i-1},0}, \mathsf{vk}_{m_1,\ldots,m_{i-1},1}))$.
4. Compute $\sigma = \mathsf{Sign}(\mathsf{sk}_{m_1,\ldots,m_\lambda}, m)$.
5. Output $\sigma^* = \left(\{(\mathsf{vk}_{m_1,\ldots,m_{i-1},0}, \mathsf{vk}_{m_1,\ldots,m_{i-1},1}), \sigma_{m_1,\ldots,m_{i-1}}\}_{i\in[\lambda]}, \sigma\right)$.

- $\mathsf{Ver}(\mathsf{vk}_\emptyset, m, \sigma^*)$:
  1. Parse $\sigma^* = \left(\{(\mathsf{vk}_{m_1,\ldots,m_{i-1},0}, \mathsf{vk}_{m_1,\ldots,m_{i-1},1}), \sigma_{m_1,\ldots,m_{i-1}}\}_{i\in[\lambda]}, \sigma\right)$.
  2. For every $i \in [\lambda]$ verify that
  $$\mathsf{Ver}(\mathsf{vk}_{m_1,\ldots,m_{i-1}}, (\mathsf{vk}_{m_1,\ldots,m_{i-1},0}, \mathsf{vk}_{m_1,\ldots,m_{i-1},1}), \sigma_{m_1,\ldots,m_{i-1}}) = 1.$$
  3. Verify that
  $$\mathsf{Ver}(\mathsf{vk}_m, m, \sigma) = 1$$
  4. If any of these checks fail output 0 and otherwise output 1.

Theorem 3. $(\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Ver}^*)$ *is existentially unforgeable against adaptive chosen message attacks.*

Proof. Suppose for the sake of contradiction that there exists a poly-size $\mathcal{A}$ and a non-negligible $\epsilon : \mathbb{N} \to [0,1]$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr[\mathcal{A}^{\mathsf{Sign}^*(\mathsf{sk}^*,\cdot)}(\mathsf{vk}^*) = (m^*, \sigma^*) \text{ s.t. } \mathsf{Ver}^*(\mathsf{vk}^*, m^*, \sigma^*) = 1 \ \wedge \ m^* \notin Q] \geq \epsilon(\lambda) \tag{1}$$

where Q denotes the set of all oracle calls that $\mathcal{A}$ makes to the oracle, and the probability is over $(\mathsf{vk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}^*(1^\lambda)$.

Denote by $\mathsf{vk}^* = \mathsf{vk}_\emptyset$ and $\mathsf{sk}^* = (k, \mathsf{sk}_\emptyset)$. Consider the inefficient oracle Sign^{**}, that is identical to Sign^* except that it replaces the PRF $F(k,\cdot)$ with a truly random function R.

Claim 1. There exists a negligible function $\mu : \mathbb{N} \to [0,1]$ such that for every $\lambda \in \mathbb{N}$,

$$\begin{aligned}&|\Pr[\mathcal{A}^{\mathsf{Sign}^{**}(\mathsf{sk}^*,\cdot)}(\mathsf{vk}^*) = (m^*, \sigma^*) \text{ s.t. } \mathsf{Ver}^*(\mathsf{vk}^*, m^*, \sigma^*) = 1 \ \wedge \ m^* \notin Q] - \\ &\Pr[\mathcal{A}^{\mathsf{Sign}^*(\mathsf{sk}^*,\cdot)}(\mathsf{vk}^*) = (m^*, \sigma^*) \text{ s.t. } \mathsf{Ver}^*(\mathsf{vk}^*, m^*, \sigma^*) = 1 \ \wedge \ m^* \notin Q]| \\ &\leq \mu(\lambda)\end{aligned}$$

The claim follows from the security of the underlying PRF F.[3] Claim 1 and Equation (1) implies that there exists a non-negligible $\delta : \mathbb{N} \to [0,1]$ such that for every $\lambda \in \mathbb{N}$

$$\Pr[\mathcal{A}^{\mathsf{Sign}^{**}(\mathsf{sk}^*,\cdot)}(\mathsf{vk}^*) = (m^*, \sigma^*) \text{ s.t. } \mathsf{Ver}^*(\mathsf{vk}^*, m^*, \sigma^*) = 1 \ \wedge \ m^* \notin Q] \geq \delta(\lambda)$$

[3] We leave the reduction as an exercise.

We next construct a poly-size $\mathcal{B}$ that uses $\mathcal{A}$ to break one-time security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$. Let $q = \mathrm{poly}(\lambda)$ be an upper bound on the number of oracle calls that $\mathcal{A}$ makes. Let $\ell = q \cdot 2\lambda + 1$. We actually construct a poly-size $\mathcal{B}$ that wins in the following game (which is slightly different than the one-time security game) with non-negligible probability:

1. For every $i \in [\ell]$ the challenger samples $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$, and sends $(\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$ to $\mathcal{B}$.
2. $\mathcal{B}$ is given oracle access to the q oracles $\mathsf{Sign}(\mathsf{sk}_1, \cdot), \ldots, \mathsf{Sign}(\mathsf{sk}_\ell, \cdot)$ and can send each oracle a single query message (adaptively chosen).
3. $\mathcal{B}$ outputs (m^*, σ^*).
4. $\mathcal{B}$ wins if and only if there exists $i \in [\ell]$ such that

$$\mathsf{Ver}(\mathsf{vk}_i, m^*, \sigma^*) = 1$$

   and m^* is not the oracle query that $\mathcal{B}$ sent to oracle $\mathsf{Sign}(\mathsf{sk}_i, \cdot)$.

We note that the existence of a poly-size $\mathcal{B}$ that wins in this game with non-negligible probability implies there there exists a poly-size $\mathcal{B}'$ that breaks the one-time security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$. At a high level $\mathcal{B}'$ on input vk does the following:

1. Choose at random $i \leftarrow [\ell]$ and set $\mathsf{vk}_i = \mathsf{vk}$.
2. For every $j \in [\ell] \setminus \{i\}$ sample $(\mathsf{vk}_j, \mathsf{sk}_j) \leftarrow \mathsf{Gen}(1^\lambda)$.
3. Emulate $\mathcal{B}$ while emulating its oracles $\mathsf{Sign}(\mathsf{sk}_j, \cdot)$ for every $j \in [\ell] \setminus \{i\}$ using $\{\mathsf{sk}_j\}_{[\ell]\setminus\{i\}}$, and emulating $\mathsf{Sign}(\mathsf{sk}_i, \cdot)$ using its own oracle.
4. Output the output of $\mathcal{B}$, denoted by (m^*, σ^*).

Note that if $\mathcal{B}$ wins in the game with probability $\epsilon(\lambda)$ then $\mathcal{B}'$ wins with probability $\epsilon(\lambda)/\ell$, since

$$\begin{aligned}
&\Pr[\mathcal{B}' \text{ wins}] \geq \\
&\Pr[\mathcal{B}' \text{ wins} |\ \mathcal{B} \text{ wins}] \cdot \Pr[\mathcal{B} \text{ wins}] = \\
&\Pr[\mathcal{B}' \text{ wins} |\ \mathcal{B} \text{ wins}] \cdot \epsilon(\lambda) = \\
&\frac{\epsilon(\lambda)}{\ell}
\end{aligned}$$

Thus, it remains to construct the adversary $\mathcal{B}$, that wins with non-negligible probability in the game described above, from the adversary $\mathcal{A}$.

The adversary $\mathcal{B}$, on input $(\mathsf{vk}_0, \mathsf{vk}_1, \ldots, \mathsf{vk}_{q\cdot 2\lambda})$, emulates $\mathcal{A}^{\mathsf{Sign}^{**}(\mathsf{sk}_0, \cdot)}(\mathsf{vk}_0)$ by emulating its oracle as follows:

1. Define $\mathsf{vk}_{\emptyset} = \mathsf{vk}_0$.
2. Every time $\mathcal{A}$ sends an oracle query m, generate $\{\mathsf{vk}_{m_1,\ldots,m_{i-1},b}\}_{i\in[\lambda],b\in\{0,1\}}$ as follows:

   For every $i \in [\lambda]$ and $b \in \{0,1\}$, if $\mathsf{vk}_{m_1,\ldots,m_{i-1},b}$ was not already defined, then define it to be the next unused key in the list $(\mathsf{vk}_1,\ldots,\mathsf{vk}_{q\cdot 2\lambda})$.

   Note that we have enough keys since we assume that $\mathcal{A}$ makes at most q calls to its signing oracle $\mathsf{Sign}^{**}(\mathsf{sk}_{\emptyset},\cdot)$.
3. Query the oracle $\mathsf{Sign}(\mathsf{sk}_{m_1,\ldots,m_{i-1}},\cdot)$ with message $(\mathsf{vk}_{m_1,\ldots,m_{i-1},0},\mathsf{vk}_{m_1,\ldots,m_{i-1},1})$ to obtain a signature $\sigma_{m_1,\ldots,m_{i-1}}$.
4. Generate a signature σ for the message m by querying $\mathsf{Sign}(\mathsf{sk}_m,\cdot)$ with query m.
5. Output

$$\sigma^* = \left(\left\{\mathsf{vk}_{m_1,\ldots,m_{i-1},0},\mathsf{vk}_{m_1,\ldots,m_{i-1},1}),\sigma_{m_1,\ldots,m_{i-1}}\right\}_{i\in[\lambda]},\sigma\right)$$

   Notice that we query each $\mathsf{Sign}(\mathsf{sk}_i,\cdot)$ only once.
6. When $\mathcal{A}$ generates an output (m^*,σ^*), parse

$$\sigma^* = \left(\left\{(\mathsf{vk}'_{m_1,\ldots,m_{i-1},0},\mathsf{vk}'_{m_1,\ldots,m_{i-1},1}),\sigma_{m_1,\ldots,m_{i-1}}\right\}_{i\in[\lambda]},\sigma\right).$$

7. Output (m^*,σ).

Notice that $\mathcal{B}$ emulates $\mathsf{Sign}^{**}(\mathsf{sk}_{\emptyset},\cdot)$ perfectly, and hence with probability $\delta(\lambda)$,

$$\mathsf{Ver}(\mathsf{vk}_{m^*},m^*,\sigma) = 1$$

where $\mathsf{vk}_{m^*} \in \{\mathsf{vk}_1,\ldots,\mathsf{vk}_{q\cdot 2\lambda}\}$, and $\mathsf{Sign}(\mathsf{sk}_{m^*},\cdot)$ was not previously queried. Thus, $\mathcal{B}$ also wins with probability $\delta(\lambda)$, as desired.

□

References

[1] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, Computer Science Laboratory, Menlo Park, CA, October 1979. Technical Report.