Lecture 17: Digital Signature Schemes

Instructor: Anna Lysyanskaya *Scribe: Apoorvaa Deshpande*

Today's Agenda

- Definition
- Lamport's construction

1 Definition

Definition 1. *A digital signature scheme for a message space M consists of PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *such that:*

1. **Correctness:** $\forall (vk, sk) \in \mathsf{KeyGen}(1^k)$, $\forall m \in M$ *and for all* $\sigma \in \mathsf{Sign}(sk, m)$,

$$\mathsf{Verify}(vk, m, \sigma) = \mathsf{Accept}$$

   The correctness notion is that of *perfect correctness* which does not allow room for any error in verification. This can be relaxed to allow the Verify algorithm to reject correct signatures with negligible probability.

2. **Security:** *For all PPT $\mathcal{A}$ $\exists$ negl. $\nu()$ such that*

$$\Pr[(vk, sk) \in \mathsf{KeyGen}(1^k)\ ;\ (Q, m', \sigma') \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(vk)\ :\ m' \notin Q \text{ and } \mathsf{Verify}(vk, m', \sigma') = \mathsf{Accept}] = \nu(k)$$

   Our adversary $\mathcal{A}$ has access to the signing oracle $\mathsf{Sign}(sk, \cdot)$ and can get signatures $\sigma_1, \sigma_2, \ldots, \sigma_n$ on his choice of messages $m_1, \ldots, m_n$. This list of message-signature pairs is outputted as Q. This cannot be tampered with and is fixed by $\mathcal{A}$'s queries.

We have a potential issue with the reduction here. Let's say $\mathcal{B}$ is using $\mathcal{A}$ to break something else, $\mathcal{B}$ is expected to answer the signing queries of $\mathcal{A}$ so that $\mathcal{A}$ can later produce a valid forgery. But if $\mathcal{B}$ is able to produce signatures himself, what would he learn from $\mathcal{A}$'s forgery? We have to design the reduction so that $\mathcal{B}$ can still learn something from $\mathcal{A}$ and use its output in a meaningful way. Let us look at a simple example which illustrates these ideas:

2 Lamport's one-time signature scheme:

Let f be a OWF and message space $M = \{0,1\}^n$

- $\mathsf{KeyGen}(1^k)$: The secret key sk is a table containing $2n$ random strings each of length k as follows:

x_0^1	x_0^2		$\cdots$		x_0^n
x_1^1	x_1^2		$\cdots$		x_1^n

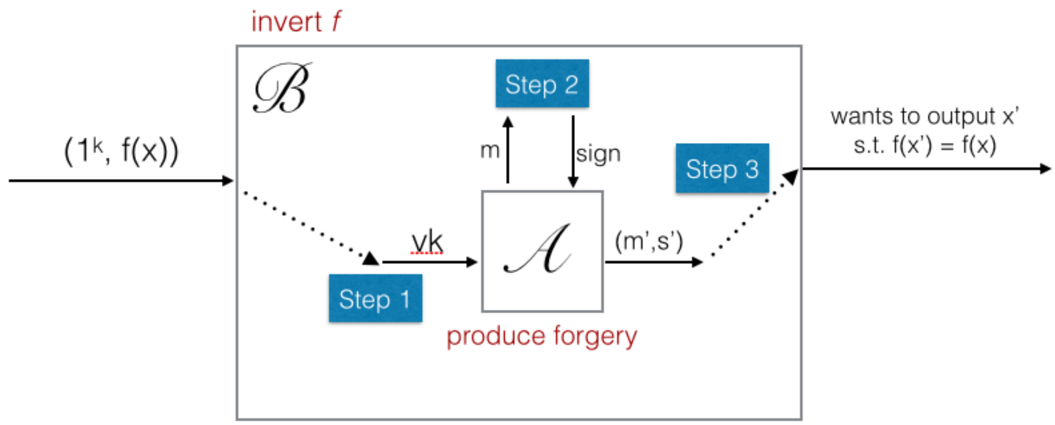
  Hence we have for $1 \le i \le n$, we have $x_b^i \leftarrow \{0,1\}^k$. Now let $y_b^i = f(x_b^i)$. Verification key vk is again a table with f applied to all strings in the secret key sk:

y_0^1	y_0^2		$\cdots$		y_0^n
y_1^1	y_1^2		$\cdots$		y_1^n

- $\mathsf{Sign}(sk, m)$: Suppose message $m = m_1 m_2 \cdots m_n$ for each $m_i \in \{0,1\}$. Reveal $x^i_{m_i}$ for $1 \leq i \leq n$ and signature $\sigma = x^1_{m_1}, x^2_{m_2}, \ldots, x^n_{m_n}$.
- $\mathsf{Verify}()$: Check that $f(x^i_{m_i}) = y^i_{m_i}$ for all i.

This construction cannot satisfy the security definition as it is, because the moment $\mathcal{A}$ has a signature on any message and its complement it knows the entire secret key. So we can allow $\mathcal{A}$ to make only one query and we will work with a weaker notion of security with a *Sign-once* oracle which answers only the first query of the adversary. And the security notion we have is *Security-once* where we use the *Sign-once* oracle instead of the usual oracle.

We can see that this signature scheme is correct. We will prove that it satisfies security-once via a reduction to OWF: If $\mathcal{A}$ can break Lamport's signature that is, if $\mathcal{A}$ can produce a valid forgery (m', σ') which verifies then $\mathcal{B}$ can use $\mathcal{A}$ to break the one-way function f. Our reduction will have the following three steps:



1. $\mathcal{B}$ receives as input $y = f(x)$ for some $x \in \{0,1\}^k$ and based on its input, it has to produce a verification key vk to give as input to $\mathcal{A}$
2. $\mathcal{B}$ is simulating the wild environment for $\mathcal{A}$ and has provided him the required vk. Additionally, $\mathcal{B}$ also needs to answer a signature query m that $\mathcal{A}$ makes and provide him the corresponding correct signature σ.
3. $\mathcal{B}$ now has to use $\mathcal{A}$'s forgery (m', σ') to output x' such that $f(x') = y = f(x)$

Let us look at each of these steps in more detail:

1. **Step 1:** $\mathcal{B}$ receives a y and chooses a random location (i, b_i) to put y in the table for vk. For the remaining $2n - 1$ entries of the table, $\mathcal{B}$ chooses x^j_b uniformly randomly from $\{0,1\}^k$ and corresponding $y^j_b = f(x^j_b)$ in vk except for $j = i$ and $b = b_i$ in which case we put y. We give this table of $2n$ values as the verification key to $\mathcal{A}$
2. **Step 2:** In this step, $\mathcal{B}$ has to produce a signature σ for $\mathcal{A}$'s query $m = m_1 \ldots m_n$. $\mathcal{B}$ can easily answer this query as long as $m_i \neq b_i$ since it knows the corresponding x values for all the remaining entries. Note that it is important for $\mathcal{B}$ to choose the location of y at random in step 1, otherwise $\mathcal{B}$ can catch $\mathcal{A}$ by querying exactly a message such that $\mathcal{A}$ is unable to answer

the signature query. So as long as $m_i \neq b_i$, $\mathcal{B}$ can answer $\mathcal{A}$'s query and give it corresponding x_b^j

3. **Step 3:** If forgery message is such that $m_i = b_i$ then output $x^i_{m_i}$ and we are guaranteed that if the forgery is valid then $f(x^i_{m_i}) = y$

Analysis:

$$\Pr[\mathcal{B} \text{ succeeds}] = \Pr[\mathcal{B} \text{ responds in step 2}]\Pr[\mathcal{B} \text{ succeeds } \mid \mathcal{B} \text{ responds in step 2}]$$

$$= \tfrac{1}{2}\Pr[m_i' = b_i]\Pr[\mathcal{B} \text{ succeeds } \mid m_i' = b_i]$$

$$= \tfrac{1}{2}.\tfrac{1}{n}.\epsilon(k)$$

We can generalize the above construction to signatures that are *secure-twice* or even generally secure by using Merkle hash trees.