Today’s Agenda

- Definition
- Lamport’s construction

1 Definition

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

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

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

   Our adversary $A$ has access to the signing oracle $\text{Sign}(sk, \cdot)$ 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 $A$’s queries.

We have a potential issue with the reduction here. Let’s say $B$ is using $A$ to break something else, $B$ is expected to answer the signing queries of $A$ so that $A$ can later produce a valid forgery. But if $B$ is able to produce signatures himself, what would he learn from $A$’s forgery? We have to design the reduction so that $B$ can still learn something from $A$ 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 $M = \{0,1\}^n$

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

  \[
  \begin{array}{cccc}
  x_0^1 & x_0^2 & \cdots & x_0^n \\
  x_1^1 & x_1^2 & \cdots & x_1^n \\
  \end{array}
  \]

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

  \[
  \begin{array}{cccc}
  y_0^1 & y_0^2 & \cdots & y_0^n \\
  y_1^1 & y_1^2 & \cdots & y_1^n \\
  \end{array}
  \]
• Sign\((sk, m)\): Suppose message \(m = m_1m_2 \ldots m_n\) for each \(m_i \in \{0, 1\}\). Reveal \(x'_{m_i}\) for \(1 \leq i \leq n\) and signature \(\sigma = x'_{m_1}x'_{m_2} \ldots x'_{m_n}\).

• Verify\(): Check that \(f(x'_{m_i}) = y'_{m_i}\) for all \(i\).

This construction cannot satisfy the security definition as it is, because the moment \(A\) has a signature on any message and its complement it knows the entire secret key. So we can allow \(A\) 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 \(A\) can break Lamport’s signature that is, if \(A\) can produce a valid forgery \((m', \sigma')\) which verifies then \(B\) can use \(A\) to break the one-way function \(f\). Our reduction will have the following three steps:

1. \(B\) 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 \(A\).

2. \(B\) is simulating the wild environment for \(A\) and has provided him the required \(vk\). Additionally, \(B\) also needs to answer a signature query \(m\) that \(A\) makes and provide him the corresponding correct signature \(\sigma\).

3. \(B\) now has to use \(A\)'s forgery \((m', \sigma')\) to output \(x'\) such that \(f(x') = y = f(x)\)

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

1. **Step 1:** \(B\) 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, \(B\) chooses \(x'_j\) uniformly randomly from \(\{0, 1\}^k\) and corresponding \(y'_j = f(x'_j)\) 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 \(A\).

2. **Step 2:** In this step, \(B\) has to produce a signature \(\sigma\) for \(A\)'s query \(m = m_1 \ldots m_n\). \(B\) 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 \(B\) to choose the location of \(y\) at random in step 1, otherwise \(B\) can catch \(A\) by querying exactly a message such that \(A\) is unable to answer
the signature query. So as long as $m_i \neq b_i$, $B$ can answer $A$’s query and give it corresponding $x^j_b$.

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:

$$
\Pr[B \text{ succeeds}] = \Pr[B \text{ responds in step 2}] \Pr[B \text{ succeeds} \mid B \text{ responds in step 2}]
= \frac{1}{2} \Pr[m'_i = b_i] \Pr[B \text{ succeeds} \mid m'_i = b_i]
= \frac{1}{2} \cdot \frac{1}{n} \cdot \epsilon(k)
$$

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