1 Recap

Last class we presented two different schemes for assuring message authentication. Particularly, we assumed that there exist two entities (Alice and Bob) who talk and they want to be sure that the messages they receive are authentic (i.e., Alice wants to be sure that the messages she reads comes from Bob, and viceversa).

We presented two different schemes to achieve this goal:

- **MAC scheme** (private key setting), briefly defined as $\pi = (\text{Gen}, \text{TAG}, \text{Ver})$:
  - $\text{Gen}(1^n) \rightarrow k$
  - $\text{TAG}(k, m) \rightarrow \sigma$
  - $\text{Ver}(m, \sigma, k) = \text{accepted/not accepted} (1/0)$
  - Correctness property: $\forall n, \forall m \in \{0, 1\}^l$
    \[ \Pr[k \leftarrow \text{Gen}(1^n), \text{Ver}(m, k, \text{TAG}(k, m)) = 1] = 1 \]

- **Signature scheme** (public key setting): briefly defined as $\pi = (\text{Gen}, \text{Sign}, \text{Ver})$:
  - $\text{Gen}(1^n) \rightarrow (s_k, p_k)$, where $s_k \neq p_k$
  - $\text{Sign}(s_k, m) \rightarrow \sigma$
  - $\text{Ver}(m, \sigma, p_k) = \text{accepted/not accepted} (1/0)$
  - Correctness property: $\forall n, \forall m \in \{0, 1\}^l$
    \[ \Pr[p_k, s_k \leftarrow \text{Gen}(1^n), \text{Ver}(m, p_k, \text{Sign}(s_k, m)) = 1] = 1 \]

But, while we showed a complete description of how to build the first scheme (MAC scheme), we built the second one (signature scheme) under two constraints: one time secure and for messages of fixed length.

The signature scheme one time secure for n-long messages ($\pi = (\text{Gen}, \text{Sign}, \text{Ver})$) was defined as follow (no proof of one time security is here provided):

- $\text{Gen}(1^n) \rightarrow (s_k, p_k)$

\[
s_k = \begin{pmatrix}
x_0^0 & \ldots & x_0^n \\
x_1^0 & \ldots & x_1^n \\
\vdots & \ddots & \vdots \\
x_n^0 & \ldots & x_n^n
\end{pmatrix}
\]

14-1
\[ p_k = \left\{ \begin{array}{c} y^0_1 \ldots y^0_n \\ y^1_1 \ldots y^1_n \end{array} \right\} \]

Where \( y^b_j = f(x^b_j) \) and \( f \) is a good one way function.

- **Sign(\( s_k, m \))**: a sequence of \( x^b_i \) is chosen (where \( b \) denotes up or down) according on the message bits. Particularly, if the \( i_{th} \) bit of \( m \) is 0 then \( x^0_i \) is used as the \( i_{th} \) component of the signature, otherwise \( x^1_i \). The final signature will be a concatenation of these \( x^b_i \). Therefore \( \sigma = x^b_1 \ldots|x^b_n = \sigma_1 \ldots|\sigma_n \).

- **Ver(\( p_k, m, \sigma \))**: if \( \forall i f(\sigma_i) = y^b_i \) (where \( y^b_i \) is \( y^0_i \) if the \( i_{th} \) bit of \( m \) is 0, \( y^1_i \) otherwise) then outputs 1, otherwise 0. Note that here for coherence with the verification function we denoted \( x^b_i \) as \( \sigma_i \).

Ideally a signature scheme should be many times secure (we well see soon what this means) and a user should be able to sign messages of different lengths.

We also gave the definition of unforgeability.

**Definition 1** A signature scheme \( \pi \) is unforgeable if:

\( \forall n \) PPT adversary \( A \), \( \exists \epsilon(n) \) (negligible function) such that:

\[ \Pr[A \text{ wins in } Exp_n] \leq \epsilon(n) \]

Where \( Exp_n \) is defined as follow:

- **Gen(1^n)** \( \rightarrow (s_k, p_k) \), where \( s_k \neq p_k \)
- \( A \) knows only \( p_k \)
- \( A \) queries many times a box which takes messages in input, signs them and outputs the signatures. That is to say that the box gets \( m_i \) and outputs \( \text{Sign}(s_k, m_i) \).
- \( A \) outputs a couple \( (m', \sigma') \)

**A wins the experiment if:**

- **Ver(\( p_k, m', \sigma' \)) = 1**
- \( m' \neq m_i \), where a \( m_i \) is the \( i_{th} \) message presented to the box.

Note that a similar definition is provided also for a MAC scheme, the only difference is that a shared secret key is used instead of a couple secret/public key.

A signature scheme is many time secure if the above definition holds. That is to say if the adversary \( A \) wins with a probability less negligible no matter how many times he queries the box (as long as he provides in output a message \( m' \) he did not use before).

Finally we said that a signature scheme is one time secure if the above definition of unforgeability holds for a particular experiment \( Exp^1_n \), where \( A \) can ask only for one message (instead of many) to be signed and then he outputs the couple \( m', \sigma' \).

Today we are going to remove both of the constraints (one time secure and messages of fixed length) we applied to our definition of signature scheme last time. Note that from now one when the expression “scheme \( \pi \)” or merely “\( \pi \)” will be used, we will refer implicitly to a “signature scheme \( \pi \)” as defined earlier in this very section.
2 Domain extension

As we said in Section 1, in the last class we saw a signature scheme which treats messages of a fixed length $n$ and which is one time secure. In this section a signature scheme which remove the first constraint is presented. Also this scheme will be one time secure. Particularly, we are going to see how to build a scheme for signing messages $l$-bit long, where $l$ is a polynomial in $n$ ($l = l(n) \gg n$).

In order to achieve this goal, a new type of functions is defined: hash functions. Informally a hash function $h$ is a function which compresses its inputs:

$$|h(x)| \ll |x|$$

That is to say a function defined as $h: \{0, 1\}^{l(n)} \rightarrow \{0, 1\}^n$. The most important property of a good hash function is that it has to be collisions resistant. That is to say that it has be computationally hard to find two inputs $x_1$ and $x_2$ such that $x_1 \neq x_2$ and $h(x_1) = h(x_2)$. Note that collisions are inevitable since such functions map an input space on a smaller output space ($2^n \ll 2^{l(n)}$). Examples of hash functions are MD5, SHA1.

The idea is that I can use a hash function to transform a string $l$-bit long to one $n$-bit long (with $l \gg n$), and use the signature scheme $\pi$ we have seen in the last class (and reported in Section 1) to sign. In this approach what I would sign is not a message itself, but a hash of it. If the hash function used is collision resistant, it would be difficult to find two different messages whose signature is the same.

Intuitively, if such a signature scheme $\pi$ is one time secure, using the same scheme with this approach it should maintain the same property (as long as $h$ is a good hash function). In fact, assuming that there exists an adversary $A$ who knows $p_k$ and $h$, and who asks for one message to be signed ($\sigma \leftarrow Sign(s_k, h(m))$), he would be able to find a message $m'$ and a $\sigma'$ ($m' \neq m$) such that $Ver(p_k, m', \sigma') = 1$, only if:

- $m' \neq m$ and $h(m') = h(m)$.
- $m' \neq m$ and $h(m') \neq h(m)$ and $Ver(p_k, m', \sigma') = 1$.

In the first case the signature verification would succeed because such an adversary was able to find a collision in $h$ (in fact if $h(m) = h(m')$, $Sign(p_k, h(m)) = Sign(p_k, h(m'))$). But if this happens that would mean that $h$ is not collision resistant, therefore it would not be a good hash function.

In the second case instead, the adversary would directly forge the signature (note that in this case the two hashes are different), but this would violate the security of the signature scheme, violating than the one time security property of $\pi$ we proved last time.

There is another question, though. In fact, no single hash functions can be collision resistant against a nPPT adversary. In fact, we defined an adversary as a family of uniform PPT machines (i.e., $M = \{M_i\}_{i \in N}$). This means that an adversary is modeled as a collection of algorithms each
one of those treats different family of inputs (particularly the \(i_{th}\) algorithm will treat inputs long \(i\)). Therefore we have that the size of the \(i_{th}\) PPT algorithm will have a size polynomial in \(i\) (i.e., \(|M_i| = s(i)\) where \(s\) is a polynomial).

Particularly, since we know that hash functions have at least one collision (as we said earlier in this section), such an adversary may know a collision \((x_1, x_2)\) for a single hash function \(h\) and hard-code it its PPT algorithms. Every time he is required to print a collision for a hash function \(h\), he merely outputs the same collision \((x_1, x_2)\).

The solution is to use a collection of hash functions.

**Definition 2** A set of functions \(H = \{h_i : D_i \rightarrow R_i\}_{i \in I}\) is a family of collision resistant hash functions (CRH) if:

- Easy to sample: Gen\((1^n) \rightarrow i\), where \(i \in I\).
- \(|R_i| < |D_i| \forall i \in I\).
- Easy to evaluate: \(\exists\) PPT \(M\) such that given \(x\), \(i \in I\) computes \(M(x, i) = h_i(x)\).
- Collision resistant: \(\forall\text{nu-PPT adversary } A \exists \epsilon(n)\), such that \(Pr[\cdot] \leq \epsilon(n)\).

Note that, since each functions in the family may have different collisions, if an adversary knows a particular collision for a function \(h_j\), he is likely to not be able to use it as valid collision for a function \(h_i\) for \(i \neq j\). Therefore the functions composing the family have to be carefully selected. Moreover, the adversary cannot know all the possible collisions (one for each possible function in the family), he would not be a nu PPT adversary otherwise.

### 2.1 Candidate scheme

Let’s now present a candidate signature scheme for \(l\)-bit messages \((\pi' = (Gen', Sign', Ver'))\) using \(\pi = (Gen, Sign, Ver)\) (as defined in Section 1) and a family of collision resistant hash functions \(H\).

\(\pi'\) is defined as follow:

- **Gen’**:
  \[
  (p'_k, s'_k) \leftarrow Gen(1^n)
  h_i \leftarrow Gen_{CRH}(1^n), \text{ where } h_i : \{0,1\}^{l(n)} \rightarrow \{0,1\}^n, l(n) << n
  s_k = < s'_k, h_i >
  p_k = < p'_k, h_i >
  
- **Sign’\((s_k, m)\)**: \(\sigma \leftarrow Sign(s'_k, h_i(m))\)

- **Ver’\((p_k, m, \sigma)\)**: Ver\((p'_k, h_i(m), \sigma)\)
Theorem 1 if $\pi$ is one time secure (OTS) and $H$ is a family of collision resistant hash functions $\Rightarrow \pi'$ is a one time secure signature scheme.

We are going to prove this theorem by contraposition. Supposing $\exists$ nu-PPT adversary $A$ such that $A$ wins in $Exp^1_n$ (as showed in Section 1) against $\pi'$ with probability $\frac{1}{p(n)}$ (where $p$ is a polynomial) for infinite $n \Rightarrow \exists$ nu-PPT adversary $B_1$ which wins $Exp^1_n$ against $\pi$ with probability $\frac{1}{q_0(n)}$ or $\exists$ a nu-PPT adversary $B_0$ which finds collisions against $H$ with probability $\frac{1}{q_1(n)}$, where $q_0$ and $q_1$ are polynomials.

In fact, as we said previously in this very section, when $A$ wins $Exp^1_n$ against $\pi'$ if one of the following is true:

- case1: $m' \neq m$ and $h_i(m') = h_i(m)$.
- case2: $m' \neq m$ and $h_i(m') \neq h_i(m)$ and $Ver'(p'_k, m', \sigma') = 1$.

In order to demonstrate this theorem by contraposition we have to build the adversary $B_0$ (which exploits $A$) and separately $B_1$ (which still exploits $A$). Note that, in interacting with $\pi'$, $A$ wants to know also $h_i$, since it has to choose an input $m$ which belongs to the domain $D_i$ (see definition of collision resistant in a collection of hash functions in Definition 2). Therefore we have to feed him also with this info.

This said, let’s start in building $B_0$. This adversary takes a hash function from the family $H$ (chosen at random) and he finds a collision (i.e., he outputs $m$ and $m'$ such that $m \neq m'$ and $h_i(m) = h_i(m')$) by exploiting $A$.

The algorithm is the following:

- $B_0$: receives $h_i$ (where $h_i \leftarrow Gen_{CRH}(1^n)$)
- $B_0$: $(p'_k, s'_k) \leftarrow Gen(1^n)$
- $B_0$: interact with $A$, which asks a message $m$ to be signed, and he signed it for $A$: $m \leftarrow A(p'_k, h_i); \sigma \leftarrow Sign(s'_k, h(m))$
- $B_0$: feeds $A$ and gets back $m', \sigma'$
  $m', \sigma' \leftarrow A(p'_k, h_i, m, \sigma)$
- $B_0$: outputs $(m, m')$.

Note that $B_0$ feeds $A$ as he expects and that $Pr[B_0 \text{ wins}] = Pr[m \neq m', h_i(m) = h_i(m')]$. Let’s now build $B_1$. This adversary participates in a game similar to the one played by $A$ (i.e., $Exp^1_n$), but against the signature scheme $\pi$.

Specifically the experiment carried on by $B_1$ on $\pi$ is:

- $(p_k, s_k) \leftarrow Gen(1^n)$
• \( a \leftarrow B_{1,1}(p_k) \) //algorithm to define

• \((a', \sigma') \leftarrow B_{1,2}(p_k, a, \sigma) \) //algorithm to define

Adversary \( B_1 \) wins if \( a \neq a' \) and \( \text{Ver}(p_k, a', \sigma') = 1 \). Note also that \( B_1 \) only knows the public key \( p_k \).

Let’s now describe the two algorithms \( B_{1,1} \) and \( B_{1,2} \).

\( B_{1,1}(p_k) \):

• \( B_1 \): \( h_i \leftarrow \text{Gen}_{CRH}(1^n) \)

• \( B_1 \): interact with A, which asks a message \( m \) to be signed:
  \[ m \leftarrow A(p_k, h_i) \]

• \( B_1 \): \( a = h_i(m) \) //note that \( m \) required by A is long \( l(n) (> n) \) bits, since A plays with \( \pi' \).

\( B_{1,2}(p_k, a, \sigma) \):

• \( B_1 \): gets \( \sigma \leftarrow \text{Sign}(s_k, a) = \text{Sign}(s_k, h_i(m)) \)

• \( B_1 \): feeds A and gets back \( m' \) and \( \sigma' \)
  \[ m', \sigma' \leftarrow A(p_k, h_i, m, \sigma) \] //note that \( \sigma \) is exactly the signature of \( m \) under \( \pi' \)

• output \( m', \sigma' \)

Let’s analyze the two constructions (\( B_0 \) and \( B_1 \)). What we claim now is that: \( \forall n \) such that A wins in \( \text{Exp}_n^1 \) against \( \pi' \) with probability \( \frac{1}{p(n)} \) (with \( p \) a polynomial), there must exists a \( b \) (\( b \) is function of \( n \)) such that \( \text{Pr}\left[ B_b \text{ wins} \right] \geq \frac{1}{2p(n)} \). Where \( b \) is either 0 or 1, and where “\( B_b \text{ wins} \)” means that \( B_b \) wins in its own game (so \( B_0 \) is able to find a collision for \( h_i \) and \( B_1 \) is able to win \( \text{Exp}_n^1 \) against \( \pi \)).

Considering a fixed \( n \), we know that the premise of our claim is true (i.e., A wins \( \text{Exp}_n^1 \) against \( \pi' \)) either because of case1 or case2 (see 2.1).

Therefore, formally:

\[
\text{Pr}[A \text{ wins } \text{Exp}_n^1 \text{ against } \pi'] = \text{Pr}[\text{case1 or case2}] > \frac{1}{p(n)}
\]  \hspace{1cm} (2)

Since \( \text{Pr}[\text{case1 or case2}] < \text{Pr}[\text{case1}] + \text{Pr}[\text{case2}] \), follows that:

\[
\text{Pr}[\text{case1}] + \text{Pr}[\text{case2}] > \frac{1}{p(n)}
\]  \hspace{1cm} (3)

Therefore there must exist a \( i \) (\( i = 1 \) or 2) such that \( \text{Pr}[\text{case } i] > \frac{1}{2p(n)} \).
• if i = 1:
Looking at the construction of $B_0$:
\[
\Pr[B_0 \text{ wins}] = \Pr[m \neq m', h_i(m) = h_i(m')] = \Pr[\text{case 1}] > \frac{1}{2p(n)}
\]

• if i = 2:
Looking at the construction of $B_1$:
\[
\Pr[B_1 \text{ wins}] = \Pr[\text{A forges signature}] = \Pr[\text{case 2}] > \frac{1}{2p(n)}
\]

Therefore if A wins $Exp^1_n$ against $\pi'$ with probability $\frac{1}{p(n)}$ for infinitely many $n$ (or for simplicity $\forall n \in N$), it means that there must exist a $b$ ($b$ is either 0 or 1) such that $B_b$ wins for also infinitely many $n$ (or for simplicity for least half of the $n$ of A). Which would violate either the one time security of the signature scheme $\pi$ or the collision resistance of H.

3 Many-time secure signature

Let us now see how to remove the second constraint we posed last time (one time security). In this section no proof is provided, but only the construction.

The idea is to use a gigantic set of $s_k$ and $p_k$, one for each possible (hash) message to be signed (note that, however, what is being signed is always a string of length $n$)

Therefore:

\[
p'_k = p'_{k_1}, \ldots, p'_{k_m}, \ldots, p'_{k_n}
\]
\[
s'_k = s'_{k_1}, \ldots, s'_{k_m}, \ldots, s'_{k_n}
\]

Everytime a message $m$ has to signed, the corresponding $s'_m$ is used. Same reasoning with the signature verification and $p'_m$. Therefore, since all the keys are independently generated and different messages are signed with different keys, as long as an adversary A has to provide a couple $(m', \sigma')$ for a message $m'$ he has not ever seen (i.e., A has to forge a signature for a message $m'$), this scheme is considered secure. In fact the adversary has no information about the secret key used for a message he has never seen. The problem here is that storing such keys requires a lot of memory. It would be good to be able to generate the couple $s'_{k_m}$ and $p'_{k_m}$ only when required and store only a small piece of information. We are going to show now how this goal can be achieved.

Assuming we have a binary tree (as depicted in Figure 1) where each node signs its children. Note that the depth of the tree is $n$ and that messages are signed only using the key pairs at the leafs of such a binary tree.

Note also that only the couple of keys at the root are provided by a trust party. Therefore one cannot only send the key he used to sign a message $m$ (i.e., the key of the $m$ leaf) since it has not considered trusted.
Therefore, a signature for a message $m$ needs to contain all the sequence of public keys (together with their signatures) that there are along the path from the leaf used to sign the message $m$ to the binary tree root. The last public key in the sequence is the one of the root of the tree, which is trusted.

For example, the signature of the first message (the one which corresponds to the first leaf on left in Figure 1) will be: $\sigma = (p_k, \sigma_0, p_{k_0}, \ldots, \sigma_{0^n}, p_{k_0^n}, \text{Sign}(s_{k_0^n}, m))$.

Note also that for an arbitrary new message, a leaf that has never chosen before has to be used to sign it. That is to say the next unused leaf node in the tree is chosen. This is crucial for the security of this construction.

A person who wants to check the signature for a message has now all the necessary information to do that. In fact he can perform the signature check all over the chain. He will use the public key of the root to check the signature of the second public key (which was the one selected at the second level of the binary tree). If the verification succeeds, he uses this second public key to check the signature for the third public key (which was the one selected at the third level of the binary tree), and so forth until he reaches the signature of the message $m$.

What we want now is an algorithm to generate the $i_{th}$ key pairs (we do not want to keep the whole the binary tree in memory). A way to do this is using a random number $k$, and for each node $i$ in the binary tree, we generate its couple of keys as follows:

$$(s_{k_i}, p_{k_i}) \leftarrow \text{Gen}(1^n, r_i)$$

$$r_i = \text{PRF}(k, i)$$

Figure 1: Binary tree of keys
Where PRF is a good pseudo-random function. In this way the only information we have to keep in memory is the couple \((s_k, p_k)\), and the only piece of information a person need to verify a signature is \(p_k\).