As a continuation of the last lecture, we will study more about public-key cryptography, focusing our attention to two topics, signature schemes and collision resistant hash functions.

1. **Signature Scheme**

   (a) Three algorithmic components of the signature scheme

   i. *KeyGen* : generates public key, which is used for verifying signature, and secrete key which is used for signing.
   
   ii. *Sign*(SK, m) = σ
   
   iii. *Verify*(PK, m, σ) = yes/no

   (b) Basic correctness requirements of the signature scheme

   Any signature σ generated by the signing algorithm *Sign* is accepted as valid by the verification algorithm *Verify*.

   (c) Definition of security

   i. Intuitive definition

   - Intuitively, security means that it is hard to "forge" a signature, where "forgery" refers to a valid signature on a message that was not signed.

   ii. Adversarial goal

   - total break : An adversary is able to create a valid signature for any message.
   - selective forgery : An adversary is able to create a valid signature for a given message chosen by someone else.
   - existential forgery : An adversary is able to create a valid signature for at least one message.

   iii. Attack

   - key-only attack : An adversary only knows the public key of the signer.
   - known message attack : An adversary knows a list of messages and their corresponding signatures.
   - chosen message attack : An adversary is able to request signatures for messages he selects.

   iv. Strongest notion of security

   - Security against an existential forgery with respect to adaptive chosen message.
v. Formal definition

\[< \text{adversary} > \quad < \text{challenger} > \]

\[
\begin{align*}
&PK \leftarrow (PK, SK) \leftarrow R \text{KeyGen} \\
&m_1 \rightarrow \\
&\sigma_1 \leftarrow Sign(SK, m_1) \\
&\vdots \\
&m_t \rightarrow \\
&\sigma_t \leftarrow Sign(SK, m_t)
\end{align*}
\]

Adversary’s output is \((m, \sigma)\), and it is a forgery if \(m\) is not contained in \(\{m_1, m_2, ..., m_t\}\) and \(\text{Verify}(PK, m, \sigma) = yes\). Security means that for all efficient adversaries, the probability that output is a forgery is negligible.

2. Collision Resistant Hash Function

(a) In general, hash function takes a lengthy input and gives an output with shorter length. Then, the family of hash function is defined as

\[
\mathcal{H} = \{H_k\}_{k \in K}
\]

and the hash function is defined as

\[
H_k = \{0, 1\}^{l'} \rightarrow \{0, 1\}^{l}
\]

where \(l'\) is much larger than \(l\).

(b) Universal hash function vs. collision resistant hash function

i. \(\epsilon\)-universal

\[
\forall x, x' \in \{0, 1\}^{l'} \text{ such that } x \neq x', \Pr[k \leftarrow R K : H_k(x) = H_k(x')] \leq \epsilon
\]

ii. Collision resistance

For all efficient algorithms \(A\),

\[
Pr[k \leftarrow R K, (x, x') \leftarrow A(k) : x \neq x' \land H_k(x) = H_k(x')] = \text{negligible}
\]

Clearly, the notion of the collision resistant hash function is much stronger that that of the universal hash function.
(c) Concrete example of collision resistant hash function

Let \( p = 2q + 1 \), where \( p \) and \( q \) are primes, and \( \gamma_1, \gamma_2 \) be random elements of order \( q \) in \( \mathbb{Z}_p^* \). The key, \( k \) for a hash function is consist of \( p, q, \gamma_1, \gamma_2 \) and the map is \( (x_1, x_2) \mapsto \gamma_1^{x_1} \gamma_2^{x_2} \). Input to the hash function is \( \{0, 1, \ldots, q - 1\} \times \{0, 1, \ldots, q - 1\} \), and the output is the element of \( \mathbb{Z}_p^* \). Then it is claimed that this function is collision resistant, assuming discrete logarithm is hard.

**Proof.** We will show that we can compute discrete logarithm assuming an adversary who can break the scheme. Let us suppose that \( A \) finds a collision with probability \( \epsilon \). For \( (x_1, x_2) \neq (x_1', x_2') \),

\[
\begin{align*}
\gamma_1^{x_1} \gamma_2^{x_2} &= \gamma_1^{x_1'} \gamma_2^{x_2'} \\
\gamma_1^{x_1 - x_1'} \gamma_2^{x_2 - x_2'} &= 1 \\
\gamma_1 &= \gamma_2^{[\frac{(x_2 - x_2')}{(x_1 - x_1')}] \mod q} \\
\log_{\gamma_2} \gamma_1 &= \left[\frac{(x_2' - x_2)}{(x_1' - x_1')}\right] \mod q
\end{align*}
\]

So, we can find discrete logarithm of \( \gamma_1 \) to the base of \( \gamma_2 \) by computing the right hand side of the last equation with \( A \).

Therefore, if discrete logarithm is hard, then the function is collision resistant. \( \square \)

(d) Application of collision resistant hash function

The following construction, using collision resistant hashing, allows one to sign long messages with any signature scheme. Suppose \( \mathcal{H} = \{H_k\}_{k \in K} \), where \( H_k : \{0, 1\}^l \rightarrow \{0, 1\}^l \) is collision resistant. Also, suppose \( \sum = \{\text{KeyGen}, \text{Sign}, \text{Verify}\} \) is a secure signature scheme with message space \( \{0, 1\}^l \). Then, we can construct a secure signature scheme for message space \( \{0, 1\}^l' \) as follows. Let us define \( \sum' = \{\text{KeyGen}', \text{Sign}', \text{Verify}'\} \), where

\[
\begin{align*}
\text{KeyGen}' : & \quad (PK, SK) \leftarrow \text{KeyGen}() \\
& \quad k \leftarrow_{\text{R}} K \\
& \quad \text{output } PK' = \{PK, k\}, \ SK' = \{SK, k\} \\
\text{Sign}'(SK', m) &= \text{Sign}(SK, H_k(m)) \\
\text{Verify}'(PK', m, \sigma) &= \text{Verify}(PK, H_k(m), \sigma)
\end{align*}
\]

**Proof.** We will not go over the full proof, but only the sketch. The strategy of the proof is that given a forger for the new scheme \( \sum' \), we will construct a collision finder for \( \{H_k\} \) and an adversary that breaks the original scheme \( \sum \). One of the two will work with probability that is not negligible, which leads to a contradiction. \( \square \)
(e) Collision resistant hash functions in practice

Let us consider more realistic examples of collision resistant hash functions such as "compression function" and chaining.

- Compression function

Compression function is a collision resistant hash function from 512bits + 160bits to 160bits. The below is the diagram of the block cipher representing the compression function.

A different diagram such as given below can represent the compression function to aid the explanation of the concept of chaining.

- Merkle-Damgaard chaining

It breaks up the input into 512 bit blocks by some means of padding scheme. Then, encode the total length as a bit field (as a 64 bit number) and place in the last block. Then the chaining is used to combine the compression functions as below (IV is the initial vector and it is fixed).
Now, we can present a theorem that combine the ideas of compression function and chained function.

**Theorem 1.** If the compression function is collision resistant, then so is the chained function.

**Proof.** Suppose the adversary breaks the chained function which means that there’s a collision in the compression function. The idea is to convert an adversary, $A$ that finds collisions in the chained function to one, $A'$ that finds collisions in compression functions.

Then, $A'$ works as follows: Let $A$ run. If $A$ finds a collision. Then, two things can happen:

- **Case 1:** $n \neq m$
  
  $n \neq m$ when the lengths encoded are different. In other words, the input to the last stage differ, and that is our collision.

- **Case 2:** $n = m$
  
  Walk backward on the chain and you will find a collision. To elaborate, starting from the last block, if the inputs to the last blocks differ, then there’s a collision, otherwise, go back one block. If the inputs to that blocks differ, then there’s a collision, otherwise go back another block. Keep going until you find a collision. If you ended up at the first block without finding a collision, since $B_2, B_3, ..., B_n$ and $C_2, C_3, ..., C_n$ are the same and $IV$ is fixed, $B_1$ has to differ from $C_1$, and there’s the collision.

$\square$

3. **Construction of a Secure Signature Scheme**

There are two components in constructing a secure signature scheme.
(a) One-way function: \( f : \{0, 1\}^{l_1} \rightarrow \{0, 1\}^{l_2} \)
(b) Collision resistant family: \( \mathcal{H} = \{ H_k \}_{k \in K} \)

Let’s consider one time signature scheme. One time signatures allow you to sign only one message. In other words, the adversary is allowed only one oracle query rather than polynomially many. Let’s first restrict ourselves to just single bit message and build up toward a \( n \)-bit message.

- 1-bit messages

Here is the signature scheme:

\[
\text{KeyGen:} \quad \begin{align*}
x_0 &\leftarrow_R \{0, 1\}^{l_1} \\
x_1 &\leftarrow_R \{0, 1\}^{l_1} \\
y_0 &\leftarrow f(x_0) \\
y_1 &\leftarrow f(x_1)
\end{align*}
\]
\[
PK = (y_0, y_1) \\
SK = (x_0, x_1)
\]

The signature of 0 is \( x_0 \), and that of 1 is \( x_1 \). Then, verify that \( x \) is a signature on \( b \in \{0, 1\} \) by \( f(x) = y_b \). Now it is claimed that this one time signature scheme for 1-bit messages is secure.

**Proof.** Suppose it is not secure. Then out of the adversary \( A \), we will build an inversion algorithm \( B \) for the one way function \( f \). Indeed, the inversion algorithm plays the role of the challenger in the signature game. On input \( y \), \( B \) runs as follows:

\[
b \leftarrow_R \{0, 1\} \\
\text{If } b = 0, \text{ then} \quad \begin{align*}
y_0 &\leftarrow y \\
x_1 &\leftarrow_R \{0, 1\}^{l_1} \\
y_1 &\leftarrow f(x_1)
\end{align*}
\]
\[
\text{If } b = 1, \text{ then} \quad \begin{align*}
y_1 &\leftarrow y \\
x_0 &\leftarrow_R \{0, 1\}^{l_1} \\
y_0 &\leftarrow f(x_0)
\end{align*}
\]

run \( A \) on the \( PK = (y_0, y_1) \)

If \( A \) asks for a signature on \( \overline{b} \), \( B \) returns \( x_{\overline{b}} \) to \( A \)
else, \( B \) aborts

L13-6
If $B$ did not abort and $A$ outputs a forgery on $b$ (because $A$ is required to forge on a new message), $B$ will learn a pre-image of $y$ under $f$. Note that if the adversary $A$ successfully forges with probability $\epsilon$ then our inversion algorithm $B$ has success probability $\epsilon/2$, because the view of $A$ does not depend on $B$’s choice of $b$. □

- $n$-bit messages

It is easy to extend the above scheme for $n$-bit messages as follows: select $2^n$ values $x_{10}, x_{20}, \ldots, x_{n0}, x_{11}, x_{21}, \ldots, x_{n1}$ for the secret key, and apply $f$ to each to get $y_{10}, y_{20}, \ldots, y_{n0}, y_{11}, y_{21}, \ldots, y_{n1}$ for the public key. To sign message $b_1 b_2 \ldots b_n$, where $b_i$ is the $i$-th bit, use $n$ 1-bit signing keys and sign the bits individually. The signature on $b_1 b_2 \ldots b_n$ is $x_{1_{b_1}} x_{2_{b_2}} \ldots x_{n_{b_n}}$. To verify $z_1, z_2, \ldots, z_n$ is a signature on $b_1 b_2 \ldots b_n$, check if for each $i$, $1 \leq i \leq m$, $f(z_i) = y_{ib_i}$. This scheme is secure if it is used to sign a single $n$-bit message.

**Proof.** The proof is the same as for the previous claim, except that $B$ has to guess not only the bit $b$, but also the message position $i$. $B$ then generates the secret key by selecting $x_{10}, x_{20}, \ldots, x_{n0}, x_{11}, x_{21}, \ldots, x_{n1}$, computes the public key $y_{10}, y_{20}, \ldots, y_{n0}, y_{11}, y_{21}, \ldots, y_{n1}$, and substitutes its input $y$ in place of $y_{ib_i}$. $B$ will succeed in finding inverse for $y$ if it can answer the query of $A$ (i.e., if the $i$-th bit of the query message is $\overline{b}$), and if the $i$-th bit of the forgery message is $b$. To compute the probability of $B$’s success, let’s consider the following games: The adversary $A$ asks for a signature on $b_1 b_2 \ldots b_n$ and produces a forgery on $b'_1 b'_2 \ldots b'_n$. Let $S_i$ be the event that $A$ successfully forge in game $i$.

- Game 0: It is the original attack game.
- Game 1: It is the same as game 0, but choose $i$ at random from $\{1, 2, \ldots, n\}$. If the forged message does not differ at position $i$, then do not let the adversary output anything. Then, $Pr[S_1] = \frac{1}{n} Pr[S_0]$.
- Game 2: Choose $b \leftarrow \{0, 1\}$. Let insist that $b_i = \overline{b}$ and $b'_i = b$. Then, $Pr[S_2] = \frac{1}{2} Pr[S_1]$.

Thus, if the adversary $A$ successfully forges with probability $\epsilon$ then success probability of $B$ is at least $\epsilon/(2n)$. □