University of Illinois, Urbana Champaign
CS/ECE 498AC3/4 Applied Cryptography

Instructor: Dakshita Khurana
Scribe: Jingyu Qian
Date: October 1, 2020

LECTURE
12

Signatures

In the last lecture, we have discussed how to perform authenticated key exchange when the adversary has the complete control of the network (i.e., the adversary can modify, inject, or delete packets). We show that with the help of a trusted third party and the scheme of certificate, an adversary cannot impersonate one communication party without being detected. We end up with introducing the important component of setting up the trusted third party, which is the signature scheme.

In this lecture, we discuss in more detail about digital signature. We define one-time and many-time security for the signature scheme by using the adversarial game. In addition, we show several real-world applications of digital signatures. Besides, we discuss how to apply trapdoor permutations and argue about the security of the naive application. Finally, we introduce the full domain hash and prove its security.

12.1 Digital Signatures

Digital signature is the public-key version of "MACs". The goal is to prevent the adversary from tampering the message without being detected.

Definition 12.1. A digital signature scheme Π includes three probabilistic polynomial-time (i.e., PPT) algorithms $(Gen, Sign, Verify)$ such that:

1. Gen is the key generation algorithm that takes as input a security parameter 1^k and outputs a pair of keys (vk, sk), where vk is the public verification key and sk is the secret signing key.
$$Gen(1^k; r) = (vk, sk).$$

2. $Sign$ is the signing function that takes as input the secret signing key sk and a message m from the message space M. It then outputs a signature σ.
$$\sigma \leftarrow Sign(m, sk; r).$$

3. $Verify$ is the verification function that takes as input a message m, a signature σ, and the public verification key vk. It then outputs a bit b, with $b = 1$ meaning accepting

and $b = 0$ meaning rejecting.

$$b = Verify(m, \sigma, vk).$$

A correct digital signature scheme ensures that $\forall m, Verify(m, Sign(m, sk; r), vk) = 1$. Intuitively, this means that if one signs the message correctly, then the verification function must accept.

12.2 One-Time Secure Signature

To formally define the security of a signature scheme, we start with a weaker one, which is the one-time signature security. To understand how we define one-time signature security, we first define the following one-time signature game.

DEFINITION 12.2. For a signature scheme $\Pi = (Gen, Sign, Verify)$, the one-time signature game $\texttt{Sig-game}^{one-time}_{A,C,\Pi}\texttt{(k)}$ is defined using a challenger C and an adversary A. In this game:

1. C uses $Gen(1^k; r)$ to generate a pair of keys (vk, sk),
2. C sends the key vk to A,
3. A sends one message $m_1 \in M$ to C,
4. C generates the signature $\sigma_1 = Sign(m_1, sk)$,
5. A outputs a message-signature pair (m, σ) such that $m \neq m_1$,
6. The output of the game is 1 if $Verify(m, \sigma, vk) = 1$ and otherwise 0.

Intuitively, for one-time signature security, we allow the adversary to query the challenger on a single message. The adversary wins if she can output a new message-signature pair (m, σ) such that $Verify(m, \sigma, vk) = 1$ and $m \neq m_1$. The formal definition of one-time signature security is given as follows.

DEFINITION 12.3. A signature scheme $\Pi = (Gen, Sign, Verify)$ is a one-time secure signature scheme, if for all PPT adversaries A, there exists a negligible function $\epsilon(k)$ such that:

$$Pr[\texttt{Sig-game}^{one-time}_{A,C,\Pi}\texttt{(k)} = 1] \leq \epsilon(k).$$

A slightly stronger one-time signature security property only requires $(m, \sigma) \neq (m_1, \sigma_1)$. In this strong one-time signature security property, we allow $m = m_1$, which means that if the signature scheme is one-time secure, the adversary cannot even come up with a new signature on the same message. This stronger version of the security property has its critical application in blockchain, where we need to ensure nobody without the knowledge of the secret signing key can sign the same message correctly.

12.3 Lamport One-Time Signature Scheme

Here we discuss how one can construct a one-time secure signature scheme. We focus on the Lamport one-time signature scheme. The construction of the Lamport one-time signature scheme relies on a one-way function f. The Lamport signature scheme is defined as follows.

DEFINITION 12.4. Let f be a one-way function. The Lamport one-time signature scheme $\Pi_{Lamport} = (Gen, Sign, Verify)$ is constructed by:

1. *Gen* takes input as 1^k. It first produces the secret signing key

$$sk = \begin{pmatrix} r_{1,0} & r_{2,0} & \dots & r_{n,0} \\ r_{1,1} & r_{2,1} & \dots & r_{n,1} \end{pmatrix},$$

   where $r_{i,0}, r_{i,1}$ are chosen uniformly from $\{0,1\}^k$. Then, it produces the public verification key

$$vk = \begin{pmatrix} y_{1,0} & y_{2,0} & \dots & y_{n,0} \\ y_{1,1} & y_{2,1} & \dots & y_{n,1} \end{pmatrix},$$

   such that $y_{i,0} = f(r_{i,0})$ and $y_{i,1} = f(r_{i,1})$.

2. $Sign$ takes as input a message m of length n where $m = m_1 \dots m_n$, and the secret signing key as above, and outputs the signature $(r_{1,m_1}, r_{2,m_2}, \dots, r_{n,m_n})$.

3. $Verify$ takes as input a message m of length n where $m = m_1 \dots m_n$, a signature $\sigma = (r_1, r_2, \dots, r_n)$, and a public verification key vk as above. It outputs 1 if and only if $\forall 1 \le i \le n : f(r_i) = y_{i,m_i}$.

Now let's prove that the Lamport one-time signature scheme is indeed a one-time secure signature scheme. Here we only use the weaker version of the one-time signature game $\texttt{Sig-game}^{one-time}_{A,C,\Pi_{Lamport}}(\texttt{k})$, which requires that the adversarial can produce a valid message-signature pair (m, σ) such that $m \neq m_1$. For the following proof, we replace m_1 with m' for clean symbols.

Proof. Assume there exists an adversary that can produce a message-signature pair (m, σ) such that $m \neq m'$ and $Verify(m, \sigma, vk) = 1$. We write m in the form of $m_1 m_2 \dots m_n$ and m' in the form of $m'_1 m'_2 \dots m'_n$. Because $m \neq m'$,

$$\exists i \in [1, n] : m_i \neq m'_i.$$

This implies that the adversary can invert the one-way function f to find r_i where $f(r_i) = y_{i,m_i}$. This violates the definition of the one-way function. ∎

Note that by only using the one-way function, it does not ensure the strong one-time signature security. In the strong one-time signature game, the adversary can find a message-signature pair (m, σ') where $m = m_1$, and $\sigma' \neq \sigma_1$. We write $\sigma' = r'_{0,m_0} \dots r'_{n,m_n}$. In this case, given the knowledge of $r_{i,m_{1_i}}$ and $y_{i,m_{1_i}} = f(r_{i,m_{1_i}})$, the adversary just needs to find $r'_{i,m_{1_i}}$ such that $y_{i,m_{1_i}} = f(r'_{i,m_{1_i}})$. This is possible because the one-way function only guarantees that given the knowledge of $y_{i,m_{1_i}}$, the adversary can not find r such that $f(r) = y_{i,m_{1_i}}$. However, the adversary now has the extra knowledge of $r_{i,m_{1_i}}$ where $y_{i,m_{1_i}} = f(r_{i,m_{1_i}})$. It is then natural to see that if we use a one-way permutation as the one-way function f, we will also ensure the strong one-time signature security.

12.4 Many-Time Secure Signature

Here we define the many-time secure signature property, which is stronger than the one-time secure signature property. Similarly, we could define the many-time signature game as follows.

DEFINITION 12.5. For a signature scheme $\Pi = (Gen, Sign, Verify)$, the many-time signature game $\texttt{Sig-game}^{q-time}_{A,C,\Pi}\texttt{(k)}$ is defined using a challenger C and an adversary A. In this game:

1. C uses $Gen(1^k; r)$ to generate a pair of keys (vk, sk),
2. C sends the key vk to A,
3. A sends q messages $m_1, \ldots, m_q \in M$ to C,
4. C generates q signatures $\sigma_1 = Sign(m_1, sk), \ldots, \sigma_q = Sign(m_q, sk)$,
5. A outputs a message-signature pair (m, σ) such that $m \notin \{m_1, \ldots, m_q\}$,
6. The output of the game is 1 if $Verify(m, \sigma, vk) = 1$ and otherwise 0.

A stronger alternative game is to allow the adversary to query a message, achieve its signature, and repeat this steps for q times. We say the adversary wins the game if she can output a new message-signature pair (m, σ) such that $Verify(m, \sigma, vk) = 1$ and $m \notin \{m_1, \ldots, m_q\}$, i.e., $\texttt{Sig-game}^{q-time}_{A,C,\Pi}\texttt{(k)}$ outputs 1. Formally, we have:

DEFINITION 12.6. A signature scheme $\Pi = (Gen, Sign, Verify)$ is a many-time secure signature scheme, if for all PPT adversaries A and for any number of queries q, there exists a negligible function $\epsilon(k)$ such that:

$$Pr[\texttt{Sig-game}^{q-time}_{A,C,\Pi}\texttt{(k)} = 1] \leq \epsilon(k).$$

Also, we have the strong many-time secure signature property, where we only require the adversary to output the message-signature pair (m, σ) such that $(m, \sigma) \notin \{(m_1, \sigma_1), \ldots (m_q, \sigma_q)\}$.

Since we have defined the many-time secure signature property, we could also easily see that Lamport one-time signature scheme is not many-time secure. In fact, it is not even two-time secure. The adversary could send $m_1 = 00\ldots0$ and $m_2 = 11\ldots1$ to the challenger. Then the adversary simply achieves the secret signing key sk from the response of the challenger where $\sigma_1 = (r_{1,0}, r_{2,0}, \ldots, r_{n,0})$ and $\sigma_2 = (r_{1,1}, r_{2,1}, \ldots, r_{n,1})$

12.5 Applications

In the previous sections, we have introduced the definition of one-time and many-time secure signature, and also illustrated the Lamport one-time signature. In this section, we will see several useful real-world application of signatures.

The first application is authenticating software updates. Initially, the server can share the verification key to the client. Later, every time the server wants to instruct the software update to the client, it will send the update message which is signed with the signing key. Then, the client accepts an update only if the verification accepts the signed update message.

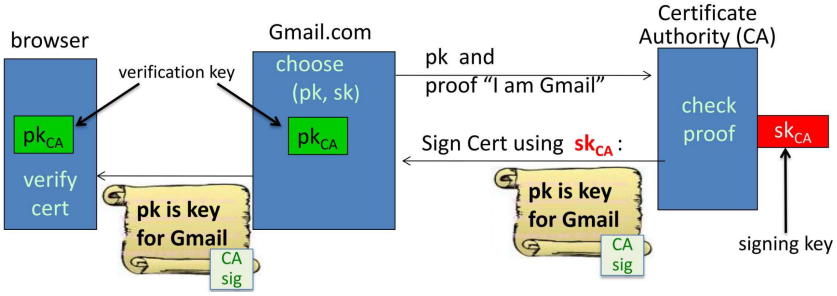

FIGURE 12.1: Certifcates.

The second application is converting a one-time authentication channel into a many-time authenticated channel. For instance, assume in the very first communication between Alice and Bob, Alice let Bob know that she is indeed Alice, and Alice provides Bob with her verification key. This could be done through a one-time authenticated channel. After this step, whenever Alice wants to send a message to Bob, she just needs to sign it, and Bob only accepts the message if the verification accepts the signature.

The third application is the domain key identified email (i.e., DKIM). The problem that DKIM tries to solve is that the bad email can claim to come from a legit domain. For instance, the bad email could claim to be from someuser@gmail.com, which in reality, is coming from domain badguy.com. Then, the user may think gmail.com is the bad source of email. To solve this problem, DKIM ensures that every email send via gmail.com is signed with the signing key of gmail.com. The recipient can then check that the email which claims to come from gmail.com is indeed signed with the corresponding signing key of gmail.com.

The fourth application is certificates, which has been discussed in the last lecture as a motivation for the signature scheme (Figure 12.1). Here, the browser needs the server's public key to set up a session key. The server cannot directly send the public key to the browser because the adversary who has the full control of the network can impersonate the server and send the adversary's public key. Thus, the server relies on a trusted third party to certify its public key. This third party is also known as the certificate authority (i.e., CA). CA has its public verification key. This verification key could be revealed to every user through a one-time authenticated channel. Thus, the attacker cannot tamper CA's verification key without being caught because it is public knowledge.

The server can give the public key (pk) and the proof of the server's true identity to CA. This proof can be a strict physical evidence (e.g., photograph, SSN, ...). After checking that pk is indeed the server's public key, CA will sign with its signing key a certificate on the message that claims pk is the server's public key. In this way, no adversary who sees the certificate across the network can produce a new signature unless they also contact CA and prove who they are.

Now, whenever a browser contacts the server, the server can send the certificate to the browser. Then, the browser can verify the signature by using CA's verification key. If the signature is accepted, the browser knows the true public key of the server.

12.6 MACs v.s. Signatures

The use case of MAC and signatures are different. MACs should be used when the signer and the verifier are the same entity or when there is a single signer and a single verifier. In both cases, public keys that could be used by many other entities are not necessary. On the other hand, signatures should be used when there are many verifiers for the same signer or when public verification is required.

As an example of different use cases, MACs cannot be used for authenticating software updates, because for verifying in MACs, the adversary needs to know the key. Then with the knowledge of the key, the adversary can also produce MACs correctly. On the other hand, signatures can be used because the secret key is not required to verify the signatures.

12.7 Signatures from Trapdoor Permutations

In this section, we will see how to construct the digital signature with trapdoor permutations. The definition of trapdoor permutations is given below.

DEFINITION 12.7. A trapdoor permutation consists of three algorithms (G, F, F^{-1}) such that:

1. G is the key generation algorithm that outputs the public key pk and the signing key sk.
2. F is a function $F(pk, \cdot) : X \to X$, where X is the input space.
3. $F^{-1}(sk, y)$ is a function that inverts the function F at y using sk.

The definition of a secure trapdoor permutations is as follows.

DEFINITION 12.8. A trapdoor permutation (G, F, F^{-1}) is secure if and only if the function $F(pk, \cdot)$ is one-way without the trapdoor sk.

A naive idea of building the signature scheme $(Gen, Sign, Verify)$ from trapdoor permutations (G, F, F^{-1}) is as follows.

- Gen use G to output the verification key vk and the signing key sk.
- $Sign(m, sk) = F^{-1}(sk, m) = \sigma$
- $Verify(m, \sigma, vk)$ checks if $F(pk, \sigma) = m$. The verification function accepts if and only if $F(pk, \sigma) = m$.

If the signature is generated correctly, the verification must accept the signature because

$$F(pk, \sigma) = F(pk, F^{-1}(sk, m)) = m.$$

However this naive construction is not secure. The adversary with the knowledge of the public key pk can generate the message-signature pair as $(m, \sigma) = (F(pk, x), x)$. In this case, the verification will accept (m, σ) because $F(pk, \sigma) = F(pk, x) = m$. This attack is called zero-time attack or no-message attack.

Therefore we need to modify the naive construction to ensure security. The idea is that we can insert randomness in the signing function to prevent the no-message attack. A

straightforward way to do that is to perform hash on the message and then sign the hash with the signing key. In this way, we define the full-domain hash signature.

DEFINITION 12.9. A full-domain hash signature is a signature scheme $(Gen, Sign, Verify)$ such that:

1. Gen use G to output the verification key vk and the signing key sk. It also specifies a function H modeled as a random oracle.
2. $Sign(m, sk) = F^{-1}(sk, H(m)) = \sigma$
3. $Verify(m, \sigma, vk)$ checks if $F(pk, \sigma) = H(m)$. The verification function accepts if and only if $F(pk, \sigma) = H(m)$.

Intuitively, the no-message attack does not break the full-domain hash because by computing $F(pk, \sigma)$, the adversary can only achieve $H(m)$. The adversary cannot continue to compute m because $H(\cdot)$ is hard to invert. In practice, a secure hash function like SHA can be used as H.

Now we will prove the many-time security of the full-domain hash signature.

Proof. Suppose there exists an adversary A that can win the many-time signature game for the full domain hash signature scheme $(Gen, Sign, Verify)$. That is to say, the adversary can produce a message-signature pair (m, σ) such that $Verify(m, \sigma, pk) = 1$. We first claims that the adversary must query the random oracle H for input m before. Otherwise, we could build a wrapper B around the adversary A that outputs $H(m)$ by computing $F(pk, \sigma)$ without querying the random oracle on input m. This is a contradiction because it violates the programmability property of the random oracle. For a random oracle, the output for the un-queried input m is not determined. Thus, the adversary should never know $H(m)$ before querying m.

Now that we know the adversary A must query the random oracle on input m, we will show that we can build another wrapper on A that violates the security, specifically the un-invertability, of the trapdoor permutation (G, F, F^{-1}). The idea of proving it is that the challenger can program the random oracle so that generating signature does not need the knowledge of the signing key. To do that, whenever the adversary asks the challenger to sign a message m_i, the challenger will uniformly select σ_i and set the random oracle so that the output for input m_i will be $F(pk, \sigma_i)$. Then the challenger sends σ_i as the signature for the message m_i to the adversary. With this idea, we could build the following wrapper B', which takes as input the public key pk and the input y. Suppose the adversary A makes q queries to the random oracle.

1. Choose j at random from $\{1, 2, \ldots, q\}$.
2. Run A.
3. Whenever A makes a query to the random oracle $H(m_i)$, answer it as follows:
   - If $i = j$, return y as the answer.
   - Else choose a σ_i uniformly at random, compute $y_i = F(pk, \sigma_i)$ and set it as the output of the random oracle on input m_i. Then return y_i as the answer.
4. Whenever A requests a signature on message m. Assume $m = m_i$. If $i \neq j$, then return σ_i as the signature. Otherwise, abort.

5. After A finishes its execution with an output (m, σ), this wrapper outputs σ.

□

We say the wrapper adversary B' wins the game, if $F^{-1}(sk, \sigma) = y$. Assume B' picks j such that $m_j = m$. Since $F^{-1}(sk, H(m)) = \sigma$ due to the definition of the adversary A and $H(m) = y$ according to the Step 3, the wrapper adversary B' has found σ such that $\sigma = F^{-1}(sk, y)$ without the knowledge of sk. Thus, if B' guesses correctly, then B' wins the game. Because $m \in \{m_1, \ldots, m_q\}$, the probability for B' to successfully guess j such that $m_j = m$ is equal to $\frac{1}{q}$. Thus the probability that the wrapper B' wins the game is equal to the probability of the adversary A wins the many-time signature game divided by q, which is not negligible. This contradicts the security property of the trapdoor permutation.

12.8 Summary

In this lecture, we discuss in detail how to define security of digital signatures. We introduce several real-world applications of digital signatures. We show how to use trapdoor permutation together with the random oracle to construct the signature scheme, full-domain hash. We prove the security of the full-domain hash signature.

Acknowledgement

These scribe notes were prepared by editing a light modification of the template designed by Alexander Sherstov.