CS 7880 Graduate Cryptography November 6, 2017

Lecture 17: Signatures from OWFs, and Identification Schemes

Lecturer: Daniel Wichs *Scribe: Xuangui Huang*

1 Topics Covered

- Signatures from OWFs
- Identification Schemes

Last time we saw signatures from Trapdoor Permutations in the random oracle model. This time we will see signatures without the use of the ideally omnipotent random oracle. We first construct signatures from one-way functions and collision resistant hash functions using the idea of chain signatures and tree signatures. We will also see signatures from the discrete log assumption in the next lecture, but before that at the end of this lecture we will introduce the concept of identification schemes, and Schnorr ID scheme in particular.

2 Signatures from OWFs

We will first construct 1-time secure signature, then improve it to be many-time secure using chain signatures and tree signatures.

2.1 One-time signature

A One-time signature scheme has the same syntax as we defined for general signature schemes:

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$, generates the pair of public and private key (pk, sk);
- $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$, signs the message m using the private key sk;
- $b \leftarrow \mathsf{Ver}_{pk}(m, \sigma)$, verifies the message against its signature using public key.

But the definition of security is changed: the adversary is allowed to perform only one signature query in the signature game before forging a signature.

2.2 Lamport's one-time signature scheme

Assume the one-way function exists and f is the one-way function. Then we can construct the following signature scheme, which is called Lamport's signature scheme:

- $\mathsf{KeyGen}(1^n)$: randomly pick $x_1^0, x_2^0, \ldots, x_\ell^0 \leftarrow \{0,1\}^n$, and $x_1^1, x_2^1, \ldots, x_\ell^1 \leftarrow \{0,1\}^n$; let $y_i^b = f(x_i^b)$ for all $i \in \{1, 2, \ldots, \ell\}$ and $b \in \{0, 1\}$, define pk as all the y_i^b's, and sk as all the x_i^b's.

- $\mathsf{Sign}_{sk}(m)$: suppose $m = (m_1, \ldots, m_\ell)$, output $\sigma = \{x_i^{m_i}\}_{i=1,\ldots,\ell}$.
- $\mathsf{Ver}_{pk}(m, \sigma)$: suppose $\sigma = \{\sigma_i\}_{i=1,\ldots,\ell}$, check if $y_i^{m_i} = f(\sigma_i)$ for all i.

The one-time security of this scheme comes from security of one-way function f. Intuitively, if there is an adversary that can forge a signature of a fresh message after one query, we can put the output value y that we want to invert at the coordinate i^* on which the bit of the queried message and the forged message are different; then the forged signature on coordinate i^* provides the answer. We guess this i^* and $m^*_{i^*}$ uniformly at random at the beginning, set $y_{i^*}^{m^*_{i^*}}$ to be y and generate other parts of the keys as defined in KeyGen, then simulate the adversary to get the forged signature σ^* and output $\sigma^*_{i^*}$. The probability that we successfully invert f on y is $\frac{1}{2\ell}$ times the probability that this adversary forges σ^* for m^*, which remains non-negligible, thus contradicting the security of f.

2.3 From one-time to stateful many-time signature: chain signature

Given a one-time signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$, we want to construct a stateful signature scheme $(\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Ver}')$ that is secure in usual sense.

The first attack on this problem is by using "Chain signatures":

1. At the beginning, use KeyGen to generate (pk_0, sk_0);
2. To sign the first message m_1, use KeyGen to generate (pk_1, sk_1), set $\sigma'_1 \leftarrow \mathsf{Sign}_{sk_0}(m_1, pk_1)$, and the signature is $\sigma_1 = ((m_1, pk_1), \sigma'_1)$;
3. To sign the second message m_2, use KeyGen to generate (pk_2, sk_2), set $\sigma'_2 \leftarrow \mathsf{Sign}_{sk_1}(m_2, pk_2)$, then the signature is $\sigma_2 = (\sigma_1, (m_2, pk_2), \sigma'_2)$;
4. $(pk_3, sk_3) \leftarrow \mathsf{KeyGen}(1^n)$, $\sigma'_3 \leftarrow \mathsf{Sign}_{sk_2}(m_3, pk_3)$, $\sigma_3 = (\sigma_1, \sigma_2, (m_3, pk_3), \sigma'_3)$;
5. $(pk_4, sk_4) \leftarrow \mathsf{KeyGen}(1^n)$, $\sigma'_4 \leftarrow \mathsf{Sign}_{sk_3}(m_4, pk_4)$, $\sigma_4 = (\sigma_1, \sigma_2, \sigma_3, (m_4, pk_4), \sigma'_4)$;
6. ...

Generally speaking, on each message, generate a new key pair, use the previous private key to sign the message and the new public key, then use this signature, the new public key, and all the previous signatures as the signature. On each chained signature σ, starting at $i = 0$ the verifier can verify m_{i+1} and pk_{i+1} using pk_i, getting public key pk_{i+1} to continue its verification along the chain.

Security comes from the fact that for each message we generate a new pair of keys. More formally, if there is a PPT A that can forge the signature σ^* for fresh message m^* after t signature queries with non-negligible probability, we will construct a PPT A' that breaks the one-time security of the original scheme. We have

$$\sigma^* = (\sigma^*_1, \ldots, \sigma^*_t, (m^*, pk^*), \sigma^{*\prime}_{t+1}),$$

but note that A may forge the chain so it is possible that $\sigma^*_i \neq \sigma_i$ for some $i \leq t$. Denote (m^*_i, pk^*_i) as the message and public key pair that σ^* contains for round i. Let $i^* \leq t$ be the largest number such that we have $\sigma^*_i = \sigma_i$ for all $i \leq i^*$. Define A' as follows:

1. Receive pk from Challenger;
2. Use KeyGen to generate (pk_i, sk_i) for all $i \neq i^*$, set $pk_{i^*} = pk$;
3. Simulate the interaction of A and Challenger of the t-time signature game using the above (pk_i, sk_i)'s, except that at the i^*+1 round (if $i^* \neq t$), when A asks the Challenger to sign (m_{i^*+1}, pk_{i^*+1}), ask the Challenger in one-time signature game to sign it and return the signature to A;
4. Get the forged message m^* and the forged signature σ^* from A;
5. - If $i^* \neq t$: extract $(m^*_{i^*+1}, pk^*_{i^*+1})$ and its signature from σ^*, send them as the forged message and signature to the Challenger of one-time signature game;
   - Otherwise: extract (m^*, pk_{t+1}) and its signature σ'_{t+1} from σ^*. send them to the Challenger.

Correctness of this reduction is obvious: if $i^* = t$, as m^* is the $(t+1)$-th message and we set pk_t to be pk, A must forge $\mathsf{Sign}_{sk}(m^*, pk_{t+1})$ in σ^*; otherwise, as we set pk_{i^*} to be pk, A must forge $\mathsf{Sign}_{sk}(m^*_{i^*+1}, pk^*_{i^*+1})$.

To get the correct i^*, simply guess it uniformly at random at the beginning of A', and the success probability remains non-negligible.

There are two downsides of this scheme:

- it's stateful: the signer has to remember all the previous messages and signatures.
- if the length of keys depends on the length of messages, as in Lamport's scheme, then we can sign only finite rounds: we have to choose pk_0 ahead of time and pk_i signs pk_{i+1} and m_{i+1}, thus the length of pk_0 already determines the number of messages we can sign; more specifically for Lamport's scheme, we can only sign a logarithmic (of length of pk_0) number of messages, i.e. we need exponential (of number of messages) length of key.
- the length of signatures grows as the number of messages increases.

2.4 Lamport's + CRHF, and tree signatures

In this subsection we deal with the second and the third downside.

To decouple the dependence of the length of keys on the length of messages, we can always use CRHF H_s to hash messages into n-bit strings before signing and verification, where s is part of the public key. A one-time secure signature scheme remains secure with this modification; otherwise, either by querying on $H_s(m_1)$ and forging $H_s(m^*)$ we can break the security of the original scheme, or $H_s(m^*)$ collides with $H_s(m_1)$ thus breaking the security of H_s, both contradicting our assumptions.

For the chain signature based on Lamport's scheme, now the total length of public key we need is only $2n^2$, no matter how many messages we want to sign. But this method doesn't address the third downside: we still need to remember all the previous signatures, which is an inherent property of chain signatures.

To deal with this problem, we can use the following so-called "tree signatures".

First we use the following procedure to generate keys:

- Let ℓ be the length of messages;
- For every binary string α of length $\leq \ell$ including the empty string ε, use KeyGen to generate key pair (pk_α, sk_α);
- For all binary string α of length $\leq \ell - 1$, define $\sigma_\alpha \leftarrow \mathsf{Sign}_{sk_\alpha}(pk_{\alpha 0}, pk_{\alpha 1})$;
- public key is pk_ε, private key is all the key pairs and σ_α's generated.

Intuitively speaking, we have a depth-ℓ complete binary tree, of which each node corresponds to a binary string of length $\leq \ell$ and for all binary string α we have nodes $\alpha 0$ and $\alpha 1$ are children of node α. Each node is also assigned a key pair (pk_α, sk_α), and a signature σ_α of the public keys of its two children under its own private key. Then each length-ℓ message $m = m_1 m_2 \cdots m_\ell$ defines a unique path from root to leaves in this tree: root (node ε), node m_1, node $m_1 m_2$, ..., till node m.

Note that the size of secrete key is exponential in ℓ. But we can generate all the key pairs and all the σ_α's on the fly. Note that to get σ_α we have to make sure that both $(pk_{\alpha 0}, sk_{\alpha 0})$ and $(pk_{\alpha 1}, sk_{\alpha 1})$ have been sampled. We can generate them from leaves to root to satisfy this requirement. Once we generate a new key pair for a new node, we have to remember it because we cannot sign two different messages using the same private key in its parent. The total number of things we need to remember is polynomial, because the total number of queries is polynomial.

Given a message $m = m_1 m_2 \cdots m_\ell$, its signature is

$$\begin{aligned}\sigma = (&\mathsf{Sign}_{sk_m}(m),\\ &pk_0, pk_1, pk_{m_1 0}, pk_{m_1 1}, pk_{m_1 m_2 0}, pk_{m_1 m_2 1}, \ldots, pk_{m_1 m_2 \cdots m_{\ell-1} 0}, pk_{m_1 m_2 \cdots m_{\ell-1} 1},\\ &\sigma_\varepsilon, \sigma_{m_1}, \sigma_{m_1 m_2}, \ldots, \sigma_{m_1 \cdots m_{\ell-1}}).\end{aligned}$$

Intuitively, along the unique path from root to node m we collect all the signatures assigned to them, the public keys assigned to them and their siblings, and its signature using private key sk_m. It is easy to see that the signature length is polynomial in ℓ and doesn't change if we want to sign more messages.

Correctness comes from the observation that we can use pk_α to verify σ_α for $pk_{\alpha 0}$ and $pk_{\alpha 1}$, therefore starting from pk_ε we can verify along the path from root to node m and then use pk_m to verify.

Security comes from the observation that if we forge the signature σ^* for a fresh message $m^* = m_1^* m_2^* \cdots m_\ell^*$, then we must forge the signature at some node α along the path from root to node m^*. Denote σ^* as

$$\begin{aligned}\sigma^* = (&\sigma^{*\prime},\\ &pk_0^*, pk_1^*, pk_{m_1^* 0}^*, pk_{m_1^* 1}^*, pk_{m_1^* m_2^* 0}^*, pk_{m_1^* m_2^* 1}^*, \ldots, pk_{m_1^* m_2^* \cdots m_{\ell-1}^* 0}^*, pk_{m_1^* m_2^* \cdots m_{\ell-1}^* 1}^*,\\ &\sigma_\varepsilon^*, \sigma_{m_1^*}^*, \sigma_{m_1^* m_2^*}^*, \ldots, \sigma_{m_1^* \cdots m_{\ell-1}^*}^*).\end{aligned}$$

In the signature queries, we have already seen some key pairs and σ_α's. Let α^* be the longest common suffix of m^* and m_i's among all i, and $\ell^* = |\alpha^*|$. Since m^* is fresh, we always have $\ell^* \leq \ell - 1$. Before forging m^*, we have already seen pk_0, pk_1, $pk_{m_1^* 0}$, $pk_{m_1^* 1}$, $pk_{m_1^* m_2^* 0}$, $pk_{m_1^* m_2^* 1}$, $\ldots$, $pk_{m_1^* \cdots m_{\ell^*}^* 0}$, $pk_{m_1^* \cdots m_{\ell^*}^* 1}$, and σ_ε, $\sigma_{m_1^*}$, $\sigma_{m_1^* m_2^*}$, $\ldots$, $\sigma_{m_1^* \cdots m_{\ell^*}^*}$.

But we might still forge them in σ^*. Define $\ell^{**} \leq \ell^*$ to be the largest number such that for all $j \leq \ell^{**}$, we have $pk_{m_1^* \cdots m_j^* 0} = pk^*_{m_1^* \cdots m_j^* 0}$, $pk_{m_1^* \cdots m_j^* 1} = pk^*_{m_1^* \cdots m_j^* 1}$, and $\sigma_{m_1^* \cdots m_j^*} = \sigma^*_{m_1^* \cdots m_j^*}$. That is, we use the true σ_α's on the nodes corresponding to the first ℓ^{**} bits and use the true pk_α's on the nodes corresponding to the first $\ell^{**}+1$ bits and their siblings.

Now look at the node $m' = m_1^* \cdots m^*_{\ell^{**}+1}$. Note that $(pk_{m'}, sk_{m'})$ must have been generated during the signature queries, because $\ell^{**} \leq \ell^*$ and for all $j \leq \ell^*$ both $(pk_{m_0^* \cdots m_j^* 0}, sk_{m_0^* \cdots m_j^* 0})$ and $(pk_{m_0^* \cdots m_j^* 1}, sk_{m_0^* \cdots m_j^* 1})$ have already been generated by definition.

- If $\ell^{**}+1 = \ell$, then $m' = m^*$ and we have forged $\sigma^{*\prime} = \mathsf{Sign}_{sk_{m'}}(m^*)$;
- otherwise, what we have forged is $\sigma^*_{m'} = \mathsf{Sign}_{sk_{m'}}(pk^*_{m'0}, pk^*_{m'1})$.

In both cases, if we substitute $(pk_{m'}, sk_{m'})$ by (pk, sk) of the one-time signature game, then after simulation of this tree signature game we can get a forged signature for the respected message under (pk, sk), thus breaking the one-time security of the original scheme. We can receive pk from the Challenger. If $\ell^{**}+1 \neq \ell$, in the simulation we need to use the one-time query to the Challenger to get $\sigma_{m'} \leftarrow \mathsf{Sign}_{sk}(pk_{m'0}, pk_{m'1})$.

To get the correct m' we can simply guess which KeyGen sample we use to generate $(pk_{m'}, sk_{m'})$ in the tree signature game. There are polynomially many KeyGen samples, thus the success probability remains non-negligible.

2.5 Further improvement

The scheme given in the previous subsection is still stateful as we need to remember all the key pairs generated. However, we can use PRF to eliminate the state. More precisely, we define $(pk_\alpha, sk_\alpha) = \mathsf{KeyGen}(1^n, F_k(\alpha))$ for all α including ε, i.e. we use $F_k(\alpha)$ as the random bits used by KeyGen, where F is some suitable PRF. (As previously defined in this class we have PRF for strings of the same length for each n, but we can always convert α to distinct strings that have this property: e.g. $|\alpha|\alpha 0^{\log \ell + \ell - \log|\alpha| - |\alpha|}$.) Then we can get all key pairs and all σ_α's directly using k. Now the private key contains only k.

It is easy to see that after this modification a secure tree signature remains secure; otherwise we can distinguish F_k from R as oracles by simulating the tree signature game so that if the oracle is F_k then it simulates the game for this new signature, but if the oracle is the true random oracle R then it simulates the game for the original tree signature.

Also notice that we only need a weaker notion of one-time security to prove the tree signature is secure. In the above reduction, after we sample $pk_{m'0}$ and $pk_{m'1}$ we can already query the Challenger on $(pk_{m'0}, pk_{m'1})$. We can do this even if we postpone our receiving of pk to set $pk_{m'}$ after this query because we're generating them from leaves to root. Thus the one-time security we need is weaker: the adversary have to query the Challenger before it gets the public key.

Note that public key also contains s, thus it decides the hash function H_s. A weaker notion of one-time security means we only need a weaker notion of CRHF to shorten messages: universal one-way hash functions (UOWHFs). In UOWHF, a hash function is secure if the attacker cannot get a colliding x' with non-negligible probability after it first selects x then gets H_s, which is exactly what we need here.

As UOWHFs can be built from OWFs and PRFs can also be built from OWFs, we can build the whole scheme using only one-way functions.

3 Identification Scheme

Suppose we want to log in websites: we have to convince those websites that we are who we are, but we don't let them learn more. More specifically, we want to interact with them in a way that after the interaction, they know who we are but they cannot imitate us to log in other websites. What we need here is the Identification Scheme.

An Identification Scheme is a tuple $(\mathsf{KeyGen}, \mathsf{P}, \mathsf{V})$ of KeyGenerator, Prover, and Verifier such that they interact in the following ways:

1. $(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$, Prover P gets pk and sk, Verifier V gets pk;
2. P interacts with V;
3. V output a value $b \in \{0, 1\}$.

We denote $b \leftarrow Output(\mathsf{P}(pk, sk) \leftrightarrow \mathsf{V}(sk))$ as the output of this scheme.

We use the "honest verifier" setting here, which means the verifier is honest but curious: it will act exactly as what it is told to do in the scheme, but it may try to learn more about the prover from the scheme in order to claim the prover's identity with other verifiers. Define $View(\mathsf{P}(pk, sk) \leftrightarrow \mathsf{V}(pk))$ as the view of verifier during the interaction.

An Identification Scheme must satisfy the following two properties:

- Correctness: $\Pr[b = 1] = 1$;
- Security: For any PPT adversary A we define the identification game $\mathsf{IDGame}_A(1^n)$ as follows:
  1. Challenger samples $(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$ amd sends pk to A;
  2. A sends "next" to Challenger and gets $tr_i \leftarrow View(\mathsf{P}(pk, sk) \leftrightarrow \mathsf{V}(pk))$, repeat for any number of times as A wants;
  3. A then interacts with $\mathsf{V}(pk)$;
  4. the output of this game is the output of V at the end of their interaction.

  Then an identification scheme is secure if for all PPT A, $\Pr[\mathsf{IDGame}_A(1^n) = 1] = negl(n)$.

3.1 Schnorr ID Scheme

Below we will see an identification scheme based on the discrete log assumption. We will see the proof of its security in the next lecture and learn how to convert any ID scheme with some specific internal structures to a signature scheme.

The Schnorr ID Scheme is as follows:

1. KeyGen: $(G, g, q) \leftarrow \mathsf{GroupGen}(1^n)$, where q is a prime, g is the generator of order q in G; $x \leftarrow \mathbb{Z}_q$, set $h = g^x$, $pk = h$, $sk = x$;
2. Prover P samples $r \leftarrow \mathbb{Z}_q$ and sends $a = g^r$ to Verifier V;

3. Verifier V samples $c \leftarrow \mathbb{Z}_q$ and sends it to P;
4. P sends $z = r + cx$ to V;
5. V outputs 1 iff $g^z = ah^c$.