CS 7880 Graduate Cryptography November 12, 2015

Lecture 17:

Lecturer: Daniel Wichs *Scribe: Giorgos Zirdelis*

1 Topics Covered

- Signatures from TDP in RO model
- Signatures from CRHF and OWFs
- Chain based signatures
- Tree based signatures

2 Signatures from TDP in the RO model

The signature scheme from trapdoor permutations in the RO model is defined succinctly as follows:

- A pair of keys $(\mathsf{VK}, \mathsf{SK})$ which is the same as with trapdoor permutations just by setting $\mathsf{VK} = \mathsf{PK}$.
- $\mathsf{sign}(\mathsf{SK}, m) = \mathsf{Inv}_{\mathsf{SK}}(\mathsf{RO}(m))$
- $\mathsf{verify}(\mathsf{VK}, m, \sigma)$: $\mathsf{RO}(m) = f_{\mathsf{PK}}(\sigma)$ with $f_{\mathsf{PK}} : D_{\mathsf{PK}} \to D_{\mathsf{PK}}$ and $\mathsf{RO} : \{0,1\}^* \to D_{\mathsf{PK}}$.

To prove security, we first define a signature game, namely $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$ with n being the security parameter. In this game, a PPT adversary $\mathcal{A}$ is allowed to make queries to the random oracle and to the signing oracle. after that, he must come up with a new message $m^\star$ and a siganture ß for that message. A pictorial view of the game is given below:

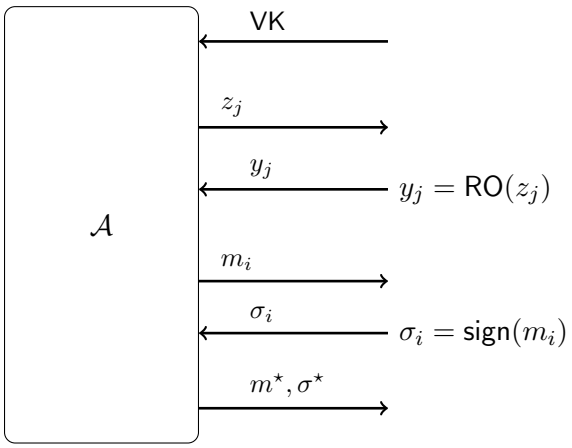


Figure 1: $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$.

$\mathcal{A}$ wins (or outputs 1) iff $m^\star \notin m_i$ and $\mathsf{RO}(m^\star) = f_{\mathsf{PK}}(\sigma^\star)$. We note that the adversary has access to the Random Oracle only by querying it.

Before we continue we make a couple of simplifying assumptions for which $\mathcal{A}$ maintains the same success probability and we can always convert an adversary to one that satisfies these assumptions:

1. $\mathcal{A}$ makes exactly $q = q(n)$ distinct queries to the Random Oracle
2. $\mathcal{A}$ always queries RO on m_i and $m^\star$

Our goal it to use an attacker $\mathcal{A}$ that can win $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$ with some non-negligible probability $\varepsilon(n)$, to create an attacker $\mathcal{B}$ that inverts the trapdoor permutation with non-negligible probability. We do this with a reduction. The usual trapdoor permutation security game is given below:

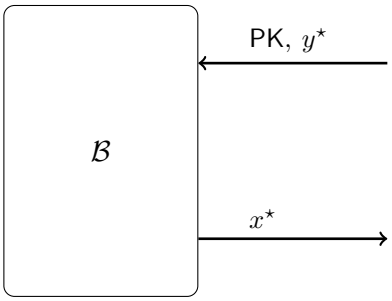
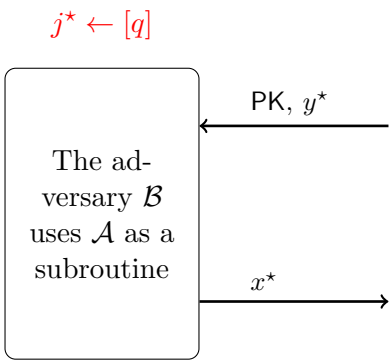


Figure 2: The trapdoor permutation game.

$\mathcal{B}$ wins (or outputs 1) iff $f_{\mathsf{PK}}(x^\star) = y^\star$. PK is a random public key of the trapdoor permutation and $y^\star$ is a random value from D_{PK}.

In the reduction, $\mathcal{B}$ has to give the input to $\mathcal{A}$ (or play $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$ with) as $\mathcal{A}$ expects it to be. The first thing that $\mathcal{B}$ does it to give $\mathcal{A}$ the key PK. After that, there are two types queries that $\mathcal{B}$ must answer to $\mathcal{A}$: random oracle and signature queries. While we will see in detail how $\mathcal{B}$ answers these queries there is a subtle point here. At some point $\mathcal{B}$ has to give $\mathcal{A}$ the image he wants to invert, i.e. $y^\star$. He has to do that when $\mathcal{A}$ queries the message $m^\star$ on the random oracle. The problem is that $\mathcal{B}$ does not know when $\mathcal{A}$ will do that. So what is the best strategy for $\mathcal{B}$ here? To make a guess! To formalize this a bit more, we define a new $\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n})$ where we pick at random a query index from 1 through q and now the probability of $\mathcal{B}$ winning the game also depends on guessing correct when $\mathcal{A}$ will query $m^\star$ on the random oracle, which essentially makes it more difficult for $\mathcal{B}$ to do so. We denote with red color the additions on the new $\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n})$ game that is illustrated below:



$\mathcal{B}$ wins (or outputs 1) iff $f_{\mathsf{PK}}(x^\star) = y^\star \wedge \ j^\star \leftarrow [q]$

Figure 3: $\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n})$.

The probability of $\mathcal{B}$ winning $\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n})$ is,

$$\Pr[\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n}) = 1] = \Pr[\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n}) = 1] \cdot \Pr[j^\star \text{ is correct} | \mathsf{SigGame}_{\mathcal{A}}(\mathsf{n}) = 1] = \frac{\varepsilon(n)}{q}.$$

This follows from the fact that $j^\star$ is random and indepentent of $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$ and also $\mathcal{A}$ does not know its value, hence $\Pr[j^\star \text{ is correct} | \mathsf{SigGame}_{\mathcal{A}}(\mathsf{n}) = 1] = \Pr[j^\star \text{ is correct}] = 1/q$. We assumed that the winning probability $\varepsilon(n)$ of $\mathcal{A}$ for $\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n})$ is non-negligible, therefore the probability of $\mathcal{B}$ winning the game is also non-negligible. Next, we give in detail how $\mathcal{B}$ works in steps (i.e. the reduction):

1. Choose $j^\star \leftarrow [q]$
2. Set $\mathsf{VK} = \mathsf{PK}$
3. On a RO query z_j:
   (a) if $j \neq j^\star$: $\begin{cases} x_j \leftarrow D_{\mathsf{PK}} \\ \text{output } y_j = f_{\mathsf{PK}}(x_j) \end{cases}$
   (b) if $j = j^\star$: output $y_j = y^\star$
4. On sign query $m_i = z_j$:
   (a) if $j \neq j^\star$: output x_j
   (b) if $j = j^\star$: quit (or output something bogus)
5. Output $x^\star = \sigma^\star$

Some observations are in order.

- There is no RO in this game.

- Steps 3 and 4 are repeated for a total of q times.
- At step 3(a) x_j is chosen at random therefore $y_j = f_{\mathsf{PK}}(x_j)$ is also random. We store the pairs (x_j, y_j) because we need them at step 4. The same holds for 3(b), because $y^\star$ is chosen at random.
- At step 4(a) we know how to answer the sign queries, because at step 3 we chose first the x_j and the y_j. Note that we have assumed that before $\mathcal{A}$ make a sign query, he first makes a RO query on the same message (moce precisely, he makes a query to what he believes to be a RO).

By the previous analysis is that the winning probability of $\mathcal{B}$ is:

$$\Pr\left[\mathcal{B}(\mathsf{PK}, y^\star) = x^\star : (\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^n), x^\star \leftarrow D_{\mathsf{PK}}, y^\star \leftarrow f_{\mathsf{PK}}(x^\star)\right] = \frac{\varepsilon(n)}{q}$$

3 Signatures from Collision-Resistant Hashing (and OWFs)

A simple start is to do one-time signatures. In that scheme the game is the same as $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$ but now the adversary is restricted to making only one query before he submits his forgery.

3.1 A construction from one-way functions

Lamport's one time signature scheme.
Let f be a OWF. The $\mathsf{KeyGen}(1^n)$ generates the following secret and verification keys with 2ℓ values each:

$$\mathsf{SK} = \begin{bmatrix} x_{0,1}, & x_{0,2}, & \dots, & x_{0,\ell} \\ x_{1,1}, & x_{1,2}, & \dots, & x_{1,\ell} \end{bmatrix}$$

where $x_{b,i} \leftarrow \{0,1\}^n$, $b \in \{0,1\}$ and $i \in [\ell]$,

$$\mathsf{VK} = \begin{bmatrix} y_{0,1} = f(x_{0,1}), & y_{0,2} = f(x_{0,2}), & \dots & y_{0,\ell} = f(x_{0,\ell}) \\ \dots & \dots & y_{b,i} = f(x_{b,i}), & \dots \end{bmatrix}$$

In order to sign a message m we look at the bits of the message and we output the corresponding parts of SK. If the i-th bit of m is 0 we output $x_{0,i}$ else we output $x_{1,i}$, i.e. $\mathsf{sign}(\mathsf{SK}, m) = (x_{m_i,i})_{i=1,\dots,\ell}$ where $m \in \{0,1\}^\ell$.

To verify a signature on a message m we check bit by bit if the signature is indeed the output of f on the message bits, i.e. $\mathsf{verify}(\mathsf{VK}, m)$: check if $f(x_{m_i,i}) = y_{m_i,i}$ for all $i \in [\ell]$.

The security of this scheme follows from the security of OWF. The adversary can see the signature of a message he chooses, but when he submits his forged message it must differ at least on one bit from the messages he chose. For that different bit, the adversary has to invert the OWF f on its relevant input from the secret key, in order for his message to have a valid signature.

Note that this scheme is inefficient because each entry in SK is an n-bit string and the VK can be used only for one message. Also the key sizes are proportional to the message length.

We can use this one-time signature scheme to sign arbitrary long messages in $\{0,1\}^*$. We do this by using a CRHF $H_s : \{0,1\}^* \to \{0,1\}^\ell$ to create an ℓ-bit digest of the message and then sign the digest.

As for the security of this scheme let the adversary pick a message m and see its signature. Now he picks a message $m^\star$. Either $H_s(m) \neq H_s(m^\star)$ and $\mathsf{sign}(H_s(m)) = \mathsf{sign}(H_s(m^\star))$ and the adversary breaks the one-time signature security or $H_s(m) = H_s(m^\star)$ and the adversary finds a collision.

4 A chain-based signature scheme

We give a short description of a scheme that allows us to sign multiple messages by using a one-time message signature scheme. Also the signing algorithm is now stateful, that is the algorithm has a state which gets updated after each message it signs.

We start by choosing a verification and singing key:cb $\mathsf{VK}_0, \mathsf{SK}_0$. The only key that is public and identifies the signing party is VK_0. The signature for a message m_1 is computed as $\sigma_1 = \mathsf{sign}(\mathsf{SK}_0, (m_1, \mathsf{VK}_1))$ where $(\mathsf{VK}_1, \mathsf{SK}_1)$ is a fresh key pair we choose. To sign a message m_2 we do the same thing, we choose a key pair $(\mathsf{VK}_2, \mathsf{SK}_2)$ and create $\sigma_2 = (\mathsf{SK}_1, (m_2, \mathsf{VK}_2))$, and so on and so forth. That is, before we sign each message m_i we choose a new key pair $(\mathsf{VK}_i, \mathsf{SK}_i)$ and create $\sigma_i = \mathsf{sign}(\mathsf{SK}_{i-1}, (m_i, \mathsf{VK}_i))$. Notice that we keep the whole chain, i.e. all key pairs and signatures from previous steps. In a way, our state at each step is to know at which part of the chain we are.

Why is this scheme secure? The adversary can see all the signatures of the chain. For him to break this scheme, he has to forge a message and signature after he sees exactly one message and signature under a key pair. Therefore, we still rely on the security of the underlying one-time signature scheme.

To check a signature down the chain, e.g. σ_i, we start for the top of the chain and verify every signatures from 1 to i.

The downside with this scheme is that chain grows proportional to the messages we sign. Morover, to send a signature σ_i of a message m_i we need to send all the previous signatures for $i = 1, \ldots, i-1$.

5 Tree-based signatures

We can do a lot better from the previous scheme in terms of size for the signatures with a tree based construction, while remain again stateful. In addition we won't have to send all the previous signatures for each new message we sign.

Let the messages be in $\{0,1\}^n$. As we previously saw, if the message is bigger than n bits he can sign an n-bit hash of the message. The idea is to have a complete binary tree $\mathcal{T}$ of height n. While $\mathcal{T}$ has 2^n leaves and we cannot store it, we will store parts of it. We associate each node of $\mathcal{T}$ with a key pair of signing and verification key, and a signature. The root node is associated with $(\mathsf{VK}_0, \mathsf{SK}_0)$ and VK_0 is the verification key of the whole scheme, i.e. everyone knows VK_0.

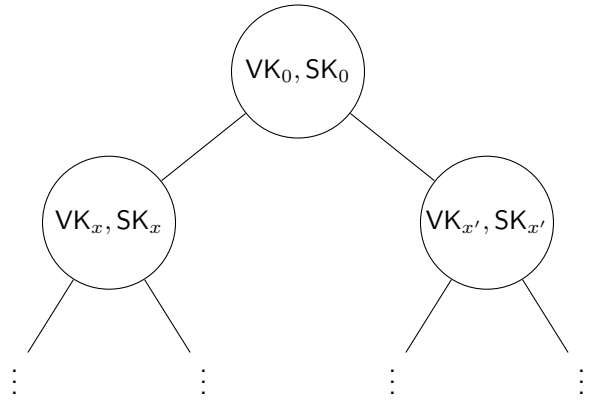


Figure 4: Tree-based signatures.

The root node $(\mathsf{VK}_0, \mathsf{SK}_0)$ is associated with signature $\sigma_0 = \mathsf{sign}(\mathsf{SK}_0, (\mathsf{VK}_x, \mathsf{VK}_{x'}))$. That is, at every level the parent node signs the two verification keys of its two child nodes. Again, let us note that is the structure we want to maintain, we don't store all the signatures and keys.

Since the messages are n-bit strings, for each message we follow a path from the root to some leaf. If the i-th bit is 0 by convention we choose the left child node on the $i+1$ tree level, otherwise we choose the right one. We output the signature we get when we reach a leaf, i.e. the signature of the parent of the leaf that corresponds to the message we want to sign.

To verify, we start from the root and verify the signature of each node of the path that leads to the leaf which represents the signed message.

This scheme remains stateful, because each time we want to sign a message we have to create on the fly everything that we haven't created yet, and keep that part of the tree. So while our signature size is fixed, the state becomes bigger but remains polynomial because we only sign a polynomial number of messages.

5.1 Get rid of the state

The verification and signing keys for each node of a tree path are chosen at random. Since there was some randomness involved in creating theses keys, it would be more convenient to remember just the randomness. If we can remember that randomness, as we traverse a tree path we can regenerate all the verification and signing keys along this path. We can achieve this using a PRF F_k. Let's assume that every tree node has a different label x. Then the randomness associated with node x is $r_x = F_k(x)$. Hence, compared to the previous stateful tree-based signature scheme, now we need to remember only $\mathsf{VK}_0, \mathsf{SK}_0$ and the key k of the PRF.