Lecture 8

Gidon Rosalki

2025-12-31

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_intro_to_crypto`

1 Digital signatures

Alice and Bob wish to communicate, but Eve completely controls the channel. We would like to assure the receiver of a message that it has not been modified. We will discuss the public key counterpart of message authentication codes. The signer holds a *secret* signing key, and the verifier knows the corresponding public *verification* key. This means that anyone can verify the signature, but only one person can create it. This is the inverse of encryption, where everyone knows the public encryption key, but only 1 person knows the private decryption key.

This has the syntax $\Pi = (Gen, Sign, \text{Vrfy})$:

- The key generation algorithm Gen on input 1^n outputs a signing key sk, and a verification key vk
- Sign takes a signing key sk, and a message m, and outputs a signature σ
- Vrfy takes a verification key vk, a message m, and a signature σ, and outputs a bit b

Correctness: For every message m

$$\Pr\left[\text{Vrfy}_{vk}\left(m, \text{Sign}_{sk}\left(m\right)\right) = 1\right] = 1$$

To compare against MACs:

Signatures	MACs
n users require only n secret keys	n users require n^2 secret
Same signature can be verified by all users	
Publicly verifiable and transferable	Privately verifiable and non transferable
Provides non repudiation	More efficient (2 - 3 orders of magnitude faster)

Table 1:

1.1 Security of Signatures

$\mathcal{A}$ knows vk, and can adaptively ask for signatures of messages of its choice. It then tries to forge a signature on a new message.

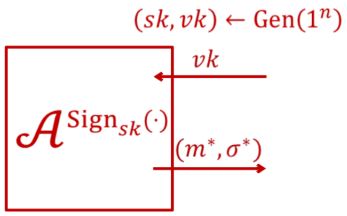

Figure 1: Signature game

We finish with Q, the set of all queries asked by $\mathcal{A}$, and

$$\text{SigForge}_{\Pi,\mathcal{A}}\left(n\right) = \begin{cases} 1, & \text{if } \text{Vrfy}_{vk}\left(m^*, \sigma^*\right) = 1 \wedge m^* \notin Q \\ 0, & \text{else} \end{cases}$$

Definition 1.1. Π *is* ***existentially unforgeable against an adaptive chosen message attack*** *if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $v\left(\cdot\right)$ *such that*

$$\Pr\left[SigForge_{\Pi,\mathcal{A}}\left(n\right) = 1\right] \leq v\left(n\right)$$

2 Constructions

One time signatures are used to construct stateful signatures, which may then be used to construct stateless signatures.

2.1 One time signatures

We will demonstrate **Lamport's One time scheme**:

- $sk = (x_0, x_1)$
- $vk = (f(x_0), f(x_1))$, where f is a one way function
- $\text{Sign}_{sk}(b) = x_b$
- Vrfy receives the message, and the signature, and checks it against the relevant side of the verification key

This way $\mathcal{A}$ needs to compute x_{1-b}, which is equivalent to computing the inverse of f.

More formally: Let f be an OWF. we will define a signature scheme $\Pi = (Gen, \text{Sign}, \text{Vrfy})$ for 1 bit messages as follows:

- Gen (1^n): Sample $x_0, x_1 \leftarrow \{0,1\}^n$, and compute $y_0 = f(x_0)$, and $y_1 = f(x_1)$, output $sk = (x_0, x_1)$ and $vk = (y_0, y_1)$
- $\text{Sign}_{sk}(b)$: Output $\sigma = x_b$
- $\text{Vrfy}_{vk}(b, \sigma)$: If $f(\sigma) = y_b$ output 1, otherwise output 0

Theorem 1. *If f is an OWF, then Π is a secure one time signature scheme for 1 bit messages*

Proof . The concept is that $\mathcal{A}$ forges a signature on $b^* \implies \mathcal{A}$, inverts $y_{b^*} = f(x_{b^*})$. Inverting $f(x_{b^*})$ is clearly hard, even when given x_{1-b^*} and $f(x_{1-b^*})$. An inverter can guess the forged bit b^* ahead of time with probability $\frac{1}{2}$.

We can construct an inverter $\mathcal{B}$ as follows, which takes as input $y = f(x)$ for some $x \leftarrow \{0,1\}^n$.

1. Choose $b^* \leftarrow \{0,1\}$ and set $y_{b^*} = y$
2. Sample $x_{1-b^*} \leftarrow \{0,1\}^n$, and set $y_{1-b^*} = f(x_{1-b^*})$
3. Run $\mathcal{A}$ on input $vk = (y_0, y_1)$
4. When $\mathcal{A}$ requests a signature on b:
   - If $b = b^*$ abort
   - If $b = 1 - b^*$ output x_{1-b^*}
5. If $\mathcal{A}$ output a forgery σ^* on b^*, output σ^*

□

So

$$\begin{aligned}\Pr\left[\mathcal{B}(f(x)) \in f^{-1}(f(x))\right] &\geq \Pr\left[\text{SigForge}_{\Pi,\mathcal{A}}(n) = 1 \wedge \mathcal{B} \text{ does not abort}\right] \\ &= \Pr\left[\text{SigForge}_{\Pi,\mathcal{A}}(n) = 1\right] \cdot \Pr\left[\mathcal{B} \text{ does not abort}\right] \\ &\Pr\left[\text{SigForge}_{\Pi,\mathcal{A}}(n) = 1\right] \cdot \frac{1}{2}\end{aligned}$$

We may note that this scheme only works one time, for a single bit. We may extend this to l bit messages by creating

$$sk = \begin{bmatrix} x_0^1 & x_0^2 & \dots & x_0^l \\ x_1^1 & x_1^2 & \dots & x_1^l \end{bmatrix} \tag{1}$$

$$vf = \begin{bmatrix} f(x_0^0) & \dots & f(x_0^l) \\ f(x_1^0) & \dots & f(x_1^l) \end{bmatrix} \tag{2}$$

Or formally: Let f be an OWF. We define the signature scheme $\Pi = (Gen, \text{Sign}, \text{Vrfy})$ for l bit messages as follows:

- Gen (1^n): For each $i \in [l]$, and $b \in \{0,1\}$, sample $x_{i,b} \leftarrow \{0,1\}^n$ and compute $y_{i,b} = f(x_{i,b})$. Output $sk = \{(x_{i,0}, x_{i,1})\}_{i\in[l]}$ and $vk = \{(y_{i,0}, y_{i,1})\}_{i\in[l]}$
- $\text{Sign}_{sk}(m = m_1 \dots m_l)$: Outputs $\sigma = (x_{1,m_1}, \dots, x_{l,m_l})$
- $\text{Vrfy}_{vk}(m = m_1 \dots m_l, \sigma = (x_1, \dots, x_l))$: If $\forall i \in [l]\ \ f(x_i) = y_{i,m_i}$ output 1, else 0

Theorem 2. *If f is an OWF, then Π is a secure, one time signature scheme for l bit messages*

Proof Idea. Suppose that $\mathcal{A}$ asks for a signature on m, and then forges on $m^* \neq m$. The inverter $\mathcal{B}$ needs to guess $i \in [l]$ such that $m_i^* \neq m_i$ as well as guess the bit m_i^*. □

2.1.1 Summary

Lamport theorised in 1979 that if OWFs exist, then for any polynomial $l = l(n)$ there is a one time signature scheme for signing l bit messages. The following theorem is known as the *Hash and Sign* paradigm:

Theorem 3. *If CRHFs exist, then there is a one time signature scheme that can sign messages of arbitrary polynomial length.*

2.2 Stateful signatures

We are now extending the game, such that $\mathcal{A}$ may request the signature of many different messages.
The signer updates the signing key after each signature.

- The initial state sk_1 is produced by $Gen : (vk, sk_1) \leftarrow Gen(1^n)$
- Signing the ith message updates sk_i to $sk_{i+1} : (\sigma, sk_{i+1}) \leftarrow \text{Sign}_{sk_i}(m_i)$
- Verification requires only vk

For existential unforgeability against an adaptive chosen message attacks

- $\mathcal{A}$ knows vk, and can adaptively ask for signatures of its choice
- The signing oracle maintains the internal state sk_i
- $\mathcal{A}$ tries to forge a signature on a new message

Let us create a stateful scheme. Let $\Pi = (Gen, \text{Sign}, \text{Vrfy})$ be a one time signature scheme for signing "sufficiently long" messages. for $m = m_1 \dots m_n \in \{0,1\}^n$, we let $m\,|_i \stackrel{def}{=} m_1 \dots m_i$ (and $m\,|_0 \stackrel{def}{=} \varepsilon$).
We will define $\Pi' = \left(Gen', \text{Sign}', \text{Vrfy}'\right)$ for signing n bit messages as follows:

- The signer's state is a binary tree with 2^n leaves
- Each node $w \in \{0,q\}^{<n}$ has a left child $w0$, and a right child $w1$
- The tree is of exponential size, but is never fully constructed

Key generation: Each node $w \in \{0,1\}^{\leq n}$ is associated with $(vk_w, sk_w) \leftarrow Gen(1^n)$. Keys are generated, and stored only when needed. The state sk_i' consists of all keys and signatures that were generated so far. $vk' = vk_\varepsilon$ and $sk_1' = sk_\varepsilon$. Note that vk_ε is the root node, with children vk_0, vk_1.

To sign a message $m \in \{0,1\}^n$:

1. Generate a path from the root, to the leaf labelled m: For each proper prefix w of m sample $(vk_{w0}, sk_{w0}), (vk_{w1}, sk_{w1}) \leftarrow Gen(1^n)$
2. Certify the path: For each proper prefix w of m, compute $\sigma_w = \text{Sign}_{sk_w}(vk_{w0}, vk_{w1})$
3. Compute $\sigma_m = \text{Sign}_{sk_m}(m)$
4. Store all generated keys as part of the updated state
5. Output the signature $\left(\left\{\sigma_{m|_i}, vk_{m|_i 0}\right\}_{i=1}^{n-1}, \sigma_m\right)$

Simple example: The message $m = 111$ receives the signature $\text{Sign}_{sk111}(111), \text{Sign}_{sk11}(vk_{111}), \text{Sign}_{sk1}(vk_{11}), \text{Sign}_{sk\varepsilon}(vk_1)$. This simple example is missing the fact that if we now want to sign 110, we need to resign 11, which is a problem from the attack scheme. To fix this, each parent provides the signature for *both* its children at once, and we thus avoid this issue.

Theorem 4. *If* Π *is a one time signature scheme, then* Π' *is existentially unforgeable against chosen message attacks*

Proof Idea. Each sk_w is used to sign exactly one "message". If w is an internal node, then sk_w is used to sign (vk_{w0}, vk_{w1}), and if w is a leaf then sk_w is used to sign w. □

Proof Idea #2. Suppose that $\mathcal{A}$ asks to forge a signature $\left(\left\{\sigma^*_{m^*|_i}, vk^*_{m^*|_i 0}, vk^*_{m^*|_i 1}\right\}_{i=0}^{n-1}, \sigma^*_{m^*}\right)$ on m^*. There are two possible cases:

1. The full path to the leaf m^* already existed, and $\mathcal{A}$ used the same path. This implies that $\mathcal{A}$ must have forged a signature that is a relative of vk_{m^*}, and did not receive any signature that is a relative of vk_{m^*}
2. The full path to leaf m^* did not exist, or $\mathcal{A}$ used a different path. This implies that $\mathcal{A}$ must have forged a signature that is a relative of $vk_{m^*|_i}$ for $i \in \{0, \dots, n-1\}$, and received exactly one signature that is a relative of $vk_{m^*|_i}$

□

This has the problem of needing to remember **all** the sks, since once we have signed a message, we cannot use sk_ε any more, which is necessary to sign another message. We can now move on to stateless signatures, and thus remove this need for state.

2.3 Stateless signatures

Instead of remembering sk_i at every stage, we use PRFs to create them on the fly. The signer's secret key sk is a seed for a PRF $F_{sk}(\cdot)$. $(r_w, r'_w) \stackrel{def}{=} F_{sk}(w)$ is used as the randomness needed for each node $w \in \{0,1\}^{\leq n}$:

- If $w \in \{0,1\}^{<n}$ then r_w is used for sampling (vk_w, sk_w) and r'_w is used for signing (vk_{w0}, vk_{w1})
- If $w \in \{0,1\}^n$, then r_w is used for sampling (vk_w, sk_w), and r'_w is used for signing w

Theorem 5. *If Π is a one time signature scheme, and F is a PRF, then Π'' is existentially unforgeable against chosen message attacks*

Proof Idea. Any adversary $\mathcal{A}$ against Π'' can be used either as an adversary against the stateful scheme Π', or as a distinguisher against the PRF F

$$\begin{aligned}\Pr\left[\text{SigForge}_{\Pi'',\mathcal{A}}(n)=1\right] &\leq \left|\Pr\left[\text{SigForge}_{\Pi'',\mathcal{A}}(n)=1\right] - \Pr\left[\text{SigForge}_{\Pi',\mathcal{A}}(n)=1\right]\right| + \Pr\left[\text{SigForge}_{\Pi',\mathcal{A}}(n)=1\right] \\ &= \left|\Pr\left[\mathcal{D}^{F_{sk}(\cdot)}(1^n)=1\right] - \Pr\left[\mathcal{D}^{f(\cdot)}(1^n)=1\right]\right| + \Pr\left[\text{SigForge}_{\Pi',\mathcal{A}}(n)=1\right]\end{aligned}$$

□

3 Certificates and public key infrastructure

Public key cryptography is great, but we need to distribute the public keys *somehow*. Keys must be authenticated in order to avoid man in the middle attacks. This is done by making use of **Certificate Authorities**:

- A **certificate** is a signature binding an identity to a public key
- We assume that we already trust the CA's verification key vk_{CA} (by hard wiring it into the browser source code or some such)
- The CA provides Alice with $cert_{CA\to A} \stackrel{def}{=} \text{Sign}_{sk_{CA}}(\text{Alice's key is } pk_A)$
- Alice then sends to Bob both pk_A, and $cert_{CA\to A}$

So for example, we can have a root, that has signed all of HUJI, www.gov.il, and CNN. HUJI then signs on CS, and Chem, and CS can sign on Alice, and Bob. This way, everyone only signs a small number of relevant keys, and I can use this chain of trust to trust someone else's key, because I trust the root node.
Certificates should not be valid indefinitely, since an employee may leave or get fired, and secret keys can get stolen. One solution is to add an expiration date, such that the signature is not valid after that date, another approach is to add a revocation list that the authority publishes, and when I received a signed key, I check it against the CA's revoked list.

4 User-server identification

We need a way to identify users to websites, like when you log in to moodle. A trivial method would be for the user to hold a password p, the server to know $y = f(p)$ for some function f, and the user identifies themselves by sending p. This is obviously terrible. It can however be *slightly* improved by using a signature scheme. The user has the signing key sk, and the server knows the verification key vk. The user identifies themselves by signing a message that the server has randomly generated for them.