Digital Signatures

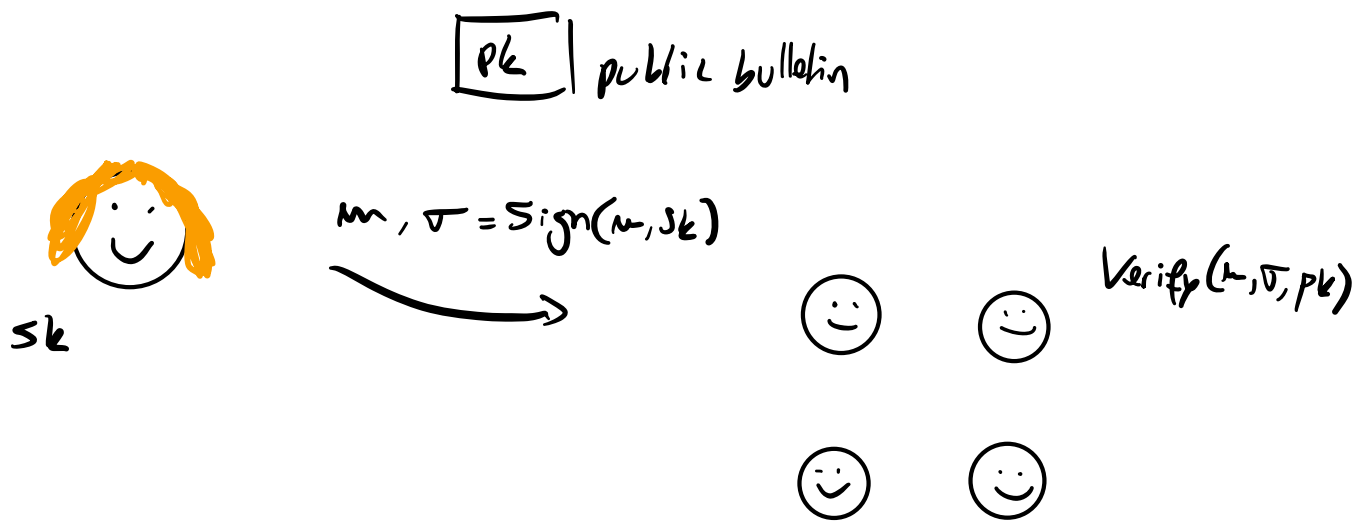
Recommended reading:

KL, Sections 12.1, 12.2, 12.4.1, 12.4.2, 12.6

In the private-key setting we first saw encryption, and then saw authentication. We will now see the "public-key" counterpart of MACs: digital signatures.

A signature scheme allows a signer to sign messages using a secret key such that anyone with the public key can verify, but can't forge.

↳ public verifiability ↳ unforgeability



Let's start by defining the syntax and desirable properties of signature schemes.

Def (Signature scheme). A signature scheme is a tuple of algorithms $(Gen, Sign, Ver)$ such that:

→ Gen is a PPT algorithm that on input 1^n generates a public-key/secret-key pair (pk, sk).

→ $Sign$ is a PPT algorithm that on input a message m and the key sk outputs a signature $\sigma = Sign(sk, m)$.

→ Ver is a deterministic polynomial-time algorithm that on input a message m, a signature σ, and the key pk outputs a bit $b = Ver(pk, m, \sigma)$.
$b = 1 \Leftrightarrow$ "signature is valid for m".

The most basic property we require is correctness: correctly generated signatures should verify correctly with very high probability. That is,

$$Pr(Ver(pk, m, Sign(sk, m)) = 1) \geq 1 - \varepsilon(n).$$

for some negl. function ε, where the probability is over the sampling $(pk, sk) \leftarrow Gen(1^n)$

Security: Intuitively, just like for MACs, we require that an adversary with access to pk and signatures of messages of their choice should not be able to forge a signature for a new message.

More formally, for every PPT adversary $\mathcal{A}$ there is a negl. function $\varepsilon(n)$ such that $\mathcal{A}(1^n)$ wins the following game with probability at most $\varepsilon(n)$:

- The challenger samples $(pk, sk) \leftarrow Gen(1^n)$ and sends pk to $\mathcal{A}$.
- $\mathcal{A}^{O_{sign}}(1^n, pk)$ can query O_{sign} on any message m of their choice, and get back $\sigma = Sign(sk, m)$. Let Q be the set of messages queried by $\mathcal{A}$.
- $\mathcal{A}$ outputs (m^*, σ^*) and wins if $m^* \notin Q$ and $Ver(pk, m^*, \sigma^*) = 1$.

This type of security is called EUACMA, for "existential unforgeability under adaptive chosen-message attacks". We'll just call these schemes "secure".

Signatures from RSA?

Recall the insecure plain RSA PKE scheme.
Let's try to turn it into a signature scheme.

→ $\text{Gen}(1^n)$: same as for plain RSA PKE, but switch pk and sk. That is,
$pk = (N, e)$ and $sk = (N, d)$

→ $\text{Sign}(sk, m) = m^d \bmod N$
($m \in \mathbb{Z}_N^*$)

→ $\text{Ver}(pk, m, \sigma)$: Check if $\sigma^e = m \bmod N$.

Why is this not secure??

Simple no-message attack: Given $pk = (N, e)$,
sample $\sigma \leftarrow \mathbb{Z}_N^*$ and output $(m = \sigma^e \bmod N, \sigma)$
(Or just choose $(m = 1, \sigma = 1)$)

Forging a signature on <u>any</u> message: Given $m \in \mathbb{Z}_N^*$,
choose distinct $m_1, m_2 \neq 1$ s.t. $m = m_1 \cdot m_2$, ask oracle
for signatures σ_1, σ_2 of m_1, m_2, and output $\sigma = \sigma_1 \cdot \sigma_2 \bmod N$.

This works since $\sigma^e = \sigma_1^e \cdot \sigma_2^e = m_1 \cdot m_2 = m \bmod N$.

How can we make this secure? Hash-then-sign

Instead of signing m, sign $H(m)$ for some hash function $H: \{0,1\}^* \to \mathbb{Z}_N^*$.

We can then prove security in the random oracle model, under the RSA assumption.

↓

H maps input unif. at random to $\mathbb{Z}_N^*$.

Signature schemes without PKE

Surprisingly, it turns out that although signatures are a "public-key" notion, they can be built just from any OWF!

We will by constructing a one-time secure signature scheme from any OWF. This is due to Lamport (published officially in 1979).

The notion of one-time security is analogous to the MAC case. We play the game above but A is only allowed to query the signing oracle once.

One-time signatures from OWFs

Let f be any OWF.

→ $\text{Gen}(1^n)$: sample $\begin{bmatrix} x_{1,0} & x_{2,0} & \cdots & x_{n,0} \\ x_{1,1} & x_{2,1} & & x_{n,1} \end{bmatrix}$

with each $x_{i,b} \sim \{0,1\}^n$

Compute $y_{i,b} = f(x_{i,b})$

Then $sk = \begin{bmatrix} x_{1,0} & x_{2,0} & \cdots & x_{n,0} \\ x_{1,1} & x_{2,1} & & x_{n,1} \end{bmatrix}$

$pk = \begin{bmatrix} y_{1,0} & y_{2,0} & \cdots & y_{n,0} \\ y_{1,1} & y_{2,1} & & y_{n,1} \end{bmatrix}$

→ $\text{Sign}(sk, m) = (x_{1,m_1}, x_{2,m_2}, \ldots, x_{n,m_n})$

→ $\text{Ver}(pk, m, \sigma)$: check that $f(\sigma_i) = y_{i,m_i}$

Thm: This signature scheme is one-time secure.

Proof: Suppose not. Let $\mathcal{A}$ be the PPT adversary that wins the one-time forgery game with probability at least $1/p(n)$ for some polynomial p and inf. many n's.

Fix such an n.

We will build a PPT algo. Inv that inverts f with probability at least $\frac{1}{2n \cdot p(n)}$.

On input $y = f(x)$ for $x \leftarrow \{0,1\}^n$, $\mathrm{Inv}(1^n, y)$ works as follows:

→ Sample $i^* \leftarrow \{1, \dots, n\}$ and $b^* \leftarrow \{0,1\}$.
For $i \neq i^*$ or $b \neq b^*$, sample $x_{i,b} \sim \{0,1\}^n$
Set $y_{i^*, b^*} = y$.

Send $(y_{i,b})_{\substack{i \in \{1,\dots,n\} \\ b \in \{0,1\}}}$ to $A(1^n)$

→ A may wish to query the oracle on some m. If $m_{i^*} = b^*$, abort. Else, send $(x_{i,m_i})_{i \in \{1,\dots,n\}}$ to A

→ A outputs (m^*, σ^*). If $m^*_{i^*} = m_{i^*}$, abort. Else, output $\sigma^*_{i^*}$ as the output of Inv.

We claim that Inv succeeds with probability at least $\frac{1}{2n \cdot p(n)}$. First, note that the choice of (i^*, s^*) is independent of A's choice of m and (m^*, σ^*). So the probability that Inv does not abort is at least $\frac{1}{2} \cdot \frac{1}{n}$, since m^* differs from m in at least one coord (wlog). Second, A produces a valid signature σ^* of m^* with prob $\geq \frac{1}{p(n)}$, and in that case $f(\sigma^*_{i^*}) = y_{i^*, m^*_{i^*}} = y$. ▨

From one-time security to EUACMA security in the stateful setting

We briefly argue how to upgrade one-time security to EUACMA security in the stateful setting where Alice (the signer) keeps state (but verifiers can be stateless).

Let's say Alice wants to sign N messges.
Alice generates N pairs $(pk_i, sk_i)_{i \in \{1,\dots,N\}}$

Initially Alice publishes $pk = (pk_1, \dots, pk_N)$.
To sign the i-th message m_i, she computes

$$\sigma_i = Sign(sk_i, m_i)$$

and outputs the signature (σ_i, i).

This solution requires the total number of signatures to be fixed beforehand. This can be avoided by having Alice generate keys on-the-fly.

→ Alice generates (pk_1, sk_1). To sign m_1 she first generates a second pair (pk_2, sk_2) and signs m_1 and second public-key pk_2!!

$$\sigma_1 = Sign(sk_1, (m_1, pk_2))$$

To sign m_2, Alice generates (pk_3, sk_3)
and outputs $\sigma_2 = \text{Sign}(sk_2, (m_2, pk_3))$.

To verify (m_2, σ_2), Bob uses pk_2, which he knows has been generated by Alice.

And so on...

Note 1: this requires us to be able to sign messages that are longer than the public keys.
This is not a problem. Given a fixed length signature scheme, we can use collision-resistant hash functions to sign messages of arbitrary length.
The proof is exactly like for MACs.

Note 2: We can turn this stateful signature scheme into a secure stateless signature scheme using PRFs.
So OWFs $\Rightarrow$ signatures!!
If you are curious, see KL, Section 12.6.3.