CS 127/CSCI E-127: Introduction to Cryptography

Prof. Salil Vadhan Fall 2013

**Lecture Notes 19:**

**Constructions of Digital Signatures, PKI**

**Reading.**

- Katz-Lindell §12.3.2,12.7,12.8

# 1 Practical Constructions

- Hash-then-sign with plain RSA
  - $pk = (N, e)$, $sk = (N, d)$ s.t. $ed \equiv 1 \pmod{\phi(N)}$.
  - $\mathsf{Sign}_{sk}(m) = (H(m))^d \bmod N$ where $H$ is SHA or MD5.
  - Intuition: adversary has "no control" over $H(m)$, so cannot forge by choosing signature first, or by exploiting special inputs on which RSA is easy to invert.
  - RSA PKCS #1: $H(m) = 001FF\cdots FF00 \circ \mathrm{SHA}(m)$.
    - No justification.
    - Doesn't seem related to one-wayness of RSA since $H(m)$ is always of very special form.
  - Random Oracle Model
    - If model $H$ as a "public" truly random function $H : \{0,1\}^* \to \mathbb{Z}_N^*$, then can justify signature based on one-wayness of RSA.
    - But no $H$ is a truly random function. Heuristically, it is hoped that some cryptographic hash functions (e.g. SHA) "behave like" a truly random function.
- El Gamal signatures
  - $pk = (p, g, \hat{x})$, $sk = (p, g, x)$ where $\hat{x} = g^x \bmod p$.
  - $\mathsf{Sign}_{sk}(m)$:
    1. Choose $y \xleftarrow{R} \mathbb{Z}_{p-1}$, let $\hat{y} = g^y \bmod p$.
    2. Find $s$ s.t. $g^m \equiv g^{x\hat{y}} \cdot \hat{y}^s \pmod{p}$.
    3. Output $\sigma = (\hat{y}, s)$.

    How does the signer compute $s$? We have $m \equiv x\hat{y} + ys \pmod{(p-1)}$ and the signer knows $X, y, \hat{y}, m$ so he can compute $(m - x\hat{y})y^{-1}$ modulo $(p-1)$.
  - $\mathsf{Vrfy}_{pk}(m, (y, s))$: Check that $g^m \equiv \hat{x}^y \cdot y^s \pmod{p}$.
  - Not secure! (Similar attacks as plain trapdoor functions.)
  - Digital Signature Standard (1991, NIST+NSA): hash-then-sign, using SHA and DSA (variant of El Gamal).

    * No proof of security (even in an idealized model).

- Bridging the gap...
  - Heuristic proofs in Random Oracle Model (as mentioned above).,
  - More efficient, truly provable signatures based on specific hard problems: Strong RSA (given $N, z$, find $y, e > 1$ s.t. $y^e \equiv z \pmod{N}$).

# 2 Theoretical Constructions

Digital signatures can be constructed from one-way functions (no trapdoors!). Signing is deterministic. This result is beyond the scope of this class. Instead we will outline a provably secure construction from collision-resistant hash functions.

## 2.1 One-time signature

We start by showing how to construct a *one-time* signature from any one-way function $f$. That means it is unforgeable for adversaries that get to make only one signing query in the unforgeability game.

**Single bit message.** Let $f$ be any one-way function.

1. The secret key is $sk = \{x_0, x_1\}$ where $x_0$ and $x_1$ are chosen at random in $\{0,1\}^k$. The public key is $pk = \{y_0, y_1\}$ where $y_0 = f(x_0)$ and $y_1 = f(x_1)$.
2. The signature of the bit $b$ is $\mathsf{Sign}_{sk}(b) = x_b$.
3. The verification is $\mathsf{Vrfy}_{pk}(b, x) = \texttt{accept}$ iff $f(x) = y_b$.

This scheme is not very efficient and gives away half of the secret key. But it is secure if the adversary asks only one query. Assume that he obtains the signature of 0, i.e. $x_0$. Then the adversary cannot produce the signature of 1, i.e. invert $y_1$, because $x_0$ and $x_1$ are chosen independently so seeing $x_0$ does not help him to invert $y_1$.

**Long messages:** Can repeat above scheme $\ell$ times to sign $\ell$-bit messages, but lengths of keys and signatures grow linearly with $\ell$. How can we avoid this blow-up?

## 2.2 One-time signatures to many-time signatures

How do we obtain a signature scheme for multiple messages?

- Idea: generate new keys "on the fly" and authenticate them with previous key.
- Start with a one-time signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$, and construct general signature scheme as follows:
  - Public key $pk_0$, secret key $sk_0$ for original scheme.
  - To sign first message $m_1$: Generate $(pk_1, sk_1) \xleftarrow{R} \mathsf{Gen}(1^k)$. Let $\sigma_1 \xleftarrow{R} \mathsf{Sign}_{sk_0}(m_1, pk_1)$. Output $\sigma_1' = (1, \sigma_1, m_1, pk_1)$.

  - To sign second message $m_2$: Generate $(pk_2, sk_2) \xleftarrow{R} \mathsf{Gen}(1^k)$. Let $\sigma_2 \xleftarrow{R} \mathsf{Sign}_{sk_1}(m_2, pk_2)$. Output $\sigma_2' = (2, \sigma_1', \sigma_2, m_2, pk_2)$.
  - etc.

- This works, but very inefficient.
- Improvements:
  - Use a tree — each key authenticates two new keys ⇒ signature length, verification time, signing time no longer grow linearly w/ number of signatures.
  - Can also make it stateless.
  - Can construct secure signatures based on any one-way function (very complicated!).
  - See Katz-Lindell.

# 3 Implementing a PKI

As discussed earlier, digital signatures are very useful in implementing a *public-key infrastructure* (PKI). If everyone has the public key of a trusted certificate authority $C$, then $C$ can issue certificates asserting the validity of other parties' public keys, e.g. $\mathrm{cert}_{C\to B} = (pk_B, \mathsf{Sign}_{sk_C}(\text{``}pk_B \texttt{ is Bob's public key''}))$. There can be multiple CA's and certificate chains like $(\mathrm{cert}_{C\to B}, \mathrm{cert}_{B\to A})$ (which certifies $A$'s public key to someone who knows $C$'s public key and trusts $C$).

What are some challenges in implementing a PKI?

- One trusted party (*certificate authority*, e.g. Verisign) has a public key known to everyone, and signs individual's public keys, e.g. signs statement like

$$m = \texttt{`Verisign certifies that } pk_{\mathrm{Alice}} \texttt{ is Alice's public key'}$$

  When starting a communication with a new party, Alice sends the *certificate*

$$(pk_{\mathrm{Alice}}, m, \mathsf{Sign}_{sk_{\mathrm{Verisign}}}(m))$$

- Many variants: Can be done hierarchically (Verisign signs Harvard's PK, Harvard signs individual students' public keys).