Digital Signatures

Debdeep Mukhopadhyay
IIT Kharagpur

What are Signature Schemes?

- Provides message integrity in the public key setting
- Counter-parts of the message authentication schemes in the public setting
- Allow a signer S who has established a public key, pk, can sign a message with his own secret key.
- Anybody who knows pk, and knows that the public key was originates by S, can verify the signature.
  - Several applications
  - Distribution of patches by a software company.

DSA vs MAC

- Both are used for integrity.
- Verification of MACs rely on the private key setting.
- However verification of the DSA is based on public key setting.
  - signatures are publicly verifiable.
  - signatures are transferable.
  - signatures provide non-repudiation.

Definition

A signature scheme is a tuple of three PPT algorithms: (Gen,Sign,Vrfy) satisfying the following:

1. The key-generation algorithm Gen takes as input a security parameter n, and outputs a pair of keys (pk,sk). pk is the public key and sk is the secret key. Assume both have length n.
2. The signing algorithm Sign, takes as input a private key sk and a message m $\in \{0,1\}^*$. It outputs a signature σ, denoted as $\sigma \leftarrow \text{Sign}_{sk}(m)$.
3. The deterministic verification algorithm Vrfy takes as input a public key pk, and a message m, and a signature σ. It outputs a bit b=1, meaning valid, and b=0 meaning invalid. We denote this as b=$\text{Vrfy}_{pk}(m,\sigma)$

Correctness of a signature scheme

It is required that for every n, every (pk,sk) output by Gen, and every message $m \in \{0,1\}^*$, it holds that:

$$\text{Vrfy}_{pk}(m, Sign_{sk}(m)) = 1$$

Security

The signature experiment $\text{Sig-forge}_{A,\Pi}(n)$:

1. Gen is run to obtain keys (pk,sk)
2. Adversary A is given pk and oracle access to $\text{Sign}_{sk}()$. The oracle returns a signature $\text{Sign}_{sk}(m)$ for any message m of the adversaries choice.

The adversary after Q requests outputs a pair (m, σ).

3. The output of the experiment is denoted to be 1 if and only if, 1)$\text{Vrfy}_{pk}(m,\sigma) = 1$, and 2)$m \notin Q$

A signature scheme Π = (Gen,Sign,Vrfy) is existentially unforgeable under an adaptive chosen message attack if for all PPT adversaries A, there exists a negligible func negl st:

$$\Pr[\text{Sig-forge}_{A,\Pi}(n)=1] \leq \text{negl}$$

RSA based Signatures

Define RSA-sign(n):

1. Gen(n): Outputs (N,e,d), where N=pq, where p and q are both n bit primes, $ed \equiv 1 \bmod \Phi(N)$.
2. Sign: On input a private key sk=(N,d), and a message $m \in Z_N^*$,

$$\sigma = m^d \bmod N$$

3. Vrfy: On input a public key pk=(N,e), and a message $m \in Z_N^*$, and a signature scheme $\sigma \in Z_N^*$, output 1 if and only if:

$$m = \sigma^e \bmod N$$

A no-message attack

- It is trivial to forge without any query at all. How?

A no-message attack

- It is trivial to forge without any query at all.

Just choose an arbitrary σ, and compute

$$m = \sigma^e \bmod N.$$

It is immediately clear that (m, σ) is always valid!!

Forging a signature on an arbitrary message

- Say the adversary wants to output a forgery of any given message m.
- The adversary just needs two signatures of chosen messages.
- How does the forgery work?

The forgery

- Adversary chooses an arbitrary m1 and obtains its sign σ1.
- It computes, m2=m/m1, and its sign σ2.
- Now note, any valid sign for m, is

$$\sigma = m^d = (m1.m2)^d$$
$$= m1^d.m2^d$$
$$= (\sigma 1.\ \sigma 2) \bmod N.$$

Question

- How many forgeries can you create with t such signature values?

Hashed RSA

- The basic idea is to modify the textbook RSA by applying some hash function H to the message before signing.
- The scheme considers a publicly known function:

$$H : \{0,1\}^* \rightarrow Z_N^*$$

- The sign σ is computed from m, as follows:

$$\sigma = [H(m)]^d \bmod N$$

The function H should be collision resistant

- The function H must be collision resistant, as otherwise one can find two messages, $m \neq m1$, st $H(m)=H(m1)$.
  - then creating a forgery is trivial.

Attacks on the hashed RSA scheme

- No message attack: Is difficult, if H is difficult to invert.

- Forging a signature on arbitrary messages:

  For the previous attack for textbook RSA to work now, we need to find three messages, m, m1 and m2 st: H(m)=H(m1).H(m2) mod N.

  This seems to be difficult if H is not efficiently invertible. Proofs of these schemes exploit that the function H is a randomly looking function: This proof models are called Random Oracle models.

The Hash and Sign Paradigm

- Apart from preventing the attacks on the RSA-sign scheme, there is another advantage:
  - it can be used for signing messages of arbitrary lengths.
  - general approach is to hash and then sign the message.
- Of course the following theorem does not apply for RSA-sign, as it is not secure itself.

Hash and Sign

$\Pi = (\mathsf{Gen}_S, \mathsf{Sign}, \mathsf{Vrfy})$, $\Pi_H = (\mathsf{Gen}_H, H)$. *A signature scheme* Π'*:*

- Gen'*: on input* 1^n *run* $\mathsf{Gen}_S(1^n)$ *to obtain* (pk, sk)*, and run* $\mathsf{Gen}_H(1^n)$ *to obtain* s*. The public key is* $pk' = \langle pk, s\rangle$ *and the private key is* $sk' = \langle sk, s\rangle$*.*
- Sign'*: on input* sk' *and* $m \in \{0,1\}^*$*,* $\sigma \leftarrow \mathsf{Sign}_{sk}(H^s(m))$*.*
- Vrfy'*: on input* pk'*,* $m \in \{0,1\}^*$ *and* σ*, output 1* $\iff$ $\mathsf{Vrfy}_{pk}(H^s(m), \sigma) = 1$*.*

If Π *is existentially unforgeable under an adaptive CMA and* Π_H *is collision resistant, then Construction is existentially unforgeable under an adaptive CMA.*

Hash and Sign

Idea: a forgery must involve either finding a collision in H or forging a signature with respect to Π.

Proof.

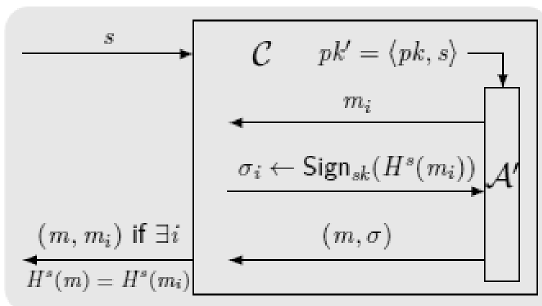
$\mathcal{A}'$ attacks Π' and output (m, σ), $m \notin \mathcal{Q}$.
SF: $\mathsf{Sigforge}_{\mathcal{A}',\Pi'}(n) = 1$.
coll: $\exists m' \in \mathcal{Q}$, $H^s(m') = H^s(m)$.

$$\Pr[\mathsf{SF}] = \Pr[\mathsf{SF} \wedge \mathsf{coll}] + \Pr[\mathsf{SF} \wedge \overline{\mathsf{coll}}] \leq \Pr[\mathsf{coll}] + \Pr[\mathsf{SF} \wedge \overline{\mathsf{coll}}].$$

Reduce $\mathcal{C}$ for Π_H to $\mathcal{A}'$. $\Pr[\mathsf{coll}] = \Pr[\mathsf{Hashcoll}_{\mathcal{C},\Pi_H}(n) = 1]$.
Reduce $\mathcal{A}$ for Π to $\mathcal{A}'$. $\Pr[\mathsf{SF} \wedge \overline{\mathsf{coll}}] = \Pr[\mathsf{Sigforge}_{\mathcal{A},\Pi}(n) = 1]$.
So both $\Pr[\mathsf{coll}]$ and $\Pr[\mathsf{SF} \wedge \overline{\mathsf{coll}}]$ are negligible. □

Reduction 1

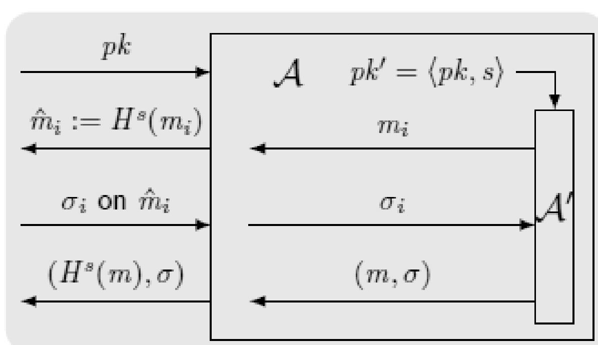
Reduce $\mathcal{C}$ for Π_H to $\mathcal{A}'$. $\mathcal{A}'$ queries the signature σ_i of i-th message m_i, $i = 1, \ldots, |\mathcal{Q}|$.



$\Pr[\mathsf{coll}] = \Pr[\mathsf{Hashcoll}_{\mathcal{C},\Pi_H}(n) = 1]$.

Reduction 2

Reduce $\mathcal{A}$ for Π to $\mathcal{A}'$.



$\Pr[\mathsf{SF} \wedge \overline{\mathsf{coll}}] = \Pr[\mathsf{Sigforge}_{\mathcal{A},\Pi}(n) = 1]$.

Lamport's One Time Signatures

The one-time signature (OTS) experiment $\mathsf{Sigforge}^{\text{1-time}}_{\mathcal{A},\Pi}(n)$:

1. $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$.
2. $\mathcal{A}$ is given input 1^n and a single query m' to $\mathsf{Sign}_{sk}(\cdot)$, and outputs (m, σ), $m \neq m'$.
3. $\mathsf{Sigforge}^{\text{1-time}}_{\mathcal{A},\Pi}(n) = 1 \iff \mathsf{Vrfy}_{pk}(m, \sigma) = 1$.

A signature scheme Π is **existentially unforgeable under an adaptive CMA** if $\forall$ PPT $\mathcal{A}$, $\exists$ negl such that:

$$\Pr[\mathsf{Sigforge}^{\text{1-time}}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

Construction of Lamport's OTS

f is a one-way function (OWF).

- Gen: *on input 1^n, for $i \in \{1, \ldots, \ell\}$:*
  1. *choose random $x_{i,0}, x_{i,1} \leftarrow \{0,1\}^n$.*
  2. *compute $y_{i,0} := f(x_{i,0})$ and $y_{i,1} := f(x_{i,1})$.*

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \quad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}.$$

- Sign: *on input sk and $m \in \{0,1\}^\ell$ with $m = m_1 \cdots m_\ell$, output $\sigma = (x_{1,m_1}, \ldots, x_{\ell,m_\ell})$.*
- Vrfy: *on input pk, $m \in \{0,1\}^\ell$ with $m = m_1 \cdots m_\ell$ and $\sigma = (x_1, \ldots, x_\ell)$, output $1 \iff f(x_i) = y_{i,m_i}$, for all i.*

Example

Signing $m = 011$

$$sk = \begin{pmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{pmatrix} \implies \sigma = (x_{1,0}, x_{2,1}, x_{3,1})$$

$\sigma = (x_1, x_2, x_3)$:

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{pmatrix} \implies \begin{matrix} f(x_1) \stackrel{?}{=} y_{1,0} \\ f(x_2) \stackrel{?}{=} y_{2,1} \\ f(x_3) \stackrel{?}{=} y_{3,1} \end{matrix}$$

Security Proof

If f is a OWF, then Construction Π is a OTS for messages of length polynomial ℓ.

Idea: If $m \neq m'$, then $\exists i^*, m_{i*} = b^* \neq m'_{i*}$. So to forge a signature on m can invert a single y_{i^*,b^*} at least.

Proof.

Reduce $\mathcal{I}$ inverting y to $\mathcal{A}$ attacking Π:

1. Construct pk: Choose $i^* \leftarrow \{1, \ldots, \ell\}$ and $b^* \leftarrow \{0, 1\}$, set $y_{i^*,b^*} := y$. For $i \neq i^*$, $y_{i,b} := f(x_{i,b})$.
2. $\mathcal{A}$ signs m' with pk: If $m'_{i_*} = b^*$, stop. Otherwise, return σ'.
3. When $\mathcal{A}$ outputs (m, σ), $\sigma = (x_1, \ldots, x_\ell)$, if $\mathcal{A}$ output a forgery at (i^*, b^*): $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$ and $m_{i^*} = b^* \neq m'_{i^*}$, then output x_{i^*,b^*}.

$$\Pr[\mathcal{I} \text{ succeeds}] \geq \frac{1}{2\ell} \Pr[\mathcal{A} \text{ succeeds}]$$

□

Stateful Signature Scheme

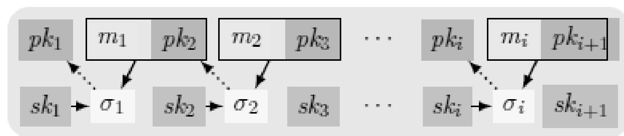
A stateful signature scheme:

- **Key-generation** algorithm $(pk, sk, s_0) \leftarrow \mathsf{Gen}(1^n)$. s_0 is initial state.
- **Signing** algorithm $(\sigma, s_i) \leftarrow \mathsf{Sign}_{sk, s_{i-1}}(m)$.
- **Verification** algorithm $b := \mathsf{Vrfy}_{pk}(m, \sigma)$.

A Simple Scheme

A simple stateful signature scheme for OTS:
Generate (pk_i, sk_i) independently, set $pk := (pk_1, \ldots, pk_\ell)$ and $sk := (sk_1, \ldots, sk_\ell)$.
Start from the state 1, sign the s-th message with sk_s, verify with pk_s, and update the state to $s + 1$.
Weakness: the upper bound ℓ must be fixed in advance.

Chain-based Signatures

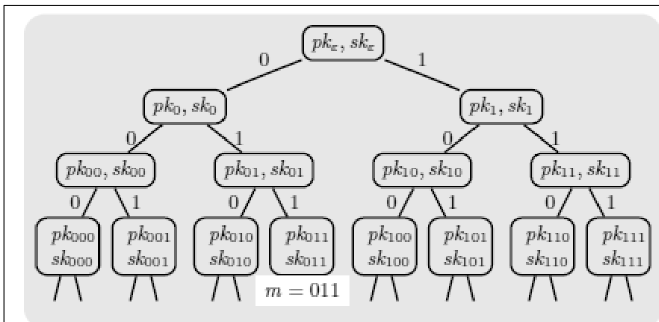


Use a single public key pk_1, sign each m_i and pk_{i+1} with sk_i:

$$\sigma_i \leftarrow \mathsf{Sign}_{sk_i}(m_i \| pk_{i+1}),$$

output $\langle pk_{i+1}, \sigma_i \rangle$, and verify σ_i with pk_i.
The signature is $(pk_{i+1}, \sigma_i, \{m_j, pk_{j+1}, \sigma_j\}_{j=1}^{i-1})$.
Weakness: stateful, not efficient, revealing all previous messages.

Tree based Signature Schemes



- the root is label by ε (empty string), each node is labeled by a string w, the left-child $w0$ and the right-child $w1$.
- the leaf is a message m, and the internal nodes are (pk_w, sk_w), where w is the prefix of m.
- each node pk_w "certifies" its child node(s) $pk_{w0} \| pk_{w1}$ or w.

The signature scheme

- It first generates keys (as needed) for all nodes on the path from root to the leaf labeled m.
  - some of these public keys may have been generated during signing previous messages, they are not generated again.
  - it certifies the path from the root to the leaf labeled as m by computing a signature on $pk_{w0}||pk_{w1}$, using secret key sk_w, for each string w that is a proper prefix of m.
  - finally, it certifies m by computing a signature on m with the private key sk_m.

The tree based signature algorithm

$\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$. *For a binary string* m, $m|_i \stackrel{\text{def}}{=} m_1 \cdots m_i$ *denote the* i*-bit prefix of* m. $\Pi^* = (\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Vrfy}^*)$:

- Gen^*: *on input* 1^n, *compute* $(pk_\varepsilon, sk_\varepsilon) \leftarrow \mathsf{Gen}(1^n)$ *and output the public key* pk_ε. *The private key and initial sate are* sk_ε.
- Sign^*: *on input* $m \in \{0,1\}^n$,
  1. *for* $i = 0$ *to* $n-1$: *compute* $(pk_{m|_i 0}, sk_{m|_i 0}) \leftarrow \mathsf{Gen}(1^n)$, $(pk_{m|_i 1}, sk_{m|_i 1}) \leftarrow \mathsf{Gen}(1^n)$, $\sigma_{m|_i} \leftarrow \mathsf{Sign}_{sk_{m|_i}}(pk_{m|_i 0} \| pk_{m|_i 1})$, *if these values are not in the state, and add them to the state.*
  2. *compute* $\sigma_m \leftarrow \mathsf{Sign}_{sk_m}(m)$, *if it is not in the state, add it.*
  3. *output* $\sigma = (\{\sigma_{m|_i}, pk_{m|_i 0}, pk_{m|_i 1}\}_{i=0}^{n-1}, \sigma_m)$.
- Vrfy^*: *on input* $pk_\varepsilon, m, \sigma$, *output 1* $\iff$
  1. $\mathsf{Vrfy}_{pk_{m|_i}}(pk_{m|_i 0} \| pk_{m|_i 1}, \sigma_{m|_i}) \stackrel{?}{=} 1$ *for all* $i \in \{0, \ldots, n-1\}$.
  2. $\mathsf{Vrfy}_{pk_m}(m, \sigma_m) \stackrel{?}{=} 1$.

Security

Π is a OTS. Construction Π^* is a secure digital signature scheme.

Idea: Reduce $\mathcal{A}$ for OTS Π to $\mathcal{A}^*$ for "tree-based" Π^*.
$\mathcal{A}^*$ queries $\ell^* = \ell^*(n)$ times, $\ell = \ell(n) = 2n\ell^* + 1$.
$\mathcal{A}$ is given input pk, generates a list of ℓ key pairs with i^*-th node pk inserted randomly. $\mathcal{A}$ runs $\mathcal{A}^*$ as a subroutine, and replies the queries from $\mathcal{A}^*$ with the list of keys. If $\mathcal{A}^*$ outputs a forgery on m, then there is one node i, for which the signature of its child C is forged, on the path from the root to m. If $i = i^*$ (with probability $\frac{1}{\ell}$), then $\mathcal{A}$ outputs a forgery on C.

$$\Pr[\mathsf{Sigforge}^{\text{1-time}}_{\mathcal{A},\Pi}(n) = 1] = \Pr[\mathsf{Sigforge}_{\mathcal{A}^*,\Pi^*}(n) = 1]/\ell(n)$$

Stateless scheme

- The states depend on the message signed.
- It is possible to generate all needed keys in the entire tree in advance, but the time complexity is exponential.

A Stateless Solution

Idea: use deterministic randomness to emulate the state of tree.

Use a PRF F and two keys k, k' (secrets). Generating pk_w, sk_w in 2 steps:

1. compute $r_w := F_k(w)$.
2. compute $(pk_w, sk_w) :=$ Gen$(1^n; r_w)$, using r_w as random coins.

k' is used to generate r'_w that is used to compute σ_w.

Digital Signature Algorithm

A PPT $\mathcal{G}$ *is on input* 1^n, *outputs* (p, q, g) *except with negligible probability: (1)* p *and* q *are primes with* $\|q\| = n$; *(2)* $q|(p-1)$ *but* $q^2 \nmid (p-1)$; *(3)* g *is a generator of the subgroup of* $\mathbb{Z}_p^*$ *of order* q.

- Gen: *on input* 1^n, *run* $(p, q, g) \leftarrow \mathcal{G}$. *A hash function* $H : \{0,1\}^* \to \mathbb{Z}_q$. *Choose* $x \leftarrow \mathbb{Z}_q$ *and set* $y := [g^x \bmod p]$. $pk = \langle H, p, q, g, y \rangle$. $sk = \langle H, p, q, g, x \rangle$.
- Sign: *on input* sk *and* $m \in \{0,1\}^*$, *choose* $k \leftarrow \mathbb{Z}_q^*$ *and set* $r := [[g^k \bmod p] \bmod q]$, $s := [(H(m) + xr) \cdot k^{-1} \bmod q]$. *Output a signature* (r, s).
- Vrfy: *on input* pk, $m \in \{0,1\}^*$, (r, s), $r \in \mathbb{Z}_q$, $s \in \mathbb{Z}_q^*$. $u_1 := [H(m) \cdot s^{-1} \bmod q]$, $u_2 := [r \cdot s^{-1} \bmod q]$. *Output* $1 \iff r \stackrel{?}{=} [[g^{u_1} y^{u_2} \bmod p] \bmod q]$.

Correctness

$r = [[g^k \bmod p] \bmod q]$ and $s = [(\hat{m}+xr)\cdot k^{-1} \bmod q]$, $\hat{m} = H(m)$.

$$\begin{aligned} g^{\hat{m}s^{-1}} y^{rs^{-1}} &= g^{\hat{m}\cdot(\hat{m}+xr)^{-1}k} g^{xr\cdot(\hat{m}+xr)^{-1}k} \pmod p \\ &= g^{(\hat{m}+xr)\cdot(\hat{m}+xr)^{-1}k} \pmod p \\ &= g^k \pmod p. \end{aligned}$$

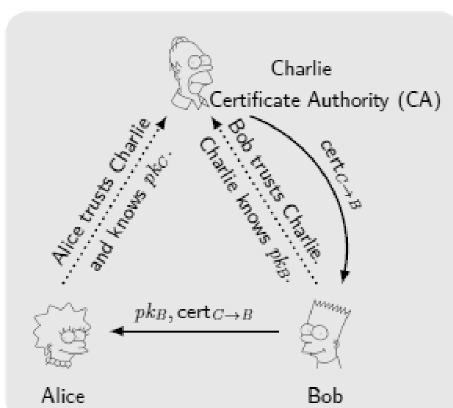
$$[[g^k \bmod p] \bmod q] = r.$$

- DSS uses the Digital Signature Algorithm (DSA).
- Security of DSS relies on the hardness of discrete log problem.

Insecurity

There is no proof of security for DSS based on discrete log assumption.

Certificates



Certificates cert$_{C\to B} \stackrel{\text{def}}{=}$ Sign$_{sk_C}$('Bob's key is pk_B').

Public Key Infrastructure (ISA)

- **A single CA**: is trusted by everybody and issues certificates.
  - Strength: simple
  - Weakness: single-point-of-failure
- **Multiple CAs**: are trusted by everybody and issue certificates.
  - Strength: robust
  - Weakness: cannikin law
- **Delegation and certificate chains**: The trust is transitive.
  - Strength: ease the burden on the root CA.
  - Weakness: difficult for management, cannikin law.
- **"Web of trust"**: No central points of trust, e.g. PGP.
  - Strength: robust, work at "grass-roots" level.
  - Weakness: difficult to manage/give a guarantee on trust.

Invalidating

- **Expiration**: include an *expiry date* in the certificate.

$$\text{cert}_{C \to B} \stackrel{\text{def}}{=} \text{Sign}_{sk_C}(\text{'bob's key is } pk_B\text{', date}).$$

- **Revocation**: explicitly revoke the certificate.

$$\text{cert}_{C \to B} \stackrel{\text{def}}{=} \text{Sign}_{sk_C}(\text{'bob's key is } pk_B\text{', } \#\#\#).$$

"###" represents the serial number of this certificate. When CA want to revoke certificates, CA generates a *certificate revocation list* (CRL) containing the serial numbers of all revoked certificates, signs the CRL along with the current data, and distributes it widely.