Digital Signatures

Chanathip Namprempre

Department of Computer Science
Reed College

Outline

Syntax of digital signature schemes

Syntax

A digital signature scheme $DS = (\mathrm{KG}, Sign, VF)$ is a triple of algorithms.

alg	input	output	notation	maybe randomized?	maybe stateful?
$\mathcal{K}$	-	key pk, sk	$(pk, sk) \xleftarrow{\$} \mathrm{KG}$	yes	no
Sign	$(pk, sk) \in Keys(\mathrm{DS})$ $M \in \{0,1\}^*$	signature $\sigma \in \{0,1\}^* \cup \{\perp\}$	$\sigma \xleftarrow{\$} \mathrm{Sign}_{sk}(M)$	yes	yes
VF	$(pk, sk) \in Keys(\mathrm{DS})$ $M, \sigma \in \{0,1\}^*$	message $b \in \{0,1\}$	$M \leftarrow \mathrm{VF}_{pk}(M, \sigma)$	no	no

Correctness

For all $(pk, sk) \in Keys(\mathrm{DS}), M \in \{0,1\}^*$,

$$\Pr\left[\sigma = \perp \text{ OR } \mathrm{VF}_{pk}(M, \sigma) = 1 \;:\; \sigma \xleftarrow{\$} \mathrm{Sign}_{sk}(M)\right] = 1\,.$$

Observations

1. Even the receiver cannnot forge.
2. The verifier does not need to have any secrets.
3. For this to work, the verifier VF must have authentic pk!
4. Usage of keys is the mirror image of that of asymmetric encryption.

Outline

Unforgeability against CMA

<u>Idea</u>: Same as *MAC* except we give the forger the public key.

Let $\mathsf{DS} = (\mathsf{KG}, \mathsf{Sign}, \mathsf{VF})$ be a DS scheme, and let A be an adversary.

Subroutine $\texttt{Initialize}$ $b \xleftarrow{\$} \{0,1\}$; $(pk, sk) \xleftarrow{\$} \mathsf{KG}$ $S \leftarrow \emptyset$ Return pk Subroutine $\texttt{Sign}(M)$ $\sigma \xleftarrow{\$} \mathsf{Sign}_{sk}(M)$ $S \leftarrow S \cup \{M\}$ Return σ Subroutine $\texttt{Finalize}(M, \sigma)$ $d \leftarrow \mathsf{VF}(pk, M, \sigma)$ Return $(d = 1 \wedge M \notin S)$	Experiment $\mathbf{Exp}^{\text{wuf-cma}}_{\mathsf{DS}}(A)$ $pk \xleftarrow{\$} \texttt{Initialize}$ $(M, \sigma) \xleftarrow{\$} A^{\texttt{Sign}}(pk)$ Return $\texttt{Finalize}(M, \sigma)$

wuf-cma advantage of A mounting a CMA against DS:

$$\mathbf{Adv}^{\text{wuf-cma}}_{\mathsf{DS}}(A) = \Pr\left[\, \mathbf{Exp}^{\text{wuf-cma}}_{\mathsf{DS}}(A) \Rightarrow \text{true} \,\right] .$$

Digital Signatures: observations about security definition

observations

1. for MAC, we give A both $MAC_K(\cdot)$ and $VF_K(\cdot,\cdot)$
2. resources :
   - t = running time
   - μ = sum of lengths of oracle queries plus length of message in forgery
   - q = number of queries to signing oracle.

Outline

RSA trapdoor signatures

Key generation : use $K_{rsa} \rightarrow (N, e), (N, p, q, d)$ such that

$$ed \equiv 1 \bmod \phi(N)$$
$$e, d \in \mathbf{Z}^*_{\phi(N)}$$
$$N = p \cdot q$$

We know

1. $RSA_{N,e}(\cdot)$ is easy
2. $RSA_{N,d} = RSA^{-1}_{N,e}$
3. without d, $RSA^{-1}_{N,e}$ is hard

So to sign M : assume $M \in \mathbf{Z}^*_N$
$\sigma \leftarrow RSA_{N,d}(M)$ [invert RSA on point M]

Scheme: textbook RSA signature

$Sign_{N,p,q,d}(M)$	$VF_{N,e}(M, x)$
If $M \notin \mathbf{Z}_N^*$ then $\perp$	If $(M \notin \mathbf{Z}_N^*$ or $x \notin \mathbf{Z}_N^*)$ then return 0
$x \leftarrow M^d \bmod N$	If $M = x^e \bmod N$ then return 1 else return 0
Return x	

Above, notice

1. *Sign* is deterministic and stateless
2. $MsgSp(N, e) = \mathbf{Z}_N^*$
3. correctness condition : pass since $RSA_{N,e}^{-1} = RSA_{N,d}$
   So $x = M^d$ and $x^e = M^{ed} = M$ ok

BUT Textbook RSA signature scheme is insecure!

Breaking textbook RSA signature scheme

Forger F1

idea: just outputs $(1, 1)$
$VF_pk(1, 1)$: if $1 = 1^e$ mod N then return 1 else return 0

Forger F2

idea: just pick x first, then compute the message M

$F^{Sign_sk(\cdot)}(N, e)$
$\quad x \xleftarrow{\$} \mathbf{Z}_N^*$
$\quad M \leftarrow x^e \bmod N$
$\quad$ return (M, x)

The verification algorithm VF will check whether $M = x^e$.
So VF returns 1.

Breaking textbook RSA signature scheme (cont.)

Forger F3

We can even forge any given message M!

$$
\begin{aligned}
&F^{Sign_s k(\cdot)}(N, e): \\
&\quad M_1 \xleftarrow{\$} \mathbf{Z}_N^* - \{1, M\} \\
&\quad M_2 \leftarrow MM_1^{-1} \bmod N \\
&\quad x_1 \leftarrow Sign_s k(M_1)\,;\; x_2 \leftarrow Sign_s k(M_2) \\
&\quad x \leftarrow x_1 x_2 \bmod N \\
&\quad \text{Return } (M, x)
\end{aligned}
$$

Bottom line

There's more to signatures than one-wayness of the underlying function!

Observations

- From attacks we have seen, RSA function is homomorphic, i.e.
  $$M^d = M_1^d M_2^d \text{ when } M = M_1 M_2$$
- Also, messages usually aren't group elements.
- To deal with these problems, we add a pre-processing step: Hash messages into $\mathbf{Z}_N^*$ first.

Outline

Hash-then-invert paradigm

scheme

Let K_{rsa} be an RSA generator with security parameter k.
Let *Keys* be the set of all moduli N that can be output by K_{rsa}.
Let *Hash* be a family of functions whose key space is *Keys* and $\forall N \in Keys,\ Hash_N : \{0,1\}^* \rightarrow \mathbf{Z}_N^*$.
Let $DS = (K_{rsa}, Sign, VF)$ be the digital signature scheme defined as

$$
\begin{array}{l|l}
Sign_{N,p,q,d}(M) & VF_{N,e}(M,x) \\
 & \\
y \leftarrow Hash_N(M) & y \leftarrow Hash_N(M) \\
x \leftarrow y^d \bmod N & y' \leftarrow x^e \bmod N \\
\text{Return } x & \text{If } y = y' \text{ then return 1 else return 0}
\end{array}
$$

How this scheme prevents attacks we have seen

Recall Forger F1

idea: just outputs $(1, 1)$
$VF_pk(1, 1)$: if $1 = 1^e$ mod N then return 1 else return 0

This works when $Hash_N(1) \equiv 1^e \pmod{N}$.

So we make sure that $Hash_N(1) \not\equiv 1 \pmod{N}$.

How this scheme prevents attacks we have seen (cont.)

Recall Forger F2

idea: just pick x first, then compute the message M

$F^{Sign_s k(\cdot)}(N, e)$

$x \xleftarrow{\$} \mathbf{Z}_N^*$

$M \leftarrow x^e \bmod N$

return (M, x)

For this to work, need M such that

$$Hash_N(M) \equiv x^e \pmod{N}$$

If *Hash* is "good," it is hard to find such M that works.

How this scheme prevents attacks we have seen (cont.)

Recall Forger F3

We can even forge any given message M!

$F^{Sign_s k(\cdot)}(N, e)$:

$M_1 \xleftarrow{\$} \mathbf{Z}_N^* - \{1, M\}$
$M_2 \leftarrow MM_1^{-1} \bmod N$
$x_1 \leftarrow Sign_s k(M_1)$; $x_2 \leftarrow Sign_s k(M_2)$
$x \leftarrow x_1 x_2 \bmod N$
Return (M, x)

For this to work, we need

$$Hash_N(M_1) \cdot Hash_N(M_2) = Hash_N(M).$$

With a “good” hash function, this is rare.

Bottom line

The hash function destroys the algebraic structure needed for the attacks to work.

BUT we also need collision-resistance!

Otherwise, we can attack.

Attack against hash-then-invert scheme if *Hash* is bad

Suppose M_1, M_2 be messages such that $\exists N$,

$$Hash_N(M_1) \equiv Hash_N(M_2) \pmod{N}$$

Then, we can forge signing algorithm when modulus is N as follows:

Forger F

Forger $F^{Sign_{N,p,q,d}(\cdot)}(N, e)$:

$x_1 \leftarrow Sign_{N,p,q,d}(M_1)$
Return (M_2, x_1)

Why does this work?

Properties we need from *Hash* for the hash-then-invert paradigm

Necessary properties of Hash are at least

- destroy algebraic properties of the messages
- CR2-KK
- ???

We want sufficient conditions!
So we need provable security.
But first, let's consider some candidate hash functions.

PKCS # 1 signature scheme

$$PKCS\text{-}Hash_N(M) = 0001\ FFFF \cdots FF00 || h(M)\ [k \text{ bits}]$$

where $h : \{0,1\}^* \rightarrow \{0,1\}^l$ and $l \geq 128$ and
h is assumed to be collision-resistant and
$k = |N|$.

(In practice, $h = SHA1(l = 160)$. Used to be $h = MD5(l = 128)$.)

Notice :

1. First 4 bits are 0.
   So as an int, $PKCS\text{-}Hash_N(\cdot) \leq N$
2. Most #s between 1 and N are in $\mathbf{Z}_N^*$.
   (There are $((p-1)(q-1)$ of them to be exact.)
3. If h is collision-resistant, then so is *PKCS-Hash*.

Would hash-then-invert with *PKCS-Hash* work ??

PKCS-Hash

PKCS-Hash seems to destroy the algebraic properties of messages, i.e.

- hard to imagine
  $$PKCS\text{-}Hash(M) = PKCS\text{-}Hash(M_1) \cdot PKCS\text{-}Hash(M_2)$$
- *PKCS-Hash* seems collision-resistant.
- BUT there's a cause of concern.
  - We assume $RSA_{N,e}$ is one-way.
  - Q: what do we invert $RSA_{N,e}$ on?

    $Sign_{N,p,q,d}(M)$
    $y \leftarrow PKCS\text{-}Hash_N(M)$
    $x \leftarrow y^d \bmod N$
    Return x

    A: We invert $RSA_{N,e}$ on output points of *PKCS-Hash*.

Security of PKCS # 1

Let S_N be the set of these points, i.e.

$$S_N = \{PKCS\text{-}Hash_N(M) : M \in \{0,1\}^*\} \ .$$

So we want $RSA_{N,e}$ to be hard to invert on points in S_N!

Let's compare the size of S_N to the size of $\mathbf{Z}_N^*$.

- $|S_N| \leq 2^{160}$
  [Front part is fixed and
  $h : \{0,1\}^* \to \{0,1\}^l$(for $SHA1 : l = 160$)]
- Recommended size for modulus is 1024.
  So $|\mathbf{Z}_N^*| \simeq 2^{1023}$.
- So $\frac{|S_N|}{|\mathbf{Z}_N^*|} \leq \frac{2^{160}}{2^{1023}} = \frac{1}{2^{863}}$

S_N is <u>much much smaller</u> than $\mathbf{Z}_N^*$!
Bottom line: $RSA_{N,e}$ <u>could be</u> hard to invert in $\mathbf{Z}_N^*$ but easy in S_N!

Full-Domain-Hash (FDH) [BR96]

To address this problem, the hash should map inputs into the entire domain, i.e.,

FDH

$$H : \{0,1\}^* \to \mathbf{Z}_N^*$$

$Sign^H_{N,p,q,d}(M)$	$VF^H_{N,e}(M,x)$
$x \leftarrow H(M)^d \bmod N$	If $H(M) = x^e \bmod N$ then return 1
Return x	else return 0

FDH has been proven secure in the random oracle model.