DIGITAL SIGNATURES

Signing by hand

<table>
<thead>
<tr>
<th>...</th>
<th>COSMO</th>
<th>ALICE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cosmo</td>
<td>Alice</td>
</tr>
</tbody>
</table>

Bank

Alice

Pay Bob $100

= ?

no

yes

Pay Bob

Don't

Problem: signature is easily copied

Inference: signature must be a function of the message that only Alice can compute

Signing electronically

ALICE

Pay Bob $100

Internet

Bank

<table>
<thead>
<tr>
<th>SIGFILE</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>101 ... 1</td>
<td></td>
</tr>
</tbody>
</table>

Alice

Problem: signature is easily copied

Inference: signature must be a function of the message that only Alice can compute
What about a MAC?

Let Bank and Alice share a key $K$

A digital signature will have additional attributes:
- Even the bank cannot forge
- Verifier does not need to share a key with signer or, indeed, have any secrets

Digital signatures

A digital signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is a triple of algorithms where

Usage

Step 1: key generation
Alice lets $(pk, sk) \leftarrow K$ and stores $sk$ (securely).

Step 2: $pk$ dissemination
Alice enables any potential verifier to get $pk$.

Step 3: sign
Alice can generate a signature $\sigma$ of a document $M$ using $sk$.

Step 4: verify
Anyone holding $pk$ can verify that $\sigma$ is Alice’s signature on $M$.

Step 1 Example: RSA Key generation with openssl

Generating a private RSA key
1. Generate an RSA private key, of size 2048, and output it to a file named key.pem:

   ```
   $ openssl genrsa -out key.pem 2048
   Generating RSA private key, 2048 bit long modulus
   .................***
   .................................................................***
   e is 65537 (0x10001)
   ```

2. Extract the public key from the key pair, which can be used in a certificate:

   ```
   $ openssl rsa -in key.pem -outform PEM -pubout -out public.pem
   writing RSA key
   ```
Step 1 Example: EC Key generation with openssl

Generating a private EC key

1. Generate an EC private key, of size 256, and output it to a file named key.pem:

   ```
   $ openssl ecparam -name prime256v1 -genkey -noout -out key.pem
   ```

2. Extract the public key from the key pair, which can be used in a certificate:

   ```
   $ openssl ec -in key.pem -pubout -out public.pem
   read EC key
   writing EC key
   ```

After running these two commands you end up with two files: key.pem and public.pem. These files are referenced in various other guides on this page when dealing with key import.

Step 2: Dissemination of public keys

The public key does not have to be kept secret but a verifier needs to know it is authentic, meaning really Alice’s public key and not someone else’s.

Alice could put her public key \( pk \) on her webpage, her Facebook, a key server or include it as an email attachment.

Common method of dissemination is via certificates as discussed later.
Step 2 Example: Search on SGS key-server

Search results for 'diffie'

<table>
<thead>
<tr>
<th>Type</th>
<th>Site/key</th>
<th>cr. time</th>
<th>exp time</th>
<th>key expr</th>
</tr>
</thead>
<tbody>
<tr>
<td>pub</td>
<td>30778/2B71D11</td>
<td>2016-05-24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>Joe <a href="mailto:diffusion@broadcom.com">diffusion@broadcom.com</a></td>
<td>2016-05-24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pub</td>
<td>30778/2B71D11</td>
<td>2016-05-24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pub</td>
<td>30778/2B71D11</td>
<td>2016-05-24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pub</td>
<td>30778/2B71D11</td>
<td>2016-05-24</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

UF-CMA Security of a DS scheme

Let \( \mathcal{DS} = (K, S, \mathcal{V}) \) be a signature scheme and \( A \) an adversary.

**Game UF-CMA\(_{DS} \)**

\[
\begin{align*}
\text{procedure Initialize} & \quad (pk, sk) \leftarrow K; \ S \leftarrow \emptyset \\
& \quad \text{return } pk \\
\text{procedure Finalize} & \quad M, \sigma \\
& \quad d \leftarrow \mathcal{V}(pk, M, \sigma) \\
& \quad \text{return } (d = 1 \land M \notin S)
\end{align*}
\]

The uf-cma advantage of \( A \) is

\[
\text{Adv}_{\text{uf-cma}}^{\mathcal{DS}}(A) = \Pr \left[ \text{UF-CMA}^A_{\mathcal{DS}} \Rightarrow \text{true} \right]
\]

UF-CMA: Explanations

The “return \( pk \)” statement in **Initialize** means the adversary \( A \) gets the public key \( pk \) as input. It does not get \( sk \).

It can call **Sign** with any message \( M \) of its choice to get back a correct signature \( \sigma \leftarrow S(sk, M) \) of \( M \) under \( sk \). Notation indicates signing algorithm may be randomized.

To win, it must output a message \( M \) and a signature \( \sigma \) that are

- **Correct:** \( \mathcal{V}(pk, M, \sigma) = 1 \)
- **New:** \( M \notin S \), meaning \( M \) was not a query to **Sign**

**Interpretation:** **Sign** represents the signer and **Finalize** represents the verifier. Security means that the adversary can’t get the verifier to accept a message that is not authentic, meaning was not already signed by the sender.
RSA signatures

Fix an RSA generator \( K_{rsa} \) and let the key generation algorithm be

\[
\text{Alg } K
\]

\[(N, p, q, e, d) \leftarrow K_{rsa} \]

\[pk \leftarrow (N, e); sk \leftarrow (N, d)\]

Return \((pk, sk)\)

We will use these keys in all our RSA-based schemes and only describe signing and verifying.

Plain RSA signature scheme

Signer \( pk = (N, e) \) and \( sk = (N, d) \)

\[
\text{Alg } S_{N,d}(y)
\]

\[x \leftarrow y^d \mod N\]

Return \( x \)

\[
\text{Alg } V_{N,e}(y, x)
\]

If \((x^e \mod N = y)\) then return 1

Else return 0

Here \( y \in \mathbb{Z}_N^* \) is the message and \( x \in \mathbb{Z}_N^* \) is the signature.

Security of plain RSA signatures

To forge signature of a message \( y \), the adversary, given \( N, e \) but not \( d \), must compute \( y^d \mod N \), meaning invert the RSA function \( f \) at \( y \).

But RSA is 1-way so this task should be hard and the scheme should be secure.

Correct?

Security of plain RSA signatures

To forge signature of a message \( y \), the adversary, given \( N, e \) but not \( d \), must compute \( y^d \mod N \), meaning invert the RSA function \( f \) at \( y \).

But RSA is 1-way so this task should be hard and the scheme should be secure.

Correct?

Of course not...
Attacks on plain RSA

\textbf{adversary} \( A(N, e) \)
\begin{align*}
\text{Return} & \ (1, 1) \\
\text{Adv}^{\text{f-cma}}_{\text{DS}}(A) & = 1 \text{ because } 1^d \equiv 1 \pmod{N}
\end{align*}

\textbf{adversary} \( A(N, e) \)
\begin{align*}
\text{Pick some distinct } & y_1, y_2 \in \mathbb{Z}_N^* - \{1\} \\
x_1 & \leftarrow \text{Sign}(y_1); \ x_2 \leftarrow \text{Sign}(y_2) \\
\text{Return} & \ (y_1 y_2 \mod N, x_1 x_2 \mod N) \\
\text{Adv}^{\text{f-cma}}_{\text{DS}}(A) & = 1 \text{ because } (y_1 y_2)^d \equiv y_1^d y_2^d \pmod{N}
\end{align*}

When Diffie and Hellman introduced public-key cryptography they suggested the DS scheme
\begin{align*}
S(sk, M) & = D(sk, M) \\
V(pk, M, \sigma) & = 1 \text{ iff } E(pk, \sigma) = M
\end{align*}
where \((E, D)\) is a public-key encryption scheme. But
- This views public-key encryption as deterministic; they really mean trapdoor permutations in our language
- Plain RSA is an example
- It doesn’t work!
Nonetheless, many textbooks still view digital signatures this way.

Other issues

In plain RSA, the message is an element of \( \mathbb{Z}_N^* \). We really want to be able to sign strings of arbitrary length.

Throwing in a hash function

Let \( H: \{0,1\}^* \rightarrow \mathbb{Z}_N^* \) be a public hash function and let \( pk = (N, e) \) and \( sk = (N, d) \) be the signer’s keys. The hash-then-decrypt scheme is
\begin{align*}
\textbf{Alg} \quad & S_{N,d}(M) \\
& y \leftarrow H(M) \\
x & \leftarrow y^d \mod N \\
\text{Return} & \ x \\
\textbf{Alg} \quad & V_{N,e}(M, x) \\
& y \leftarrow H(M) \\
& \text{If } (x^e \mod N = y) \text{ then return } 1 \\
& \text{Else return } 0
\end{align*}
Succinctly,
\[ S_{N,d}(M) = H(M)^d \mod N \]
Different choices of \( H \) give rise to different schemes.
What we need from $H$

Suppose we have an adversary $C$ that can find a collision for $H$. Then we can break DS via

adversary $A(N, e)$

$(M_1, M_2) \xrightarrow{S} C$

$s_1 \leftarrow \text{Sign}(M_1)$

Return $(M_2, s_1)$

This works because $H(M_1) = H(M_2)$ implies $M_1, M_2$ have the same signatures:

$s_1 = S_{N, d}(M_1) = H(M_1)^d \mod N = H(M_2)^d \mod N = S_{N, d}(M_2)$

Conclusion: $H$ needs to be collision-resistant

---

RSA PKCS#1 signatures

Signer has $pk = (N, e)$ and $sk = (N, d)$ where $|N| = 1024$. Let $h: \{0, 1\}^* \rightarrow \{0, 1\}^{160}$ be a hash function (like SHA1) and let $n = 1024/8 = 128$.

Then

$$H_{PKCS}(M) = 00||01||FF||\ldots||FF||h(M)$$

And

$$S_{N,d}(M) = H_{PKCS}(M)^d \mod N$$

---

Does 1-wayness prevent forgery?

Forger $A$’s goal

Inverter $A$’s goal

Recall

$H_{PKCS}(M) = 00||01||FF||\ldots||FF||h(M)$

But first $n - 20 = 108$ bytes out of $n$ are fixed so $H_{PKCS}(M)$ does not look “random” even if $h$ is a RO or perfect.

We cannot hope to show RSA PKCS#1 signatures are secure assuming (only) that RSA is 1-way no matter what we assume about $h$ and even if $h$ is a random oracle.

Problem: 1-wayness of RSA does not imply hardness of computing $y^d \mod N$ if $y$ is not random
Goal

We will validate the hash-then-decrypt paradigm

\[ S_{N,d}(M) = H(M)^d \mod N \]

by showing the signature scheme is provably UF-CMA assuming RSA is 1-way as long as \( H \) is a RO.

This says the paradigm has no "structural weaknesses" and we should be able to get security with "good" choices of \( H \).

A "good" choice of \( H \) might be something like

\[ H(M) = \text{first } n \text{ bytes of } \text{SHA1}(1 \parallel M) \parallel \text{SHA1}(2 \parallel M) \parallel \cdots \parallel \text{SHA1}(11 \parallel M) \]

Full-Domain-Hash (FDH) [BR96]

Signer public key is \( pk = (N, e) \) and secret key is \( sk = (N, d) \)

\[
\begin{align*}
\text{Alg} & \quad S_{N,d}(M) \\
\text{Alg} & \quad V_{N,e}^H(M, x)
\end{align*}
\]

Return \( H(M)^d \mod N \)

If \( (x^e \mod N = H(M)) \) then return 1
Else return 0

This is a random oracle model scheme where both algorithms have access to the random oracle \( H : \{0,1\}^* \rightarrow \mathbb{Z}_N^* \). An instantiation may be to set \( H(M) \) to the first \( |N| \) bits of

\[ \text{SHA1512}(0^8||M)||\text{SHA1512}(0^71||M)|| \cdots \]

Security of FDH in RO model

Theorem: [BR96] Let \( K_{\text{rsa}} \) be a RSA generator and \( DS = (K, S, V) \) the associated FDH RO-model signature scheme. Let \( A \) be a uf-cma adversary making \( q_s \) signing queries and \( q_H \) queries to the RO \( H \) and having running time at most \( t \). Then there is an inverter \( I \) such that

\[ \text{Adv}_{DS}^{\text{uf-cma}}(A) \leq (q_s + q_H + 1) \cdot \text{Adv}_{K_{\text{rsa}}}^{\text{ow}}(I) \]

Furthermore the running time of \( I \) is that of \( A \) plus the time for \( O(q_s + q_H + 1) \) computations of the RSA function.
Assume $A$
- Makes no $\text{Sign}$ queries
- Makes exactly one $H$-query $M$
- Then outputs a forgery $(M, \sigma)$

Let us see how to build $I$ so that

$$\text{Adv}_{\text{uf-cma}}^{\text{DS}}(A) = \text{Adv}_{K_{\text{rsa}}}^{\text{OW}}(I).$$

**The case $q_s = 0$ and $q_H = 1$**

Assume $A$
- Makes no $\text{Sign}$ queries
- Makes exactly one $H$-query $M$
- Then outputs a forgery $(M, \sigma)$

Let us see how to build $I$ so that

$$\text{Adv}_{\text{uf-cma}}^{\text{DS}}(A) = \text{Adv}_{K_{\text{rsa}}}^{\text{OW}}(I).$$

**The inverter for the case $q_s = 0$ and $q_H = 1$**

**adversary** $I(N, e, y)$

$$(M, \sigma) \leftarrow A^{\text{HSim}}(N, e)$$

Return $\sigma$

<table>
<thead>
<tr>
<th>subroutine</th>
<th>HSim($M$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return $y$</td>
<td></td>
</tr>
</tbody>
</table>

Then

$$\text{Adv}_{\text{uf-cma}}^{\text{DS}}(A) = \text{Adv}_{K_{\text{rsa}}}^{\text{OW}}(I)$$

because

$$\sigma \equiv \text{HSim}(M)^d \equiv y^d \pmod{N}$$

**The case $q_s = 0$ and $q_H > 1$**

Assume $A$
- Makes no $\text{Sign}$ queries
- Makes $H$-queries $M_1, \ldots, M_{q_H}$
- Then outputs a forgery $(M, \sigma)$ such that $M \in \{M_1, \ldots, M_{q_H}\}$

Let us see how to build $I$ so that

$$\text{Adv}_{\text{uf-cma}}^{\text{DS}}(A) = \text{Adv}_{K_{\text{rsa}}}^{\text{OW}}(I).$$

**Inverter for the case $q_s = 0$ and $q_H > 1$**

How about, as before, return $y$ in response to a $H$-query:

**adversary** $I(N, e, y)$

$$(M, \sigma) \leftarrow A^{\text{HSim}}(N, e)$$

Return $\sigma$

<table>
<thead>
<tr>
<th>subroutine</th>
<th>HSim($M$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return $y$</td>
<td></td>
</tr>
</tbody>
</table>

Say $A$’s queries are $M_1, \ldots, M_q$ and $M = M_i$. Then if $\sigma^e \equiv H(M_i) \pmod{N}$ we have $\sigma^e \equiv y \pmod{N}$ so $I$ wins so

$$\text{Adv}_{\text{uf-cma}}^{\text{DS}}(A) = \Pr[\sigma^e \equiv y^d \pmod{N}] = \text{Adv}_{K_{\text{rsa}}}^{\text{OW}}(I).$$

Rght?
Inverter for the case \( q_s = 0 \) and \( q_H > 1 \)

How about, as before, return \( y \) in response to a \( H \)-query:

\[
\text{adversary } I(N, e, y) \\
(M, \sigma) \xleftarrow{\$} A^{H\text{Sim}}(N, e) \\
\text{return } \sigma
\]

Say \( A \)'s queries are \( M_1, \ldots, M_q \) and \( M = M_i \). Then if \( \sigma^e \equiv y \pmod{N} \) we have \( \sigma^e \equiv y \pmod{N} \) so \( I \) wins so

\[
\text{Adv}_{\text{DS}}^{\text{uf-cma}}(A) = \Pr[\sigma^e \equiv y; \pmod{N}] = \text{Adv}^{\text{OW}}_{K_{\text{rsa}}}(I).
\]

Right? \textbf{NO}.

This is wrong because the answers to \( A \)'s queries are not independent, meaning HSim does not look like a "real" RO.

For example, what if \( A \) made queries \( M_1 = M_2 \) and on getting back \( y_1, y_2 \) aborted if \( y_1 = y_2 \)? \( A \)'s advantage in the simulation could be 0.

The case \( q_s > 0 \)

How can the Inverter \( I \) (not knowing \( d \)) return the signature \( H(M)^d \pmod{N} \) in response to \text{Sign} query \( M \)?

Trick: When \( M_i \) is queried to \( H \), Inverter will
- pick \( x_i \xleftarrow{\$} \mathbb{Z}_N^* \) and let \( y_i \leftarrow x_i^e \pmod{N} \)
- Return \( H(M_i) = y_i \)

Then if there is a \text{Sign}(M') query it can return \( x_i \) as the signature.
Simplification

Assume that if $A$ makes $Sign$ query $M$, it has previously made $H$-query $M$. Outputs $(M, \sigma)$ then it has previously made $H$-query $M$ and not made $Sign$ query $M$.

Can easily modify $A$ to have these properties at the cost of increasing the number of $H$-queries to $q = q_s + q_H + 1$.

Also assume $A$ never repeats a $H$-query.

Analysis intuition

Let $i$ be such that $A$ outputs $(M, \sigma)$ with $M = M_i$. Then if $i = g$

- $\sigma^g \equiv H(M_i) \equiv y \pmod{N}$ so inverter finds $\sigma = y^d \pmod{N}$
- All $A$’s queries are correctly answered

Since $i = g$ with probability $1/q$ we have

$$\text{Adv}_{K_{RSA}}^{ow}(I) \geq \frac{1}{q} \cdot \text{Adv}_{DS}^{\text{cma}}(A).$$

Fundamental Lemma variant

A formal proof can be given based on the following:

**Lemma** [BR06] Let $G_i, G_j$ be identical-until-bad games and $A$ an adversary. Then for any $y$

$$\Pr \left[ G_i^A \Rightarrow y \land G_i^A \text{ doesn’t set bad} \right] = \Pr \left[ G_j^A \Rightarrow y \land G_j^A \text{ doesn’t set bad} \right]$$
PSS [BR96]

Signer \( pk = (N, e) \) and \( sk = (N, d) \)

Algorithm \( S_{N,d}^{h,g_1,g_2}(M) \)
- \( r \leftarrow \{0,1\}^{160} \)
- \( w \leftarrow h(M \parallel r) \)
- \( r^* \leftarrow g_1(w) \oplus r \)
- \( y \leftarrow 0 \parallel w \parallel r^* \parallel g_2(w) \)
- \( \text{return } y^d \mod N \)

Algorithm \( V_{N,e}^{h,g_1,g_2}(M, x) \)
- \( y \leftarrow x^e \mod N \)
- \( b \leftarrow w \parallel r^* \parallel P \leftarrow y \)
- \( r^* \leftarrow r^* \oplus g_1(w) \)
- \( \text{if } (g_2(w) \neq P) \text{ then return } 0 \)
- \( \text{if } (b = 1) \text{ then return } 0 \)
- \( \text{if } (h(M \parallel r) \neq w) \text{ then return } 0 \)
- \( \text{return } 1 \)

Here \( h, g_1: \{0,1\}^* \rightarrow \{0,1\}^{160} \) and \( g_2: \{0,1\}^* \rightarrow \{0,1\}^{k-321} \) are random oracles where \( k = |N| \).

Exercise

Let \( K_{\text{rsa}} \) be a RSA key generator with security parameter \( k \geq 2048 \). Let the algorithms of signature scheme \( DS = (K, S, V) \) be defined as follows, with notation explained on the next slide:

\[
\text{Alg } K \\
(N, p, q, e, d) \leftarrow K_{\text{rsa}} ; \ pk \leftarrow (N, e) ; sk \leftarrow (N, d) ; \text{ Return } (pk, sk)
\]

\[
\text{Alg } S_{N,d}(M) \\
\text{If } |M| \neq 4096 \text{ then return } \bot \\
x_1 \leftarrow \langle 1 \rangle \parallel M[1] ; x_2 \leftarrow \langle 2 \rangle \parallel M[2] \\
y \leftarrow H(x_1) \cdot H(x_2) \mod N \\
s \leftarrow y^d \mod N \\
\text{Return } s
\]

\[
\text{Alg } V_{N,e}(M, s) \\
\text{If } |M| \neq 4096 \text{ then return } 0 \\
x_1 \leftarrow \langle 1 \rangle \parallel M[1] ; x_2 \leftarrow \langle 2 \rangle \parallel M[2] \\
y \leftarrow H(x_1) \cdot H(x_2) \mod N \\
\text{If } s^e \equiv y \mod N \\
\text{then return } 1 \text{ else return } 0
\]

Exercise

Above, \( H: \{0,1\}^* \rightarrow Z_N^* \) is a public, collision resistant hash function. A valid message \( M \) is a 4096 bit string and is viewed as a pair of 2048 bit blocks, \( M = M[1]M[2] \). By \( \parallel \) we denote concatenation, and by \( \langle i \rangle \) we denote the encoding of integer \( i \) as a binary string of exactly two bits.

Present in pseudocode a \( O(k^3) \)-time adversary \( A \) making at most three queries to its \textbf{Sign} oracle and achieving \( \text{Adv}^{\text{cma}}_{DS}(A) = 1 \).
ElGamal Signatures

Let \( G = \mathbb{Z}_p^* = \langle g \rangle \) where \( p \) is prime.
Signer keys: \( pk = X = g^x \in \mathbb{Z}_p^* \) and \( sk = x \leftarrow \mathbb{Z}_{p-1} \)

\[
\text{Alg } S_x(m)
\]
\[
k \leftarrow \mathbb{Z}_{p-1}^*
\]
\[
r \leftarrow g^k \mod p
\]
\[
s \leftarrow (m - xr) \cdot k^{-1} \mod (p - 1)
\]
Return \( (r, s) \)

Correctness check: If \( (r, s) \leftarrow S_x(m) \), then, in the group \( G \) we have:

\[
X' \cdot r^s = g^{xr} \cdot g^{ks} = g^{xr+ks} \equiv g^{x(m-1)k} \mod (p-1) = g^{x(m-1)-xr} = g^m
\]

So \( V_X(m, (r, s)) = 1 \).

Security of ElGamal Signatures

Signer keys: \( pk = X = g^x \in \mathbb{Z}_p^* \) and \( sk = x \leftarrow \mathbb{Z}_{p-1} \)

\[
\text{Alg } V_X(m, (r, s))
\]
\[
k \leftarrow \mathbb{Z}_{p-1}^*
\]
\[
r \leftarrow g^k \mod p
\]
\[
s \leftarrow (m - xr) \cdot k^{-1} \mod (p - 1)
\]
Return \( (r, s) \)

Suppose given \( X = g^x \) and \( m \) the adversary wants to compute \( r, s \) so that \( X' \cdot r^s \equiv g^m \mod p \). It could:
- Pick \( r \) and try to solve for \( s = \text{DLog}_{\mathbb{Z}_p^*}(g^m X^{-r}) \)
- Pick \( s \) and try to solve for \( r \ ...? \)

Forgery of ElGamal Signatures

Adversary has better luck if it picks \( m \) itself:

\[
\text{Adversary } A(X)
\]
\[
r \leftarrow g^X \mod p; s \leftarrow (-r) \mod (p - 1); m \leftarrow s
\]
Return \( (m, (r, s)) \)

Then:

\[
X' \cdot r^s \mod p = X' (gX)^s \mod p = X' (r+s)g^s \mod p = X' (r+s) \mod (p-1) g^m \mod p = g^m \mod p
\]

So \( (r, s) \) is a valid forgery on \( m \).

ElGamal with hashing

Let \( G = \mathbb{Z}_p^* = \langle g \rangle \) where \( p \) is prime.
Signer keys: \( pk = X = g^x \in \mathbb{Z}_p^* \) and \( sk = x \leftarrow \mathbb{Z}_{p-1} \)

\[
 H : \{0, 1\}^* \rightarrow \mathbb{Z}_{p-1} \) a hash function.

\[
\text{Alg } S_x(M)
\]
\[
m \leftarrow H(M)
\]
\[
k \leftarrow \mathbb{Z}_{p-1}^*
\]
\[
r \leftarrow g^k \mod p
\]
\[
s \leftarrow (m - xr) \cdot k^{-1} \mod (p - 1)
\]
Return \( (r, s) \)

\[
\text{Alg } V_X(m, (r, s))
\]
\[
m \leftarrow H(M)
\]
\[
\text{If } (r \not\in G \text{ or } s \not\in \mathbb{Z}_{p-1}) \text{ then return 0}
\]
\[
\text{If } (X' \cdot r^s \equiv g^m \mod p) \text{ then return 1}
\]
else return 0
**ElGamal with hashing**

Let $G = \mathbb{Z}_p^* = \langle g \rangle$ where $p$ is a prime.

Signer keys: $pk = X = g^x \in \mathbb{Z}_p^*$ and $sk = x \leftarrow \mathbb{Z}_{p-1}$

$H : \{0,1 \}^* \rightarrow \mathbb{Z}_{p-1}$ a hash function.

**Alg** $S_x(M)$

$m \leftarrow H(M)$

$k \leftarrow \mathbb{Z}_{p-1}^*$

$r \leftarrow g^k \mod p$

$s \leftarrow (m - xr) \cdot k^{-1} \mod (p - 1)$

Return $(r,s)$

**Alg** $V_X(M, (r,s))$

$m \leftarrow H(M)$

If $(r \notin G$ or $s \notin \mathbb{Z}_{p-1})$ then return 0

If $(X^r \cdot r^s \equiv g^m \mod p)$ then return 1

else return 0

Requirements on $H$:

- Collision-resistant
- One-way to prevent previous attack

**Discussion**

**DSA**

Let $p$ be a 1024-bit prime. For DSA, let $q$ be a 160-bit prime dividing $p - 1$.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>signing cost</th>
<th>verification cost</th>
<th>signature size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElGamal</td>
<td>1 1024-bit exp</td>
<td>1 1024-bit exp</td>
<td>2048 bits</td>
</tr>
<tr>
<td>DSA</td>
<td>1 160-bit exp</td>
<td>1 160-bit exp</td>
<td>320 bits</td>
</tr>
</tbody>
</table>

By a “$e$-bit exp” we mean an operation $a, n \mapsto a^n \mod p$ where $a \in \mathbb{Z}_p^*$ and $n$ is an $e$-bit integer. A 1024-bit exponentiation is more costly than a 160-bit exponentiation by a factor of $1024/160 \approx 6.4$.

DSA is in FIPS 186.

**DSA as shown works only over the group of integers modulo a prime, but there is also a version ECDSA of it for elliptic curve groups.**

In ElGamal and DSA/ECDSA, the expensive part of signing, namely the exponentiation, can be done off-line.

No proof that ElGamal or DSA is UF-CMA under a standard assumption (DL, CDH, ...) is known, even if $H$ is a RO. Proofs known for variants.

The Schnorr scheme works in an arbitrary (prime-order) group. When implemented in a 160-bit elliptic curve group, it is as efficient as ECDSA. It can be proven UF-CMA in the random oracle model under the discrete log assumption [PS,AABN]. The security reduction, however, is quite loose.
Exercise

Let \( p \) be a prime of bit length \( k \geq 1024 \) such that \( (p - 1)/2 \) is also prime, and let \( g \) be a generator of the group \( G = \mathbb{Z}_p^* \). (Here \( p, g \) are public quantities.) Let \( q = p - 1 \) be the order of \( G \). Consider the digital signature scheme \( DS = (K, S, V) \) whose component algorithms are depicted below, where the message \( m \) is in \( \mathbb{Z}_q^* \):

**Alg** \( K \)

\[
\begin{align*}
x &\leftarrow \mathbb{Z}_q^*; X \leftarrow g^x; y &\leftarrow \mathbb{Z}_q^*; Y \leftarrow g^y
\end{align*}
\]

return \( ((X, Y), (x, y)) \)

**Alg** \( S((x, y), m) \)

If \( m \notin \mathbb{Z}_q^* \) then return \( \perp \)

\( z \leftarrow (y + xm) \mod q \)

return \( z \)

**Alg** \( V((X, Y), m, z) \)

if \( m \notin \mathbb{Z}_q^* \) then return 0

if \( z \notin \mathbb{Z}_q^* \) then return 0

if \( g^z \equiv YX^m \pmod{p} \) then return 1

else return 0

1. Prove that \( V((X, Y), m, z) = 1 \) for any key-pair \( ((X, Y), (x, y)) \) that might be output by \( K \), any message \( m \in \mathbb{Z}_q^* \), and any \( z \) that might be output by \( S((x, y), m) \).

2. Present in pseudocode a \( O(k^2) \)-time adversary \( A \) making at most two queries to its \( \text{Sign} \) oracle and achieving \( \text{Adv}^{\text{f-cma}}_{DS}(A) = 1 \).

Randomization in signatures

We have seen many randomized signature schemes: PSS, ElGamal, DSA/ECDSA, Schnorr, ...

Re-using coins across different signatures is not secure, but there are (other) ways to make these schemes deterministic without loss of security.