Announcements

• HW 8 out Wednesday evening
  • Due **Wednesday Apr 10** at 11:59PM on Gradescope
  • Covers
    • RSA
    • little bit of IND-CCA PKE
Recap of last lecture
Symmetric-key Message Authentication

We want Alice to generate a tag for the message $m$ which is hard to generate without the secret key $k$. Can also alter/inject more messages!
We want Alice to generate a signature for the message $m$ which is hard to forge without the secret/signing key $sk$. 

Can also alter/inject more messages!
Does PKE not solve this?

Anybody can encrypt, and no way for recipient to check.

Alice

sk

Bob

pk

Can toggle between m and m'

How can Bob check?
New primitive: Digital Signatures
Digital Signatures: Definition

A triple of PPT algorithms \((\text{Gen}, \text{Sign}, \text{Verify})\) such that

- Key generation: \(\text{Gen}(1^n) \rightarrow (\text{sk}, \text{pk})\)
- Message signing: \(\text{Sign}(\text{sk}, m) \rightarrow \sigma\)
- Signature verification: \(\text{Verify}(\text{pk}, m, \sigma) \rightarrow b \in \{0,1\}\)

**Correctness:** For all \(vk, sk, m: \text{Verify}(pk, m, \text{Sign}(sk, m)) = 1\)
EUF-CMA for Signatures

\[ \Pr \left[ m^* \notin \{ m_i \} \quad \text{and} \quad \text{Verify}(\text{pk}, m^*, \sigma^*) = 1 \right] = \text{negl}(\lambda) \]
**Strong EUF-CMA for Signatures**

\[
\begin{align*}
\text{Challenger} & \quad \text{pk} \quad \text{Adversary} \\
& \quad m_i \quad \sigma_i \\
& \quad (m^*, \sigma^*) \\
\Pr \left[ (m^*, \sigma^*) \notin \{(m_i, \sigma_i)\} \text{ and } \right. \\
& \left. \text{Verify}(pk, m^*, \sigma^*) = 1 \right] = \text{negl}(\lambda)
\end{align*}
\]
<table>
<thead>
<tr>
<th>Signatures</th>
<th>MACs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$ users require $n$ key-pairs</td>
<td>$n$ users require $n^2$ keys</td>
</tr>
<tr>
<td>Publicly Verifiable</td>
<td>Privately Verifiable</td>
</tr>
<tr>
<td>Transferable</td>
<td>Not Transferable</td>
</tr>
<tr>
<td>Provides Non-Repudiation (is this a good thing or a bad thing?)</td>
<td>Does not provide Non-Rep.</td>
</tr>
</tbody>
</table>
Let \( (\text{Gen, Sign, V}) \) be a signature scheme.

Suppose an attacker is able to find \( m_0 \neq m_1 \) such that

\[
V(\text{pk}, m_0, \sigma) = V(\text{pk}, m_1, \sigma)
\]

for all \( \sigma \) and keys \( (\text{pk}, \text{sk}) \leftarrow \text{Gen} \).

Can this signature be secure?

- Yes, the attacker cannot forge a signature for either \( m_0 \) or \( m_1 \)
- No, signatures can be forged using a chosen msg attack
- It depends on the details of the scheme
Alice generates a (pk,sk) and gives pk to her bank.

Later Bob shows the bank a message \( m=\text{“pay Bob 100$”} \) properly signed by Alice, i.e. \( \text{Verify}(pk,m,sig) = 1 \)

Alice says she never signed \( m \). Is Alice lying?

Alice is lying: existential unforgeability means Alice signed \( m \) and therefore the Bank should give Bob 100$ from Alice’s account.

Bob could have stolen Alice’s signing key and therefore the bank should not honor the statement.

What a mess: the bank will need to refer the issue to the courts.
Applications
Applications

Code signing:
• Software vendor signs code
• Clients have vendor’s pk. Install software if signature verifies.

software vendor

initial software install (pk)

[ software update #1 , sig ]

[ software update #2 , sig ]

many clients
More generally:

One-time authenticated channel (non-private, one-directional) \[\Rightarrow\] many-time authenticated channel

Initial software install is authenticated, but not private

\((pk, sk) \leftarrow \text{Gen}\)

\(m_1\) \(\rightarrow\) \(\text{sig}_1 \leftarrow S(sk, m_1)\)

\(m_2\) \(\rightarrow\) \(\text{sig}_2 \leftarrow S(sk, m_2)\)

\[\vdots\]

eavesdrop, but not modify
Important application: Certificates

Problem: browser needs server’s public-key to setup a session key

Solution: server asks trusted 3rd party (CA) to sign its public-key pk

Server uses Cert for an extended period (e.g. one year)
Certificates: example

Important fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Number</td>
<td>58147444488373890497</td>
</tr>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Signature Algorithm</td>
<td>SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)</td>
</tr>
<tr>
<td>Parameters</td>
<td>none</td>
</tr>
<tr>
<td>Not Valid Before</td>
<td>Wednesday, July 31, 2013 4:59:24 AM Pacific Daylight Time</td>
</tr>
<tr>
<td>Not Valid After</td>
<td>Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time</td>
</tr>
<tr>
<td>Public Key Info</td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>Elliptic Curve Public Key (1.2.840.10045.2.1)</td>
</tr>
<tr>
<td>Parameters</td>
<td>Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)</td>
</tr>
<tr>
<td>Public Key</td>
<td>65 bytes: 04 71 6C DD E0 0A C9 76 ...</td>
</tr>
<tr>
<td>Key Size</td>
<td>256 bits</td>
</tr>
<tr>
<td>Key Usage</td>
<td>Encrypt, Verify, Derive</td>
</tr>
<tr>
<td>Signature</td>
<td>256 bytes: 8A 38 FE D6 F5 E7 F6 59 ...</td>
</tr>
</tbody>
</table>
What entity generates the CA’s secret key $sk_{CA}$?
Signing email: DKIM (domain key identified mail)

Problem: bad email claiming to be from someuser@gmail.com but in reality, mail is coming from domain badguy.com

⇒ Incorrectly makes gmail.com look like a bad source of email

Solution: gmail.com (and other sites) sign every outgoing mail

Gmail user

Gmail.com

Signing key

From: bob@gmail.com

body

sig

Gmail.com

Recipients

DNS

query

verify sig

Gmail pk
When to use signatures

Generally speaking:

• If one party signs and **one** party verifies:  **use a MAC**
  – Often requires interaction to generate a shared key
  – Recipient can modify the data and re-sign it before passing the data to a 3rd party

• If one party signs and **many** parties verify:  **use a signature**
  – Recipients **cannot** modify received data before passing data to a 3rd party (non-repudiation)
Constructions
Simpler Goal: EUF-CMA for 1-time Signatures

Pr\[\begin{align*} &\text{Verify}(pk, m^*, \sigma^*) = 1 \\ &m^* \neq m_1 \text{ and} \end{align*}\] = negl(\(\lambda\))
Lamport (One-time) Signatures from OWFs

Signing Key sk: \( \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \)

Public Key pk: \( \begin{pmatrix} y_0 = f(x_0) \\ y_1 = f(x_1) \end{pmatrix} \)

Signing a bit \( b \): The signature is \( \sigma = x_b \)

Verifying \((b, \sigma)\): Check if \( f(\sigma) = y_b \)

**Claim**: Assuming \( f \) is a OWF, no PPT adversary can produce a signature of \( \overline{b} \) given a signature of \( b \).
Lamport (One-time) Signatures for $n$ bits

Secret Key $sk$: \[
\begin{pmatrix}
    x_{1,0} & x_{2,0} & \ldots & x_{n,0} \\
    x_{1,1} & x_{1,1} & \ldots & x_{n,1}
\end{pmatrix}
\]

Public Key $pk$: \[
\begin{pmatrix}
    y_{1,0} & y_{2,0} & \ldots & y_{n,0} \\
    y_{1,1} & y_{2,1} & \ldots & y_{n,1}
\end{pmatrix}
\]  where $y_{i,b} = f(x_{i,b})$.

Signing $m = (m_1, \ldots, m_n)$: \[
\sigma = (x_{1,m_1}, x_{2,m_2}, \ldots, x_{n,m_n})
\]

**Claim**: Assuming $f$ is a OWF, no PPT adv can produce a signature of $m$ given a signature of a single $m' \neq m$.

**Claim**: Can forge signature on any message given the signatures on (some) two messages.
Lamport (One-time) Signatures for arbitrary bits

**Secret Key sk:**
\[
\begin{pmatrix}
    x_{1,0} & x_{2,0} & \cdots & x_{n,0} \\
    x_{1,1} & x_{1,1} & \cdots & x_{n,1}
\end{pmatrix}
\]

**Public Key pk:**
\[
\begin{pmatrix}
    y_{1,0} & y_{2,0} & \cdots & y_{n,0} \\
    y_{1,1} & y_{2,1} & \cdots & y_{n,1}
\end{pmatrix}
\text{ where } y_{i,b} = f(x_{i,b}).
\]

**Signing** $m$:
1. $z := H(m)$
2. $\sigma = (z_{1,m_1}, z_{2,m_2}, \ldots, z_{n,m_n})$

**Claim**: Assuming $H$ is CRH and $f$ is a OWF, no PPT adv can produce a signature of $m$ given a signature of a single $m' \neq m$.

**Claim**: Can forge signature on any message given the signatures on (some) two messages.
Constructing a Signature Scheme

Step 0. Still one-time, but arbitrarily long messages.

Step 1. Many-time: Stateful, Growing Signatures.

Step 2. How to Shrink the signatures.

Step 3. How to Shrink Alice’s storage.

Step 4. How to make Alice stateless.

Step 5 (optional). How to make Alice stateless and deterministic.
So far, only one-time security…
Constructing a Signature Scheme

**Theorem** [Naor-Yung’89, Rompel’90]
(EUF-CMA-secure) Signature schemes exist assuming that one-way functions exist.

**TODAY:**
(EUF-CMA-secure) Signature schemes exist assuming that collision-resistant hash functions exist.
(Many-time) Signature Scheme

In four+ steps

Step 1. Stateful, Growing Signatures. Idea: Signature *Chains*

Step 2. How to Shrink the signatures. Idea: Signature *Trees*

Step 3. How to Shrink Alice’s storage.
   Idea: *Pseudorandom Trees*

Step 4. How to make Alice stateless.
   Idea: *Randomization*

Step 5 *(optional)*. How to make Alice stateless and deterministic. Idea: *PRFs.*
Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice starts with a secret signing Key $sk_0$

When signing a message $m_1$:
- Generate a new pair $(sk_1, pk_1)$
- Produce signature $\sigma_1 \leftarrow \text{Sign}(sk_0, m_1 || pk_1)$
- Output $pk_1 || \sigma_1$.
- Remember $pk_1 || m_1 || \sigma_1$ as well as $sk_1$.

To verify a signature $pk_1 || \sigma_1$ for message $m_1$:
- Run $\text{Verify}(pk_0, pk_1 || m_1, \sigma_1) = 1$
Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice starts with a secret signing Key $sk_0$

When signing a message $m_1$:

Generate a new pair $(sk_1, pk_1)$

Produce signature $\sigma_1 \leftarrow \text{Sign}(sk_0, m_1 || pk_1)$

Output $pk_1 || \sigma_1$.

Remember $pk_1 || m_1 || \sigma_1$ as well as $sk_1$.

$$\sigma_1 \quad m_1$$

$$pk_0 \quad \rightarrow \quad pk_1$$
Step 1: Stateful Many-time Signatures

**Idea: Signature Chains.**

Alice starts with a secret signing Key $sk_0$

When signing the next message $m_2$
- Generate a new pair $(sk_2, pk_2)$
- Produce signature $\sigma_2 \leftarrow \text{Sign}(sk_1, m_2 \| pk_2)$
- Output ???

\[
\begin{array}{c|c}
\text{Alice} & \text{pk}_0 \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\sigma_1 \\
pk_0 \longrightarrow pk_1 \\
m_1
\end{array}
\]
Step 1: Stateful Many-time Signatures

**Idea: Signature Chains.**

Alice starts with a secret signing Key $sk_0$

When signing the next message $m_2$

Generate a new pair $(sk_2, pk_2)$

Produce signature $\sigma_2 \leftarrow \text{Sign}(sk_1, m_2 || pk_2)$

Output $pk_2 || \sigma_2$??

$\sigma_1 \quad m_1$

$pk_0 \quad \rightarrow \quad pk_1$
Step 1: Stateful Many-time Signatures

**Idea: Signature Chains.**

Alice starts with a secret signing Key $sk_0$

When signing the next message $m_2$

Generate a new pair $(sk_2, pk_2)$

Produce signature $\sigma_2 \leftarrow \text{Sign}(sk_1, m_2 \| pk_2)$

Output $pk_1 \| pk_2 \| \sigma_2$??

$\sigma_1 \quad m_1$

$pk_0 \quad \quad \quad \quad \rightarrow \quad pk_1$
Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice starts with a secret signing Key $sk_0$

When signing the next message $m_2$

Generate a new pair $(sk_2, pk_2)$

Produce signature $\sigma_2 \leftarrow \text{Sign}(sk_1, m_2 \| pk_2)$

Output $(pk_1 \| m_1 \| \sigma_1) \| pk_2 \| \sigma_2$

(additionally) remember $pk_2 \| m_2 \| \sigma_2$ as well as $sk_2$. 

\[
\begin{array}{c|c}
\text{Alice} & \text{pk}_0 \\
\hline
\end{array}
\]
Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Two major problems:

1. Alice is stateful: Alice needs to remember a whole lot of things, $O(T)$ information after $T$ steps.

2. The signatures grow: Length of the signature of the $T$-th message is $O(T)$.

$$
\tau_1 \quad m_1 \quad \tau_2 \quad m_2 \quad \tau_3 \quad m_3 \quad \tau_4 \quad m_4
$$

$$
VK_0 \quad VK_1 \quad VK_2 \quad VK_3 \quad VK_4 \quad \ldots
$$
(Many-time) Signature Scheme

In four+ steps

Step 1. Stateful, Growing Signatures. Idea: Signature *Chains*

Step 2. How to Shrink the signatures. Idea: Signature *Trees*
Step 2. How to Shrink the signatures.

\[ VK_\epsilon \]
Alice (the *stateful* signer) computes many \((VK, SK)\) pairs and arranges them in a tree of depth = sec. param. \(\lambda\).
Step 2. How to Shrink the signatures.

Signature of the first message $m_0$:

Use $VK_{000}$ to sign $m_0$.

“Authenticate” $VK_{000}$ using the “signature path”. 

**Step 2.** How to Shrink the signatures.

$VK_\epsilon$

$VK_0 \leftarrow \sigma_\epsilon$

$VK_1 \leftarrow \sigma_\epsilon$

$VK_00 \leftarrow \sigma_0$

$VK_01 \leftarrow \sigma_0$

$VK_{10} \leftarrow \sigma_0$

$VK_{11} \leftarrow \sigma_0$

$VK_{00} \leftarrow \sigma_{00}$

$VK_{01} \leftarrow \sigma_{00}$

$VK_{10} \leftarrow \sigma_{00}$

$VK_{11} \leftarrow \sigma_{00}$

Signature of the first message $m_0$

$(\sigma_\epsilon \leftarrow \text{Sign}(SK_\epsilon, VK_0 || VK_1))$,  
$(\sigma_{00} \leftarrow \text{Sign}(SK_{000}, VK_{000} || VK_{001}))$,  
$(\tau_0 \leftarrow \text{Sign}(SK_{000}, m_0))$
Step 2. How to Shrink the signatures.

\[ VK_\epsilon \]

\[ VK_0 \leftarrow \sigma_\epsilon \rightarrow VK_1 \]

\[ VK_{00} \leftarrow \sigma_0 \rightarrow VK_{01} \]

\[ VK_{000} \leftarrow \tau_0 \rightarrow VK_{001} \]

Authentication Path for \( VK_{000} \):

\[
(\sigma_\epsilon \leftarrow \text{Sign}(SK_\epsilon, VK_0 || VK_1), \sigma_0 \leftarrow \text{Sign}(SK_0, VK_{00} || VK_{01}), \sigma_{00} \leftarrow \text{Sign}(SK_{00}, VK_{000} || VK_{001}))
\]
Step 2. How to Shrink the signatures.

Signature of the first message $m_0$:
(Authentication path for $VK_{000}$,
$\tau_0 \leftarrow \text{Sign}(SK_{000}, m_0)$)
Step 2. How to Shrink the signatures.

Signature of the second message $m_1$:
(Authentication path for $VK_{001}$,
$\tau_0 \leftarrow \text{Sign}(SK_{001}, m_1)$)
Step 2. How to Shrink the signatures.

\[ VK_\epsilon \]

\[ VK_0 \]

\[ VK_{00} \]

\[ VK_{000} \]

\[ VK_{001} \]

\[ VK_{010} \]

\[ VK_{011} \]

\[ VK_{100} \]

\[ VK_{101} \]

\[ VK_{110} \]

\[ VK_{111} \]

Signature of the third message \( m_2 \):

(Authentication path for \( VK_{010} \),

\[ \tau_2 \leftarrow \text{Sign}(SK_{010}, m_2) \])
Step 2. How to Shrink the signatures.

GOOD NEWS: Each verification key (incl. at the leaves) is used only once, so one-time security suffices!
Step 2. How to Shrink the signatures.

GOOD NEWS: Signatures consist of $\lambda$ one-time signatures and do not grow with time!
Step 2. How to Shrink the signatures.

BAD NEWS:  
Signer generates and keeps the entire ( \( \approx 2^\lambda \)-size) signature tree in memory!
(Many-time) Signature Scheme

In four+ steps

Step 1. Stateful, Growing Signatures. Idea: Signature *Chains*

Step 2. How to Shrink the signatures. Idea: Signature *Trees*

Step 3. How to Shrink Alice’s storage.
   Idea: *Pseudorandom Trees*
Step 3. Pseudorandom Signature Trees.

Tree of pseudorandom values:

The signing key is a PRF key $K$.

Populate the nodes with $r_x = PRF(K, x)$.

Use $r_x$ to derive the keys $(VK_x, SK_x) \leftarrow Gen(1^\lambda, r_x)$. 
Step 3. Pseudorandom Signature Trees.

Tree of pseudorandom values:

The signing key is a PRF key $K$.

Populate the nodes with $r_x = PRF(K, x)$.

Use $r_x$ to derive the keys $(VK_x, SK_x) \leftarrow Gen(1^\lambda; r_x)$. 
Step 3. Pseudorandom Signature Trees.

GOOD NEWS: Short signatures and small storage for the signer
Step 3. Pseudorandom Signature Trees.

BAD NEWS: 
Signer needs to keep a counter indicating which leaf (which tells her which secret key) to use next.
(Many-time) Signature Scheme

In four+ steps

Step 1. Stateful, Growing Signatures. Idea: Signature *Chains*

Step 2. How to Shrink the signatures. Idea: Signature *Trees*

Step 3. How to Shrink Alice’s storage.
   Idea: *Pseudorandom Trees*

Step 4. How to make Alice stateless.
   Idea: *Randomization*
Step 4. Statelessness via Randomization

Signature of a message $m$:
Pick a random leaf $r$. Use $VK_r$ to sign $m$.

$$\sigma_r \leftarrow \text{Sign}(SK_r, m)$$

Output $(r, \sigma_r, \text{authentication path for } VK_r)$
Step 4. Statelessness via Randomization

GOOD NEWS: No need to keep state.
Step 4. Statelessness via Randomization

Key Idea:
If the signer produces \( q \) signatures, the probability she picks the same leaf twice is \( \leq q^2/2^\lambda \).
(Many-time) Signature Scheme

In four+ steps

Step 1. Stateful, Growing Signatures. Idea: Signature *Chains*

Step 2. How to Shrink the signatures. Idea: Signature *Trees*

Step 3. How to Shrink Alice’s storage.
   Idea: *Pseudorandom Trees*

Step 4. How to make Alice stateless.
   Idea: *Randomization*

Step 5. Making the Signer Deterministic.

Key Idea:

Generate $r$ pseudo-randomly.

Have another PRF key $K'$ and let $r = PRF(K', m)$
That’s it for the construction.