Signature Schemes

Chester Rebeiro
IIT Madras
Recall : MACs

**MACs allow Bob to be certain that**
- the message has originated from Alice
- the message was not tampered during communication

**MAC cannot**
- prevent Bob from creating forgeries (i.e., messages in the name of Alice)
- cannot prove Authenticity to someone without sharing the secret key $K$

**Digital Signatures solve both these problems**
Digital Signatures

• A token sent along with the message that achieves
  – Authentication
  – Non-repudiation
  – Integrity

• Based on public key cryptography
Public key Certificates

Important application of digital signatures

Bob’s Certificate{
  Bob’s public key in plaintext
  Signature of the certifying authority
  other information
}

To communicate with Bob, Alice gets his public key from a trusted authority (TA)
A trusted authority could be a Government agency, Verisign, etc.

A signature from the TA, ensures that the public key is authentic.
Digital Signature

**Alice**

- Alice’s Private Key
- Message: x = “Attack at Dawn!!”
- Digital Signature: y = \( \text{sig}_K(x) \)

**Verifying Function**

- **Input**: Digital signature, message
- **Output**: true or false
  - true if signature valid
  - false otherwise

**Signing Function**

- **Input**: Message (x) and Alice’s private key
- **Output**: Digital Signature of Message
Digital Signatures (Formally)

**Definition**: A *signature scheme* is a five-tuple \((\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})\), where the following conditions are satisfied:

1. \(\mathcal{P}\) is a finite set of possible *messages*
2. \(\mathcal{A}\) is a finite set of possible *signatures*
3. \(\mathcal{K}\), the *keyspace*, is a finite set of possible *keys*
4. For each \(K \in \mathcal{K}\), there is a *signing algorithm* \(\text{sig}_K \in \mathcal{S}\) and a corresponding *verification algorithm* \(\text{ver}_K \in \mathcal{V}\). Each \(\text{sig}_K : \mathcal{P} \rightarrow \mathcal{A}\) and \(\text{ver}_K : \mathcal{P} \times \mathcal{A} \rightarrow \{\text{true}, \text{false}\}\) are functions such that the following equation is satisfied for every message \(x \in \mathcal{P}\) and for every signature \(y \in \mathcal{A}\):

\[
\text{ver}_K(x, y) = \begin{cases} 
\text{true} & \text{if } y = \text{sig}_K(x) \\
\text{false} & \text{if } y \neq \text{sig}_K(x).
\end{cases}
\]

A pair \((x, y)\) with \(x \in \mathcal{P}\) and \(y \in \mathcal{A}\) is called a *signed message*.
If Mallory can create a valid digital signature such that \( \text{ver}_K(x, y) = \text{TRUE} \) for a message not previously signed by Alice, then the pair \((x, y)\) forms a forgery.
Security Models for Digital Signatures

Assumptions

Goals of Attacker

• **Total break:**
  Mallory can determine Alice’s private key
  (therefore can generate any number of signed messages)

• **Selective forgery:**
  Given a message $x$, Mallory can determine $y$, such that $(x, y)$ is a valid signature from Alice

• **Existential forgery:**
  Mallory is able to create $y$ for some $x$, such that $(x, y)$ is a valid signature from Alice
Security Models for Digital Signatures

Assumptions

- **Key-only attack**: Mallory only has Alice’s public key (i.e. only has access to the verification function, \( \text{ver} \))
- **Known-message attack**: Mallory only has a list of messages signed by Alice \((x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), \ldots\).
- **Chosen-message attack**: Mallory chooses messages \(x_1, x_2, x_3, \ldots \ldots\) and tricks Alice into providing the corresponding signatures \(y_1, y_2, y_3\) (resp.)

Goals of Attacker

Weak

Strong
First Attempt making a digital signature (using RSA)

\[
\begin{align*}
b, n & \quad \text{public} \\
a, p, q & \quad \text{private} \\
n = pq; \ a \equiv b-1 \mod \phi(n)
\end{align*}
\]

\[
\begin{align*}
sig(x) & \{ \\
y & \equiv x^a \mod n \\
return (x, y) & 
\}
\end{align*}
\]

\[
\begin{align*}
ver(x, y) & \{ \\
if (x \equiv y^b \mod n) & \text{ return TRUE} \\
else & \text{ return FALSE}
\}
\end{align*}
\]

\[x \text{ is the message here} \]
\[\text{and } (x, y) \text{ the signature} \]
A Forgery for the RSA signature (First Forgery)

\[ b, n \quad \text{public} \]
\[ a, p, q \quad \text{private} \]
\[ n = pq; \quad a \equiv b-1 \mod \varphi(n) \]

\[ \text{sig}(x) \{ \]
\[ \quad y \equiv x^a \mod n \]
\[ \quad \text{return } (x, y) \]
\[ \} \]

\[ \text{ver}_k(x, y) \{ \]
\[ \quad (x, y) \]
\[ \quad \text{if } (x \equiv y^b \mod n) \quad \text{return } \text{TRUE} \]
\[ \quad \text{else return } \text{FALSE} \]
\[ \} \]

\[ \text{forgery}() \{ \]
\[ \quad \text{select a random } y \]
\[ \quad \text{compute } x \equiv y^b \mod n \]
\[ \quad \text{return } (x, y) \]
\[ \} \]

Key only, existential forgery
Second Forgery

Suppose Alice creates signatures of two messages $x_1$ and $x_2$

\[
y_1 = \text{sig}(x_1) \rightarrow y_1 \equiv x_1^a \mod n \quad (x_1, y_1)
\]

\[
y_2 = \text{sig}(x_2) \rightarrow y_2 \equiv x_2^a \mod n \quad (x_2, y_2)
\]

Mallory can use the *multiplicative property of RSA* to create a forgery

\[
(x_1 x_2 \mod n, y_1 y_2 \mod n) \quad \text{is a forgery}
\]

\[
y_1 y_2 \equiv x_1^a x_2^a \mod n
\]

Known message, existential forgery
RSA Digital Signatures

Incorporate a hash function in the scheme to prevent forgery

\[ \begin{align*}
\text{sig}(x) & \{ \\
& z = h(x) \\
& y \equiv z^a \mod n \\
& \text{return } (x, y) \\
\} \\
\end{align*} \]

\[ \begin{align*}
\text{ver}_K(x, y) & \{ \\
& z = h(x) \\
& \text{if } (z \equiv y^b \mod n) \text{ return TRUE} \\
& \text{else return FALSE} \\
\} \\
\end{align*} \]

x is the message here, (x, y) the signature and h is a hash function
How does the hash function help?

Preventing the First Forgery

```c
forgery()
{
    select a random $y$
    compute $z' \equiv y^b \mod n$
    compute $I^{st}$ preimage: $x s.t. \ z' = h(x)$
    return $(x, y)$
}
```

Forgery becomes equivalent to the first preimage attack on the hash function
How does the hash function help?

Preventing the Second Forgery

\[(x_1 x_2 \mod n, y_1 y_2 \mod n) \text{ is difficult} \]

\[y_1 y_2 \equiv h(x_1)^a h(x_2)^a \mod n \]

\[\neq x_1^a x_2^a \mod n \]

creating such a forgery is unlikely
How does the hash function help?

Another Forgery prevented

```java
def forgery(x, y):
    compute $h(x)$
    compute $\text{II}^{nd}$ preimage: find $x'$ s.t. $h(x) = h(x')$ and $x \neq x'$
    return $(x', y)$
```

Given a valid signature $(x,y)$ find $(x',y)$
creating such a forgery is equivalent to solving the $2^{nd}$ preimage problem of the hash function.
ElGamal Signature Scheme

• 1985
• Variant adopted by NIST as the DSA
  (DSA: standard for digital signature algorithm)
• Based on the difficult of the discrete log problem
ElGamal Signing

**Initialization**

Choose a large prime $p$
Let $\alpha \in \mathbb{Z}_p^*$ be a primitive element
Choose $a$  \hspace{0.5cm} (0 < a \leq p - 1)
Compute $\beta \equiv \alpha^a \mod p$

**Public Parameters** : $p$, $\alpha$, $\beta$

**Private key** : $a$

**Signing Message x**

```
sig(x){
    select a secret random $k$ s.t. $\gcd(k, p - 1) = 1$
    $\gamma \equiv \alpha^k \mod p$
    $\delta \equiv (x - a\gamma)k^{-1} \mod p - 1$
    $y = (\gamma, \delta)$
    return $(x, y)$
}
```

The use of a random secret $k$ for every signature makes ElGamal non-deterministic
## ElGamal Verifying

### Initialization
- Choose a large prime $p$
- Let $\alpha \in \mathbb{Z}_p^*$ be a primitive element
- Choose $a$ $\quad$ ($0 < a \leq p - 1$)
- Compute $\beta \equiv \alpha^a \mod p$
- Public Parameters : $p, \alpha, \beta$
- Private key : $a$

### Verifying Signature $(x,y)$

```
ver(x, (γ, δ)) {
    compute $t_1 \equiv \alpha^x \mod p$
    compute $t_2 \equiv \beta^γγ^δ \mod p$
    if ($t_1 = t_2$)
        return TRUE
    else
        return FALSE
}
```
ElGamal Correctness

**Signing Message x**

\[
\text{sig}(x) = \left\{ \begin{array}{l}
\text{select a secret random } k \\
\gamma = \alpha^k \mod p \\
\delta = (x - a\gamma)k^{-1} \mod p - 1 \\
y = (\gamma, \delta) \\
\text{return } (x, y)
\end{array} \right.
\]

**Initialization**

Choose a large prime \( p \)
Let \( \alpha \in \mathbb{Z}_p^* \) be a primitive element
Choose \( a \) \( (0 < a \leq p - 1) \)
Compute \( \beta = \alpha^a \mod p \)

Public Parameters: \( p, \alpha, \beta \)
Private key: \( a \)

**Verifying Signature (x,y)**

\[
\text{ver}(x, (\gamma, \delta)) = \left\{ \begin{array}{l}
\text{compute } t_1 = \alpha^x \mod p \\
\text{compute } t_2 = \beta^\gamma \gamma^\delta \mod p \\
\text{if } (t_1 = t_2) \text{ return TRUE} \\
\text{else return FALSE}
\end{array} \right.
\]

**First note that**

\[
a\gamma + k\delta \equiv x \mod (p - 1)
\]

\[
t_2 = \beta^\gamma \gamma^\delta \mod p \quad t_1 = \alpha^x \mod p
\]

\[
\equiv (\alpha^a)^\gamma + (\alpha^k)^\delta \mod p
\]

\[
\equiv \alpha^{ay+ks} \mod p
\]

\[
\equiv \alpha^x \mod p
\]

if the signature is valid, \( t_1 = t_2 \)
**Example**

Signature of message $x = 100$

\[
k = 213 \quad (chosen \ randomly)
\]
\[
k^{-1} \mod (p - 1) = 431
\]
\[
\gamma = \alpha^k \mod p
\]
\[
= 2^{213} \mod 467
\]
\[
= 29
\]
\[
\delta = (x - a\gamma)k^{-1} \mod p - 1
\]
\[
= (100 - 2 \cdot 29)431 \mod 466
\]
\[
= 51
\]

Verifying

\[
p = 467
\]
\[
\alpha = 2
\]
\[
a = 127
\]
\[
\beta \equiv \alpha^a \mod p
\]
\[
= 2^{127} \mod 467
\]
\[
= 132
\]

\[
\beta^\gamma \delta \mod p = 132^{29}29^{51} \mod 467 = 189
\]
\[
\alpha^x \mod p = 2^{100} \mod p = 189
\]

TRUE
Security of ElGamal Signature Scheme
(against Selective forgery)

Given an $x$, Mallory needs to find $(\gamma, \delta)$ such that $\text{ver}(x, (\gamma, \delta)) = \text{TRUE}$

Attempt 1
Choose a value for $\gamma$, then try to compute $\delta \; \text{s.t.} \; \beta^\gamma \gamma^\delta \equiv \alpha^x \text{ mod } p$
$\delta = \log_\gamma \alpha^x \beta^{-\gamma}$
This is the intractable discrete log problem

Attempt 2
Choose a value for $\delta$, then try to compute $\gamma \; \text{s.t.} \; \beta^\gamma \gamma^\delta \equiv \alpha^x \text{ mod } p$
This is not related to the discrete log problem. There is no known solution for this.

Attempt 3
Choose value for $\gamma$ and $\delta$ simultaneously, $\text{s.t.} \; \beta^\gamma \gamma^\delta \equiv \alpha^x \text{ mod } p$
No way known.
Security of ElGamal Signature Scheme
\textit{(against Existential forgery)}

Mallory needs to find an \((x, (\gamma, \delta))\) such that \(\text{ver}(x, (\gamma, \delta)) = \text{TRUE}\)

The one-parameter forgery

\begin{align*}
\text{choose some } i & \quad (0 \leq i \leq p - 2). \\
\text{form } \gamma & \equiv \alpha^i \beta \mod p \\
\delta & \equiv -\gamma \mod (p - 1) \\
x & \equiv i\delta \mod (p - 1). \\
\text{then, } \text{ver}(x, (\gamma, \delta)) = \text{TRUE} \\
\alpha^x & \equiv \beta^\gamma \gamma^\delta \mod p \\
\text{RHS} & \equiv \beta^\gamma (\alpha^i \beta)^\delta \mod p \\
& \equiv \beta^{\gamma+\delta} \alpha^{i\delta} \mod p \\
& \equiv \alpha^{\alpha\gamma+\delta} \alpha^{i\delta} \mod p \\
& \equiv \alpha^{\alpha\gamma-\gamma+i\delta} \mod p \\
& \equiv \alpha^{i\delta} \mod p \\
& \equiv \alpha^x \mod p = \text{LHS}
\end{align*}
Security of ElGamal Signature Scheme (against Existential forging)

Mallory needs to find an \((x, (\gamma, \delta))\) such that \(ver(x, (\gamma, \delta)) = TRUE\)

The two-parameter forgery

choose some \(i, j\)  \((0 \leq i, j \leq p - 2; \gcd(j, p - 1) = 1)\)

form  \(\gamma \equiv \alpha^i \beta^j \mod p\)

\[\delta \equiv -\gamma^{-1} \mod (p - 1)\]

\[x \equiv \gamma j^{-1} \mod (p - 1)\]

then, \(ver(x, (\gamma, \delta)) = TRUE\)

Prevent Existential Forgeries by hashing the message
Improper use of ElGamal’s Signature Scheme

1. What if \( k \) is not a secret?

\[
\text{if } \gcd(y, p - 1) = 1 \text{ then}
\]
secret \( a \) can be computed as follows
\[
a = (x - k\delta)y^{-1} \mod (p - 1).
\]

The secret key ‘\( a \)’ is retrieved and Mallory can create many forgeries.

\[
sig(x)\
\begin{align*}
\text{select a secret random } k \\
\gamma & \equiv \alpha^k \mod p \\
\delta & \equiv (x - a\gamma)k^{-1} \mod p - 1 \\
y & = (\gamma, \delta) \\
\text{return } (x, y)
\end{align*}
\]
Improper use of ElGamal’s Signature Scheme

2. What if \( k \) is reused?

Let's say we have two different messages \( x_1 \) and \( x_2 \) signed with the same \( k \).

The signatures are \((\gamma, \delta_1)\) and \((\gamma, \delta_2)\) then,

\[
\beta^\gamma \gamma^{\delta_1} \equiv \alpha^{x_1} \pmod{p}
\]

\[
\beta^\gamma \gamma^{\delta_2} \equiv \alpha^{x_2} \pmod{p}.
\]

dividing

\[
\alpha^{x_1-x_2} \equiv \gamma^{\delta_1-\delta_2} \pmod{p}.
\]

Representing in terms of \( \alpha \)

\[
\alpha^{x_1-x_2} \equiv \alpha^{k(\delta_1-\delta_2)} \pmod{p},
\]

=>

\[
x_1 - x_2 \equiv k(\delta_1 - \delta_2) \pmod{p - 1}.
\]
Improper use of ElGamal’s Signature Scheme

\[ x_1 - x_2 \equiv k(\delta_1 - \delta_2) \pmod{p - 1}. \]

Now let \( d = \gcd(\delta_1 - \delta_2, p - 1) \). Since \( d \mid (p - 1) \) and \( d \mid (\delta_1 - \delta_2) \), it follows that \( d \mid (x_1 - x_2) \). Define

\[
\begin{align*}
x' &= \frac{x_1 - x_2}{d} \\
\delta' &= \frac{\delta_1 - \delta_2}{d} \\
p' &= \frac{p - 1}{d}.
\end{align*}
\]

Then the congruence becomes:

\[ x' \equiv k\delta' \pmod{p'} . \]

Since \( \gcd(\delta', p') = 1 \), we can compute

\[ \epsilon = (\delta')^{-1} \pmod{p'} . \]

Then value of \( k \) is determined modulo \( p' \) to be

\[ k = x'\epsilon \pmod{p'} . \]

This yields \( d \) candidate values for \( k \):

\[ k = x'\epsilon + ip' \pmod{(p - 1)} \]

for some \( i, \ 0 \leq i \leq d - 1 \). Of these \( d \) candidate values, the (unique) correct one can be determined by testing the condition

\[ \gamma \equiv \alpha^k \pmod{p} . \]
ElGamal Signature Length

- Generally p is a prime of length 1024 bits
- The signature comprises of \((\gamma, \delta)\) which is of length 2048 bits

Schnorr’s Signature Scheme is a modification of the ElGamal signature scheme with signatures of length around 320 bits
**Initialization**

Choose a large prime $p$ (1024 bit)
Choose another prime $q$ (160 bit) s.t. $q \mid p - 1$
Find $\alpha$ of order $q$ ($\alpha$ creates a subgroup of order $q$)
Choose $a$ $(0 < a \leq q - 1)$
Compute $\beta \equiv \alpha^a \mod p$

Public Parameters: $p$, $q$, $\alpha$, $\beta$
Private key: $a$

- Choose some $\alpha$
- And compute $\alpha^{(p-1)/q} \mod p$
### DSA (Signing Function)

#### Initialization

- Choose a large prime $p$ (1024 bit)
- Choose another prime $q$ (160 bit) s.t. $q | p - 1$
- Find $\alpha$ of order $q$ (\alpha creates a subgroup of order $q$)
- Choose $a$ \((0 < a \leq q - 1)\)
- Compute $\beta \equiv a^\alpha \mod p$

Public Parameters: $p, q, \alpha, \beta$

Private key: $a$

#### Signing Message $x$

```plaintext
sig(x) {
    select a secret random $k$ s.t. $\gcd(k, q) = 1$
    $\gamma \equiv (\alpha^k \mod p) \mod q$
    $\delta \equiv (SHA(x) + a\gamma)k^{-1} \mod q$
    $y = (\gamma, \delta)$
    return $(x, y)$
}
```

The use of a random secret $k$ for every signature makes ElGamal non-deterministic.
**DSA (Verifying Function)**

### Initialization

- Choose a large prime \( p \) (1024 bit)
- Choose another prime \( q \) (160 bit) s.t. \( q \mid p - 1 \)
- Find \( \alpha \) of order \( q \) (\( \alpha \) creates a subgroup of order \( q \))
- Choose \( a \) (0 < \( a \leq q - 1 \))
- Compute \( \beta \equiv \alpha^a \mod p \)

Public Parameters: \( p, q, \alpha, \beta \)

Private key: \( a \)

### Signing Message \( x \)

\[
\text{sig}(x) \{
\text{select a secret random } k \text{ s.t. } \gcd(k, q) = 1 \\
\gamma \equiv (\alpha^k \mod p) \mod q \\
\delta \equiv (SHA(x) + a\gamma)k^{-1} \mod q \\
y = (\gamma, \delta) \\
\text{return } (x, y)
\}
\]

### Verifying Signature

\[
\text{ver}(x, (\gamma, \delta)) \{
\text{compute } w \equiv \delta^{-1} \mod q \\
\text{compute } t_1 \equiv w \cdot SHA(x) \mod q \\
\text{compute } t_2 \equiv w \cdot \gamma \mod q \\
\text{compute } v \equiv (\alpha^t_1 \cdot \beta^t_2 \mod p) \mod q \\
\text{if } (v \equiv \gamma \mod q) \text{ return TRUE} \\
\text{else return FALSE}
\}
\]
**DSA (Correctness)**

**Initialization**

Public Parameters: \( p, q, \alpha, \beta \ (\beta \equiv \alpha^a \mod p) \)

Private key: \( a \)

**Signing Message \( x \)**

\[
sig(x)\{
    \text{select a secret random } k \text{ s.t. } \gcd(k, q) = 1 \\
    \gamma \equiv (\alpha^k \mod p) \mod q \\
    \delta \equiv (\text{SHA}(x) + a\gamma)k^{-1} \mod q \\
    y = (\gamma, \delta) \\
    \text{return } (x, y)
\}
\]

**Verifying Signature**

\[
\text{ver}(x, (\gamma, \delta))\{
    \text{compute } w \equiv \delta^{-1} \mod q \\
    \text{compute } t_1 \equiv w \cdot \text{SHA}(x) \mod q \\
    \text{compute } t_2 \equiv w \cdot \gamma \mod q \\
    \text{compute } v \equiv (\alpha^{t_1} \cdot \beta^{t_2} \mod p) \mod q \\
    \text{if } (v \equiv \gamma \mod q) \text{ return TRUE} \\
    \text{else return FALSE}
\}
\]

\[
\delta \equiv (\text{SHA}(x) + a\gamma)k^{-1} \mod q \\
k \equiv (\text{SHA}(x) + a\gamma)\delta^{-1} \mod q \\
= (w\text{SHA}(x) + wa\gamma) \mod q \\
k \equiv (t_1 + at_2) \mod q
\]

\[
\alpha^k \equiv \alpha^{(t_1 + at_2) \mod q} \mod p \\
\alpha^k \equiv \alpha^{t_1} \beta^{t_2} \mod p \\
\text{Take } \mod q \text{ on both sides} \\
\gamma \equiv (\alpha^{t_1} \beta^{t_2} \mod p) \mod q
\]
Security of DSA

• There are two ways to attack the DSA (attempt to recover the secret $a$)
  – Target the large cyclic group $\mathbb{Z}_p$
  – Target the smaller group $\mathbb{Z}_q$

Could you techniques such as Index Calculus. For a 1024 bit $p$, this method offers security of 80 bits

Cannot apply Index Calculus relies on Pollard rho for solving the discrete log, For 160 bit $q$, this offers security of 80 bits
Security of DSA

• There are two ways to attack the DSA (attempt to recover the secret $a$)
  – Target the large cyclic group $Z_p$
  – Target the smaller group $Z_q$

Could you techniques such as Index Calculus. For a 1024 bit $p$, this method offers security of 80 bits

Cannot apply Index Calculus relies on Pollard rho for solving the discrete log, For 160 bit $q$, this offers security of 80 bits

Thus the size of $p$ dictates the size of $q$. 

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>160</td>
<td>320</td>
</tr>
<tr>
<td>2048</td>
<td>224</td>
<td>448</td>
</tr>
<tr>
<td>3072</td>
<td>256</td>
<td>512</td>
</tr>
</tbody>
</table>
Schnorr Signature Scheme
(uses a hash function to get smaller signatures)

Initialization
Choose a large prime \( p \) (of size 1024 bits)
Choose a smaller prime \( q \) (of size 160 bits) and \( q \mid (p-1) \)
Let \( \alpha_0 \in \mathbb{Z}_p^* \) be a primitive element

then \( \alpha = \alpha_0^{(p-1)/q} \mod p \) is the \( q \)th root of 1 mod \( p \)
Choose \( a \) randomly from \( (0 \leq a < q) \)
Compute \( \beta = \alpha^a \mod q \)

Private: \( a \)
Private: \( \alpha, \beta, p, q \)

Signing Message \( x \)
\[
\text{sig}(x)\{ \\
\quad \text{select a secret random } k \text{ s.t. } 1 \leq k \leq q-1. \\
\quad \gamma = h(x \parallel \alpha^k \mod p) \\
\quad \delta = k + a\gamma \mod p \\
\quad y = (\gamma, \delta) \\
\quad \text{return } (x, y) \\
\}
\]

Verifying Signature \((x, y)\)
\[
\text{ver}(x, (\gamma, \delta))\{ \\
\quad \text{compute } t_1 \equiv h(x \parallel \alpha^\delta \beta^{-\gamma} \mod p) \\
\quad \text{if } (t_1 = \gamma) \text{ return } \text{TRUE} \\
\quad \text{else return } \text{FALSE} \\
\}
\]