"Optimal" One-time Signature of an n-bit message

\[ m - \text{input} \]
\[ h - \text{message digest}, h = \text{hash}(m) \]
\[ n = |h|\text{or } \log(\max(h)) \]
\[ w = \log(n) - 1 \]
\[ z = \min(\#1\text{bits}(h), \#0\text{bits}(h)) \]
\[ SK = X_0, X_1, [x_0, \ldots, x_{n/2-1}], [x'_{0,0}, x'_{1,1}, \ldots, x'_{w-1,0}, x'_{w-1,1}] \]
\[ PK = Y_0, Y_1, [y_0, \ldots, y_{n/2-1}], [y'_{0,0}, y'_{1,1}, \ldots, y'_{w-1,0}, y'_{w-1,1}] \]
\[ f(x) = y \quad (\text{for all } x, y) \]
\[ \text{Signature} : \left[ X_0 \mid X_1, [x_{i_0}, \ldots, x_{i_{z-1}}], [x'_{0,0}, x'_{1,1}, \ldots, x'_{w-1,0}, x'_{w-1,1}] \right] \]
**One-time Signature of n-bit message**

*Example:*

\[ h = 0100110100011000 \]

\[ n = 16, w = 3, z = 6 = 110 \]

\[ \text{SIG}(m) = m, X_1, (x_1, x_4, x_5, x_7, x_{11}, x_{12}), (x_{0,1}, x_{1,1}, x_{2,0}) \]

**Is it possible to forge a valid message?**

---

**One-time Signature Chains**

How to sign \( n \) 2-bit messages

\[ Y_{0,0} = f(\ldots f(x_{0,0})\ldots) \]

\[ Y_{0,1} = f(\ldots f(x_{0,1})\ldots) \]

\[ Y_{1,0} = f(\ldots f(x_{1,0})\ldots) \]

\[ Y_{1,1} = f(\ldots f(x_{1,1})\ldots) \]

Publish all \( Y \) values

Suppose: \( M_1 = 01, M_2 = 11, \ldots, M_n = 00 \)

\[ \text{Sig}(M_1) = f(\ldots f(x_{0,1})\ldots), f(\ldots f(x_{1,0})\ldots) \]

\[ \text{Sig}(M_2) = f(\ldots f(x_{0,1})\ldots), f(\ldots f(x_{1,1})\ldots) \]

\[ \ldots \]

\[ \text{Sig}(M_n) = x_{0,0}, x_{1,0} \]

Example app: e-coins...

Used in S/Key (OTP, Opie)
Rabin’s One-Time Signatures

- \( K_i \) - random secret keys \( (0 < i < 2n) \)
- Signature: \( S_i = E(K_i, H(\text{MSG})) \)
- Public keys: \( P_i = E(K_i, i) \)

\[ \Omega = \{ R_i \mid 0 < i < 2n \} \]

\[ \forall R_i \in \Omega \]

\[ D(K_R, S_R) = H(\text{MSG}) \]
\[ D(K_R, P_R) = i \]

\( n \) - security parameter
- Inefficient
- On-line

Pick \( n \) random numbers:

- \( R_i, 0 < i < 2n \)

\( \Sigma = \{ S_i \mid 0 < i < 2n \} , \)
\[ P = \{ P_i \mid 0 < i < 2n \}, \]

Rabin’s One-Time Signatures (contd)

DISPUTES

\[ \{ K_i \mid 0 < i < 2n \} \]

Has \( P \)

\[ MSG, \ \Sigma \]

1) \( D(K_i, P_i) = i \)
   if not, Bob wins

2) \( E(K_i, H(\text{MSG})) = S_i \)
   for at most \( n \) \( S_i \)
   Alice wins, else Bob wins

Note: Alice, Bob communicate with Court via authentic channels
Public Key Cryptography

- Merkle
- Hellman
- Diffie
- Rivest
- Shamir
- Adleman
- Rabin
- McEliece
- ElGamal

- RSA
- Diffie-Hellman KE
- Rabin
- El Gamal
- Elliptic Curve Duals
- Gillou-Quisquater
- Fiat/Shamir
- Knapsacks
- etc., etc.

What's the main idea?

- Everyone can encrypt for you
- No one can decrypt but you
- No pre-established pairwise secret info
- How is this different from conventional (shared-key) encryption?
- Do you know who encrypts for you?
- Does encryptor know who will receive?
RSA (1976-8)

Let $n = pq$ where $p \neq q$ - (large) primes
$e, d \in \mathbb{Z}_n$ and $ed = 1 \mod \Phi(n)$

note that: $\Phi(n) = (p - 1)(q - 1)$

Secrets: $p, q, d$

Publics: $n, e$

Encryption: message $= m$
$E(x) = y = m^e \mod n$

Decryption: ciphertext $= y$
$D(y) = x' = y^d \mod n$

Why does it all work?

$x \in \mathbb{Z}_n^*$

$x^{ed} = x^{1 \mod \Phi(n)} \mod n = x^{c^*\Phi(n)+1} \mod n = x$

But, recall that:

$g^{\Phi(n)} = 1 \mod n$ (Lagrange)
How does it all work?

Example: \( p=17 \ q=13 \ n=221 \ \phi(n)=192 = 3^4 \cdot 2 \)
pick \( e=5, \ d=77 \)
\( x=5, \ E(x)=3125 \mod 221 = 31 \)
\( D(y)=31^{77}=6.83676142775442000196395599558e+114 \mod 221 = 5 \)

Example: \( p=5 \ q=7 \ n=35 \ \phi(n)=24 = 3 \cdot 2^3 \)
pick \( e=11, \ d=11 \)
\( x=2, \ E(x)=2048 \mod 35 = 18=y \)
\( y=18, \ D(y)=6.426841007923e+13 \mod 35 = 2 \)

Why is it secure?

Conjecture: breaking RSA is \textit{polynomially equivalent} to factoring \( n \). Recall that \( n \) is very, very large!

Why: \( n \) has unique factors \( p,q \)
Given \( p,q \) computing \( \phi(n) = (p-1)(q-1) \) is easy:
\[
ed \equiv 1 \mod \phi(n)
\]
Use extended Euclidean!
Exponentiation Costs

- Integer multiplication -- $O(b^2)$  \( b \) -- bit length of base \( m \)
- Modular reduction -- $O(b^2)$
- Thus, modular multiplication -- $O(b^2)$
- Modular exponentiation -- $m^e \mod n$
- Naïve method: \( e \)-1 modular products -- $O(b^2e)$
  BUT \( e \) is large, as large as?

- Let \( L = |e| \) (e.g., \( L=1024 \) for 1024-bit RSA exponent)
- We can assume \( b \) and \( L \) are close in length
- Square-and-multiply method works in $O(b^3)$ time…
  $O(b^2*2L)$

Square-and-Multiply

\[
\text{compute } m^e \mod n \\
\begin{align*}
l & = \text{sizeof}(n); \\
\text{temp} & = 1; \\
\text{for} \ (i = l - 1; i >= 0; i --) \\
& \quad \{ \\
& \quad \quad \text{temp}^* = \text{temp}; \\
& \quad \quad \text{temp} \mod n; \\
& \quad \quad \text{if} \ (e[i]) \\
& \quad \quad \quad \{ \\
& \quad \quad \quad \quad \text{temp}^* = m; \\
& \quad \quad \quad \quad \text{temp} \mod n; \\
& \quad \quad \} \\
& \quad \}\end{align*}
\]

From left to right in \( e \)

- Example 1: \( e=100 \)
- Example 2: \( e=10000000 \)
- Example 3: \( e=11111111 \)

\( N=35, e=11, m=2 \)
Let:

\[ d_p = d \mod (p - 1) \]
\[ d_q = d \mod (q - 1) \]

compute:

\[ M_p = C^{d_p} \mod p \]
\[ M_q = C^{d_q} \mod q \]

and solve:

\[ M = M_p \mod p \]
\[ M = M_q \mod q \]

\[ M = [M_p q(q^{-1} \mod p) + M_q p(p^{-1} \mod q)] \mod (pq) \]