Week 10 -11 : Public Key Cryptosystem and Digital Signatures
1. Public Key Encryptions
RSA, ElGamal,
RSA- PKC(1/3)

- 1st public key cryptosystem
- Believed to be secure if IFP is hard and worldwide standard for last 30 years
RSA- PKC(2/3)

- **Key generation (KeyGen)**
  - Select two large (1,024 bits or larger) primes \( p, q \)
  - Compute modulus \( n = pq \), and \( \phi(n) = (p-1)(q-1) \)
  - Pick an integer \( e \) relatively prime to \( \phi(n) \), \( \gcd(e, \phi(n)) = 1 \)
  - Compute \( d \) such that \( ed = 1 \mod \phi(n) \) How??
  - Public key \((n, e)\) : public
  - Private key \(d\) : keep secret (may hold \( p \) and \( q \) securely.)

- **Encryption(Enc) / Decryption (Dec)**
  - E: \( C = M^e \mod n \) for \( 0 < M < n \)
  - D: \( M = C^d \mod n \)
  - Proof) \( C^d = (M^e)^d = M^{ed} = M^{k\phi(n) + 1} = M \{M^{\phi(n)}\}^k = M \)

- **Special Property**
  - \((M^e \mod n)^d \mod n = (M^d \mod n)^e \mod n\) for \( 0 < M < n \)
RSA as Trapdoor One-way Function

Message $M$ \rightarrow Ciphertext $C = f(M) = M^e \mod n$

Ciphertext $C$ \rightarrow Private key (trapdoor information) \rightarrow Message $M = f^{-1}(C) = C^d \mod n$

Public key

$n = pq$ ($p$ & $q$: primes)

$ed = 1 \mod (p-1)(q-1)$
RSA - PKC (3/3)

• Key Generation
  – \( p = 3, \ q = 11 \)
  – \( n = pq = 33, \ \phi(n) = (p-1)(q-1) = 2 \times 10 = 20 \)
  – \( e = 3 \) s.t. \( \gcd(e, \ \phi(n)) = (3, 20) = 1 \)
  – Choose \( d \) s.t. \( ed = 1 \mod \phi(n) \), \( 3d = 1 \mod 20 \), \( d = 7 \)
  – Public key \( = \{e, n\} = \{3, 33\} \), private key \( = \{d\} = \{7\} \)

• Encryption
  – \( M = 5 \)
  – \( C = M^e \mod n = 5^3 \mod 33 = 26 \)

• Decryption
  – \( M = C^d \mod n = 26^7 \mod 33 = 5 \)
Exercise

Let’s practice RSA key generation, encryption, and decryption

1) p=5, q=7 (by hand calculation, Quiz!!) if M=3
2) p=2,357, q=2,551 (using big number calculator) if M=5,000
3) p=885,320,963, q=238,855,417 (using big number calculator) if M=10,000

1. Key generation

2. Encryption

3. Decryption
Selecting Primes $p$ and $q$

- Idea: Prevent from feasible factorization
  
  1. $|p| \approx |q|$ to avoid ECM (Elliptic Curve Method for factoring)
  
  2. $p-q$ must be large to avoid trial division
  
  3. $p$ and $q$ are strong prime
     - $p-1$ has large prime factor $r$ (Pollard’s $p-1$)
     - $p+1$ has large prime factor (William’s $p+1$)
     - $r-1$ has large prime factor (Cyclic attack)
Integer Factorization Problem (IFP)

- Problem: Given a composite number $n$, find its prime factors

  - Application: Used to construct RSA-type public key cryptosystems

- (Probabilistic sub-exponential) Algorithms to solve IFP
  - Quadratic sieve
  - General Number Field Sieve
  - etc.
Quadratic Sieve (1/3)

- Factor n (=pq) using the quadratic sieve algorithm

- Basic principle:
  Let n be an integer and suppose there exist integers x and y with
  \( x^2 = y^2 \pmod{n} \), but \( x \neq \pm y \pmod{n} \). Then \( \gcd(x-y, n) \) gives a
  nontrivial factor of n.

- Example
  Consider n=77
  72=-5 mod 77, 45=-32 mod 77
  72*45 = (-5)*(-32) mod 77
  \( 2^3*3^4*5 = 2^5*5 \pmod{77} \)
  \( 9^2 = 2^2 \pmod{77} \)
  \( \gcd(9-2,77)=7, \gcd(9+2,77)=11 \)
  77=11*7  Factorization
Quadratic Sieve (2/3)

Example: factor \( n = 3837523 \).

Observe
\[
\begin{align*}
9398^2 &= 5^5 \times 19 \pmod{3837523} \\
19095^2 &= 2^2 \times 5 \times 11 \times 13 \times 19 \pmod{3837523} \\
1964^2 &= 3^2 \times 13^3 \pmod{3837523} \\
17078^2 &= 2^6 \times 3^2 \times 11 \pmod{3837523}
\end{align*}
\]

Then, we have
\[
(9398 \times 19095 \times 1964 \times 17078)^2 = (2^4 \times 3^2 \times 5^3 \times 11 \times 13^2 \times 19)^2 \pmod{3837523}
\]
\[
2230387^2 = 2586705^2 \pmod{3837523}
\]
Compute \( \text{gcd}(2230387-2586705, 3837523) \Rightarrow 1093 \pmod{3837523} \)
\[
3837523 / 1093 = 3511 \pmod{3837523}
\]

\( 3837523 = 1093 \times 3511 \) \( \Leftarrow \) Note that Verification is easy !!
Quadratic Sieve (3/3)

1. Initialization: a sequence of quadratic residues $Q(x) = (m+x)^2 - n$ is generated for small values of $x$ where $m = \lfloor \sqrt{n} \rfloor$.

2. Forming the factor base: the base consists of small primes. $FB = \{-1, 2, p_1, p_2, \ldots, p_{t-1}\}$

3. Sieving: the quadratic residues $Q(x)$ are factored using the factor base till $t$ full factorizations of $Q(x)$ have been found.

4. Forming and solving the matrix: Find a linear combination of $Q(x)$’s which gives the quadratic congruence. The congruence gives a nontrivial factor of $n$ with the probability $\frac{1}{2}$.

http://www.answers.com/topic/quadratic-sieve?cat=technology
General Number Field Sieve (GNFS)

- Most efficient algorithm known for factoring integers larger than 100 digits.
- Asymptotic running time: sub-exponential

\[ L_n \left[ \frac{1}{3}, 1.526 \right] = O \left( e^{(1.526 + o(1))(\ln n)^{1/3} (\ln \ln n)^{2/3}} \right) \]

**Complexity of algorithm**

\[ L_n [\alpha, c] = O \left( e^{c (\ln n)^{\alpha} (\ln \ln n)^{1-\alpha}} \right) \]

- If \( \alpha = 0 \), polynomial time algorithm
- If \( \alpha \geq 1 \), exponential time algorithm
- If \( 0 < \alpha < 1 \), sub-exponential time algorithm
## RSA Challenge

<table>
<thead>
<tr>
<th>Digits</th>
<th>Year</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-100</td>
<td>'91.4. 7</td>
<td>Q.S.</td>
</tr>
<tr>
<td>RSA-110</td>
<td>'92.4. 75</td>
<td>Q.S.</td>
</tr>
<tr>
<td>RSA-120</td>
<td>'93.6. 830</td>
<td>Q.S.</td>
</tr>
<tr>
<td>RSA-129</td>
<td>'94.4.(AC94) 5,000</td>
<td>Q.S.</td>
</tr>
<tr>
<td>RSA-130</td>
<td>'96.4.(AC96) ?</td>
<td>NFS</td>
</tr>
<tr>
<td>RSA-140</td>
<td>'99.2 (AC99) ?</td>
<td>NFS</td>
</tr>
<tr>
<td>RSA-155</td>
<td>'99.8 8,000</td>
<td>GNFS</td>
</tr>
<tr>
<td>RSA-160</td>
<td>'03.1</td>
<td>Lattice Sieving+HW</td>
</tr>
<tr>
<td>RSA-174</td>
<td>'03.12</td>
<td>Lattice Sieving +HW</td>
</tr>
<tr>
<td>RSA-200</td>
<td>'05.5</td>
<td>GNFS+HW</td>
</tr>
</tbody>
</table>

*MIPS : 1 Million Instruction Per Second for 1 yr = 3.1 \times 10^{13} \text{ instruction}.*

• Expectation: 1,024-bit by 2018 !!!!*
We have factored RSA-200 by GNFS.

The factors are:

\[ p = 3532461934402770121272604978198464368671197400197625023649303468776121253679423200058547956528088349 \]

\[ q = 7925869954478333033347085841480059687737975857364219960734330341455767872818152135381409304740185467 \]

http://www.loria.fr/~zimmerma/records/rsa200
RSA-232 (768 bit)

Factorization of a 768-bit RSA modulus

version 1.21, January 13, 2010

Thorsten Kleinjung\textsuperscript{1},
Kazumaro Aoki\textsuperscript{2}, Jens Franke\textsuperscript{3}, Arjen K. Lenstra\textsuperscript{1}, Emmanuel Thomé\textsuperscript{4},
Joppe W. Bos\textsuperscript{1}, Pierrick Gaudry\textsuperscript{4}, Alexander Kruppa\textsuperscript{4}, Peter L. Montgomery\textsuperscript{5,6},
Dag Arne Osvik\textsuperscript{1}, Herman te Riele\textsuperscript{6}, Andrey Timofeev\textsuperscript{6}, and Paul Zimmermann\textsuperscript{4}

\textsuperscript{1} EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland
\textsuperscript{2} NTT, 3-9-11 Midori-cho, Musashino-shi, Tokyo, 180-8585 Japan
\textsuperscript{3} University of Bonn, Department of Mathematics, Beringstraße 1, D-53115 Bonn, Germany
\textsuperscript{4} INRIA CNRS LORIA, Équipe CARAMEL - bâtiment A, 615 rue du jardin botanique,
\textsuperscript{5} Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
\textsuperscript{6} CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

using the hard disk and one core on
compute the exponents of all prime
quare root using the implementation
ex-core processor. The first one (and
20:16 GMT on December 12, 2009:

1770479498371376856891
1743087737814467999489
3227915816434308764267
3810270092798736308917.
ctorizations of the factors ±1 can be

\textbf{Abstract}. This paper reports on the factorization of the 768-bit number RSA-768 by the number field sieve factoring method and discusses some implications for RSA.

\textbf{Keywords}: RSA, number field sieve
Security of RSA (1/2)

- Common Modulus attack:
  - If multiple entities share the same modulus \( n=pq \) with different pairs of \((e_i, d_i)\), this is not secure.
    
    **Do not share the same modulus!**

- Cryptanalysis: If the same message \( M \) was encrypted to different users

  User \( u_1 \) : \( C_1 = M^{e_1} \mod n \)

  User \( u_2 \) : \( C_2 = M^{e_2} \mod n \)

  If \( \gcd(e_1, e_2) = 1 \), there are \( a \) and \( b \) s.t. \( ae_1 + be_2 = 1 \mod n \) then,

  \[
  (C_1)^a(C_2)^b \mod n = (M^{e_1})^a(M^{e_2})^b \mod n = M^{ae_1+be_2} \mod n = M \mod n
  \]
Security of RSA(2/2)

❖ Cycling attack

If \( f(f(\ldots f(M))) = f(M) \) where \( f(M) = M^e \mod n \)?
If a given ciphertext appears after some iterations, we can recover the plaintext at collusion point.

E.g., Let \( C = M^e \mod n \)

If \( (((C^e)^e)^e) \mod n = C^{e^k} \mod n = C \),
then \( C^{e^{k-1}} \mod n = M \) for some \( k \).

❖ Multiplicative attack (homomorphistic property of RSA)

\( (M_1^e) \times (M_2^e) \mod n = (M_1 \times M_2)^e \mod n \)
Security of PKC

- Security goals
  - One-wayness (OW): the adversary who sees a ciphertext is not able to compute the corresponding message.
  - Indistinguishability (IND): observing a ciphertext, the adversary learns nothing about the plaintext. Also known as semantic security.
  - Non-malleability (NM): observing a ciphertext for a message $m$, the adversary cannot derive another ciphertext for a meaningful plaintext $m'$ related to $m$.

- Original RSA encryption is not secure since
  - IND: deterministic encryption
  - NM: for example, from $c=m^e$, $c'=2^e c = (2m)^e$ is easily obtained. It cannot be used in bidding scenario.
Formal Definition of IND

\[ b \in \mathbb{R}\{0,1\} \]

Challenge: \( C = E(m_b) \)

The adversary wins if he guesses \( b \) correctly with a probability significantly greater than \( \frac{1}{2} \).
Security Def. of PKC

- Assume the existence of Decryption Oracle
  - Mimics an attacker’s access to the decryption device

- Attacking Method
  - Chosen Plaintext Attack (CPA): the adversary can encrypt any plaintext of his choice. In PKC, this is always possible.
  - Non-adaptive Chosen Ciphertext Attack (CCA1): the attacker has access to the decryption oracle before he sees a ciphertext that he wishes to manipulate (aka. lunchtime attack)
  - Adaptive Chosen Ciphertext Attack (CCA2): the attacker has access to the decryption oracle before and after he sees a ciphertext $c$ that he wishes to manipulate (but, he is not allowed to query the oracle about the target ciphertext $c$.)
Making RSA to IND-CCA2

- **RSA encryption without padding**
  - Deterministic encryption
  - Multiplicative property: \( m_1^e \cdot m_2^e = (m_1m_2)^e \mod n \)
  - Many attacks possible
  - Redundancy checking is required

- **RSA encryption with OAEP**
  - RSA encryption after OAEP (Optimal Asymmetric Encryption Padding)
  - Proposed by Bellare and Rogaway
  - Probabilistic encoding of message before encryption
  - RSA becomes a probabilistic encryption
  - Secure against IND-CCA2
RSA with OAEP

- **OAEP → RSA encryption**
  
  \[
  \begin{align*}
  s &= m \oplus G(r) \\
  t &= r \oplus H(s) \\
  c &= E(s, t)
  \end{align*}
  \]

  Encryption padding

  RSA encryption

- **RSA decryption → OAEP**
  
  \[
  \begin{align*}
  (s, t) &= D(c) \\
  r &= t \oplus H(s) \\
  m &= s \oplus G(r)
  \end{align*}
  \]

  RSA decryption

  Decryption padding

(Note) OAEP looks like a kind of Feistel network
PKCS #1 v2.0, v2.1..
Diffie-Hellman / ElGamal-type Systems

- **Domain parameter generation**
  - Based on the hardness of DLP
  - Generate a large (1,024 bits or larger) prime $p$
  - Find generator $g$ that generates the cyclic group $\mathbb{Z}_p^*$
  - Domain parameter = \{\(p, g\)\}

- **Key generation**
  - Pick a random integer $x \in [1, p-1]$
  - Compute $y = g^x \mod p$
  - Public key (\(p, g, y\)) : public
  - Private key $x$ : keep secret

- **Applications**
  - Public key encryption
  - Digital signatures
  - Key agreement
Discrete Logarithm Problem (DLP)

- Problem:
  Given $g$, $y$, and prime $p$, find an integer $x$, if any, such that $y = g^x \mod p \ (x = \log_g y)$

- Application: Used to construct Diffie-Hellman & ElGamal-type public key systems: DH, DSA, KCDSA ...

- Algorithms to solve DLP:
  - Shank’s Baby Step Giant Step
  - Index calculus
Shank’s Baby Step, Giant Step algorithm

➢ Problem: find an integer \( x \), if any, such that \( y = g^x \mod p \) \((x=\log_g y)\)

➢ Algorithm

1. Choose an integer \( N = \lceil \sqrt{p-1} \rceil \)
2. Computes \( g^j \mod p \), for \( 0 \leq j < N \)
3. Computes \( yg^{-Nk} \mod p \), for \( 0 \leq k < N \)
4. Look for a match between the two lists. If a match is found,
   \[ g^j = yg^{-Nk} \mod p \]
   Then \( y = g^x = g^{j+Nk} \)

We solve the DLP. \( x = j + Nk \)
Index Calculus (1/2)

Problem: find an integer $x$, if any, such that $y = g^x \mod p$ ($x=\log_g y$)

Algorithm

1. Choose a factor base $S = \{p_1, p_2, \ldots p_m\}$ which are primes less than a bound $B$.
2. Collect linear relations
   1. Select a random integer $k$ and compute $g^k \mod p$
   2. Try to write $g^k$ as a product of primes in $S$

$$g^k = \prod_i p_i^{a_i} \mod p, \text{ then } k = \sum_i a_i \log_p p_i \mod p - 1$$

3. Find the logarithms of elements in $S$ solving the linear relations
4. Find $x$
   For a random $r$, compute $yg^r \mod p$ and try to write it as a product of primes in $S$.

$$yg^r = \prod_i p_i^{b_i} \mod p, \text{ then } x = -r + \sum_i b_i \log_p p_i \mod p - 1$$
Index Calculus (2/2)

- Example: Let $p=131$, $g=2$, $y=37$. Find $x=\log_2{37 \mod 131}$

- Solution
  
  Let $B=10$, $S=\{2,3,5,7\}$

  
  $2^1 = 2 \mod 131$
  $2^8 = 5^3 \mod 131$
  $2^{12} = 5 \times 7 \mod 131$
  $2^{14} = 3^2 \mod 131$
  $2^{34} = 3 \times 5^2 \mod 131$

  $1 = \log_2{2 \mod 130}$
  $8 = 3 \times \log_2{5 \mod 130}$
  $12 = \log_2{5} + \log_2{7 \mod 130}$
  $14 = 2 \times \log_2{3 \mod 130}$
  $34 = \log_2{3} + 2 \times \log_2{5 \mod 130}$

  $\log_2{2} = 1$
  $\log_2{5} = 46$
  $\log_2{7} = 96$
  $\log_2{3} = 72$

  $37 \times 2^{43} = 3 \times 5 \times 7 \mod 131$
  $\log_2{37} = -43 + \log_2{3} + \log_2{5} + \log_2{7 \mod 130} = 41$

  Solution: $2^{41} \mod 131 = 37$

- Complexity of best known algorithm for solving DLP:

  $$L_p \left[ \frac{1}{3}, 1.923 \right] = O \left( e^{\left( 1.923 + o(1) \right) \left( \ln p \right)^{\frac{1}{3}} \left( \ln \ln p \right)^{\frac{2}{3}}} \right)$$
ElGamal Encryption Scheme

- **Keys & parameters**
  - Domain parameter = \( \{p, g\} \)
  - Choose \( x \in [1, p-1] \) and compute \( y = g^x \mod p \)
  - Public key \((p, g, y)\)
  - Private key \(x\)

- **Encryption: \( m \to (C_1, C_2) \)**
  - Pick a random integer \( k \in [1, p-1] \)
  - Compute \( C_1 = g^k \mod p \)
  - Compute \( C_2 = m \times y^k \mod p \)

- **Decryption**
  - \( m = C_2 \times C_1^{-x} \mod p \)
  - \( C_2 \times C_1^{-x} = (m \times y^k) \times (g^k)^{-x} = m \times (g^x)^k \times (g^k)^{-x} = m \mod p \)
Key Generation
- Let p = 23, g = 7
- Private key x = 9
- Public key y = \( g^x \mod p = 7^9 \mod 23 = 15 \)

Encryption: m \( \rightarrow \) (\( C_1 \), \( C_2 \))
- Let m = 20
- Pick a random number k = 3
- Compute \( C_1 = g^k \mod p = 7^3 \mod 23 = 21 \)
- Compute \( C_2 = m \times y^k \mod p = 20 \times 15^3 \mod 23 = 20 \times 17 \mod 23 = 18 \)
- Send (\( C_1 \), \( C_2 \)) = (21, 18) as a ciphertext

Decryption
- \( m = C_2 / C_1^x \mod p = 18 / 21^9 \mod 23 = 18 / 17 \mod 23 = 20 \)
2. Digital Signatures

RSA, ElGamal, DSA, KCDSA, Schnorr
Digital Signature

- When do you use Digital Signature?
  - Electronic version of handwritten signature on electronic document
  - Signing using private key (only by the signer)
  - Verification using public key (by everyone)

- Hash then sign: $\text{sig}(h(m))$
  - Efficiency in computation and communication
Requirement of DS

- Security requirements for digital signature
  - Unforgeability (위조 방지)
  - User authentication (사용자 인증)
  - Non-repudiation (부인 방지)
  - Unalterability (변조 방지)
  - Non-reusability (재사용 방지)

- Services provided by digital signature
  - Authentication
  - Data integrity
  - Non-Repudiation
Signing & Verification

✓ Combine Hash with Digital Signature and use PKC
✓ Provide Authentication and Non-Repudiation
✓ (Ex.) RSA, ElGamal DSA, KCDSA, ECDSA, EC-KCDSA
Security of Digital Signature

- **Forgery**
  - **Total break**: adversary is able to find the secret for signing, so he can forge then any signature on any message.
  - **Selective forgery**: adversary is able to create valid signatures on a message chosen by someone else, with a significant probability.
  - **Existential forgery**: adversary can create a pair (message, signature), s.t. the signature of the message is valid.

- **Attacking**
  - **Key-only attack**: Adversary knows only the verification function (which is supposed to be public).
  - **Known message attack**: Adversary knows a list of messages previously signed by Alice.
  - **Chosen message attack**: Adversary can choose what messages wants Alice to sign, and he knows both the messages and the corresponding signatures.
RSA-Signing

- **Key generation**
  - Choose two large (512 bits or more) primes p & q
  - Compute modulus $n = pq$, and $\phi(n) = (p-1)(q-1)$
  - Pick an integer e relatively prime to $\phi(n)$, $\text{gcd}(e, \phi(n)) = 1$
  - Compute d such that $ed = 1 \mod \phi(n)$
  - Public key $(n, e)$: publish
  - Private key d: keep secret (may keep p and q securely.)

- **Signing / Verifying**
  - S: $s = m^d \mod n$ for $0 < m < n$
  - V: $m =? s^e \mod n$
  - S: $s = h(m)^d \mod n$ --- hashed version
  - V: $h(m) =? s^e \mod n$

- **RSA signature without padding**
  - Deterministic signature, no randomness introduced
Forging RSA-signature

- RSA signature forgery: Attack based on the multiplicative property of RSA.
  \[ y_1 = (m_1)^d \quad y_2 = (m_2)^d, \]
  then \( (y_1 y_2)^e = m_1 m_2 \)
  Thus, \( y_1 y_2 \) is a valid signature of \( m_1 m_2 \)

- This is an existential forgery using a known message attack.

- (HW) RSA-PSS required like RSA-OAEP
ElGamal Signature

- **Keys & parameters**
  - Domain parameter = \{p, g\}
  - Choose \( x \in [1, p-1] \) and compute \( y = g^x \mod p \)
  - Public key (p, g, y)
  - Private key x

- **Signature generation**: (r, s)
  - Pick a random integer \( k \in [1, p-1] \)
  - Compute \( r = g^k \mod p \)
  - Compute \( s \) such that \( m = xr + ks \mod p-1 \)

- **Signature verification**
  - \( y^r r^s \mod p =? g^m \mod p \)
    - If equal, accept the signature (valid)
    - If not equal, reject the signature (invalid)
Digital Signature Algorithm (DSA)

Private: \( x \)
Public: \( p, q, g, y \)

- **Signing**
  
  Pick a random \( k \) s.t. \( 0 < k < q \)
  
  \[
  r = (g^k \mod p) \mod q \\
  s = k^{-1}(\text{SHA1}(m) + xr) \mod q
  \]

- **Verifying**
  
  \[ w = s^{-1} \mod q \]
  
  \[
  u1 = \text{SHA1}(m) \times w \mod q \\
  u2 = r \times w \mod q \\
  v = (g^{u1} \times y^{u2} \mod p) \mod q \\
  v =? r
  \]
KCDSA

| Private: | $x$ |
| Public:  | $p, q, g, y$ |

$z = h(Cert\_Data)$

$p : 768 + 256k$ (k=0 ~ 5) bit prime
$q : 160 + 32k$ (k=0~3) bit prime, $q \mid p-1$
$g : \text{generator of order } q$
$x : 0 < x < q$
$y = g^{x'} \mod p, x' = x^{-1} \mod q$

**Signing**

Pick a random $k$ s.t. $0 < k < q$

\[
\begin{align*}
  r &= \text{HAS160}(g^k \mod p) \\
  e &= r \oplus \text{HAS160}(z \ || \ m) \\
  s &= x(k - e) \mod q
\end{align*}
\]

**Verifying**

\[
\begin{align*}
  e &= r \oplus \text{HAS160}(z \ || \ m) \\
  v &= y^s \cdot g^e \mod p \\
  \text{HAS160}(v) &= r
\end{align*}
\]
Schnorr Signature Scheme

- **Domain parameters**
  - \( p \) = a large prime (~size 1024 bit), \( q \) = a prime (~size 160 bit)
  - \( q \) = a large prime divisor of \( p-1 \) (\( q \mid p-1 \))
  - \( g \) = an element of \( \mathbb{Z}_p \) of order \( q \), i.e., \( g \neq 1 \) & \( g^q = 1 \mod p \)
  - Considered in a subgroup of order \( q \) in modulo \( p \)

- **Keys**
  - Private key \( x \in_R [1, q-1] \): a random integer
  - Public key \( y = g^x \mod p \)

- **Signature generation**: \((r, s)\)
  - Pick a random integer \( k \in_R [1, q-1] \)
  - Compute \( r = h(g^k \mod p, m) \)
  - Compute \( s = k - xr \mod q \)

- **Signature verification**
  - \( r =? h(y^r g^s \mod p, m) \)
Advanced Digital Signature

• Blind signature
• One-time signature
  – Lamport scheme or Bos-Chaum scheme
• Undeniable signature
  – Chaum-van Antwerpen scheme
• Fail-stop signature
  – van Heyst-Peterson scheme
• Proxy signature
• Group (Ring) signature: group member can generate signature if dispute occurs, identify member. etc.
Blind Signature(I)

Without B seeing the content of message M, A can get a signature of M from B.

RSA scheme, B’s public key : {n, e}, private key : {d}

(1) random number
(2) blinding
(3) signing
(4) unblinding

A (customer)          B (Bank)

(1) select random k
s.t. \( \gcd(n, k) = 1 \),
\( 1 < k < n - 1 \)

(2) \( m^* = m^e k \mod n \)

(3) \( s^* = (m^*)^d \mod n \)

(4) \( s = k^{-1} s^* \mod n \)

(signature of M by B : \( k^{-1} (m^e)^d = k^{-1} m^e k^d = m^d \))

\( f(m) : \text{blinded message} \)
Blind Signature(II)

(Preparation) \( p=11, q=3, n=33, \phi(n)= 10 \times 2 = 20 \)
\[
gcd(d, \phi(n))=1 \implies d=3, \quad ed = 1 \mod \phi(n) \implies 3d = 1 \mod 20 \implies e=7
\]
B: public key :\{n,e\}=\{33,7\}, private key =\{d\}=\{3\}

(1) A’s blinding of \( m=5 \)
   - select \( k \) s.t. \( \gcd(k,n)=1. \ \gcd(k,33)=1 \implies k=2 \)
   - \( m^* = m^k \mod n = 5^2 \mod 33 = 640 \mod 33 = 13 \mod 33 \)

(2) B’s signing without knowing the original \( m \)
   - \( s^* = (m^*)^d \mod n = 13^3 \mod 33 = 2197 \mod 33 = 19 \mod 33 \)

(3) A’s unblinding
   - \( s=k^{-1} s^* \mod n \) (\( 2k^{-1}=1 \mod 33 \implies k=17 \))
     - \( 17 \times 19 \mod 33 = 323 = 26 \mod 33 \)

* Original Signature : \( m^d \mod n = 5^3 \mod 33 = 125 = 26 \mod 33 \)
3. Key Exchange

Diffie-Hellman
DH Key Agreement

Domain Parameters:
\( p, g \)

Choose
\( X_a \in [1, p-1] \)
\( Y_a = g^{X_a} \mod p \)

Choose
\( X_b \in [1, p-1] \)
\( Y_b = g^{X_b} \mod p \)

Compute the shared key:
\( K_a = Y_b^{X_a} = g^{X_b X_a} \mod p \)

Compute the shared key:
\( K_b = Y_a^{X_b} = g^{X_a X_b} \mod p \)
Diffie-Hellman Problem

- **Computational Diffie-Hellman (CDH) Problem**
  
  Given \( Y_a = g^{X_a} \mod p \) and \( Y_b = g^{X_b} \mod p \),
  
  compute \( K_{ab} = g^{X_a X_b} \mod p \)

- **Decision Diffie-Hellman (DDH) Problem**
  
  Given \( Y_a = g^{X_a} \mod p \) and \( Y_b = g^{X_b} \mod p \),
  
  distinguish between \( K_{ab} = g^{X_a X_b} \mod p \) and a random string

- **Discrete Logarithm Problem (DLP)**
  
  Given \( Y = g^X \mod p \), compute \( X = \log_b Y \)

The Security of the Diffie-Hellman key agreement depends on the difficulty of CDH problem.
MIMT in DH Scheme

$X_b : \text{private}$

$Y_b = g^{X_b} : \text{public}$

$Y_b \\ Y_c \\ Y_c = g^{X_c}$ for some $X_c$

Bob computes the session key

$K_b = Y_c^{X_b} = g^{X_c X_b}$

$Y_c \\ Y_a \\ Y_a = g^{X_a}$

Alice computes the session key

$K_a = Y_c^{X_a} = g^{X_c X_a}$

Adversary computes both session keys

$K_b = Y_b^{X_c} = g^{X_c X_b}$

$K_a = Y_a^{X_c} = g^{X_c X_a}$

Man-in-the-middle attack comes from no authentication
DH Key Agreement with Certified Key

Domain Parameters
\[ p, g \]

- choose \( X_a \in [1, p-1] \)
  \[ Y_a = g^{X_a} \mod p \]

- choose \( X_b \in [1, p-1] \)
  \[ Y_b = g^{X_b} \mod p \]

Certified key
\( Y_a \) and \( Y_b \)

- compute the shared key
  \[ K_a = Y_b^{X_a} = g^{X_bX_a} \mod p \]

- compute the shared key
  \[ K_b = Y_a^{X_b} = g^{X_aX_b} \mod p \]

- Interaction is not required
- Agreed key is fixed, long-term use
Elliptic Curve (1/2)

- **Weierstrass form of Elliptic Curve**
  \[ y^2 + a_1 xy + a_3 = x^3 + a_2 x^2 + a_4 x + a_6 \]

- **Example (over rational field)**
  \[ y^2 = x^3 - 4x + 1 \]
  \[ E(Q) = \{(x,y) \in \mathbb{Q}^2 \mid y^2 = x^3 - 2x + 2\} \cup O_E \]
  \[ P = (2, 1), \quad -P = (2, -1) \]
  \[ [2]P = (12, -41) \]
  \[ [3]P = (91/25, 736/125) \]
  \[ [4]P = (5452/1681, -324319/68921) \]
Elliptic Curve (2/2)

- Example (over finite field GF(p) : p = 13)
  - P = (2,1), −P = (2, 12), [2]P = (12, 11)
  - Hasse’s Theorem : \( p - 2\sqrt{p} \leq \# of E(p) \leq p + 2\sqrt{p} \)
  - Scalar multiplication: [d]P

- Elliptic Curve Discrete Logarithm
  - Base of Elliptic Curve Cryptosystem (ECC)

\[ y = g^x \mod p \quad \leftrightarrow \quad Q = [d]P \]

Find x for given g, p, Y

Find d for given P, Q
ECC

- **Advantages**
  - Breaking PKC over Elliptic Curve is much harder.
  - We can use much shorter key about 1/6.
  - Encryption/Decryption is much faster than other PKCs.
  - Suitable for restricted environments like mobile phone, smart card.

- **Disadvantages**
  - It’s new technique ➜ There may be new attacks.
  - Too complicated to understand.
  - ECC is a minefield of patents.
    : e.g., US patents  
    4587627/739220 – Normal Basis, 5272755 – Curve over GF(p)  
    5463690/5271051/5159632 – p=2^q-c for small c, etc…
Implementation

- **RSA Encryption/Decryption**

<table>
<thead>
<tr>
<th></th>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS#1-v1.5</td>
<td>1.49 ms</td>
<td>18.05 ms</td>
</tr>
<tr>
<td>PKCS#1-OAEP</td>
<td>1.41 ms</td>
<td>18.09 ms</td>
</tr>
</tbody>
</table>

- **Signature**

<table>
<thead>
<tr>
<th></th>
<th>Signing</th>
<th>Verifying</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS#1-v1.5</td>
<td>18.07 ms</td>
<td>1.24 ms</td>
</tr>
<tr>
<td>PKCS#1-PSS</td>
<td>18.24 ms</td>
<td>1.28 ms</td>
</tr>
<tr>
<td>DSA with SHA1</td>
<td>2.75 ms</td>
<td>9.85 ms</td>
</tr>
<tr>
<td>KCDSA with HAS160</td>
<td>2.42 ms</td>
<td>9.55 ms</td>
</tr>
</tbody>
</table>

- **Modular Exponentiation vs. Scalar Multiplication of EC**

<table>
<thead>
<tr>
<th></th>
<th>M.E. (1024-bit)</th>
<th>S.M. (GF(2^{162}))</th>
<th>S.M. (GF(p))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>52.01 ms</td>
<td>2.24 ms</td>
<td>1.17 ms</td>
</tr>
</tbody>
</table>
## Equivalent Key Size

<table>
<thead>
<tr>
<th>Bits of security</th>
<th>Symmetric key algorithms</th>
<th>FFC (e.g., DSA, D-H)</th>
<th>IFC (e.g., RSA)</th>
<th>ECC (e.g., ECDSA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>2TDEA(^1)</td>
<td>(L = 1024)</td>
<td>(k = 1024)</td>
<td>(f = 160-223)</td>
</tr>
<tr>
<td>112</td>
<td>3TDEA</td>
<td>(L = 2048)</td>
<td>(k = 2048)</td>
<td>(f = 224-255)</td>
</tr>
<tr>
<td>128</td>
<td>AES-128</td>
<td>(L = 3072)</td>
<td>(k = 3072)</td>
<td>(f = 256-383)</td>
</tr>
<tr>
<td>192</td>
<td>AES-192</td>
<td>(L = 7680)</td>
<td>(k = 7680)</td>
<td>(f = 384-511)</td>
</tr>
<tr>
<td>256</td>
<td>AES-256</td>
<td>(L = 15360)</td>
<td>(k = 15360)</td>
<td>(f = 512+)</td>
</tr>
</tbody>
</table>

Recommendation for the Transition of Cryptographic Algorithm and Key Sizes, NIST800-121, Jan. 2010.
# Key Length by NIST

<table>
<thead>
<tr>
<th>Date</th>
<th>Minimum of Strength</th>
<th>Symmetric Algorithms</th>
<th>Asymmetric</th>
<th>Discrete Logarithm Group</th>
<th>Elliptique Curve</th>
<th>Hash (A)</th>
<th>Hash (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007 - 2010</td>
<td>80</td>
<td>2TDEA*</td>
<td>1024</td>
<td>160</td>
<td>1024</td>
<td>160</td>
<td>SHA-1**</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-224</td>
<td>SHA-224</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-256</td>
<td>SHA-256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-384</td>
<td>SHA-384</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-512</td>
<td>SHA-512</td>
</tr>
<tr>
<td>2011 - 2030</td>
<td>112</td>
<td>3TDEA</td>
<td>2048</td>
<td>224</td>
<td>2048</td>
<td>224</td>
<td>SHA-224</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-256</td>
<td>SHA-256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-384</td>
<td>SHA-384</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-512</td>
<td>SHA-512</td>
</tr>
<tr>
<td>&gt; 2030</td>
<td>128</td>
<td>AES-128</td>
<td>3072</td>
<td>256</td>
<td>3072</td>
<td>256</td>
<td>SHA-256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-384</td>
<td>SHA-256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-512</td>
<td>SHA-384</td>
</tr>
<tr>
<td>&gt;&gt; 2030</td>
<td>192</td>
<td>AES-192</td>
<td>7680</td>
<td>384</td>
<td>7680</td>
<td>384</td>
<td>SHA-384</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-256</td>
<td>SHA-256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-384</td>
<td>SHA-384</td>
</tr>
<tr>
<td>&gt;&gt;&gt; 2030</td>
<td>256</td>
<td>AES-256</td>
<td>15360</td>
<td>512</td>
<td>15360</td>
<td>512</td>
<td>SHA-384</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHA-256</td>
<td>SHA-512</td>
</tr>
</tbody>
</table>