Introduction to Cryptography
ECE 597XX/697XX

Part 6

Introduction to
Public-Key Cryptography

Israel Koren

Content of this part

♦ Symmetric Cryptography Revisited
♦ Principles of Asymmetric Cryptography
♦ Practical Aspects of Public-Key Cryptography
♦ Important Public-Key Algorithms
♦ Essential Number Theory for Public-Key Algorithms
Two properties of symmetric (secret-key) crypto-systems:

- The same secret key $K$ is used for encryption and decryption.
- Encryption and decryption are very similar (or even identical) functions.

Symmetric Cryptography: Analogy

Safe with a strong lock, only Alice and Bob have a copy of the key:

- Alice encrypts → locks message in the safe with her key
- Bob decrypts → uses his copy of the key to open the safe
**Symmetric Cryptography: Shortcomings**

- Symmetric algorithms, e.g., AES or 3DES, are very secure, fast & widespread but:
- Key distribution problem: The secret key must be transported securely
- Number of keys: In a network, each pair of users requires an individual key
  \[ \text{n users in the network require } \frac{n \cdot (n-1)}{2} \text{ keys, each user stores } (n-1) \text{ keys} \]

  **Example:**
  - 6 users (nodes)
  - \[ \frac{6 \cdot 5}{2} = 15 \] keys (edges)

- Alice or Bob can cheat each other, because they have identical keys.
  **Example:** Alice can claim that she never ordered a TV online from Bob (he could have fabricated her order). To prevent this: „non-repudiation“

**Idea behind Asymmetric Cryptography**

**New Idea:**
Use the „good old mailbox“ principle:
- **Everyone** can drop a letter
- **But: Only the owner** has the correct key to open the box

1976: first publication of such an algorithm by Diffie and Hellman, and also by Merkle.
Asymmetric (Public-Key) Cryptography

Principle: “Split up” the key

\[
\begin{array}{c}
K \\
\text{Public Key } (K_{\text{pub}}) \\
(\text{Encrypt}) \\
\text{Secret Key } (K_{\text{pr}}) \\
(\text{Decrypt})
\end{array}
\]

During the key generation, a key pair \( K_{\text{pub}} \) and \( K_{\text{pr}} \) is computed.

Asymmetric Cryptography: Analogy

Safe with public lock and private lock:

- Alice deposits (encrypts) a message with the - not secret - public key \( K_{\text{pub}} \)
- Only Bob has the - secret - private key \( K_{\text{pr}} \) to retrieve (decrypt) the message
# Basic Protocol for Public-Key Encryption

Alice

Bob

\[ K_{\text{pubB}} \]

\( (K_{\text{pubB}}, K_{\text{prB}}) = K \)

\( x \)

\[ y = e_{K_{\text{pubB}}}(x) \]

\( y \)

\( x = d_{K_{\text{prB}}}(y) \)

→ Key Distribution Problem solved *

*) at least for now; public keys need to be authenticated, cf. Chap. 13 of Understanding Cryptography

---

# Security Mechanisms of Public-Key Cryptography

Here are main mechanisms that can be realized with asymmetric cryptography:

- **Key Distribution** (e.g., Diffie-Hellman key exchange, RSA) without a pre-shared secret (key)
- **Nonrepudiation and Digital Signatures** (e.g., RSA, DSA or ECDSA) to provide message integrity
- **Identification** using challenge-response protocols with digital signatures
- **Encryption** (e.g., RSA / ElGamal)
- **Disadvantage:** Computationally very intensive (1000 times slower than symmetric Algorithms)
Basic Key Transport Protocol 1/2

In practice: Hybrid systems, incorporating asymmetric and symmetric algorithms

1. **Key exchange** (for symmetric schemes) and digital signatures are performed with (slow) asymmetric algorithms

2. **Encryption** of data is done using (fast) symmetric ciphers, e.g., block ciphers or stream ciphers

Example: Hybrid protocol with AES as the symmetric cipher

Choose random symmetric key $K$

$y_1 = e_{K_{pubB}}(K)$

$y_2 = AES_K(x)$

$K = d_{K_{pubB}}(y_1)$

$x = AES^{-1}_K(y_2)$

Key Exchange (asymmetric)

Data Encryption (symmetric)
How to build Public-Key Algorithms

Asymmetric schemes are based on a „one-way function“ $f()$:

- Computing $y = f(x)$ is computationally easy
- Computing $x = f^{-1}(y)$ is computationally very hard

One way functions are based on mathematically hard problems. Three main families:

- **Factoring integers** (RSA, ...):
  Given a composite integer $n$, find its prime factors (Multiply two primes: easy)
- **Discrete Logarithm** (Diffie-Hellman, ElGamal, DSA, ...):
  Given $a$, $y$ and $m$, find $x$ such that $a^x = y \mod m$ (Exponentiation $a^x$: easy)
- **Elliptic Curves (EC)** (ECDH, ECDSA): Generalization of discrete logarithm

Note: The problems are considered mathematically hard, but no proof exists (so far).

Key Lengths and Security Levels

<table>
<thead>
<tr>
<th>Symmetric</th>
<th>ECC</th>
<th>RSA, DL</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 Bit</td>
<td>128 Bit</td>
<td>700 Bit</td>
<td>Only short term security (a few hours or days)</td>
</tr>
<tr>
<td>80 Bit</td>
<td>160 Bit</td>
<td>1024 Bit</td>
<td>Medium security (except attacks from big governmental institutions)</td>
</tr>
<tr>
<td>128 Bit</td>
<td>256 Bit</td>
<td>3072 Bit</td>
<td>Long term security (without quantum computers)</td>
</tr>
</tbody>
</table>

- The exact complexity of RSA (factoring) and DL (Index-Calculus) is difficult to estimate
- The development of quantum computers would probably be the end for ECC, RSA & DL
Euclidean Algorithm 1/2

♦ Compute the greatest common divisor \( \gcd(r_0, r_1) \) of two integers \( r_0 \) and \( r_1 \)

♦ \( \gcd \) is easy for small numbers:
  1. factor \( r_0 \) and \( r_1 \)
  2. \( \gcd = \) highest common factor

♦ Example:
  \( r_0 = 84 = 2 \cdot 3 \cdot 7 \)
  \( r_1 = 30 = 2 \cdot 3 \cdot 5 \)
  \( \Rightarrow \) The \( \gcd \) is the product of all common prime factors:
  \( 2 \cdot 3 = 6 = \gcd(30, 84) \)

♦ But: Factoring is very complicated for large numbers

Euclidean Algorithm 2/2

♦ Observation: \( \gcd(r_0, r_1) = \gcd(r_0 - r_1, r_1) \)

⇒ Core idea:
  • Reduce the problem of finding the \( \gcd \) of two given numbers to that of the \( \gcd \) of two smaller numbers
  • Repeat process recursively
  • The final \( \gcd(r_i, 0) = r_i \) is the answer to the original problem

Example: \( \gcd(r_0, r_1) \) for \( r_0 = 27 \) and \( r_1 = 21 \)

\[
\begin{array}{c|c|c|c|c}
21 & 6 & & \\
6 & 6 & 6 & 3 \\
3 & 3 & & \\
\end{array}
\]

\( \gcd(27, 21) = \gcd(1 \cdot 21 + 6, 21) = \gcd(21, 6) \)

\( \gcd(21, 6) = \gcd(3 \cdot 6 + 3, 6) = \gcd(6, 3) \)

\( \gcd(6, 3) = \gcd(2 \cdot 3 + 0, 3) = \gcd(3, 0) = 3 \)

Very efficient method even for long numbers: complexity grows linearly with number of bits
Extended Euclidean Algorithm (EEA)

- Extend the Euclidean algorithm to find modular inverse of \( r_1 \mod r_0 \)
- EEA computes \( s, t \), and the gcd:
  \[ \gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1 \]
- \( \gcd(r_0, r_1) = 1 \) in order for the inverse to exist
- Reduce the equation \( \mod r_0 \) :
  \[ s \cdot 0 + t \cdot r_1 \equiv 1 \mod r_0 \]
  \[ r_1 \cdot t \equiv 1 \mod r_0 \]

- \( t \) is the inverse of \( r_1 \mod r_0 \)
- EEA uses recursive formulae to calculate \( s \) and \( t \) in each step
  - Express current remainder \( r_i \) as:
  
    \[ r_i = s_i r_0 + t_i r_1 \]
  - Last iteration:

\[ r_i = \gcd(r_0, r_1) = s_i r_0 + t_i r_1 = s r_0 + t r_1 \]

Extended Euclidean Algorithm - Example

\[ \gcd(973, 301) = 7 \]
\[ r_0 = 973; \; r_1 = 301 \]

\[ s = 13; \; t = -42 \]

\[ \gcd(973, 301) = 7 = [13]973 + [-42]301 = 12649 - 12642 \]

EEA can be expressed using recursive formulae for \( s_i, t_i \)

\[
\begin{align*}
s_0 &= 1 & t_0 &= 0 \\
s_1 &= 0 & t_1 &= 1
\end{align*}
\]

\[
\begin{align*}
1.1 & \; i = i + 1 \\
1.2 & \; r_i = r_{i-2} \mod r_{i-1} \\
1.3 & \; q_{i-1} = (r_{i-2} - r_i) / r_{i-1} \\
1.4 & \; s_i = s_{i-2} - q_{i-1} \cdot s_{i-1} \\
1.5 & \; t_i = t_{i-2} - q_{i-1} \cdot t_{i-1}
\end{align*}
\]

\[
\begin{align*}
\text{DO} & \; \text{WHILE } r_i \neq 0
\end{align*}
\]

Adapted from Paar & Pelzl, “Understanding Cryptography,” and other sources
EEA – calculating a modular inverse

Example: Calculate the modular inverse of 12 mod 67:
♦ Using EEA we obtain $-5 \cdot 67 + 28 \cdot 12 = 1$
♦ Hence 28 is the inverse of 12 mod 67.

\begin{align*}
gcd(67, 12) = \gcd(12, 7) &= \gcd(5, 2) = \gcd(2, 1) \\
67 &= 12 \cdot 5 + 7 = (1)67 + (-5)12 \\
12 &= 7 + 5 \Rightarrow 5 = 12 - 7 = (-1)67 + (6)12 \\
7 &= 5 + 2 \Rightarrow 2 = 7 - 5 = (2)67 + (-11)12 \\
5 &= 2 \cdot 2 + 1 \Rightarrow 1 = 5 - 2 \cdot 2 = (-5)67 + (28)12
\end{align*}

\begin{align*}
\begin{array}{c|c|c|c}
 i & q_{i-1} & r_i & s_i & t_i \\
2 & 5 & 7 & 1 & -5 \\
3 & 1 & 5 & -1 & 6 \\
4 & 1 & 2 & 2 & -11 \\
5 & 2 & 1 & -5 & 28 \\
\end{array}
\end{align*}

♦ Check: $28 \cdot 12 = 336 \equiv 1 \pmod{67}$

Euler's Phi Function 1/2

♦ Important for public-key systems, e.g., RSA:
  Given the set of the $m$ integers \{0, 1, 2, ..., $m-1$\},
  How many numbers in the set are relatively prime to $m$?
♦ Answer: Euler's Phi function $\Phi(m) \ (\text{totonet function})$
♦ Example: sets \{0, 1, ..., 5\} ($m=6$), and \{0, 1, ..., 4\} ($m=5$)

\begin{align*}
gcd(0, 6) &= 6 \\
gcd(1, 6) &= 1 \quad \Rightarrow \quad \gcd(0, 5) = 5 \\
gcd(2, 6) &= 2 \quad \Rightarrow \quad \gcd(1, 5) = 1 \\
gcd(3, 6) &= 3 \quad \Rightarrow \quad \gcd(2, 5) = 1 \\
gcd(4, 6) &= 2 \quad \Rightarrow \quad \gcd(3, 5) = 1 \\
gcd(5, 6) &= 1 \quad \Rightarrow \quad \gcd(4, 5) = 1
\end{align*}

⇒ $\Phi(5) = 4; \ \Phi(6) = 2$

♦ Testing one gcd per number in the set is extremely slow for large $m$. 
Euler's Phi Function 2/2

- If canonical factorization of $m$ known: $m = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$ (where $p_i$ primes and $e_i$ positive integers)
- then calculate Phi according to the relation
  $$\Phi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_i-1})$$
- Phi especially easy for $e_i = 1$, e.g., $m = p \cdot q \Rightarrow \Phi(m) = (p-1) \cdot (q-1)$
- Examples: $m = 6 = 3 \cdot 2 \Rightarrow \Phi(6) = (3-1)(2-1) = 2$
  $m = 899 = 29 \cdot 31$: $\Phi(899) = (29-1) \cdot (31-1) = 28 \cdot 30 = 840$
- Note: Finding $\Phi(m)$ is computationally easy if factorization of $m$ is known
  (otherwise the calculation of $\Phi(m)$ is computationally very hard for large numbers)

Fermat's Little Theorem

- Given a prime $p$ and an integer $a$: $a^p \equiv a \pmod{p}$
- Can be rewritten as $a^{p-1} \equiv 1 \pmod{p}$
  $a \cdot a^{p-2} \equiv 1 \pmod{p}$
- Use: Find modular inverse, if $p$ is prime.
- Comparing with definition of the modular inverse
  $a \cdot a^{-1} \equiv 1 \pmod{m}$
  the modular inverse modulo a prime $p$ is
  $a^{-1} \equiv a^{p-2} \pmod{p}$

Example: $a = 2, p = 7$ $a^{p-2} = 2^5 = 32 \equiv 4 \pmod{7}$
  verify: $2 \cdot 4 \equiv 1 \pmod{7}$
- Fermat's Little Theorem works only modulo a prime $p$
Euler’s Theorem

♦ Generalization of Fermat’s little theorem to any integer modulus
♦ Given two relatively prime integers $a$ and $m$:
♦ Example: $m=18$, $a=5$

1. Calculate Euler’s Phi Function
   \[ \Phi(18) = \Phi(3^2 \cdot 2) = (3^2 - 3^1)(2^1 - 2^0) = 6 \]

2. Verify Euler’s Theorem
   \[ 5^{\Phi(18)} = 5^6 = 25^3 = 7^3 \mod 18 \Rightarrow 7^3 = 343 = 18 \cdot 19 + 1 \mod 18 \]

♦ Fermat’s little theorem = special case of Euler’s Theorem
♦ for a prime $p$:
   \[ \Phi(p) = (p^1 - p^0) = p - 1 \]
   \[ a^{\Phi(p)} = a^{p-1} \equiv 1 \pmod{p} \]

Lessons Learned

♦ Public-key algorithms have capabilities that symmetric ciphers don’t have, in particular digital signature and key establishment functions.
♦ Public-key algorithms are computationally intensive (a nice way of saying that they are slow), and hence are poorly suited for bulk data encryption.
♦ Only three families of public-key schemes are widely used. This is considerably fewer than in the case of symmetric algorithms.
♦ The extended Euclidean algorithm allows us to compute modular inverses quickly, which is important for almost all public-key schemes.
♦ Euler’s phi function gives us the number of elements smaller than an integer $n$ that are relatively prime to $n$. This is important for the RSA.