Cryptography and Data Security

Chapter 9

PUBLIC-KEY CRYPTOGRAPHY AND RSA

RAPEEPORN PUNYAYUTTHAKARN
CISSP, CISA, OSSA
Principles of Public-Key Cryptosystems
Main Problems

- The main problems in cryptography today
  - Privacy
  - Integrity

- Privacy
  - The right of an individual to be secure from unauthorized disclosure of information about oneself that is contained in documents

- Integrity
  - The ability to ensure that information is not modified except by people who are explicitly intended to modify it
Solutions by Cryptography

- Cryptography is the way to protect aspect of message
- Solutions
  - Privacy: Encryption
  - Integrity: Authentication
- Encryption
  - The process of disguising a message or data in such a way as to hide its substance.
Solutions by Cryptography

- **Authentication**
  - Verification of the identity of the entities
    - Impersonate
  - Verification that the original contents of information have not been altered or corrupted
    - Substitute
Encryption

Alice → Message → Bob

Signifies
Space: Communication line
Time: Storage

Eve

Network Sniffer
Trapping the line
Encryption

Condition:

\[ C = E(m_1, K') \]
\[ m_2 = D(C, K) = m_1 \]

- **Easy**
  - K' must be kept secret
  - "Symmetric Cryptography"

- **Hard**
  - K' can be made public
  - "Asymmetric Cryptography"
Authentication

Alice  Message  Bob

Signifies
Space: Communication line
Time: Storage

Eve
Active Eavesdropper
Authentication

Alice

Encryption

K

(MESSAGE, Authenticator)

Bob

Decryption

K'

Yes

Message

No

Eve

Message

K' must be kept secret

“Symmetric Cryptography”

K' can be made public

“Asymmetric Cryptography”

“Public Key Infrastructure”

DIGITAL SIGNATURE

K' must be kept secret

“Symmetric Cryptography”

K' can be made public

“Asymmetric Cryptography”

“Public Key Infrastructure”

DIGITAL SIGNATURE
Secret-Key Cryptography

- traditional **secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed communications are compromised
- also is **symmetric**, parties are equal
- hence does not protect sender from receiver forging a message & claiming is sent by sender
Public-Key Cryptography

- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- uses clever application of number theoretic concepts to function
- complements **rather than** replaces symmetric key crypto
Why Public-Key Cryptography?

- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in classified community
Public-Key Cryptography

(a) Encryption

Plaintext input → Encryption algorithm (e.g., RSA) → Transmitted ciphertext → Decryption algorithm (reverse of encryption algorithm) → Plaintext output
Public-Key Cryptography

- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - A **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - A **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**

- **is asymmetric** because
  - Those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures
Public-Key Characteristics

- Public-Key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm & encryption key
  - it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)
Public-Key Cryptosystems: Privacy
Public-Key Cryptosystems: Authentication
Public-Key Applications

- can classify uses into 3 categories:
  - **encryption/decryption** (provide secrecy)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)

- some algorithms are suitable for all uses, others are specific to one
### Application for Public-Key Cryptosystems

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Encryption/Decryption</th>
<th>Digital Signature</th>
<th>Key Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Elliptic Curve</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Diffie-Hellman</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DSS</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Security of Public-Key Schemes

- like private key schemes brute force exhaustive search attack is always theoretically possible
  - but keys used are too large (>512bits)
- security relies on
  - easy (en/decrypt) and
  - hard (cryptanalysis) problems
- more generally the hard problem is known, but is made hard enough to be impractical to break
- it is slow compared to secret key schemes
The RSA Algorithm
RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (e.g. 1024 bits)
- security due to cost of factoring large numbers
  - factorization takes $O(e^{\log n \log \log n})$ operations (hard)
RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random: \( p, q \)
- computing their system modulus: \( n = p \cdot q \)
  - note \( \varphi(n) = (p-1)(q-1) \)
- selecting at random the encryption key \( e \)
  - where \( 1 < e < \varphi(n), \gcd(e, \varphi(n)) = 1 \)
- solve following equation to find decryption key \( d \)
  - \( e \cdot d = 1 \mod \varphi(n) \) and \( 0 \leq d \leq n \)
- publish their public encryption key: \( PU = \{e, n\} \)
- keep secret private decryption key: \( PR = \{d, n\} \)
RSA Use

- to encrypt a message $M$ the sender:
  - obtains **public key** of recipient $PU=\{e, n\}$
  - computes: $C = M^e \mod n$, where $0 \leq M < n$

- to decrypt the ciphertext $C$ the owner:
  - uses their **private key** $PR=\{d, n\}$
  - computes: $M = C^d \mod n$

- note that the message $M$ must be smaller than the modulus $n$ (block if needed)
Why RSA Works

- because of Euler's Theorem:
  - \(a^{\phi(n)} \mod n = 1\) where \(\gcd(a, n) = 1\)

- in RSA have:
  - \(n = p \cdot q\)
  - \(\phi(n) = (p-1)(q-1)\)
  - carefully chose \(e\) & \(d\) to be inverses \(\mod \phi(n)\)
  - hence \(e \cdot d = 1 + k \cdot \phi(n)\) for some \(k\)

- hence:
  \[
  C^d = M^{e \cdot d} = M^{1+k \cdot \phi(n)} = M^1 \cdot (M^{\phi(n)})^k \\
  = M^1 \cdot (1)^k = M^1 = M \mod n
  \]
### The RSA Algorithm

#### Key Generation
- Select $p$, $q$ \( p \text{ and } q \text{ both prime, } p \neq q \)
- Calculate $n = p \times q$
- Calculate $\phi(n) = (p - 1)(q - 1)$
- Select integer $e$ \( \gcd(\phi(n), e) = 1; 1 < e < \phi(n) \)
- Calculate $d$ \( d \equiv e^{-1} \pmod{\phi(n)} \)
- Public key \( PU = \{e, n\} \)
- Private key \( PR = \{d, n\} \)

#### Encryption
- Plaintext: \( M < n \)
- Ciphertext: \( C = M^e \pmod{n} \)

#### Decryption
- Ciphertext: \( C \)
- Plaintext: \( M = C^d \pmod{n} \)
RSA Example - Key Setup

1. Select primes: \( p = 17 \) & \( q = 11 \)
2. Compute \( n = pq = 17 \times 11 = 187 \)
3. Compute \( \varphi(n) = (p-1)(q-1) = 16 \times 10 = 160 \)
4. Select \( e \): \( \gcd(e, 160) = 1 \); choose \( e = 7 \)
5. Determine \( d \): \( de = 1 \mod 160 \) and \( d < 160 \)
   Value is \( d = 23 \) since \( 23 \times 7 = 161 = 10 \times 160 + 1 \)
6. Publish public key \( \text{PU} = \{7, 187\} \)
7. Keep secret private key \( \text{PR} = \{23, 187\} \)
sample RSA encryption/decryption is:
given message $M = 88$ (nb. $88 < 187$)

encryption:
$C = 88^7 \mod 187 = 11$

decryption:
$M = 11^{23} \mod 187 = 88$
Exponentiation

- can use the Square and Multiply Algorithm
- a fast, efficient algorithm for exponentiation
- concept is based on repeatedly squaring base
- and multiplying in the ones that are needed to compute the result
- look at binary representation of exponent
- only takes $O(\log_2 n)$ multiples for number $n$
  - eg. $7^5 = 7^4 \cdot 7^1 = 3 \cdot 7 = 10 \mod 11$
  - eg. $3^{129} = 3^{128} \cdot 3^1 = 5 \cdot 3 = 4 \mod 11$
Efficient Encryption

- Encryption uses exponentiation to power e.
- Hence if e small number of 1 to minimize the multiplication operation.
  - Often choose e = 65537 \((2^{16} - 1)\) \(\Rightarrow\) 1000000000000001
  - Also see choices of e = 3 \(\Rightarrow\) 11
  - Or e = 17 \(\Rightarrow\) 10001
- But if e too small (eg e = 3) can attack.
  - Using Chinese remainder theorem & 3 messages with different moduli.
- If e fixed must ensure \(\gcd(e, \phi(n)) = 1\).
  - Ie reject any p or q not relatively prime to e.
Efficient Decryption

- decryption uses exponentiation to power d
  - this is likely large, insecure if not
- can use the Chinese Remainder Theorem (CRT) to compute mod p & q separately. then combine to get desired answer
  - approx 4 times faster than doing directly
- only owner of private key who knows values of p & q can use this technique
RSA Key Generation

- **users of RSA must:**
  - determine two primes at random: \( p, q \)
  - select either \( e \) or \( d \) and compute the other

- **primes** \( p, q \) **must not be easily derived from modulus** \( n=p \cdot q \)
  - means must be sufficiently large
  - now, no good technique yielding arbitrarily large prime
  - typically random pick and use probabilistic test
    - Miller-Rabin algorithm

- **exponents** \( e, d \) **are inverses, so use Inverse algorithm** to compute the other
  - Extended Euclid’s algorithm
RSA Security

possible approaches to attacking RSA are:
- brute force key search (infeasible given size of numbers)
- mathematical attacks (based on difficulty of computing $\phi(n)$, by factoring modulus $n$)
- timing attacks (on time running of decryption)
- chosen ciphertext attacks (given properties of RSA)
Factoring Problem

- mathematical approach takes 3 forms:
  - factor $n = p \cdot q$, hence compute $\varphi(n)$ and then $d$
  - determine $\varphi(n)$ directly and compute $d$
  - find $d$ directly

- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663) bit with LS
  - biggest improvement comes from improved algorithm
    - Quadratic Sieve
    - Generalized Number Field Sieve
    - Lattice Sieve
  - currently assume 1024-2048 bit RSA is secure
## Progress in Factorization

<table>
<thead>
<tr>
<th>Number of Decimal Digits</th>
<th>Approximate Number of Bits</th>
<th>Date Achieved</th>
<th>MIPS-years</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>332</td>
<td>April 1991</td>
<td>7</td>
<td>Quadratic sieve</td>
</tr>
<tr>
<td>110</td>
<td>365</td>
<td>April 1992</td>
<td>75</td>
<td>Quadratic sieve</td>
</tr>
<tr>
<td>120</td>
<td>398</td>
<td>June 1993</td>
<td>830</td>
<td>Quadratic sieve</td>
</tr>
<tr>
<td>129</td>
<td>428</td>
<td>April 1994</td>
<td>5000</td>
<td>Quadratic sieve</td>
</tr>
<tr>
<td>130</td>
<td>431</td>
<td>April 1996</td>
<td>1000</td>
<td>Generalized number field sieve</td>
</tr>
<tr>
<td>140</td>
<td>465</td>
<td>February 1999</td>
<td>2000</td>
<td>Generalized number field sieve</td>
</tr>
<tr>
<td>155</td>
<td>512</td>
<td>August 1999</td>
<td>8000</td>
<td>Generalized number field sieve</td>
</tr>
<tr>
<td>160</td>
<td>530</td>
<td>April 2003</td>
<td></td>
<td>Lattice sieve</td>
</tr>
<tr>
<td>174</td>
<td>576</td>
<td>December 2003</td>
<td></td>
<td>Lattice sieve</td>
</tr>
<tr>
<td>200</td>
<td>663</td>
<td>May 2005</td>
<td></td>
<td>Lattice sieve</td>
</tr>
</tbody>
</table>
Timing Attacks

- developed by Paul Kocher in mid-1990’s
- exploit timing variations in operations
- infer operand size based on time taken
- RSA exploits time taken in exponentiation
- countermeasures
  - use constant exponentiation time
    - Performance degradation
  - add random delays
    - Make sure that enough noise is added
  - blind values used in calculations
    - Multiply the ciphertext by a random number before doing exp.
    - 2-10% penalty
Summary

have considered:
- principles of public-key cryptography
- RSA algorithm, implementation, security