Cryptography and Network Security
Chapter 9
Fifth Edition
by William Stallings
Lecture slides by Lawrie Brown
(with edits by RHB)

Chapter 9 – Public Key Cryptography and RSA

Every Egyptian received two names, which were known respectively as the true name and the good name, or the great name and the little name; and while the good or little name was made public, the true or great name appears to have been carefully concealed.

—The Golden Bough, Sir James George Frazer

Outline

• will consider:
  – principles of public-key cryptography
  – RSA algorithm, implementation, security

Private-Key Cryptography

• traditional private/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 and claiming it’s sent by sender (repudiation problem)
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 private key cryptography (efficiency reasons)

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 (NSA (60’s (claimed)), CESG (1970 (documented)))

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 related private-key, known only to the recipient, used to decrypt messages, and sign (create) signatures
- infeasible to determine private key from public (requires solving a hard problem)
- is asymmetric because
  - those who encrypt messages or verify signatures cannot decrypt messages or create signatures
**Public-Key Cryptography**

![Diagram of Public-Key Cryptography]

**Symmetric vs Public-Key**

<table>
<thead>
<tr>
<th>Conventional Encryption</th>
<th>Public-Key Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Needed to Work:</strong></td>
<td></td>
</tr>
<tr>
<td>1. The same algorithm with the same key is used for encryption and decryption.</td>
<td>1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption.</td>
</tr>
<tr>
<td>2. The sender and receiver must share the algorithm and the key.</td>
<td>2. The sender and receiver must each have one of the matched pair of keys (not the same one).</td>
</tr>
<tr>
<td><strong>Needed for Security:</strong></td>
<td></td>
</tr>
<tr>
<td>1. The key must be kept secret.</td>
<td>1. One of the two keys must be kept secret.</td>
</tr>
<tr>
<td>2. It must be impossible or at least impractical to decipher a message if no other information is available.</td>
<td>2. It must be impossible or at least impractical to decipher a message if no other information is available.</td>
</tr>
<tr>
<td>3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.</td>
<td>3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.</td>
</tr>
</tbody>
</table>

**Public-Key Cryptosystems**

Combining secrecy and 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

<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>Diffe-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>
Public-Key Requirements

• 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)
• these are formidable requirements which only a few algorithms have satisfied

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
  (PK schemes are generic and super-polynomial ... can always choose a bigger instance, unlike block ciphers)
• security relies on a large enough difference in difficulty between easy (en/decrypt) and hard (cryptanalyse) problems
• more generally the hard problem is ‘known’, but is made hard enough to be impractical to break
• requires the use of very large numbers
• hence is slow compared to private key schemes

Public-Key Requirements

• need a trapdoor one-way function
• one-way function has
  – $Y = f(X)$ easy
  – $X = f^{-1}(Y)$ infeasible
• a trap-door one-way function has
  – $Y = f_k(X)$ easy, if $k$ and $X$ are known
  – $X = f_k^{-1}(Y)$ easy, if $k$ and $Y$ are known
  – $X = f_k^{-1}(Y)$ infeasible, if $Y$ known but $k$ not known
• a practical public-key scheme depends on a suitable trap-door one-way function

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
  – nb. exponentiation takes $O((\log n)^3)$ operations (easy)
• uses large integers (eg. 1024 bits)
• security due to cost of factoring large numbers
  – nb. factorization takes $O(e^{\log n \log \log n})$ operations (superpolynomial, hard)
RSA En/decryption

- 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 $\text{GCD}(a, n) = 1$
- in RSA have:
  - $n = p \cdot q$
  - $\phi(n) = (p-1)(q-1)$
  - carefully chose $e$ and $d$ to be inverses $\mod \phi(n)$
  - hence $e \cdot d = 1 + k \cdot \phi(n)$ for some $k$
- hence:
  - $C^d = M^{e \cdot d} = M^{1+k \cdot \phi(n)} = M^1 \cdot (M^{\phi(n)})^k$
  - $= M^1 \cdot (1)^k = M^1 = M \mod n$
(provided $M$ and $n$ coprime (still OK if not))

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 $\phi(n) = (p-1)(q-1)$
  - selecting at random the encryption key $e$
    - where $1 < e < \phi(n)$, $\text{gcd}(e, \phi(n)) = 1$
  - solve following equation to find decryption key $d$
    - $e \cdot d = 1 \mod \phi(n)$ and $0 \leq d \leq n$
  - publish their public encryption key: $PU = \{e, n\}$
  - keep secret private decryption key: $PR = \{d, n\}$

<table>
<thead>
<tr>
<th>Key Generation</th>
<th>Encryption</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select $p, q$</td>
<td>$M &lt; n$</td>
<td>$M = n$</td>
</tr>
<tr>
<td>Calculate $n = p \cdot q$</td>
<td>$C = M^e \mod n$</td>
<td>$C$</td>
</tr>
<tr>
<td>Calculate $\phi(n) = (p-1)(q-1)$</td>
<td>$PU = {e, n}$</td>
<td>$PU$</td>
</tr>
<tr>
<td>Select integer $e$</td>
<td>$\text{gcd}(e, \phi(n)) = 1$, $1 &lt; e &lt; \phi(n)$</td>
<td>$\phi(n) = (p-1)(q-1)$</td>
</tr>
<tr>
<td>Calculate $d$</td>
<td>$d = e^{-1} \mod \phi(n)$</td>
<td>$PR = {d, n}$</td>
</tr>
</tbody>
</table>

Figure 9.5 The RSA Algorithm
RSA Example - Key Setup

1. Select primes: \( p = 17 \); \( q = 11 \)
2. Calculate \( n = pq = 17 \times 11 = 187 \)
3. Calculate \( \varphi(n) = (p-1)(q-1) = 16 \times 10 = 160 \)
4. Select \( e \): \( \text{GCD}(e,160) = 1 \); choose \( e = 7 \)
5. Derive \( d \): \( de \equiv 1 \pmod{160} \) and \( d < 160 \)
   Get \( d = 23 \) since \( 23 \times 7 = 161 = 10 \times 160 + 1 \)
6. Publish public key: \( PU = \{7,187\} \)
7. Keep private key secret: \( PR = \{23,187\} \)

RSA Example - En/Decryption

- sample RSA encryption/decryption is:
- given message \( M = 88 \) (nb. \( 88 < 187 \))
- encryption: \( C = 88^7 \pmod{187} = 11 \)
- decryption: \( M = 11^{23} \pmod{187} = 88 \)

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. \( 7^5 = 7^4 \cdot 7^1 = 3 \cdot 7 = 10 \pmod{11} \)
  - eg. \( 3^{129} = 3^{128} \cdot 3^1 = 5 \cdot 3 = 4 \pmod{11} \)

Exponentiation

Computing \( a^b \pmod{n} \)

\[
f = 1 \\
\text{for } i = k \text{ downto } 0 \text{ do } f = (f \times f) \pmod{n} \\
\text{if } b_i \equiv 1 \text{ then } f = (f \times a) \pmod{n}
\]

return \( f \)

Here, integer \( b \) is the bitstring \( b_k b_{k-1} \ldots b_0 \)
Efficient Encryption

- Encryption uses exponentiation to power $e$
- Hence if $e$ small, this will be faster
  - Often choose $e = 65537 = 2^{16} - 1$
  - Also see choices of $e = 3$ or $e = 17$
- But if $e$ too small (e.g., $e = 3$) can attack
  - Using Chinese remainder theorem and 3 messages with different moduli
- If $e$ fixed must ensure $\text{GCD}(e, \varphi(n)) = 1$
  - I.e., reject any $p$ or $q$ where $p-1$ or $q-1$ are 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 $d \mod p$ and $d \mod q$ separately; then combine to get answer
  - Approx 4 times faster than doing directly
- Only owner of private key who knows values of $p$ and $q$ 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
  - Typically guess and use probabilistic test
- Exponents $e, d$ are inverses, so use Inverse algorithm to compute the other

RSA Security

- Possible approaches to attacking RSA are:
  - Brute force key search - Infeasible given size of numbers
  - Mathematical attacks - Based on difficulty of computing $\varphi(n)$, by factoring modulus $n$
  - Timing attacks - On 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
    - cf QS to GNFS to LS
  - currently assume 1024-2048 bit RSA is secure
    - ensure \( p, q \) of similar size and matching other constraints

Progress in Factoring

<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>820</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>451</td>
<td>April 1996</td>
<td>1000</td>
<td>generalised number field sieve</td>
</tr>
<tr>
<td>140</td>
<td>485</td>
<td>February 1999</td>
<td>2000</td>
<td>generalised number field sieve</td>
</tr>
<tr>
<td>155</td>
<td>512</td>
<td>August 1999</td>
<td>8000</td>
<td>generalised number field sieve</td>
</tr>
<tr>
<td>160</td>
<td>530</td>
<td>April 2000</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>603</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
  - eg. multiplying by small vs large number
  - or IF’s varying which instructions executed
- infer operand size based on time taken
- RSA exploits time taken in exponentiation
- countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations
Chosen Ciphertext Attacks

- RSA is vulnerable to a Chosen Ciphertext Attack (CCA)
- attacker chooses ciphertexts and gets decrypted plaintext back
- choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis
- can counter with random pad of plaintext
- or use Optimal Asymmetric Encryption Padding (OASP)