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 make it work
- complements **rather than** replaces private key cryptography (efficiency reasons)

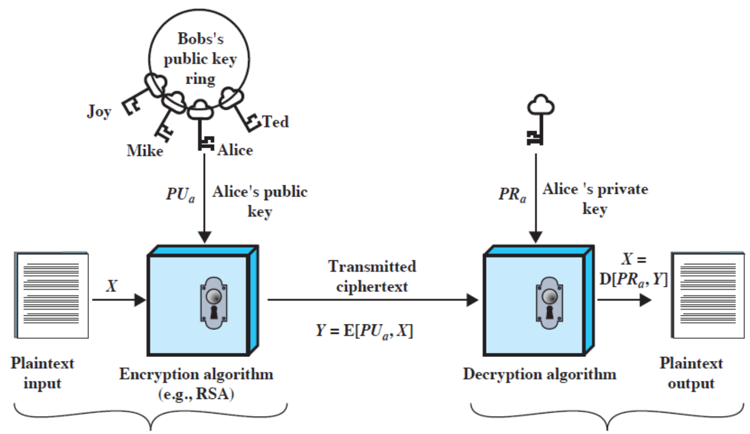
Why Public-Key Cryptography?

- 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

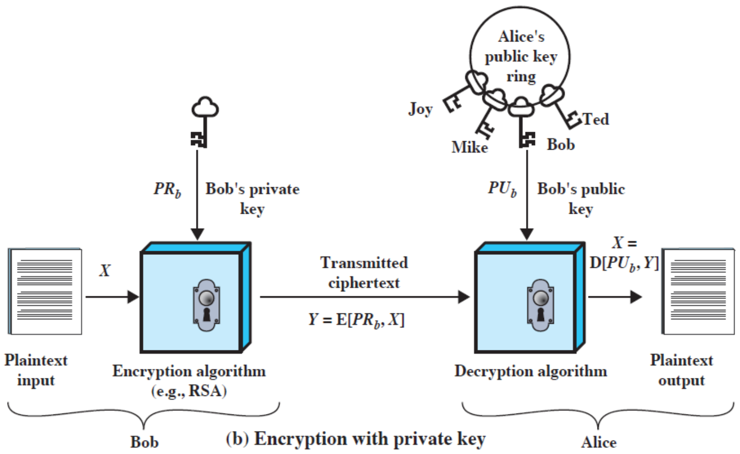
- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a related **private-key**, known only to the recipient, used to **decrypt messages**, and **sign**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



(a) Encryption with public key

Public-Key Cryptography

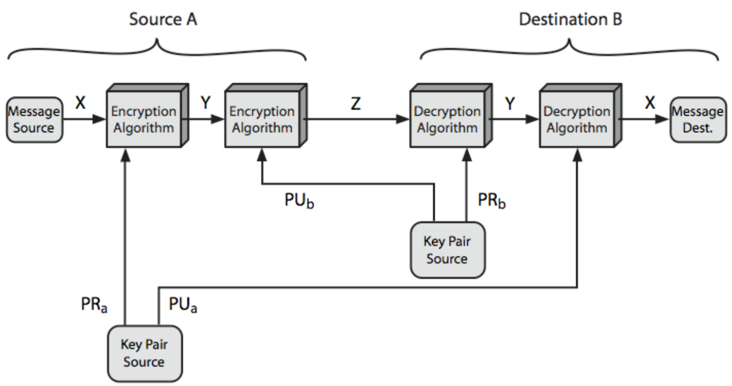


(b) Encryption with private key

Symmetric vs Public-Key

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

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

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

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

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

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

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, or 2048 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: $C = M^e \bmod n$, where $0 \leq M < n$
- to decrypt the ciphertext C the owner:
  - uses their private key $PR = \{d, n\}$
  - computes: $M = C^d \bmod n$
- note that the message M must be smaller than the modulus n (block if needed)

RSA Key Setup

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

Key Generation

Select p, q	p and q 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:	$C = M^e \bmod n$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Figure 9.5 The RSA Algorithm

Why RSA Works

- because of Euler's Theorem:
  - $a^{ø(n)} \bmod n = 1$ where $GCD(a, n) = 1$
- in RSA have:
  - $n = p.q$
  - $ø(n) = (p-1)(q-1)$
  - carefully chose e and d to be inverses $\bmod ø(n)$
  - hence $e.d = 1 + k.ø(n)$ for some k
- hence :

$$C^d = M^{e.d} = M^{1+k.ø(n)} = M^1.(M^{ø(n)})^k$$
$$= M^1.(1)^k = M^1 = M \bmod n$$

(provided M and n coprime (still OK if not))

RSA Example - Key Setup

1. Select primes: $p = 17$; $q = 11$
2. Calculate $n = pq = 17 \times 11 = 187$
3. Calculate $\varnothing(n) = (p-1)(q-1) = 16\text{x}10 = 160$
4. Select e: $GCD(e,160) = 1$; choose $e = 7$
5. Derive d: $de = 1 \bmod 160$ and $d < 160$
   Get $d = 23$ since $23\text{x}7 = 161 = 10\text{x}160+1$
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:
  $C = 88^7 \bmod 187 = 11$
- decryption:
  $M = 11^{23} \bmod 187 = 88$

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. $7^5 = 7^4.7^1 = 3.7 = 10 \bmod 11$
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \bmod 11$

Exponentiation

Computing $a^b \bmod n$

```
f = 1
for i = k downto 0
    do f = (f x f) mod n
       if b_i == 1 then
          f = (f x a) mod 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 (eg. $e = 3$) can attack
  - using Chinese remainder theorem and 3 messages with different moduli
- if `e` fixed must ensure $GCD(e, ø(n)) = 1$
  - ie 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 `mod p` and `mod q` separately; then combine to get answer
  - approx 4 times faster than doing it 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 $n = p.q$
  - means `p,q` 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 ø(`n`), by factoring modulus `n`
  - timing attacks - on running of decryption
  - chosen ciphertext attacks - given properties of RSA

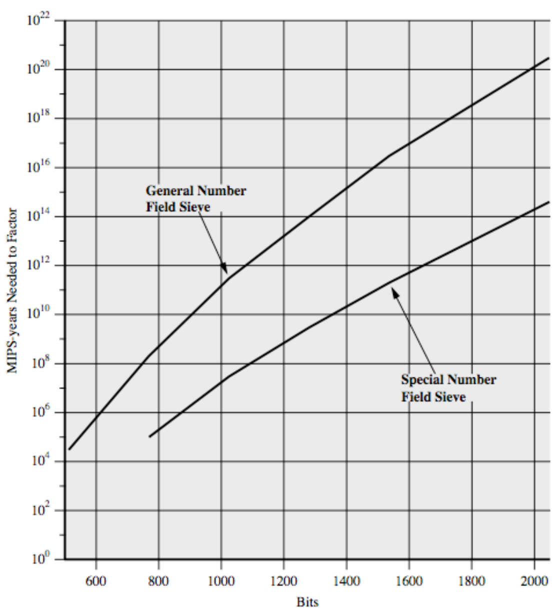
Factoring Problem

- mathematical approach takes 3 different forms:
  - factor n = p.q, hence compute ø(n) and then d
  - determine ø(n) directly and compute d
  - find d directly
- currently believe all these 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

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	quadratic sieve
110	365	April 1992	75	quadratic sieve
120	398	June 1993	830	quadratic sieve
129	428	April 1994	5000	quadratic sieve
130	431	April 1996	1000	generalized number field sieve
140	465	February 1999	2000	generalized number field sieve
155	512	August 1999	8000	generalized number field sieve
160	530	April 2003	—	Lattice sieve
174	576	December 2003	—	Lattice sieve
200	663	May 2005	—	Lattice sieve

Progress in Factoring



Timing Attacks

- developed by Paul Kocher in mid-1990's
- exploit timing variations in operations
  - eg. multiplying by small vs. large number
  - or varying which instructions executed
- infer operand size based on time taken
- For RSA, exploits time taken for 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)
- based on C(P1 x P2) = C(P1) x C(P2)
- 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)

Optimal Asymmetric Encryption Padding (OASP)