Chapter 9

Public Key Cryptography, RSA And Key Management

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- The most widely used public-key cryptosystem is RSA. The difficulty of attacking RSA is based on the difficulty of finding the prime factors of a composite number.

- RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and *n - 1 for some n. A typical size for n is 1024 bits*

RSA Algorithm

Plaintext is encrypted in blocks and each block's binary value $2^k < n$

Block size $k < \log_2(n)$

Encryption (plaintext block M, Ciphertext C)

$$C = M^e \bmod n \quad , \text{ where } 0 \le M < n$$

Decryption

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Both sender and receiver know (n)

The sender knows (e)	The receiver know (d)
Public key of PU ={e, n}	Private key of PR ={d, n}

For public-key requirements

Find values of *e, d, n such that* $M^{ed} \bmod n = M$ *for all* $M < n$

Symbol	Expression	Meaning
gcd	$\gcd(i, j)$	Greatest common divisor; the largest positive integer that divides both i and j with no remainder on division.
φ	$\phi(n)$	The number of positive integers less than n and relatively prime to n. This is Euler's totient function.

RSA Key Setup

To find a relationship $M^{ed} \bmod n = M$

Each user generates a public/private key pair by:

Select two large primes at random: p, q n=p.q

$\phi(n)$ is the Euler totient function

$$\varnothing(n)=(p-1)(q-1) \qquad \varnothing(pq)=(p-1)(q-1)$$

Select at random the encryption key e

where $1<e<\varnothing(n)$, $\gcd(e,\varnothing(n))=1$

Find decryption key d

$$ed \equiv 1 \bmod \phi(n)$$

$$d \equiv e^{-1} \bmod \phi(n)$$

- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}
- The private key consists of {d, n} and the public key consists of {e, n}. Suppose that user A has published its public key and that user B wishes to send the message M to A. Then B calculates $C = M^e \bmod n$ and transmits C. On receipt of this ciphertext, user A decrypts by calculating $M = C^d \bmod n$.

RSA Example - Key Setup

1. Select primes: p=17 & q=11
2. Calculate $n = pq$ =17 x 11=187
3. Calculate $\varnothing(n)=(p-1)(q-1)$=16x10=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 23x7=161= 10x16+1

   – repeat *d = (k φ(n) + 1) / e*

   – increment *k until you get an integer value*

   – 1 × 160 + 1 = 161; d = 161/7 = 23
6. Publish public key PU={7,187}
7. Keep secret private key PR={23,187}

- sample RSA encryption/decryption is:
- given message $M = 88$ (nb. $88<187$)
- encryption:

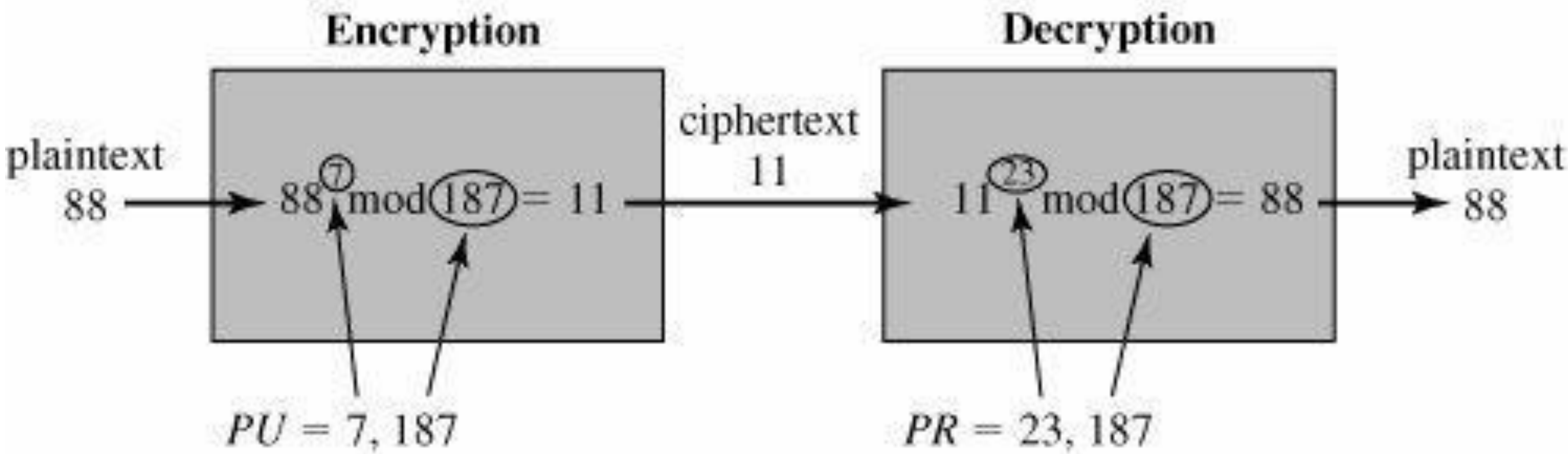
  $C = 88^7 \bmod 187 = 11$

- decryption:

  $M = 11^{23} \bmod 187 = 88$

PU = {7, 187}　　　　PR = {23, 187}

Let M = 88



Exponentiation in Modular Arithmetic

- We can use property of modular arithmetic

  $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$

- To calculate $x^{11} \bmod n$
  - $x^{11} = x^{1+2+8} = (X)(X^2)(X^8)$
  - compute $x \bmod n$, $x^2 \bmod n$, $x^4 \bmod n$, $X^8 \bmod n$
  - calculate $[(X \bmod n) \times (X^2 \bmod n) \times (X^8 \bmod n)] \bmod n$

Example

$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (881 \bmod 187)] \bmod 187$

$88^1 \bmod 187 = 88$

$88^2 \bmod 187 = 7744 \bmod 187 = 77$

$88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$

$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894{,}432 \bmod 187 = 11$

Example

11^{23} **mod 187=[(11^1 mod 187)×(11^2 mod 187)×(11^4 mod 18^7)×(11^8 mod 18^7) × (11^8 mod 187)] mod 187**

11^1 mod 187 = 11

11^2 mod 187 = 121

11^4 mod 187 = 14,641 mod 187 = 55

11^8 mod 187 = 214,358,881 mod 187 = 33

11^{23} mod 187=(11 × 121 × 55 × 33 × 33) mod 187=79,720,245 mod 187=88

RSA Key Generation

- **Before the application of the public-key cryptosystem, each participant must generate a pair of keys, which requires finding primes and computing inverses.**
- 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 must be sufficiently large
  - typically guess and use probabilistic test
- exponents e, d are inverses, so use Inverse algorithm to compute the other

Finding Inverses

- Can extend Euclid's algorithm:

EXTENDED EUCLID (*m, b)*

1. (A1, A2, A3) = (1, 0, *m);*
   (B1, B2, B3) = (0, 1, *b)*
2. **if** B3 = 0
   return A3 = gcd (*m, b); no inverse*
3. **if** B3 = 1
   return B3 = gcd (*m, b); B2 =* b^{-1} *mod m*
4. Q = A3 div B3
5. (T1, T2, T3) =(A1 – Q B1, A2 – Q B2, A3 – Q B3)
6. (A1, A2, A3)=(B1, B2, B3)
7. (B1, B2, B3) =(T1, T2, T3)
8. **goto** 2

Inverse of 550 in GF(1759)

Q	A1	A2	A3	B1	B2	B3
—	1	0	1759	0	1	550
3	0	1	550	1	–3	109
5	1	–3	109	–5	16	5
21	–5	16	5	106	–339	4
1	106	–339	4	–111	355	1

Inverse of 7 in GF(160)

Q	A1	A2	A3	B1	B2	B3
—	1	0	160 (φ)	0	1	7 (e)
22	0	1	7	1	–22	6
1	1	–22	6	-1	23 Inverse (d)	1 gcd

Factoring Problem

- mathematical approach takes 3 forms:
  - n= p.q, hence ø(n)=(p-1)(q-1)and then d

$$d \equiv e^{-1} \pmod{\phi(n)}$$

  - determine ∅(n) directly without P & q and compute d

$$d \equiv e^{-1} \pmod{\phi(n)}$$

  - find d directly without ∅(n)

Key Management

- Public-key encryption helps address key distribution problems
- Have two aspects of this:
  - distribution of public keys
  - use of public-key encryption to distribute secret keys

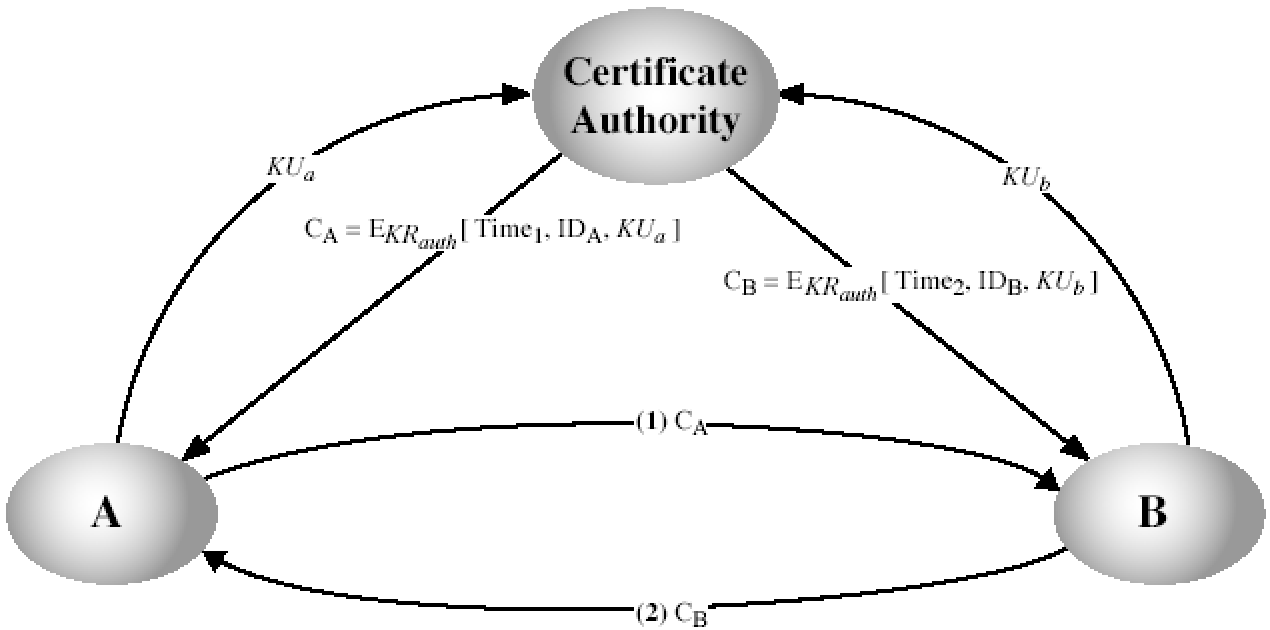
Distribution of Public Keys

- Can be considered as using of:
  - Public-key certificates

Public-Key Certificates

- Certificates allow key exchange without real-time access to public-key authority
- A certificate binds **identity to public key**
  – Usually with other info such as period of validity, rights of use etc
- With all contents **signed by a trusted Public-Key or Certificate Authority (CA)**
- Can be verified by anyone who knows the public-key authorities public-key

Public-Key Certificates



1. A sends a message to the public-key authority containing a request for the current public key of B.
2. The authority responds with a message that is encrypted using the authority's private key, PR_{auth} Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:
   - B's public key, *PUb* which A can use to encrypt messages destined for B
   - The original request, to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority
3. A stores B's public key and also uses it to encrypt a message to B
4. B retrieves A's public key from the authority in the same manner as A retrieved B's public key. At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange.

Public-Key Distribution of Secret Keys

- use previous methods to obtain public-key
- can use for secrecy or authentication
- but public-key algorithms are slow
- so usually want to use private-key encryption to protect message contents
- hence need a session key
- have several alternatives for negotiating a suitable session

Simple Secret Key Distribution

- proposed by Merkle in 1979
  - –A generates a new temporary public key pair
  - –A sends B the public key and their identity
  - –B generates a session key K sends it to A encrypted using the supplied public key
  - –A decrypts the session key and both use

Diffie-Hellman Key Exchange

- first public-key type scheme proposed
- by Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that James Ellis (UK CESG) secretly proposed the concept in 1970
- is a practical method for public exchange of a secret key
- used in a number of commercial products

Diffie-Hellman Key Exchange

- a public-key distribution scheme
  - –cannot be used to exchange an arbitrary message
  - –rather it can establish a common key
  - –known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

Diffie-Hellman Setup

- all users agree on global parameters:
  - –large prime integer or polynomial q
  - –α a primitive root mod q
- each user (eg. A) generates their key
  - –chooses a secret key (number): $X_A < q$
  - –compute their **public key:** $y_A = \alpha^{x_A} \bmod q$
- each user makes public that key y_A

Diffie-Hellman Key Exchange

- shared session key for users A & B is KAB:

$$
\begin{aligned}
K_{AB} &= \alpha^{x_A.x_B} \bmod q \\
&= y_A^{x_B} \bmod q \quad \text{(which } \mathbf{B} \text{ can compute)} \\
&= y_B^{x_A} \bmod q \quad \text{(which } \mathbf{A} \text{ can compute)}
\end{aligned}
$$

- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the same key as before, unless they choose new public-keys
- attacker needs an x, must solve discrete log

Diffie-Hellman Example

- Users Alice & Bob who wish to swap keys:
- Agree on prime q=353 and α=3
- Select random secret keys:
  - **A** chooses x_A=97, **B** chooses x_B=233
- Compute public keys:
  - $y_A = 3^{97} \bmod 353 = 40$ (Alice)
  - $y_B = 3^{233} \bmod 353 = 248$ (Bob)
- Compute shared session key as:

$$K_{AB} = y_B^{x_A} \bmod 353 = 248^{97} = 160 \quad \text{(Alice)}$$
$$K_{AB} = y_A^{x_B} \bmod 353 = 40^{233} = 160 \quad \text{(Bob)}$$