7 Public-Key Cryptography

Ch9,10 in textbook

Yanwei Yu

E-mail: ywyu@ustc.edu.cn

Outline

- Principles of Public-Key Cryptosystems
- The RSA Algorithm
- Distribution of Public Keys
- Elliptic Curve Cryptography

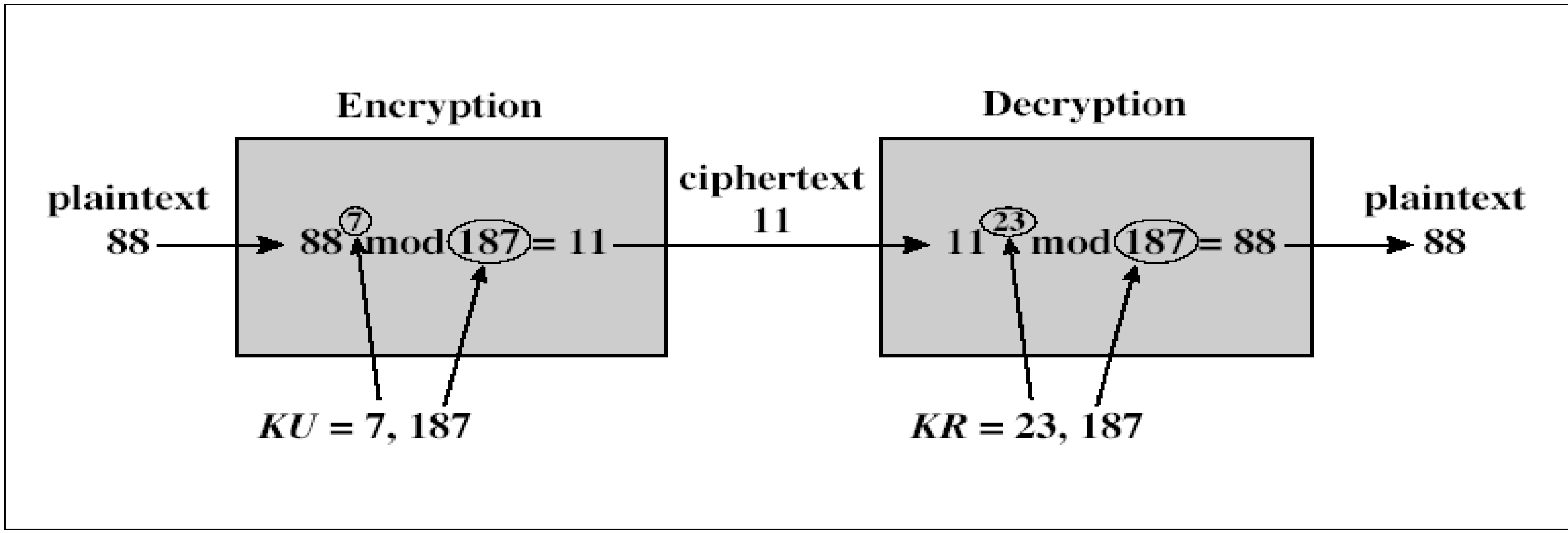
Outline

- Principles of Public-Key Cryptosystems
- The RSA Algorithm
- Distribution of Public Keys
- Elliptic Curve Cryptography

Evolution of Cryptography

- Before 1976, all cryptographic systems have been based on the elementary tools of substitution and permutation
  - calculated by hand
  - with the development of the rotor encryption/decryption machine. The electromechanical rotor enabled the development of fiendishly complex cipher systems.
  - With the availability of computers, even more complex systems were devised, the most prominent of which was the Lucifer effort at IBM that culminated in the Data Encryption Standard (DES).
- 1976, the concept of public-key cryptography is developed by Diffie and Hellman.
- public-key algorithms are based on mathematical functions rather than on substitution and permutation

Public-key Encryption



Public key: 7 and 187 , Private key: 23

Plain-text 88 cannot be concluded from only 7, 187 and cipher-text 11

Mathematics is so wonderful!

Common Misconceptions and Facts about Public-key Encryption

Misconceptions

- public-key encryption is more secure from cryptanalysis than is symmetric encryption
- public-key encryption is a general-purpose technique that has made symmetric encryption obsolete(过时)
- Public-key distribution is easy compared to secret key distribution

Facts

- security of any encryption scheme depends on the length of the key and the computational cost involved in breaking a cipher
- symmetric encryption will not be abandoned and public-key cryptography is used for key management and signature applications.
- authenticity of distributed public key should be assured

Private-Key Cryptography

- **traditional private/symmetric/secret/ single key cryptography uses <u>one key</u>**
- **<u>shared by both</u> sender and receiver**
- **also is <u>symmetric</u>, parties are <u>equal</u>**
- **used for data <u>confidentiality</u> applications.**

Why Public-Key Cryptography?

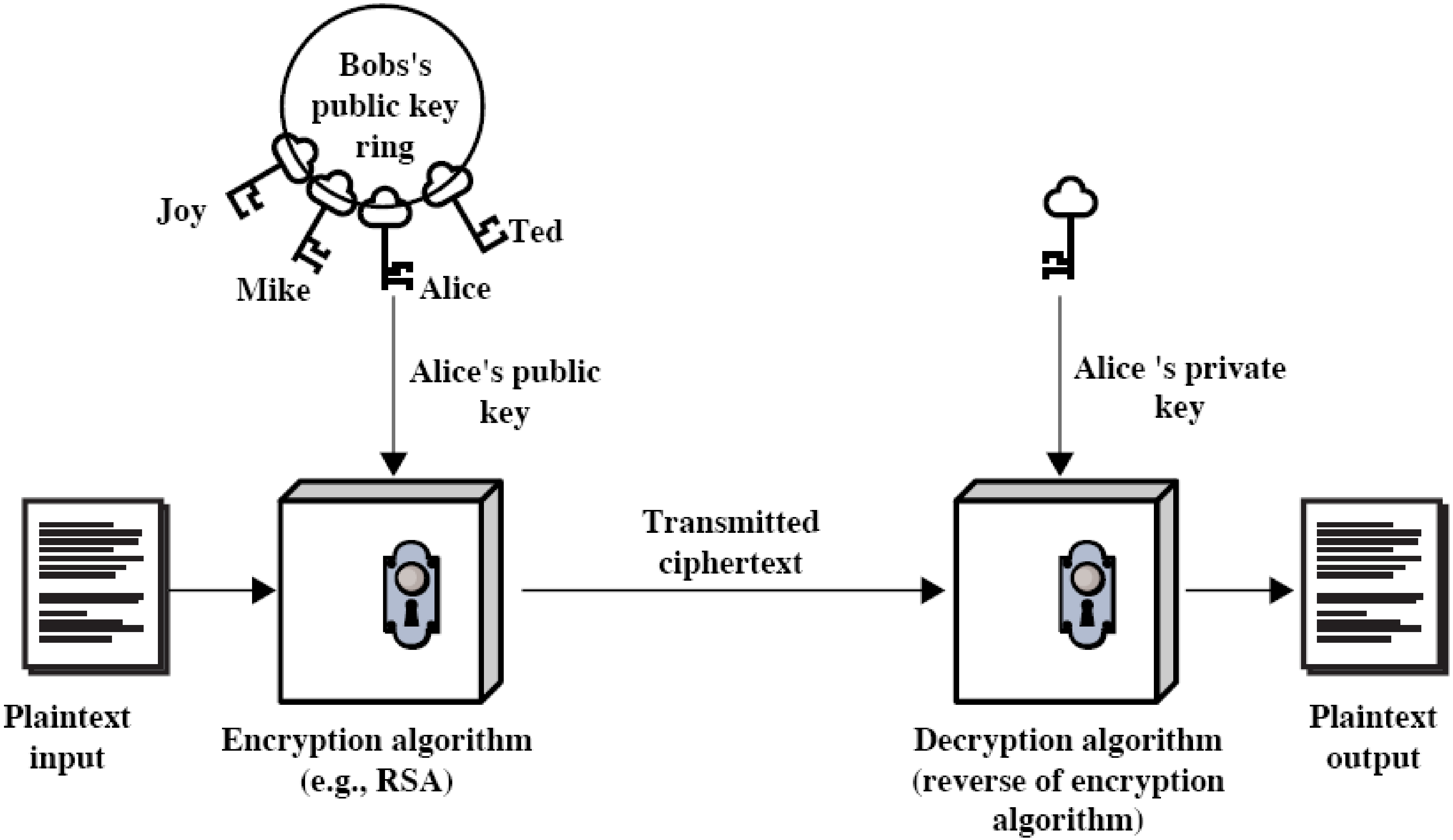
- developed to address two key issues:
  - key distribution – how to have secure communications in general without having to trust a KDC with your key
  - digital signatures – how to verify a message comes intact(完整的) from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman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

Public-Key Cryptography

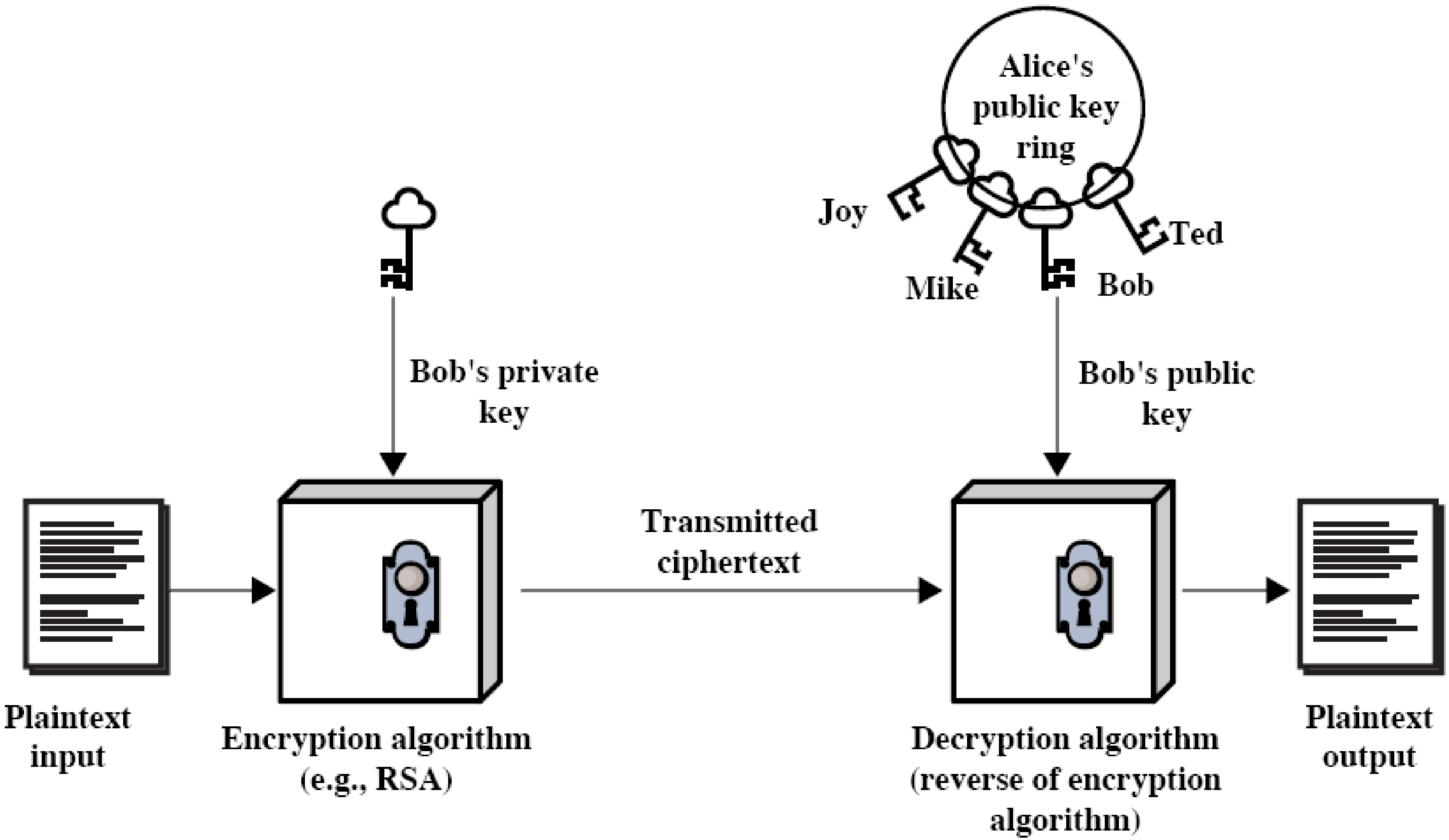
- probably most significant advance in the 3000 year history of cryptography
- uses two keys – a public & a private key
- asymmetric since parties are not equal
- Security depends on number theoretic problems.
- complements rather than replaces private key crypto
  - Digital Envelope

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**



(a) Encryption



(b) Authentication

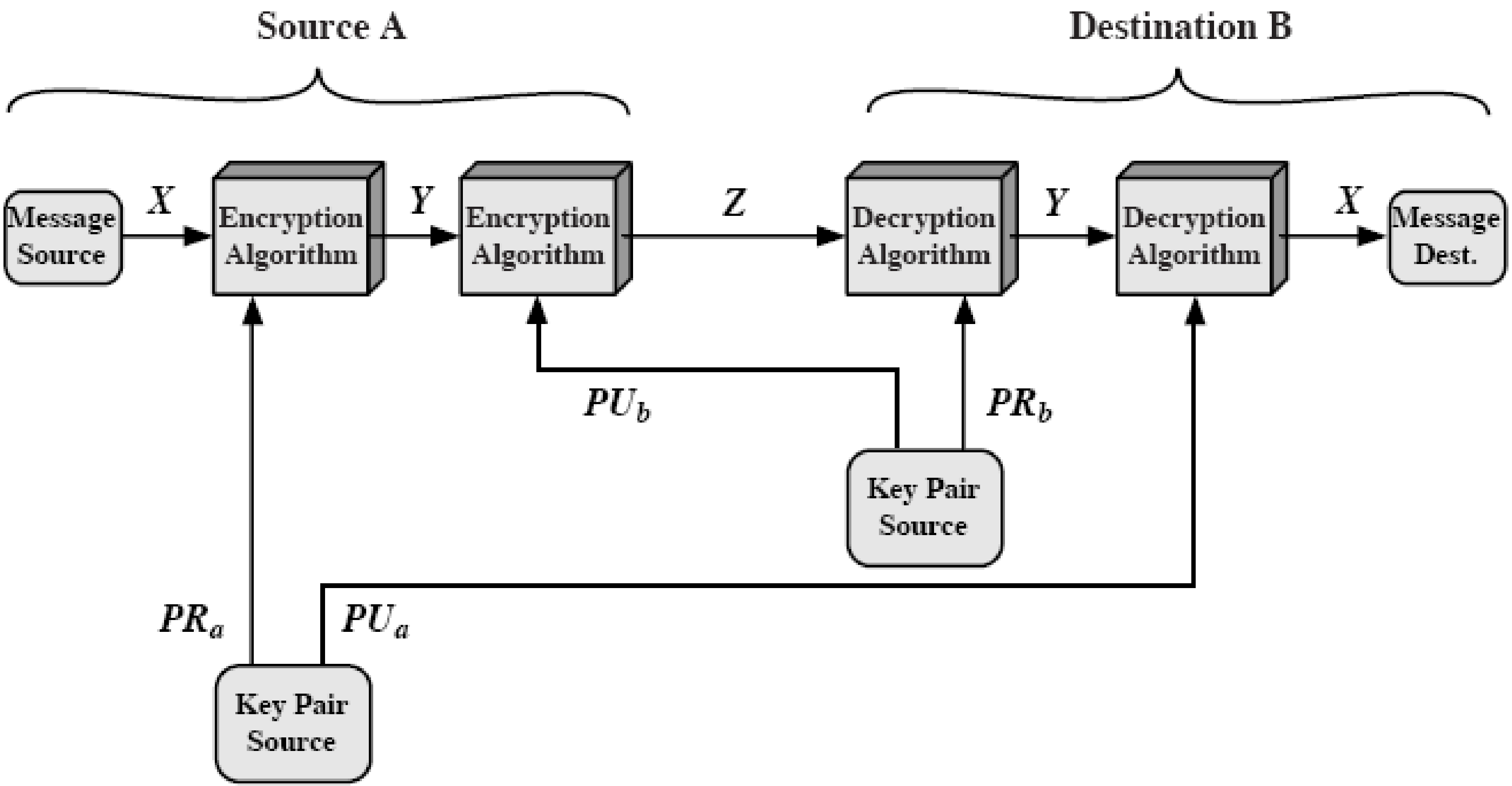


Figure 9.4 Public-Key Cryptosystem: Authentication and Secrecy

Conventional Encryption

- **Needed to Work:**
  - The same algorithm with the same key is used for encryption and decryption.
  - The sender and receiver must share the algorithm and the key.
- **Needed for Security:**
  - The key must be kept secret.
  - It must be impossible or at least impractical to decipher a message if no other information is available.
  - Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.

Public-Key Encryption

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

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 Characteristics

- **Public-Key algorithms rely on two keys where:**
  - **either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)**
  - **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**

Requirements for Public-Key Cryptography

①computationally easy to generate a pair (Pu_b and PR_b).

②computationally easy to compute cipher-text for a sender A knowing the public key and the plain-text M : $C = E(PU_b, M)$

③computationally easy to recover the original message for the receiver B knowing cipher-text and private key : $M = D(PR_b, C)$

④computationally infeasible for an adversary, knowing the public key PU_b, to determine <u>private key PR_b</u>.

⑤computationally infeasible for an adversary, knowing the public key Pu_b and a cipher-text C, to <u>recover M</u>.

⑥$M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$ (not necessary)

- By now, <u>only a few</u> algorithms (RSA, elliptic curve cryptography, Diffie-Hellman, DSS) have received widespread acceptance in the several decades. Why?
- Key Point: discover a suitable <u>trap-door</u> <u>one-way function</u>

$Y = f_k(X)$ 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 is known but k is not known

- <u>one-way function</u>:
  - $Y = f(X)$ easy
  - $X = f^{-1}(Y)$ infeasible
- <u>one-way hash function</u>:
  - maps an arbitrarily large data to a fixed output.
  - used for authentication

Public-Key Cryptanalysis

Attack

- Also vulnerable to a brute-force attack
- find some way to compute the private key given the public key
- probable-message attack
  - public-key encryption is currently confined to key management and signature applications, hence message is short, E.g. secret Key for DES is 56 bits.

Countermeasure

- Use large keys
- No
- Append some random bits to such simple messages

Security vs. Efficiency

- Requires large-size keys used (>512bits)
  - like private key schemes brute force exhaustive search attack is always theoretically possible
- Requires the use of very large numbers
  - 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
- hence is slow compared to private key schemes

Outline

- Principles of Public-Key Cryptosystems
- The RSA Algorithm
- Distribution of Public Keys
- Elliptic Curve Cryptography

Concepts from number theory

- **Prime number:** (c.f. Section 2.4 in textbook)
  - is an integer that can only be divided without remainder by positive and negative values of itself and 1.
- **Greatest Common Divisor**:(c.f. Section 2.2 in textbook)
  - gcd[a(x), b(x)] is the polynomial of maximum degree that divides both a(x) and b(x).
- **Euler's totient function(欧拉函数)** ø(n) (c.f. Section 2.5 in textbook)
  - ø(n)defined as the number of positive integers less than n and relatively prime to n
- **Euler's Theorem(欧拉定理）**(c.f. Section 2.5 in textbook)
  - $a^{ø(n)}$mod n = 1 where gcd(a,n)=1.
- **Fermat's Theorem(费马定理)**(c.f. Section 2.5 in textbook)
  - a^{p-1} mod p=1, where p is prime and a is a positive integer not divisible by p.
- **Chinese remainder theorem(CRT)(中国剩余定理)**(c.f. Section 2.7)
  - provides a way to manipulate (potentially very large) numbers mod M in terms of tuples of smaller numbers.

```
OpenSSL> prime 136
88 is not prime
```

88 is by Hex

- **Modular arithmetic exhibits the following properties: (c.f. Section 2.3 in textbook)**
  - **[(a mod n) + (b mod n)] mod n = (a + b) mod n**
  - **[(a mod n)-(b mod n)] mod n = (a - b) mod n**
  - **[(a mod n) x (b mod n)] mod n = (a x b) mod n**
- **E.g.**

11 mod 8 = 3; 15 mod 8 = 7

[(11 mod 8) + (15 mod 8)] mod 8 = 10 mod 8 = 2
(11 + 15) mod 8 = 26 mod 8 = 2

[(11 *mod* 8) (15 *mod* 8)] *mod* 8 = 4 *mod* 8 = 4
(11 15) *mod* 8 = 4 *mod* 8 = 4

[(11 mod 8) x (15 mod 8)] mod 8 = 21 mod 8 = 5
(11 x 15) mod 8 = 165 mod 8 = 5

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 (hard)

RSA Key Setup

- selecting two large primes at random： p, q
- computing their system modulus n=p.q
  - note Euler's totient function(欧拉函数) ø(n)=(p-1)(q-1)
  - ø(n)defined as the number of positive integers less than n and relatively prime to n
- 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 mod ø(n) and 0≤d≤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: $C = M^e \bmod n$, where $0 \le M < n$
- 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)

Why RSA Works

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

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

$$= M^1.(1)^k = M^1 = M \bmod n$$

(detail proof can be seen at Appendix R in textbook)

RSA Example - Key Setup

1. **Select primes:** p=17 & q=11
2. **Compute** $n = pq$ =17 x 11=187
3. **Compute** $ø(n)=(p–1)(q-1)$=16 x 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 23x7=161= 10x160+1
6. **Publish public key PU={7,187}**
7. **Keep secret private key PR={23,187}**

RSA Example - En/Decryption

- given message M = 88 (nb. 88<187)
- PU={7,187}, PR={23,187}
- encryption:

  $C = 88^7 \bmod 187 = 11$
- decryption:

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

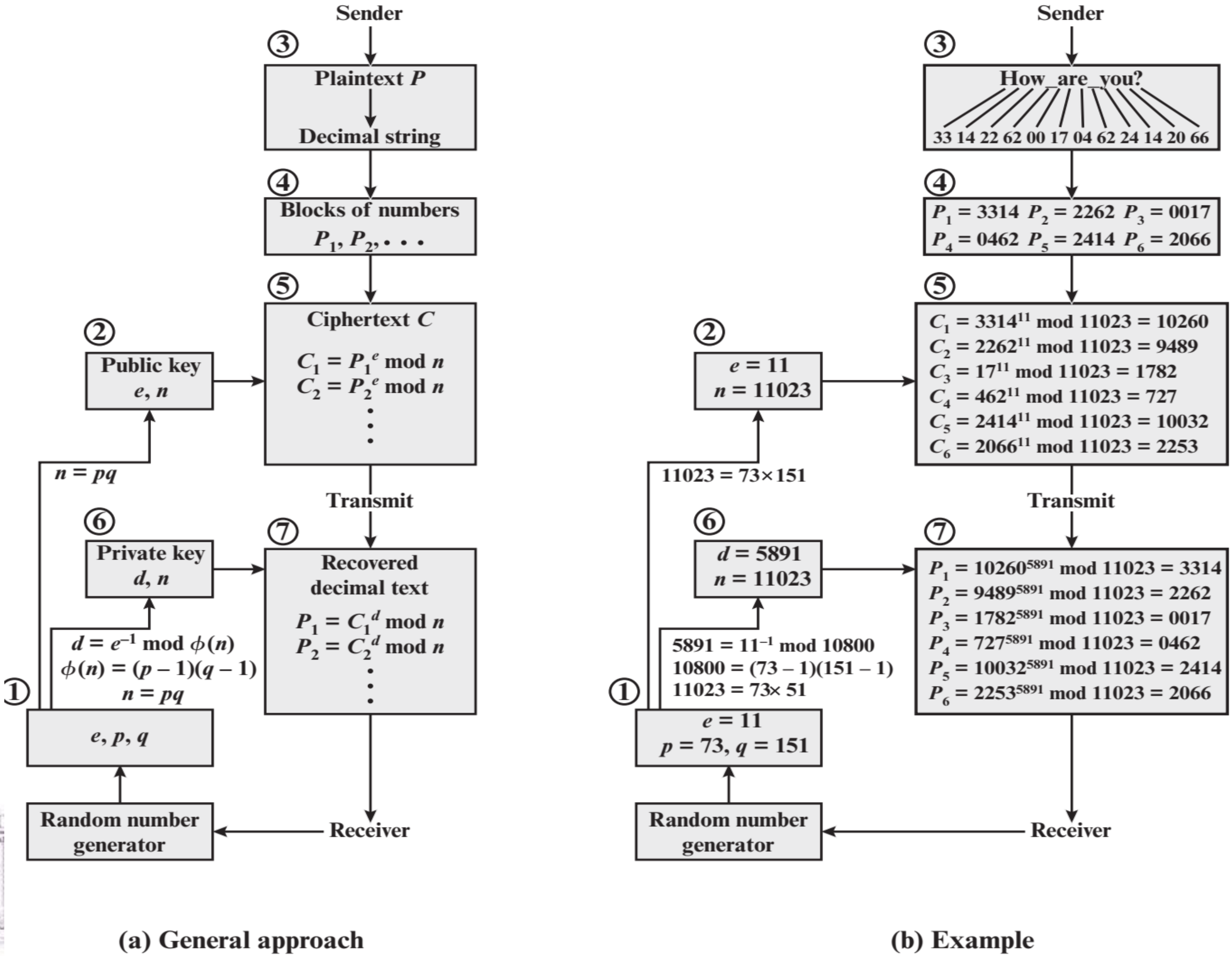


Figure 9.7 RSA Processing of Multiple Blocks

Computational Aspects

- Exponentiation in Modular Arithmetic
- Efficient Operation Using Public Key
- Efficient Operation Using Private Key
- Key Generation

Exponentiation in Modular Arithmetic

- **Use a fast, efficient algorithm for exponentiation** can use the Square and Multiply Algorithm
  - 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. $7^5 = 7^4 * 7^1 = (7^2)^2 * 7^1 = 3*7 = 10 \bmod 11$
  - eg. $3^{129} = 3^{128} * 3^1 = (3^2)^2)^2)^2)^2)^2)^2 * 3^1 = 5*3 = 4 \bmod 11$

```
c = 0; f = 1
for i = k downto 0
    do {c = 2 x c
       f = (f x f) mod n}
    if b_i == 1 then
       {c = c + 1
         f = (f x a) mod n }
 return f
```

Target: Compute $f=a^b \bmod n$

Where $b=(b_k b_{k-1} \ldots b_0)_2$

E.g. compute $7^5 \bmod 11=10$

(namely, $a=7$, $b=5=(101)_2$, $n=11$)

i	2	1	0
b_i	1	0	1
c	1	2	5
f	7	$5=7^2 \bmod 11$	$3=5^2 \bmod 11$; $f=10=(3*7) \bmod 11$

Efficient Operation Using Public Key

- uses exponentiation to power e
- hence if e has small number of 1 bits, this will be faster
  - often choose $e=65537 = 2^{16}+1$
  - also see choices of $e=3$ or $e=17=2^4+1$
- but if e too small (eg e=3) can attack
  - using Chinese remainder theorem & 3 encypted messages with different modulus
- if e fixed must ensure $\gcd(e,ø(n))=1$
  - note prime e cannot ensure $\gcd(e,ø(n))=1$
  - reject any p,q where (p-1) or (q-1) is not relatively prime to e

```
OpenSSL> genrsa -?
usage: genrsa [args] [numbits]
 -des            encrypt the generated key with DES
 -des3           encrypt the generated key with DES
 -idea           encrypt the generated key with IDEA
 -aes128, -aes192, -aes256
                 encrypt PEM output with cbc aes
 -out file       output the key to 'file
 -passout arg    output file pass phrase source
 -f4             use F4 (0x10001) for the E value
 -3              use 3 for the E value
 -engine e       use engine e, possibly a hardware d
 -rand file;file;...
                 load the file (or the files in the
                 the random number generator
```

Assume $A \in Z_M$, $A \leftrightarrow (a_1, a_2, \ldots, a_k)$

can be precalculated

$$A \equiv \left(\sum_{i=1}^{k} a_i \boxed{c_i}\right) (\bmod M)$$

$$\boxed{M = \prod_{i=1}^{k} m_i \quad \text{where } \gcd(m_i, m_j) = 1 \text{ for } 1 \leq i, j \leq k, \text{ and } i \neq j.}$$

$M_i = M/m_i$ for $1 \leq i \leq k$.

$$c_i = M_i \times (M_i^{-1} \bmod m_i) \quad \text{for } 1 \leq i \leq k$$

$$\boxed{a_i = A \bmod m_i \text{ for } 1 \leq i \leq k.}$$

- **Use of CRT**
  - **provides a way to manipulate (potentially very large) numbers mod M in terms of tuples of smaller numbers.**
  - **This can be useful when M is 150 digits or more.**
  - **But note that it is necessary to know beforehand the factorization of M.**

To represent 973 mod 1813 as a pair of numbers mod 37 and 49,

define

$$m_1 = 37$$
$$m_2 = 49$$
$$M = 1813$$
$$A = 973$$

We also have $M_1 = 49$ and $M_2 = 37$. Using the extended Euclidean algorithm, we compute $M_1^{-1} = 34 \bmod m_1$ and $M_2^{-1} = 4 \bmod m_2$. (Note that we only need to compute each M_i and each M_i^{-1} once.) Taking residues modulo 37 and 49, our representation of 973 is $(11, 42)$, because $973 \bmod 37 = 11$ and $973 \bmod 49 = 42$.

So, 973 $\leftrightarrow$ (11,42)

If $A \leftrightarrow (a_1, a_2, \ldots, a_k)$

$B \leftrightarrow (b_1, b_2, \ldots, b_k)$

Then

$$(A + B) \bmod M \leftrightarrow ((a_1 + b_1) \bmod m_1, \ldots, (a_k + b_k) \bmod m_k)$$
$$(A - B) \bmod M \leftrightarrow ((a_1 - b_1) \bmod m_1, \ldots, (a_k - b_k) \bmod m_k)$$
$$(A \times B) \bmod M \leftrightarrow ((a_1 \times b_1) \bmod m_1, \ldots, (a_k \times b_k) \bmod m_k)$$

$$(A + B) \bmod M \leftrightarrow ((a_1 + b_1) \bmod m_1, \ldots, (a_k + b_k) \bmod m_k)$$

- **Now suppose we want to add 678 to 973. What do we do to (11, 42)? (973 ↔ (11,42))**
- **First we compute (678) ↔(678 mod 37, 678 mod 49)=(12, 41).**
- **Then we add the tuples element-wise and reduce (11+12 mod 37, 42+41 mod 49)=(23, 34).**
- **To verify that this has the correct effect, we compute**

$$\begin{aligned}(23, 34) \leftrightarrow a_1 M_1 M_1^{-1} + a_2 M_2 M_2^{-1} \bmod M \\ &= [(23)(49)(34) + (34)(37)(4)] \bmod 1813 \\ &= 43350 \bmod 1813 \\ &= 1651\end{aligned}$$

  - **Note:** M_i^{-1} is the multiplicative inverse of M_1 modulo m_1 and M_2^{-1} is the multiplicative inverse of M_2 modulo m_2.
- **check that it is equal to (973+678) mod 1813=1651**

$$(A \times B) \bmod M \leftrightarrow ((a_1 \times b_1) \bmod m_1, \ \ldots \ , (a_k \times b_k) \bmod m_k)$$

- **Suppose we want to multiply 1651 (mod 1813) by 73.**
- **We multiply (23, 34) by 73 and reduce to get (23*73 mod 37, 34*73 mod 49)=(14, 32).**
- **It is easily verified that**

$$\begin{aligned}(14, 32) &\leftrightarrow [(14)(49)(34) + (32)(37)(4)] \bmod 1813 \\ &= 865 \\ &= 1651 \times 73 \bmod 1813\end{aligned}$$

- if e too small (eg e=3) can attack
  - using Chinese remainder theorem & 3 encypted messages with different modulus

- An Attacker knows C1,C2,C3 and e=3, he wants to guess message M(note: M< min(n1,n2,n3)), where
  - $C1=M^3$ mod n1
  - $C2=M^3$ mod n2
  - $C3=M^3$ mod n3
  - n1,n2,n3 are pairwise relatively prime
- Attacker can derive $C=M^3$ mod（n1*n2*n3) from C1,C2,C3 by using CRT. from M^3<n1*n2*n3, further derive M^3=C. Hence $M=C^{1/3}$
- Countermeasues: adding a unique pseudorandom bit string as padding to each instance of M to be encrypted

Efficient Operation Using Private Key

- 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 calculating “C^d mod n” directly
- only owner of private key who knows values of p & q can use this technique

- **$M = C^d \bmod n$**
- **Using CRT, $M = (V_pX_p + V_qX_q) \bmod n$**
  - **$V_p = C^d \bmod p$**
  - **$V_q = C^d \bmod q$**
  - **$X_p = q \times (q^{-1} \bmod p)$**
  - **$X_q = p \times (p^{-1} \bmod q)$**
- **X_p and X_q can be precalculated**

- If gcd(C,p)==1
  - Using Fermat's Theorem, $C^{(p-1)} \bmod p=1$. For some k1 and K2, $d=k1*(p-1)+k2$ where $d>(p-1)$, hence
  - $Vp = C^{d} \bmod p = C^{k1*(p-1)+k2} \bmod p = (C^{(p-1)})^{k1}*C^{k2} \bmod p$
    $= 1*C^{k2} \bmod p = C^{d \bmod (p-1)} \bmod p$
  - Compute Vq in two case
    - Case $Gcd(C,q)\neq 1$, $Vq=0$.
    - Case $Gcd(C,q)==1$, $Vq = C^{d} \bmod q = C^{d \bmod (q-1)} \bmod q$.
- Else //$Gcd(C,p)\neq 1$
  - //Assume $C=k*p$, then must have $gcd(C,q)==1$ because p, q are prime and $C<n==p*q$.
  - $Vp = C^{d} \bmod p = (k*p)^{d} \bmod p = 0$
  - $Vq = C^{d} \bmod q = C^{d \bmod (q-1)} \bmod q$

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 must be sufficiently large**
  - **typically guess and use probabilistic test**
- **exponents e, d are inverses, so use the <u>extended Euclid's algorithm</u> to compute the other**

Euclidean algorithm

- gcd(a, b)=gcd(b, a mod b) (if a>b)
- E.g. gcd(55, 22) = gcd(22, 55 mod 22) = gcd(22, 11) = 11

- c.f. Section 2.5 in textbook

EXTENDED Euclidean Algorithm

——Finding the Multiplicative Inverse in GF(p)

Extended Euclid(f, e) (f >e) (ed mod f =1)

input：two positive integer e, f and f>e

output：d

1. (A1,A2,A3)←(1,0,f);(B1,B2,B3)←(0,1,e);
2. if B3=0 then return no inverse; //gcd(f,e) ≠1
3. if B3=1 then return d=B2; //gcd(f,e)=1
4. Q=A3/B3；
5. (T1,T2,T3)←(A1-QB1,A2-QB2,A3-QB3);
6. (A1,A2,A3)←(B1,B2,B3);
7. (B1,B2,B3)←(T1,T2,T3);
8. goto 2

Throughout the computation, the following relationships hold:

$fT1 + eT2 = T3$，$fA1 + eA2 = A3$，$fB1 + eB2 = B3$

Table 4.4. Finding the Multiplicative 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

- **gcd(1759,550)=gcd(550,109)=gcd(109,5)=gcd(5,4)=gcd(4,1)=1**
- **1759*(-111)+550*355=1, hence 550*355=1759*111+1=1 mod 1759**

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 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 equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663) bit with LS (格筛法)
  - biggest improvement comes from improved algorithm
    - cf QS(二次筛法) to GNFS(一般数域筛法) to SNFS(特殊数域筛法) to LS(格筛法)
  - currently assume 1024-2048 bit RSA is secure
    - ensure p, q of similar size and matching other constraints

Table 9.5 Progress in RSA Factorization

Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

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)
  - attackers chooses ciphertexts & gets decrypted plaintext back
- choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis
  - E(PU,M1)*E(PU,M2)=E(PU,M1*M2)

- **We can decrypt $C = M^e$ using a CCA as follows:**

  1) Compute $X = (C \times 2^e) \bmod n$.

  2) Submit X as a chosen ciphertext and receive back $Y = X^d \bmod n$.

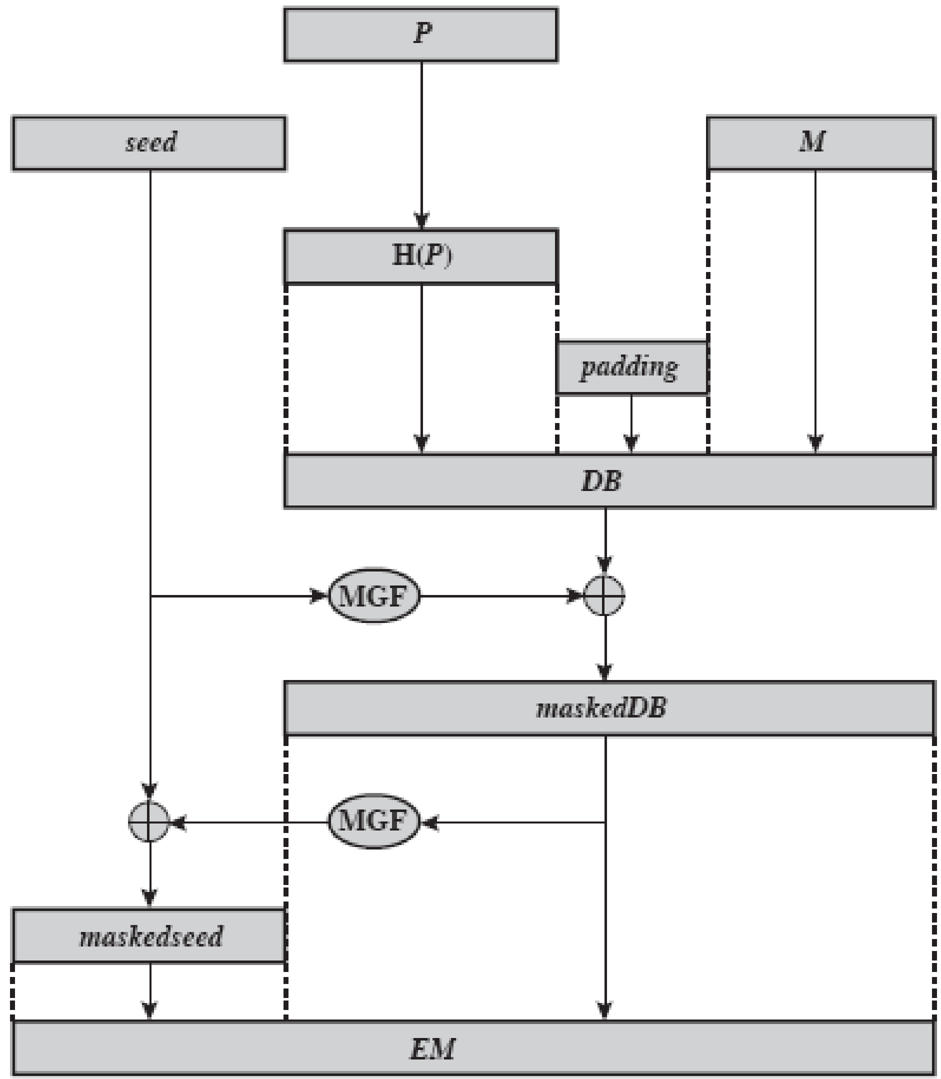
- **But now note the following:**

$X= (C \bmod n) * (2^e \text{ mode } n) = (M^e \bmod n) * (2^e \text{ mode } n) = (2M)^e \bmod n$ (Assume 2M<n)

then, $Y=2M$ and hence $M=2^{-1}*Y \bmod n$

- **Countermeasures:**
  - **counter with random pad of plaintext**

    $E(PU,P(M1))*E(PU,P(M2))=E(PU,P(M1)*P(M2)) \neq E(PU,P(M1*M2))$
  - **Pad M to EM by using Optimal Asymmetric Encryption Padding (OAEP)**
  - **Then Encrypt EM by RSA algorithm.**



P = encoding parameters
M = message to be encoded
H = hash function

DB = data block
MGF = mask generating function
EM = encoded message

Figure 9.9 Encryption Using Optimal Assymetric Encryption Padding (OAEP)

```
OpenSSL> rsautl -?
Usage: rsautl [options]
-in file        input file
-out file       output file
-inkey file     input key
-keyform arg    private key format - default PEM
-pubin          input is an RSA public
-certin         input is a certificate carrying an RSA public key
-ssl            use SSL v2 padding
-raw            use no padding
-pkcs           use PKCS#1 v1.5 padding (default)
-oaep           use PKCS#1 OAEP
-sign           sign with private key
-verify         verify with public key
-encrypt        encrypt with public key
-decrypt        decrypt with private key
-hexdump        hex dump output
-engine e       use engine e, possibly a hardware device.
-passin arg    pass phrase source
```

Summary

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

Outline

- Principles of Public-Key Cryptosystems
- The RSA Algorithm
- Distribution of Public Keys
  - Sec 14.3
- Elliptic Curve Cryptography

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 (introduced in lecture 6)

Distribution of Public Keys

- can be considered as using one of:
  - public announcement(发布)
  - publicly available directory(目录)
  - public-key authority(授权)
  - public-key certificates(证书)

Public Announcement

Figure 14.10 Uncontrolled Public-Key Distribution

Public Announcement

- users distribute public keys to recipients or broadcast to community at large
  - eg. append PGP keys to email messages or post to news groups or email list
- major weakness is forgery
  - anyone can create a key claiming to be someone else (eg. A) and broadcast it
  - until forgery is discovered, forger is able to read all encrypted messages intended for A and can authenticate message.

Publicly Available Directory

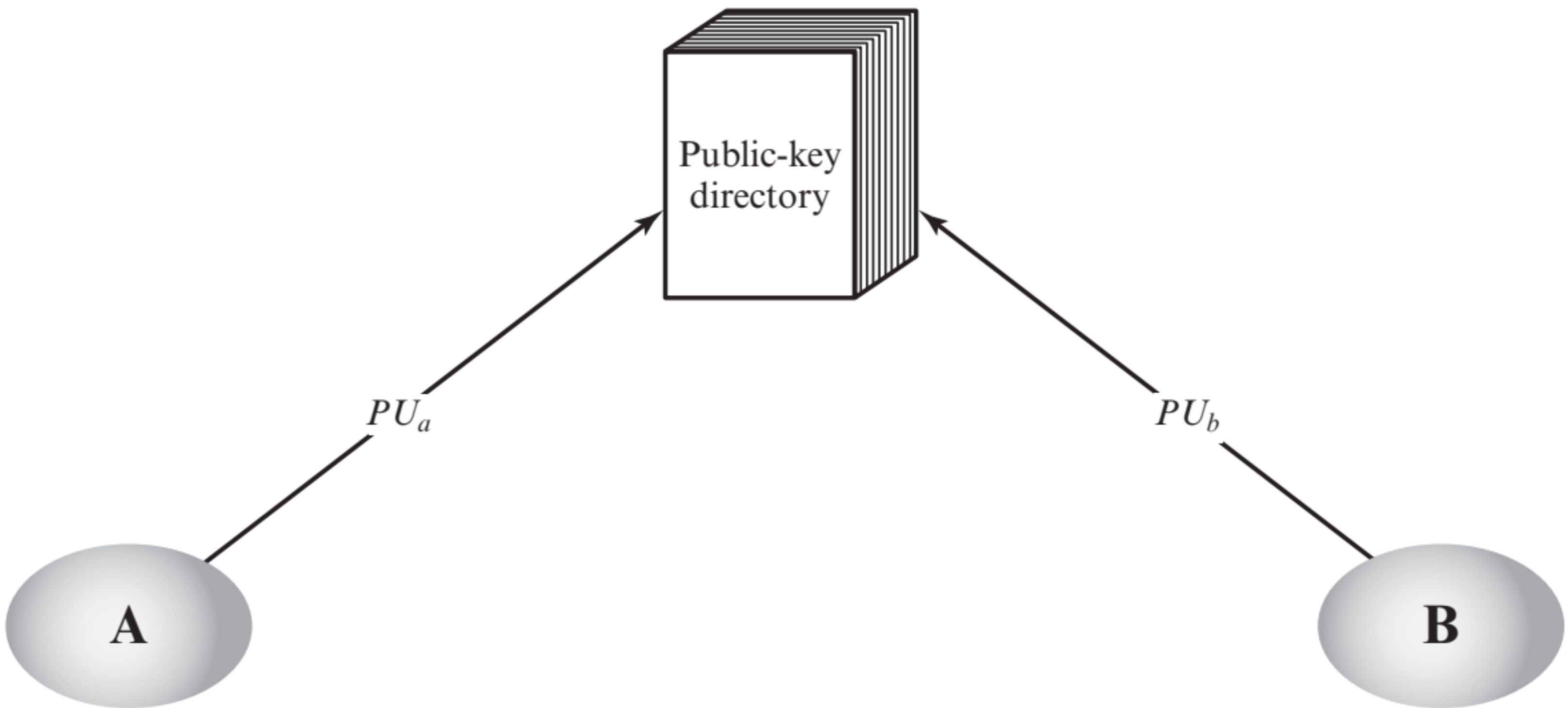
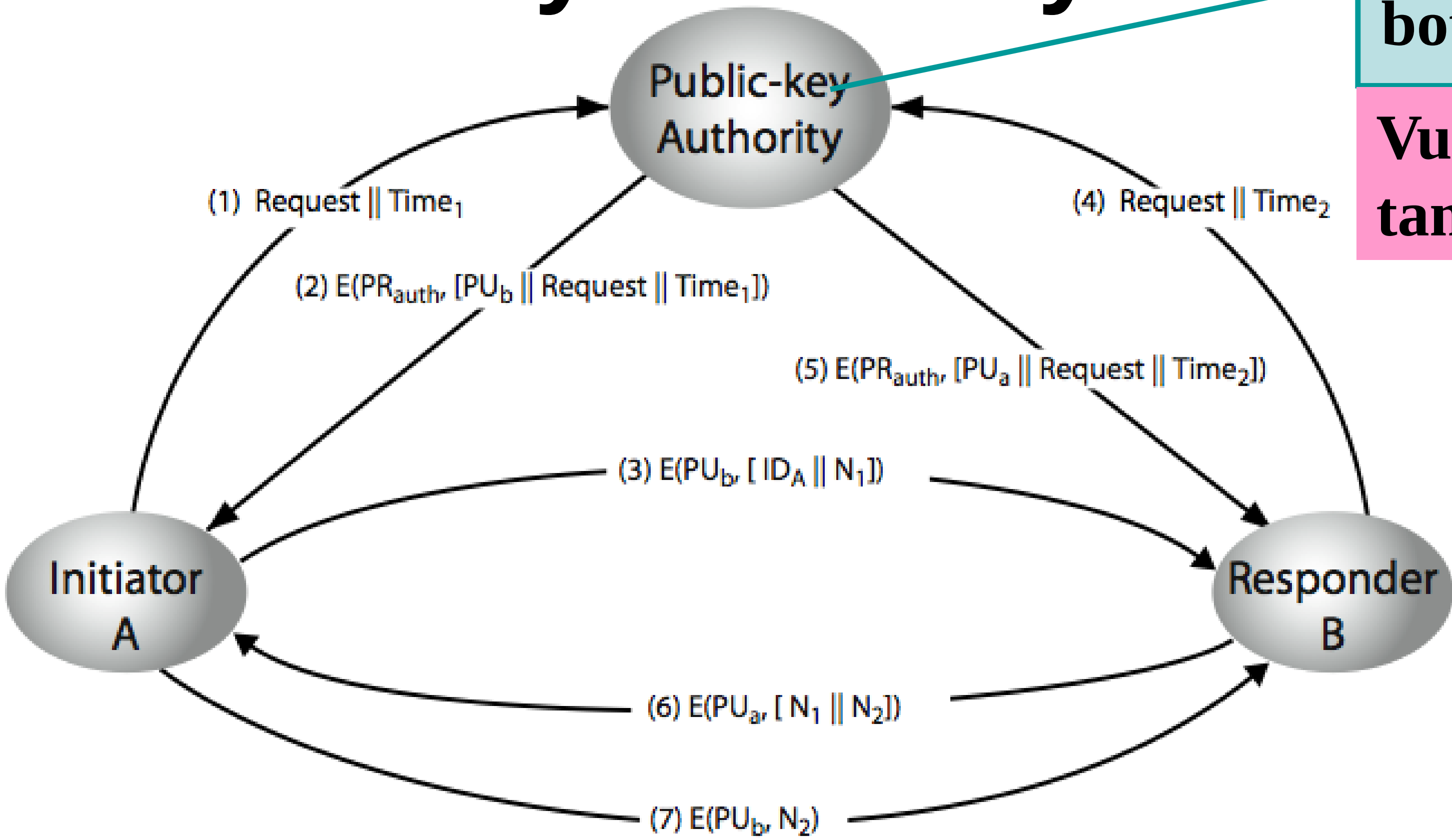


Figure 14.11 Public-Key Publication

Publicly Available Directory

- can obtain greater security by registering keys with a public directory
- Maintenance and distribution of the public directory by trusted entity or organization
- directory must be trusted with properties:
  - contains {name,public-key} entries
  - participants register securely with directory
  - participants can replace key at any time
  - directory is periodically published
  - directory can be accessed electronically
- still vulnerable to tampering or forgery

Public-Key Authority



bottleneck

Vulnerable to tampering

Assumes：

1) a central authority maintains a dynamic directory of public keys of all participants.
2) each participant reliably knows public key for authority
3) only the authority knowing the corresponding private key

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 Certificate Authority (CA)

Public-Key Certificates

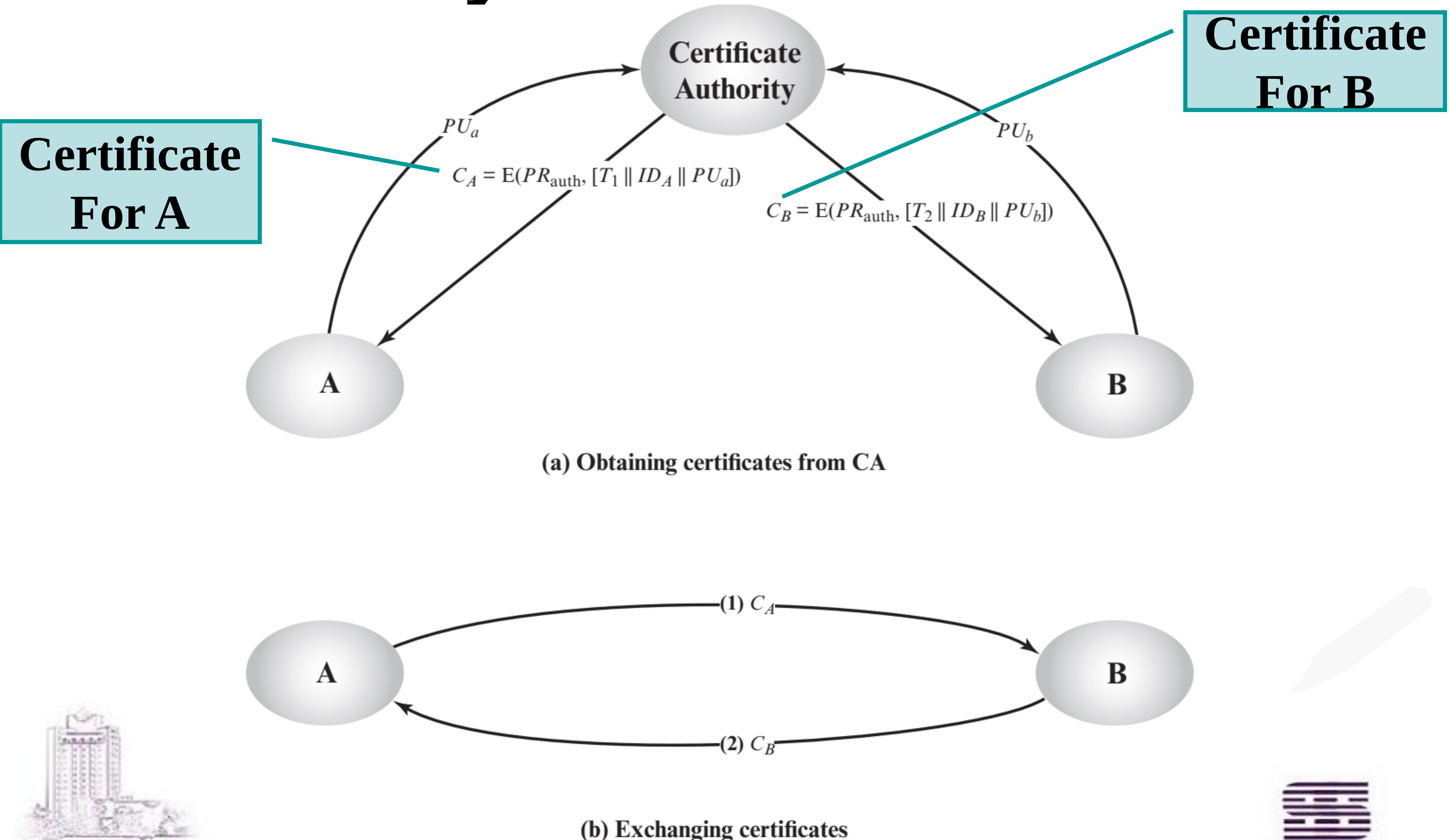
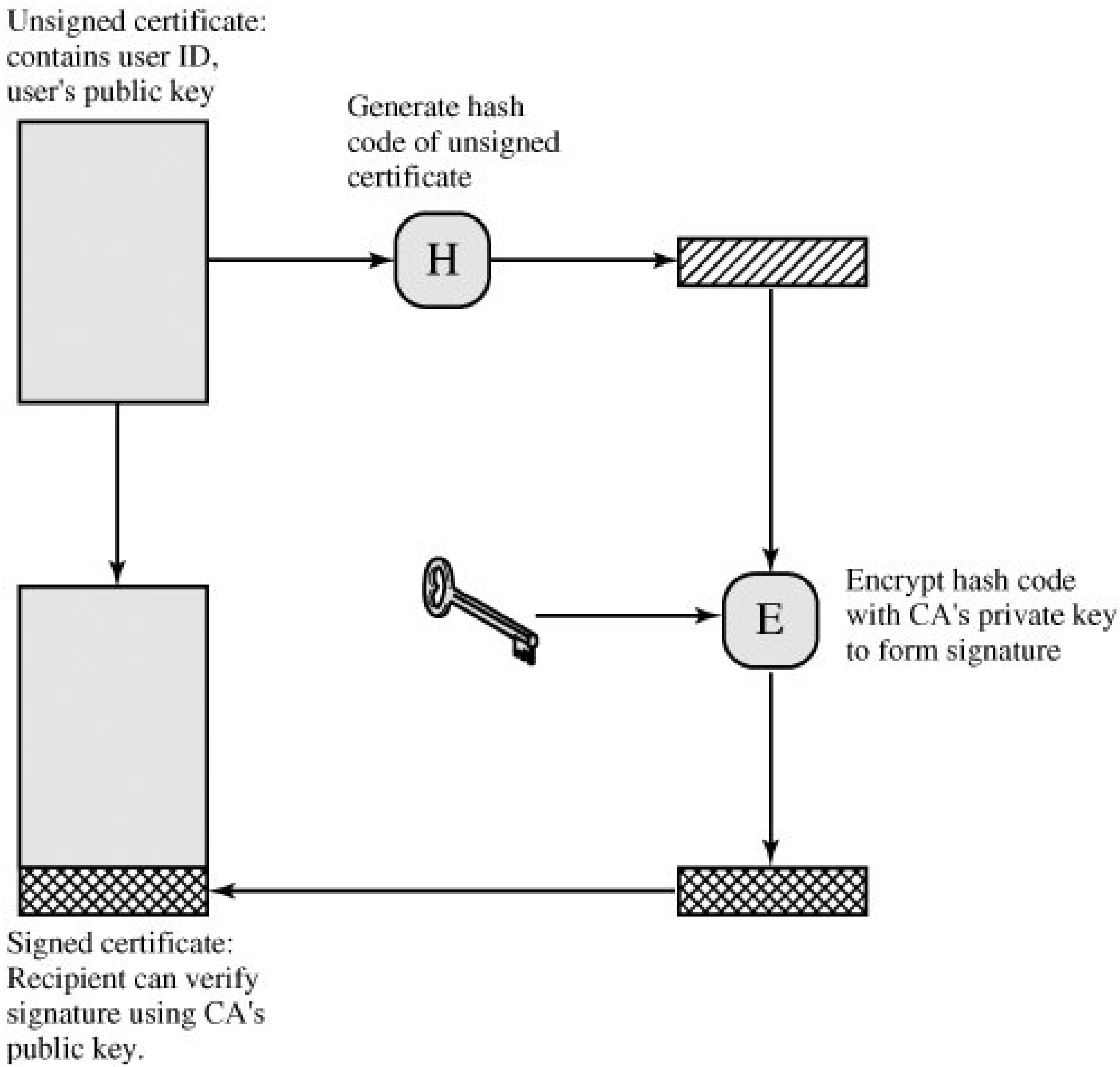
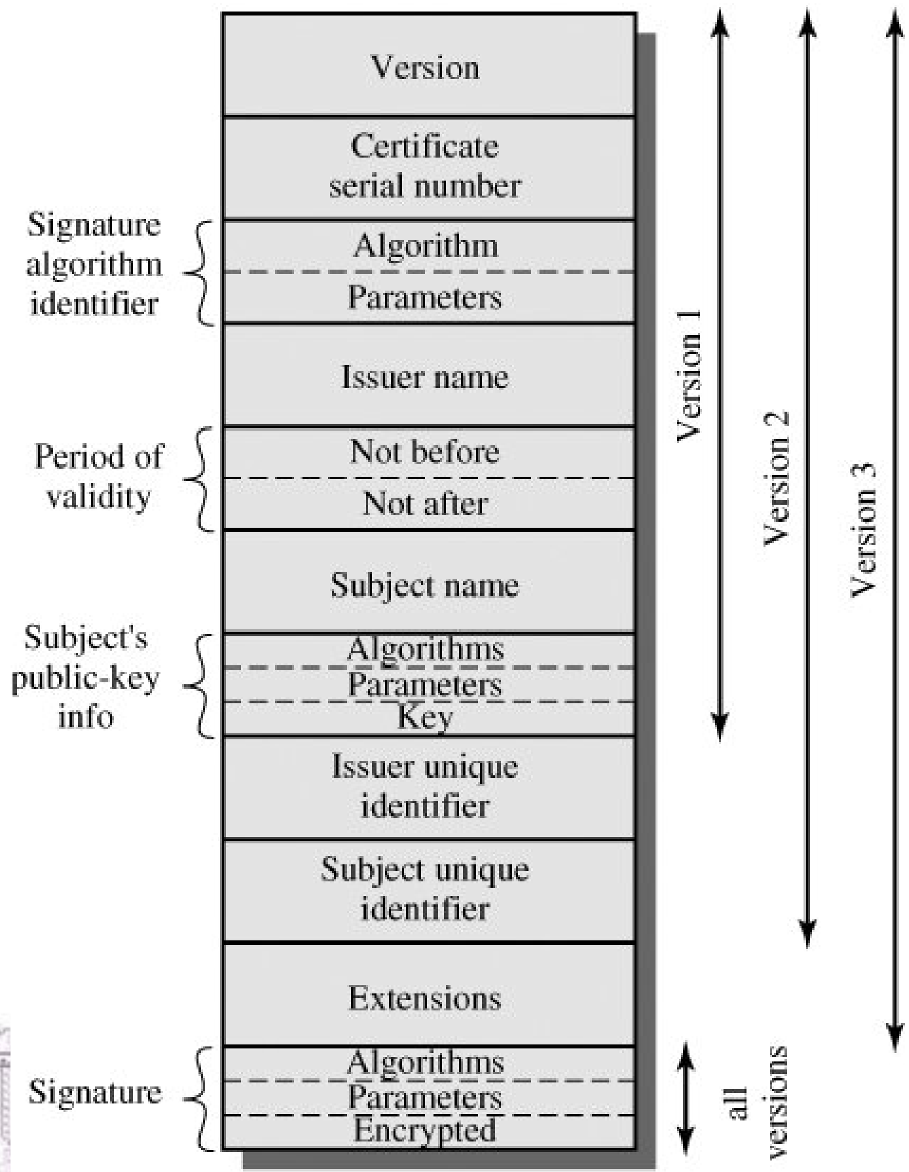


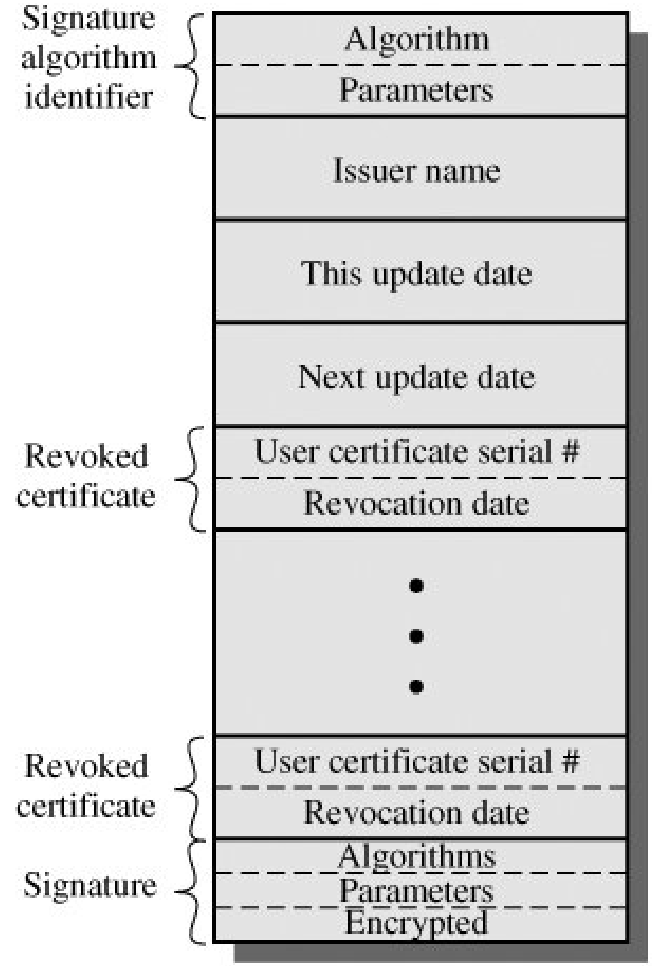
Figure 14.13 Exchange of Public-Key Certificates

Figure 14.3. Public-Key Certificate Use





(a) X.509 certificate



(b) Certificate revocation list

Requirements on Public-Key Certificates

- **Any participant can read a certificate to determine the name and public key of the certificate's owner.**
- **Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.**
- **Only the certificate authority can create and update certificates.**

Outline

- Principles of Public-Key Cryptosystems
- The RSA Algorithm
- Distribution of Public Keys
- Elliptic Curve Cryptography
  – D-H ---〉 ElGamal Cryptography --〉ECC

ElGamal Cryptography

- public-key cryptosystem related to D-H
- so uses exponentiation in a finite (Galois)
- with security based difficulty of computing discrete logarithms, as in D-H
- each user (eg. A) generates their key
  - chooses a secret key (number): $1 < x_A < q-1$
  - compute their public key: $y_A = a^{x_A} \bmod q$
    - Difficult to get x_A from Y_A (discrete logarithms problem)

ElGamal Message Exchange

- Bob encrypt a message to send to A computing
  - represent message M in range 0 <= M <= q-1
    - longer messages must be sent as blocks
  - chose random integer `k` with `1 <= k <= q-1`
  - compute one-time key $K = y_A^{k}$ mod q
  - encrypt M as a pair of integers (C_1,C_2) where
    - $C_1 = a^K \bmod q$; $C_2 = KM \bmod q$
    - Difficult to get k from C_1 (discrete logarithms problem)
- A then recovers message by
  - recovering key K as $K = C_1^{x_A} \bmod q$
  - computing M as $M = C_2 K^{-1} \bmod q$
- a unique k must be used each time
  - otherwise result is insecure

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

Table 8.3 Powers of Integers, Modulo 19

ElGamal Example

- use field GF(19), q=19 and a=10
- Alice computes her key:
  - A chooses $x_A=5$ & computes $y_A=10^5 \bmod 19 = 3$
- Bob send message m=17 as (11,5) by
  - chosing random k=6
  - computing $K = y_A^k \bmod q = 3^6 \bmod 19 = 7$
  - computing $C_1 = a^k \bmod q = 10^6 \bmod 19 = 11$;
    $C_2 = KM \bmod q = 7.17 \bmod 19 = 5$
- Alice recovers original message by computing:
  - recover $K = C_1^{x_A} \bmod q = 11^5 \bmod 19 = 7$
  - compute inverse $K^{-1} = 7^{-1} = 11$
  - recover $M = C_2\, K^{-1} \bmod q = 5.11 \bmod 19 = 17$

Elliptic Curve Cryptography

- Most of the products and standards
  - RSA signature: ANSI X9.31, PKCS#1
  - RSA encryption: ANSI X9.42, PKCS#1
- key length for secure RSA use has increased over recent years, hence imposes a significant load on applications using RSA.
- an alternative is to use elliptic curves (ECC)
- ECC offers same security with smaller bit sizes
- ECC has been Used in some standards
  - IEEE P1863a, ANSI X9.62, ANSI X9.63

Real Elliptic Curves

- an elliptic curve is defined by an equation in two variables x & y, with coefficients
- consider a cubic elliptic curve
  - Form: $y^2 = x^3 + ax + b$, where x,y,a,b are all real numbers
  - also define zero point O
  - E(a,b) is set of points satisfying above equations together with zero point O
  - E(a, b) defines a group if $4a^3+27b^2 \neq 0$
- have addition rules for elliptic curve in geometrical terms
  - If three points on an elliptic curve lie on a straight line, their sum is O
  - geometrically sum of P+Q is reflection of the intersection R

c.f. Sec2.8.1

- <u>order(阶)</u> of a (mod n)
  - least positive exponent m satisfying $a^m=1$ (mod n)
  - the length of the period generated by a
- <u>primitive root</u> of n： （本原根）
  - is a when order of a (mod n) is ø(n)
  - is a when $a, a^2, ..., a^{ø(n)}$ are distinct (mod n) and are all relatively prime to n
  - Especially, for a prime number p, if a is a primitive root of p, then $a, a^2, ..., a^{p-1}$ are distinct (mod p)
  - Not all integers have primitive roots
  - In fact, the only integers with primitive roots are those of the form 2, 4, p^a, and $2p^a$, where p is any odd prime and a is a positive integer

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

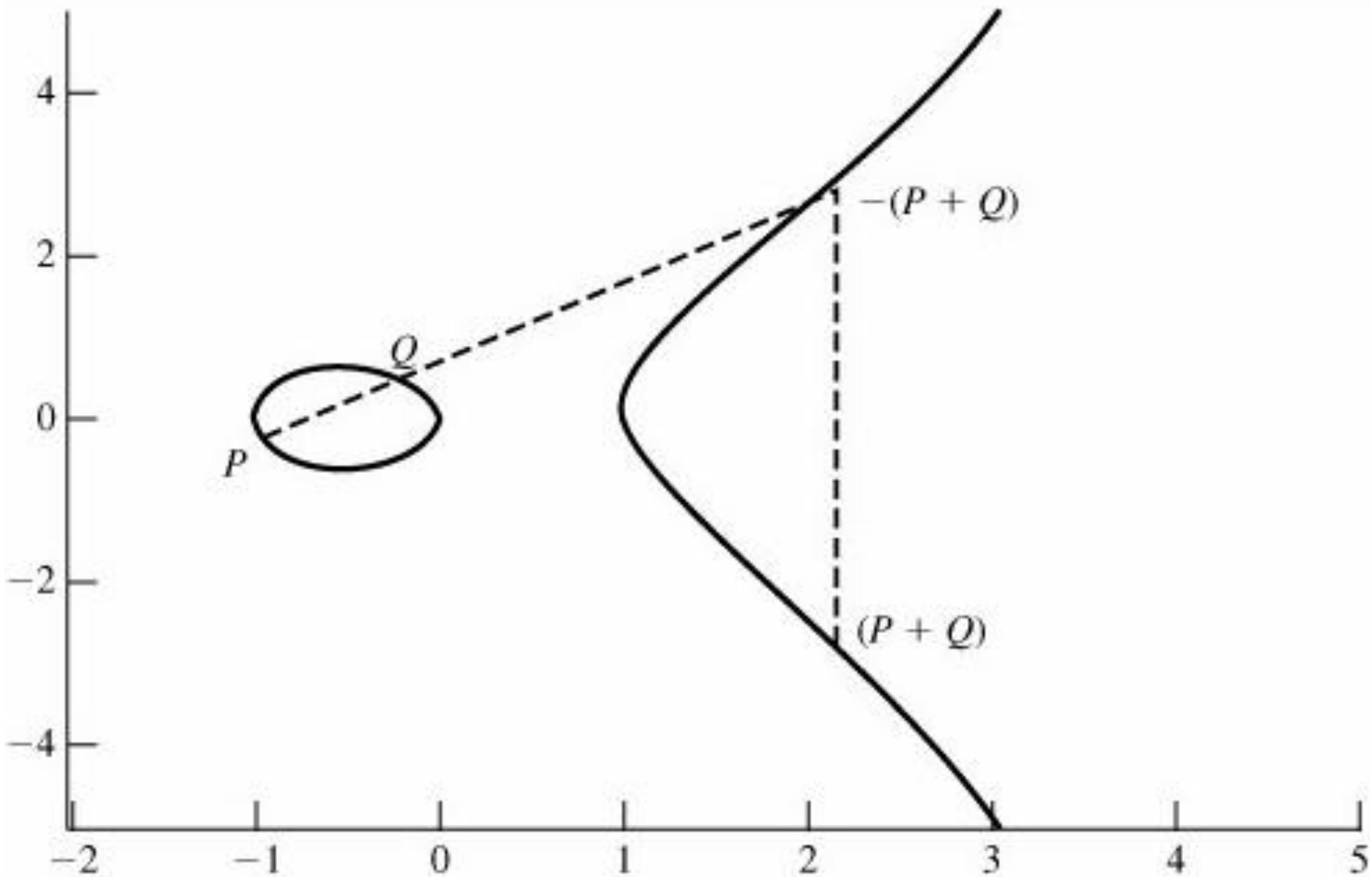
Table 8.3 Powers of Integers, Modulo 19

- **<u>Group</u> is a set of elements with a binary operation**
  - **Elements**
  - **Operation**

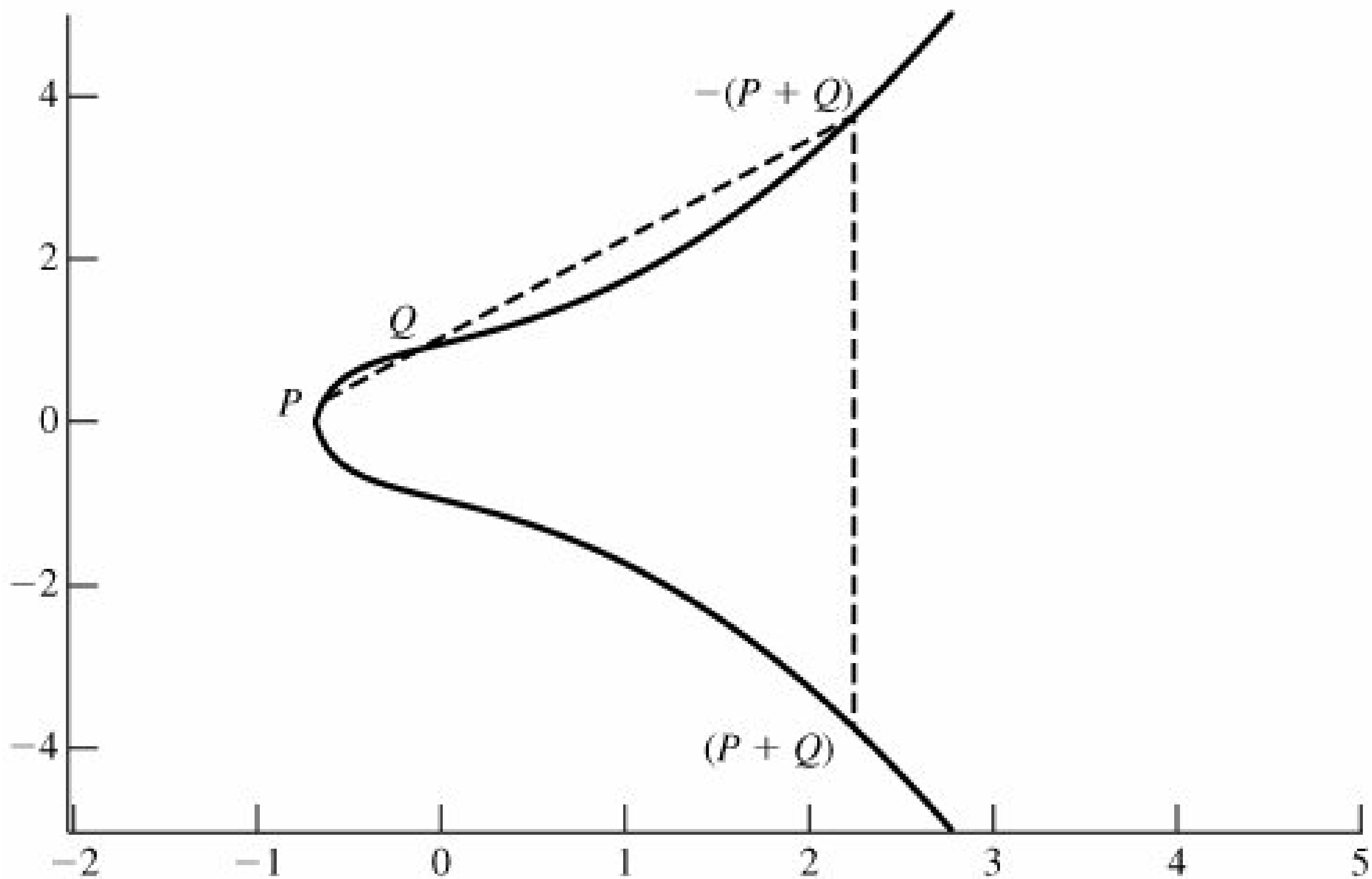
Abelian group

(A1) Closure under addition:	If a and b belong to S, then $a + b$ is also in S
(A2) Associativity of addition:	$a + (b + c) = (a + b) + c$ for all a, b, c in S
(A3) Additive identity:	There is an element 0 in R such that $a + 0 = 0 + a = a$ for all a in S
(A4) Additive inverse:	For each a in S there is an element $-a$ in S such that $a + (-a) = (-a) + a = 0$
(A5) Commutativity of addition:	$a + b = b + a$ for all a, b in S

Real Elliptic Curve Example

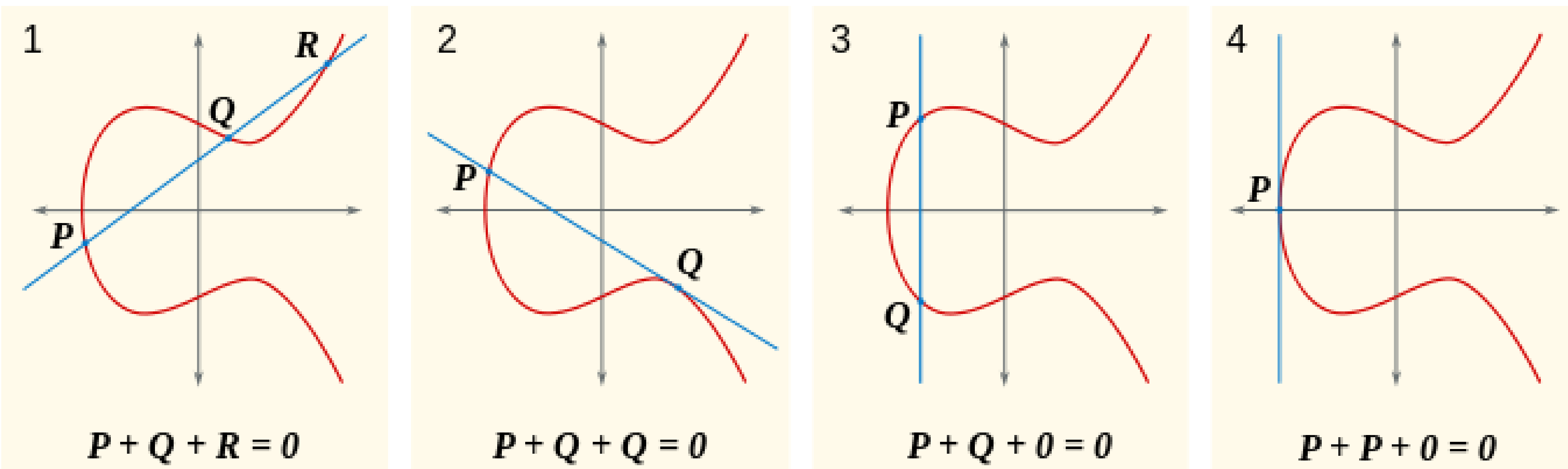


(a) $y^2 = x^3 - x$



(b) $y^2 = x^3 + x + 1$

Addition Example



Finite Elliptic Curves

- Elliptic curve cryptography uses curves whose variables & coefficients are finite
- $E_q(a,b)$ have two families commonly used:
  - prime curves $E_p(a,b)$ defined over Z_p
    - use integers modulo a prime
    - best in software
  - binary curves $E_{2^n}(a,b)$ defined over $GF(2^n)$
    - use polynomials with binary coefficients
    - best in hardware

Table 4.3. Arithmetic in GF(7)

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

(a) Addition modulo 7

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

(b) Multiplication modulo 7

w	$-w$	w^{-1}
0	0	—
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

(c) Additive and multiplicative inverses modulo 7

Table 4.5. Arithmetic in GF(2^3)

	+	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
000	0	0	1	2	3	4	5	6	7
001	1	1	0	3	2	5	4	7	6
010	2	2	3	0	1	6	7	4	5
011	3	3	2	1	0	7	6	5	4
100	4	4	5	6	7	0	1	2	3
101	5	5	4	7	6	1	0	3	2
110	6	6	7	4	5	2	3	0	1
111	7	7	6	5	4	3	2	1	0

(a) Addition

	×	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
000	0	0	0	0	0	0	0	0	0
001	1	0	1	2	3	4	5	6	7
010	2	0	2	4	6	3	1	7	5
011	3	0	3	6	5	7	4	1	2
100	4	0	4	3	7	6	2	5	1
101	5	0	5	1	4	2	7	3	6
110	6	0	6	7	1	5	3	2	4
111	7	0	7	5	2	1	6	4	3

(b) Multiplication

w	$-w$	w^{-1}
0	0	—
1	1	1
2	2	5
3	3	6
4	4	7
5	5	2
6	6	3
7	7	4

(c) Additive and multiplicative inverses

Elliptic Curve Cryptography

- ECC addition is analog of modulo multiply
- ECC repeated addition is analog of modulo exponentiation
- need "hard" problem equiv to discrete log
  - `Q=kP`, where Q,P belong to a prime curve
  - is "easy" to compute Q given k,P
  - but "hard" to find k given Q,P
  - known as the elliptic curve logarithm problem
- Certicom example: $E_{23}(9,17)$

Certicom Example

- Consider the group $E_{23}(9, 17)$ defined by the equation $y^2 \bmod 23 = (x^3 + 9x + 17) \bmod 23$.
- What is the discrete logarithm k of Q = (4, 5) to the base P = (16, 5)?
- The brute-force method is to compute multiples of P until Q is found.
  - Thus, P = (16, 5); 2P = (20, 20); 3P = (14, 14); 4P = (19, 20); 5P = (13, 10); 6P = (7, 3); 7P = (8, 7); 8P (12, 17); 9P = (4, 5).
  - Because 9P = (4, 5) = Q, the discrete logarithm Q = (4, 5) to the base P = (16, 5) is k = 9.
- In a real application, k would be so large as to make the brute-force approach infeasible.

ECC Diffie-Hellman

- can do key exchange analogous to D-H
- users select a suitable curve $E_q(a,b)$
  - q is a large integer, which is either a prime number p or an integer of the form 2^m
- select base point $G=(x_1,y_1)$
  - with large order n s.t. $nG=O$
- A & B select private keys $n_A<n$, $n_B<n$
- compute public keys: $P_A=n_A G$, $P_B=n_B G$
- compute shared key: $K=n_A P_B$, $K=n_B P_A$
  - same since $K=n_A n_B G$
- attacker would need to find k, hard

ECC Encryption/Decryption

- several alternatives, will consider simplest
- must first encode any message M as a point on the elliptic curve P_m
- select suitable curve & point G as in D-H
- User A chooses private key $n_A<n$ and computes public key $P_A=n_AG$
- User B encrypt P_m : $C_m=\{kG,\ P_m+kP_A\}$, k random
- User A decrypt C_m compute:

$$P_m+kP_A-n_A(kG) = P_m+k(n_AG)-n_A(kG) = P_m$$

```
OpenSSL> ecparam -?
unknown option -?
ecparam [options] <infile >outfile
where options are
 -inform arg        input format - default PEM (DER or PEM)
 -outform arg       output format - default PEM
 -in  arg           input file  - default stdin
 -out arg           output file - default stdout
 -noout             do not print the ec parameter
 -text              print the ec parameters in text form
 -check             validate the ec parameters
 -C                 print a 'C' function creating the parameters
 -name arg          use the ec parameters with 'short name' name
 -list_curves       prints a list of all currently available curve 'short names'
 -conv_form arg     specifies the point conversion form
                    possible values: compressed
                                     uncompressed (default)
                                     hybrid
 -param_enc arg     specifies the way the ec parameters are encoded
                    in the asn1 der encoding
                    possible values: named_curve (default)
                                     explicit
 -no_seed           if 'explicit' parameters are choosen do not use the seed
 -genkey            generate ec key
 -rand file         files to use for random number input
 -engine e          use engine e, possibly a hardware device
```

```
C:\OpenSSL\bin>openssl ecparam -list curves
  secp112r1 : SECG/WTLS curve over a 112 bit prime field
  secp112r2 : SECG curve over a 112 bit prime field
  secp128r1 : SECG curve over a 128 bit prime field
  secp128r2 : SECG curve over a 128 bit prime field
  secp160k1 : SECG curve over a 160 bit prime field
  secp160r1 : SECG curve over a 160 bit prime field
  secp160r2 : SECG/WTLS curve over a 160 bit prime field
  secp192k1 : SECG curve over a 192 bit prime field
  secp224k1 : SECG curve over a 224 bit prime field
  secp224r1 : NIST/SECG curve over a 224 bit prime field
  secp256k1 : SECG curve over a 256 bit prime field
  secp384r1 : NIST/SECG curve over a 384 bit prime field
  secp521r1 : NIST/SECG curve over a 521 bit prime field
  prime192v1: NIST/X9.62/SECG curve over a 192 bit prime field
  prime192v2: X9.62 curve over a 192 bit prime field
  prime192v3: X9.62 curve over a 192 bit prime field
  prime239v1: X9.62 curve over a 239 bit prime field
  prime239v2: X9.62 curve over a 239 bit prime field
  prime239v3: X9.62 curve over a 239 bit prime field
  prime256v1: X9.62/SECG curve over a 256 bit prime field
  sect113r1 : SECG curve over a 113 bit binary field
  sect113r2 : SECG curve over a 113 bit binary field
  sect131r1 : SECG/WTLS curve over a 131 bit binary field
```

```
C:\OpenSSL\bin>openssl ecparam -name secp112r1 -genkey -text
ASN1 OID: secp112r1
-----BEGIN EC PARAMETERS-----
BgUrgQQABg==
-----END EC PARAMETERS-----
Loading 'screen' into random state - done
-----BEGIN EC PRIVATE KEY-----
MD4CAQEEDsEtQ2v2XEqJsjtv4okFoAcGBSuBBAAGoSADHgAEMeXPnGcX7JSUhzNe
5OC3eItXd3NE7mh2ZhAmrg==
-----END EC PRIVATE KEY-----

C:\OpenSSL\bin>openssl ecparam -genkey -name secp160r1 -out eckey.pem -text
Loading 'screen' into random state - done
```

ECC Security

- relies on elliptic curve logarithm problem
- fastest method is “Pollard rho method”
- compared to factoring, can use much smaller key sizes than with RSA etc
- for equivalent key lengths computations are roughly equivalent
- hence for similar security ECC offers significant computational advantages

Comparable Key Sizes for Equivalent Security

Symmetric scheme (key size in bits)	ECC-based scheme (size of *n* in bits)	RSA/DSA (modulus size in bits)
56	**112**	**512**
80	**160**	**1024**
112	**224**	**2048**
128	**256**	**3072**
192	**384**	**7680**
256	**512**	**15360**

Review: Requirements for Public-Key Cryptography

①computationally easy to generate a pair (Pu_b and PR_b).

②computationally easy to compute cipher-text for a sender A knowing the public key and the plain-text M : $C = E(PU_b, M)$

③computationally easy to recover the original message for the receiver B knowing cipher-text and private key : $M = D(PR_b, C)$

④computationally infeasible for an adversary, knowing the public key PU_b, to determine <u>private key PR_b</u>.

⑤computationally infeasible for an adversary, knowing the public key Pu_b and a cipher-text C, to <u>recover M</u>.

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

Key Terms

- chosen ciphertext attack (CCA)
- digital signature
- key exchange
- one-way function
- optimal asymmetric encryption padding (OAEP)
- Diffie–Hellman key exchange
- RSA
- private key & public key
- public-key cryptography
- public-key cryptosystems
- public-key encryption
- trap-door one-way function
- ElGamal cryptography
- elliptic curve cryptography (ECC)

Review Questions

- 9.1 What is a public key certificate?
- 9.2 What are the roles of the public and private key?
- 9.3 What are three broad categories of applications of public-key cryptosystems?
- 9.4 What requirements must a public-key cryptosystems fulfill to be a secure algorithm?
- What is one-way function?
- What is trap-door one-way function?
- Problems 9.11, 9.15, 9.18

Review Questions

- **10.1 Briefly explain Diffie–Hellman key exchange.**
- **10.2 What is an elliptic curve?**
- **10.3 What is the zero point of an elliptic curve?**
- **10.4 What is the sum of three points on an elliptic curve that lie on a straight line?**

- **14.6 List four general categories of schemes for the distribution of public keys.**
- **14.8 What is a public-key certificate?**

Symmetric Enc vs. public-key Enc

- Symmetric encryption
- One key
- Secure distribution of secret key
- Rely on complex substitution and permutation
- Slow
- For data confidentiality applications

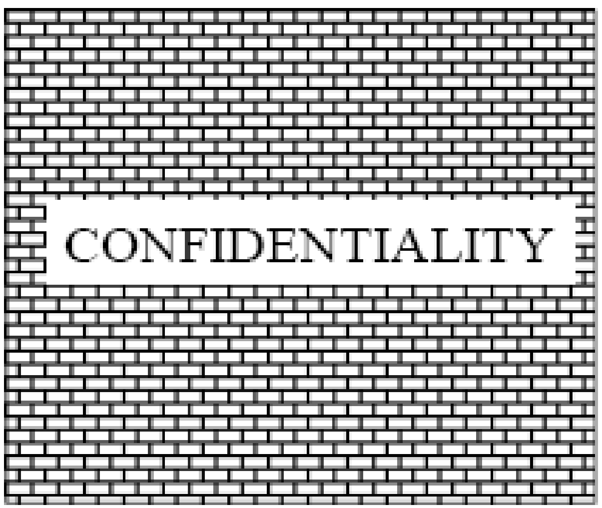
- Public-key encryption
- Two key
- Authentication of Public key
- Rely on mathematical difficult problems
- much slower
- For confidentiality (protect secret keys) and authentication applications

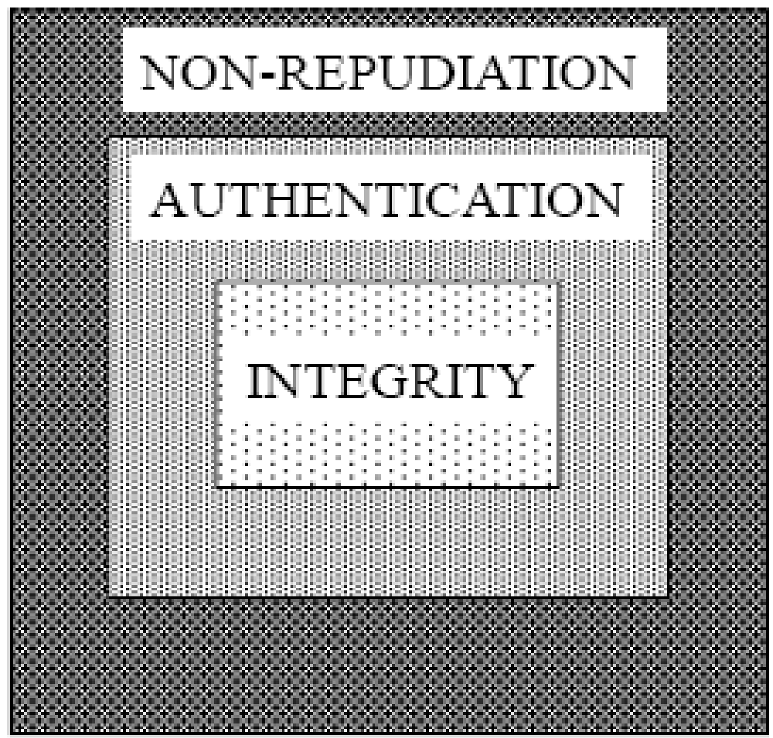
Encryption Summary

- Provide data confidentiality: assure authorized use can read message
- Other remaining problems:
  - – Message source? (message authentication)
  - – Sender deny? (Non-Repudiation)
  - – Message is modified? (Data Integrity)
  - – one is the claimed one? (ID authentication)

Next Lecture

- **For Message authentication and Data Integrity**





Thanks!