Public Key Cryptography

Prof. A. A. Daptardar
HIT, Nidasoshi

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 & 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 private key cryptography

Why Public-Key Cryptography?

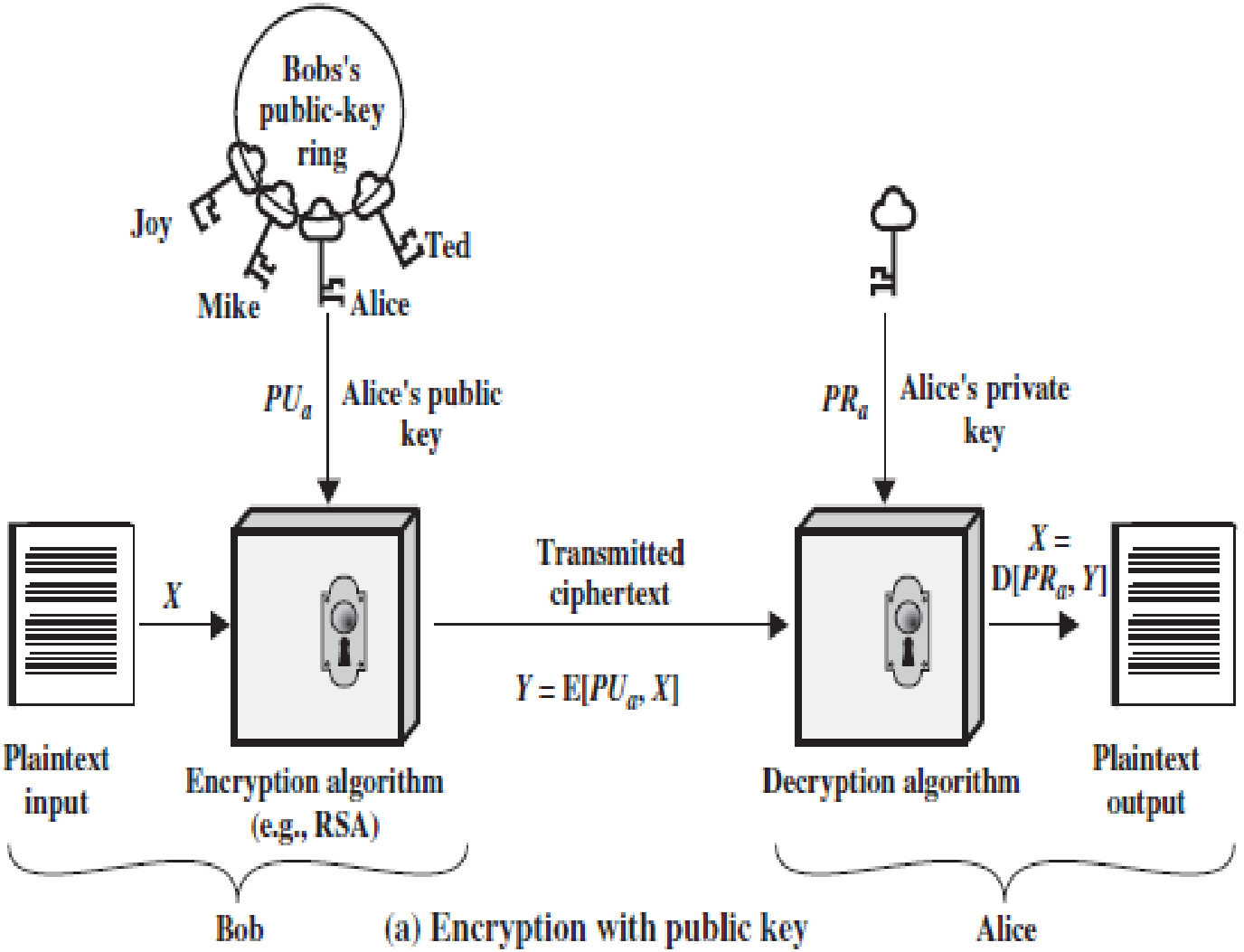
- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman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

Public-Key Cryptography

- **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 **public-key encryption scheme has six ingredients :**
  - **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
  - **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
  - **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
  - **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
  - **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

Public-Key Cryptography



- The essential steps are the following:
  - Each user generates a pair of keys to be used for the encryption and decryption of messages.
  - Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. Each user maintains a collection of public keys obtained from others.
  - If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
  - When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

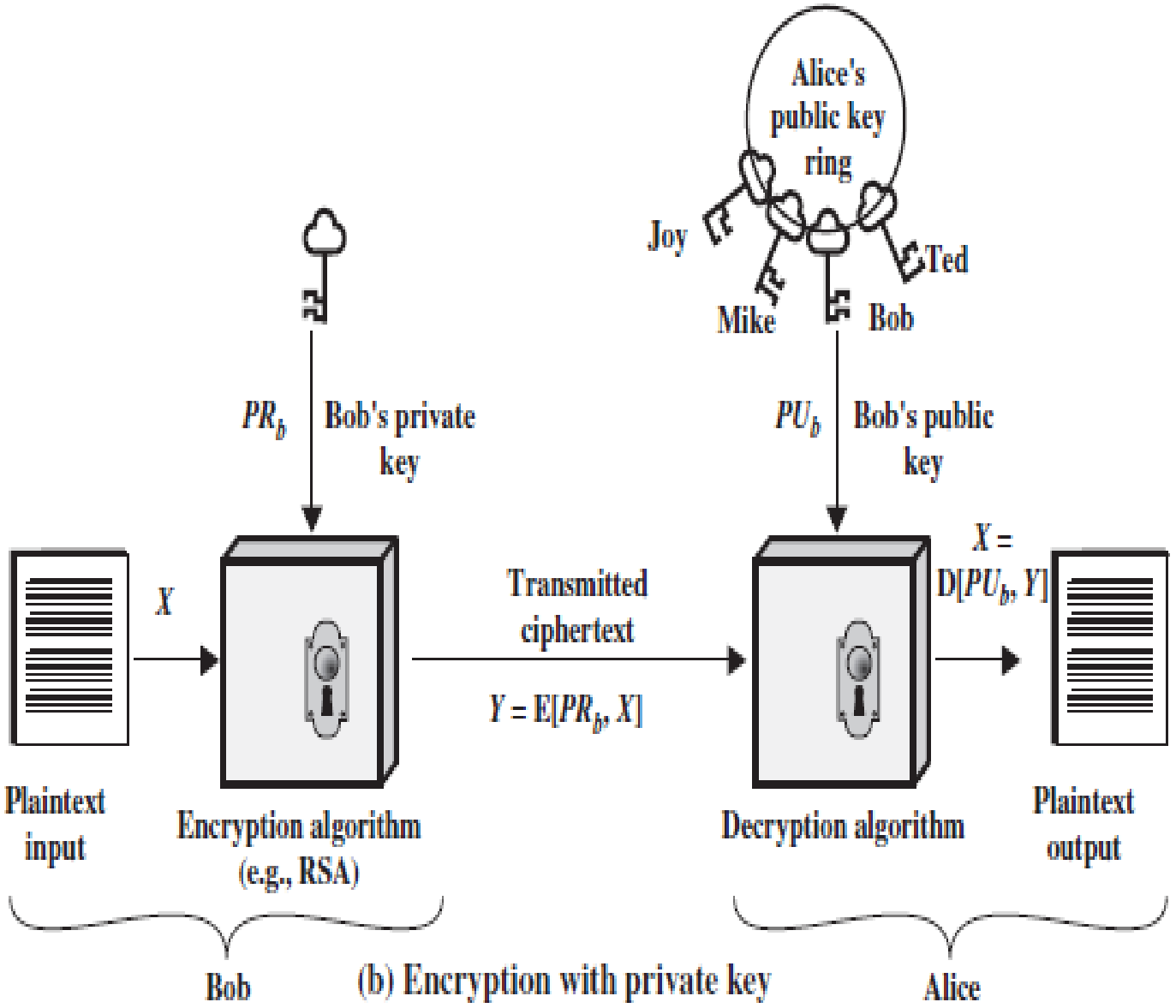
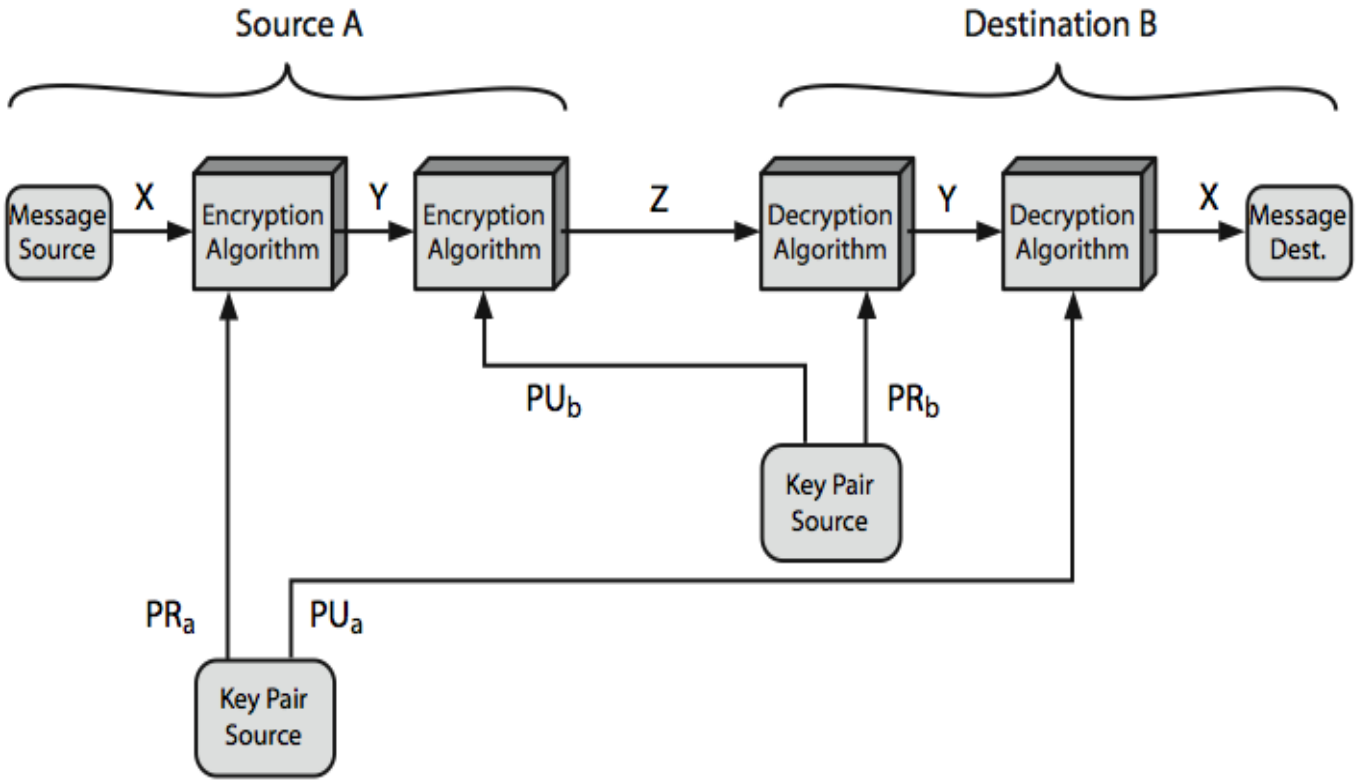


Figure 9.1 Public-Key Cryptography

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



Public-Key Applications

- can classify uses into 3 categories:
- **encryption/decryption** (provide secrecy) : The sender encrypts a message with the recipient's public key.
- **digital signatures** (provide authentication) : The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the Message.
- **key exchange** (of session keys) : Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.
- some algorithms are suitable for all uses, others are specific to one

Table 9.3 Applications for Public-Key Cryptosystems

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

Requirements for Public-Key Cryptography

- It is computationally easy for a party B to generate a pair (public key *PUb,* private key *PRb).*
- It is computationally easy for a sender A, knowing the public key and the message to be encrypted, *M, to generate the corresponding ciphertext:*

$$C = E(PU_b, M)$$

- It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

  $M = D(PR_b, C) = D[PR_b, E(PU_b, M)]$

- It is computationally infeasible for an adversary, knowing the public key, PU_b, to determine the private key, PR_b.
- It is computationally infeasible for an adversary, knowing the public key, PU_b, and a ciphertext, *C, to recover the original message, M.*

- The two keys can be applied in either order:

$$M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$$

One-way Function

- It is one that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy, whereas the calculation of the inverse is infeasible:

$Y = f(X)$	*easy*
$X = f^{-1}(Y)$	*infeasible*

- Easy is defined to mean a problem that can be solved in polynomial time as a function of input length.
- Thus, if the length of the input is n bits, then the time to compute the function is proportional to n^a, where a is a fixed constant.
- Such algorithms are said to belong to the class **P.**
- In general, we can say a problem is infeasible if the effort to solve it grows faster than polynomial time as a function of input size.

Trap-door One-way Function

- A trap-door one-way function, which is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known.
- With the additional information the inverse can be calculated in polynomial time.

- We can summarize as follows: A trapdoor one-way function is a family of invertible functions f_k, *such that*
- $Y = f_k(X)$ *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*

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 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)
- The RSA scheme is a cipher in which the plaintext and ciphertext are integers between 0 to n-1 for some n.
- A typical size for n is 1024 bits or 309 decimal digits.

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

Key Generation by Alice

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p - 1)(q - 1)$	
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 by Bob with Alice's Public Key

Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Public Key

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

Figure 9.5 The RSA Algorithm

RSA Use

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

Why RSA Works

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

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

RSA Example - Key Setup

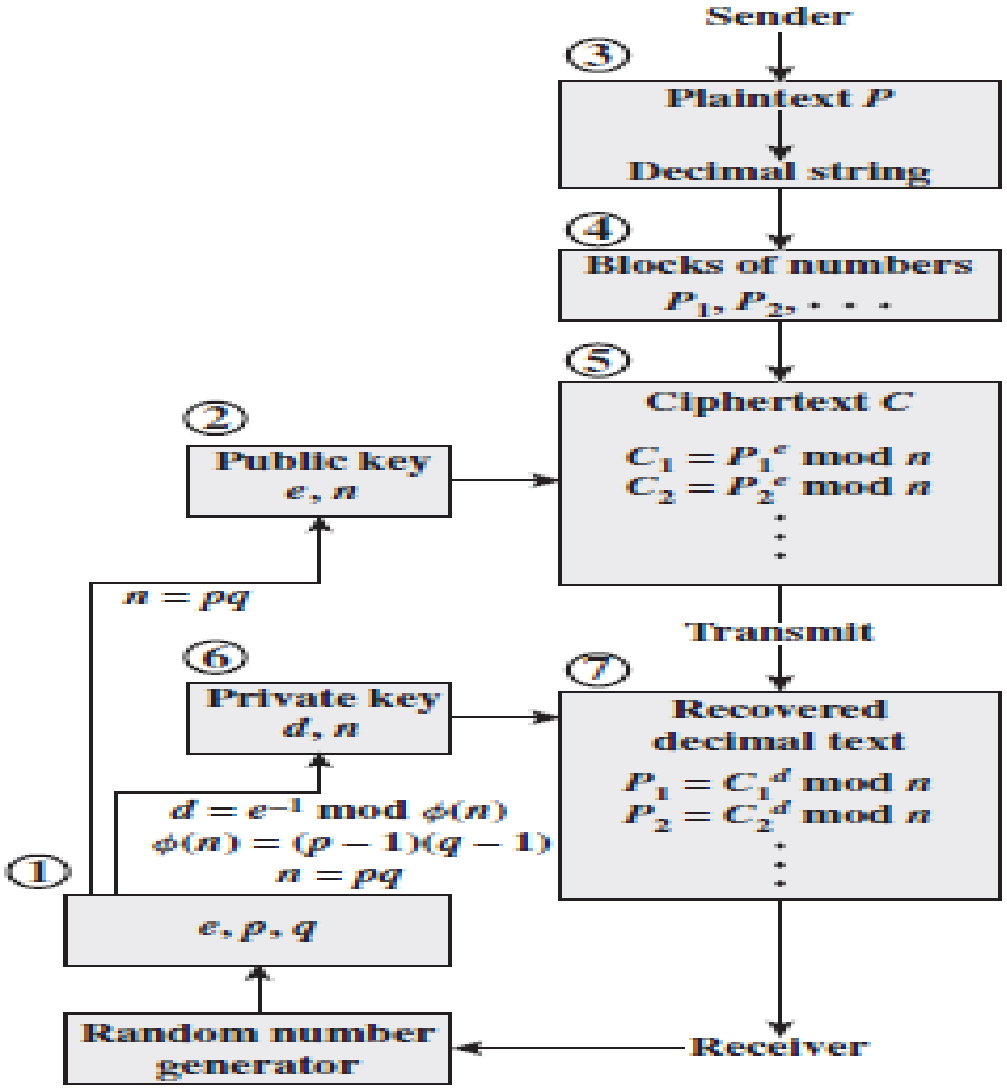
1. Select primes: $p=17$ & $q=11$
2. Compute $n = pq = 17 \times 11 = 187$
3. Compute $\varnothing(n)=(p-1)(q-1)=16 \times 10=160$
4. Select e: $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $de=1 \bmod 160$ and $d < 160$ Value is $d=23$ since $23 \times 7=161=10 \times 160+1$
6. Publish public key $PU=\{7,187\}$
7. Keep secret private key $PR=\{23,187\}$

RSA Example - En/Decryption

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

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

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



(a) General approach

Figure 9.7 RSA Processing of Multiple Blocks

Computational Aspects

- Two issues to consider:
  - Encryption/Decryption
  - Key Generation

Exponentiation in Modular Arithmetic

- Both encryption and decryption in RSA involve raising an integer to an integer power, mod n.
- If the exponentiation is done over the integers and then reduced modulo n, the intermediate values would be gargantuan.
- Another consideration is the efficiency of exponentiation, because with RSA, we are dealing with potentially large exponents.
- To see how efficiency might be increased, consider that we wish to compute x^{16}.

$$x^{16} = x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x$$

- We can achieve the same final result with only four multiplications if we repeatedly take the square of each partial result, successively forming (x^2, x^4, x^8, x^{16}).
- More generally, suppose we wish to find the value $a^b \bmod n$ *with a, b, and m* positive integers.
- If we express *b as a binary number* $b_k b_{k-1} \ldots b_0$, *then we have*

$$b = \sum_{b_i \neq 0} 2^i$$

Exponentiation

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

Figure 9.8 Algorithm for Computing $a^b \bmod n$

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 & 3 messages with different modulii
- 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.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 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 $\varnothing(n)$ and then d
  - determine $\varnothing(n)$ 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 GHFS to LS
  - currently assume 1024-2048 bit RSA is secure
    - ensure p, q of similar size and matching other constraints

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
can counter with random pad of plaintext
or use Optimal Asymmetric Encryption Padding (OASP)

Diffie Hellman Key Exchange

Diffie-Hellman Key Exchange

- The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent symmetric encryption of messages.
- The algorithm itself is limited to the exchange of secret values.
- The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms.

Discrete Logarithm

- A primitive root of a prime number p is one whose powers modulo p generate all the integers from 1 to p - 1.
- That is, if a is a primitive root of the prime number p, then the numbers

  $a \bmod p,\ a^2 \bmod p, \ldots\ldots\ldots,\ a^{p-1} \bmod p$

- are distinct and consist of the integers from 1 through p-1 in some permutation.

- For any integer b and a primitive root a of prime number p, we can find a unique exponent i such that
- $b = a^i \pmod{p}$ where $0 \ldots i \ldots (p - 1)$
- The exponent i is referred to as the discrete logarithm of b for the base a, mod p.
- We express this value as $\text{dlog}_{a,p}(b)$.

The Algorithm

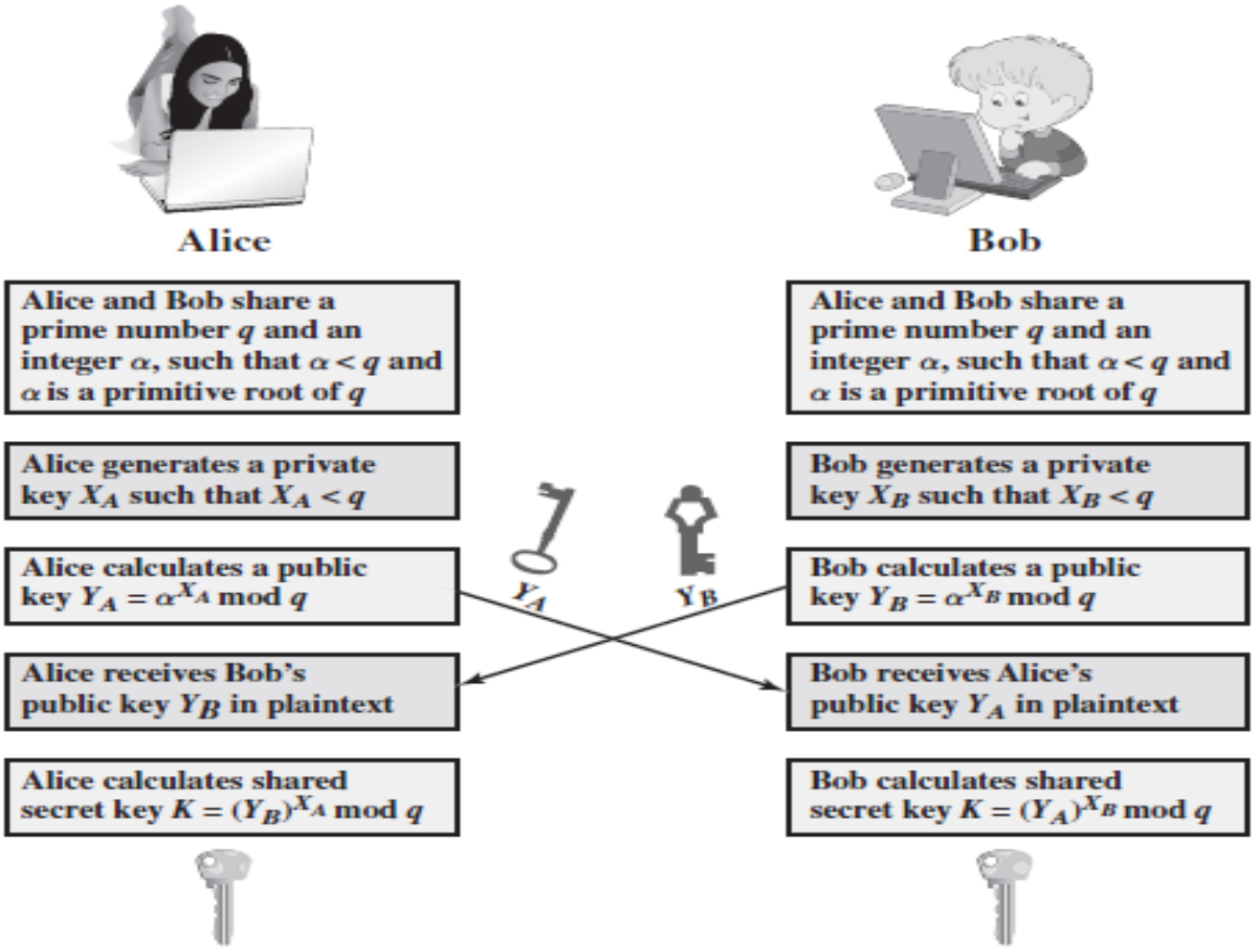


Figure 10.1 The Diffie-Hellman Key Exchange

- For this scheme, there are two publicly known numbers:
- a prime number q and an integer a that is a primitive root of q.
- Suppose the users A and B wish to create a shared key.
- User A selects a random integer $X_A < q$ and computes $Y_A = a^{XA} \bmod q$.
- Similarly, user B independently selects a random integer $X_B < q$ and computes

$$Y_B = a^{XB} \bmod q.$$

- Each side keeps the X value private and makes the Y value available publicly to the other side.
- Thus, X_A is A's private key and Y_A is A's corresponding public key, and similarly for B.
- User A computes the key as

  $K = (Y_B)^{XA} \bmod q$ and
- user B computes the key as

  $K = (Y_A)^{XB} \bmod q$.
- These two calculations produce identical results:

$$
\begin{aligned}
K &= (Y_B)^{X_A} \bmod q \\
&= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\
&= (\alpha^{X_B})^{X_A} \bmod q && \text{by the rules of modular arithmetic} \\
&= \alpha^{X_B X_A} \bmod q \\
&= (\alpha^{X_A})^{X_B} \bmod q \\
&= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}
$$

- consider an adversary who can observe the key exchange and wishes to determine the secret key K.
- Because X_A and X_B are private, an adversary only has the following ingredients to work with: q, a, Y_A, and Y_B.
- Thus, the adversary is forced to take a discrete logarithm to determine the key.
- For example, to determine the private key of user B, an adversary must compute

$$X_B = \text{dlog}_{a,q}(Y_B)$$

- The adversary can then calculate the key K in the same manner as user B calculates it.
- That is, the adversary can calculate K as

$$K = (Y_A)^{XB} \bmod q$$

- The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms.

Example

- Consider the prime number q = 353 and a primitive root of 353, in this case a = 3.
- A and B select private keys $X_A = 97$ and $X_B = 233$, respectively.
- A computes $Y_A = 3^{97} \bmod 353 = 40$.
- B computes $Y_B = 3^{233} \bmod 353 = 248$.

- After they exchange public keys, each can compute the common secret key:
- A computes $K = (Y_B)^{XA} \bmod 353$

  $= 248^{97} \bmod 353 = 160.$
- B computes $K = (Y_A)^{XB} \bmod 353$

  $= 40^{233} \bmod 353 = 160.$

- Consider a Diffie-Hellman scheme with a common prime q = 11 and a primitive root a = 2.
  a. Show that 2 is a primitive root of 11.
  b. If user A has public key $Y_A = 9$, what is A's private key X_A?
  c. If user B has public key $Y_B = 3$, what is the secret key K shared with A?

Key Exchange Protocols

- Suppose that user A wishes to set up a connection with user B and use a secret key to encrypt messages on that connection.
- User A can generate a one-time private key X_A, calculate Y_A, and send that to user B.
- User B responds by generating a private value X_B, calculating Y_B, and sending YB to user A.
- Both users can now calculate the key.
- The necessary public values q and a would need to be known ahead of time.
- Alternatively, user A could pick values for q and a and include those in the first message.

Another Example

- Suppose that a group of users (e.g., all users on a LAN) each generate a long-lasting private value Xi (for user i) and calculate a public value Yi.
- These public values, together with global public values for q and a, are stored in some central directory.
- At any time, user j can access user i's public value, calculate a secret key, and use that to send an encrypted message to user A.

- If the central directory is trusted, then this form of communication provides both confidentiality and a degree of authentication.
- Because only i and j can determine the key, no other user can read the message (confidentiality).
- Recipient i knows that only user j could have created a message using this key (authentication).
- However, the technique does not protect against replay attacks.

Man-in-the-Middle Attack

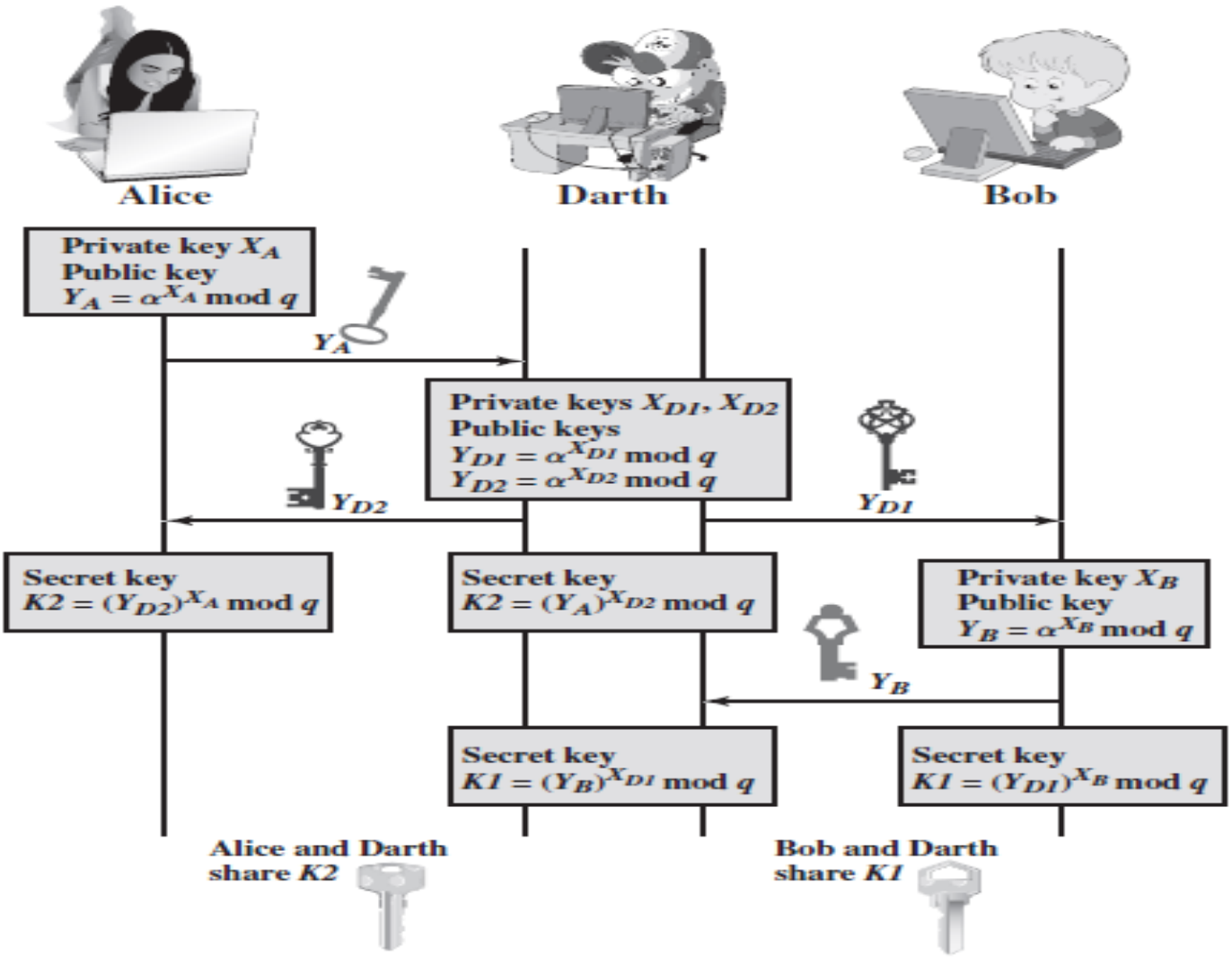


Figure 10.2 Man-in-the-Middle Attack

- The attack proceeds as follows :

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts YA and transmits Y_{D1} to Bob. Darth also calculates

   $K2 = (Y_A)^{XD2} \bmod q.$
4. Bob receives Y_{D1} and calculates

   $K1 = (Y_{D1})^{XB} \bmod q.$
5. Bob transmits Y_B to Alice.
6. Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates

   $K1 = (Y_B)^{XD1} \bmod q.$
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{XA} \bmod q.$

- At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key K1 and Alice and Darth share secret key K2.
- All future communication between Bob and Alice is compromised in the following way.
  1. Alice sends an encrypted message M: E(K2, M).
  2. Darth intercepts the encrypted message and decrypts it to recover M.
  3. Darth sends Bob E(K1, M) or E(K1, M'), where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

Elgamal Cryptographic System

- In 1984, T. Elgamal announced a public-key scheme based on discrete logarithms, closely related to the Diffie-Hellman technique [ELGA84, ELGA85].
- The Elgamal2 cryptosystem is used in some form in a number of standards including the digital signature standard (DSS), and the S/MIME e-mail standard.

- The global elements of Elgamal are a prime number *q and α, which is a primitive root of q.*
- *User A generates a private/public key pair* as follows:
  - 1. Generate a random integer X_A, such that $1 < X_A < q - 1$.
  - 2. Compute $YA = a^{XA} \bmod q$.
  - 3. A's private key is X_A and A's public key is {q, α, YA}.

- Any user B that has access to A's public key can encrypt a message as follows:
  - 1. Represent the message as an integer M in the range $0 \leq M \leq q - 1$. Longer messages are sent as a sequence of blocks, with each block being an integer less than q.
  - 2. Choose a random integer k such that $1 \leq k \leq q - 1$.
  - 3. Compute a one-time key $K = (Y_A)^k \bmod q$.
  - 4. Encrypt M as the pair of integers (C1, C2) where
  - $C_1 = \alpha^k \bmod q$; $C_2 = KM \bmod q$

- User A recovers the plaintext as follows:
  - Recover the key by computing

    $K = (C_1)^{XA} \bmod q.$
  - Compute $M = (C_2 K^{-1}) \bmod q.$

Global Public Elements	
q	prime number
α	$\alpha < q$ and α a primitive root of q

Key Generation by Alice	
Select private X_A	$X_A < q - 1$
Calculate Y_A	$Y_A = \alpha^{X_A} \bmod q$
Public key	$\{q, \alpha, Y_A\}$
Private key	X_A

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < q$
Select random integer k	$k < q$
Calculate K	$K = (Y_A)^k \bmod q$
Calculate C_1	$C_1 = \alpha^k \bmod q$
Calculate C_2	$C_2 = KM \bmod q$
Ciphertext:	(C_1, C_2)

Decryption by Alice with Alice's Private Key	
Ciphertext:	(C_1, C_2)
Calculate K	$K = (C_1)^{X_A} \bmod q$
Plaintext:	$M = (C_2 K^{-1}) \bmod q$

Figure 10.3 The Elgamal Cryptosystem

- We can restate the Elgamal process as follows, using Figure 10.3.
  - 1. Bob generates a random integer k.
  - 2. Bob generates a one-time key K using Alice's public-key components Y_A, q, and k.
  - 3. Bob encrypts k using the public-key component α, yielding C1. C1 provides sufficient information for Alice to recover K.
  - 4. Bob encrypts the plaintext message M using K.
  - 5. Alice recovers K from C_1 using her private key.
  - 6. Alice uses K^{-1} to recover the plaintext message from C_2.