



National Chung Cheng University

Department of Information Management

Public Key Cryptography and RSA

Pei-Ju (Julian) Lee

National Chung Cheng University

Information Security

pjlee@mis.ccu.edu.tw

Fall, 2016



Public-Key Encryption

- Asymmetric, use of two separate keys
- V.S. Symmetric encryption, which uses only one key

Asymmetric Keys

Two related keys, a public key and a private key, that are used to perform complementary operations, such as encryption and decryption or signature generation and signature verification.

Public Key Certificate

A digital document issued and digitally signed by the private key of a Certification Authority that binds the name of a subscriber to a public key. The certificate indicates that the subscriber identified in the certificate has sole control and access to the corresponding private key.

Public Key (Asymmetric) Cryptographic Algorithm

A cryptographic algorithm that uses two related keys, a public key and a private key. The two keys have the property that deriving the private key from the public key is computationally infeasible.

Public Key Infrastructure (PKI)

A set of policies, processes, server platforms, software and workstations used for the purpose of administering certificates and public-private key pairs, including the ability to issue, maintain, and revoke public key certificates.



Misconceptions Concerning Public-Key Encryption

- More **secure** from cryptanalysis than **symmetric** encryption?
 - The security of any encryption scheme depends on **the length of the key** and the **computational work** involved in breaking a cipher
- Is a general-purpose technique that has made symmetric encryption obsolete?
 - **Computational** overhead of current public-key encryption schemes, there seems no foreseeable likelihood that symmetric encryption will be abandoned
- **Key distribution** is trivial when using **public-key encryption**, compared to the rather cumbersome handshaking involved with key distribution centers for symmetric encryption?
 - In fact, some form of protocol is needed, generally involving a central agent, and the procedures involved are not simpler nor any more efficient



Principles of Public-Key Cryptosystems

- The concept evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption:

Key distribution

- How to have secure communications in general without having to trust a KDC (key distribution center) with your key

Digital signatures

- How to verify that a message comes intact from the claimed sender

- Whitfield Diffie and Martin Hellman from Stanford University came up with a method that addressed both problems in 1976. They also discovered the public-key encryption



Problems cont'd

- **Key distribution** under symmetric encryption requires either
 - (1) two communicants **already share a key**, which somehow has been distributed to them; or
 - (2) the use of a **key distribution center (KDC)**
- **Digital signatures**
 - could a method be devised that would stipulate, to the satisfaction of all parties, that a digital message had been sent by a particular person?



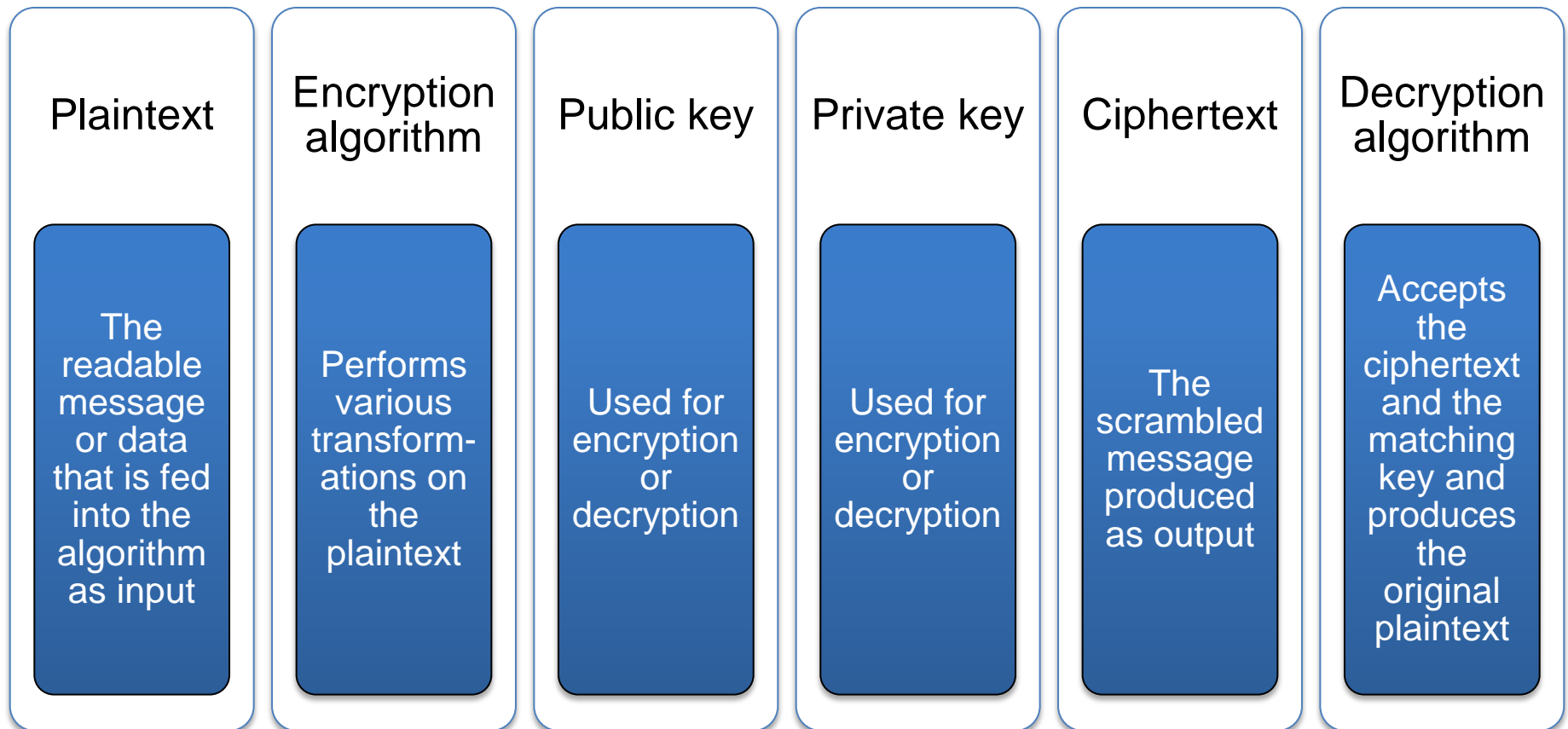
Public-Key Cryptosystems

- Asymmetric algorithms rely on one key for encryption and a different but related key for decryption
- Characteristics
 - It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.
 - Either of the two related keys can be used for encryption, with the other used for decryption. (Ex. RSA)

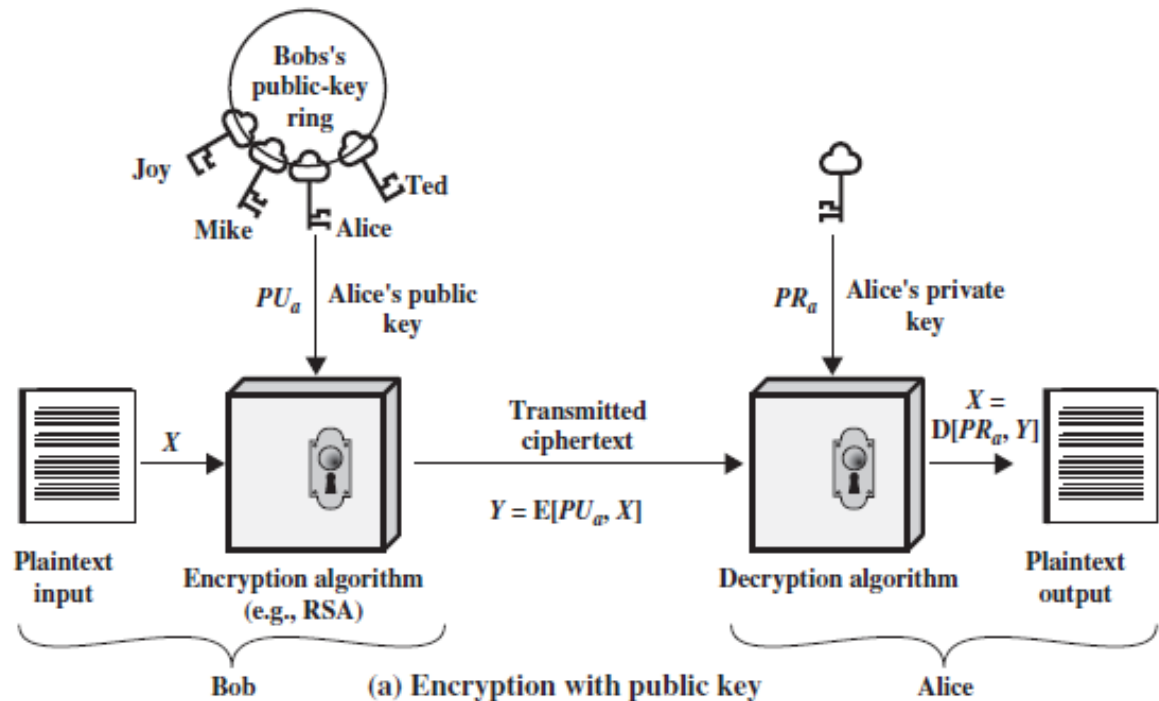


Public-Key Cryptosystems

- A public-key encryption scheme has six ingredients



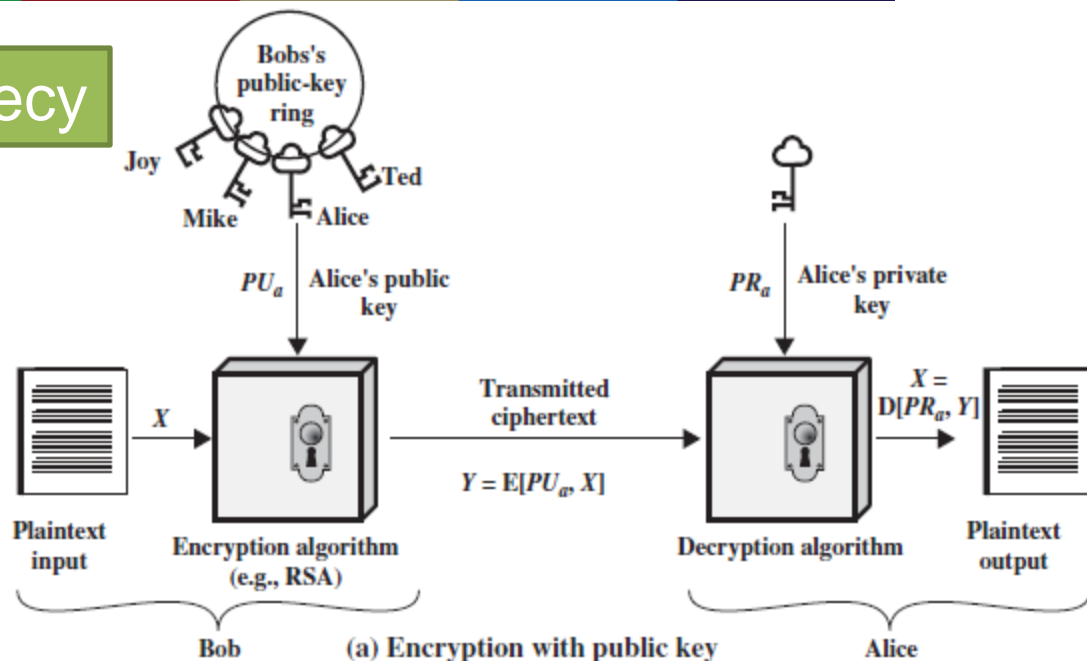
Public-Key Cryptography



- Steps:
 1. Each user generates a pair of keys for E/D
 2. Each user places one of the two keys in a public register or other accessible file.
 3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key
 4. When Alice receives the message, she decrypts it using her private key.

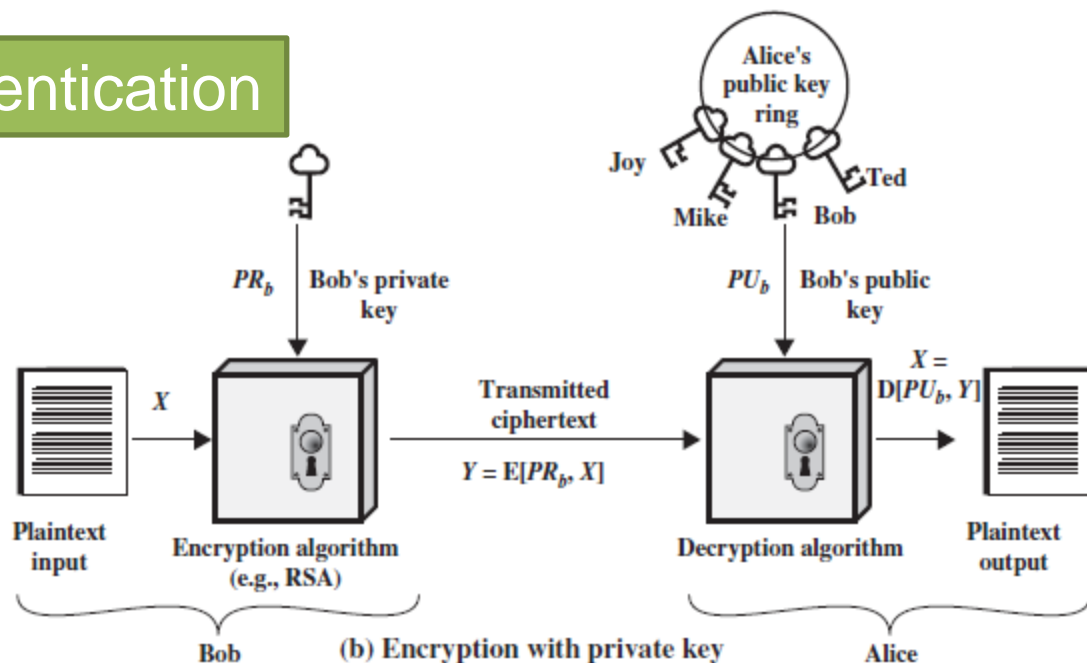
- As long as a **user's private key** remains protected and secret, incoming communication is secure.

Secrecy



- At any time, a system can **change** its private key and **publish** the companion public key to replace its old public key.

Authentication



Authentication does not provide confidentiality, i.e. message being sent is safe from alteration but not from eavesdropping



Conventional and Public-Key Encryption

Conventional Encryption	Public-Key Encryption
<p><i>Needed to Work:</i></p> <ol style="list-style-type: none">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. <p><i>Needed for Security:</i></p> <ol style="list-style-type: none">1. The key must be kept secret.2. It must be impossible or at least impractical to decipher a message if the key is kept secret.3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	<p><i>Needed to Work:</i></p> <ol style="list-style-type: none">1. One algorithm is used for encryption and a related algorithm for 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). <p><i>Needed for Security:</i></p> <ol style="list-style-type: none">1. One of the two keys must be kept secret.2. It must be impossible or at least impractical to decipher a message if one of the keys is kept secret.3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.



Discrimination

- To discriminate between the symmetric and public-key encryption
 - The key used in **symmetric** encryption as a **secret key**
 - The two keys used for **asymmetric** encryption are referred to as the **public key** and the **private key**

Symmetric encryption
Conventional Encryption
Classical Encryption

Asymmetric encryption
Public-Key Encryption

Public-Key Cryptosystem: Secrecy

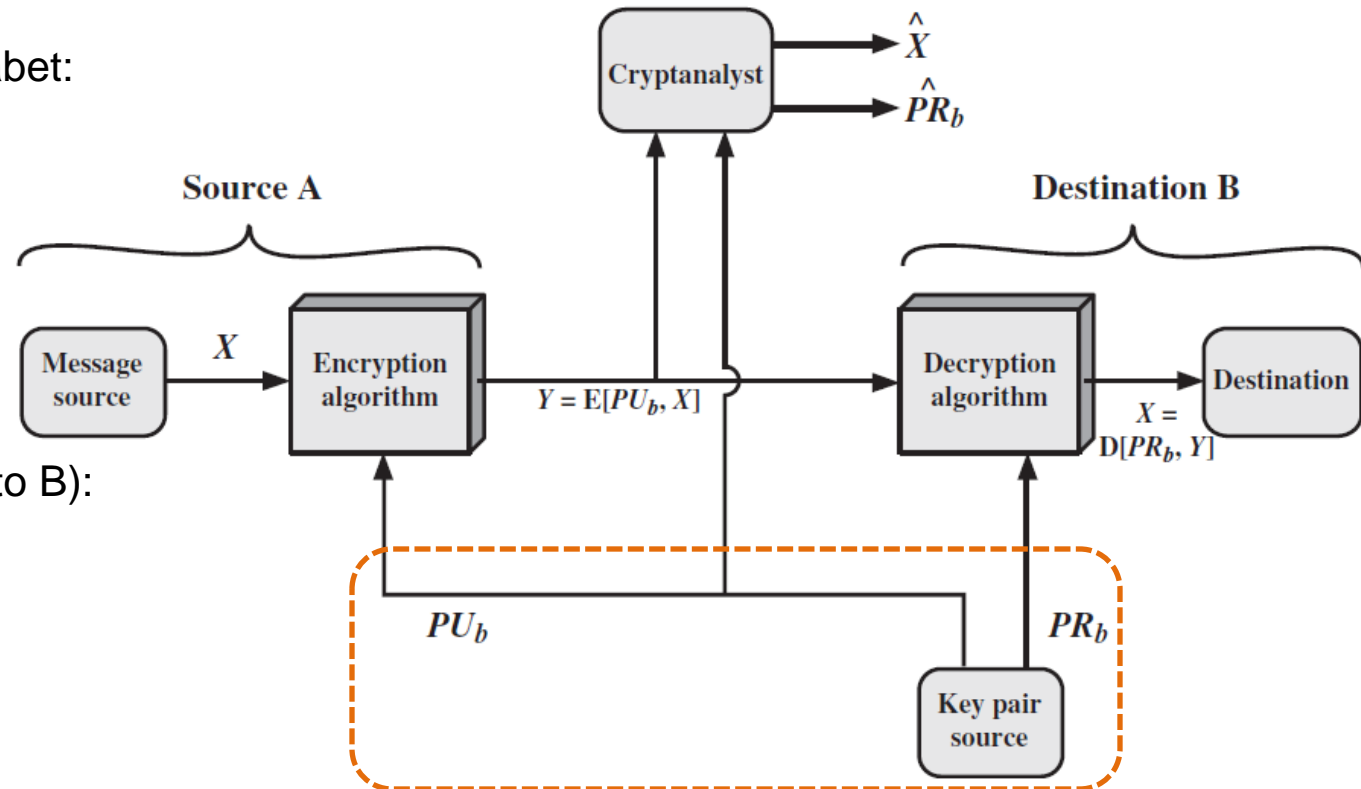
- The M elements of X are letters in some finite alphabet:
 $X = [X_1, X_2, \dots, X_M]$

- Public Key: PU_b

- Private Key (known only to B):
 PR_b

- Ciphertext:
 $Y = E(PU_b, X)$

- Plaintext:
 $X = D(PR_b, Y)$



- Adversary knows:

– Y

– PU_b

– E/D algorithm

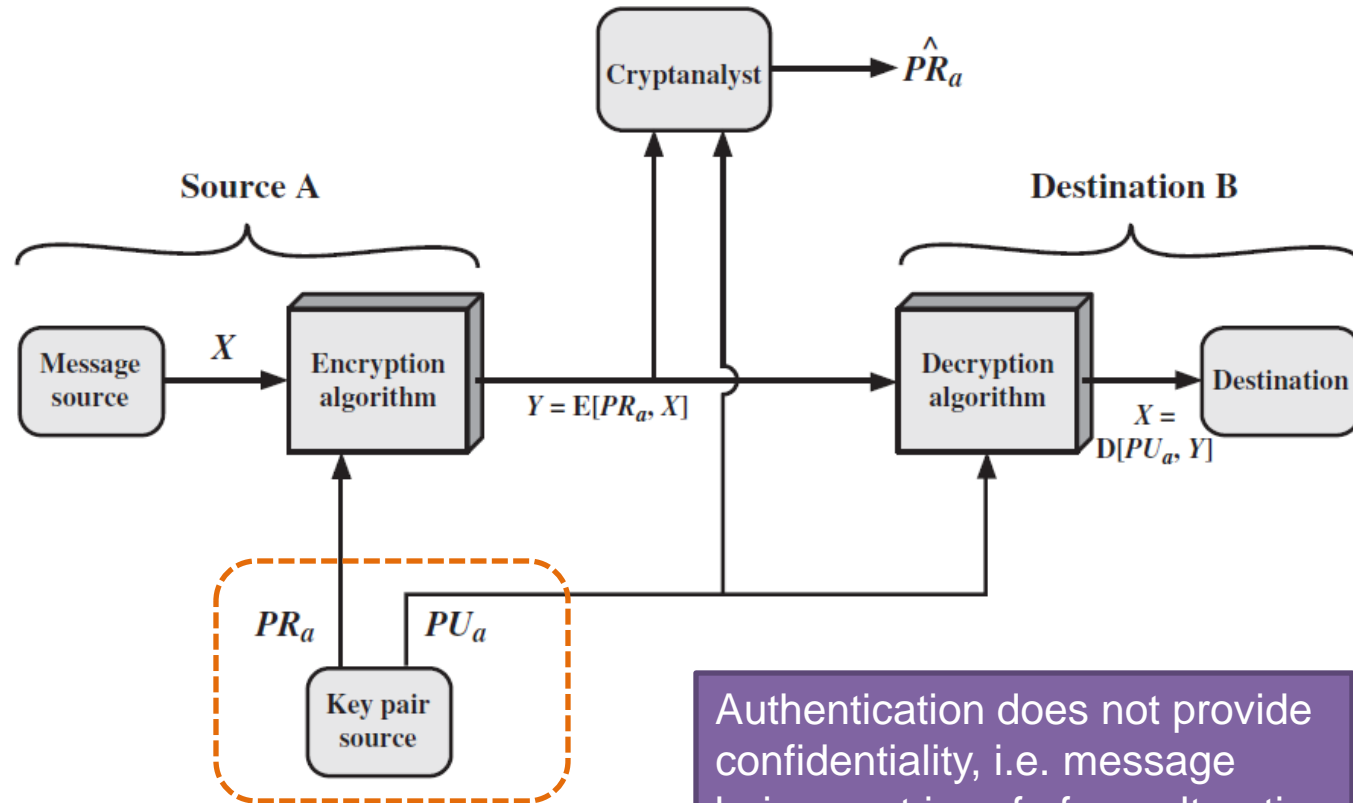
Attempt to recover X or PR_b

- If only interested in particular message: \hat{x}

- If interested in read future message: \widehat{PR}_b

Public-Key Cryptosystem: Authentication

- Use of public-key to provide authentication
- The encrypted message serves as a **digital signature**. (only A can prepare the message)
- $Y = E(PR_b, X)$
• $X = D(PU_b, Y)$
- Data integrity



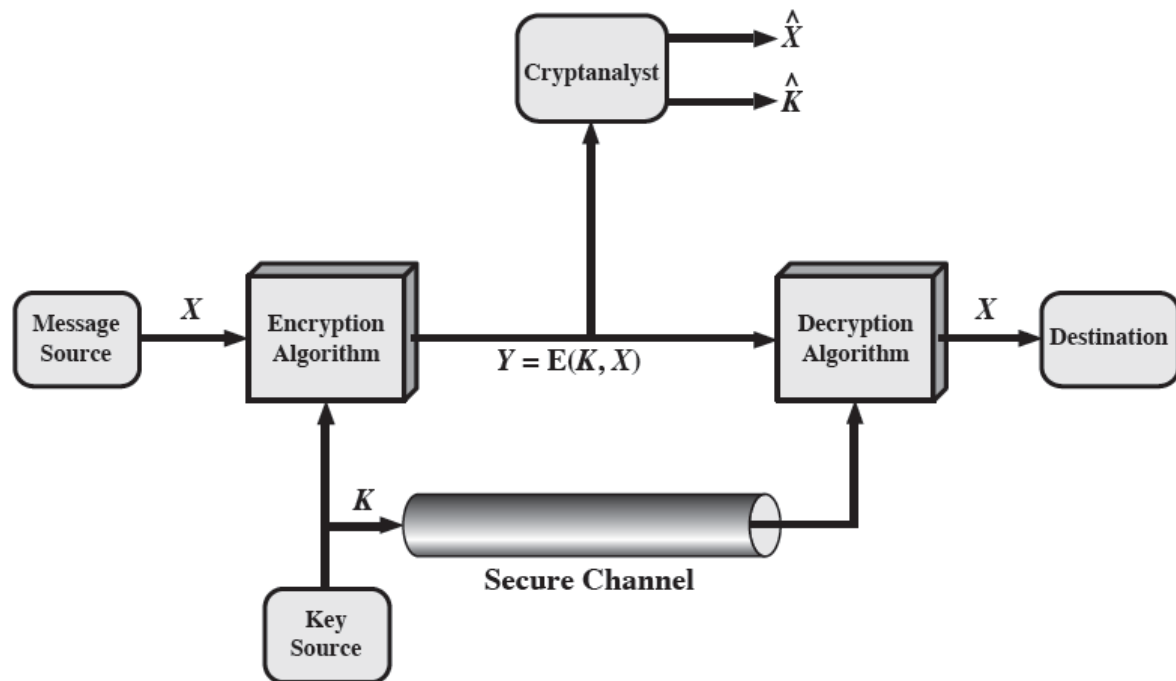
Authentication does not provide confidentiality, i.e. message being sent is safe from alteration but not from eavesdropping

- More efficient way of archiving: encrypt a small block of bits, **authenticator**, instead of encrypt the whole plaintext for authentication.
- The authenticator is infeasible to change the document without changing the indicator

Recall: Model of Symmetric Cryptosystem

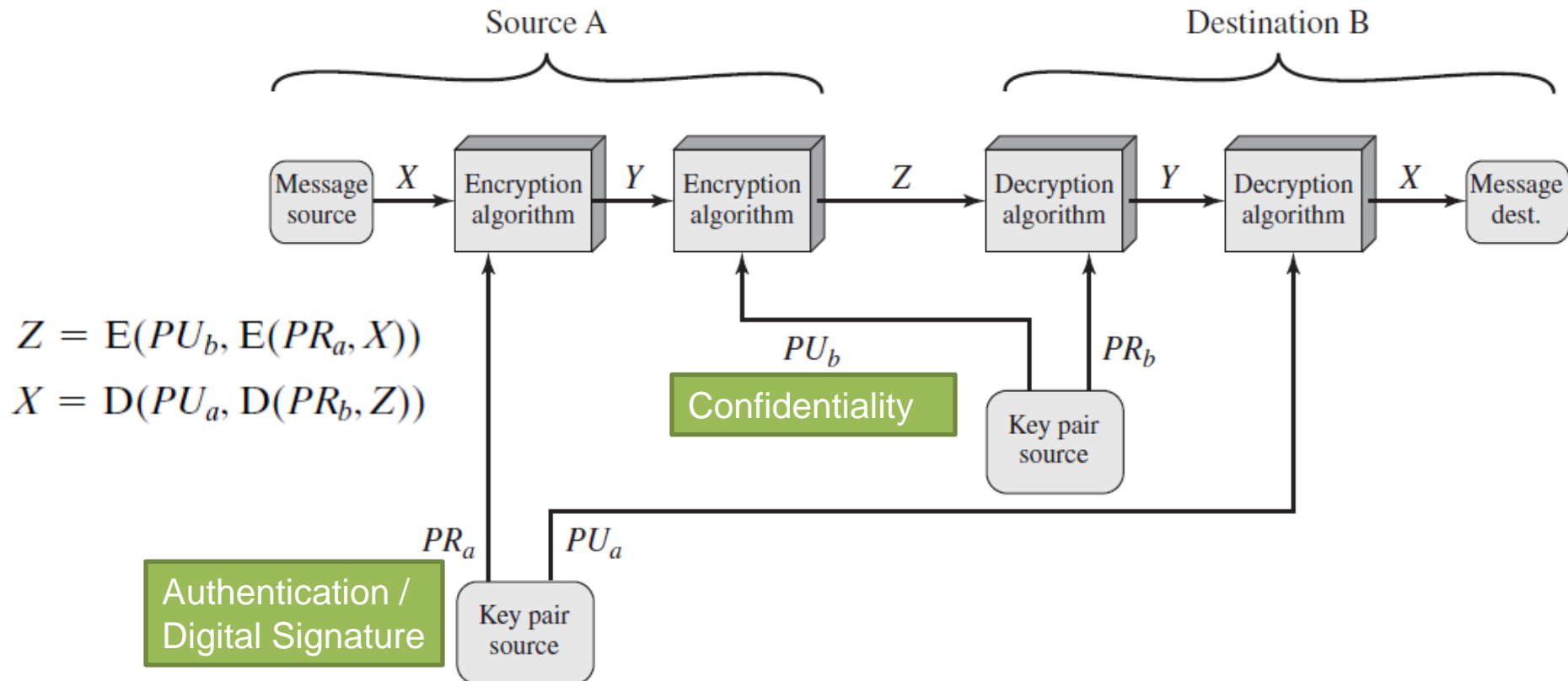
- Don't need to keep the algorithm secret; but only keep the **key** secret

- Plaintext: $X=[X_1, \dots, X_M]$, letter, binary code
- Key: $K=[K_1, \dots, K_j]$
- Ciphertext: $Y=[Y_1, \dots, Y_N]$
- $Y=E(K, X)$: Y is produced by using encryption algorithm E as a function of the plaintext X , with the specific function determined by the value of the key K
- $X=D(K, Y)$
- Plaintext estimate: \hat{X}
- Key estimate: \hat{K}



Authentication and Secrecy

- Previous figures (authentication)' Disadvantage: doesn't provide confidentiality, the message is safe from alteration but not from eavesdropping
- Double use of the public-key scheme
- Exercise 4 times of the public-key algorithm





Applications for Public-Key Cryptosystems

- Depending on the application, the sender uses either:
 - the sender's private key, or the receiver's public key, or both.
- Public-key cryptosystems can be classified into **three categories**: (Some algorithms are suitable for all three applications, whereas others can be used only for one or two)

Encryption/decryption

- The sender encrypts a message with the recipient's public key

Digital signature

- The sender "signs" a message with its private key

Key exchange

- Two sides cooperate to exchange a session key, involving the private key(s) of one or both parties



Public-Key Requirements

1. It is computationally easy for a party B to generate a pair (public-key PU_b , private key PR_b)
 2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, to generate the corresponding ciphertext: $C = E(PU_b, M)$
 3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:
 $M = D(PR_b, C) = D[PR_b, E(PU_b, M)]$
 4. It is computationally infeasible for an adversary, knowing the public key PU_b , to determine the private key PR_b
 5. It is computationally infeasible for an adversary, knowing the public key PU_b and a ciphertext C , to recover the original message M
 6. The two keys can be applied in either order
 $M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$
- These are [formidable requirements](#), only a few algorithms (RSA, elliptic curve cryptography, Diffie-Hellman, DSS) have received widespread acceptance



Cont'd

- Previous requirements are boil down for a **trap-door one-way function**
 - One way function
 - 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: 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
 - Infeasible: if the effort to solve it grows faster than polynomial time as a function of input size. For example, if the length of the input is n bits and the time to compute the function is proportional to 2^n



Cont'd

- A trap-door 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 known but k not known
 - Easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known
- A practical public-key scheme depends on a suitable trap-door one-way function



Public-Key Cryptanalysis

- 1. Brute-force attack (as with symmetric encryption)
 - Countermeasure: use large keys
 - But Key size must be small enough for practical E/D
 - Key sizes that have been proposed make E/D too slow
 - » Therefore, Public-key encryption is only used to key management and signature applications
- 2. To compute the private key given the public key
 - To date it has not been proven that this attack is infeasible
- 3. Probable-message attack
 - Ex. A message were to be sent of a (56-bit DES key). An adversary could encrypt all possible 56-bit DES keys using the public key and could discover the encrypted key by matching the transmitted ciphertext
 - This attack can be thwarted by appending some **random bits** to such simple messages



Rivest-Shamir-Adleman (RSA) Scheme

- Developed in 1977 at MIT by Ron Rivest, Adi Shamir & Len Adleman [RIVE78]
- Most widely used general-purpose approach to public-key encryption
- Is a 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, or 309 decimal digits
 - That is, n is less than 2^{1024}



RSA Algorithm

- Makes use of an expression with **exponentials**
- Plaintext is encrypted in **blocks**
 - each block having a binary value less than some number n
 - i.e. the **block size** is less than or equal to $\log_2(n)+1$



RSA Algorithm

- Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C

$$C = M^e \bmod n$$

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

- n : both sender and receiver must know n
- e : the sender knows e
- d : only the receiver knows d
- Public key: $PU = \{e, n\}$
- Private key: $PR = \{d, n\}$



Algorithm Requirements

- For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:
 - 1. It is possible to find values of e, d, n such that $M^{ed} \bmod n = M$ for all $M < n$
 - 2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$
 - 3. It is infeasible to determine d given e and n



1st requirement

$$M^{ed} \bmod n = M$$

- The preceding relationship holds if e and d are multiplicative inverses modulo $\Phi(n)$

- $\Phi(n)$: Euler totient function.

- For p, q prime, $n = p \times q$, $\Phi(pq) = (p - 1)(q - 1)$

- Thus, for e and d

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

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

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

- This is true only if d (and therefore e) is relatively prime to $\Phi(n)$. Equivalently, $\gcd(\Phi(n), d) = 1$



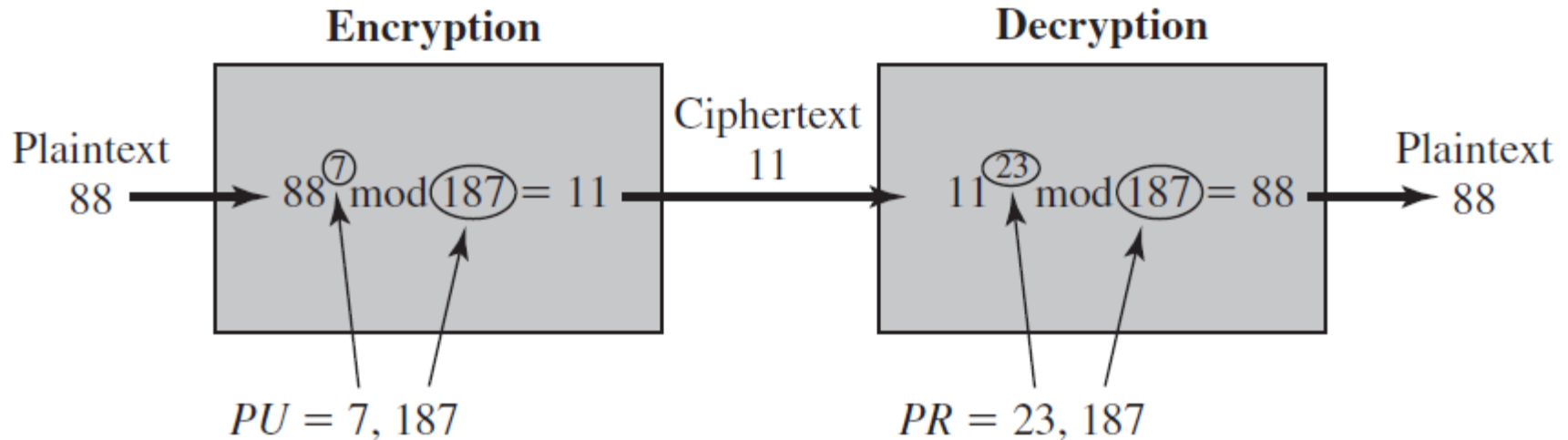
Cont'd

- Private key consists of $\{d, n\}$ and the public key consists of $\{e, n\}$.
- Ingredients:
 - p, q , two prime numbers (private, chosen)
 - $n = pq$ (public, calculated)
 - e , with $\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$ (public, chosen)
 - $d \equiv e^{-1} \pmod{\phi(n)}$ (private, calculated)
- User B wishes to send the message M to A:
 - $C = M^e \bmod n, M = C^d \bmod n$

- $M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n = M^{(ed \bmod \phi(n))} \bmod n = M^1 \bmod n = M$

Example

- Alice generates a public/private key pair; Bob encrypts using Alice's public key; and Alice decrypts using her private key



Cont'd

Key Generation by Alice

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 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:	$M = C^d \bmod n$

1.Key generation

Key Generation by Alice

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	$PR = \{d, n\}$

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 \times 11 = 187$.
3. Calculate $\phi(n) = (p - 1)(q - 1) = 16 \times 10 = 160$.
4. Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
5. Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because $23 \times 7 = 161 = (1 \times 160) + 1$; d can be calculated using the extended Euclid's algorithm (Chapter 4).

The resulting keys are public key $PU = \{7, 187\}$ and private key $PR = \{23, 187\}$.



2. Encryption

Encryption by Bob with Alice's Public Key

Plaintext: $M < n$

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

plaintext input of $M = 88$

we need to calculate $C = 88^7 \bmod 187$

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$

$$88^1 \bmod 187 = 88$$

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

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

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



3.Decryption

Decryption by Alice with Alice's Public Key

Ciphertext:

C

Plaintext:

$$M = C^d \bmod n$$

we calculate $M = 11^{23} \bmod 187$:

$$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

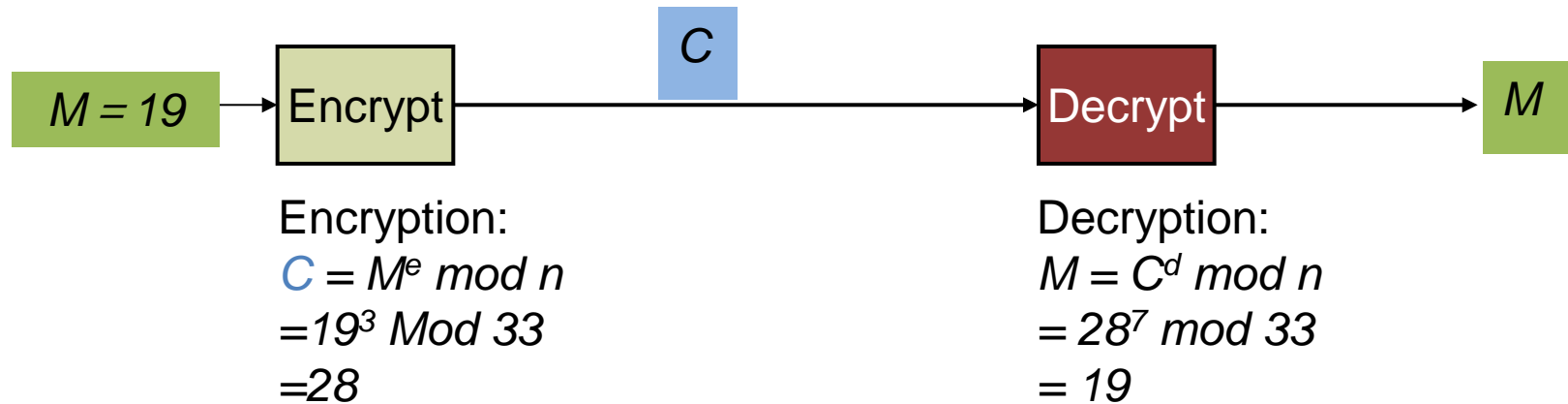
$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14,641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214,358,881 \bmod 187 = 33$$

$$\begin{aligned} 11^{23} \bmod 187 &= (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 \\ &= 79,720,245 \bmod 187 = 88 \end{aligned}$$

Example



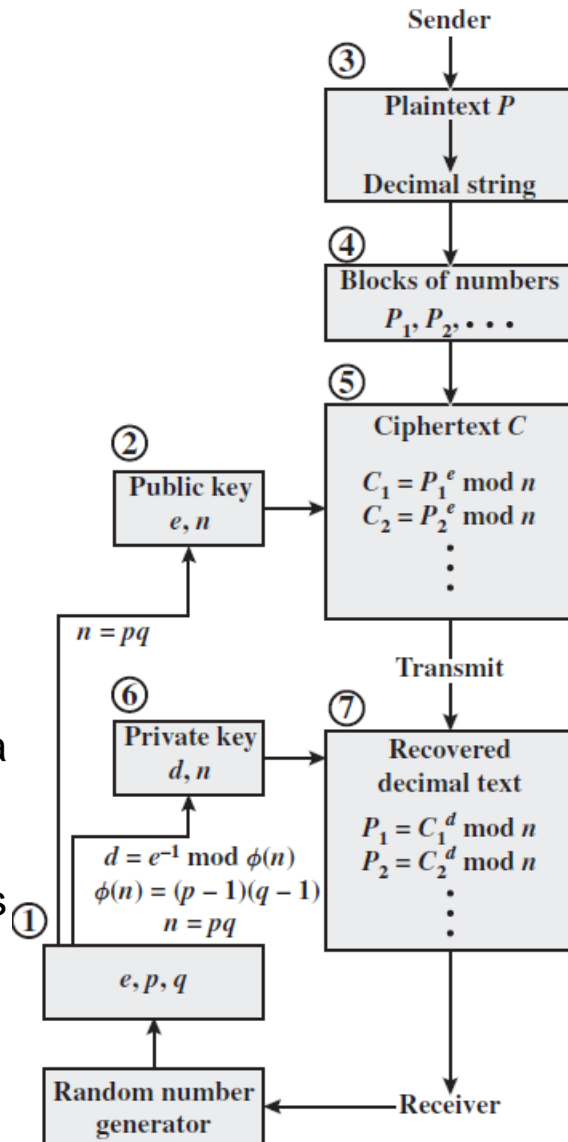
Recipient :
PU = {**e**, **n**}
PR = {**d**, **n**}

1. Recipient choose
 $p=3, q=11 ; n = p \cdot q = 33$
2. Compute $\phi(n) = (p-1) \times (q-1) = (3-1)(11-1) = 2 \times 10 = 20$
3. Find $e=3$ let $(e, \phi(n))=1$
 $\{e, n\} = (3, 33)$: Recipient's PU
4. According to $e \cdot d \equiv 1 \bmod 20$
compute $d = 7$
 d : PR

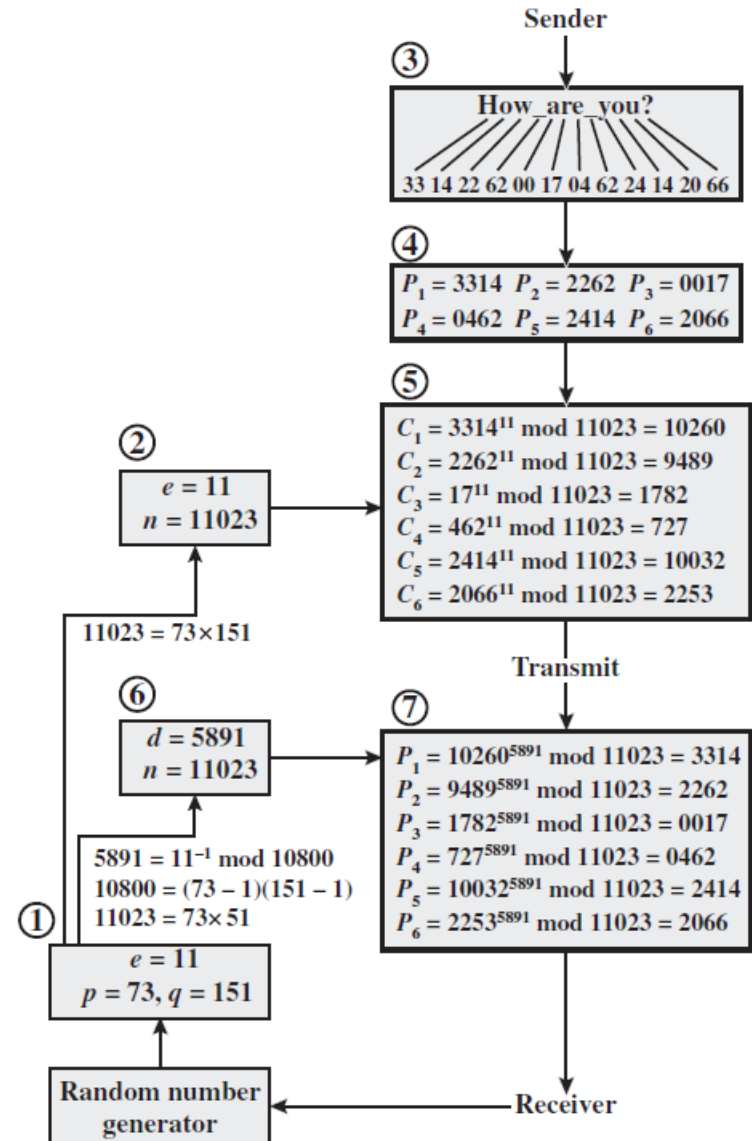
RSA

Processing of Multiple Blocks

- Plaintext: alphanumeric string.
 - Each plaintext symbol is assigned a unique code of two decimal digits (e.g., a = 00, A = 26).
- A plaintext block consists of four decimal digits, or two alphanumeric characters.



(a) General approach



(b) Example



Complexity of the Computation

- Two issues to consider:

- Encryption/decryption
- Key generation

- Exponentiation in Modular Arithmetic

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

- We can reduce intermediate results modulo n

- Fast Modular Exponentiation Algorithm

- Ex. $X^{11} \bmod n = x^{1+2+8} \bmod n = ((x)(x^2)(x^8)) \bmod n$
 $= [(x \bmod n)(x^2 \bmod n)(x^8 \bmod n)] \bmod n$

Express as a binary number $b_k b_{k-1} \dots b_0$

$$a^b \bmod n \qquad b = \sum_{b_i \neq 0} 2^i \qquad a^b = a^{\left(\sum_{b_i \neq 0} 2^i\right)} = \prod_{b_i \neq 0} a^{(2^i)}$$

$$a^b \bmod n = \left[\prod_{b_i \neq 0} a^{(2^i)} \right] \bmod n = \left(\prod_{b_i \neq 0} \left[a^{(2^i)} \bmod n \right] \right) \bmod n$$

Cont'd

Table 9.4 Result of the Fast Modular Exponentiation Algorithm for $a^b \bmod n$, where $a = 7$, $b = 560 = 1000110000$, and $n = 561$

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
f	7	49	157	526	160	241	298	166	67	1

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

Ex. $3^3 \bmod 7$

- For small public key, i.e. 3, is vulnerable to a simple attack

Note: The integer b is expressed as a binary number $b_k b_{k-1} \dots b_0$.



Efficient Operation Using the Public Key

- Efficient Operation Using the Public Key
 - Specific choice of e to speed up the RSA operation
 - Most common choice of e : 65537 ($2^{16}+1$), 3, 17
 - Small public key (Ex.3) is vulnerable to a simple attack
 - Ex. $e = 3$, unique n for 3 RSA users:
 $C_1 = M^3 \bmod n_1$
 $C_2 = M^3 \bmod n_2$
 $C_3 = M^3 \bmod n_3$
One can use the Chinese Remainder Theorem to compute $M^3 \bmod (n_1 n_2 n_3)$
 - Note: the RSA requires the user selects e that is relatively prime to $\Phi(n)$.
 - Thus, we have to select primes p and q carefully to prevent $\gcd(\Phi(n), e) \neq 1$ if e is already selected.



Cont'd

- Efficient Operation Using the **Private Key**
 - Cannot choose a small constant value of d , otherwise it is vulnerable to a brute-force attack and other cryptanalysis
 - Using CRT to speed up computation of $M = C^d \bmod n$

$$V_p = C^d \bmod p \quad V_q = C^d \bmod q$$

$$X_p = q \times (q^{-1} \bmod p) \quad X_q = p \times (p^{-1} \bmod q)$$

$$M = (V_p X_p + V_q X_q) \bmod n$$

Using Fermat's theorem that if p and q are relatively prime then $a^{p-1} \equiv 1 \pmod{p}$

$$V_p = C^d \bmod p = C^{d \bmod (p-1)} \bmod p$$

$$V_q = C^d \bmod q = C^{d \bmod (q-1)} \bmod q$$



Key Generation

- Before the application of the public-key cryptosystem each participant must generate a pair of keys:
 - Determine two prime numbers p and q
 - Select either e or d and calculate the other
- Because the value of $n = pq$ will be known to any potential adversary, primes must be chosen from a sufficiently large set
 - The method used for **finding large primes** must be reasonably efficient
 - Currently, there are **no useful techniques** for finding large primes
 - Instead, we have to pick at random an odd number **and test whether that number is prime**



Procedure for Picking a Prime Number

- A variety of tests for primality are merely determine that a given integer is *probably* prime.
 - Ex. Miller-Rabin algorithm (chapter 8)
 - Generally, the prime number picking procedures is:
 1. Pick an odd integer n at random (e.g. using a pseudorandom number generator)
 2. Pick an integer $a < n$ at random
 3. Perform the probabilistic primality test (e.g. Miller-Rabin) with a as a parameter. If n fails the test, reject the value n and go to step 1
 4. If n has passed a sufficient number of tests, accept n ; otherwise, go to step 2

Note: this process is performed only when a new pair (PU, PR) is needed.



Cont'd

- Having determined prime numbers p and q
 - Selecting e and calculating d or vice versa
 - Assuming select e such that $\gcd(\Phi(n), e) = 1$ and then calculate $d \equiv e^{-1} \pmod{\Phi(n)}$
 - Using Extended Euclid's algorithm (Chapter 4)
 - Generate a series of random numbers that are multiplicative inverse to each other
 - Testing each against $\Phi(n)$ until a number relatively prime to $\Phi(n)$ is found
 - How many random numbers must we test to find a usable number, that is, a number relatively prime to $\Phi(n)$?
 - It can be shown easily that the probability that two random numbers are relatively prime is about 0.6 (Problem 8.2)



Recall

- Extended Euclidean algorithm

$$ax + by = d = \gcd(a, b)$$

If $\gcd(a, b) = 1$ then we have $ax + by = 1$

$$[(ax \bmod a) + (by \bmod a)] \bmod a = 1 \bmod a$$

$$0 + (by \bmod a) = 1$$

$$y = b^{-1}$$

- Thus, we can yield the value of the **multiplicative inverse of b** if $\gcd(a, b)=1$

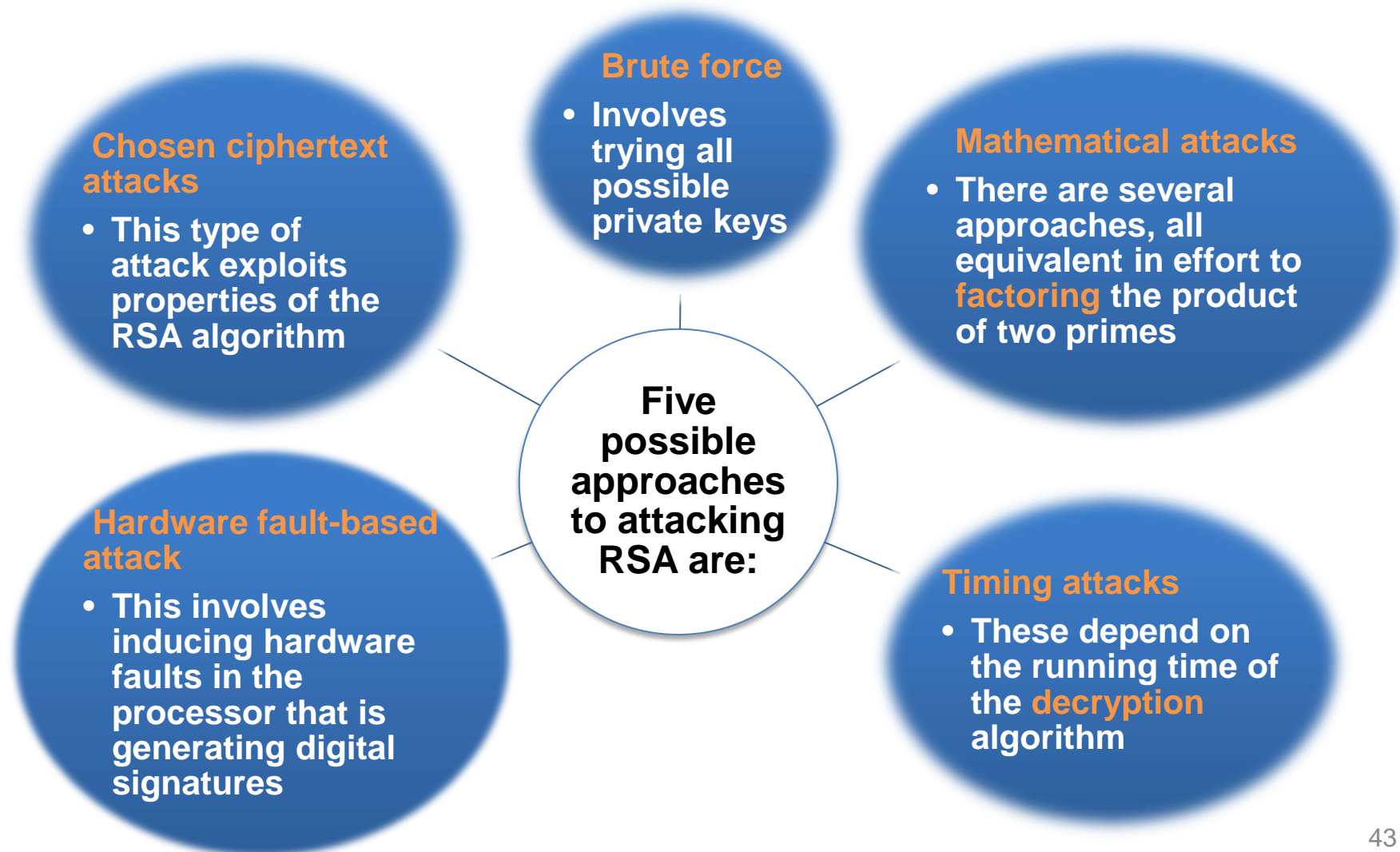
- Chapter 4

Cont'd

- The smallest positive *value* of $ax+by = \gcd(a, b)$

Extended Euclidean Algorithm			
Calculate	Which satisfies	Calculate	Which satisfies
$r_{-1} = a$		$x_{-1} = 1; y_{-1} = 0$	$a = ax_{-1} + by_{-1}$
$r_0 = b$		$x_0 = 0; y_0 = 1$	$b = ax_0 + by_0$
$r_1 = a \bmod b$ $q_1 = \lfloor a/b \rfloor$	$a = q_1b + r_1$	$x_1 = x_{-1} - q_1x_0 = 1$ $y_1 = y_{-1} - q_1y_0 = -q_1$	$r_1 = ax_1 + by_1$
$r_2 = b \bmod r_1$ $q_2 = \lfloor b/r_1 \rfloor$	$b = q_2r_1 + r_2$	$x_2 = x_0 - q_2x_1$ $y_2 = y_0 - q_2y_1$	$r_2 = ax_2 + by_2$
$r_3 = r_1 \bmod r_2$ $q_3 = \lfloor r_1/r_2 \rfloor$	$r_1 = q_3r_2 + r_3$	$x_3 = x_1 - q_3x_2$ $y_3 = y_1 - q_3y_2$	$r_3 = ax_3 + by_3$
•	•	•	•
•	•	•	•
•	•	•	•
$r_n = r_{n-2} \bmod r_{n-1}$ $q_n = \lfloor r_{n-2}/r_{n-1} \rfloor$	$r_{n-2} = q_nr_{n-1} + r_n$	$x_n = x_{n-2} - q_nx_{n-1}$ $y_n = y_{n-2} - q_ny_{n-1}$	$r_n = ax_n + by_n$
$r_{n+1} = r_{n-1} \bmod r_n = 0$ $q_{n+1} = \lfloor r_{n-1}/r_n \rfloor$	$r_{n-1} = q_{n+1}r_n + 0$		$d = \gcd(a, b) = r_n$ $x = x_n; y = y_n$

The Security of RSA

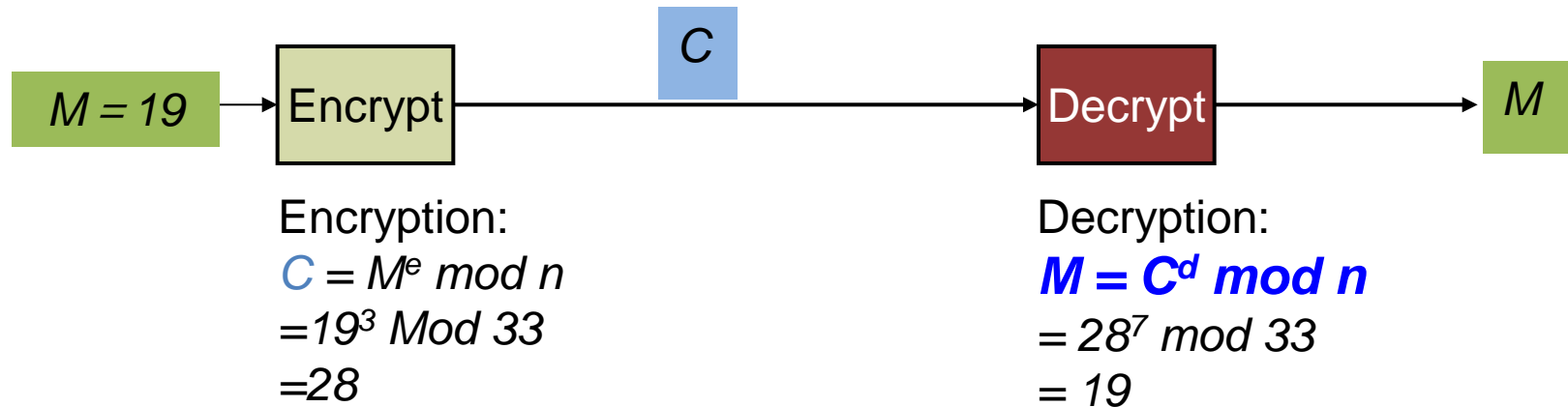




Question

- What mathematical properties of RSA algorithm make it secure?

Recall



Recipient :
PU = { e, n }
PR = { d, n }

1. Recipient choose
 $p=3, q=11 ; n = p \cdot q = 33$
2. Compute $\phi(n) = (p-1) \times (q-1) = (3-1)(11-1) = 2 \times 10 = 20$
3. Find $e=3$ let $(e, \phi(n))=1$
 $\{e, n\} = (3, 33)$: Recipient's PU
4. According to $e \cdot d \equiv 1 \bmod \phi(n)$
compute $d = 7$
 d : PR



Brute-force Approach

- The defense for RSA is the same for other cryptosystems, to use a larger key space
 - The larger number of bits in d
 - However, in both key generation and E/D are complex and the system runs slowly



Factoring Problem

- Three approaches to attacking RSA mathematically:
 1. Factor n into its two prime factors. This enables calculation of $\phi(n) = (p - 1) \times (q - 1)$, which in turn enables determination of $d = e^{-1} \pmod{\phi(n)}$
 2. Determine $\phi(n)$ directly without first determining p and q . Again this enables determination of $d = e^{-1} \pmod{\phi(n)}$
 3. Determine d directly without first determining $\phi(n)$



Cont'd

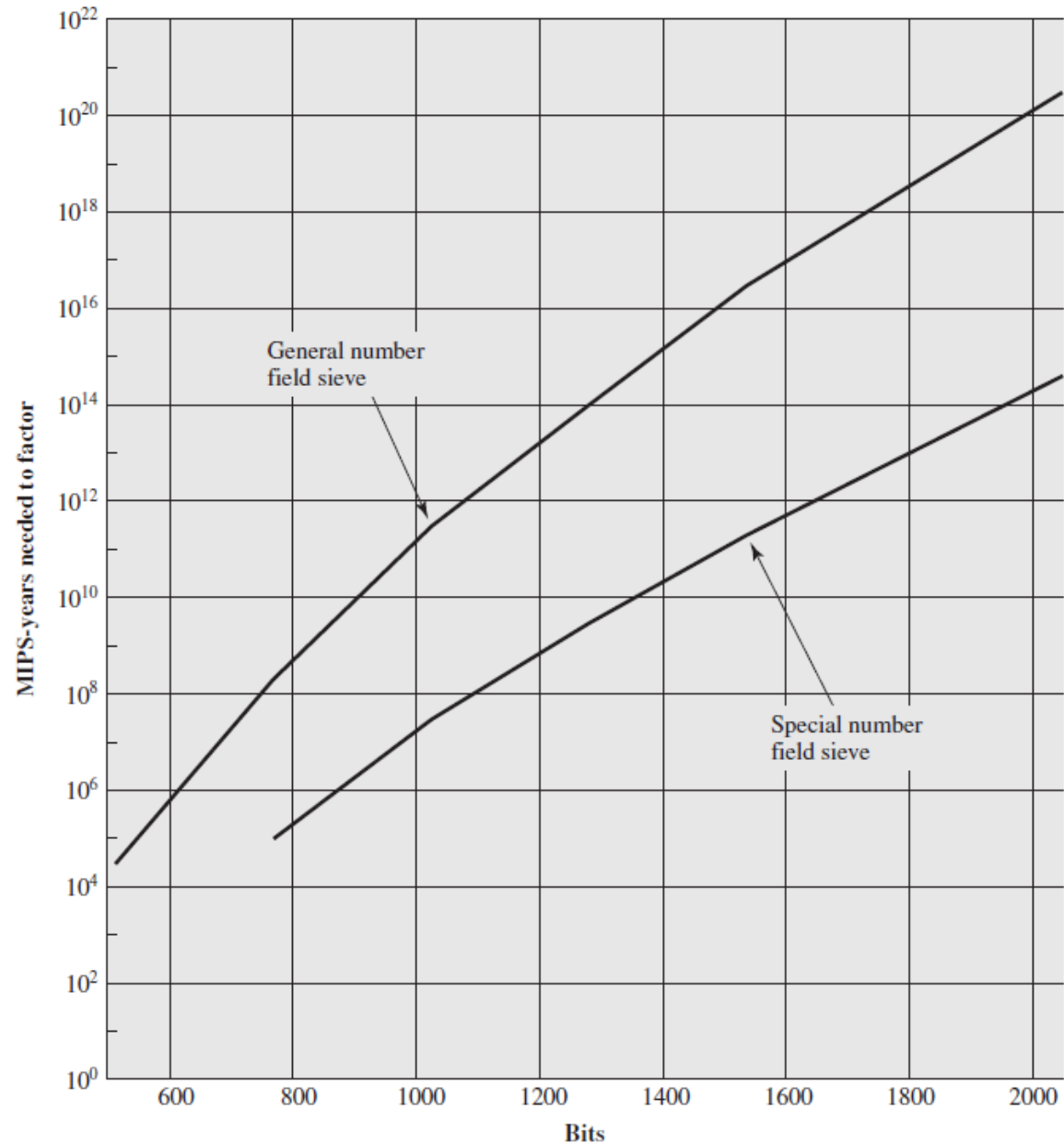
- Most discussions of the cryptanalysis of RSA have focused on the 1st task of factoring n into its two prime factors
 - For 2: Determining $\Phi(n)$ given n is equivalent to factoring n
 - For 3: With presently known algorithms, determining d given e and n appears to be at least as time-consuming as the factoring problem
- Hence, we can use factoring performance as a benchmark against which to evaluate the security of RSA



Cont'd

- $n = p * q$, p and q are primes
- $\Phi(n) = (p-1) * (q-1)$
- Example.
 - $10 = 2 * 5$; $60 = 2 * 2 * 3 * 5$
 - Factoring 3337
- Factoring Attacks on RSA
 - Quadratic sieve
 - GNFS (generalized number field sieve)
 - SNFS (special number field sieve)

- MIPS-years needed to factor
- Million-instructions-per-second processor running for one year, which is about 3×10^{13} instructions executed. A 1 GHz Pentium is about a 250-MIPS machine.





Progress in RSA Factorization

Number of Decimal Digits	n	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

- It is not unreasonable to expect that 1024 bit RSA moduli can be factored well within the next decade



Constraints

- To avoid values of n that may be factored easily:
 - Large size of n
 - Constraints on p and q :
 1. p and q should differ in length by only a few digits.
 - Thus, for a 1024-bit key, both p and q should be on the order of magnitude of 10^{75} to 10^{100}
 2. Both $(p - 1)$ and $(q - 1)$ should contain a large prime factor
 3. $\gcd(p - 1, q - 1)$ should be small

Note: if $e < n$ and $d < n^{1/4}$, then d can be easily determined



Recall: Discrete Logarithms

- Recall from Euler's theorem, for every a and n that are **relatively prime**, $a^{\Phi(n)} \equiv 1 \pmod{n}$
 - $\Phi(n)$, Euler's totient function, is the number of positive integers less than n and relatively prime to n
- If a and n are relatively prime, then there is at least one integer m that satisfies this equation, $a^m \equiv 1 \pmod{n}$, namely $m = \Phi(n)$
- Ex. consider the powers of 7, modulo 19.

$7^1 \equiv$	$7 \pmod{19}$
$7^2 = 49 = 2 \times 19 + 11 \equiv$	$11 \pmod{19}$
$7^3 = 343 = 18 \times 19 + 1 \equiv$	$1 \pmod{19}$
$7^4 = 2401 = 126 \times 19 + 7 \equiv$	$7 \pmod{19}$
$7^5 = 16807 = 884 \times 19 + 11 \equiv$	$11 \pmod{19}$

$$7^m \equiv 1 \pmod{19}, m = 3$$



Cont'd

- If the **base integer** a generates (via powers) the set of nonzero integers modulo 19. Each such integer is called a **primitive root** of the modulus 19.
- More generally:
 - The highest possible exponent to which a number can belong (mod n) is $\Phi(n)$
 - If a is a primitive root of n , then its powers $a, a^2, \dots, a^{\Phi(n)}$ are distinct (mod n) and are all relatively prime to n
 - Ex. Primitive roots for 19: 2, 3, 10, 13, 14, 15.



Cont'd

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



Logrithm function

- Ordinary logrithm function
 - the logarithm is the inverse operation to exponentiation
 - $b^y = x$, so $\log_b(x) = y$
 - Ex. $2^6 = 64$, $\log_2(64) = 6$
 - Other properties

$$\log_x(1) = 0$$

$$\log_x(x) = 1$$

$$\log_x(yz) = \log_x(y) + \log_x(z)$$

$$\log_x(y^r) = r \times \log_x(y)$$



Discrete Logarithm

- A primitive root, a , of a prime number p is one whose powers modulo p generate all the integers from 1 to $p - 1$, and the numbers $a \bmod p$, $a^2 \bmod p$, \dots , $a^{p-1} \bmod p$ are distinct
- For any integer b and a primitive root a of prime number p , we can find a unique exponent i such that $b \equiv a^i \pmod{p}$ where $0 \leq i \leq (p - 1)$
- The exponent i : the discrete logarithm of b for the base a , mod p
- We express this value as $\text{dlog}_{a,p}(b) = i$

Example

Here is an example using a nonprime modulus, $n = 9$. Here $\phi(n) = 6$ and $a = 2$ is a primitive root. We compute the various powers of a and find

$$\begin{aligned}2^0 &= 1 & 2^4 &\equiv 7 \pmod{9} \\2^1 &= 2 & 2^5 &\equiv 5 \pmod{9} \\2^2 &= 4 & 2^6 &\equiv 1 \pmod{9} \\2^3 &= 8\end{aligned}$$

This gives us the following table of the numbers with given discrete logarithms (mod 9) for the root $a = 2$:

Logarithm	0	1	2	3	4	5
Number	1	2	4	8	7	5

To make it easy to obtain the discrete logarithms of a given number, we rearrange the table:

Number	1	2	4	5	7	8
Logarithm	0	1	2	5	4	3

a	1	2	4	5	7	8
$\text{Log}_{2,9}(a)$	0	1	2	5	4	3

(a) Discrete logarithms to the base 2, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{2,19}(a)$	18	1	13	2	16	14	6	3	8	17	12	15	5	7	11	4	10	9

(b) Discrete logarithms to the base 3, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{3,19}(a)$	18	7	1	14	4	8	6	3	2	11	12	15	17	13	5	10	16	9

(c) Discrete logarithms to the base 10, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{10,19}(a)$	18	17	5	16	2	4	12	15	10	1	6	3	13	11	7	14	8	9

(d) Discrete logarithms to the base 13, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{13,19}(a)$	18	11	17	4	14	10	12	15	16	7	6	3	1	5	13	8	2	9

(e) Discrete logarithms to the base 14, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{14,19}(a)$	18	13	7	8	10	2	6	3	14	5	12	15	11	1	17	16	4	9

(f) Discrete logarithms to the base 15, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{15,19}(a)$	18	5	11	10	8	16	12	15	4	13	6	3	7	17	1	2	14	9



Cont'd

- Consider the equation, $y = g^x \bmod p$
Given y , g , and p , it is very difficult to calculate x
(take the discrete logarithm, $a^m \equiv 1 \pmod{n}$, i.e. $1 = a^m \pmod{n}$)
- Example.
 - If $3^x \bmod 17 = 15$, then $x = 6$
 - $3^x \bmod 13 = 7$, no solution!

Timing Attack

- $a^b \bmod n$

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

- Timing attacks are to all public-key cryptography systems
- A snooper can determine a private key by keeping track of how long a computer takes to decipher messages

- If the observed time to execute the decryption algorithm is always slow when this particular iteration is slow with a 1 bit, then this bit is assumed to be 1



Countermeasures

Constant exponentiation time

- Ensure that all exponentiations take the **same** amount of time before returning a result; this is a simple fix but does degrade performance

Random delay

- Better performance could be achieved by adding a **random delay** to the exponentiation algorithm to confuse the timing attack

Blinding

- Multiply the **ciphertext** by a **random number** before performing exponentiation; this process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack



Blinding Feature of RSA

- The RSA incorporates a blinding feature into some of its products:


The private-key operation $M = C_d \bmod n$ is implemented as follows.

1. Generate a secret random number r between 0 and $n - 1$.
2. Compute $C' = C(r^e) \bmod n$, where e is the public exponent.
3. Compute $M' = (C')^d \bmod n$ with the ordinary RSA implementation.
4. Compute $M = M'r^{-1} \bmod n$. In this equation, r^{-1} is the multiplicative inverse of $r \bmod n$; see Chapter 4 for a discussion of this concept. It can be demonstrated that this is the correct result by observing that $r^{ed} \bmod n = r \bmod n$.



Fault-Based Attack

- An attack on a **processor** that is generating RSA digital signatures
 - Induces faults in the signature computation by **reducing the power** to the processor
 - The faults cause the software to produce **invalid signatures** which can then be analyzed by the attacker to recover the private key
 - It requires that the attacker have **physical access** to the target machine and is able to directly control the input power to the processor



Chosen Ciphertext Attack (CCA)

- The adversary
 1. chooses a number of ciphertexts
 2. given the corresponding plaintexts, decrypted with the target's private key
 3. select a plaintext
 4. encrypt it with the target's public key
 5. be able to get the plaintext back by having it decrypted with the private key
 6. yield information needed for cryptanalysis



Cont'd

- Based on the property of RSA:

$$E(PU, M_1) \times E(PU, M_2) = E(PU, [M_1 \times M_2])$$

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

1. Compute $X = (C \times 2^e) \bmod n$.
2. Submit X as a chosen ciphertext and receive back $Y = X^d \bmod n$.

$$\begin{aligned} X &= (C \bmod n) \times (2^e \bmod n) \\ &= (M^e \bmod n) \times (2^e \bmod n) \\ &= (2M)^e \bmod n \end{aligned}$$

$$Y = (2M) \bmod n$$

From this, we can deduce M



Cont'd

- Countermeasure:
 - Randomly pad the plaintext prior to encryption. This randomizes the ciphertext so Equation (9.2) no longer holds.
$$E(PU, M_1) \times E(PU, M_2) = E(PU, [M_1 \times M_2])$$
 - Modifying the plaintext using a procedure known as optimal asymmetric encryption padding (OAEP)