Principles of Public-Key Cryptosystems

- The concept of public-key cryptography 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 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 achieved a breakthrough in 1976 by coming up with a method that addressed both problems and was radically different from all previous approaches to cryptography

Public-Key Cryptosystems

- A public-key encryption scheme has six ingredients:

Plaintext

The readable message or data that is fed into the algorithm as input

Encryption algorithm

Performs various transforma-tions on the plaintext

Public key

Used for encryption or decryption

Private key

Used for encryption or decryption

Ciphertext

The scrambled message produced as output

Decryption algorithm

Accepts the ciphertext and the matching key and produces the original plaintext

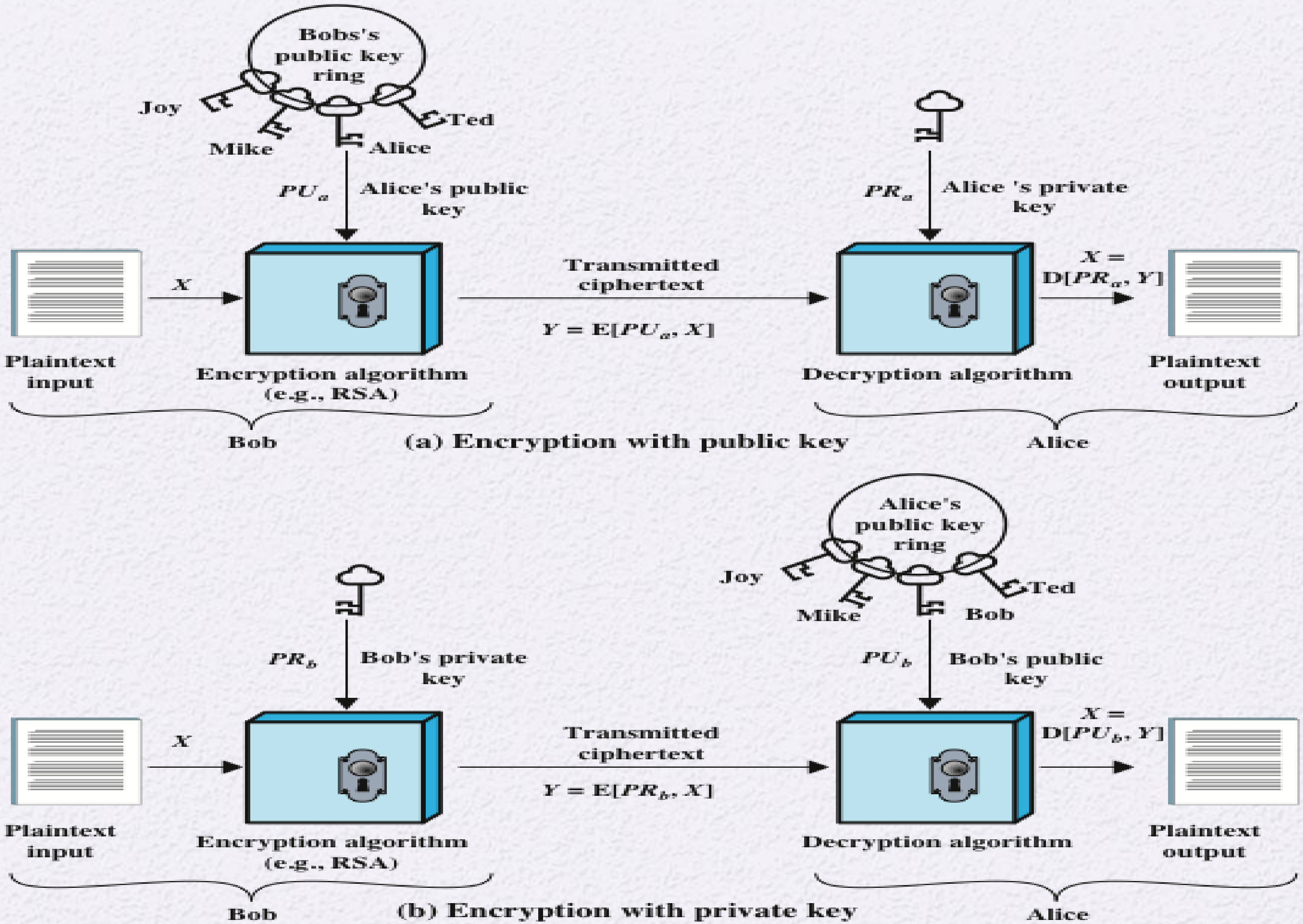


Figure 9.1 Public-Key Cryptography

Table 9.2
Conventional and Public-Key Encryption

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if the key is kept secret. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and 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). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if 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.

Public-Key Cryptosystem: Secrecy

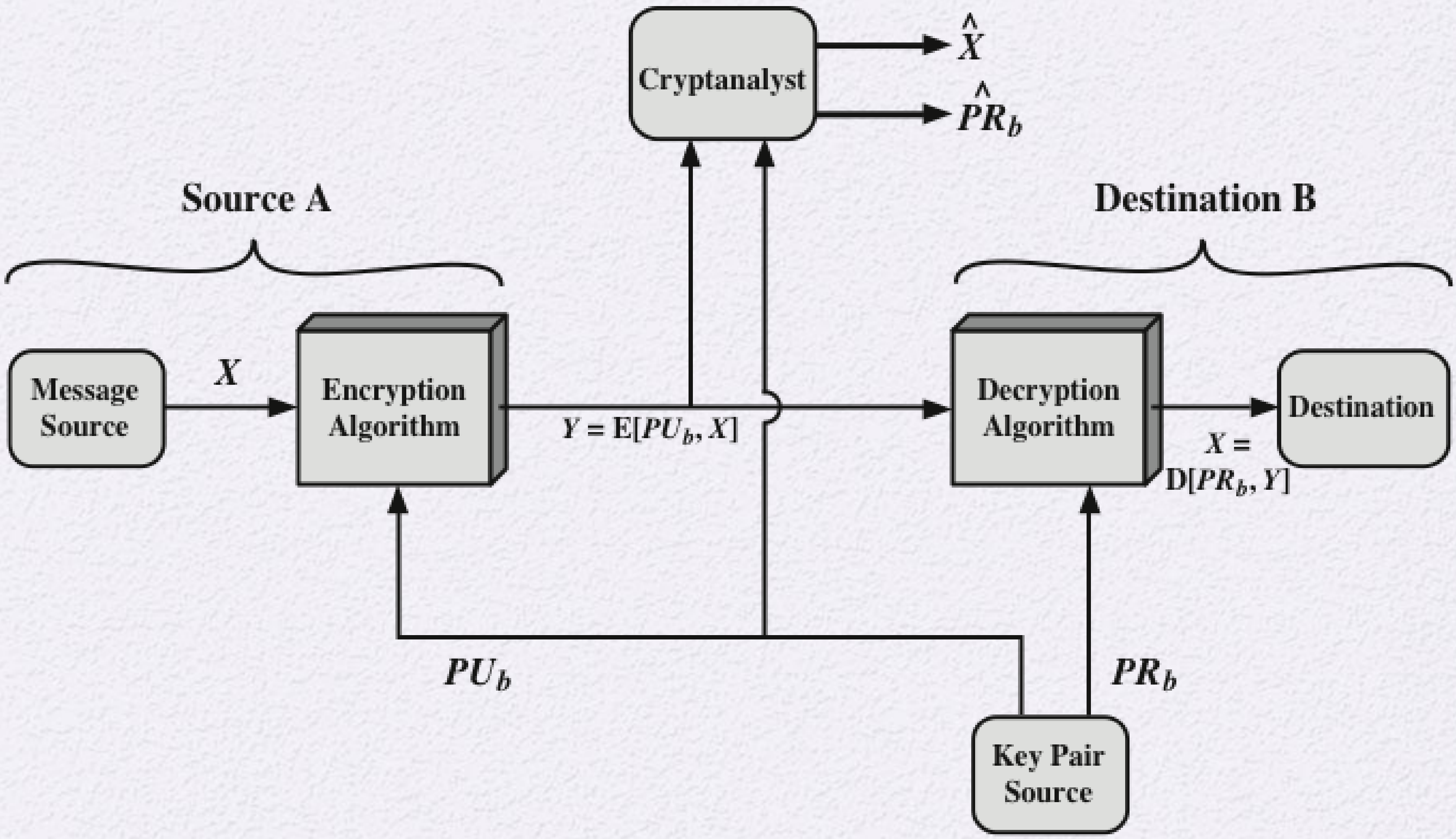


Figure 9.2 Public-Key Cryptosystem: Secrecy

Public-Key Cryptosystem: Authentication

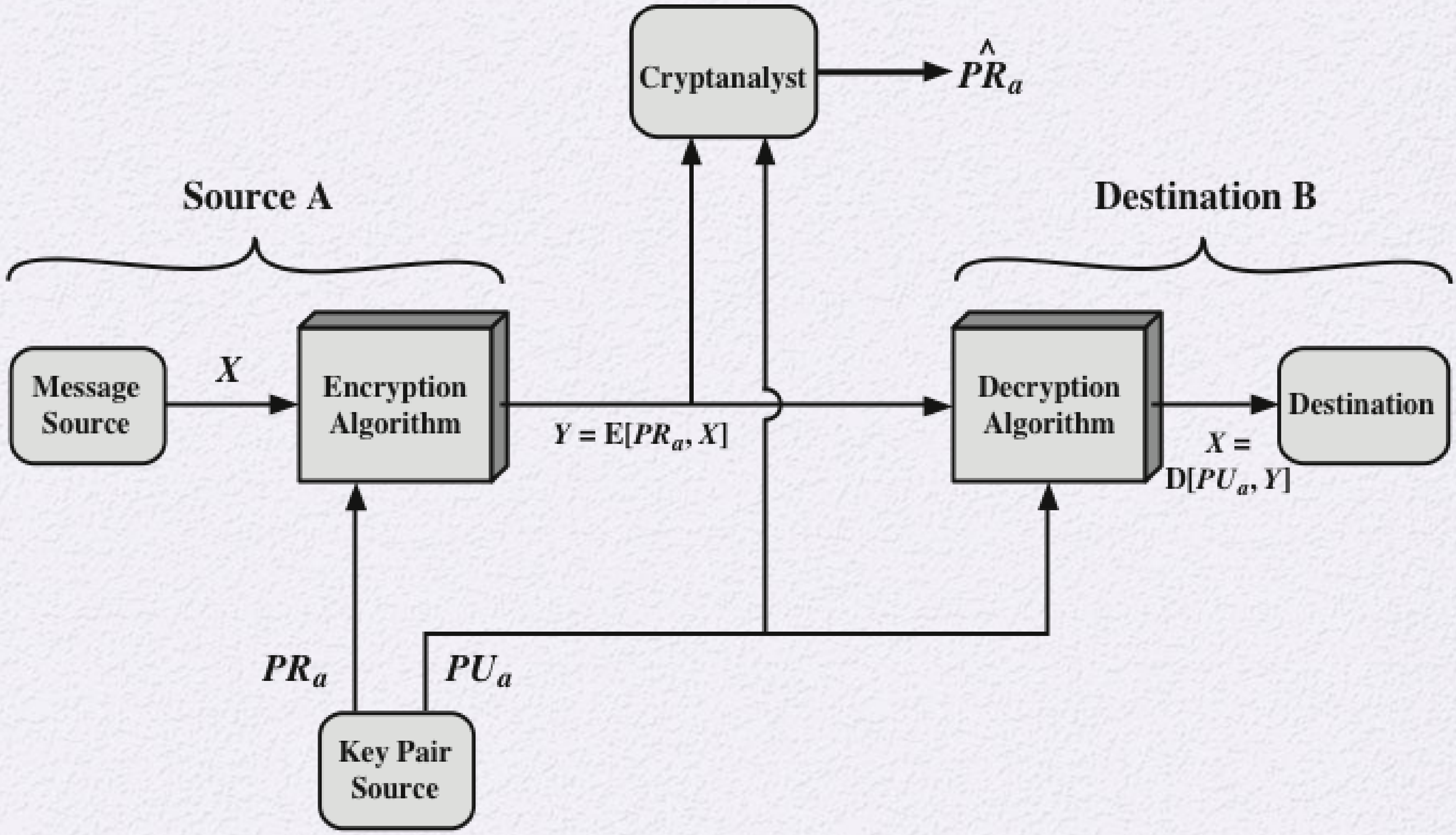


Figure 9.3 Public-Key Cryptosystem: Authentication

Public-Key Cryptosystem: Authentication and Secrecy

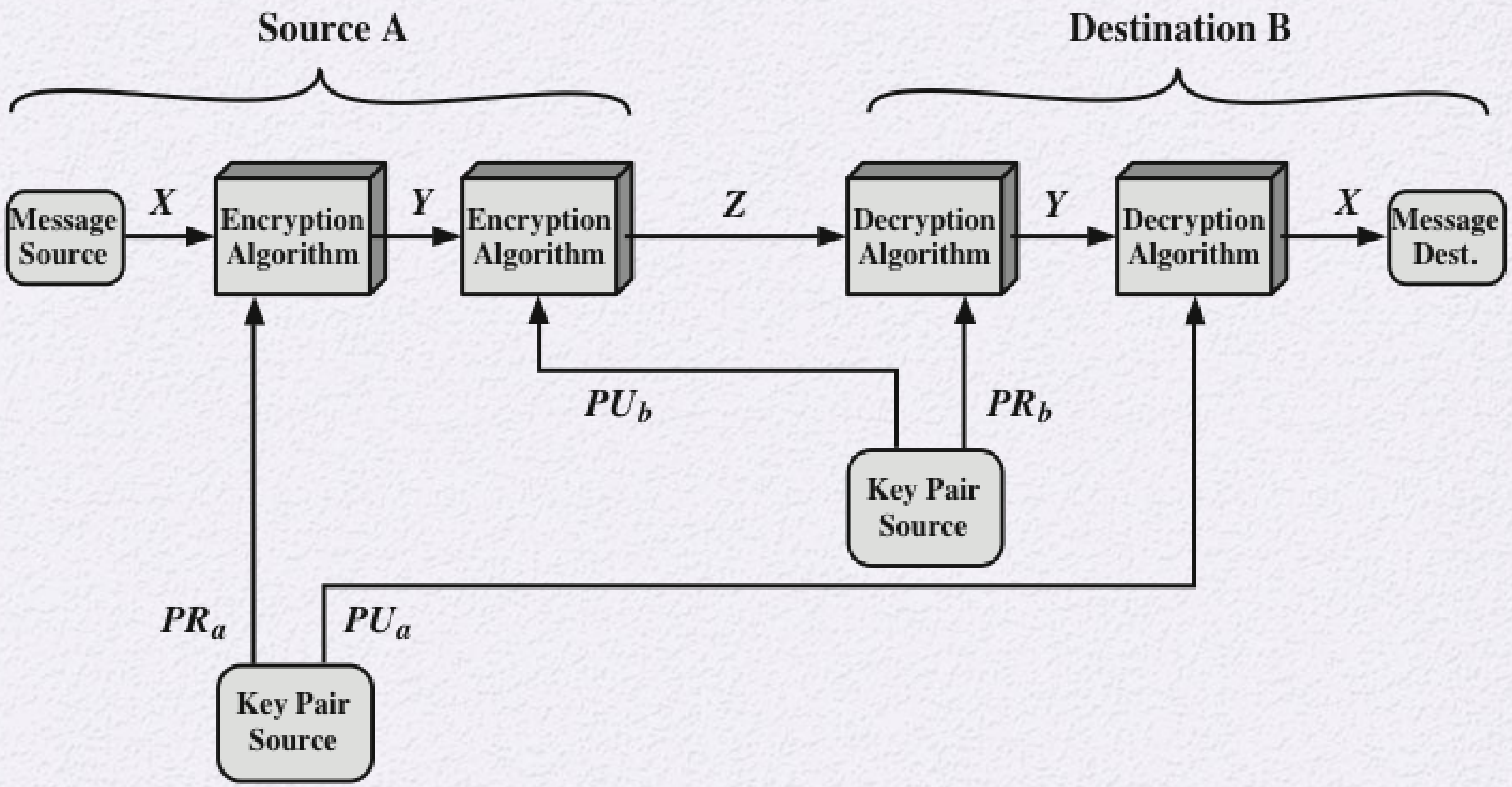


Figure 9.4 Public-Key Cryptosystem: Authentication and Secrecy

Applications for Public-Key Cryptosystems

- Public-key cryptosystems can be classified into three categories:

- **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

- Some algorithms are suitable for all three applications, whereas others can be used only for one or two

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

Table 9.3 Applications for Public-Key Cryptosystems

Public-Key Requirements

- Conditions that these algorithms must fulfill:
  - It is computationally easy for a party B to generate a pair (public-key PU_b, private key PR_b)
  - It is computationally easy for a sender A, knowing the public key and the message to be encrypted, to generate the corresponding ciphertext
  - It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message
  - It is computationally infeasible for an adversary, knowing the public key, to determine the private key
  - It is computationally infeasible for an adversary, knowing the public key and a ciphertext, to recover the original message
  - The two keys can be applied in either order

Mathematical Background

Congruence

- For a positive integer n, two integers a and b are said to be congruent modulo n, written as

$$a \equiv b \pmod{n}$$

- We call it as "a is congruent to b modulo n" which implies that
  - $a-b$ is an integer multiple of n OR
  - n divides $a-b$
- Examples:

$$-8 \equiv 7 \pmod{5}$$
$$2 \equiv -3 \pmod{5}$$
$$-3 \equiv -8 \pmod{5}$$
$$38 \equiv 2 \pmod{12}$$

Relatively Prime

- Two numbers are **relatively prime** if they share only one factor, namely 1.
- For example, 10 and 21 are relatively prime. Neither is prime, but the numbers that evenly divide 10 are 1, 2, 5 and 10, whereas the numbers that evenly divide 21 are 1, 3, 7 and 21.
- The only number in both lists is 1, so the numbers are relatively prime.

Greatest Common Divisor

- If two numbers are relatively prime their GCD is 1.
- m and n are relatively prime means gcd(m, n) = 1
- There is a simple algorithm to calculate the gcd of two integers – Euclidean Algorithm

Example of Euclidean Algorithm

- Calculate the GCD of 1156 and 112

Divisor	Dividend	Quotient	Remainder
112	1156	10	36
36	112	3	4
4	36	9	0

GCD of 1156 and 112

When you get a zero remainder, the remainder before it is the GCD

	1156
2	578
2	289
17	17

	112
2	56
2	28
2	14
2	7

$1156 = 2^2 \times 17^2 = \boxed{2 \times 2} \times 17 \times 17$

$112 = 2^4 \times 7^1 = \boxed{2 \times 2} \times 2 \times 2 \times 7$

Euler's totient Function

- Euler's totient or phi function, $\varphi(n)$
  - counts the number of positive integers less than or equal to n that are relatively prime to n
  - For a prime number **p** $\varnothing(p) = p-1$
  - For prime numbers **p** and **q** $\varnothing(pq)=(p-1)*(q-1)$

<u>Example:</u>

- If $n=9$ then numbers 1, 2, 4, 5, 7 and 8, are relatively prime to 9. Therefore, $\varphi(9)=6$
- If $n=11$ then numbers 1, 2,3, 4, 5,6,7,8,9 and 10, are relatively prime to 11. Therefore, $\varphi(11)=11-1=10$ all numbers less than 11 because 11 is a prime number

Factoring a Number ...

- For example, factoring 15 is simple, it is 3 * 5. But what about 6,320,491,217?
- Now how about a 155-digit number? Or 200 digits or more? In short, factoring numbers takes a certain number of steps, and the number of steps increases subexponentially as the size of the number increases. That means even on supercomputers, if a number is sufficiently large, the time to execute all the steps to factor it would be so great that it could take years to compute.

Public-Key Requirements

- Conditions that these algorithms must fulfill:
  - It is computationally easy for a party B to generate a pair (public-key PU_b, private key PR_b)
  - It is computationally easy for a sender A, knowing the public key and the message to be encrypted, to generate the corresponding ciphertext
  - It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message
  - It is computationally infeasible for an adversary, knowing the public key, to determine the private key
  - It is computationally infeasible for an adversary, knowing the public key and a ciphertext, to recover the original message
  - The two keys can be applied in either order

RSA

- Developed by Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman in 1978
- Consists of:
  - A **Public Key**: the product of two large prime numbers, along with an auxiliary value.
  - A **Private Key**: The prime factors (must be kept secret).
  - **Security**: If the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message.
- Involves three steps:
  - Key generation → Encryption → Decryption

RSA

- RSA - named after Rivest, Shamir and Adleman, the inventors - was the first public-key scheme which was capable of signatures as well as encryption.
- It is the easiest to understand as well as the most popular to implement
- RSA obtains its security from the difficulty of factoring large numbers.

RSA Algorithm - Key Generation

1. First choose two large prime numbers (100's of digits), *p* and *q*, and find their product, *n*. *n* is also called modulus in RSA jargon.
2. Compute $z = (p-1)(q-1)$
3. Next choose a number *e*, relatively prime to $z = (p-1)(q-1)$ - this is the encryption key.
   - $e < n, gcd(e, \phi(n)) = 1$
4. Finally compute *d* such that the product of *e* and *d* is congruent to $1 \bmod ((p-1)(q-1))$. This is the decryption key.
   - $e.d \equiv 1 \bmod \phi(n), 0 < d < n$

RSA Algorithm - Key Generation

5. Obviously, *d* can only be recovered if you reveal *p* and *q*, or if *p* and *q* are recovered from *n*, the modulus. Since we are assuming the factorization of *n* to be too hard to attempt, *d* cannot be recovered from *e*. Or so it is currently speculated. It has not, so far, been proven.
6. Now *e* and *n* together form the public key, while *d* and *n* together form the private key.

RSA Algorithm - Encryption

- To encrypt a plaintext message block M, compute:
  - $C = M^e \bmod n$
- To decrypt the block, compute:
  - $M=C^d \bmod n$
- Each plaintext block must be smaller than the value of n.

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

Figure 9.5 The RSA Algorithm

RSA Example

- $p = 3$
- $q = 11$
- $n = p \times q = 33$ -- This is the *modulus*
- $z = (p\text{-}1) \times (q\ \text{-}1) = 20$ -- This is the totient function $\phi(n)$. There are 20 relative primes to 33. What are they? 1, 2, 4, 5, 7, 8, 10, 13, 14, 16, 17, 19, 20, 23, 25, 26, 28, 29, 31, 32
- $d = 7$ -- 7 and 20 have no common factors but 1
- $7e = 1 \bmod 20$
- $e = 3$
- $C = M^e \pmod{n}$
- $M = C^d \pmod{n}$

RSA Example

Plaintext (M)			Ciphertext (C)		After Decryption	
Symbolic	Numeric	M^e	$M^e (\text{mod } n)$	C^d	$C^d (\text{mod } n)$	Symbolic
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	01	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	05	E

Example of RSA Algorithm

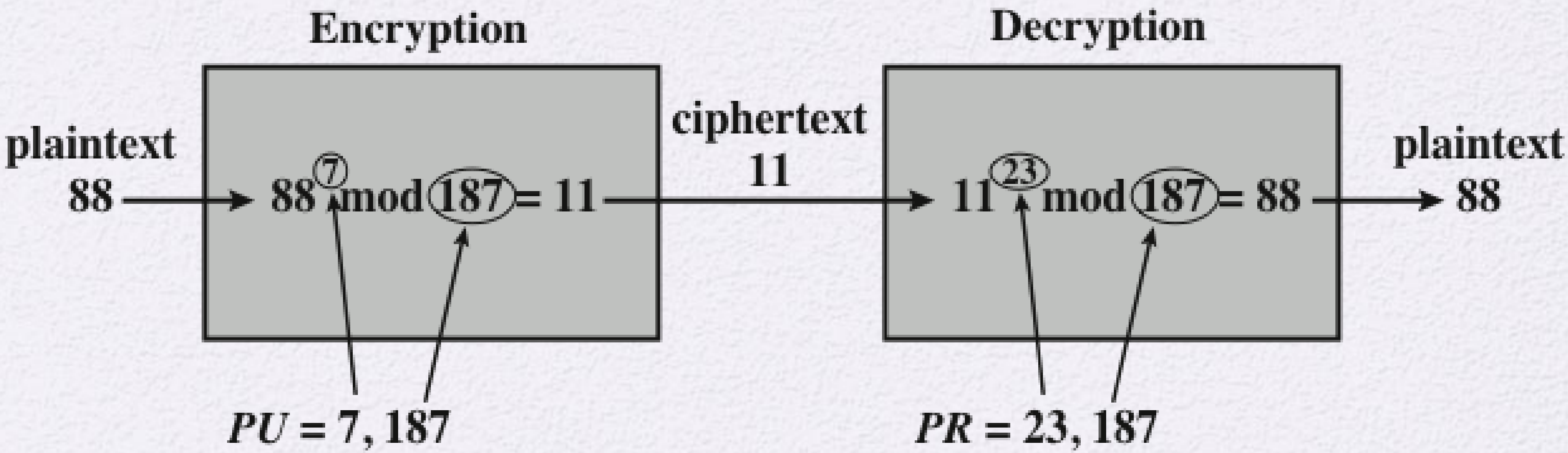
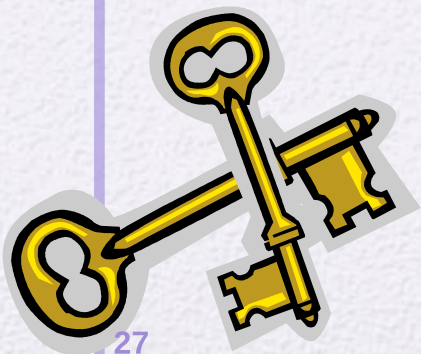


Figure 9.6 Example of RSA Algorithm

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



Procedure for Picking a Prime Number

- Pick an odd integer n at random
- Pick an integer $a < n$ at random
- Perform the probabilistic primality test with a as a parameter. If n fails the test, reject the value n and go to step 1
- If n has passed a sufficient number of tests, accept n; otherwise, go to step 2

The Security of RSA

Five possible approaches to attacking RSA are:

Brute force
- Involves trying all possible private keys

Mathematical attacks
- There are several approaches, all equivalent in effort to factoring the product of two primes

Timing attacks
- These depend on the running time of the decryption algorithm

Hardware fault-based attack
- This involves inducing hardware faults in the processor that is generating digital signatures

Chosen ciphertext attacks
- This type of attack exploits properties of the RSA algorithm

Factoring Problem

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

Timing Attacks

- Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages
- Are applicable not just to RSA but to other public-key cryptography systems
- Are alarming for two reasons:
  - It comes from a completely unexpected direction
  - It is a ciphertext-only attack

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