Computer Network

CSE 303

Network Security

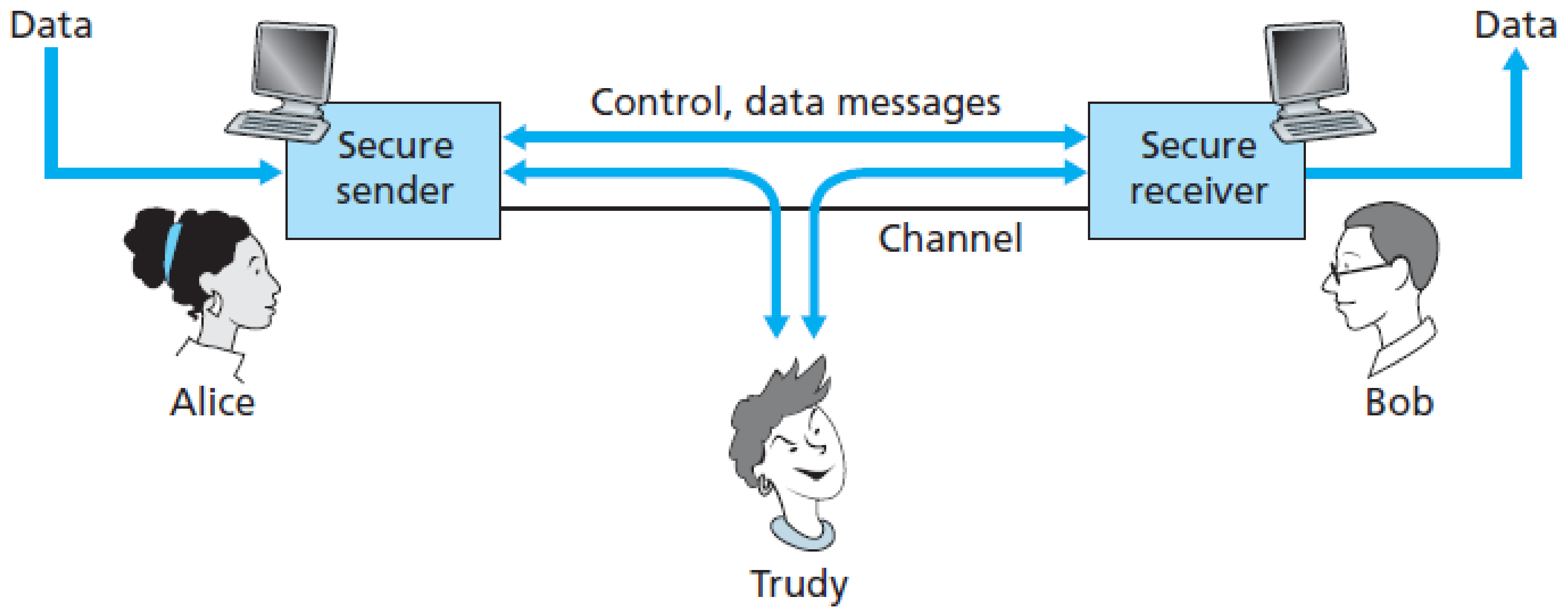


Figure 8.1 ♦ Sender, receiver, and intruder (Alice, Bob, and Trudy)

Network Security

- There are some prevalent and damaging classes of Internet attacks, including malware attacks, denial of service, sniffing, source masquerading, and message modification and deletion.
- Properties of **secure communication:**
  - *Confidentiality*
  - *Message integrity*
  - *End-point authentication*
  - *Operational security*

Network Security

- An intruder can potentially perform:
  - *eavesdropping*—sniffing and recording control and data messages on the channel.
  - *modification, insertion,* or *deletion* of messages or message content.

Principles of Cryptography

- Cryptographic techniques allow a sender to disguise data so that an intruder can gain no information from the intercepted data.
- The receiver, of course, must be able to recover the original data from the disguised data.
- Suppose now that Alice wants to send a message to Bob. Alice's message in its original form (for example, "Bob, I love you. Alice") is known as **plaintext**, or **cleartext**.
- Alice encrypts her plaintext message using an **encryption algorithm** so that the encrypted message, known as **ciphertext**, looks unintelligible to any intruder.

Principles of Cryptography

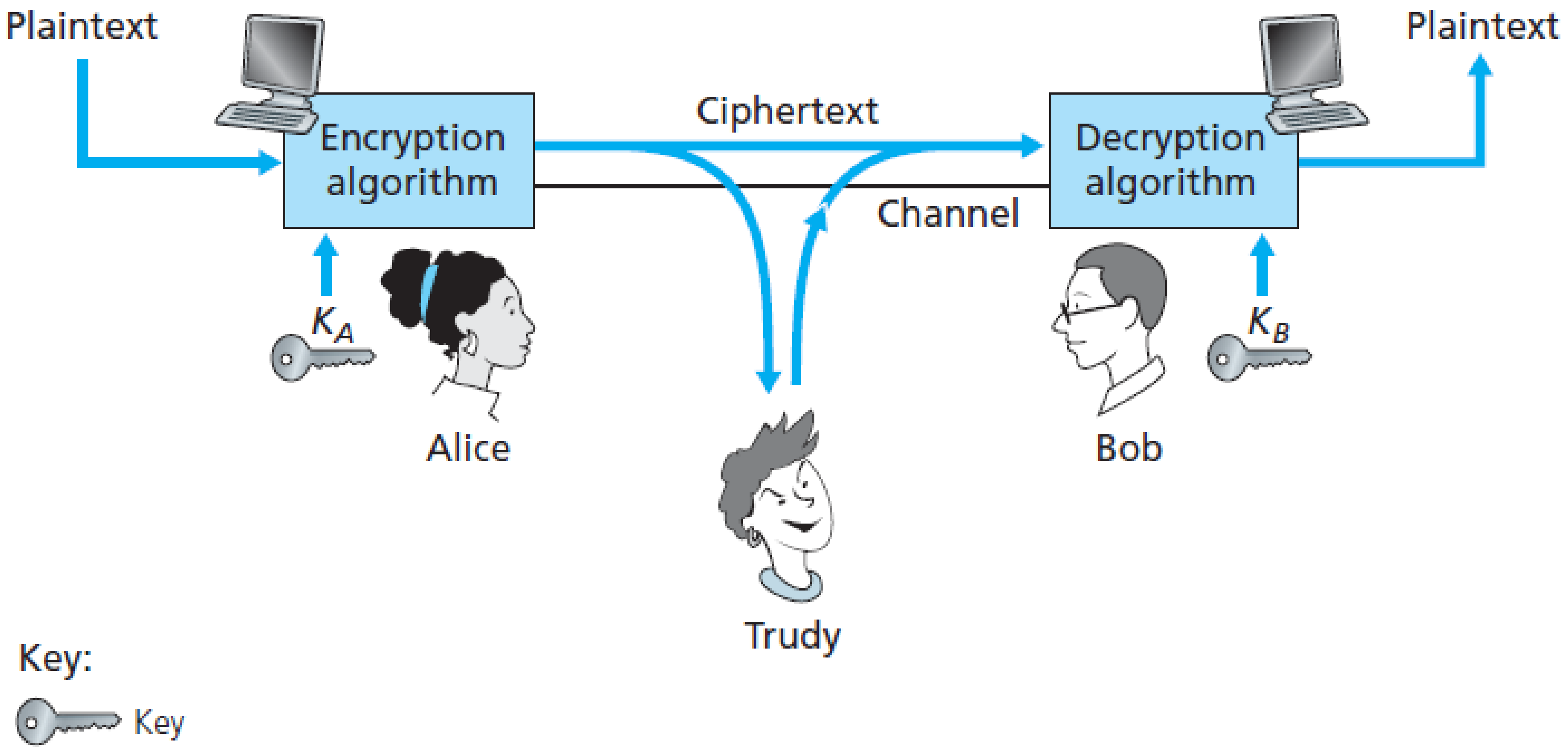


Figure 8.2 ♦ Cryptographic components

Principles of Cryptography

- Alice provides a **key**, K_A, a string of numbers or characters, as input to the encryption algorithm.
- The encryption algorithm takes the key and the plaintext message, *m,* as input and produces ciphertext as output.
- The notation $K_A(m)$ refers to the ciphertext form (encrypted using the key K_A) of the plaintext message, *m*.
- Bob receives an encrypted message *KA*(*m*), he decrypts it by computing $K_B(K_A(m)) = m.$

Principles of Cryptography

- In **symmetric key systems**, Alice's and Bob's keys are identical and are secret.
- In **public key systems**, a pair of keys is used. One of the keys is known to both Bob and Alice (indeed, it is known to the whole world). The other key is known only by either Bob or Alice (but not both).

Symmetric Key Cryptography

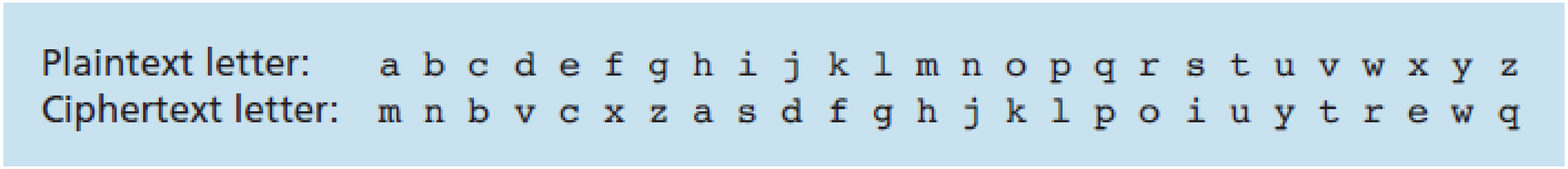
- All cryptographic algorithms involve substituting one thing for another, for example, taking a piece of plaintext and then computing and substituting the appropriate ciphertext to create the encrypted message.
- A very old, very simple symmetric key algorithm attributed to Julius Caesar, known as the **Caesar cipher**
- The Caesar cipher would work by taking each letter in the plaintext message and substituting the letter that is *k* letters later (allowing wraparound; that is, having the letter *z* followed by the letter *a*) in the alphabet.

Symmetric Key Cryptography

- For example if *k* = 3, then the letter *a* in plaintext becomes *d* in ciphertext; *b* in plaintext becomes *e* in ciphertext, and so on. Here, the value of *k* serves as the key.
- As an example, the plaintext message “bob, i love you. alice” becomes “ere, l oryh brx. dolfh” in ciphertext.
- An improvement on the Caesar cipher is the **monoalphabetic cipher**, which also substitutes one letter of the alphabet with another letter of the alphabet.

Symmetric Key Cryptography

- Any letter can be substituted for any other letter, as long as each letter has a unique substitute letter, and vice versa.



Plaintext letter:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext letter:	m	n	b	v	c	x	z	a	s	d	f	g	h	j	k	l	p	o	i	u	y	t	r	e	w	q

Figure 8.3 ♦ A monoalphabetic cipher

- The plaintext message “bob, i love you. alice” becomes “nkn, s gktc wky. mgsbc.”

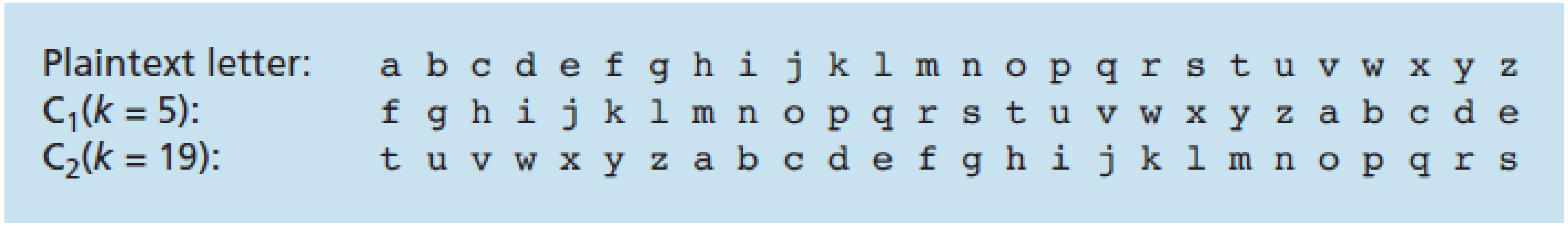
polyalphabetic encryption

- Five hundred years ago, techniques improving on monoalphabetic encryption, known as **polyalphabetic encryption**, were invented.
- The idea behind polyalphabetic encryption is to use multiple monoalphabetic ciphers, with a specific monoalphabetic cipher to encode a letter in a specific position in the plaintext message.
- Thus, the same letter, appearing in different positions in the plaintext message, might be encoded differently.

polyalphabetic encryption

- An example of a polyalphabetic encryption scheme:
  - It has two Caesar ciphers (with *k* = 5 and *k* = 19), shown as rows.
  - We might choose to use these two Caesar ciphers, C1 and C2, in the repeating pattern C1, C2, C2, C1, C2.
  - That is, the first letter of plaintext is to be encoded using C1, the second and third using C2, the fourth using C1, and the fifth using C2.
  - The pattern then repeats, with the sixth letter being encoded using C1, the seventh with C2, and so on.

polyalphabetic encryption



Plaintext letter:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
$C_1(k = 5)$:	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
$C_2(k = 19)$:	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s

Figure 8.4 ♦ A polyalphabetic cipher using two Caesar ciphers

- The plaintext message **“bob, i love you.”** is thus encrypted

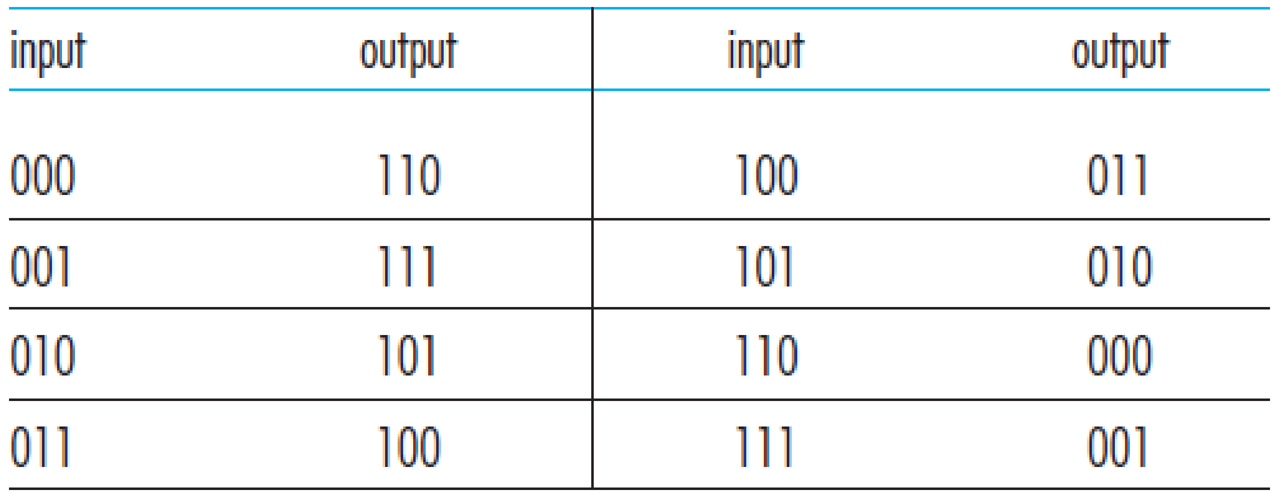
“ghu, n etox dhz.”

Block Ciphers

- There are two broad classes of symmetric encryption techniques:
  - **stream ciphers** and
  - **block ciphers**.
- In a block cipher, the message to be encrypted is processed in blocks of *k* bits.
- For example, if *k* = 64, then the message is broken into 64-bit blocks, and each block is encrypted independently.
- To encode a block, the cipher uses a one-to-one mapping to map the *k*-bit block of cleartext to a *k*-bit block of ciphertext.

Block Ciphers

- Let’s look at an example. Suppose that *k* = 3, so that the block cipher maps 3-bit inputs (cleartext) to 3-bit outputs (ciphertext).



input	output	input	output
000	110	100	011
001	111	101	010
010	101	110	000
011	100	111	001

Table 8.1 ♦ A specific 3-bit block cipher

Block Ciphers

- This block cipher breaks the message up into 3-bit blocks and encrypts each block according to the above mapping.
- You should verify that the message **010**110**001**111 gets encrypted into **101**000**111**001.
- There are 2^3 (= 8) possible inputs (listed under the input columns).
- These eight inputs can be permuted in 8! = 40,320 different ways. Since each of these permutations specifies a mapping, there are 40,320 possible mappings.

Block Ciphers

- With only 40,320 mappings (when k = 3), this can quickly be accomplished on a desktop PC.
- To thwart brute-force attacks, block ciphers typically use much larger blocks, consisting of k = 64 bits or even larger. Note that the number of possible mappings for a general k-block cipher is $2^k!$, which is astronomical for even moderate values of k (such as k = 64).

Block Ciphers

- The function first breaks a 64-bit block into 8 chunks, with each chunk consisting of 8 bits.
- Each 8-bit chunk is processed by an 8-bit to 8-bit table, which is of manageable size.

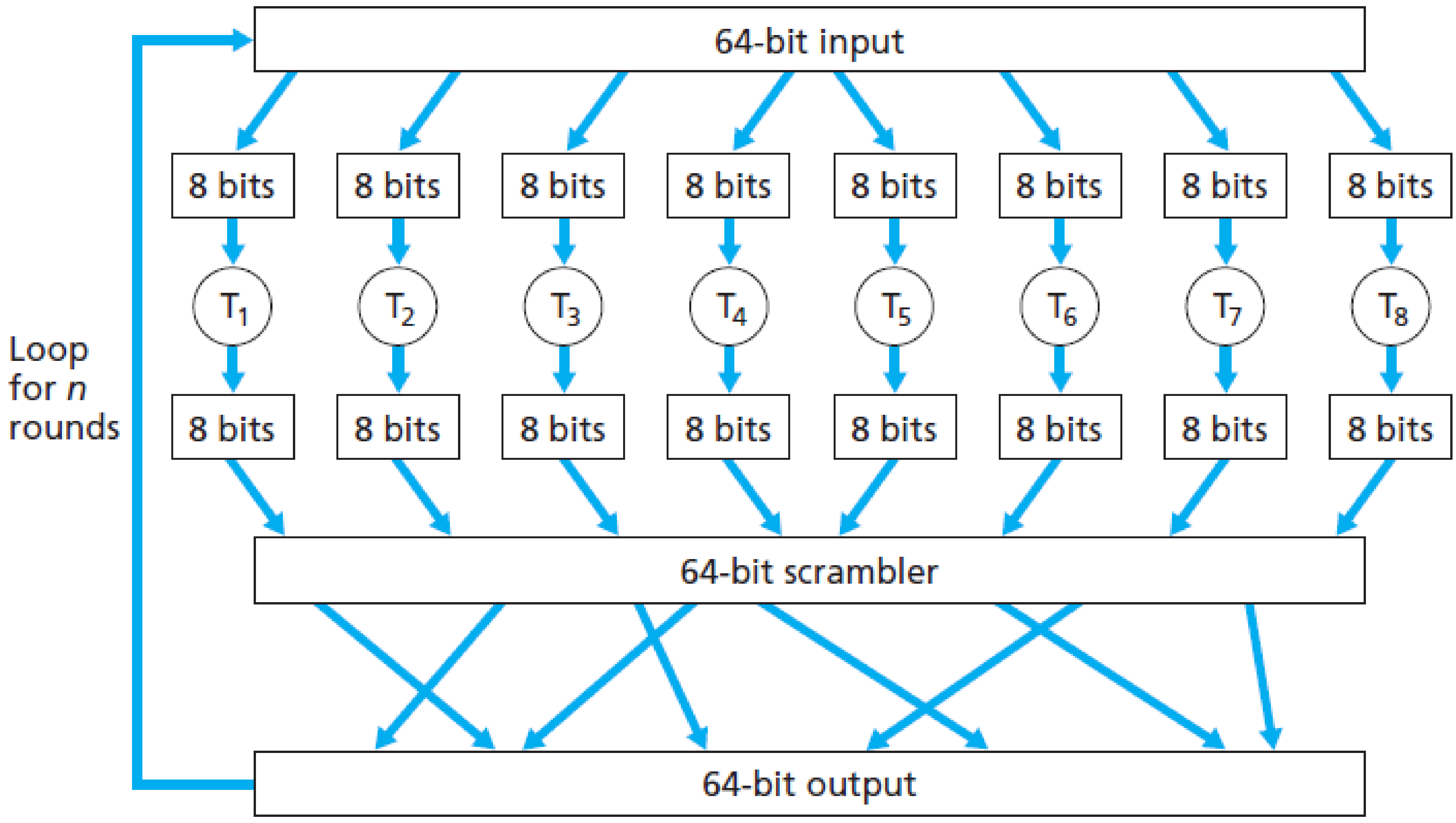


Figure 8.5 ♦ An example of a block cipher

Public Key Encryption

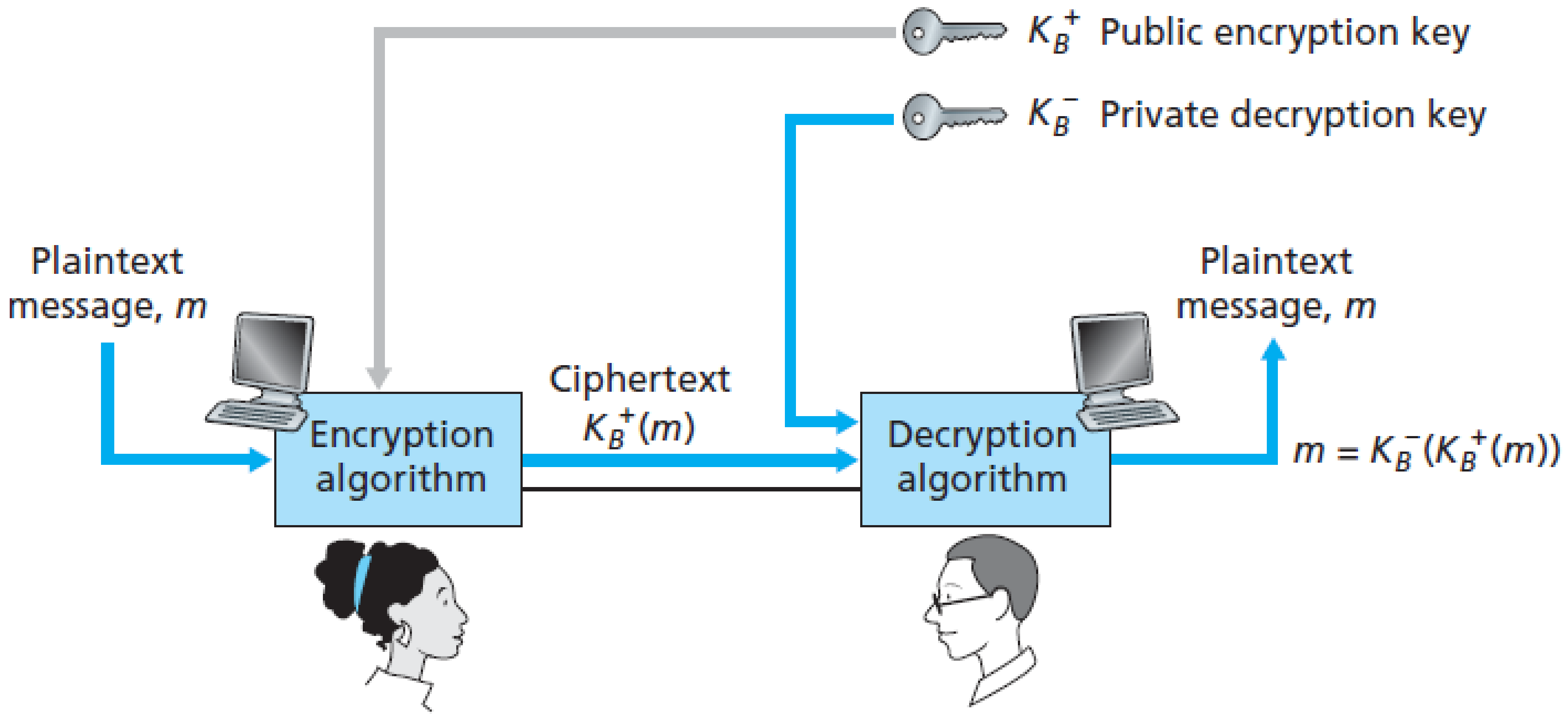


Figure 8.6 ♦ Public key cryptography

Public Key Encryption

- Suppose Alice wants to communicate with Bob.
- Alice sharing a single secret key (as in the case of symmetric key systems).
- Bob has two keys—a **public key** that is available to *everyone* in the world (including Trudy the intruder) and a **private key** that is known only to Bob.
- We will use the notation K_B^+ and K_B^- to refer to Bob’s public and private keys, respectively.
- In order to communicate with Bob, Alice first fetches Bob’s public key.

Public Key Encryption

- Alice then encrypts her message, *m,* to Bob using Bob's public key and a known (for example, standardized) encryption algorithm; that is, Alice computes K_B^+ (m).
- Bob receives Alice's encrypted message and uses his private key and a known (for example, standardized) decryption algorithm to decrypt Alice's encrypted message.
- That is, Bob computes $K_B^-(K_B^+(m))$.

RSA

- While there may be many algorithms that address these concerns,
- The **RSA algorithm** (named after its founders, Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman) has become almost synonymous with public key cryptography.

RSA

- Let’s see how RSA works:
  - RSA makes extensive use of arithmetic operations using modulo-*n* arithmetic.
  - *x* mod *n* simply means the remainder of *x* when divided by *n*; so, for example, 19 mod 5 = 4.
  - In modular arithmetic, one performs the usual operations of addition, multiplication, and exponentiation.

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$$
$$[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$$
$$[(a \bmod n) \bullet (b \bmod n)] \bmod n = (a \bullet b) \bmod n$$

RSA

- for example,
  - [(19 mod 5)+(18 mod 5)] mod 5 = (4+3)mod 5= 7 mod 5 = **2**
  - (19+18) mod 5 = 37 mod 5 = **2**

RSA

- Now suppose that Alice wants to send to Bob an RSA-encrypted message
- let’s always keep in mind that a message is nothing but a bit pattern, and every bit pattern can be uniquely represented by an integer number (along with the length of the bit pattern).
- For example, suppose a message is the bit pattern 1001; this message can be represented by the decimal integer 9. Thus, when encrypting a message with RSA, it is equivalent to encrypting the unique integer number that represents the message.

RSA

- There are two interrelated components of RSA:
  - The choice of the public key and the private key
  - The encryption and decryption algorithm
- To generate the public and private RSA keys, Bob performs the following steps:
  - Choose two large prime numbers, *p* and *q*. How large should *p* and *q* be? The larger the values, the more difficult it is to break RSA, but the longer it takes to perform the encoding and decoding. RSA Laboratories recommends that the product of *p* and *q* be on the order of 1,024 bits.

RSA

- Compute ***n* = *pq*** and ***z* = (*p* – 1)(*q* – 1).**
- Choose a number, ***e,*** less than ***n,*** that has no common factors (other than 1) with ***z.*** (In this case, ***e*** and ***z*** are said to be relatively prime.) The letter ***e*** is used since this value will be used in encryption.
- Find a number, ***d,*** such that ***ed* – 1** is exactly divisible (that is, with no remainder) by ***z***. The letter ***d*** is used because this value will be used in decryption. Put another way, given ***e,*** we choose ***d*** such that

$$ed \bmod z = 1$$

RSA

- The public key that Bob makes available to the world,
  - *KB*+, is the pair of numbers(*n*, *e*); his private key,
  - *KB*–, is the pair of numbers (*n*, *d*).
- The encryption by Alice and the decryption by Bob are done as follows:
  - Suppose Alice wants to send Bob a bit pattern represented by the integer number *m* (with $m < n$). To encode, Alice performs the exponentiation m^e, and then computes the integer remainder when m^e is divided by *n*. In other words, the encrypted value, *c,* of Alice's plaintext message, *m,* is $c = m^e \bmod n$
  - The bit pattern corresponding to this ciphertext *c* is sent to Bob.

RSA

– To decrypt the received ciphertext message, *c,* Bob computes

$$m = cd \bmod n$$

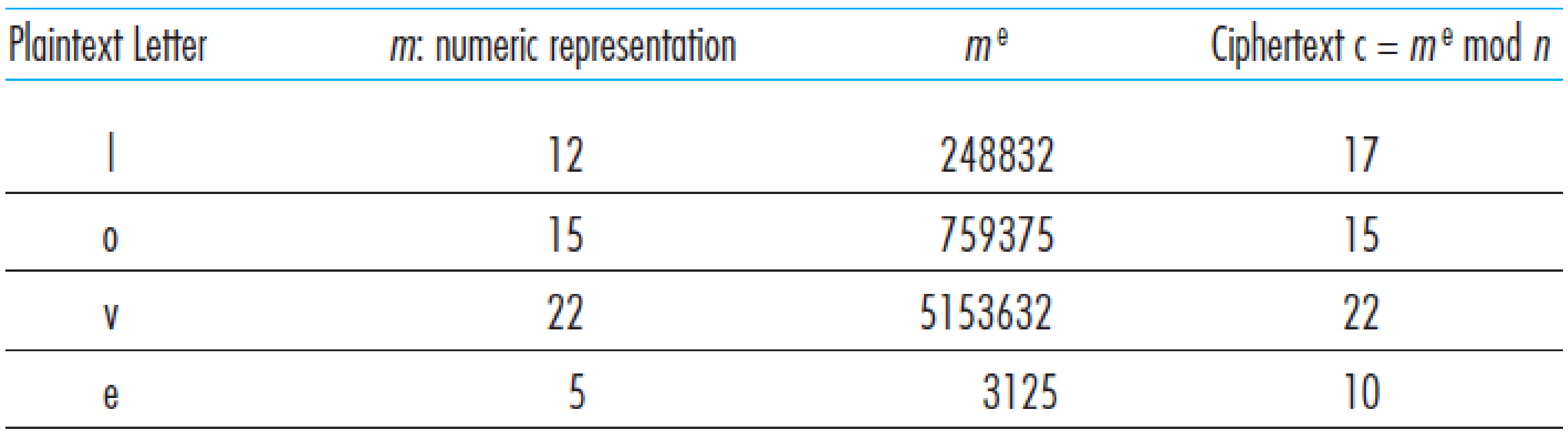
– which requires the use of his private key (*n*,*d*).

RSA

- For example of RSA,
- suppose Bob chooses *p* = 5 and *q* = 7.
- Then *n* = 35 and *z* = 24. Bob
- chooses *e* = 5, since 5 and 24 have no common factors. Finally, Bob chooses *d* = 29, since 5 29 – 1 (that is, *ed* – 1) is exactly divisible by 24.
- Bob makes the two values, *n* = 35 and *e* = 5, public and keeps the value *d* = 29 secret.
- Observing these two public values, suppose Alice now wants to send the letters *l*, *o*, *v*, and *e* to Bob. Interpreting each letter as a number between 1 and 26 (with *a* being 1, and *z* being 26),

RSA

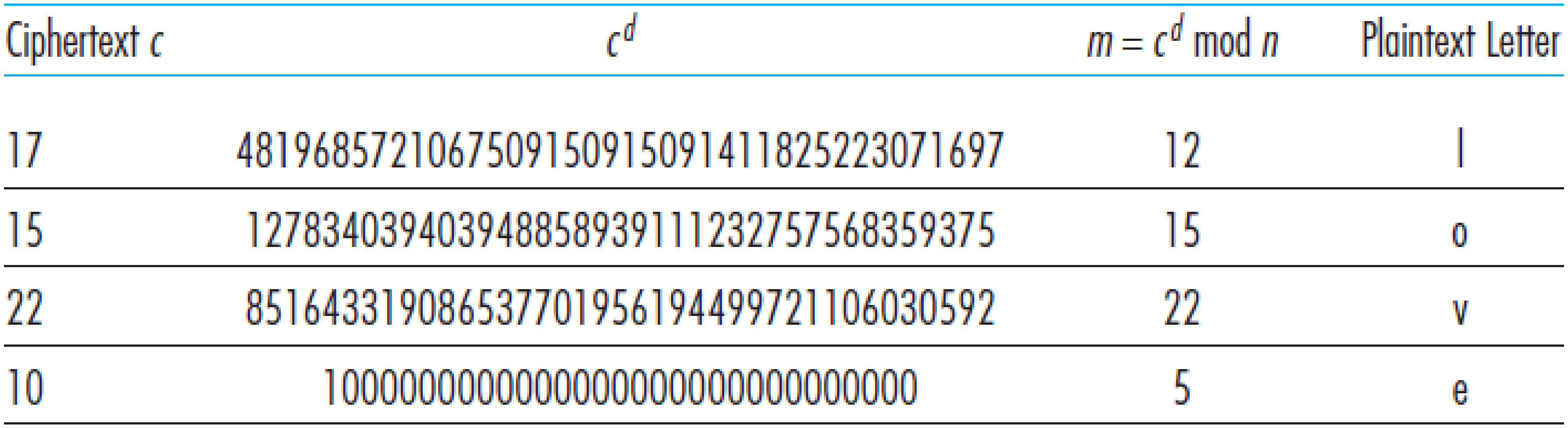
- Alice and Bob perform the following encryption and decryption:



Plaintext Letter	m: numeric representation	m^e	Ciphertext $c = m^e \bmod n$
l	12	248832	17
o	15	759375	15
v	22	5153632	22
e	5	3125	10

Table 8.2 ♦ Alice's RSA encryption, $e = 5$, $n = 35$

RSA



Ciphertext c	c^d	$m = c^d \bmod n$	Plaintext Letter
17	481968572106750915091411825223071697	12	l
15	127834039403948858939111232757568359375	15	o
22	851643319086537701956194499721106030592	22	v
10	100000000000000000000000000000	5	e

Table 8.3 ♦ Bob's RSA decryption, $d = 29$, $n = 35$

SMTP

- SMTP, defined in RFC 5321, is at the heart of Internet electronic mail.
- SMTP transfers messages from senders’ mail servers to the recipients’ mail servers.
- SMTP is much older than HTTP.
- It restricts the body (not just the headers) of all mail messages to simple 7-bit ASCII.
- But today, in the multimedia era, the 7-bit ASCII restriction is a bit of a pain—it requires binary multimedia data to be encoded to ASCII before being sent over SMTP

SMTP

- To illustrate the basic operation of SMTPL: Suppose Alice wants to send Bob a simple ASCII message.
  – Alice invokes her user agent for e-mail, provides Bob’s e-mail address (for example, bob@someschool.edu), composes a message, and instructs the user agent to send the message.
  – Alice’s user agent sends the message to her mail server, where it is placed in a message queue.

SMTP

- The client side of SMTP, running on Alice’s mail server, sees the message in the message queue. It opens a TCP connection to an SMTP server, running on Bob’s mail server.
- After some initial SMTP handshaking, the SMTP client sends Alice’s message into the TCP connection.
- At Bob’s mail server, the server side of SMTP receives the message. Bob’s mail server then places the message in Bob’s mailbox.
- Bob invokes his user agent to read the message at his convenience.

SMTP

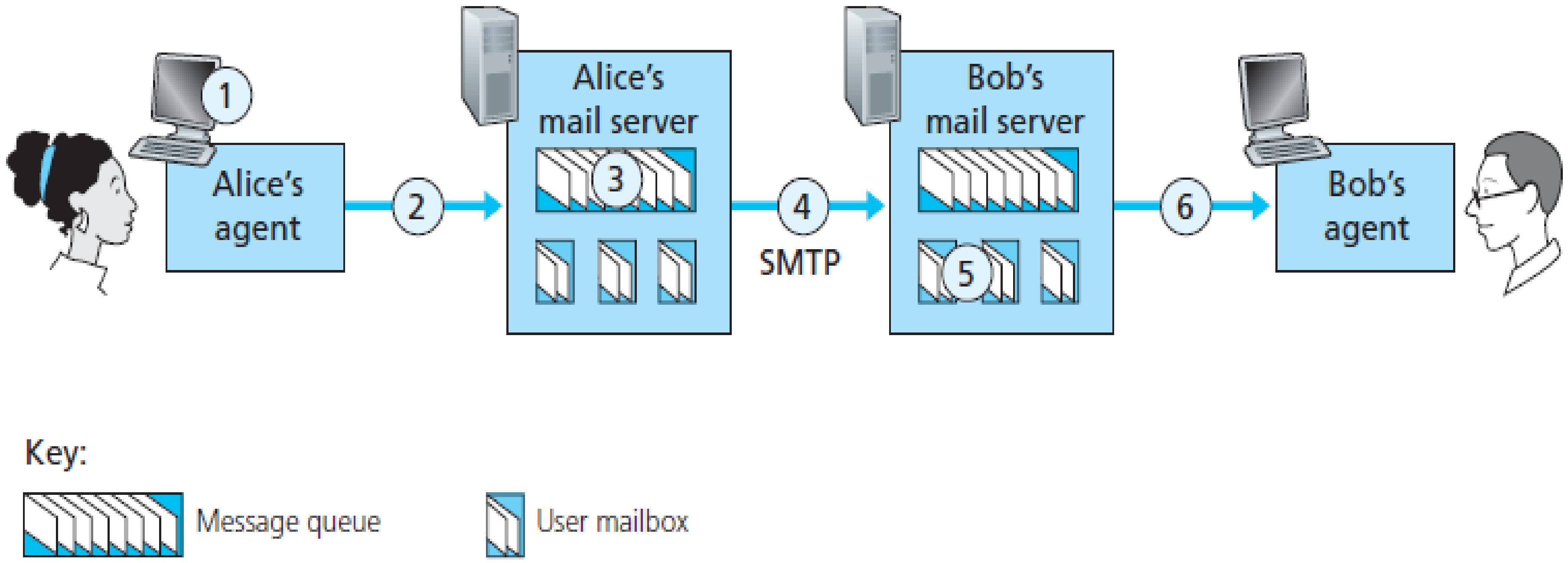


Figure 2.17 ♦ Alice sends a message to Bob

mail access protocols

- There are currently a number of popular mail access protocols, including
  - **Post Office Protocol (POP)**
  - **Post Office Protocol (POP)—Version 3 (POP3)**
  - **Internet Mail Access Protocol (IMAP)**

END