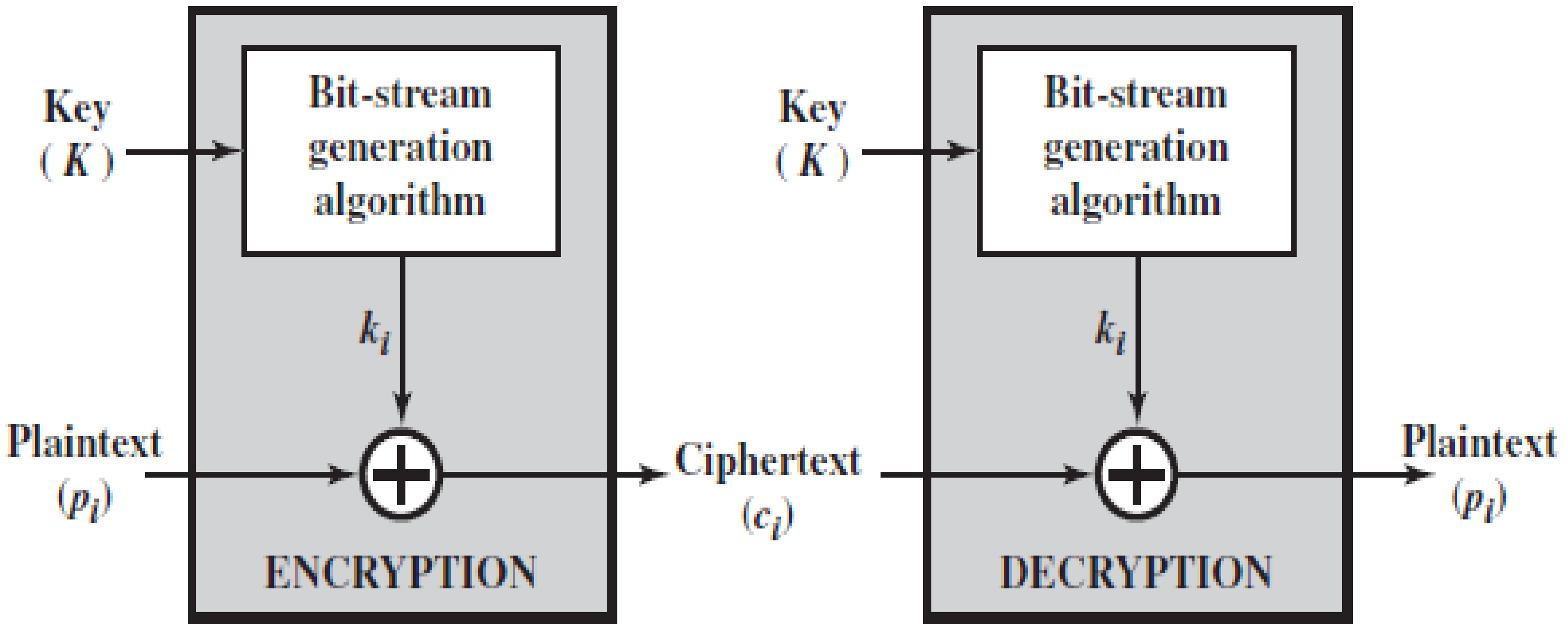
Conventional Symmetric Key Encryption

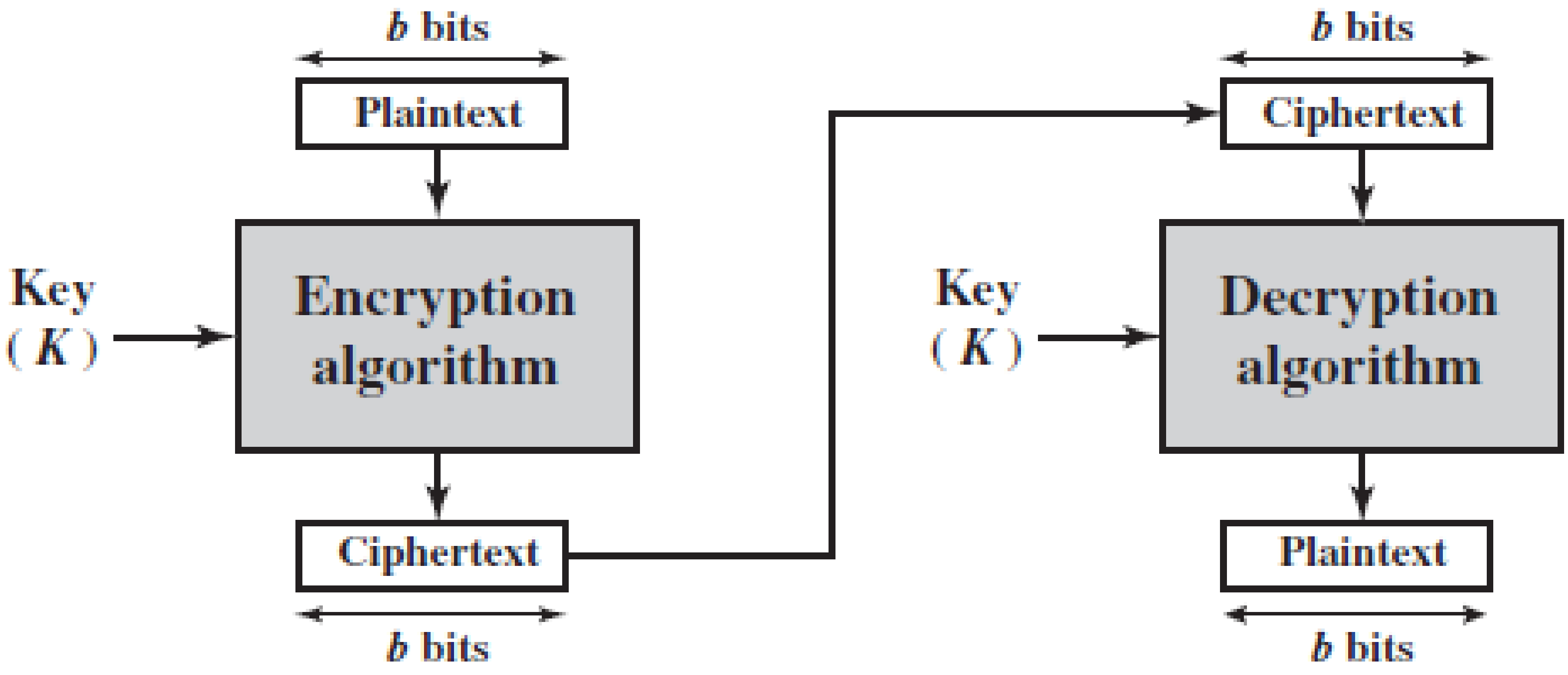
Stream Ciphers and Block Ciphers

- A stream cipher is one that encrypts a digital data stream one bit or one byte at a time.
- Examples of classical stream ciphers are the autokeyed Vigenère cipher and the Vernam cipher.
- In this approach (Figure), the bit-stream generator is a key-controlled algorithm and must produce a bit stream that is cryptographically strong.



(a) Stream cipher using algorithmic bit-stream generator

- A block cipher is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length.
- Typically, a block size of 64 or 128 bits is used. As with a stream cipher, the two users share a symmetric encryption key (Figure).
- In general, they seem applicable to a broader range of applications than stream ciphers.



(b) Block cipher

Diffusion and Confusion

- In diffusion, the statistical structure of the plaintext is dissipated into long-range statistics of the ciphertext.
- This is achieved by having each plaintext digit affect the value of many ciphertext digits; generally, this is equivalent to having each ciphertext digit be affected by many plaintext digits.

- An example of diffusion is to encrypt a message M = m1, m2, m3, . . . of characters with an averaging operation:

$$y_n = \left(\sum_{i=1}^{k} m_{n+i} \right) \bmod 26$$

adding k successive letters to get a ciphertext letter y_n.

- In a binary block cipher, diffusion can be achieved by repeatedly performing some permutation on the data followed by applying a function to that permutation.

- An example of diffusion is to encrypt a message M = m1, m2, m3, . . . of characters with an averaging ope

$$y_n = \left(\sum_{i=1}^{k} m_{n+i} \right) \bmod 26$$

adding k successive letters to get a ciphertext letter y_n.

- In a binary block cipher, diffusion can be achieved by repeatedly performing some permutation on the data followed by applying a function to that permutation.

- The diffusion mechanism make the statistical relationship between the plaintext and ciphertext as complex as possible in order to prevent attempts to deduce the key.
- Confusion seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, to prevent attempts to discover the key.
- This is achieved by the use of a complex substitution algorithm.

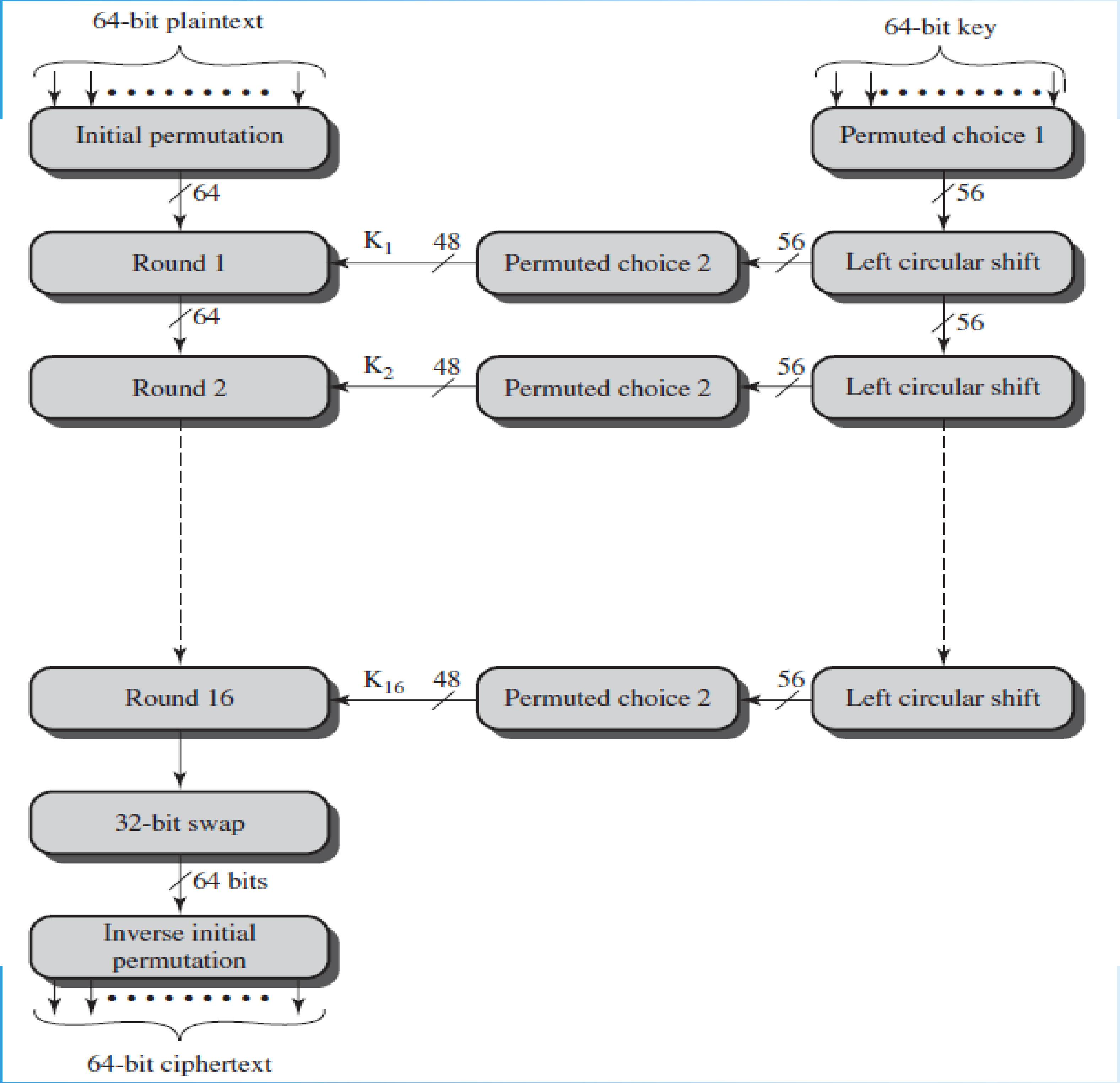
The Data Encryption Standard

- Until the introduction of the Advanced Encryption Standard (AES) in 2001, the Data Encryption Standard (DES) was the most widely used encryption scheme.
- DES was issued in 1977 by the National Bureau of Standards.
- The algorithm itself is referred to as the Data Encryption Algorithm (DEA). For DEA, data are encrypted in 64-bit blocks using a 56-bit key.

- The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption.

DES Encryption

- The overall scheme for DES encryption is illustrated in Figure.


64-bit plaintext
64-bit key
Initial permutation
Permuted choice 1
64
56
Round 1
K1
48
Permuted choice 2
56
Left circular shift
64
56
Round 2
K2
48
Permuted choice 2
56
Left circular shift
Round 16
K16
48
Permuted choice 2
56
Left circular shift
32-bit swap
64 bits
Inverse initial permutation
64-bit ciphertext

- The processing of the plaintext proceeds in three phases.
- First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.
- This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions.

- The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key.
- The left and right halves of the output are swapped to produce the *preoutput*.
- Finally, the *preoutput* is passed through a permutation [IP^{-1}] that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.

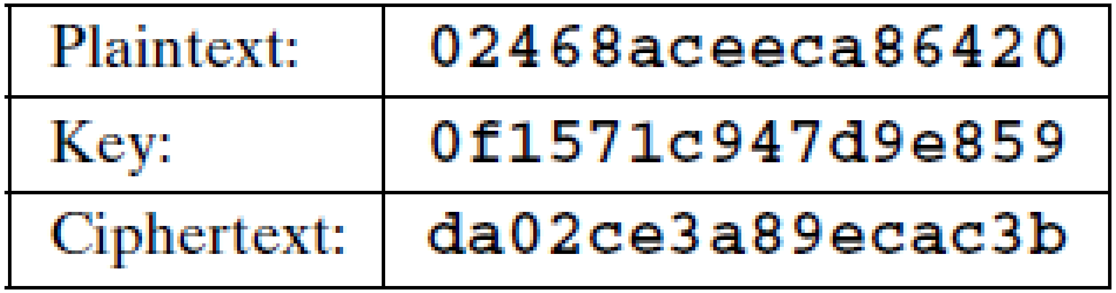
- The right-hand portion of Figure shows the way in which the 56-bit key is used.
- Initially, the key is passed through a permutation function.
- Then, for each of the sixteen rounds, a subkey (K_i) is produced by the combination of a left circular shift and a permutation.
- The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

DES Decryption

- Decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed.
- Additionally, the initial and final permutations are reversed.

DES Example

- For this example, the plaintext is a hexadecimal palindrome.



Plaintext:	02468aceeca86420
Key:	0f1571c947d9e859
Ciphertext:	da02ce3a89ecac3b

- Table shows the progression of the algorithm. The first row shows the 32-bit values of the left and right halves of data after the initial permutation.

Round	K_i	L_i	R_i
IP		5a005a00	3cf03c0f
1	1e030f03080d2930	3cf03c0f	bad22845
2	0a31293432242318	bad22845	99e9b723
3	23072318201d0c1d	99e9b723	0bae3b9e
4	05261d3824311a20	0bae3b9e	42415649
5	3325340136002c25	42415649	18b3fa41
6	123a2d0d04262a1c	18b3fa41	9616fe23
7	021f120b1c130611	9616fe23	67117cf2
8	1c10372a2832002b	67117cf2	c11bfc09
9	04292a380c341f03	c11bfc09	887fbc6c
10	2703212607280403	887fbc6c	600f7e8b
11	2826390c31261504	600f7e8b	f596506e
12	12071c241a0a0f08	f596506e	738538b8
13	300935393c0d100b	738538b8	c6a62c4e
14	311e09231321182a	c6a62c4e	56b0bd75
15	283d3e0227072528	56b0bd75	75e8fd8f
16	2921080b13143025	75e8fd8f	25896490
IP^{-1}		da02ce3a	89ecac3b

Note: DES subkeys are shown as eight 6-bit values in hex format

- The next 16 rows show the results after each round. Also shown is the value of the 48-bit subkey generated for each round.
- Note that $L_i = R_{i-1}$.
- The final row shows the left- and right-hand values after the inverse initial permutation, forming the ciphertext.

The Avalanche Effect

- A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext.
- In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext.
- This is referred to as the avalanche effect.

- Using the previous example, Table 3.3 shows the result when the fourth bit of the plaintext is changed, so that the plaintext is 12468aceeca86420.
- The second column of the table shows the intermediate 64-bit values at the end of each round for the two plaintexts.
- The third column shows the number of bits that differ between the two intermediate values.

Table 3.3 Avalanche Effect in DES: Change in Plaintext

Round		δ
	02468aceeca86420 12468aceeca86420	1
1	3cf03c0fbad22845 3cf03c0fbad32845	1
2	bad2284599e9b723 bad3284539a9b7a3	5
3	99e9b7230bae3b9e 39a9b7a3171cb8b3	18
4	0bae3b9e42415649 171cb8b3ccaca55e	34
5	4241564918b3fa41 ccaca55ed16c3653	37
6	18b3fa419616fe23 d16c3653cf402c68	33
7	9616fe2367117cf2 cf402c682b2cefbc	32
8	67117cf2c11bfc09 2b2cefbc99f91153	33
9	c11bfc09887fbc6c 99f911532eed7d94	32
10	887fbc6c600f7e8b 2eed7d94d0f23094	34
11	600f7e8bf596506e d0f23094455da9c4	37
12	f596506e738538b8 455da9c47f6e3cf3	31
13	738538b8c6a62c4e 7f6e3cf34bc1a8d9	29
14	c6a62c4e56b0bd75 4bc1a8d91e07d409	33
15	56b0bd7575e8fd8f 1e07d4091ce2e6dc	31
16	75e8fd8f25896490 1ce2e6dc365e5f59	32
IP^{-1}	da02ce3a89ecac3b 057cde97d7683f2a	32

- Table 3.4 shows a similar test using the original plaintext of with two keys that differ in only the fourth bit position: the original key, 0f1571c947d9e859, and the altered key, 1f1571c947d9e859.
- Again, the results show that about half of the bits in the ciphertext differ and that the avalanche effect is pronounced after just a few rounds.

Table 3.4 Avalanche Effect in DES: Change in Key

Round		δ
	02468aceeca86420 02468aceeca86420	0
1	3cf03c0fbad22845 3cf03c0f9ad628c5	3
2	bad2284599e9b723 9ad628c59939136b	11
3	99e9b7230bae3b9e 9939136b768067b7	25
4	0bae3b9e42415649 768067b75a8807c5	29
5	4241564918b3fa41 5a8807c5488dbe94	26
6	18b3fa419616fe23 488dbe94aba7fe53	26
7	9616fe2367117cf2 aba7fe53177d21e4	27
8	67117cf2c11bfc09 177d21e4548f1de4	32
9	c11bfc09887fbc6c 548f1de471f64dfd	34
10	887fbc6c600f7e8b 71f64dfd4279876c	36
11	600f7e8bf596506e 4279876c399fdc0d	32
12	f596506e738538b8 399fdc0d6d208dbb	28
13	738538b8c6a62c4e 6d208dbbb9bdeeaa	33
14	c6a62c4e56b0bd75 b9bdeeaad2c3a56f	30
15	56b0bd7575e8fd8f d2c3a56f2765c1fb	33
16	75e8fd8f25896490 2765c1fb01263dc4	30
IP^{-1}	da02ce3a89ecac3b ee92b50606b62b0b	30

The Strength of DES

The Use of 56-Bit Keys

- With a key length of 56 bits, there are 2^{56} possible keys, which is approximately $7.2 * 10^{16}$ keys. Thus a brute-force attack appears impractical.
- Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years to break the cipher.

Table 3.5 Average Time Required for Exhaustive Key Search

Key Size (bits)	Cipher	Number of Alternative Keys	Time Required at 10^9 Decryptions/s	Time Required at 10^{13} Decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	2^{55} ns $= 1.125$ years	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	2^{127} ns $= 5.3 \times 10^{21}$ years	5.3×10^{17} years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	2^{167} ns $= 5.8 \times 10^{33}$ years	5.8×10^{29} years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	2^{191} ns $= 9.8 \times 10^{40}$ years	9.8×10^{36} years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	2^{255} ns $= 1.8 \times 10^{60}$ years	1.8×10^{56} years
26 characters (permutation)	Monoalphabetic	$2! = 4 \times 10^{26}$	2×10^{26} ns $= 6.3 \times 10^{9}$ years	6.3×10^{6} years

The Nature of DES Algorithm

- Another concern is the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm.
- The focus of concern has been on the eight substitution tables, or S-boxes, that are used in each iteration.
- Because the design criteria for these boxes, no one has so far succeeded in discovering the fatal weaknesses in the S-boxes.

Timing Attacks

- A timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts.
- A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs.

Block Cipher Design Principles

Number of Rounds

- The greater the number of rounds, the more difficult it is to perform cryptanalysis.
- In general, the criterion should be that the number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple brute-force key search attack.
- This criterion was certainly used in the design of DES.

Design of S-box

- We would like the algorithm to have good avalanche properties.
- Recall that, this means a change in one bit of the input should produce a change in many bits of the output.
- The strict avalanche criterion (SAC) states that any output bit j of an S-box should change with probability 1/2 when any single input bit i is inverted for all i, j.

- The bit independence criterion (BIC) states that output bits j and k should change independently when any single input bit i is inverted for all i, j, and k.
- The SAC and BIC criteria appear to strengthen the effectiveness of the confusion function.

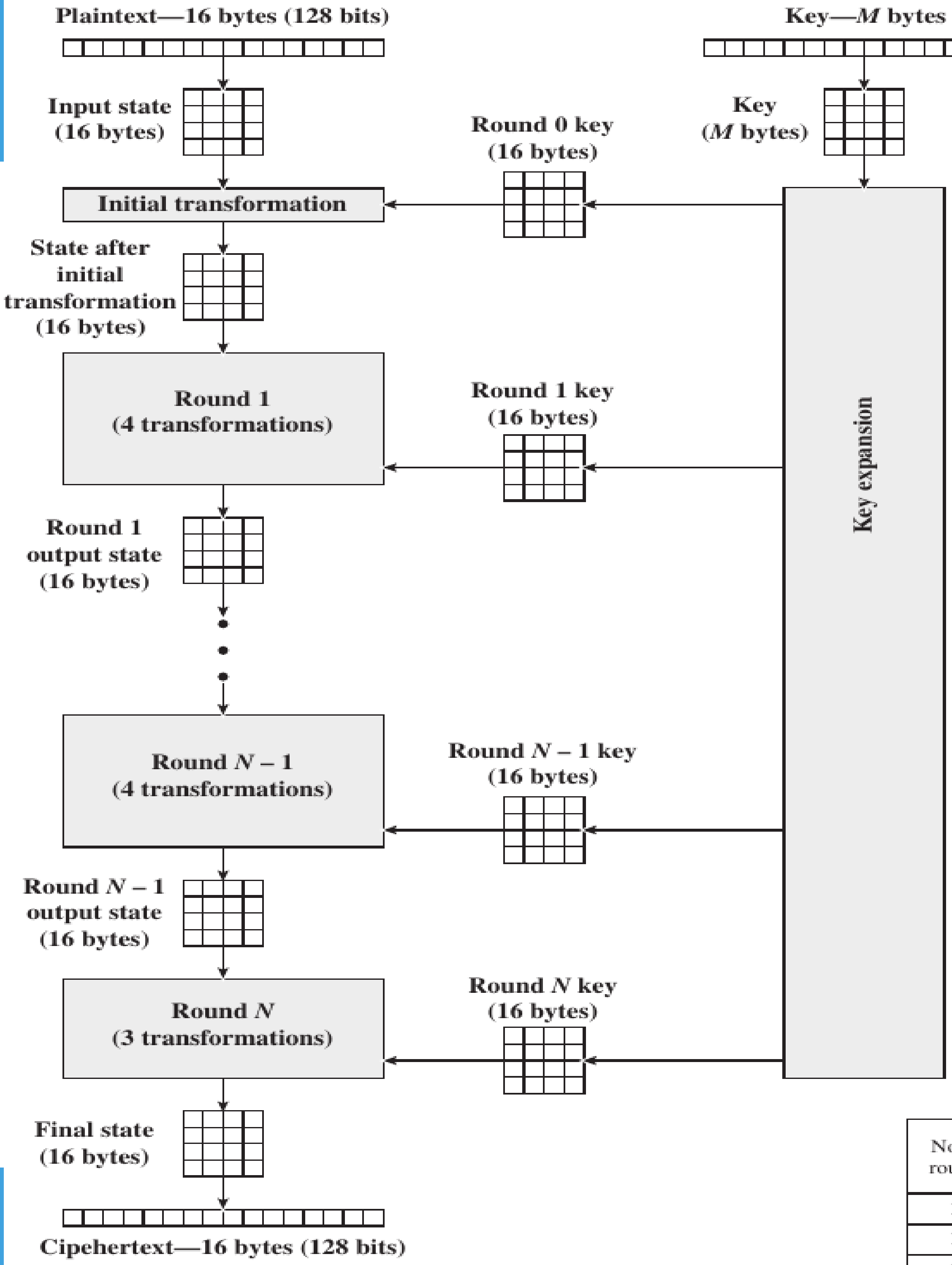
Key Schedule Algorithm

- With any block cipher, the key is used to generate one subkey for each round.
- In general, we would like to select subkeys to maximize the difficulty of deducing individual subkeys and the difficulty of working back to the main key.

Advanced Encryption Standard

General Structure

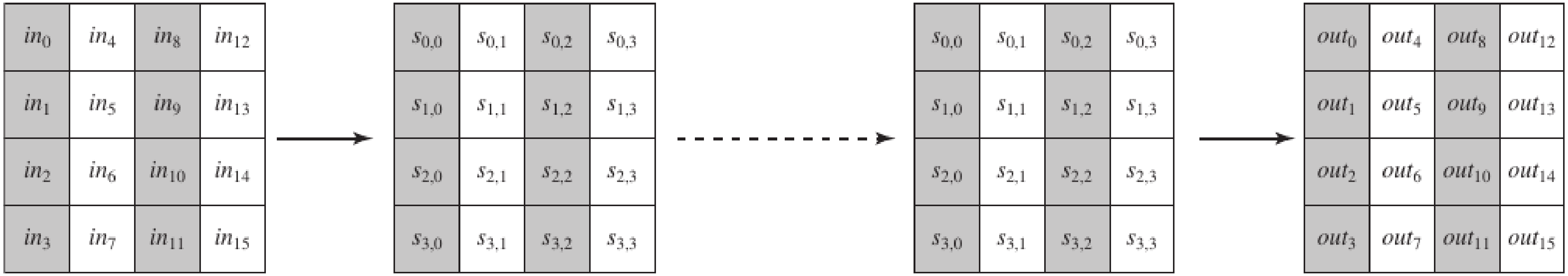
- Figure 5.1 shows the overall structure of the AES encryption process.
- The cipher takes a plaintext block size of 128 bits, or 16 bytes.
- The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length.
- The input to the encryption and decryption algorithms is a single 128-bit block. This block is depicted as a 4 * 4 square matrix of bytes.


Plaintext—16 bytes (128 bits)
Key—M bytes
Input state (16 bytes)
Key (M bytes)
Round 0 key (16 bytes)
Initial transformation
State after initial transformation (16 bytes)
Round 1 (4 transformations)
Round 1 key (16 bytes)
Key expansion
Round 1 output state (16 bytes)
Round N – 1 (4 transformations)
Round N – 1 key (16 bytes)
Round N – 1 output state (16 bytes)
Round N (3 transformations)
Round N key (16 bytes)
Final state (16 bytes)
Cipehertext—16 bytes (128 bits)

No. of rounds	Key Length (bytes)
10	16
12	24
14	32

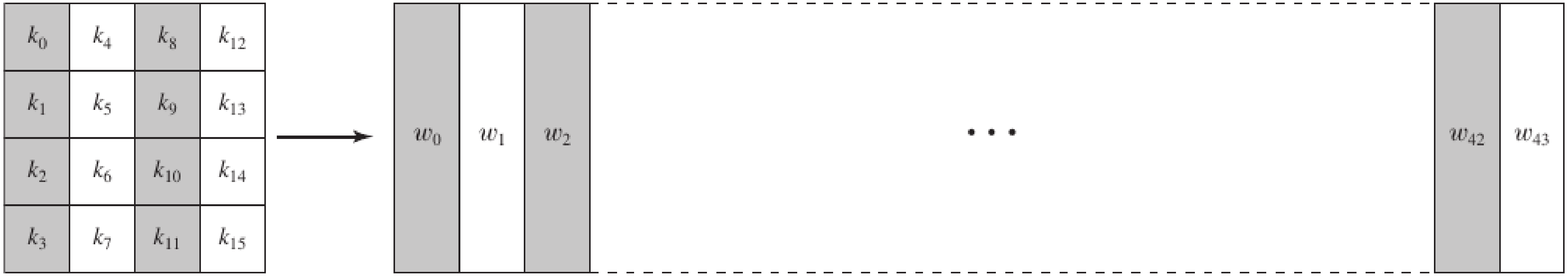
Figure 5.1 AES Encryption Process

- This block is copied into the **State** array, which is modified at each stage of encryption or decryption. After the final stage, **State** is copied to an output matrix (Figure).
- Similarly, the key is depicted as a square matrix of bytes.
- This key is then expanded into an array of key schedule words (Figure). Each word is four bytes, and the total key schedule is 44 words for the 128-bit key.



in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 in_{11} in_{15}
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
out_0 out_4 out_8 out_{12}
out_1 out_5 out_9 out_{13}
out_2 out_6 out_{10} out_{14}
out_3 out_7 out_{11} out_{15}

(a) Input, state array, and output

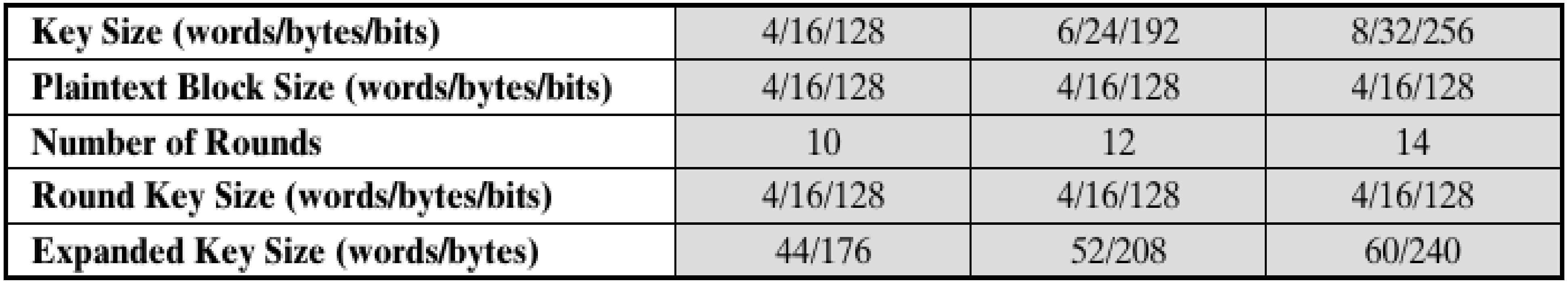


k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 • • • w_{42} w_{43}

(b) Key and expanded key

- The cipher consists of N rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key.
- The first N - 1 rounds consist of four distinct transformation functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey.
- The final round contains only three transformations, and there is a initial single transformation (AddRoundKey) before the first round, which can be considered Round 0.

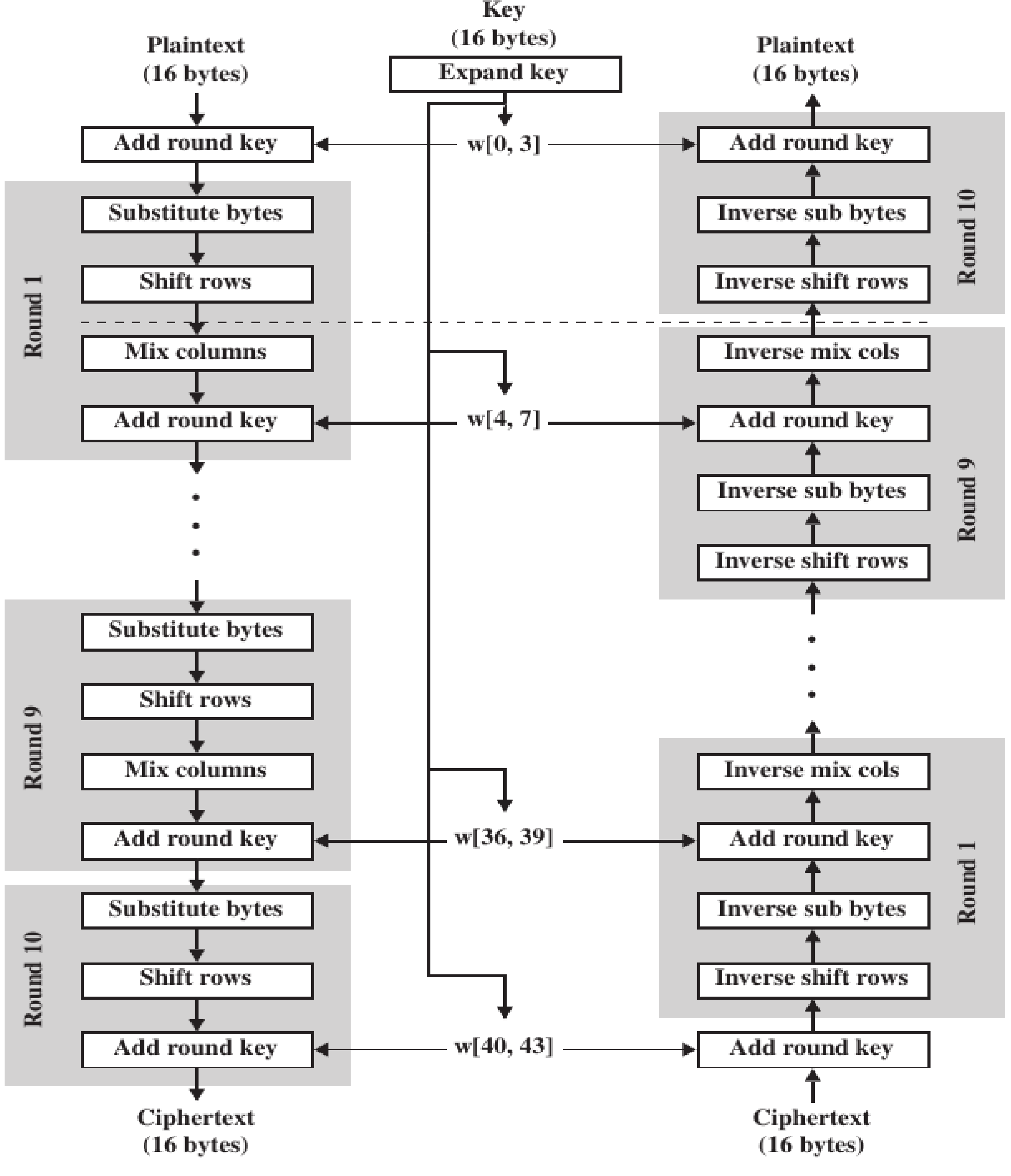
- Each transformation takes one or more 4 * 4 matrices as input and produces a 4 * 4 matrix as output.
- Also, the key expansion function generates N + 1 round keys, each of which is a distinct 4 * 4 matrix.



Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext Block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of Rounds	10	12	14
Round Key Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded Key Size (words/bytes)	44/176	52/208	60/240

Detailed Structure

1. AES processes the entire data block as a single matrix during each round using substitutions and permutation.
2. The key that is provided as input is expanded into an array of forty-four 32-bit words, w[i]. Four distinct words (128 bits) serve as a round key for each round (Figure).



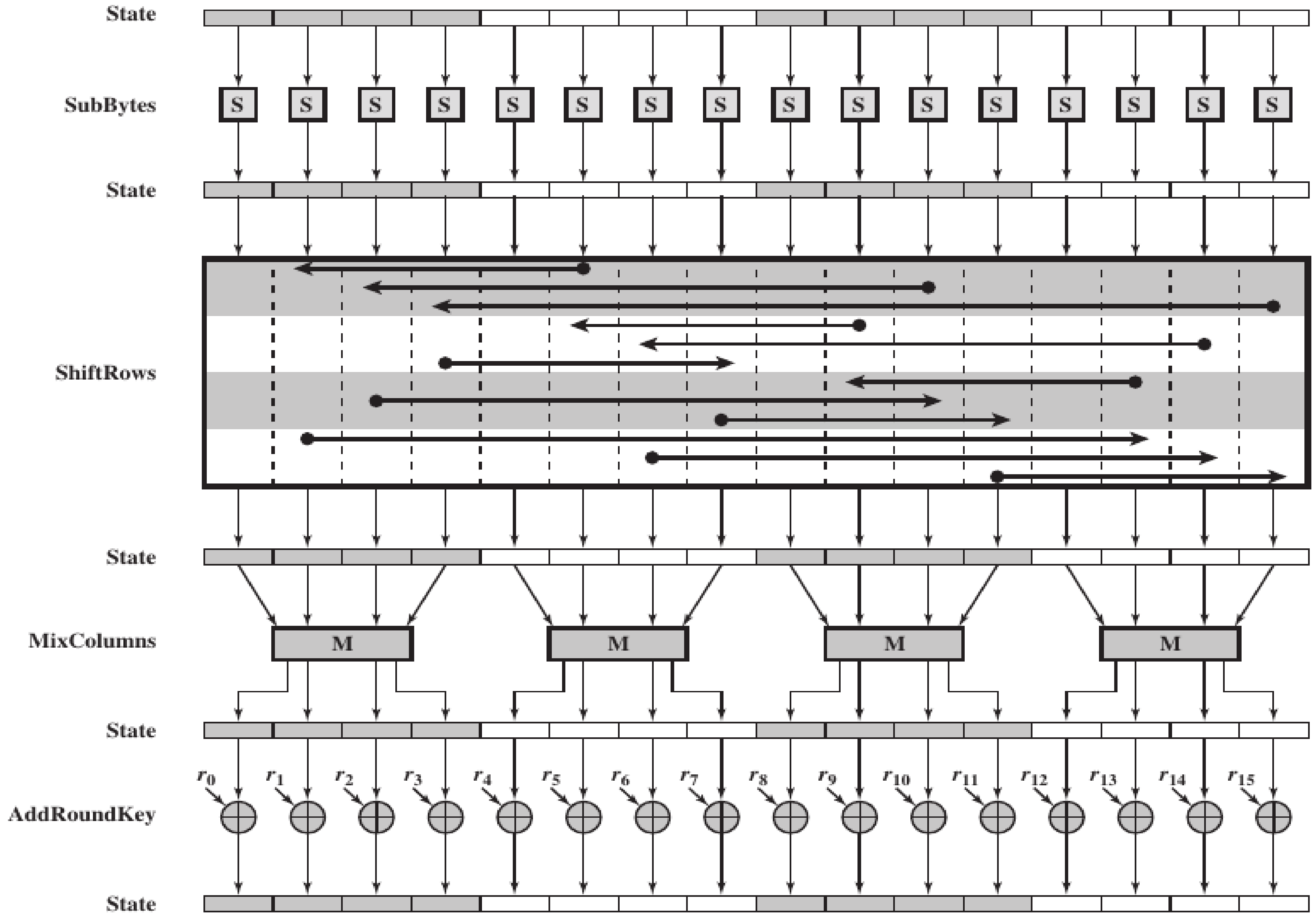
(a) Encryption (b) Decryption

3. Four different stages are used, one of permutation and three of substitution:

- Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block
- ShiftRows: A simple permutation
- MixColumns: A substitution that makes use of arithmetic over GF(2^8)
- AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key

4. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Figure 5.4 depicts the structure of a full encryption round.

5. Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.



State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

Figure 5.4 AES Encryption Round

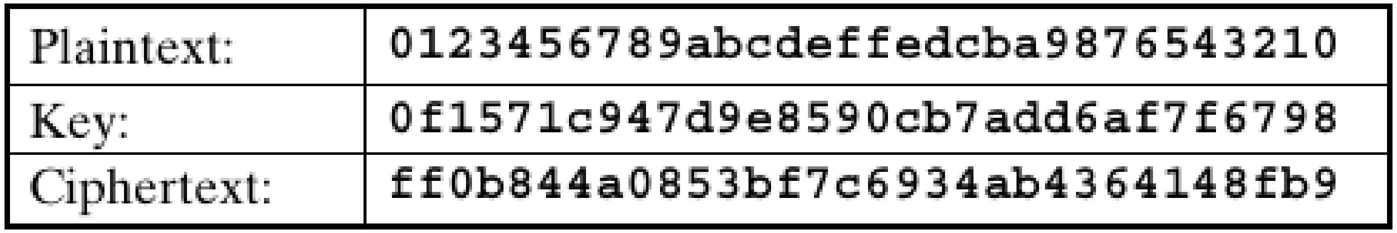
6. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.

7. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that $A \oplus B \oplus B = A$.

8. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm.

An AES Example

- The plaintext, key, and resulting ciphertext are



Plaintext:	0123456789abcdeffedcba9876543210
Key:	0f1571c947d9e8590cb7add6af7f6798
Ciphertext:	ff0b844a0853bf7c6934ab4364148fb9

- Next, Table 5.4 shows the progression of State through the AES encryption process.
- The first column shows the value of State at the start of a round. For the first row, State is just the matrix arrangement of the plaintext.
- The second, third, and fourth columns show the value of State for that round after the SubBytes, ShiftRows, and MixColumns transformations, respectively. The fifth column shows the round key.

Table 5.4 AES Example

Start of Round	After SubBytes	After ShiftRows	After MixColumns	Round Key
01 89 fe 76				0f 47 0c af
23 ab dc 54				15 d9 b7 7f
45 cd ba 32				71 e8 ad 67
67 ef 98 10				c9 59 d6 98
0e ce f2 d9	ab 8b 89 35	ab 8b 89 35	b9 94 57 75	dc 9b 97 38
36 72 6b 2b	05 40 7f f1	40 7f f1 05	e4 8e 16 51	90 49 fe 81
34 25 17 55	18 3f f0 fc	f0 fc 18 3f	47 20 9a 3f	37 df 72 15
ae b6 4e 88	e4 4e 2f c4	c4 e4 4e 2f	c5 d6 f5 3b	b0 e9 3f a7
65 0f c0 4d	4d 76 ba e3	4d 76 ba e3	8e 22 db 12	d2 49 de e6
74 c7 e8 d0	92 c6 9b 70	c6 9b 70 92	b2 f2 dc 92	c9 80 7e ff
70 ff e8 2a	51 16 9b e5	9b e5 51 16	df 80 f7 c1	6b b4 c6 d3
75 3f ca 9c	9d 75 74 de	de 9d 75 74	2d c5 1e 52	b7 5e 61 c6
5c 6b 05 f4	4a 7f 6b bf	4a 7f 6b bf	b1 c1 0b cc	c0 89 57 b1
7b 72 a2 6d	21 40 3a 3c	40 3a 3c 21	ba f3 8b 07	af 2f 51 ae
b4 34 31 12	8d 18 c7 c9	c7 c9 8d 18	f9 1f 6a c3	df 6b ad 7e
9a 9b 7f 94	b8 14 d2 22	22 b8 14 d2	1d 19 24 5c	39 67 06 c0
71 48 5c 7d	a3 52 4a ff	a3 52 4a ff	d4 11 fe 0f	2c a5 f2 43
15 dc da a9	59 86 57 d3	86 57 d3 59	3b 44 06 73	5c 73 22 8c
26 74 c7 bd	f7 92 c6 7a	c6 7a f7 92	cb ab 62 37	65 0e a3 dd
24 7e 22 9c	36 f3 93 de	de 36 f3 93	19 b7 07 ec	f1 96 90 50
f8 b4 0c 4c	41 8d fe 29	41 8d fe 29	2a 47 c4 48	58 fd 0f 4c
67 37 24 ff	85 9a 36 16	9a 36 16 85	83 e8 18 ba	9d ee cc 40
ae a5 c1 ea	e4 06 78 87	78 87 e4 06	84 18 27 23	36 38 9b 46
e8 21 97 bc	9b fd 88 65	65 9b fd 88	eb 10 0a f3	eb 7d ed bd
72 ba cb 04	40 f4 1f f2	40 f4 1f f2	7b 05 42 4a	71 8c 83 cf
1e 06 d4 fa	72 6f 48 2d	6f 48 2d 72	1e d0 20 40	c7 29 e5 a5
b2 20 bc 65	37 b7 65 4d	65 4d 37 b7	94 83 18 52	4c 74 ef a9
00 6d e7 4e	63 3c 94 2f	2f 63 3c 94	94 c4 43 fb	c2 bf 52 ef
0a 89 c1 85	67 a7 78 97	67 a7 78 97	ec 1a c0 80	37 bb 38 f7
d9 f9 c5 e5	35 99 a6 d9	99 a6 d9 35	0c 50 53 c7	14 3d d8 7d
d8 f7 f7 fb	61 68 68 0f	68 0f 61 68	3b d7 00 ef	93 e7 08 a1
56 7b 11 14	b1 21 82 fa	fa b1 21 82	b7 22 72 e0	48 f7 a5 4a
db a1 f8 77	b9 32 41 f5	b9 32 41 f5	b1 1a 44 17	48 f3 cb 3c
18 6d 8b ba	ad 3c 3d f4	3c 3d f4 ad	3d 2f ec b6	26 1b c3 be
a8 30 08 4e	c2 04 30 2f	30 2f c2 04	0a 6b 2f 42	45 a2 aa 0b
ff d5 d7 aa	16 03 0e ac	ac 16 03 0e	9f 68 f3 b1	20 d7 72 38
f9 e9 8f 2b	99 1e 73 f1	99 1e 73 f1	31 30 3a c2	fd 0e c5 f9
1b 34 2f 08	af 18 15 30	18 15 30 af	ac 71 8c c4	0d 16 d5 6b
4f c9 85 49	84 dd 97 3b	97 3b 84 dd	46 65 48 eb	42 e0 4a 41
bf bf 81 89	08 08 0c a7	a7 08 08 0c	6a 1c 31 62	cb 1c 6e 56
cc 3e ff 3b	4b b2 16 e2	4b b2 16 e2	4b 86 8a 36	b4 ba 7f 86
a1 67 59 af	32 85 cb 79	85 cb 79 32	b1 cb 27 5a	8e 98 4d 26
04 85 02 aa	f2 97 77 ac	77 ac f2 97	fb f2 f2 af	f3 13 59 18
a1 00 5f 34	32 63 cf 18	18 32 63 cf	cc 5a 5b cf	52 4e 20 76
ff 08 69 64				
0b 53 34 14				
84 bf ab 8f				
4a 7c 43 b9				

Avalanche Effect

- Table 5.5 shows the result when the eighth bit of the plaintext is changed. The second column of the table shows the value of the State matrix at the end of each round for the two plaintexts.
- Note that after just one round, 20 bits of the State vector differ. After two rounds, close to half the bits differ.
- Table 5.6 shows the change in State matrix values when the same plaintext is used and the two keys differ in the eighth bit. One round produces a significant change.
- This avalanche effect is stronger than that for DES.

Table 5.5 Avalanche Effect in AES: Change in Plaintext

Round		Number of Bits that Differ
	0123456789abcdeffedcba9876543210 0023456789abcdeffedcba9876543210	1
0	0e3634aece7225b6f26b174ed92b5588 0f3634aece7225b6f26b174ed92b5588	1
1	657470750fc7ff3fc0e8e8ca4dd02a9c c4a9ad090fc7ff3fc0e8e8ca4dd02a9c	20
2	5c7bb49a6b72349b05a2317ff46d1294 fe2ae569f7ee8bb8c1f5a2bb37ef53d5	58
3	7115262448dc747e5cdac7227da9bd9c ec093dfb7c45343d689017507d485e62	59
4	f867aee8b437a5210c24c1974cffeabc 43efdb697244df808e8d9364ee0ae6f5	61
5	721eb200ba06206dcbd4bce704fa654e 7b28a5d5ed643287e006c099bb375302	68
6	0ad9d85689f9f77bc1c5f71185e5fb14 3bc2d8b6798d8ac4fe36a1d891ac181a	64
7	db18a8ffa16d30d5f88b08d777ba4eaa 9fb8b5452023c70280e5c4bb9e555a4b	67
8	f91b4fbfe934c9bf8f2f85812b084989 20264e1126b219aef7feb3f9b2d6de40	65
9	cca104a13e678500ff59025f3bafaa34 b56a0341b2290ba7dfdfbddcd8578205	61
10	ff0b844a0853bf7c6934ab4364148fb9 612b89398d0600cde116227ce72433f0	58

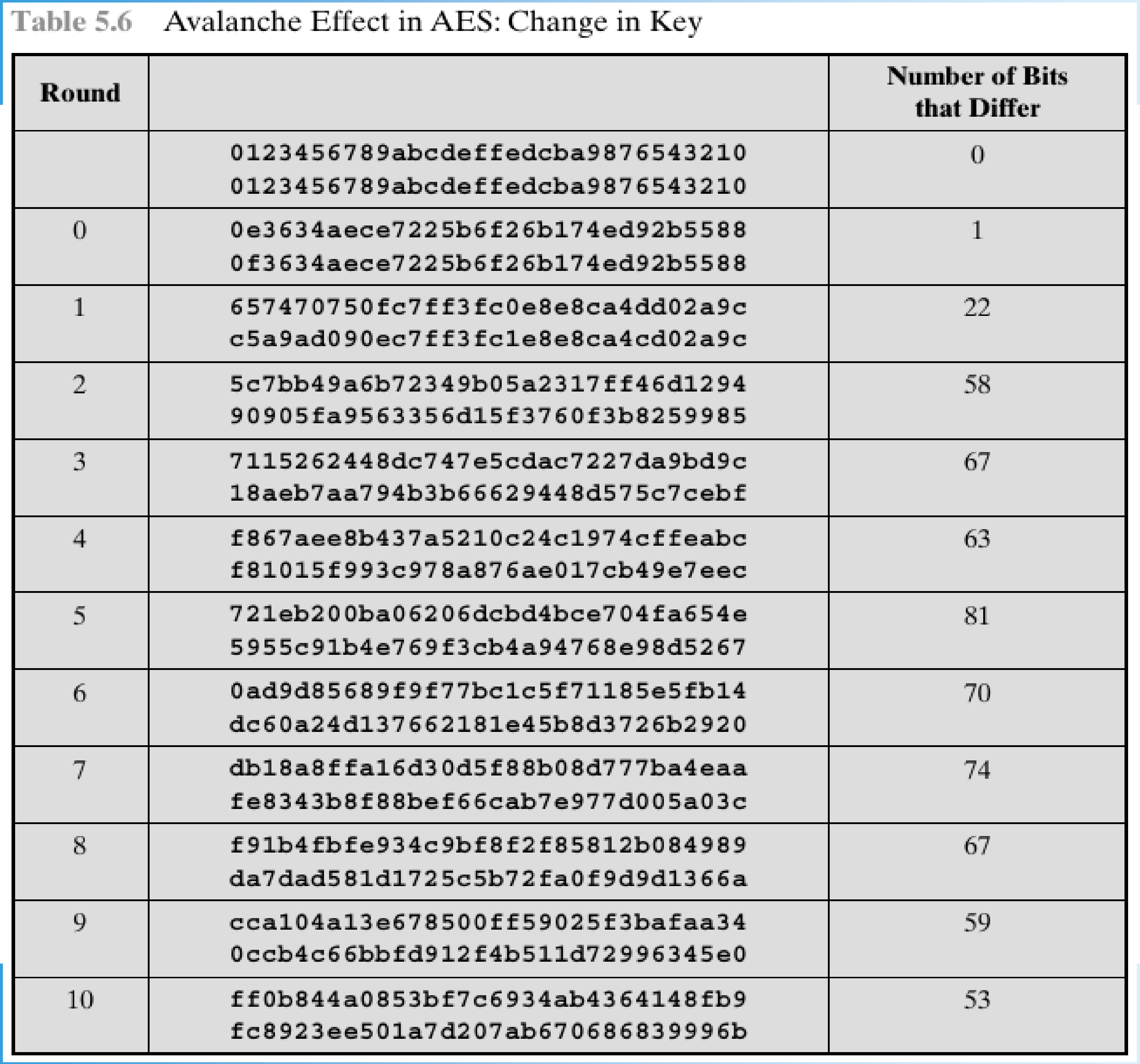


Table 5.6 Avalanche Effect in AES: Change in Key

Round		Number of Bits that Differ
	0123456789abcdeffedcba9876543210 0123456789abcdeffedcba9876543210	0
0	0e3634aece7225b6f26b174ed92b5588 0f3634aece7225b6f26b174ed92b5588	1
1	657470750fc7ff3fc0e8e8ca4dd02a9c c5a9ad090ec7ff3fc1e8e8ca4cd02a9c	22
2	5c7bb49a6b72349b05a2317ff46d1294 90905fa9563356d15f3760f3b8259985	58
3	7115262448dc747e5cdac7227da9bd9c 18aeb7aa794b3b66629448d575c7cebf	67
4	f867aee8b437a5210c24c1974cffeabc f81015f993c978a876ae017cb49e7eec	63
5	721eb200ba06206dcbd4bce704fa654e 5955c91b4e769f3cb4a94768e98d5267	81
6	0ad9d85689f9f77bc1c5f71185e5fb14 dc60a24d137662181e45b8d3726b2920	70
7	db18a8ffa16d30d5f88b08d777ba4eaa fe8343b8f88bef66cab7e977d005a03c	74
8	f91b4fbfe934c9bf8f2f85812b084989 da7dad581d1725c5b72fa0f9d9d1366a	67
9	cca104a13e678500ff59025f3bafaa34 0ccb4c66bbfd912f4b511d72996345e0	59
10	ff0b844a0853bf7c6934ab4364148fb9 fc8923ee501a7d207ab670686839996b	53

Multiple Encryption and Triple DES

- Given the potential vulnerability of DES to a brute-force attack, there has been considerable interest in finding an alternative.
- An alternative is to use multiple encryption with DES and multiple keys.

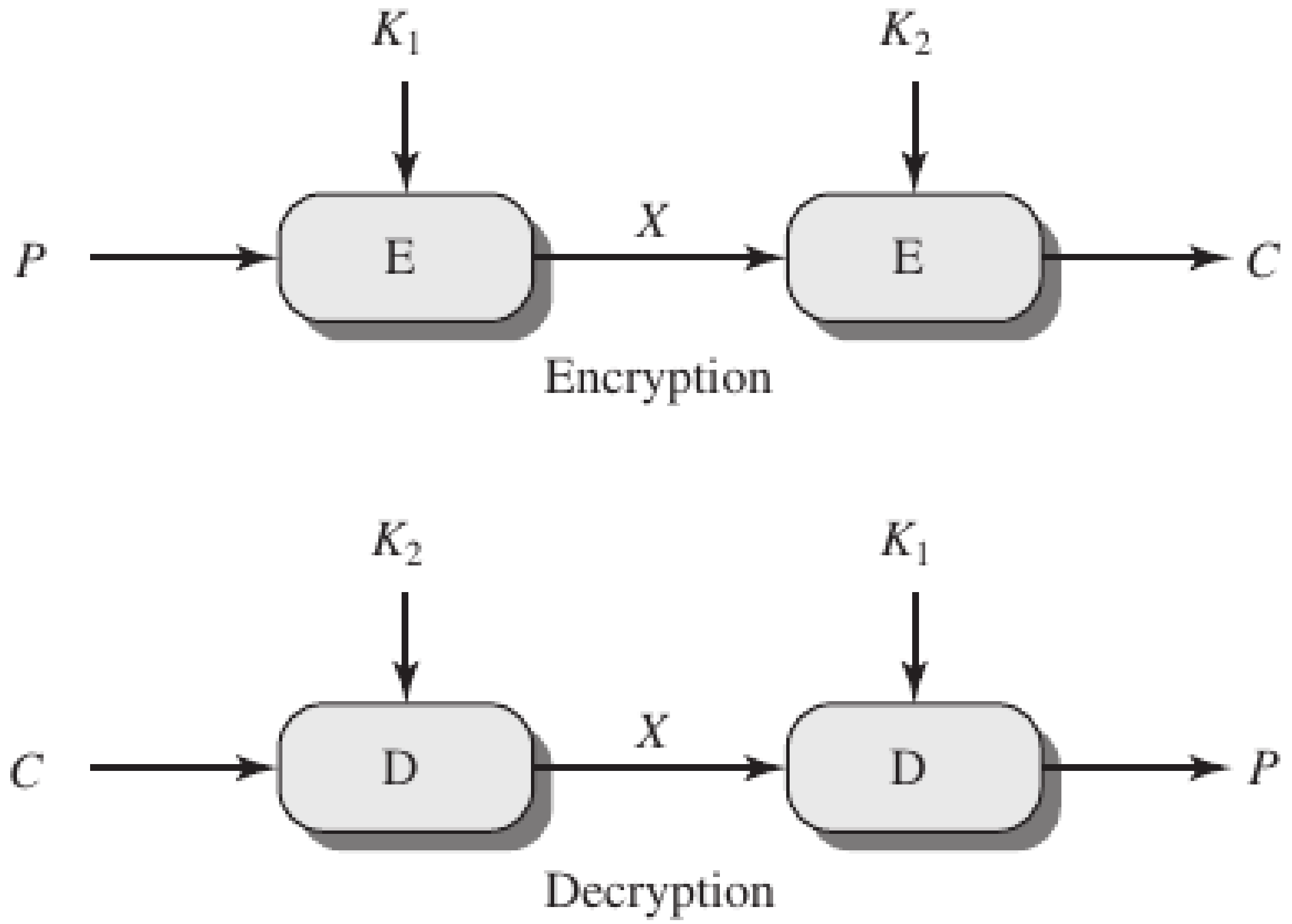
Double DES

- The simplest form of multiple encryption has two encryption stages and two keys (Figure 6.1a).
- Given a plaintext P and two encryption keys K_1 and K_2, ciphertext C is generated as

$$C = E(K_2, E(K_1, P))$$

- Decryption requires that the keys be applied in reverse order:

$$P = D(K_1, D(K_2, C))$$



(a) Double encryption

- There is a way to attack this scheme, known as a meet-in-the-middle attack. It is based on the observation that, if we have

$$C = E(K_2, E(K_1, P))$$

then (see Figure 6.1a)

$$X = E(K_1, P) = D(K_2, C)$$

- Given a known pair (P, C), the attack works.

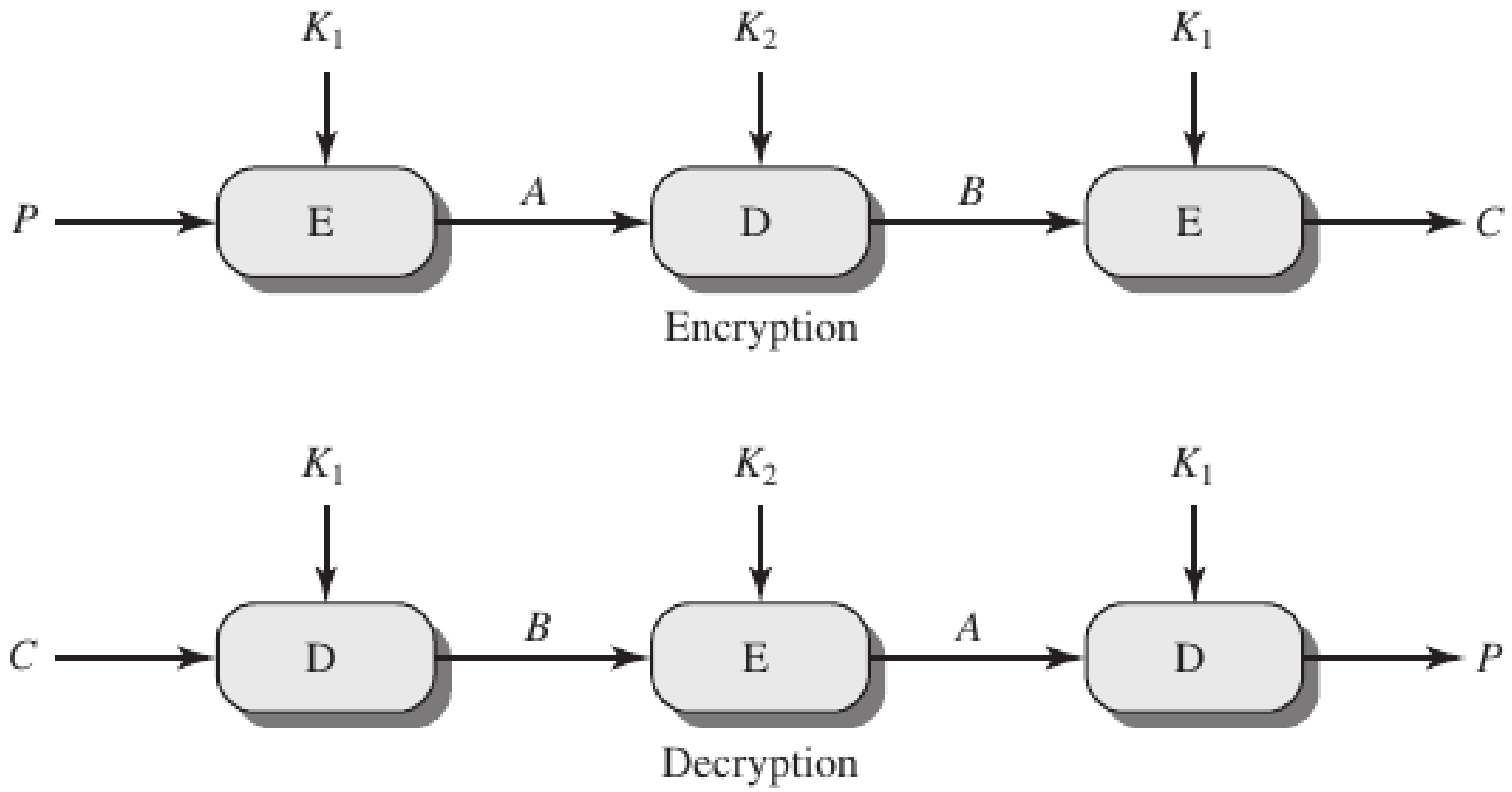
Triple DES with Two Keys

- Tuchman proposed a triple encryption method that uses only two keys. The function follows an encrypt-decrypt-encrypt (EDE) sequence:

$$C = E(K_1, D(K_2, E(K_1, P)))$$

$$P = D(K_1, E(K_2, D(K_1, C)))$$

- Currently, there are no practical cryptanalytic attacks on 3DES.



(b) Triple encryption

Triple DES with Three Keys

- Three-key 3DES has an effective key length of 168 bits and is defined as

$$C = E(K_3, D(K_2, E(K_1, P)))$$

- Backward compatibility with DES is provided by putting $K_3 = K_2$ or $K_1 = K_2$.

Public-Key Cryptography

Principles of public key cryptosystems

Public-Key Cryptosystems have the following important characteristic.

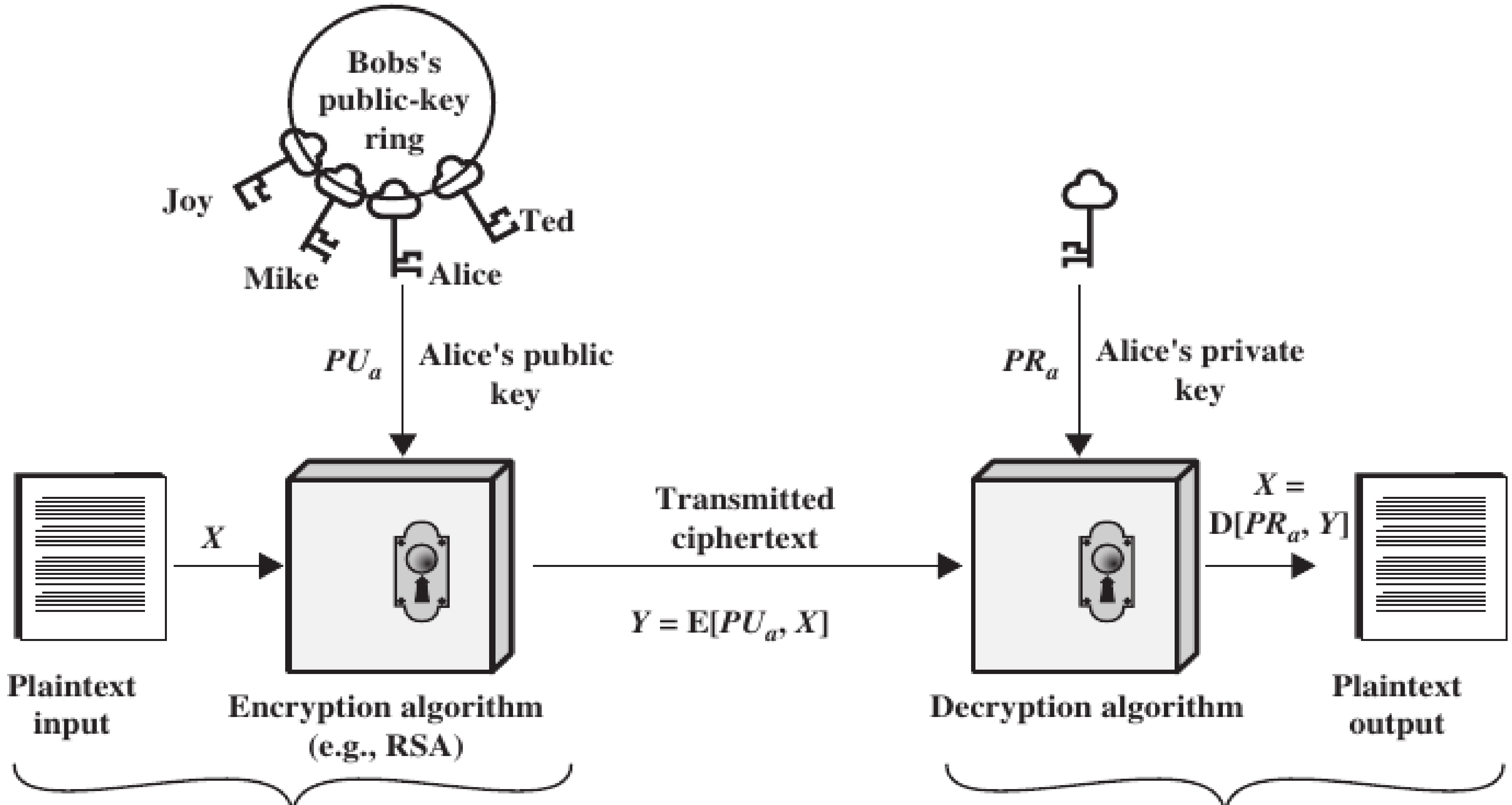
- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, exhibit the following characteristic.

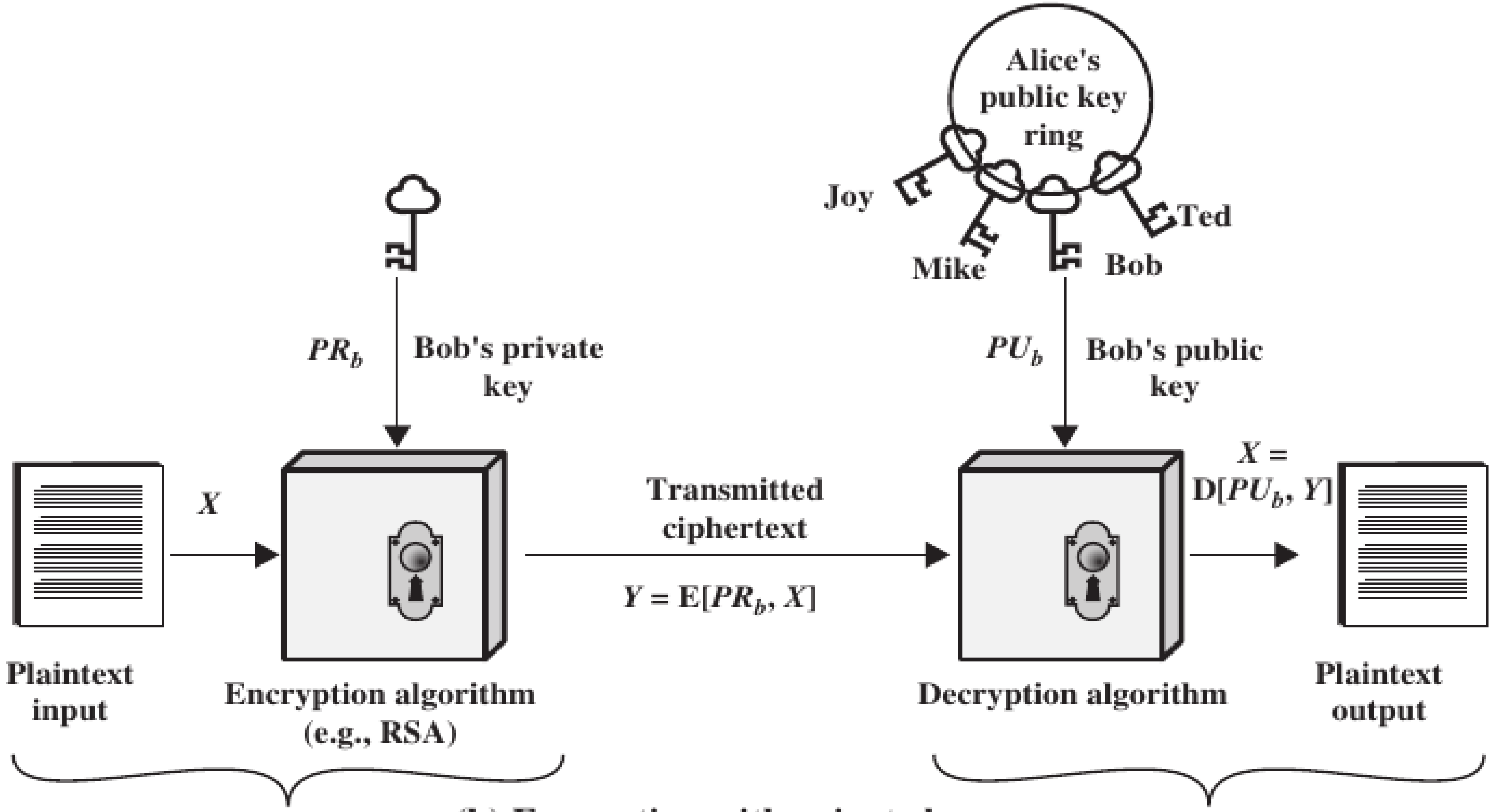
- Either of the two related keys can be used for encryption, with the other used for decryption.

A public-key encryption scheme has six ingredients (Figure):

- 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.



(a) Encryption with public key



(b) Encryption with private key

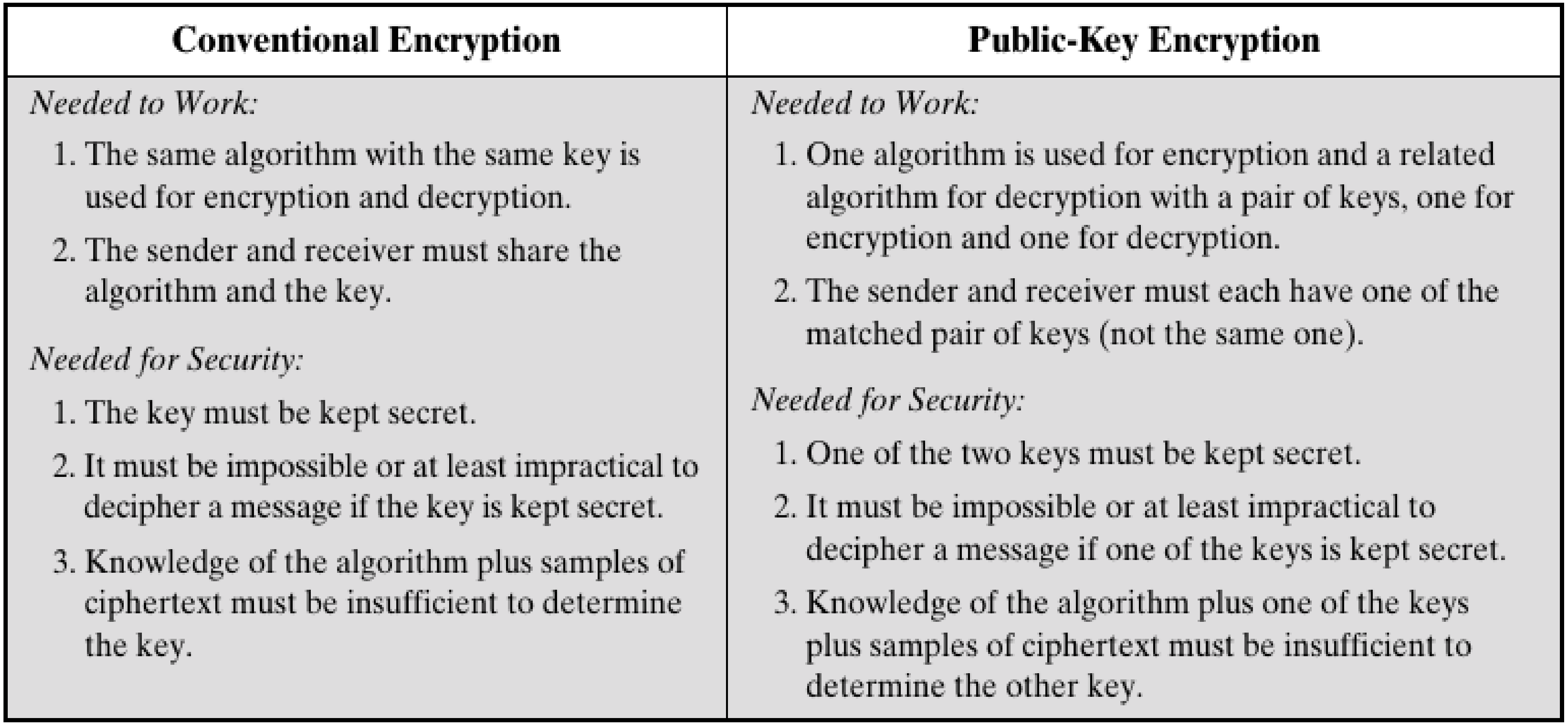
- Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the key.
- Decryption algorithm: This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. 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 (key ring).

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. No other recipient can decrypt the message because only Alice knows Alice’s private key.

Table 9.2 Conventional and Public-Key Encryption



Conventional Encryption	**Public-Key Encryption**
Needed to Work:	*Needed to Work:*
1. The same algorithm with the same key is used for encryption and decryption.	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 share the algorithm and the key.	2. The sender and receiver must each have one of the matched pair of keys (not the same one).
Needed for Security:	*Needed for Security:*
1. The key must be kept secret.	1. One of the two keys must be kept secret.
2. It must be impossible or at least impractical to decipher a message if the key is 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 samples of ciphertext must be insufficient to determine the key.	3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

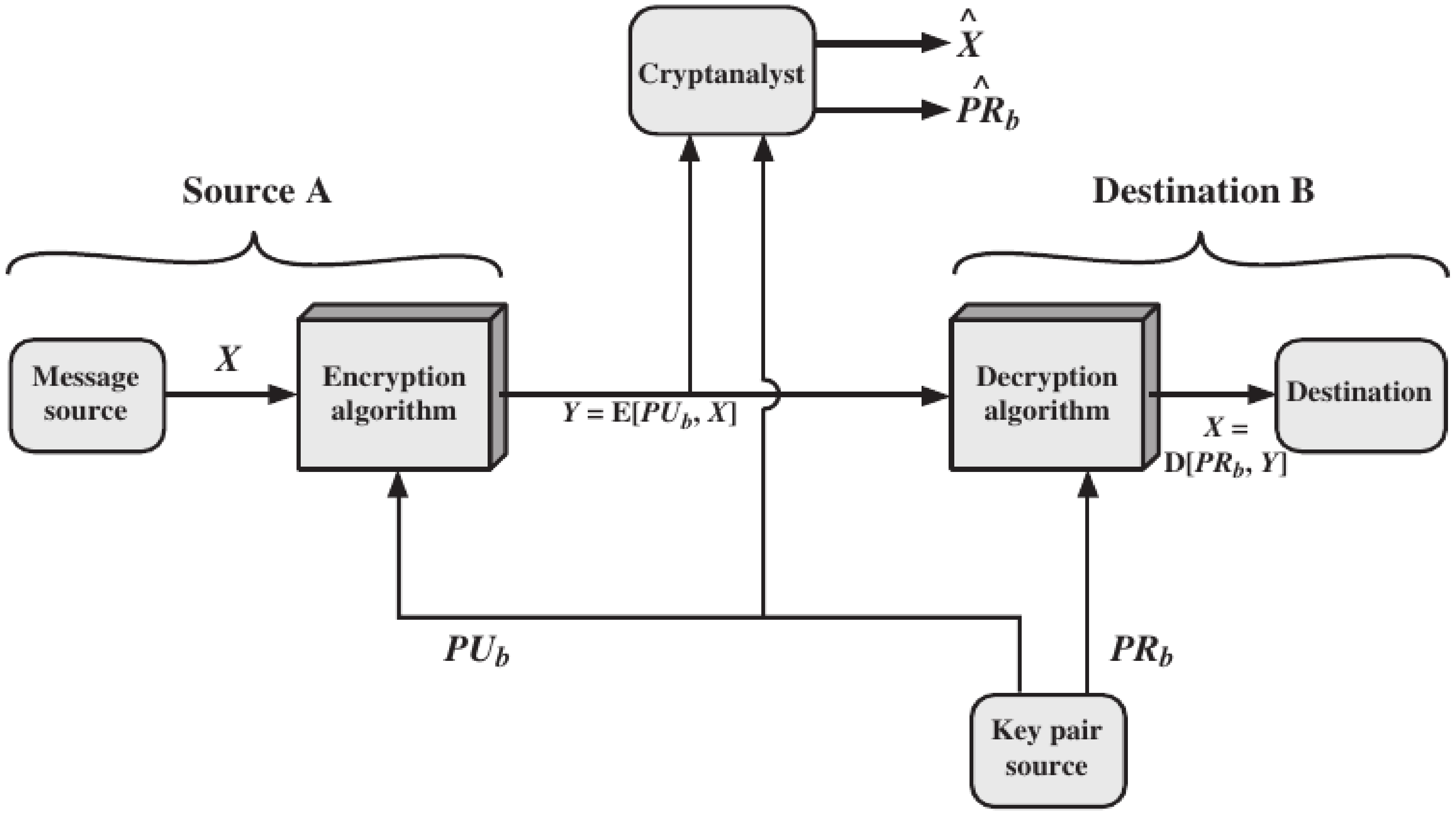


Figure 9.2 Public-Key Cryptosystem: Secrecy

- Whereas the scheme illustrated (encryption with public key) provides *confidentiality*, the next scheme (Figure 9.3) (encryption with private key) show the use of public-key encryption to provide *authentication.*
- Because the message was encrypted using A’s private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a digital signature.
- In addition, it is impossible to alter the message without access to A’s private key, so the message is authenticated both in terms of source and in terms of data integrity.

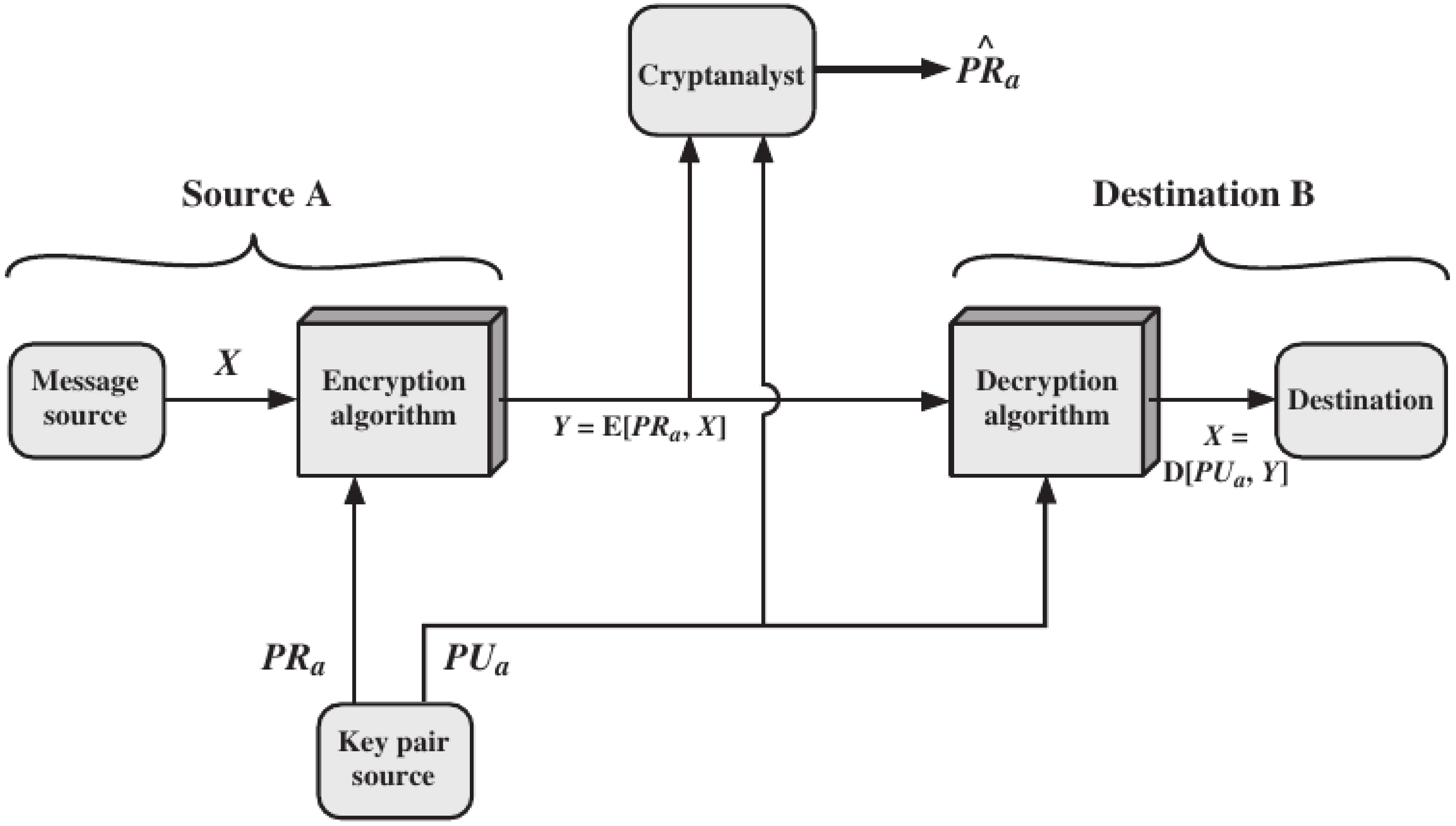


Figure 9.3 Public-Key Cryptosystem: Authentication

- It is important to emphasize that the encryption process (Figure 9.3) does not provide confidentiality. That is, the message being sent is safe from alteration but not from eavesdropping.
- It is possible to provide both the authentication function and confidentiality by a double use of the public-key scheme (Figure 9.4):

$$Z = E(PU_b, E(PR_a, X))$$

$$X = D(PU_a, D(PR_b, Z))$$

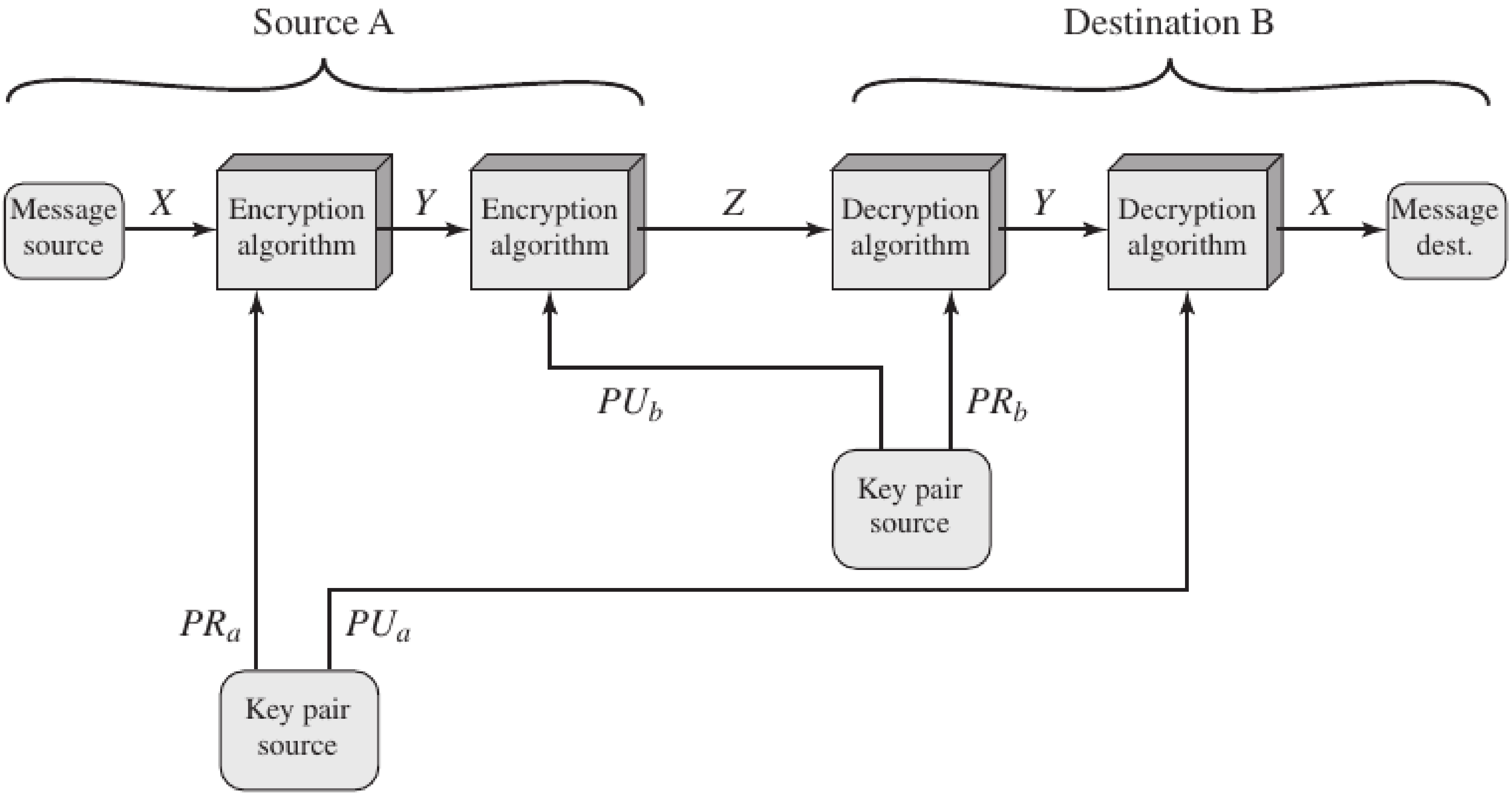


Figure 9.4 Public-Key Cryptosystem: Authentication and Secrecy

Applications for Public-Key Cryptosystems

- 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. Several different approaches are possible, involving the private key(s) of one or both parties.

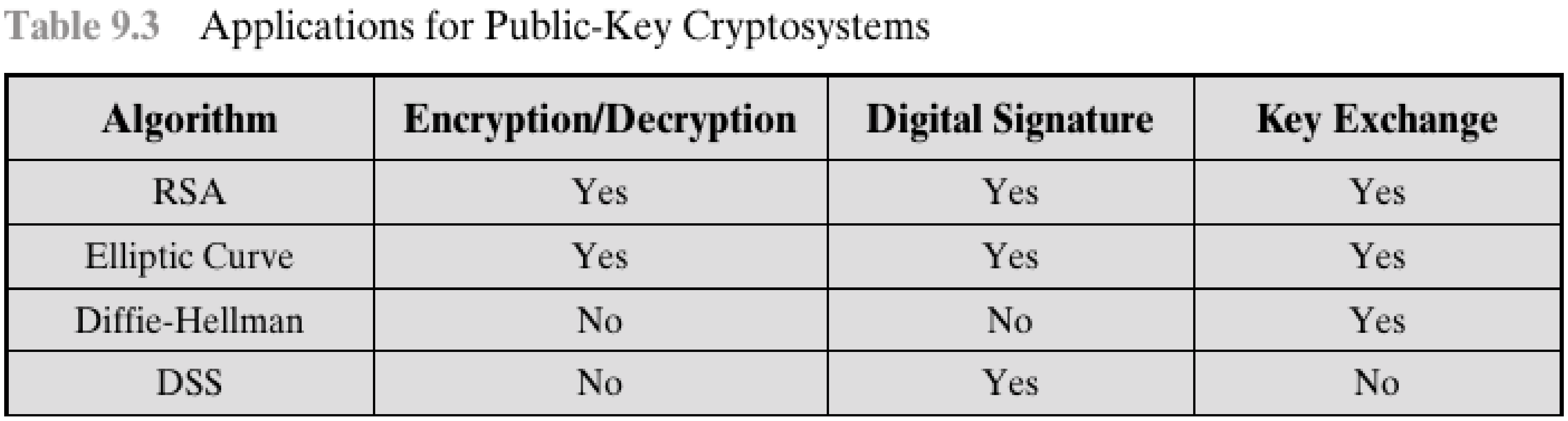


Table 9.3 Applications for Public-Key Cryptosystems

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

- A *one-way function* 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 (solved in polynomial time), whereas the calculation of the inverse is infeasible (fuzzier concept):

$$Y = f(X) \quad \text{easy}$$

$$X = f^{-1}(Y) \quad \text{infeasible}$$

- 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 trap-door one-way function is a family of invertible functions f_k, such that

$Y = \mathrm{f}_k(X)$ easy, if k and X are known

$X = \mathrm{f}_k^{-1}(Y)$ easy, if k and Y are known

$X = \mathrm{f}_k^{-1}(Y)$ infeasible, if Y is known but k is not known

- Thus, the development of a practical public-key scheme depends on discovery of a suitable trap-door one-way function.

The RSA Algorithm

- The first successful public-key system was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT.
- The RSA scheme 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}.

Description of the Algorithm

- Plaintext is encrypted in blocks, with each block having a binary value less than some number n.
- That is, the block size must be less than or equal to $\log_2(n) + 1$ bits.

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

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

- Both sender and receiver must know the value of n. The sender knows the value of e, and only the receiver knows the value of d.
- This is a public-key encryption algorithm with a public key of PU = {e, n} and a private key of PR = {d, n}.

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, and 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.

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

- The preceding relationship holds if e and d are multiplicative inverses modulo $\phi(n)$, where $\phi(n)$ is the Euler totient function.
- For p, q prime, $\phi(pq) = (p - 1)(q - 1)$. The relationship between e and d can be expressed as

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

- This is equivalent to saying

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

- That is, e and d are multiplicative inverses mod ϕ(n).
- Note that, according to the rules of modular arithmetic, this is true only if d (and therefore e) is relatively prime to ϕ(n).
- Equivalently, gcd(ϕ(n), d) = 1.

The ingredients of RSA scheme are the following:

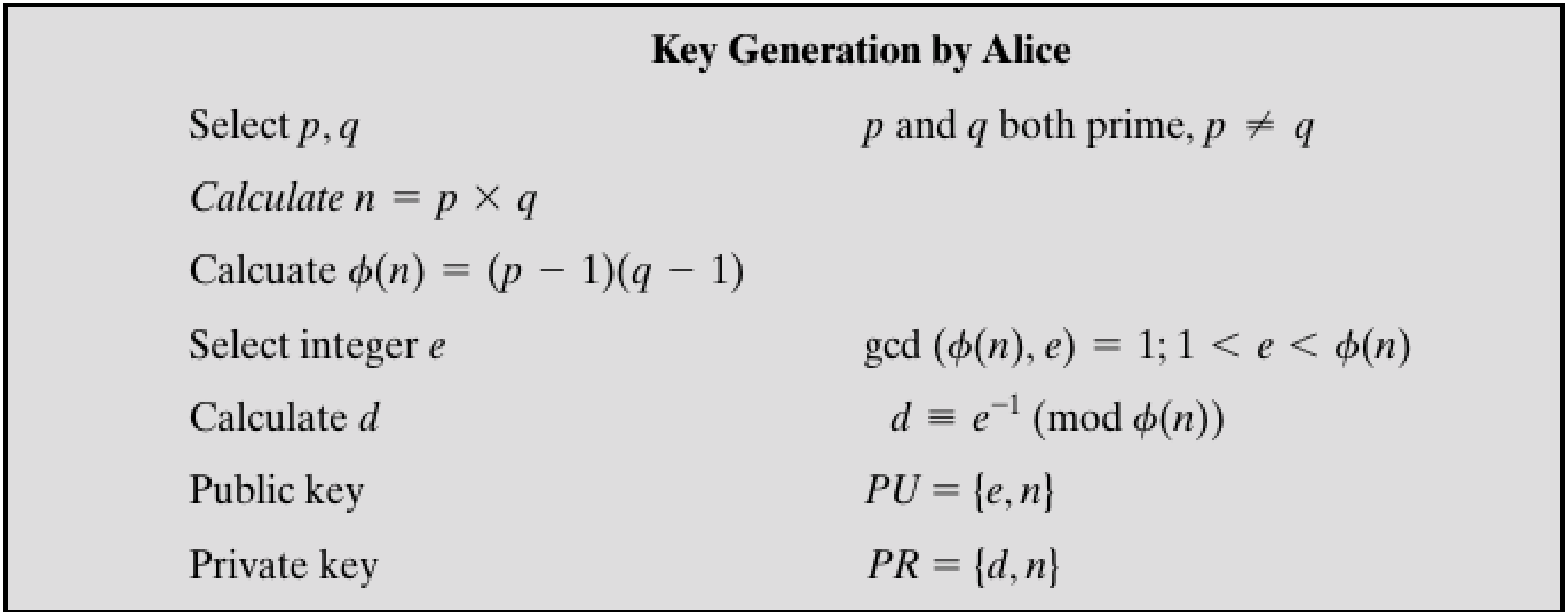
p, q, two prime numbers

$n = pq$

e, with $\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$

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

Figure summarizes the RSA algorithm.



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:	$M = C^d \bmod n$

An example is shown in Figure 9.6. The keys were generated as follows.

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 * 11 = 187$.
3. Calculate $\phi(n) = (p - 1)(q - 1) = 16 * 10 = 160$.
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 $23 * 7 = 161 = (1 * 160) + 1$; d can be calculated using the extended Euclid’s algorithm.

- The resulting keys are public key PU = {7, 187} and private key PR = {23, 187}.
- The example shows the use of these keys for a plaintext input of M = 88.
- For encryption, we need to calculate $C = 88^7$ mod 187. Exploiting the properties of modular arithmetic, we can do this as follows.

$$
\begin{aligned}
88^7 \bmod 187 &= [(88^4 \bmod 187) \times (88^2 \bmod 187) \\
&\quad \times (88^1 \bmod 187)] \bmod 187 \\
88^1 \bmod 187 &= 88 \\
88^2 \bmod 187 &= 7744 \bmod 187 = 77 \\
88^4 \bmod 187 &= 59{,}969{,}536 \bmod 187 = 132 \\
88^7 \bmod 187 &= (88 \times 77 \times 132) \bmod 187 = 894{,}432 \bmod 187 = 11
\end{aligned}
$$

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

$$\begin{aligned}
11^{23} \bmod 187 &= [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \\
&\quad \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187 \\
11^1 \bmod 187 &= 11 \\
11^2 \bmod 187 &= 121 \\
11^4 \bmod 187 &= 14{,}641 \bmod 187 = 55 \\
11^8 \bmod 187 &= 214{,}358{,}881 \bmod 187 = 33 \\
11^{23} \bmod 187 &= (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 \\
&= 79{,}720{,}245 \bmod 187 = 88
\end{aligned}$$

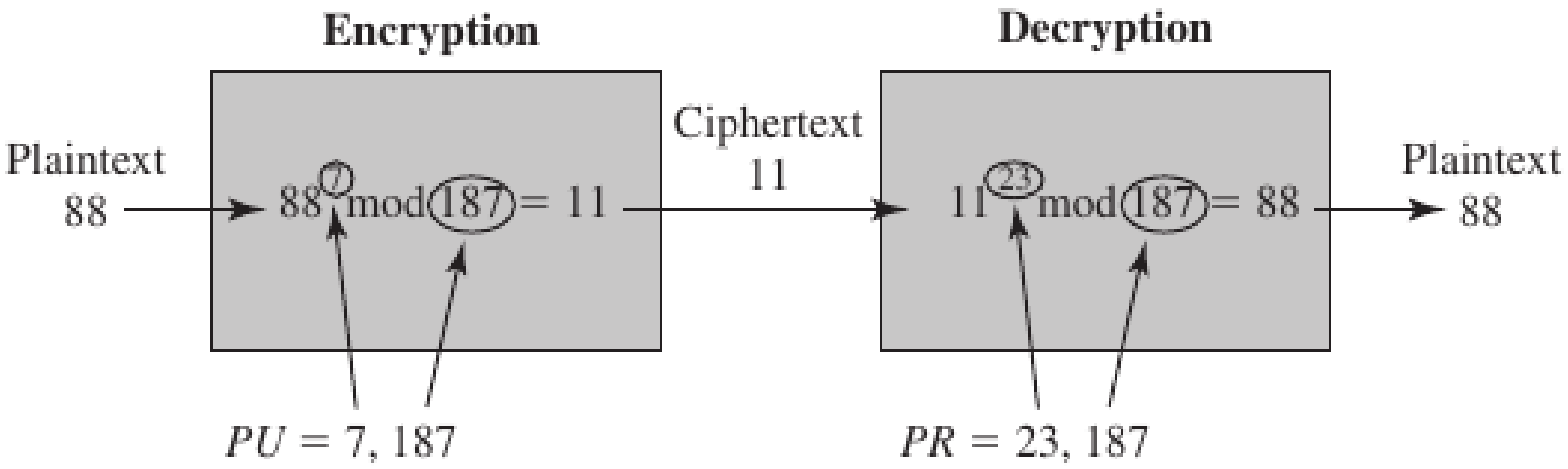
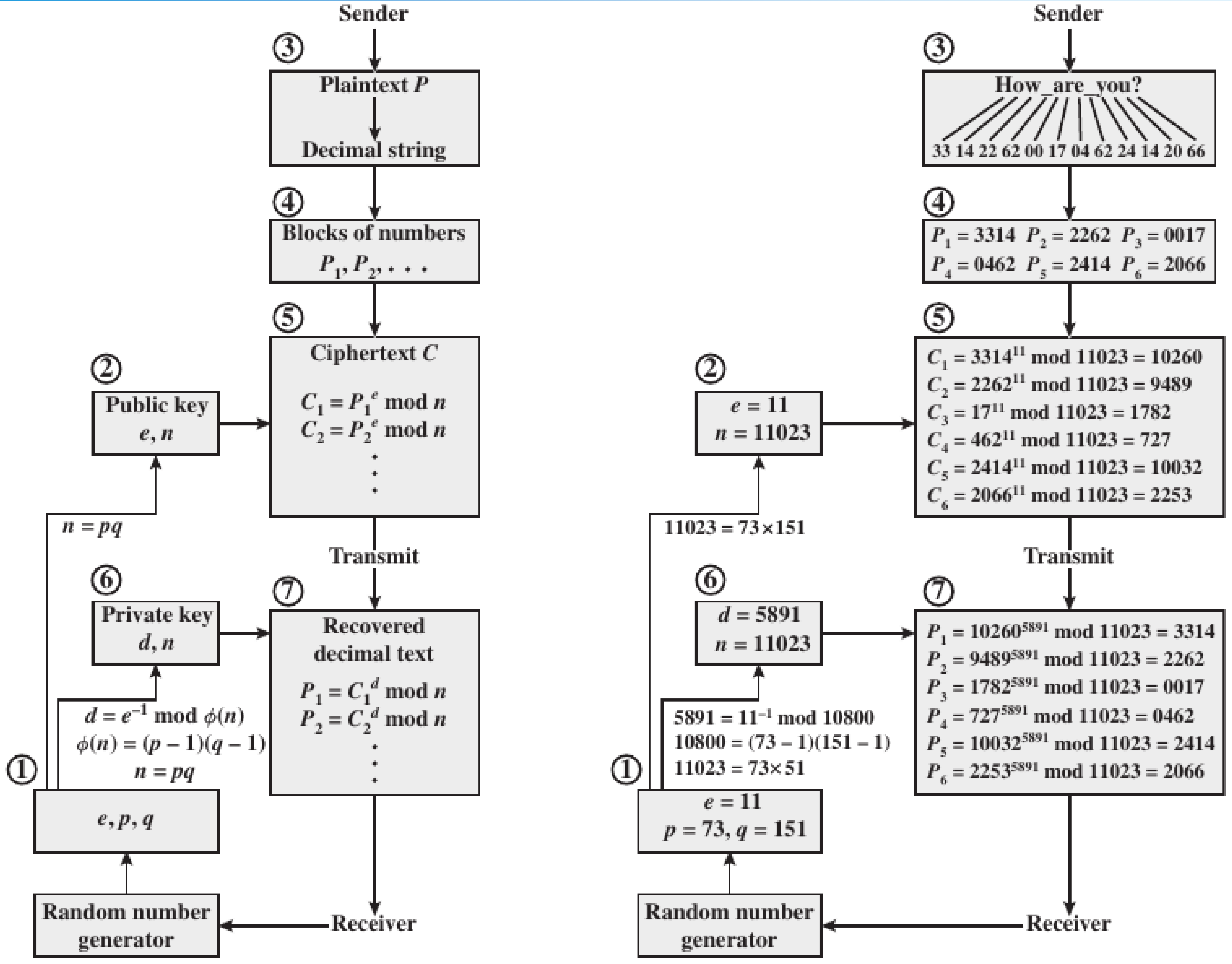


Figure 9.6 Example of RSA Algorithm

- Figure 9.7 illustrates the use of RSA to process multiple blocks of data.
- In this simple example, the plaintext is an 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.


Sender
③
Plaintext P
Decimal string
④
Blocks of numbers
$P_1, P_2, \ldots$
⑤
Ciphertext C
$C_1 = P_1^e \bmod n$
$C_2 = P_2^e \bmod n$
②
Public key
e, n
$n = pq$
Transmit
⑥
Private key
d, n
⑦
Recovered decimal text
$P_1 = C_1^d \bmod n$
$P_2 = C_2^d \bmod n$
$d = e^{-1} \bmod \phi(n)$
$\phi(n) = (p-1)(q-1)$
$n = pq$
①
e, p, q
Random number generator
Receiver
Sender
③
How_are_you?
33 14 22 62 00 17 04 62 24 14 20 66
④
$P_1 = 3314$ $P_2 = 2262$ $P_3 = 0017$
$P_4 = 0462$ $P_5 = 2414$ $P_6 = 2066$
⑤
$C_1 = 3314^{11} \bmod 11023 = 10260$
$C_2 = 2262^{11} \bmod 11023 = 9489$
$C_3 = 17^{11} \bmod 11023 = 1782$
$C_4 = 462^{11} \bmod 11023 = 727$
$C_5 = 2414^{11} \bmod 11023 = 10032$
$C_6 = 2066^{11} \bmod 11023 = 2253$
②
$e = 11$
$n = 11023$
$11023 = 73 \times 151$
Transmit
⑥
$d = 5891$
$n = 11023$
⑦
$P_1 = 10260^{5891} \bmod 11023 = 3314$
$P_2 = 9489^{5891} \bmod 11023 = 2262$
$P_3 = 1782^{5891} \bmod 11023 = 0017$
$P_4 = 727^{5891} \bmod 11023 = 0462$
$P_5 = 10032^{5891} \bmod 11023 = 2414$
$P_6 = 2253^{5891} \bmod 11023 = 2066$
$5891 = 11^{-1} \bmod 10800$
$10800 = (73-1)(151-1)$
$11023 = 73 \times 51$
①
$e = 11$
$p = 73, q = 151$
Random number generator
Receiver

Diffie-Hellman Key Exchange

- The first published public-key algorithm.
- 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 Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms.

- We can define the discrete logarithm in the following way: 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

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

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

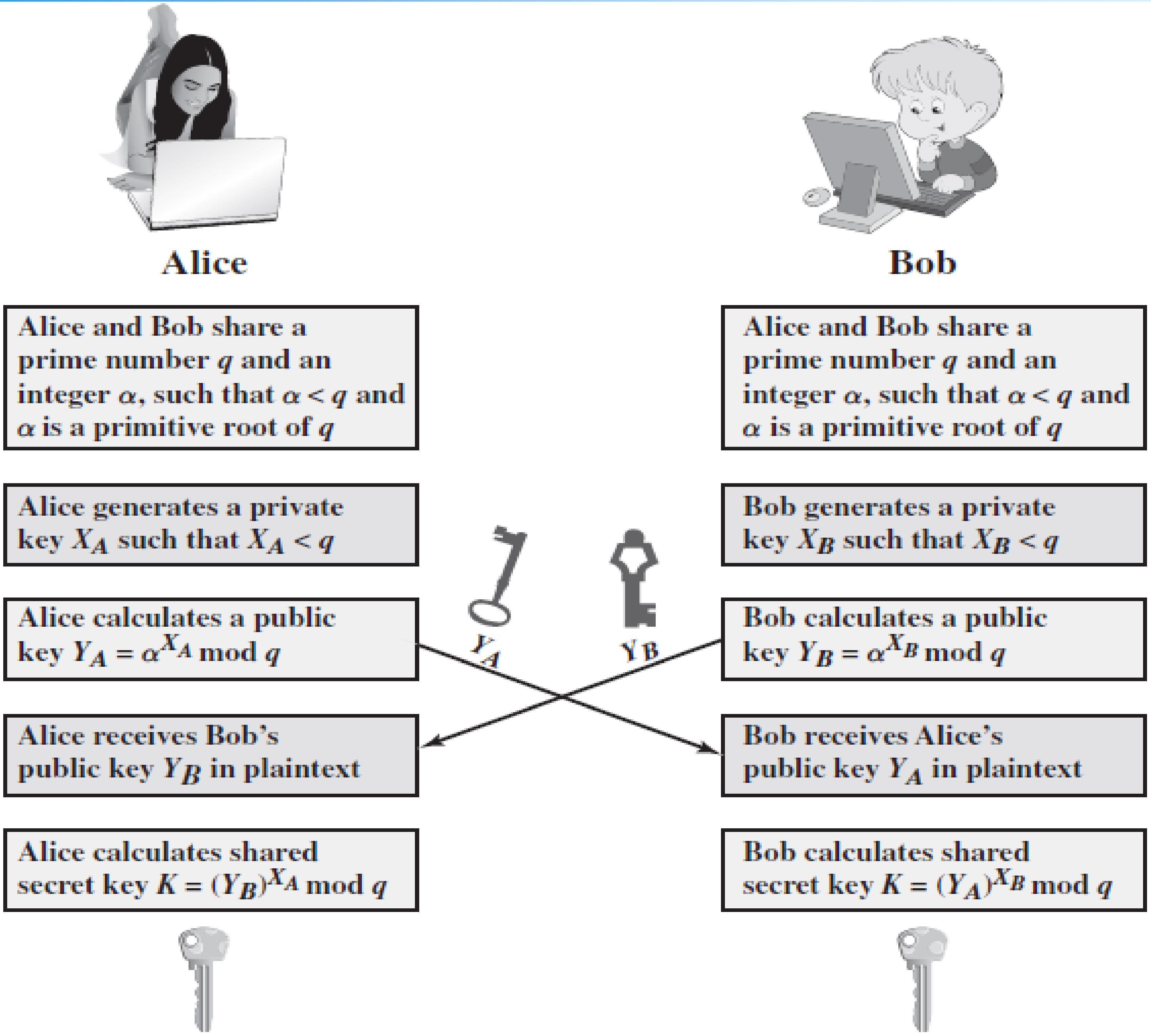
- For any integer *b* and a primitive root *a* of prime number *p*, we can find a unique exponent *i* such that

$$b \equiv a^i \pmod{p} \qquad \text{where } 0 \le i \le (p-1)$$

- The exponent *i* is referred to as the discrete logarithm of *b* for the base *a*, mod *p*.
- We express this value as $d\,log_{a,p}(b)$.

The Algorithm

- Figure 10.1 summarizes the Diffie-Hellman key exchange algorithm.
- For this scheme, there are two publicly known numbers: a prime number q and an integer α that is a primitive root of q.
- Suppose the users A and B wish to create a shared key.



Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q

Alice generates a private key X_A such that $X_A < q$

Alice calculates a public key $Y_A = \alpha^{X_A} \bmod q$

Alice receives Bob's public key Y_B in plaintext

Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$

Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q

Bob generates a private key X_B such that $X_B < q$

Bob calculates a public key $Y_B = \alpha^{X_B} \bmod q$

Bob receives Alice's public key Y_A in plaintext

Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$

Figure 10.1 The Diffie-Hellman Key Exchange

User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{X_A} \bmod q$. Similarly, user B independently selects a random integer $X_B < q$ and computes $Y_B = \alpha^{X_B} \bmod q$. 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 $K = (Y_B)^{X_A} \bmod q$ and user B computes the key as $K = (Y_A)^{X_B} \bmod q$. These two calculations produce identical results:

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

- Now 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, α, 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

$$X_B = \mathrm{d}\,\log_{\alpha,q}(Y_B)$$

- The adversary can calculate *K* as

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

- 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.
- For large primes, the latter task is considered infeasible.

An example

- Key exchange is based on the use of the prime number q = 353 and a primitive root of 353, in this case α = 3.
- A and B select private keys X_A = 97 and X_B = 233, respectively.
- Each computes its public key:

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 $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160.$

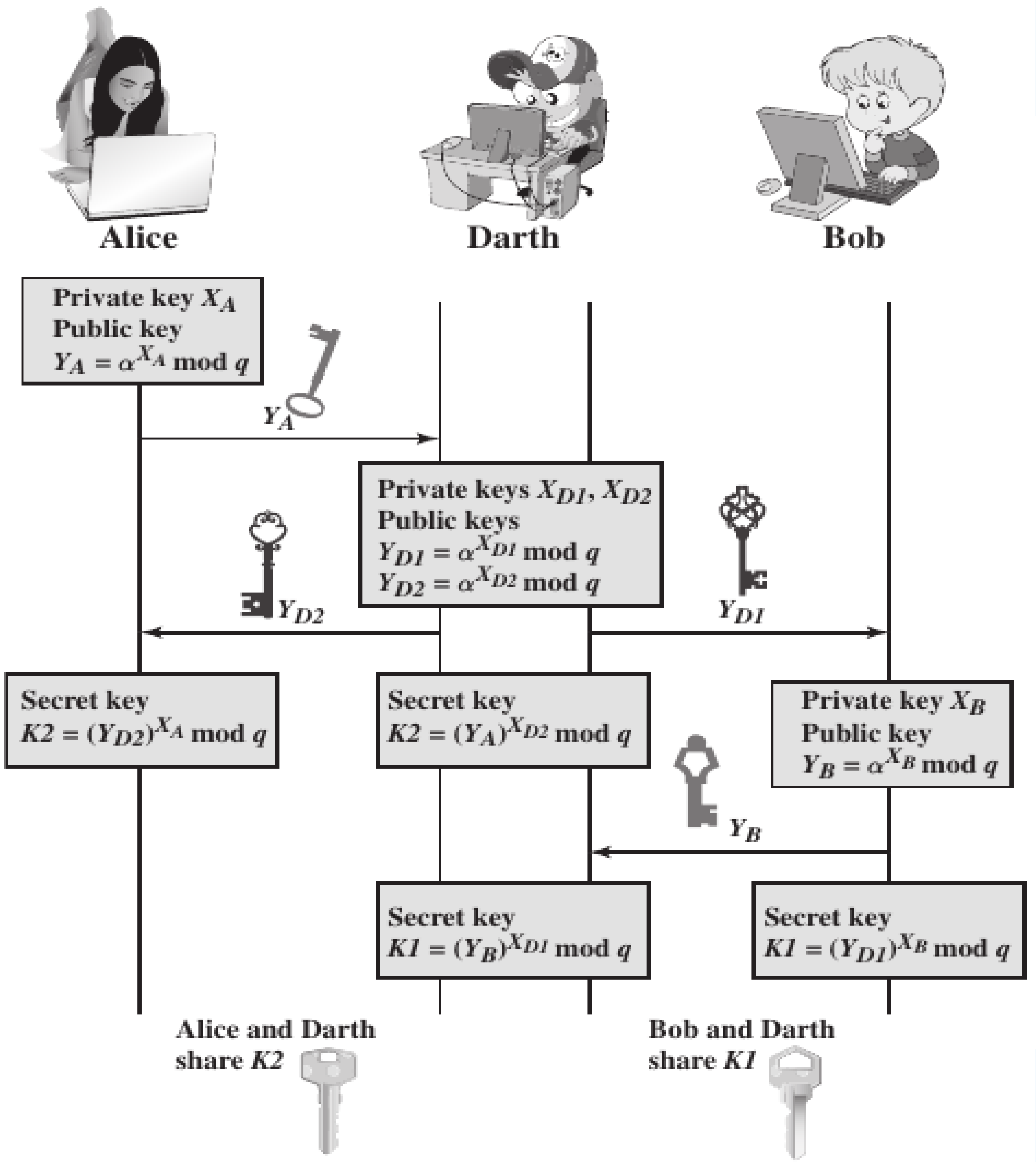
B computes $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160.$

Key Exchange Protocols

- Figure 10.1 shows a simple protocol that makes use of the Diffie-Hellman calculation.
- 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 it to user A.
- Both users can now calculate the key.

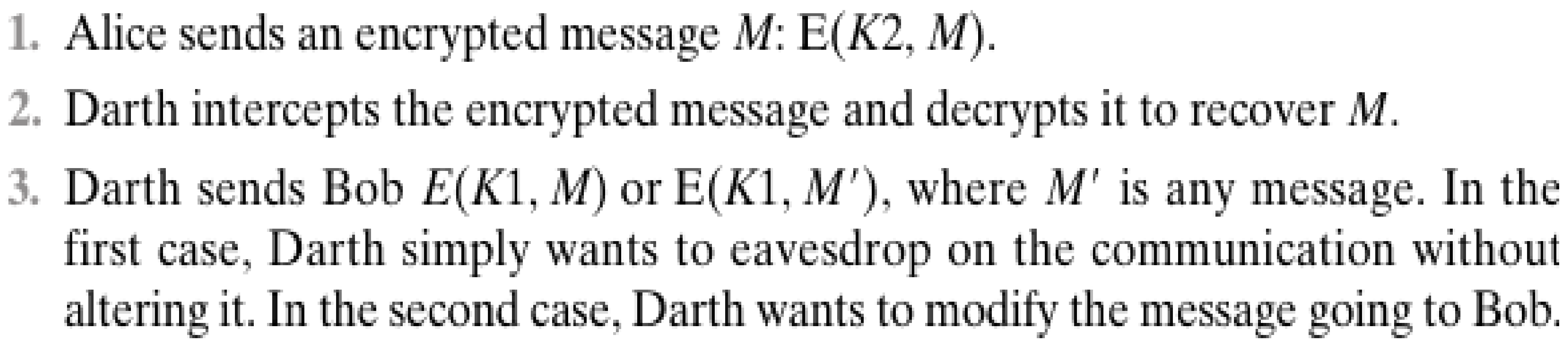
Man-in-the-Middle Attack

Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows (Figure 10.2).


Alice
Darth
Bob
Private key X_A
Public key
$Y_A = \alpha^{X_A} \bmod q$
Y_A
Private keys X_{D1}, X_{D2}
Public keys
$Y_{D1} = \alpha^{X_{D1}} \bmod q$
$Y_{D2} = \alpha^{X_{D2}} \bmod q$
Y_{D2}
Y_{D1}
Secret key
$K2 = (Y_{D2})^{X_A} \bmod q$
Secret key
$K2 = (Y_A)^{X_{D2}} \bmod q$
Private key X_B
Public key
$Y_B = \alpha^{X_B} \bmod q$
Y_B
Secret key
$K1 = (Y_B)^{X_{D1}} \bmod q$
Secret key
$K1 = (Y_{D1})^{X_B} \bmod q$
Alice and Darth share $K2$
Bob and Darth share $K1$

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 Y_A and transmits Y_{D1} to Bob. Darth also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_B} \bmod q$.
5. Bob transmits Y_B to Alice.
6. Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)^{X_{D1}} \bmod q$.
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$.

- 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: $\mathrm{E}(K2, M)$.
2. Darth intercepts the encrypted message and decrypts it to recover M.
3. Darth sends Bob $E(K1, M)$ or $\mathrm{E}(K1, M')$, 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

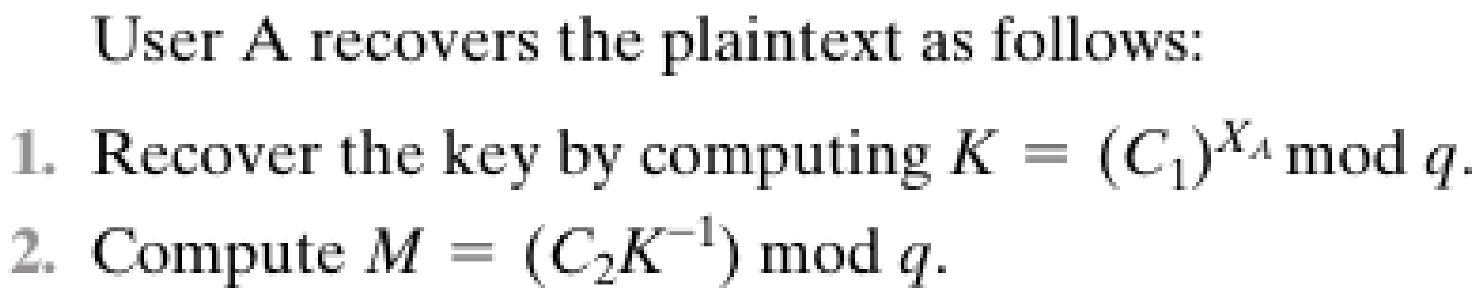
- In 1984, T. Elgamal announced a public-key scheme based on discrete logarithms, closely related to the Diffie-Hellman technique.
- The global elements of Elgamal are a prime number q and α, which is a primitive root of q. User A generates a private/public key pair as follows:

1. Generate a random integer X_A, such that $1 < X_A < q - 1$.
2. Compute $Y^A = \alpha^{X_A} \bmod q$.
3. A's private key is X_A and A's public key is $\{q, \alpha, Y_A\}$.

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 (C_1, C_2) where

$$C_1 = \alpha^k \bmod q; C_2 = KM \bmod q$$



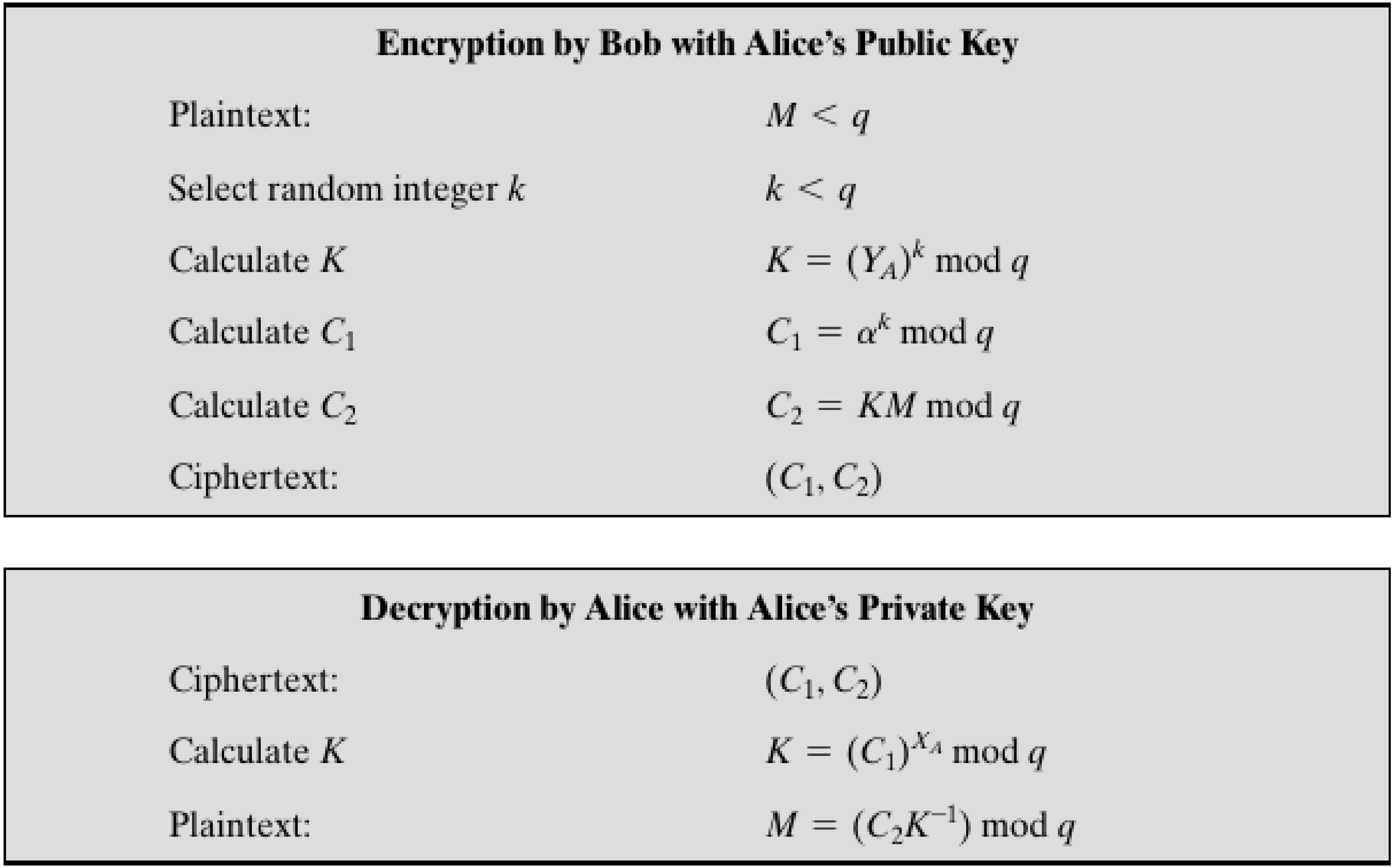
User A recovers the plaintext as follows:

1. Recover the key by computing $K = (C_1)^{X_A} \bmod q$.
2. Compute $M = (C_2K^{-1}) \bmod q$.

These steps are summarized in Figure 10.3.

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:	$M = (C_2K^{-1}) \bmod q$

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 C_1. C_1 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.

Elliptic Curve Arithmetic

- The principal attraction of ECC, compared to RSA, is that it appears to offer equal security for a far smaller key size, thereby reducing processing overhead.
- ECC is fundamentally more difficult to explain than either RSA or Diffie-Hellman.

- For Diffie-Hellman keys are generated by exponentiation (defined as repeated multiplication).

$$a^k \bmod q = \underbrace{(a \times a \times \ldots \times a)}_{k \text{ times}} \bmod q$$

- To attack Diffie-Hellman, the attacker must determine *k* given *a* and a^k; this is the discrete logarithm problem.

- For elliptic curve cryptography, an operation over elliptic curves, called addition, is used. Multiplication is defined by repeated addition.

$$a \times k = \underbrace{(a + a + \ldots + a)}_{k \text{ times}}$$

- Cryptanalysis involves determining *k* given *a* and (*a x k*).

- An elliptic curve is defined by an equation in two variables with coefficients. For cryptography, the variables and coefficients are restricted to elements in a finite field.
- Elliptic curves are not ellipses.
- They are so named because they are described by cubic equations, similar to those used for calculating the circumference of an ellipse.

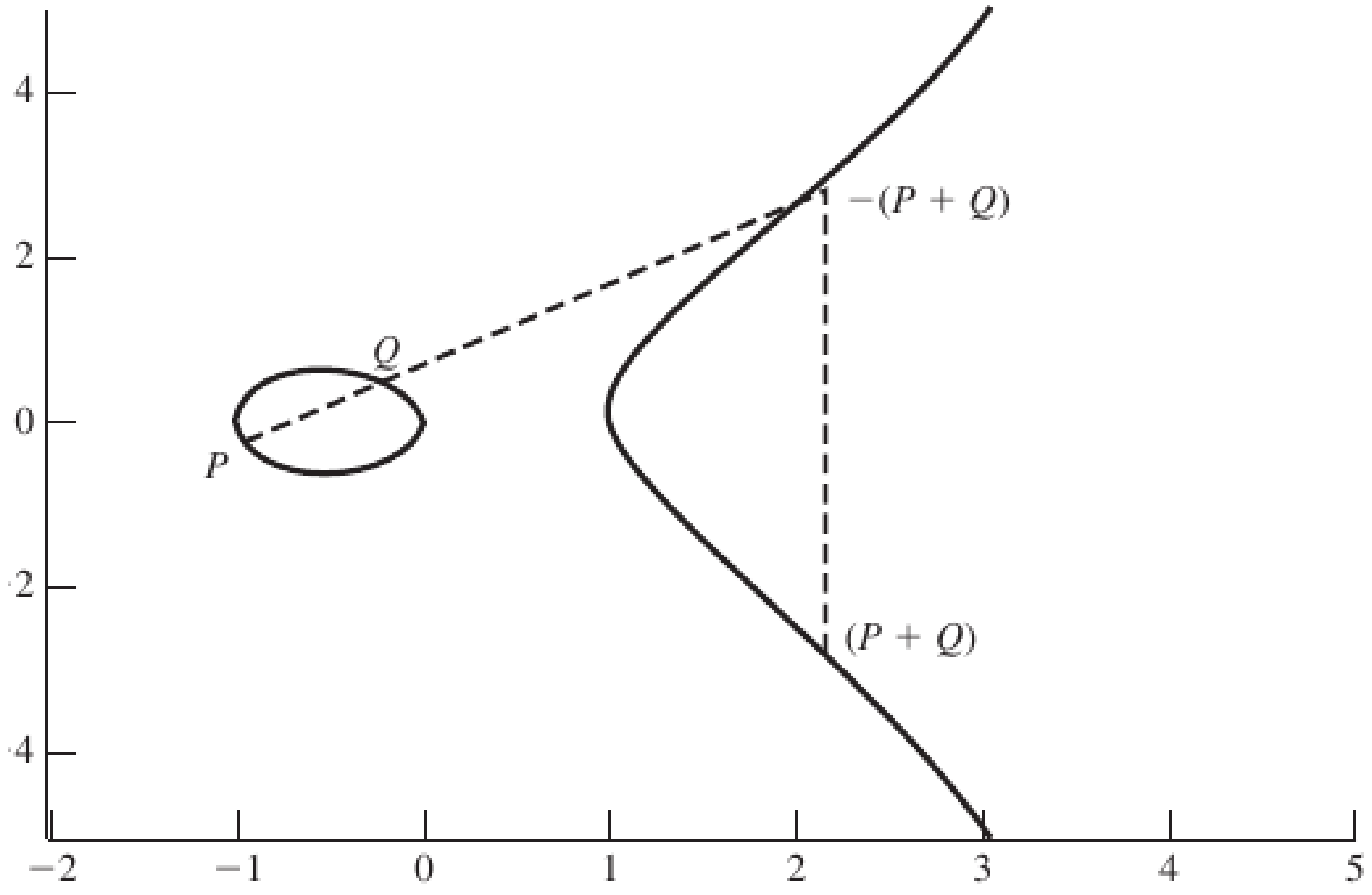
- In general, cubic equations for elliptic curves take the following form (Equation (10.1)

$$y^2 = x^3 + ax + b$$

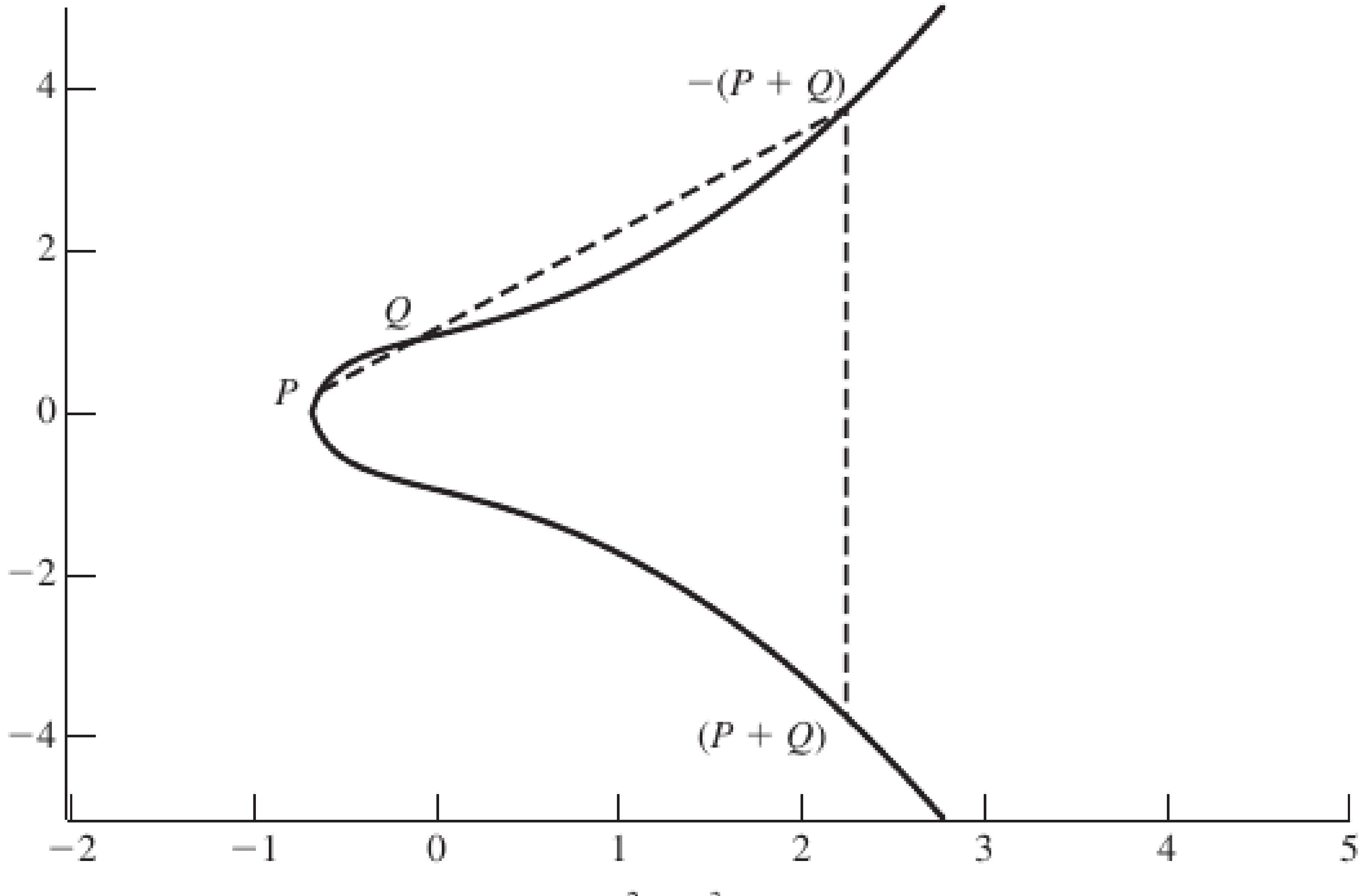
where a, b are real numbers and x and y take on values in the real numbers.

- Also included in the definition of an elliptic curve is a single element denoted O and called the *point at infinity* or the *zero point.*

- Consider the set of points *E(a, b)* consisting of all of the points *(x, y)* that satisfy Equation (10.1) together with the element O.
- Using a different value of the pair *(a, b)* results in a different set *E(a, b)*.
- Using this terminology, the two curves in Figure 10.4 depict the sets *E(-1, 0)* and *E(1, 1)*, respectively.



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



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

Figure 10.4 Example of Elliptic Curves

Elliptic Curves over Z_p

- Elliptic curve cryptography makes use of elliptic curves in which the variables and coefficients are all restricted to elements of a finite field Z_p.
- For elliptic curves over Z_p, we limit ourselves to equations of the form of Equation (10.1).

$$y^2 \bmod p = (x^3 + ax + b) \bmod p \qquad \textbf{(10.5)}$$

For example, Equation (10.5) is satisfied for $a = 1, b = 1, x = 9, y = 7, p = 23$:

$$7^2 \bmod 23 = (9^3 + 9 + 1) \bmod 23$$
$$49 \bmod 23 = 739 \bmod 23$$
$$3 = 3$$

- Now consider the set $E_p(a, b)$ consisting of all pairs of integers (x, y) that satisfy Equation (10.5), together with a point at infinity O. The coefficients a and b and the variables x and y are all elements of Z_p.

The rules for addition over $E_p(a, b)$, correspond to the algebraic technique described for elliptic curves defined over real numbers. For all points $P, Q \in E_p(a, b)$:

1. $P + O = P$.
2. If $P = (x_P, y_P)$, then $P + (x_P, -y_P) = O$. The point $(x_P, -y_P)$ is the negative of P, denoted as -P. For example, in $E_{23}(1, 1)$, for P = (13, 7), we have -P = (13, -7). But -7 mod 23 = 16. Therefore, -P = (13, 16).

3. If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ with $P \neq -Q$, then $R = P + Q = (x_R, y_R)$ is determined by the following rules:

$$x_R = (\lambda^2 - x_P - x_Q) \bmod p$$
$$y_R = (\lambda(x_P - x_R) - y_P) \bmod p$$

$$\lambda = \begin{cases} \left(\dfrac{y_Q - y_P}{x_Q - x_P}\right) \bmod p & \text{if } P \neq Q \\[2ex] \left(\dfrac{3x_P^2 + a}{2y_P}\right) \bmod p & \text{if } P = Q \end{cases}$$

4. Multiplication is defined as repeated addition; for example, 4P = P + P + P + P.

For example, let P = (3, 10) and Q = (9, 7) in $E_{23}(1, 1)$. Then

$$\lambda = \left(\frac{7 - 10}{9 - 3}\right) \bmod 23 = \left(\frac{-3}{6}\right) \bmod 23 = \left(\frac{-1}{2}\right) \bmod 23 = 11$$

$$x_R = (11^2 - 3 - 9) \bmod 23 = 109 \bmod 23 = 17$$

$$y_R = (11(3 - 17) - 10) \bmod 23 = -164 \bmod 23 = 20$$

So P + Q = (17, 20). To find 2P,

$$\lambda = \left(\frac{3(3^2) + 1}{2 \times 10}\right) \bmod 23 = \left(\frac{5}{20}\right) \bmod 23 = \left(\frac{1}{4}\right) \bmod 23 = 6$$

The last step in the preceding involves taking multiplicative inverse of 4 in Z_{23}.

This can be done using the extended Euclidean algorithm. To confirm, note that (6 * 4) mod 23 = 24 mod 23 = 1.

$$x_R = (6^2 - 3 - 3) \bmod 23 = 30 \bmod 23 = 7$$
$$y_R = (6(3 - 7) - 10) \bmod 23 = (-34) \bmod 23 = 12$$

and 2P = (7, 12).

Elliptic Curve Cryptography

- The addition operation in ECC is the counterpart of modular multiplication in RSA, and multiple addition is the counterpart of modular exponentiation.
- Consider the equation $Q = kP$ where $Q, P \in E_p(a, b)$ and $k < p$. It is relatively easy to calculate Q given k and P, but it is hard to determine k given Q and P.
- This is called the discrete logarithm problem for elliptic curves.

- For example, consider the group $E_{23}(9,17)$. This is the group 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.

Analog of Diffie-Hellman Key Exchange

- Key exchange using elliptic curves can be done in the following manner.
- First pick a large integer q, which is either a prime number p or an integer of the form 2^m, and elliptic curve parameters a and b for Equation (10.5) or Equation (10.7).
- This defines the elliptic group of points $E_q(a, b)$. Next, pick a base point $G = (x_1, y_1)$ in $E_p(a, b)$ whose order is a very large value n.
- The order n of a point G on an elliptic curve is the smallest positive integer n such that $nG = 0$.

A key exchange between users A and B can be accomplished as follows (Figure 10.7):

1. A selects an integer n_A less than n. This is A's private key. A then generates a public key $P_A = n_A \times G$; the public key is a point in $E_q(a, b)$.
2. B similarly selects a private key n_B and computes a public key P_B.
3. A generates the secret key $k = n_A \times P_B$. B generates the secret key $k = n_B \times P_A$.

Global Public Elements	
$E_q(a, b)$	elliptic curve with parameters a, b, and q, where q is a prime or an integer of the form 2^m
G	point on elliptic curve whose order is large value n

User A Key Generation	
Select private n_A	$n_A < n$
Calculate public P_A	$P_A = n_A \times G$

User B Key Generation

Select private n_B	$n_B < n$
Calculate public P_B	$P_B = n_B \times G$

Calculation of Secret Key by User A

$K = n_A \times P_B$

Calculation of Secret Key by User B

$K = n_B \times P_A$

Figure 10.7 ECC Diffie-Hellman Key Exchange

- The two calculations in step 3 produce the same result because

$$n_{\mathrm{A}} \times P_B = n_A \times (n_B \times G) = n_B \times (n_A \times G) = n_B \times P_A$$

- To break this scheme, an attacker would need to be able to compute k given G and kG, which is assumed to be hard.

- As an example, take p = 211; $E_p(0,-4)$, which is equivalent to the curve $y^2 = x^3 - 4$; and G=(2,2).
- One can calculate that 240G = O.
- A’s private key is $n_A = 121$, so A’s public key is $P_A = 121(2, 2) = (115, 48)$.
- B’s private key is $n_B = 203$, so B’s public key is 203(2, 3) = (130, 203).
- The shared secret key is 121(130, 203) = 203(115, 48) = (161, 69).

Elliptic Curve Encryption/Decryption

- The first task in this system is to encode the plaintext message *m* to be sent as an (x, y) point P_m.
- It is the point P_m that will be encrypted as a ciphertext and subsequently decrypted.
- Note that we cannot simply encode the message as the x or y coordinate of a point, because not all such coordinates are in $E_q(a,b)$.

- As with the key exchange system, an encryption/decryption system requires a point G and an elliptic group $E_q(a, b)$ as parameters.
- Each user A selects a private key n_A and generates a public key $P_A = n_A \times G$.
- To encrypt and send a message P_m to B, A chooses a random positive integer k and produces the ciphertext C_m consisting of the pair of points:

$$C_m = \{kG, P_m + kP_B\}$$

- Note that A has used B's public key P_B. To decrypt the ciphertext, B multiplies the first point in the pair by B's private key and subtracts the result from the second point:

$$P_m + kP_B - n_B(kG) = P_m + k(n_BG) - n_B(kG) = P_m$$

- A has masked the message P_m by adding kP_B to it. Nobody but A knows the value of k.
- For an attacker to recover the message, the attacker would have to compute k given G and kG, which is assumed to be hard.

- Consider an example. The global public elements are $q = 257$; $E_q(a, b) = E_{257}(0, -4)$, which is equivalent to the curve $y^2 = x^3 - 4$; and $G = (2, 2)$.
- Bob's private key is $n_B = 101$, and his public key is $P_B = n_B G = 101(2, 2) = (197, 167)$.
- Alice wishes to send a message to Bob that is encoded in the elliptic point $P_m = (112, 26)$.

- Alice chooses random integer $k = 41$ and computes $kG = 41(2, 2) = (136, 128)$, $kP_B = 41(197, 167) = (68, 84)$ and $P_m + kP_B = (112, 26) + (68, 84) = (246, 174)$.
- Alice sends the ciphertext $C_m = (C_1, C_2) = \{(136, 128), (246, 174)\}$ to Bob.
- Bob receives the ciphertext and computes $C_2 - n_B C_1 = (246, 174) - 101(136, 128) = (246, 174) - (68, 84) = (112, 26)$.