UNIT –2

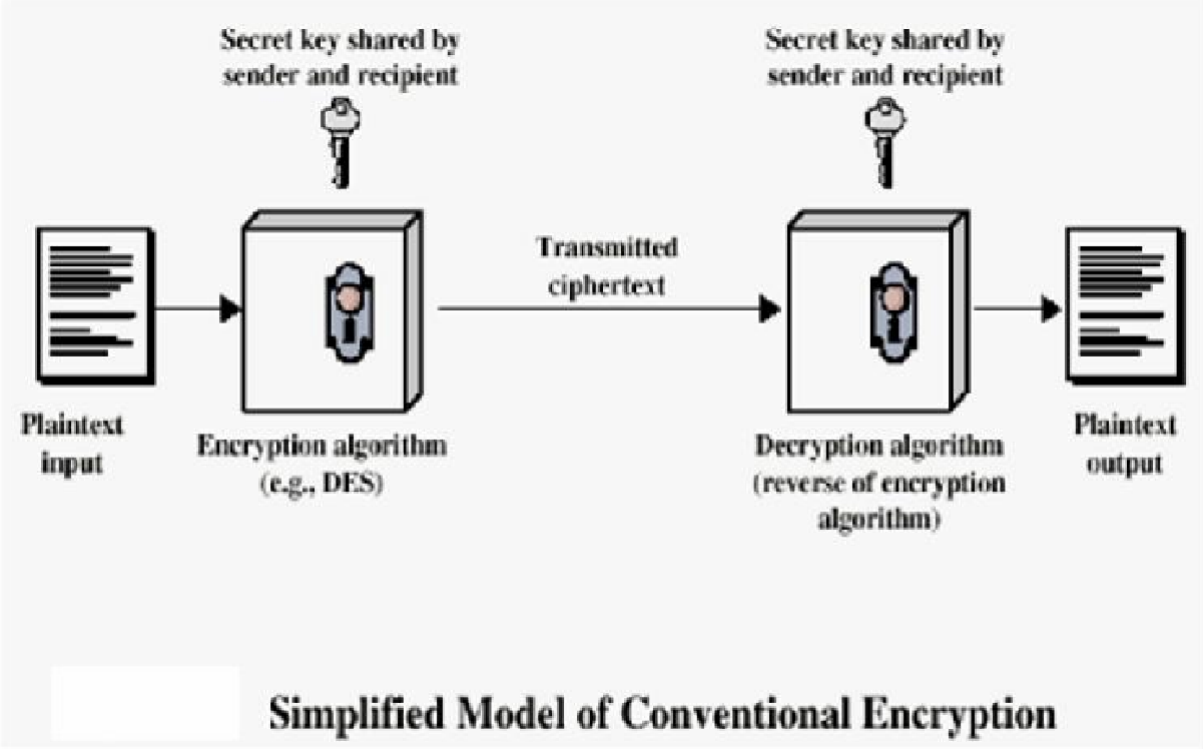
Symmetric Key Ciphers: Block Cipher Principles and Algorithms (DES, AES, and Blowfish), Differential and Linear Cryptanalysis, Block Cipher Modes of Operations, Stream Ciphers, RC4, Location and Placement of encryption function, Key Distribution.
Asymmetric Key Ciphers: *Principles of Public Key Cryptosystems, Algorithms (RSA, Diffie- Hellman, ECC), Key Distribution.*

CONVENTIONAL ENCRYPTION PRINCIPLES

A Conventional/Symmetric encryption scheme has five ingredients

1. ***Plain Text***: This is the original message or data which is fed into the algorithm as input.

2. ***Encryption Algorithm***: This encryption algorithm performs various substitutions and transformations on the plain text.
3. ***Secret Key***: The key is another input to the algorithm. The substitutions and transformations performed by algorithm depend on the key.
4. ***Cipher Text***: This is the scrambled (unreadable) message which is output of the encryption algorithm. This cipher text is dependent on plaintext and secret key. For a given plaintext, two different keys produce two different cipher texts.
5. ***Decryption Algorithm***: This is the reverse of encryption algorithm. It takes the cipher text and secret key as inputs and outputs the plain text.



Simplified Model of Conventional Encryption

The important point is that the security of conventional encryption depends on the secrecy of the key, not the secrecy of the algorithm i.e. it is not necessary to keep the algorithm secret, but only the key is to be kept secret. This feature that algorithm need

not be kept secret made it feasible for wide spread use and enabled manufacturers develop low cost chip implementation of data encryption algorithms. With the use of conventional algorithm, the principal security problem is maintaining the secrecy of the key.

FEISTEL CIPHER STRUCTURE

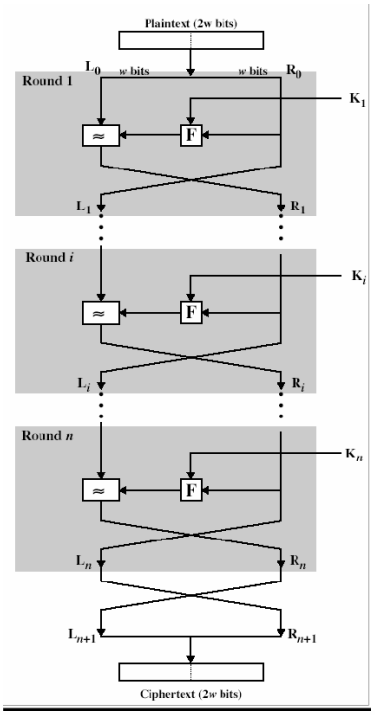
The input to the encryption algorithm are a plaintext block of length 2w bits and a key K. the plaintext block is divided into two halves L_0 and R_0. The two halves of the data pass through „n" rounds of processing and then combine to produce the ciphertext block. Each round „i" has inputs L_{i-1} and R_{i-1}, derived from the previous round, as well as the subkey K_i, derived from the overall key K. in general, the subkeys K_i are different from K and from each other.

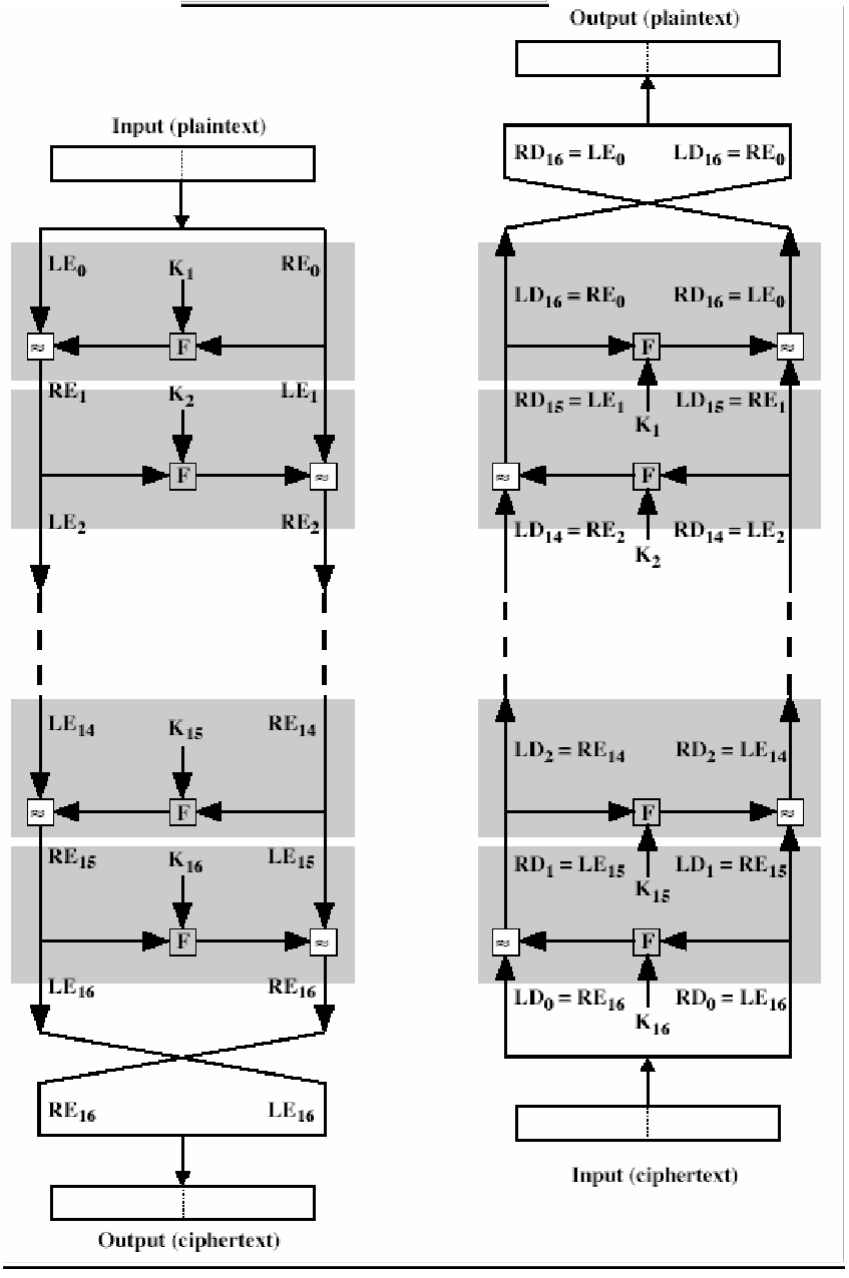
All rounds have the same structure. A substitution is performed on the left half of

the data (as similar to S-DES). This is done by applying a round function F to the right half

of the data and then taking the XOR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey k_i. Following this substitution, a permutation is performed that consists of the interchange of the two halves of the data. This structure is a particular form of the substitution-permutation network. The exact realization of a Feistel network depends on the choice of the following parameters and design features:

- **Block size** - Increasing size improves security, but slows cipher
- **Key size** - Increasing size improves security, makes exhaustive key searching harder, but may slow cipher
- **Number of rounds** - Increasing number improves security, but slows cipher
- **Subkey generation** - Greater complexity can make analysis harder, but slows cipher
- **Round function** - Greater complexity can make analysis harder, but slows cipher
- **Fast software en/decryption & ease of analysis -** are more recent concerns for practical use and testing


Plaintext (2w bits)
Round 1
Round i
Round n
Ciphertext (2w bits)


Input (plaintext)
Output (ciphertext)
Output (plaintext)
Input (ciphertext)

The process of decryption is essentially the same as the encryption process. The rule is as follows: use the cipher text as input to the algorithm, but use the subkey k_i in reverse order. i.e., k_n in the first round, k_{n-1} in second round and so on. For clarity, we use the notation LE_i and RE_i for data traveling through the decryption algorithm. The diagram below indicates that, at each round, the intermediate value of the decryption process is same (equal) to the corresponding value of the encryption process with two halves of the value swapped.
., RE_i || LE_i (or) equivalently RD_{16-i} || LD_{16-i}

After the last iteration of the encryption process, the two halves of the output are swapped, so that the cipher text is RE_{16} || LE_{16}. The output of that round is the cipher text. Now take the cipher text and use it as input to the same algorithm. The input to the first round is RE_{16} || $LE_{16,}$ which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process. Now we will see how the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process.
First consider the encryption process, $LE_{16} = RE_{15}$
$RE_{16} = LE_{15(+)} F (RE_{15,} K_{16})$

On the decryption side, $LD_1 = RD_0 = LE_{16} = RE_{15}$ $RD_1 = LD_{0 (+)} F (RD_{0,} K_{16})$
$= RE_{16} F (RE_{15,} K_{16})$
$= [LE_{15} F (RE_{15,} K_{16})] F (RE_{15,} K_{16})$
$= LE_{15}$
Therefore, $LD_1 = RE_{15}$ $RD_1 = LE_{15}$ In general, for the i_{th} iteration of the encryption algorithm, $LE_i = RE_{i-1}$ $RE_i = LE_{i-1} F (RE_{i-1}, K_i)$
Finally, the output of the last round of the decryption process is RE_0 || LE_0. A 32-bit swap recovers the original plaintext.

DEFINITIONS

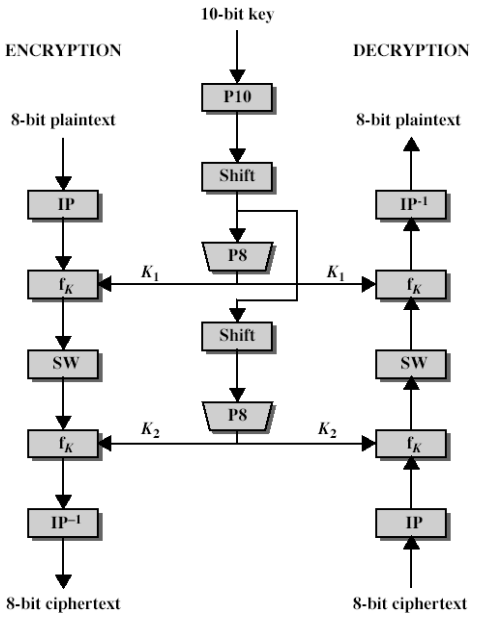
Encryption: Converting a text into code or cipher.
Converting computer data and messages into something, incomprehensible use a key, so that only a holder of the matching key can reconvert them.
Conventional or Symmetric or Secret Key or Single Key encryption:
Uses the same key for encryption & decryption.
Public Key encryption: Uses different keys for encryption & decryption

Conventional Encryption Principles

- An encryption scheme has five ingredients:

1. Plaintext – Original message or data.
2. Encryption algorithm – performs substitutions & transformations on plaintext.
3. Secret Key – exact substitutions & transformations depend on this
4. Ciphertext - output ie scrambled input.
5. Decryption algorithm - converts ciphertext back to plaintext.

SIMPLIFIED DATA ENCRYPTION STANDARD (S-DES)



The figure above illustrates the overall structure of the simplified DES. The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.

The encryption algorithm involves five functions:

- an initial permutation (IP)
- a complex function labeled f_k, which involves both permutation and substitution operations and depends on a key input
- a simple permutation function that switches (SW) the two halves of the data
- the function f_k again
- a permutation function that is the inverse of the initial permutation

The function f_k takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. Here a 10-bit key is used from which two 8-bit subkeys are generated. The key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K2).

The encryption algorithm can be expressed as a composition composition$_1$ of functions: $IP^{-1} \circ f_{K2} \circ SW \circ f_{k1} \circ IP$

Which can also be written as

$$Ciphertext = IP^{-1}(f_{K2}(SW(f_{k1}(IP(plaintext)))))$$

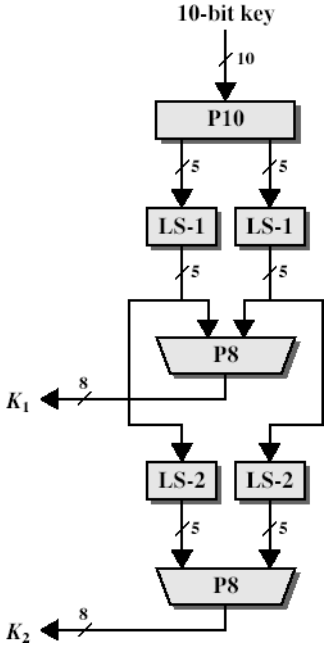
Where

K1 = P8 (Shift (P10 (Key)))

K2 = P8 (Shift (shift (P10 (Key))))

Decryption can be shown as

Plaintext = IP^{-1} (f_{K1} (SW (f_{k2} (IP (ciphertext)))))



S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm. First, permute the key in the following fashion. Let the 10-bit key be designated as (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10). Then the permutation P10 is defined as:

P10 (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10) = (k3, k5, K2, k7, k4, k10 10, k1, k9, k8, k6)

P10

can be concisely defined by the display:

P10									
3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on. For example, the key (1010000010) is permuted to (10000 01100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000). Next we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

The result is subkey 1 (K1). In our example, this yields (10100100). We then go back to the pair of 5-bit strings produced by the two LS-1 functions and performs a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce K2. In our example, the result is (01000011).

S-DES encryption

Encryption involves the sequential application of five functions.

Initial and Final Permutations The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function:

IP							
2	6	3	1	4	8	5	7

This retains all 8 bits of the plaintext but mixes them up. Consider the plaintext to be 11110011.

Permuted output = 10111101

At the end of the algorithm, the inverse permutation is used:

IP –1							
4	1	3	5	7	2	8	6

The Function f_k

The most complex component of S-DES is the function f_k, which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let L and R be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to f K, and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings.

Then we let f_k(L, R) = (L (+) F(R, SK), R)

Where SK is a subkey and (+) is the bit-by-bit exclusive-OR function.

e.g., permuted output = 1011 1101 and suppose F (1101, SK) = (1110) for some key SK. Then f K(10111101) = 10111110, 1101 = 01011101

We now describe the mapping F. The input is a 4-bit number (n1 n2 n3 n4). The first operation is an expansion/permutation operation:

E/P							
4	1	2	3	2	3	4	1

R= 1101 E/P output = 11101011 It is clearer to depict the result in this fashion:

$$\begin{array}{c|cc|c} n_4 & n_1 & n_2 & n_3 \\ n_2 & n_3 & n_4 & n_1 \end{array}$$

The 8-bit subkey K1 = (k11, k12 12, k13 13, k14 14, k15 15, k16 16, k17 17, k18) is added to this value using exclusive-OR:

$$\begin{array}{c|cc|c} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_2 \oplus k_{13} & n_3 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{array}$$

Let us rename these 8 bits:

$$\begin{array}{c|cc|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \end{array}$$

The first 4 bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2-bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2-bit output.

These two boxes are defined as follows:

$$S0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

The S-boxes operate as follows. The first and fourth input bits are treated as a 2-bit number that specify a row of the S-box, and the second and third input bits specify a column of the S-box. The entry in that row and column, in base 2, is the 2-bit output. For example, if (p0,0 p0,3) =) (00) and (p0,1 p0,2) = (10), then the output is from row 0, column 2 of S0, which is 3, or (11) in) binary. Similarly, (p1,0 p1,3) and (p1,1 p1,2) are used to index into a row and column of S1 to produce an additional 2 bits. Next, the 4 bits produced by S0 and S1 undergo a further permutation as follows:

P4			
2	4	3	1

The output of P4 is the output of the function F.

The Switch Function The function f K only alters the leftmost 4 bits of the input. The switch function (SW) interchanges the left and right 4 bits so that the second instance of f K operates

on a different 4 bits. In this second instance, the E/P, S0, S1, and P4 functions are the same. The key input is K2. Finally apply inverse permutation to get the ciphertext

DATA ENCRYPTION STANDARD (DES)

The main standard for encrypting data was a symmetric algorithm known as the Data Encryption Standard (DES). However, this has now been replaced by a new standard known as the Advanced Encryption Standard (AES) which we will look at later. DES is a 64 bit block cipher which means that it encrypts data 64 bits at a time. This is contrasted to a stream cipher in which only one bit at a time (or sometimes small groups of bits such as a byte) is encrypted. DES was the result of a research project set up by International Business Machines (IBM) Corporation in the late 1960's which resulted in a cipher known as LUCIFER. In the early 1970's it was decided to commercialize LUCIFER and a number of significant changes were introduced. IBM was not the only one involved in these changes as they sought technical advice from the National Security Agency (NSA) (other outside consultants were involved but it is likely that the NSA were the major contributors from a technical point of view). The altered version of LUCIFER was put forward as a proposal for the new national encryption standard requested by the National Bureau of Standards (NBS)3 . It was finally adopted in 1977 as the Data Encryption Standard - DES (FIPS PUB 46). Some of the changes made to LUCIFER have been the subject of much controversy even to the present day. The most notable of these was the key size. LUCIFER used a key size of 128 bits however this was reduced to 56 bits for DES. Even though DES actually accepts a 64 bit key as input, the remaining eight bits are used for parity checking and have no effect on DES's security. Outsiders were convinced that the 56 bit key was an easy target for a brute force attack4 due to its extremely small size. The need for the parity checking scheme was also questioned without satisfying answers. Another controversial issue was that the S-boxes used were

designed under classified conditions and no reasons for their particular design were ever given. This led people to assume that the NSA had introduced a "trapdoor" through which they could decrypt any data encrypted by DES even without knowledge of the key. One startling discovery was that the S-boxes appeared to be secure against an attack known as Differential Cryptanalysis which was only publicly discovered by Biham and Shamir in 1990. This suggests that the NSA were aware of this attack in 1977; 13 years earlier! In
fact the DES designers claimed that the reason they never made the design specifications for the S-boxes available was that they knew about a number of attacks that weren't public knowledge at the time and they didn't want them leaking - this is quite a plausible claim as differential cryptanalysis has shown. However, despite all this controversy, in 1994 NIST reaffirmed DES for government use for a further five years for use in areas other than "classified". DES of course isn't the only symmetric cipher. There are many others, each with varying levels of complexity. Such ciphers include: IDEA, RC4, RC5, RC6 and the new Advanced Encryption Standard (AES). AES is an important algorithm and was originally meant to replace DES (and its more secure variant triple DES) as the standard algorithm for non-classified material. However as of 2003, AES with key sizes of 192 and 256 bits has been found to be secure enough to protect information up to top secret. Since its creation, AES had underdone intense scrutiny as one would expect for an algorithm that is to be used as the standard. To date it has withstood all attacks but the search is still on and it remains to be seen whether or not this will last. We will look at AES later in the course.

INNER WORKING OF DES

DES (and most of the other major symmetric ciphers) is based on a cipher known as the Feistel block cipher. It consists of a number of rounds where each round contains bit-shuffling, non- linear substitutions (S-boxes) and exclusive OR operations. As with most encryption schemes, DES expects two inputs - the plaintext to be encrypted and the secret key. The manner in which the plaintext is accepted, and the key arrangement used for encryption and decryption, both determine the type of cipher it is. DES is therefore a symmetric, 64 bit block cipher as it uses the same key for both encryption and decryption and only operates on 64 bit blocks of data at a time5 (be they plaintext or ciphertext). The key size used is 56 bits, however a 64 bit (or eight-byte) key is actually input. The least significant bit of each byte is either used for parity (odd for DES) or set arbitrarily and does not increase the security in any way. All blocks are numbered from left to right which makes the eight bit of each byte the parity bit.
Once a plain-text message is received to be encrypted, it is arranged into 64 bit blocks required for input. If the number of bits in the message is not evenly divisible by 64, then the last block will be padded. Multiple permutations and substitutions are incorporated throughout in order to increase the difficulty of performing a cryptanalysis on the cipher

OVERALL STRUCTURE

Figure below shows the sequence of events that occur during an encryption operation. DES performs an initial permutation on the entire 64 bit block of data. It is then split into 2, 32 bit sub-blocks, Li and Ri which are then passed into what is

known as a round (see figure 2.3), of which there are 16 (the subscript i in Li and Ri indicates the current round). Each of the rounds are identical and the effects of increasing their number is twofold - the algorithms security is increased and its temporal efficiency decreased. Clearly these are two conflicting outcomes and a compromise must be made. For DES the number chosen was 16, probably to guarantee the elimination of any correlation between the ciphertext and either the plaintext or key6 . At the end of the 16th round, the 32 bit Li and Ri output quantities are swapped to create what is known as the pre-output. This [R16, L16] concatenation is permuted using a function which is the exact inverse of the initial permutation. The output of this final permutation is the 64 bit ciphertext.

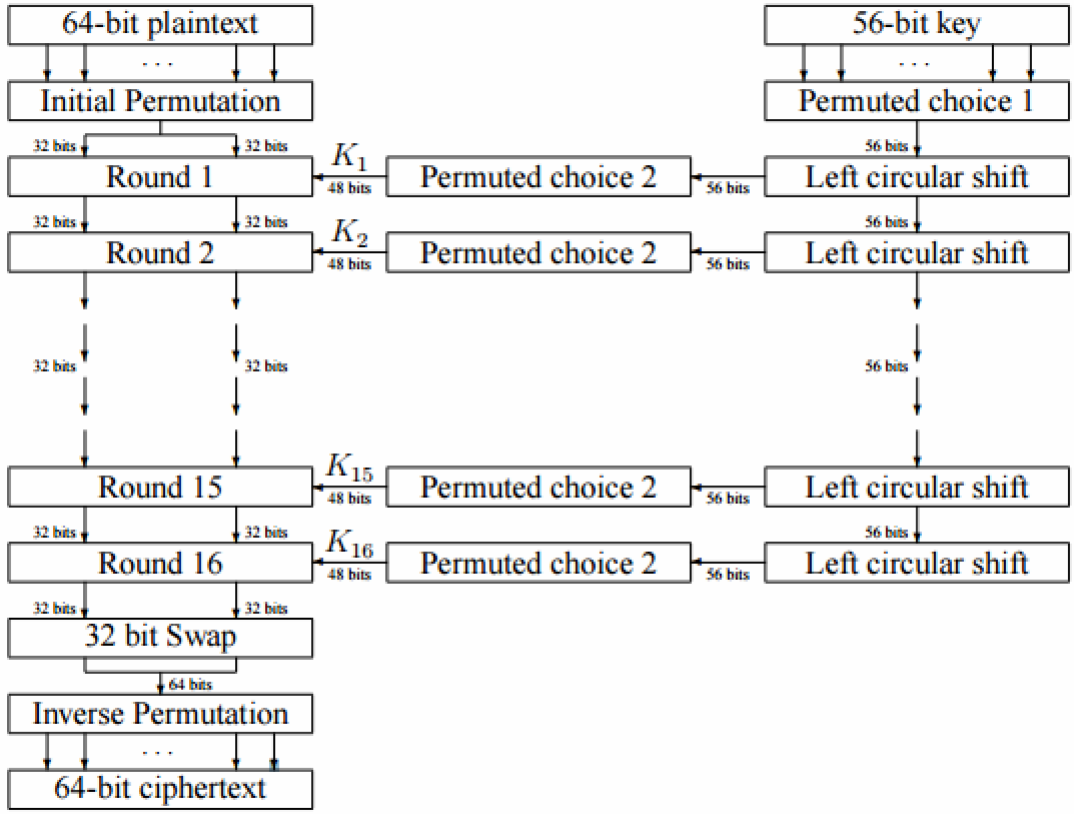


Figure : Flow Diagram of DES algorithm for encrypting data.

So in total the processing of the plaintext proceeds in three phases as can be seen from the left hand side of figure

1. Initial permutation (IP - defined in table 2.1) rearranging the bits to form the "permuted input".
2. Followed by 16 iterations of the same function (substitution and permutation). The output of the last iteration consists of 64 bits which is a function of the plaintext and key. The left and right halves are swapped to produce the preoutput.
3. Finally, the preoutput is passed through a permutation (IP−1 - defined in table 2.1) which is simply the inverse of the initial permutation (IP). The output of IP−1 is the 64- bit ciphertext

(a) Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(b) Inverse Initial Permutation (IP^{-1})

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

(c) Expansion Permutation (E)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

(d) Permutation Function (P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Table 2.1: Permutation tables used in DES.

As figure shows, the inputs to each round consist of the Li , Ri pair and a 48 bit subkey which is a shifted and contracted version of the original 56 bit key. The use of the key can be seen in the right hand portion of figure 2.2: • Initially the key is passed through a permutation function (PC1 - defined in table 2.2) • For each of the 16 iterations, a subkey (Ki) is produced by a combination of a left circular shift and a permutation (PC2 - defined in table 2.2) which is the same for each iteration. However, the resulting subkey is different for each iteration because of repeated shifts.

(a) Input Key

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

(b) Permuted Choice One (PC-1)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

(c) Permuted Choice Two (PC-2)

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

(d) Schedule of Left Shifts

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 2.2: DES key schedule.

DETAILS OF INDIVIDUAL ROUNDS

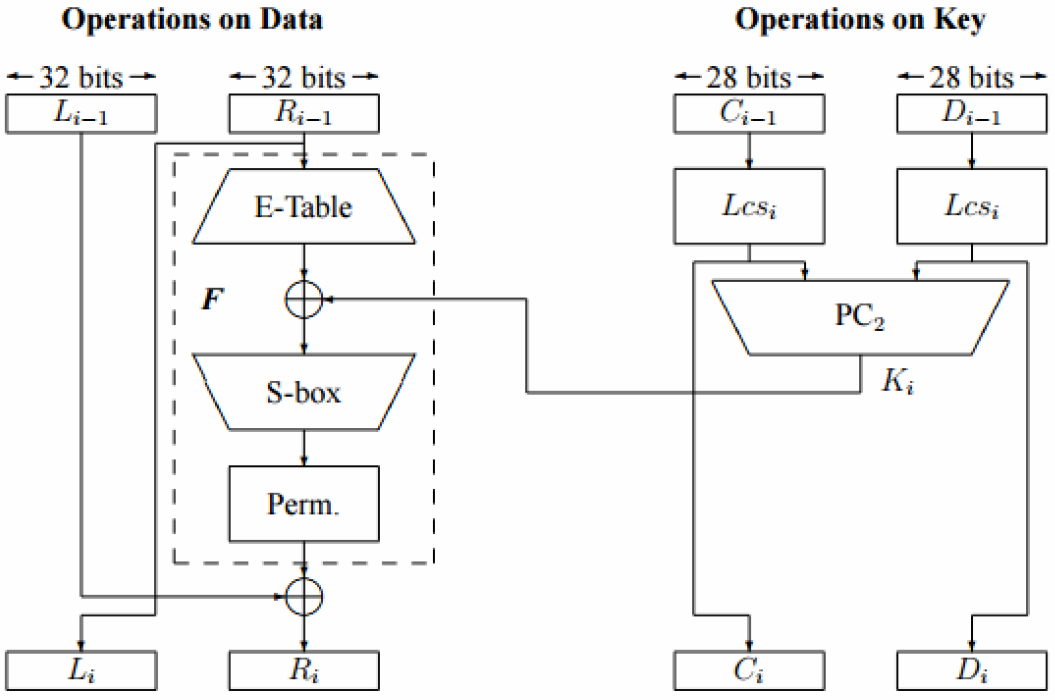


Figure : Details of a single DES round.

The main operations on the data are encompassed into what is referred to as the cipher function and is labeled F. This function accepts two different length inputs of 32 bits and 48 bits and outputs a single 32 bit number. Both the data and key are operated on in parallel, however the operations are quite different. The 56 bit key is split into two 28 bit halves Ci and Di (C and D being chosen so as not to be confused with L and R). The value of the key used in any round is simply a left cyclic shift and a permuted contraction of that used in the previous round.
Mathematically, this can be written as

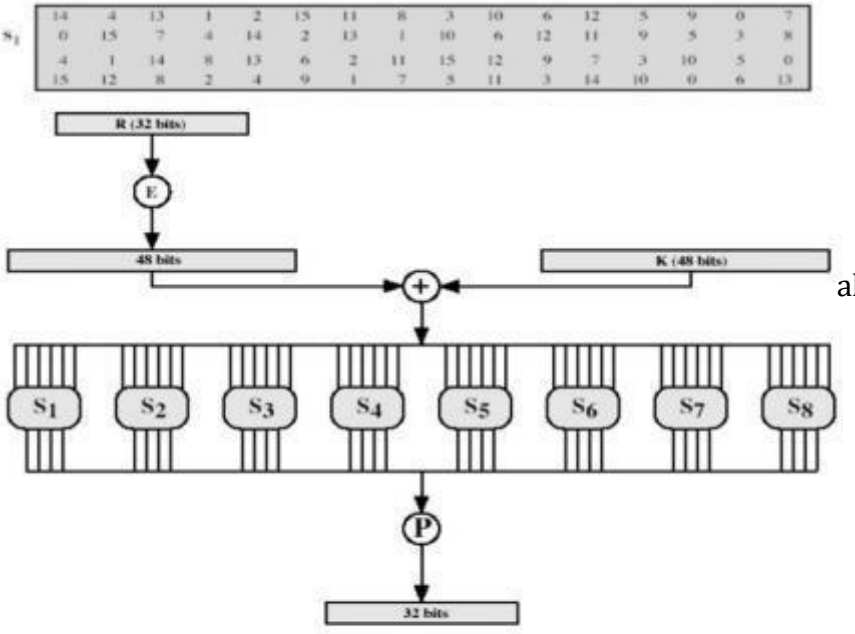
$$Ci = Lcsi(Ci-1),$$
$$Di = Lcsi(Di-1)$$
$$Ki = P\,C2(Ci, Di)$$

where Lcsi is the left cyclic shift for round i, Ci and Di are the outputs after the shifts, P C2(.) is a function which permutes and compresses a 56 bit number into a 48 bit number and Ki is the actual key used in round i. The number of shifts is either one or two and is determined by the round number i. For i = {1, 2, 9, 16} the number of shifts is one and for every other round it is two

OX Details

ADVANCED ENCRYPTION ALGORITHM (AES)

- AE
- AE
  Mo
  bits
- Enc
  12
- Exc al.
- Ea
  ste
  ar
  wh
  and
- To
  bes
  byt

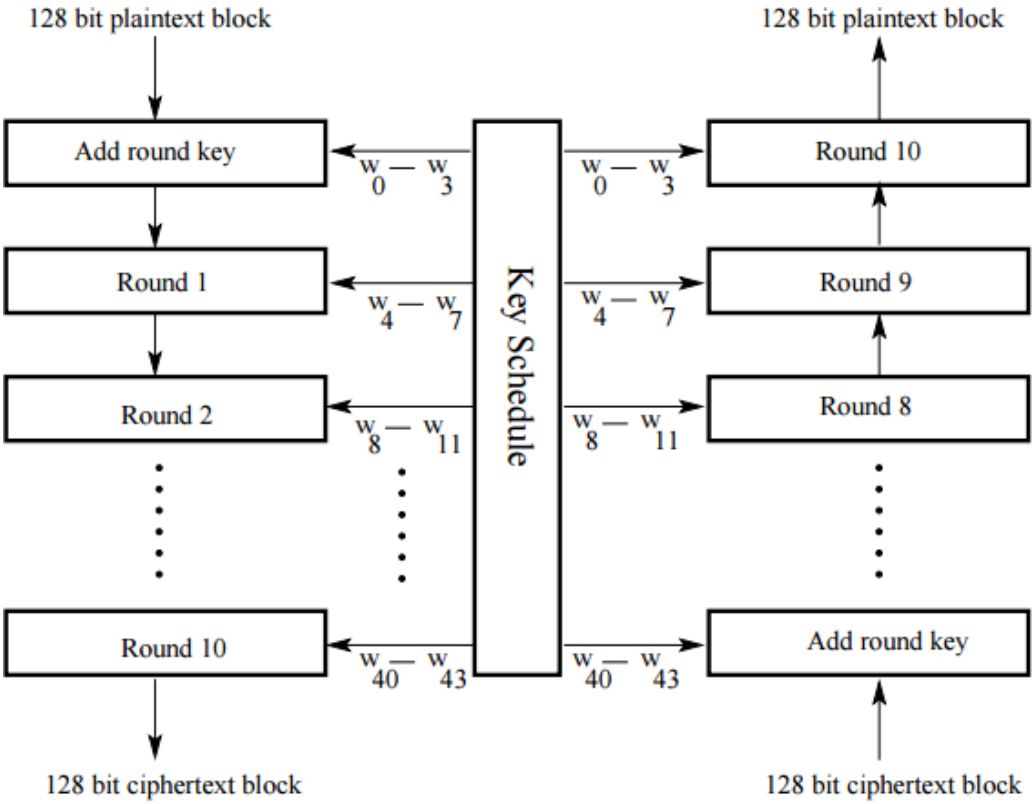


$$\begin{bmatrix} byte_0 & byte_4 & byte_8 & byte_{12} \\ byte_1 & byte_5 & byte_9 & byte_{13} \\ byte_2 & byte_6 & byte_{10} & byte_{14} \\ byte_3 & byte_7 & byte_{11} & byte_{15} \end{bmatrix}$$

of the DES algorithm.

tails.

Therefore, the first four bytes of a 128-bit input block occupy the first column in the 4 × 4 matrix of bytes. The next four bytes occupy the second column, and so on.
The 4×4 matrix of bytes shown above is referred to as the state array in AES.



AES Encryption AES Decryption

The algorithm begins with an Add round key stage followed by 9 rounds of four stages and a tenth round of three stages.

This applies for both encryption and decryption with the exception that each stage of a round the decryption algorithm is the inverse of its counterpart in the encryption algorithm.

The four stages are as follows: 1. Substitute bytes 2. Shift rows 3. Mix Columns 4. Add Round Key

Substitute Bytes

• This stage (known as SubBytes) is simply a table lookup using a 16 × 16 matrix of byte values called an s-box.

• This matrix consists of all the possible combinations of an 8 bit sequence (28 = 16 × 16

= 256).

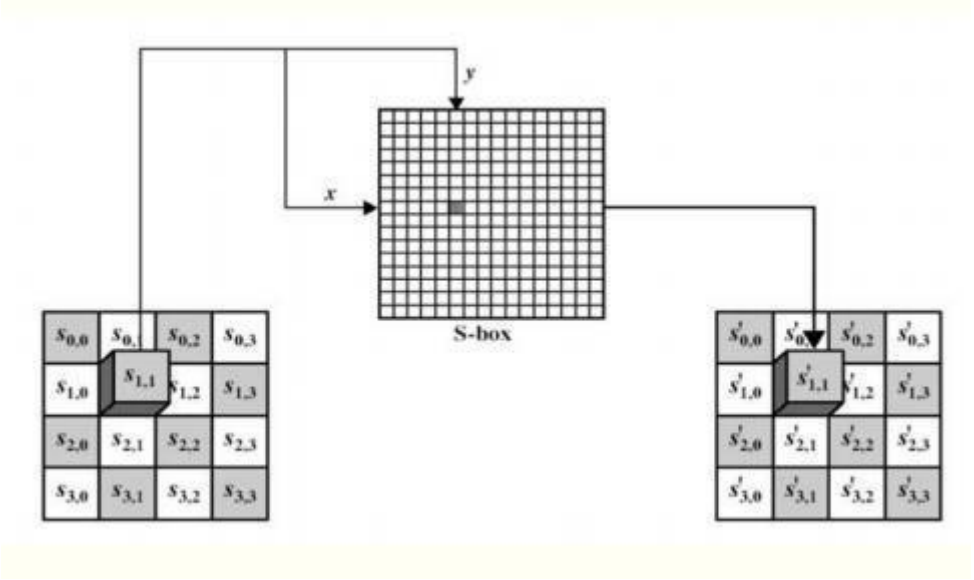
• However, the s-box is not just a random permutation of these values and there is a well defined method for creating the s-box tables.

• The designers of Rijndael showed how this was done unlike the s-boxes in DES for which no rationale was given.Our concern will be how state is effected in each round.

• For this particular round each byte is mapped into a new byte in the following way: the leftmost nibble of the byte is used to specify a particular row of the s-box and the rightmost nibble specifies a column.

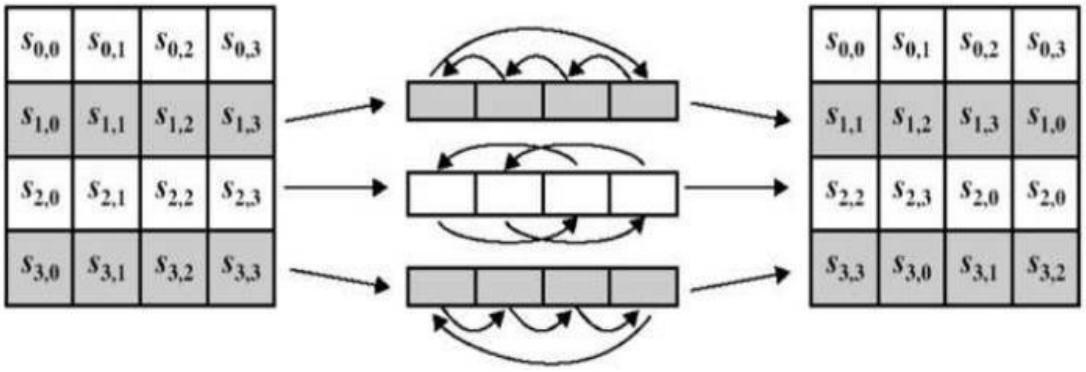
• For example, the byte {95} (curly brackets represent hex values in FIPS PUB 197) selects row 9 column 5 which turns out to contain the value {2A}.

• This is then used to update the state matrix.



Shift Row Transformation

• This stage (known as ShiftRows) is shown in figure below.
• Simple permutation an nothing more.
• It works as follow: – The first row of state is not altered. – The second row is shifted 1 bytes to the left in a circular manner. – The third row is shifted 2 bytes to the left in a circular manner. – The fourth row is shifted 3 bytes to the left in a circular manner.

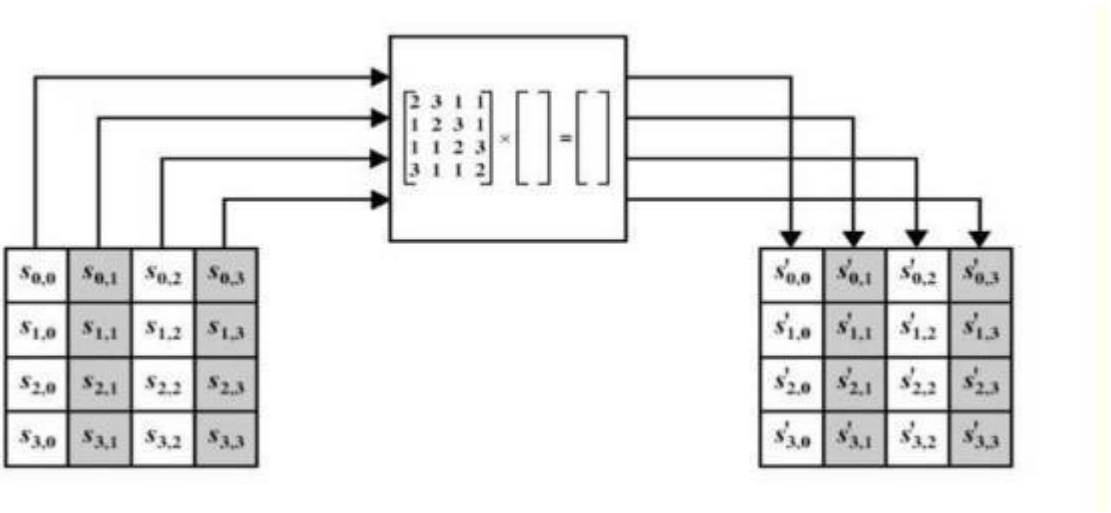


MIX COLUMN TRANSFORMATION

• This stage (known as MixColumn) is basically a substitution
• Each column is operated on individually. Each byte of a column is mapped into a new value that is a function of all four bytes in the column.
• The transformation can be determined by the following matrix multiplication on state
• Each element of the product matrix is the sum of products of elements of one row and one column.
• In this case the individual additions and multiplications are performed in GF(28).
• The MixColumns transformation of a single column j (0 ≤ j ≤ 3) of state can be expressed as:

s ′ 0,j = (2 • s0,j) ⊕ (3 • s1,j) ⊕ s2,j ⊕ s3,j s ′ 1,j = s0,j ⊕ (2 • s1,j) ⊕ (3 • s2,j) ⊕ s3,j s ′ 2,j = s0,j ⊕ s1,j ⊕ (2 • s2,j) ⊕ (3 • s3,j) s ′ 3,j = (3 • s0,j) ⊕

$s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j})$



ADD ROUND KEY TRANSFORMATION

• In this stage (known as AddRoundKey) the 128 bits of state are bitwise XORed with the 128 bits of the round key.

• The operation is viewed as a columnwise operation between the 4 bytes of a state column and one word of the round key.

• This transformation is as simple as possible which helps in efficiency but it also effects every bit of state.

• The AES key expansion algorithm takes as input a 4-word key and produces a linear array of 44 words. Each round uses 4 of these words as shown in figure.

• Each word contains 32 bytes which means each subkey is 128 bits long. Figure 7 show pseudocode for generating the expanded key from the actual key.

BLOWFISH ALGORITHM

- a symmetric block cipher designed by Bruce Schneier in 1993/94
- characteristics
  - fast implementation on 32-bit CPUs
  - compact in use of memory
  - simple structure for analysis/implementation
  - variable security by varying key size
- has been implemented in various products

BLOWFISH KEY SCHEDULE

- uses a 32 to 448 bit key, 32-bit words stored in K-array K_j ,j from 1 to 14
- used to generate
  - 18 32-bit subkeys stored in P array, P_1 ….P_{18}
  - four 8x32 S-boxes stored in $S_{i,j}$, each with 256 32-bit entries

Subkeys and S-Boxes Generation:

1. initialize P-array and then 4 S-boxes in order using the fractional part of pi P_1 (left most 32-bit), and so on,,, $S_{4,255.}$
2. XOR P-array with key-Array (32-bit blocks) and reuse as needed: assume we have up to k_{10} then P_{10} XOR $K_{10,}$, P_{11} XOR K_1 … P_{18} XOR K_8
3. Encrypt 64-bit block of zeros, and use the result to update P_1 and $P_{2.}$

4. encrypting output form previous step using current P & S and replace P_3 and P_4. Then encrypting current output and use it to update successive pairs of P.
5. After updating all P's (last :P_{17} P_{18}), start updating S values using the encrypted output from previous step.
    - requires 521 encryptions, hence slow in re-keying
    - Not suitable for limited-memory applications.

BLOWFISH ENCRYPTION

- uses two main operations: addition modulo 2^{32}, and XOR
- data is divided into two 32-bit halves L_0 & R_0

for i = 1 to 16 do

$R_i = L_{i-1}$ XOR P_i;

$L_i = F[R_i]$ XOR R_{i-1};

$L_{17} = R_{16}$ XOR P_{18};

$R_{17} = L_{16}$ XOR P_1

7;

- where

$F[a,b,c,d] = ((S_{1,a} + S_{2,b}) \text{ XOR } S_{3,c}) + S_{4,d}$

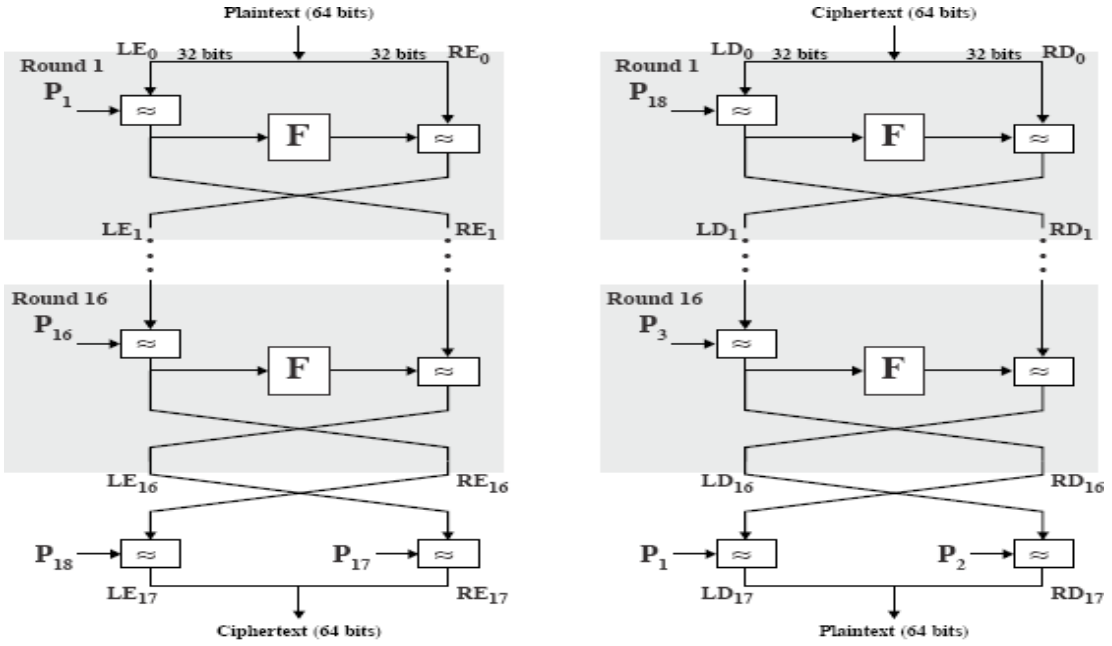


Figure 6.3 Blowfish Encryption and Decryption

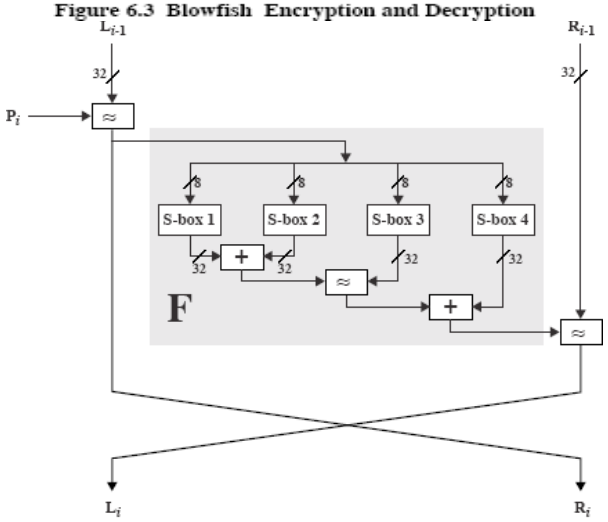


Figure 6.4 Detail of Single Blowfish Round

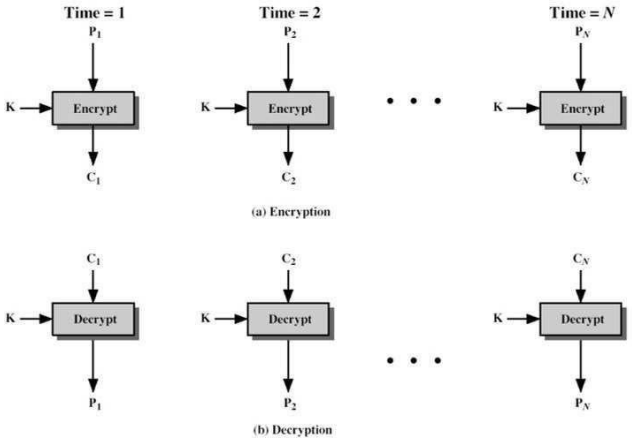
BLOCK CIPHER MODES OF OPERATIONS

- Direct use of a block cipher is inadvisable
- Enemy can build up "code book" of plaintext/ciphertext equivalents
- Beyond that, direct use only works on messages that are a multiple of the cipher block size in length
- Solution: five standard Modes of Operation: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR).

Electronic Code Book

- Direct use of the block cipher
- Used primarily to transmit encrypted keys
- Very weak if used for general-purpose encryption; never use it for a file or a message.
- Attacker can build up codebook; no semantic security

• We write {P}k → C to denote "encryption of plaintext P with key k to produce ciphertext C"



Cipher Block Chaining

• We would like that same plaintext blocks produce different ciphertext blocks.

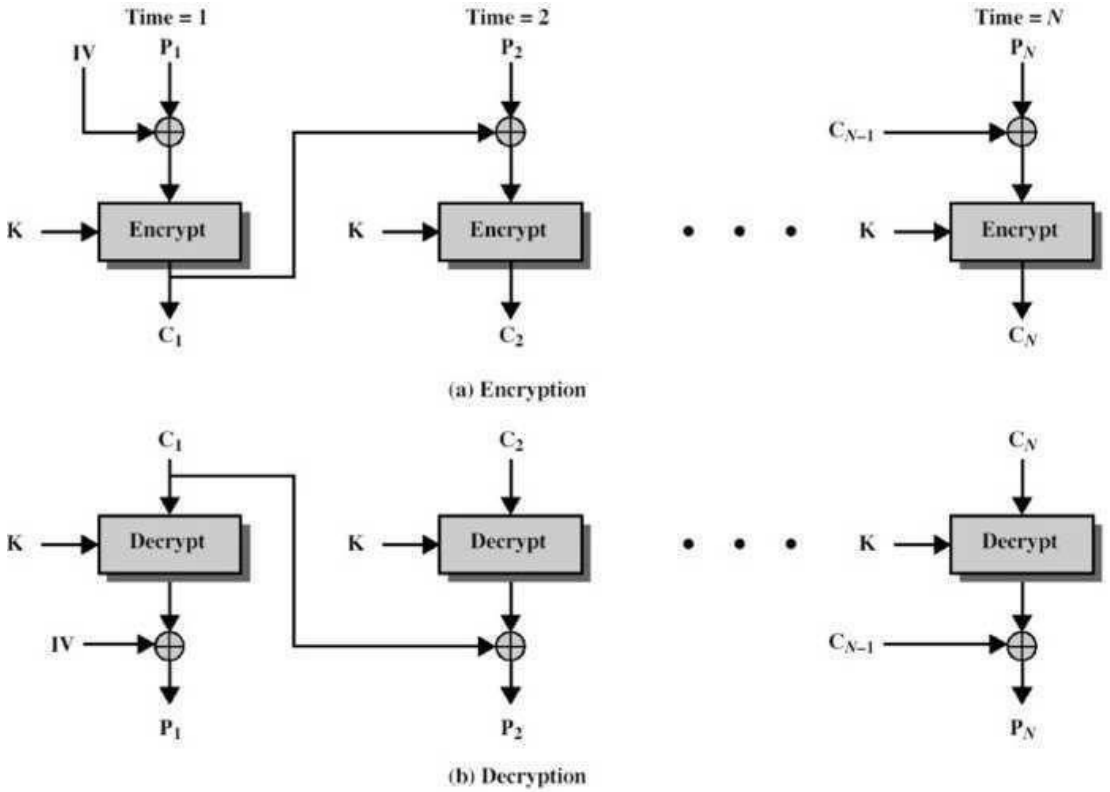
• Cipher Block Chaining (see figure) allows this by XORing each plaintext with the Ciphertext from the previous round (the first round using an Initialisation Vector (IV)).

• As before, the same key is used for each block.

• Decryption works as shown in the figure because of the properties of the XOR operation,

i.e. IV ⊕ IV ⊕ P = P where IV is the Initialisation Vector and P is the plaintext.

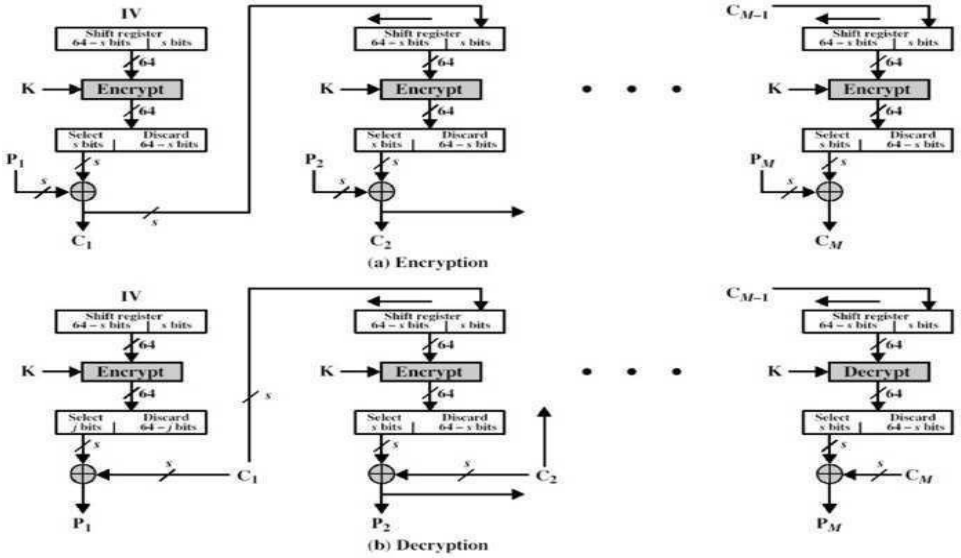
• Obviously the IV needs to be known by both sender and receiver and it should be kept secret along with the key for maximum security.



Cipher Feedback (CFB) Mode

• The Cipher Feedback and Output Feedback allows a block cipher to be converted into a stream cipher.

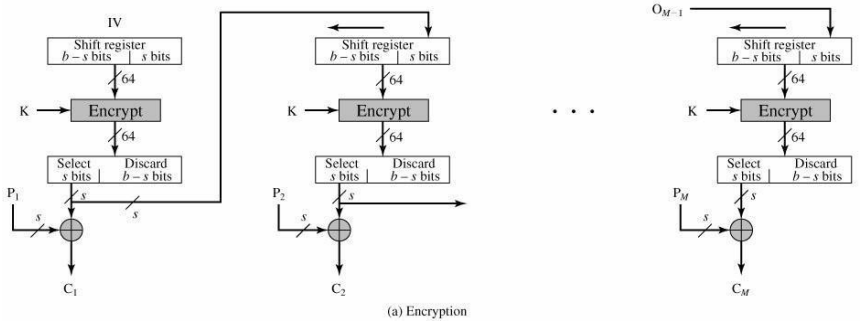
• This eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time.
• Figure shows the CFB scheme.
• In this figure it assumed that the unit of transmission is s bits; a common value is s = 8.
• As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext (which is split into s bit segments).
• The input to the encryption function is a shift register equal in length to the block cipher of the algorithm (although the diagram shows 64 bits, which is block size used by DES, this can be extended to other block sizes such as the 128 bits of AES).
• This is initially set to some Initialisation Vector (IV).

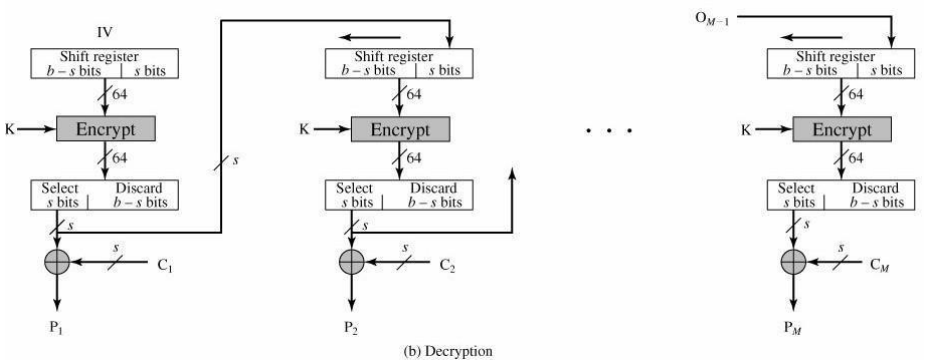


OUTPUT FEEDBACK (OFB) MODE

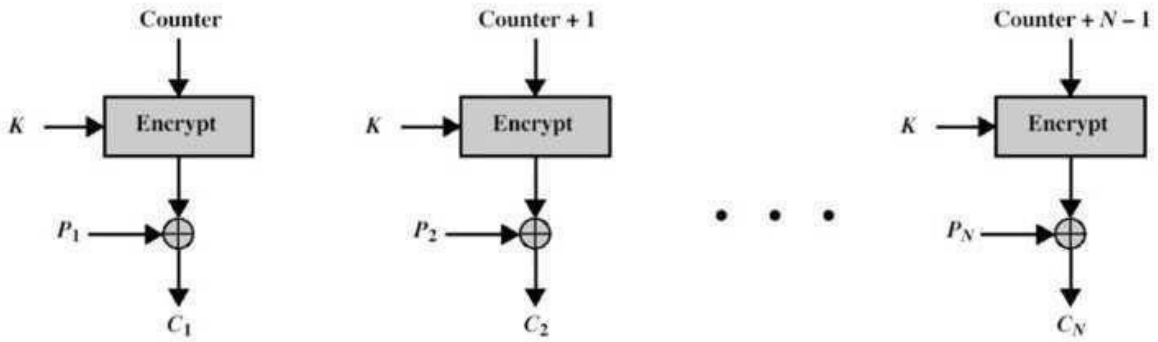
- The Output Feedback Mode is similar in structure to that of CFB, as seen in figure 13.
- As can be seen, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register.
- One advantage of the OFB method is that bit errors in transmission do not propagate.
- For example, if a bit error occurs in C1 only the recovered value of P1 is affected; subsequent plaintext units are not corrupted.

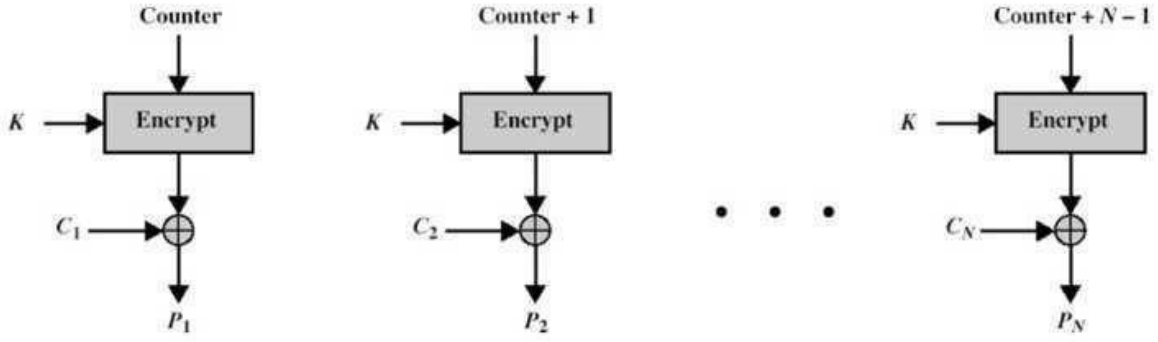
With CFB, C1 also serves as input to the shift register and therefore causes additional corruption downstream.





Counter Mode





PUBLIC KEY CRYPTOGRAPHY

The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography. It is *asymmetric*, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. Public key schemes are neither more nor less secure than private key (security depends on the key size for both). Public-key cryptography *complements rather than replaces* symmetric cryptography. Both also have issues with key distribution, requiring the use

of some suitable protocol. The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption:

1.) ***key distribution*** – how to have secure communications in general without having to trust a KDC with your key

2.) ***digital signatures*** – how to verify a message comes intact from the claimed sender

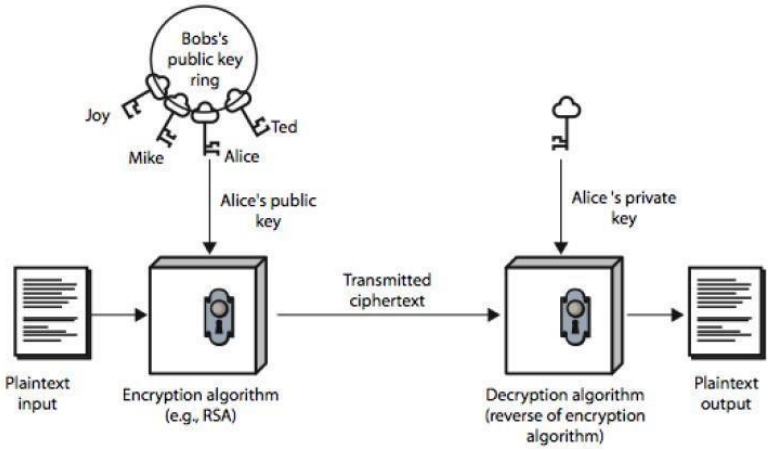
Public-key/two-key/asymmetric cryptography involves the use of **two** keys:

- a *public-key*, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
- a *private-key*, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**.
- is **asymmetric** because those who encrypt messages or verify signatures cannot decrypt messages or create signatures

Public-Key algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristics:

- it is computationally infeasible to find decryption key knowing only algorithm & encryption key
- it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
- either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms like RSA)

The following figure illustrates public-key encryption process and shows that a public-key encryption scheme has six ingredients: plaintext, encryption algorithm, public & private keys, ciphertext & decryption algorithm.



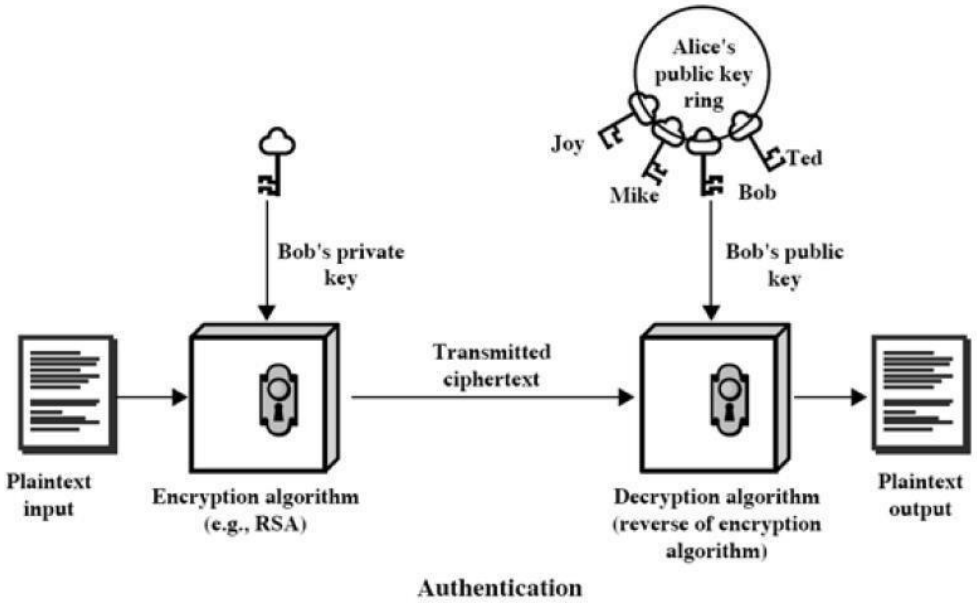
The essential steps involved in a public-key encryption scheme are given below: 1.) Each user generates a pair of keys to be used for encryption and decryption.
2.) Each user places one of the two keys in a public register and the other key is kept private.
3.) If B wants to send a confidential message to A, B encrypts the message using A's public key.
4.) When A receives the message, she decrypts it using her private key. Nobody else can decrypt the message because that can only be done using A's private key (Deducing a private key should be infeasible).
5.) If a user wishes to change his keys –generate another pair of keys and publish the public one: no interaction with other users is needed.
Notations used in Public-key cryptography:

- The public key of user A will be denoted **KUA**.
- The private key of user A will be denoted **KRA**.
- Encryption method will be a function E.
- Decryption method will be a function D.
- If B wishes to send a plain message X to A, then he sends the cryptotext

$Y=E(KU_A,X)$

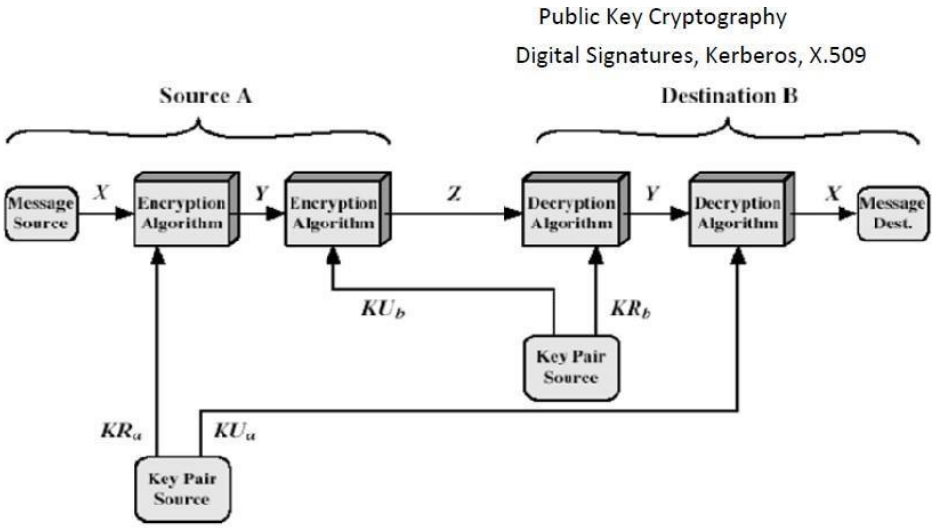
- The intended receiver A will decrypt the message: $D(KR_A,Y)=X$

The first attack on Public-key Cryptography is the attack on Authenticity. **An attacker may impersonate user B**: he sends a message $E(KU_A,X)$ and claims in the message to be B –A has no guarantee this is so. To overcome this, B will encrypt the message using his private key: $Y=E(KR_B,X)$. Receiver decrypts using B's public key KR_B. This shows the authenticity of the sender because (supposedly) he is the only one who knows the private key. The entire encrypted message serves as a digital signature. This scheme is depicted in the following figure:



Authentication

But, a drawback still exists. Anybody can decrypt the message using B's public key. So, secrecy or confidentiality is being compromised. One can provide both *authentication and confidentiality* using the public-key scheme twice:



Public-Key Cryptosystem: Secrecy and Authentication

B encrypts X with his private key: Y=E(KRB,X) B encrypts Y with A's public key: Z=E(KUA,Y)

A will decrypt Z (and she is the only one capable of doing it): Y=D(KRA,Z)

A can now get the plaintext and ensure that it comes from B (he is the only one who knows his private key): decrypt Y using B's public key: X=E(KUB,Y).

Applications for public-key cryptosystems:

1.) **Encryption/decryption**: sender encrypts the message with the receiver's public key.

2.) **Digital signature**: sender "signs" the message (or a representative part of the message) using his private key

3.) **Key exchange**: two sides cooperate to exchange a secret key for later use in a secret- key cryptosystem.

The main requirements of Public-key cryptography are:

1. Computationally easy for a party B to generate a pair (public key KUb, private key KRb).
2. Easy for sender A to generate ciphertext:
3. Easy for the receiver B to decrypt ciphertect using private key:
4. Computationally infeasible to determine private key (KRb) knowing public key (KUb)
5. Computationally infeasible to recover message M, knowing KUb and ciphertext C
6. Either of the two keys can be used for encryption, with the other used for decryption:

$$\mathbf{M= D_{KRb}[E_{KUb}(M)]=D_{KUb}[E_{KRb}(M)]}$$

Easy is defined to mean a problem that can be solved in polynomial time as a function of input length. A problem is infeasible if the effort to solve it grows faster than polynomial time as a function of input size. Public-key cryptosystems usually rely on difficult math functions rather than S-P networks as classical cryptosystems. **One-way function** is one, easy to calculate in one direction, infeasible to calculate in the other direction (i.e., the inverse is infeasible to compute). **Trap-door function** is a difficult function that becomes easy if some extra information is known. Our aim to find 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.

Security of Public-key schemes:

- Like private key schemes brute force **exhaustive search** attack is always theoretically possible. But keys used are too large (>512bits).
- Security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalyse) problems. More generally the **hard** problem is known, its just made too hard to do in practise.
- Requires the use of **very large numbers,** hence is **slow** compared to private key schemes

RSA ALGORITHM

RSA is the best known, and by far the most widely used general public key encryption algorithm, and was first published by Rivest, Shamir & Adleman of MIT in 1978 [RIVE78]. Since that time RSA has reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption. The RSA scheme is a block cipher in which the plaintext and the ciphertext are integers between 0 and n- 1 for some fixed n and typical size for n is 1024 bits (or 309 decimal digits). It is based on exponentiation in a finite (Galois) field over integers modulo a prime, using large integers (eg. 1024 bits). Its security is due to the cost of factoring

large numbers. RSA involves a public-key and a private-key where the public key is known to all and is used to encrypt data or message. The data or message which has been encrypted using a public key can only be decryted by using its corresponding private-key. Each user generates a key pair

public and private key using the following steps:

- *each user selects two large primes at random - p, q*
- *compute their system modulus n=p.q*
- *calculate ø(n), where* ***ø(n)=(p-1)(q-1)***
- *selecting at random the encryption key e, where 1<e<ø(n),and****gcd(e,ø(n))=1***
- *solve following equation to find decryption key d:* ***e.d=1 mod ø(n)*** *and0≤d≤n*
- *publish their public encryption key:* ***KU={e,n}***
- *keep secret private decryption key:****KR={d,n}***

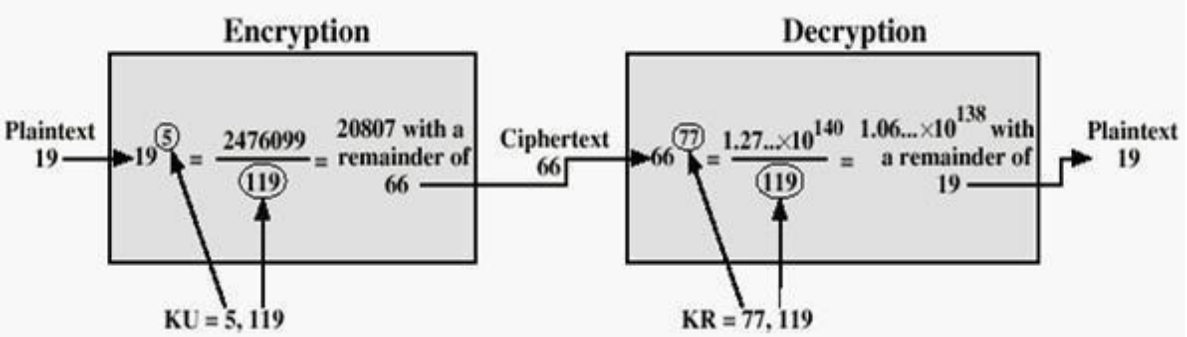
Both the sender and receiver must know the values of n and e, and only the receiver knows the value of d. Encryption and Decryption are done using the following equations. To encrypt a message M the sender:
– obtains **public key** of recipient *KU={e,n}*
– computes: **C=Me mod n**, where 0≤M<n To decrypt the ciphertext C the owner:
– uses their private key *KR={d,n}*
– computes: **M=Cd mod n = (Me) d mod n = Med mod n**
For this algorithm to be satisfactory, the following requirements are to be met.
a) Its possible to find values of e, d, n such that **Med = M mod n** for all M<n
b) It is relatively easy to calculate Me and C for all values of M < n.
c) It is impossible to determine d given e and n

The way RSA works is based on Number theory: **Fermat's little theorem**: if **p** is prime and **a** is positive integer not divisible by **p**, then **ap-1 ≡ 1 mod p. Corollary**: For any positive integer **a** and prime **p**, **ap ≡ a mod p.**

Fermat's theorem, as useful as will turn out to be does not provide us with integers d,e we are looking for –Euler's theorem (a refinement of Fermat's) does. Euler's function associates to any positive integer **n,** a number φ**(n)**: the number of positive integers smaller than **n** and relatively prime to **n.** For example, φ**(37) = 36**
i.e. φ**(p) = p-1** for any prime **p.** For any two primes **p,q**, φ**(pq)=(p-1)(q-1). Euler's theorem**: for any relatively prime integers a,n we have **a**φ**(n)≡1 mod n. Corollary: F**or any integers a,n we have **a**φ**(n)+1≡a mod n Corollary:** Let p,q be two odd primes and n=pq. Then: φ(n)=(p-1)(q-
1) For any integer m with 0<m<n, m(p-1)(q-1)+1 ≡ m mod n For any integers k,m with 0<m<n, mk(p-1)(q-1)+1 ≡ m mod n Euler's theorem provides us the numbers d, e such that Med=M mod n. We have to choose d,e such that ed=kφ(n)+1, or equivalently, d≡e- 1mod φ(n)

An example of RSA can be given as, Select primes: *p*=17 & *q*=11 Compute *n* = *pq*
=17×11=187
Compute ø(*n*)=(*p*–1)(*q*-1)=16×10=160 Select e *:* gcd(e,160)=1; choose *e*=7
Determine d*: de*=1 mod 160 and *d* < 160 Value is d=23 since 23×7=161= 10×160+1

Publish public key KU={7,187}
Keep secret private key KR={23,187} Now, given message M = 88 (nb. 88<187)
encryption: C = 887 mod 187 = 11
decryption: M = 1123 mod 187 = 88
Another example of RSA is given as,
Let **p = 11, q = 13, e = 11, m = 7**
n = pq i.e. n= 11*13 = 143
ø(n)= (p-1)(q-1) i.e. (11-1)(13-1) = 120
e.d=1 mod ø(n) i.e. 11d mod 120 = 1 i.e. (11*11) mod 120=1; so d = 11 public key :{11,143} and private key: {11,143}
C=Me mod n, so ciphertext = 711mod143 = 727833 mod 143; i.e. C = 106
M=Cd mod n, plaintext = 10611 mod 143 = 1008 mod 143; i.e. M = 7



For RSA key generation,

 users of RSA must:

– determine two primes at random - p, q
– select either e or d and compute the other

 primes p,q must not be easily derived from modulus N=p.q
– means must be sufficiently large
– typically guess and use probabilistic test

 exponents e, d are inverses, so use Inverse algorithm to compute the other

Security of RSA

There are three main approaches of attacking RSA algorithm.
Brute force key search (infeasible given size of numbers) As explained before, involves trying all possible private keys. Best defence is using large keys.
Mathematical attacks (based on difficulty of computing ø(N), by factoring modulus N) There are several approaches, all equivalent in effect to factoring the product of two primes. Some of them are given as:
– factor N=p.q, hence find ø(N) and then d
– determine ø(N) directly and find d
– find d directly

The possible defense would be using large keys and also choosing large numbers for p and q, which should differ only by a few bits and are also on the order of magnitude 10_{75} to 10_{100}. And gcd (p-1, q-1) should be small.

DIFFIE-HELLMAN KEY EXCHANGE

Diffie-Hellman key exchange (**D-H**) is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. The D-H algorithm depends for its effectiveness on the difficulty of computing discrete logarithms.

First, a primitive root of a prime number p, can be defined as one whose powers generate all the integers from 1 to p-1. If a is a primitive root of the prime number p, then the numbers, ***a mod p, a_2 mod p,..., a_{p-1} mod 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 $b \equiv a^i \pmod{p}$ where $0 \leq i \leq (p\ 1)$ .The exponent i is referred to as the discrete logarithm of b for the base a, mod p. We express this value as ***$dlog_{a,p}$ (b).*** The algorithm is summarized below:

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

User A Key Generation	
Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation	
Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A
$K = (Y_B)^{X_A} \bmod q$

Generation of Secret Key by User B
$K = (Y_A)^{X_B} \bmod q$

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 exchange a key. 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 $X_A < q$ 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. User A computes the key as $K = (Y_B)_{XA} \bmod q$ and user B computes the key as $K = (Y_A)_{XB} \bmod q$. These two calculations produce identical results.

Discrete Log Problem

The (discrete) exponentiation problem is as follows: Given a base a, an exponent b and a modulus p, calculate c such that $a_b \equiv c \pmod p$ and $0 \le c < p$. It turns out that this problem is fairly easy and can be calculated "quickly" using fast-exponentiation. The discrete log problem is the inverse problem: Given a base a, a result c ($0 \le c < p$) and a modulus p,

calculate the exponent b such that $a_b \equiv c \pmod p$. It turns out that no one has found a quick way to solve this problem With DLP, if P had 300 digits, X_a and X_b have more than 100 digits, it would take longer than the life of the universe to crack the method.

Examples for D-H key distribution scheme:

1) Let p = 37 and g = 13.

Let Alice pick a = 10. Alice calculates $13_{10} \pmod{37}$ which is 4 and sends that to Bob. Let Bob pick b = 7. Bob calculates $13_7 \pmod{37}$ which is 32 and sends that to Alice. (Note: 6 and 7 are secret to Alice and Bob, respectively, but both 4 and 32 are known by all.)

- Alice receives 32 and calculates $32_{10} \pmod{37}$ which is 30, the secret key.
- Bob receives 4 and calculates $4_7 \pmod{37}$ which is 30, the same secret key.

2) Let p = 47 and g = 5. Let Alice pick a = 18. Alice calculates $5_{18} \pmod{47}$ which is 2 and sends that to Bob. Let Bob pick b = 22. Bob calculates $5_{22} \pmod{47}$ which is 28 and sends that to Alice.

- Alice receives 28 and calculates $28_{18} \pmod{47}$ which is 24, the secret key.
- Bob receives 2 and calculates $2_{22} \pmod{47}$ which is 24, the same secret key

Man-in-the-Middle Attack on D-H protocol

Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows:

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates $K2 = (Y_A)_{XD2} \bmod q$.
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})_{XE} \bmod q$.
5. Bob transmits X_A to Alice.
6. Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)_{XD1} \bmod q$.
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})_{XA} \bmod q$.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key K1 and Alice and Darth share secret key K2. All future communication between Bob and Alice is compromised in the following way:

1. Alice sends an encrypted message M: E(K2, M).

2. Darth intercepts the encrypted message and decrypts it, to recover M.
3. Darth sends Bob E(K1, M) or E(K1, M'), where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.
The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates.

ELLIPTIC CURVE CRYPTOGRAPHY (ECC)

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. The use of elliptic curves in cryptography was suggested independently by Neal Koblitz and Victor S. Miller in 1985. The principal attraction of ECC compared to RSA is that it appears to offer equal security for a far smaller bit size, thereby reducing the processing overhead.

Elliptic Curve over GF(p)

Let GF(p) be a finite field, p > 3, and let a, b

- GF(p) are constant such that ***4a3 + 27b2 ≡ 0 (mod p).*** An elliptic curve, E(a,b)(GF(p)), is defined as the set of points (x,y) C GF(p) * GF(p) which satisfy the equation

y2 ≡ x3 + ax + b (mod p), together with a special point, O, *called the point at infinity*.
Let P and Q be two points on E(a,b)(GF(p)) and O is the point at infinity.

- P+O = O+P = P

- If P = (x1,y1) then -P = (x1 ,-y1) and P + (-P) = O.

- If P = (x1,y1) and Q = (x2,y2), and P and Q are not O.

then P +Q = (x3 ,y3) where
x3 = ƛ 2 - x1 - x2
y3 = ƛ (x1 - x3) - y1 and
ƛ = (y2-y1)/(x2-x1) if P ≠ Q
ƛ = (3x12+a)/ 2y1 if P = Q

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \; for \_ x_1 \neq x_2 \\ \dfrac{3x_1^{\,2} + a}{2y_1} \; for \_ x_1 = x_2 \end{cases}$$

86

An elliptic curve may be defined over any finite field GF(q). For GF(2_m), the curve has a different form:- ***y_2 + xy = x_3 + ax_2 + b,*** *where b !=0.*

Cryptography with Elliptic Curves

The addition operation in ECC is the counterpart of modular multiplication in RSA, and multiple addition is the counterpart of modular exponentiation. To form a cryptographic system using elliptic curves, some kind of hard problem such as discrete logarithm or factorization of prime numbers is needed. Considering the equation, Q=kP, where Q,P are points in an elliptic curve, it is “easy” to compute Q given k,P , but “hard” to find k given Q,P. This is known as the elliptic curve logarithm problem. k could be so large as to make brute- force fail.

ECC Key Exchange

Pick a prime number p= 2180 and elliptic curve parameters a and b for the equation ***y2 ≡ x3 + ax + b (mod p)*** which defines the elliptic group of points Ep(a,b). Select generator point G=(x1,y1) in Ep(a,b) such that the smallest value for which nG=O be a very large prime number. Ep(a,b) and G are parameters of the cryptosystem known to all participants. The following steps take place:

• A & B select private keys nA<n, nB<n

• compute public keys: PA=nA×G, PB=nB×G

• Compute shared key: K=nA×PB, K=nB×PA {same since K=nA×nB×G }

ECC Encryption/Decryption As with key exchange system, an encryption/decryption system requires a point G and and elliptic group Ep(a,b) as parameters. First thing to be done is to encode the plaintext message m to be sent as an x-y point **Pm**. Each user chooses private key nA<n and computes public key PA=nA×G. To encrypt and send a message to Pm to B, A chooses a random positive integer k and produces the ciphertext **Cm** consisting of the pair of points **Cm={kG, Pm+kPb}.** here, A uses B's public key. To

decrypt the ciphertext, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point **Pm+*k*Pb – nB(*kG*) = Pm+*k*(nB*G*) – nB(*kG*) = Pm** A has masked the message **Pm** by adding ***k*Pb** to it. Nobody but A knows the value of k, so even though **Pb** is a public key, nobody can remove the mask k**Pb.** For an attacker to recover the message, he has to compute k given G and kG, which is assumed hard.

Security of ECC To **protect** a 128 bit AES key it would take a RSA Key Size of 3072 bits whereas an ECC Key Size of 256 bits.

Computational Effort for Cryptanalysis of Elliptic Curve Cryptography Compared to RSA

Key Size	MIPS-Years
150	3.8×10^{10}
205	7.1×10^{18}
234	1.6×10^{28}

(a) Elliptic Curve Logarithms using the Pollard rho Method

Key Size	MIPS-Years
512	3×10^{4}
768	2×10^{8}
1024	3×10^{11}
1280	1×10^{14}
1536	3×10^{16}
2048	3×10^{20}

(b) Integer Factorization using the General Number Field Sieve

Hence for similar security ECC offers significant computational advantages.

Applications of ECC:

- Wireless communication devices
- Smart cards
- Web servers that need to handle many encryption sessions
- Any application where security is needed but lacks the power, storage and computational power that is necessary for our current

cryptosystems

KEY MANAGEMENT

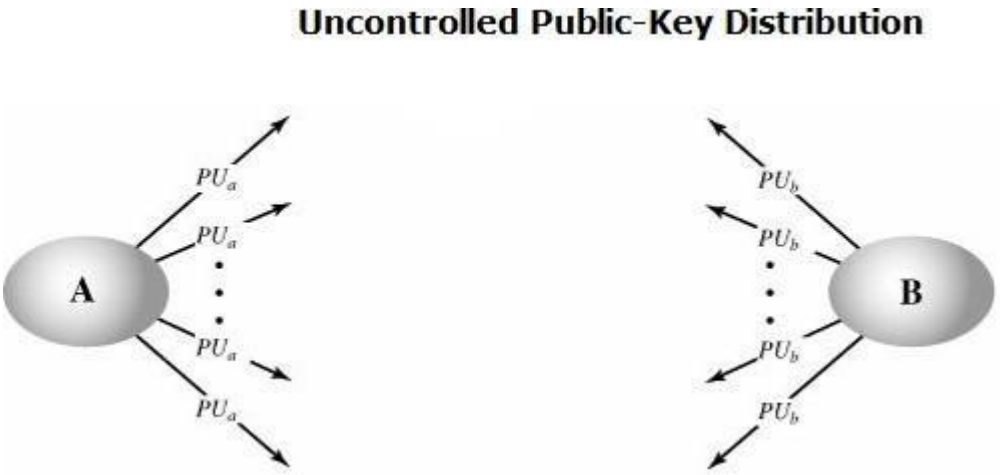
One of the major roles of public-key encryption has been to address the problem of key distribution. Two distinct aspects to use of public key encryption are present.

- The distribution of public keys.
- Use of public-key encryption to distribute secret keys.

Distribution of Public Keys The most general schemes for distribution of public keys are given below

PUBLIC ANNOUNCEMENT OF PUBLIC KEYS

Here any participant can send his or her public key to any other participant or broadcast the key to the community at large. For example, many PGP users have adopted the practice of appending their public key to messages that they send to public forums.

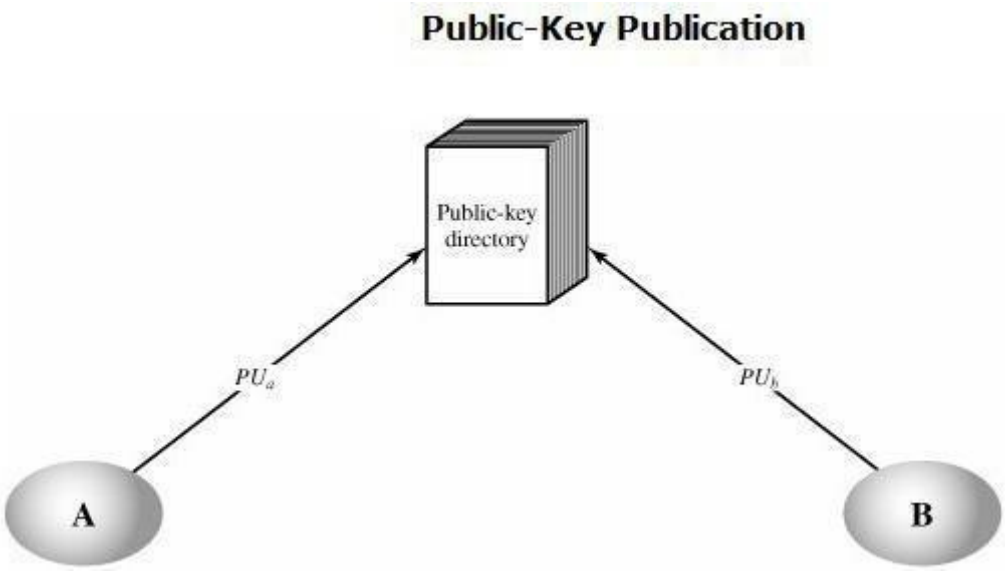


Though this approach seems convenient, it has a major drawback. Anyone can forge such a public announcement. Some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until the time when A discovers about the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

PUBLICLY AVAILABLE DIRECTORY

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization. It includes the following elements:

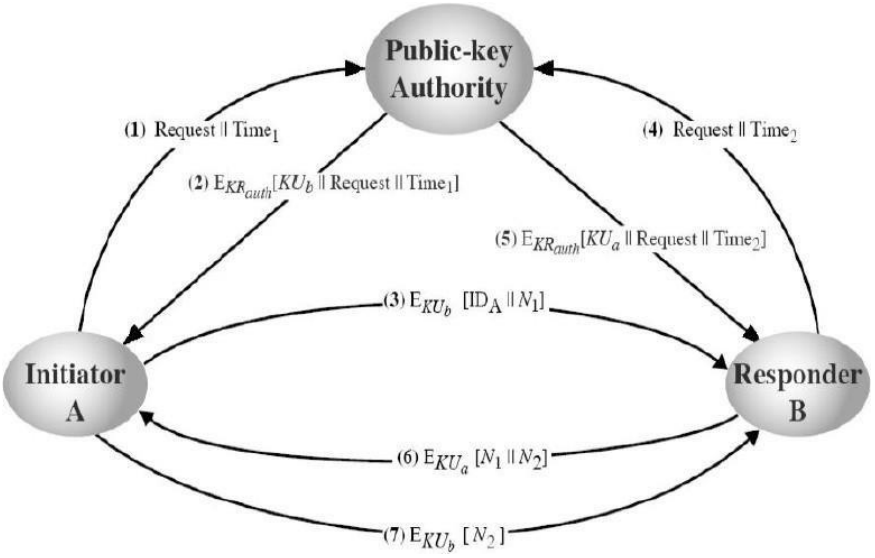
1. The authority maintains a directory with a {name, public key} entry for each participant.
2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.



3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way.
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory. This scheme has still got some vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Or else, the adversary may tamper with the records kept by the authority.

PUBLIC-KEY AUTHORITY

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. This scenario assumes the existence of a public authority (whoever that may be) that maintains a dynamic directory of public keys of all users. The public authority has its own (private key, public key) that it is using to communicate to users. Each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key. For example, consider that Alice and Bob wish to communicate with each other and the following steps take place and are also shown in the figure below:



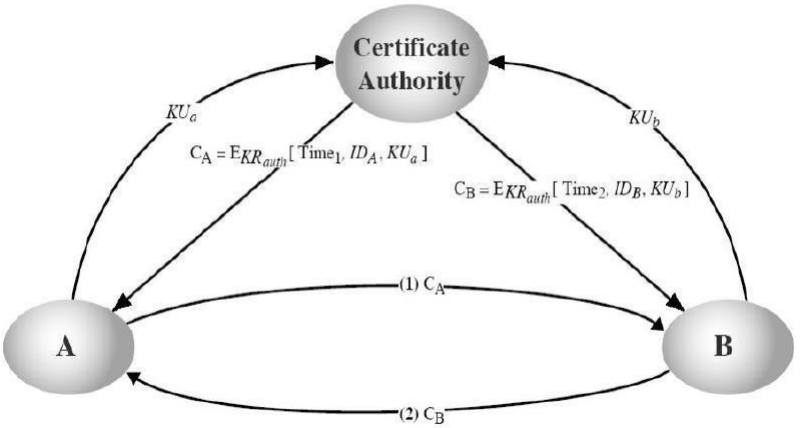
1.) Alice sends a **timestamped** message to the central authority with a request for Bob's public key (the time stamp is to mark the moment of the request)
2.) The authority sends back a message encrypted with its private key (for authentication) –message contains Bob's public key and the original message of Alice – this way Alice knows this is not a reply to an old request;
3.) Alice starts the communication to Bob by sending him an encrypted message containing her identity ID_A and a nonce N_1 (to identify uniquely this transaction)
4.) Bob requests Alice's public key in the same way (step 1)
5.) Bob acquires Alice's public key in the same way as Alice did. (Step-2)
6.) Bob replies to Alice by sending an encrypted message with N_1 plus a new generated nonce N_2 (to identify uniquely the transaction)
7.) Alice replies once more encrypting Bob's nonce N_2 to assure bob that its correspondent is Alice
Thus, a total of seven messages are required. However, the initial four messages need be used only infrequently because both A and B can save the other's public key for future use, a technique known as caching. Periodically, a user should request fresh copies of the public keys of its correspondents to ensure currency.

PUBLIC-KEY CERTIFICATES

The above technique looks attractive, but still has some drawbacks. For any communication between any two users, the central authority must be consulted by both users to get the newest public keys i.e. the central authority must be online 24 hours/day. If the central authority goes offline, all secure communications get to a halt. This clearly leads to an undesirable bottleneck. A further improvement is to use certificates, which can be used to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. A certificate binds an **identity** to **public key**, with all contents **signed** by a trusted Public-Key or Certificate Authority (CA). A user can present his or her public key to the authority in a secure manner, and obtain a certificate. The user can then publish the certificate. Anyone needed this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority. This certificate issuing scheme does have the following

requirements:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.
4. Any participant can verify the currency of the certificate.



Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form **$C_A = E(PR_{auth}, [T||ID_A||PU_a])$** where PR_{auth} is the private key used by the authority and T is a timestamp. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows: $D(PU_{auth}, C_A) = D(PU_{auth}, E(PR_{auth}, [T||ID_A||PU_a]))$

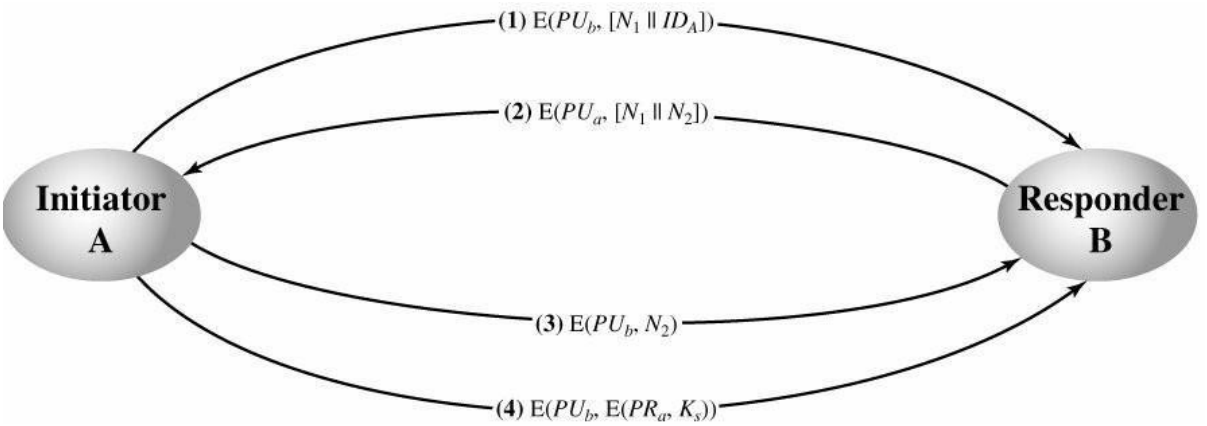
$= (T||ID_A||PU_a)$ The recipient uses the authority's public key, PU_{auth} to decrypt the certificate. Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority. The elements ID_A and PU_a provide the recipient with the name and public key of the certificate's holder. The timestamp T validates the currency of the certificate. The timestamp counters the following scenario. A's private key is learned by an adversary. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages. In this context, the compromise of a private key is comparable to the loss of a credit card. The owner cancels the credit card number but is at risk until all possible communicants are aware that the old credit card is obsolete. Thus, the timestamp serves as something like an expiration date. If a certificate is sufficiently old, it is assumed to be expired.

One scheme has become universally accepted for formatting public-key certificates: the

X.509 standard. X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME.

SECRET KEY DISTRIBUTION WITH CONFIDENTIALITY AND AUTHENTICATION

It is assumed that A and B have exchanged public keys by one of the schemes described earlier. Then the following steps occur:



1. A uses B's public key to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.
2. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2) Because only B could have decrypted message (1), the presence of N_1 in message (2) assures A that the correspondent is B.
3. A returns N_2 encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key K_s and sends M = E(PU_b, E(PR_a, K_s)) to B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.
5. B computes D(PU_a, D(PR_b, M)) to recover the secret key.

The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.