Unit 2

# COMPUTER-BASED SYMMETRIC-KEY CRYPTOGRAPHIC ALGORITHMS

MKK

# INTRODUCTION

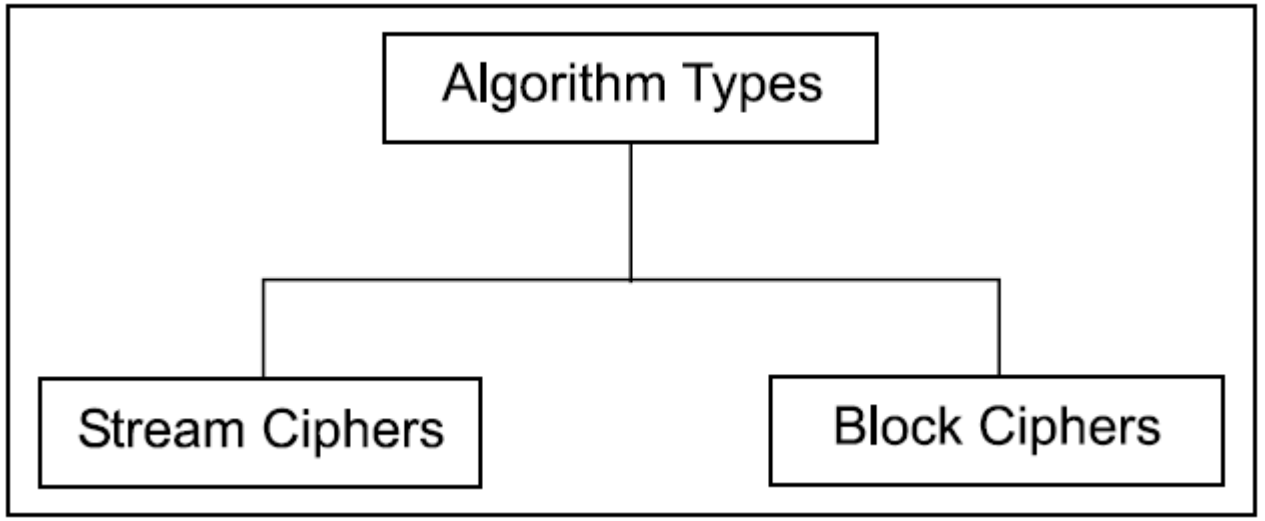
- Computers perform encryption quite easily and fast. Symmetric-key cryptography has been quite popular over a number of years. Of late, it is losing the edge to asymmetric key cryptography.

# ALGORITHM TYPES AND MODES

- Before we discuss real-life computer-based cryptography algorithms, let us discuss two key aspects of such algorithms: algorithm types and algorithm modes.
- An algorithm type defines what size of plain text should be encrypted in each step of the algorithm.
- The algorithm mode defines the details of the cryptographic algorithm, once the type is decided.

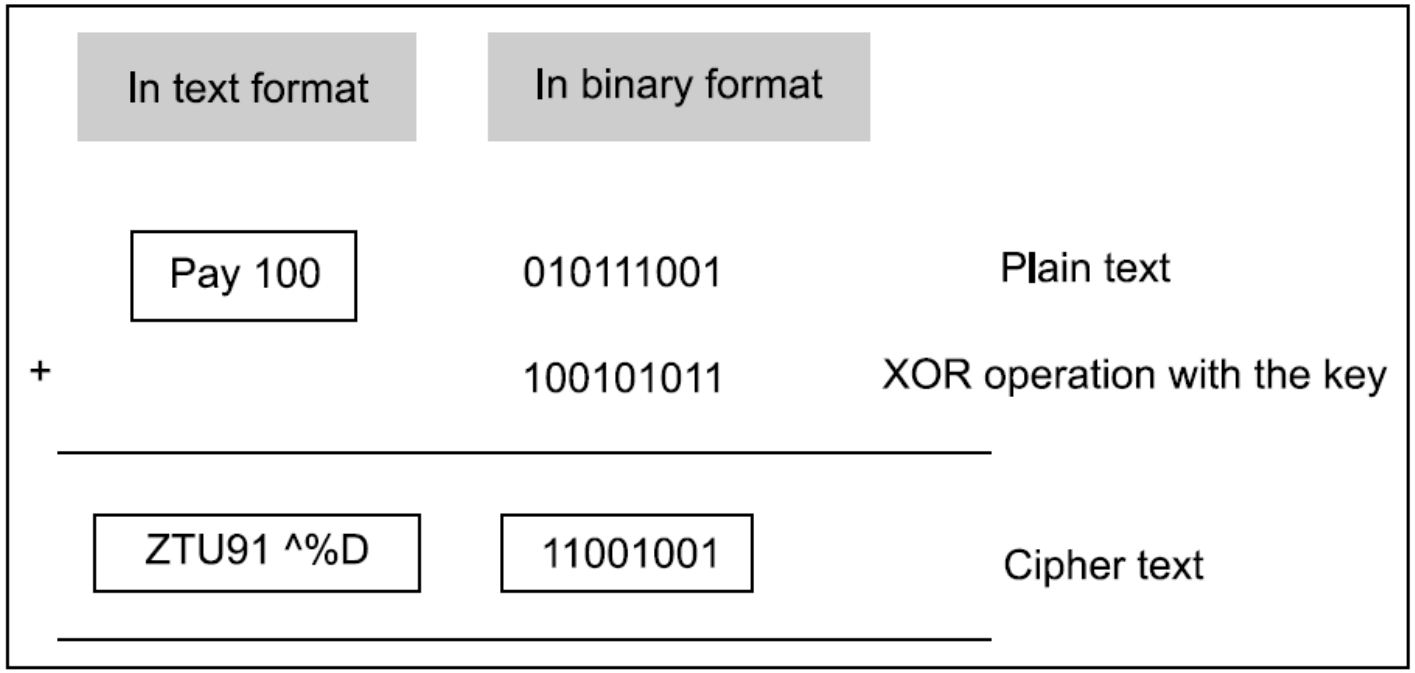
**Algorithm Types**

Regardless of the techniques used at a broad level, the generation of cipher text from plain text itself can be done in two basic ways, stream ciphers and block ciphers.



## *1. Stream Ciphers*

- In stream ciphers, the plain text is encrypted one bit at a time. Suppose the original message (plain text) is Pay 100 in ASCII (i.e. text format). When we convert these ASCII characters to their binary values, let us assume that it translates to 01011100 (In reality, the binary text would be much larger as each text character takes seven bits).
- Let us also assume that we apply the XOR logic as the encryption algorithm.



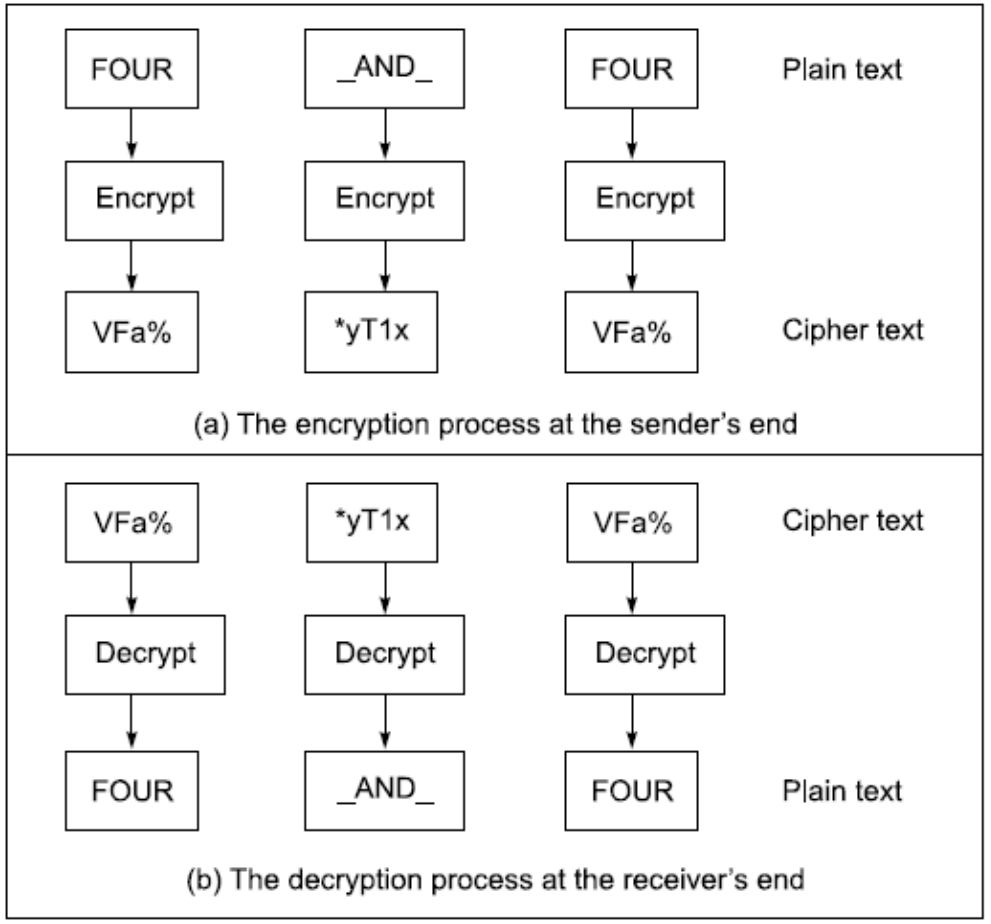

*XOR is reversible—when used twice, it produces the original values. This is useful in cryptography.*

- Stream-cipher technique involves the encryption of one plain-text bit at a time. The decryption also happens one bit at a time.

MKK

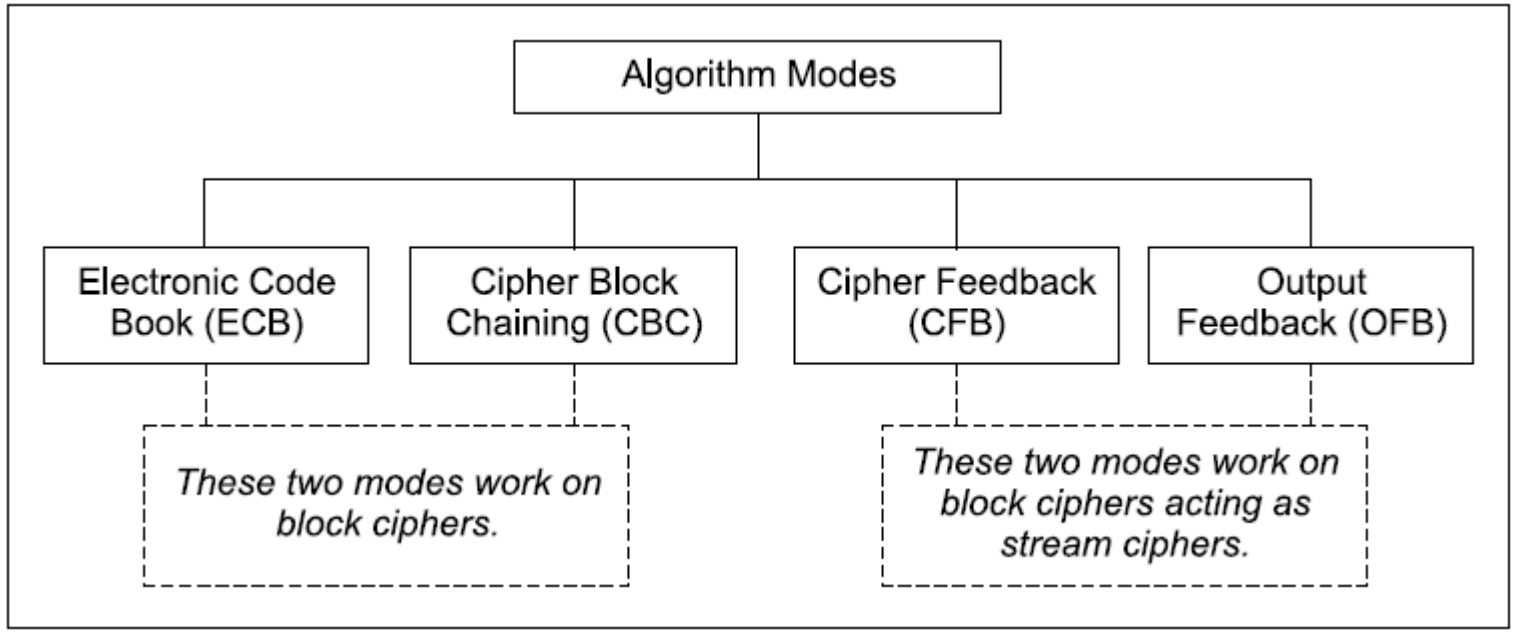
## *2. Block Ciphers*

- In block ciphers, rather than encrypting one bit at a time, a block of bits is encrypted at one go.
- Suppose we have a plain text FOUR_AND_FOUR that needs to be encrypted. Using block cipher, FOUR could be encrypted first, followed by _AND_ and finally FOUR.
- Thus, one block of characters gets encrypted at a time.
- The block-cipher technique involves encryption of one block of text at a time.
  Decryption also takes one block of encrypted text at a time.
- Practically, the blocks used in a block cipher generally contain 64 bits or more. That is why block ciphers are used more often in computer-based cryptographic algorithms as compared to stream ciphers.



## Algorithm Modes

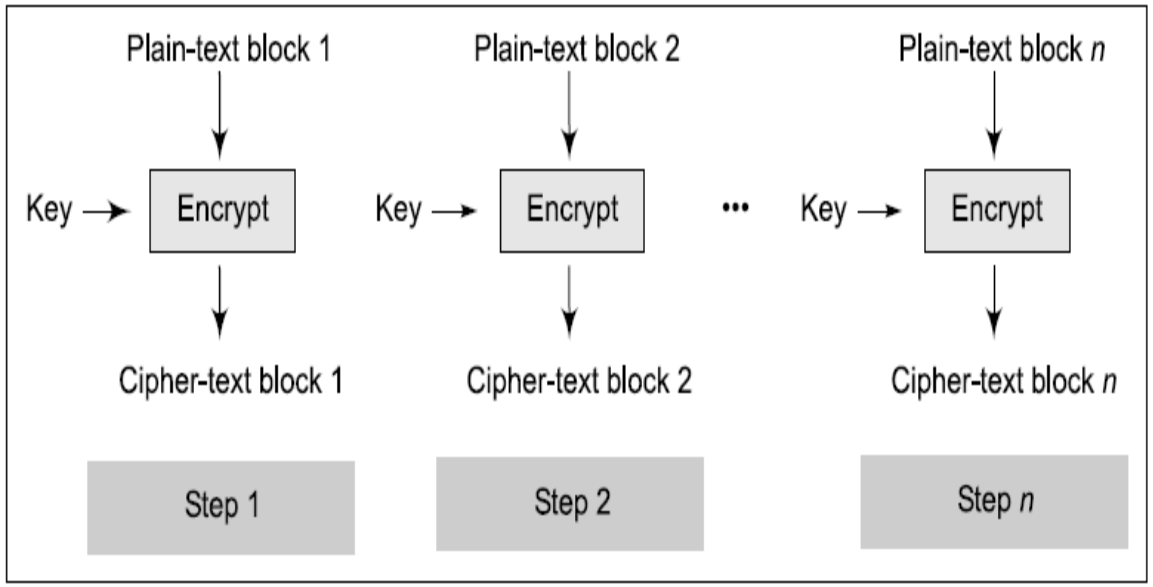
- An algorithm mode is a combination of a series of the basic algorithm steps on block cipher, and some kind of feedback from the previous step.
- We shall discuss it now, as it forms the basis for the computer based security algorithms. There are four important algorithm modes, namely
- *Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB).*

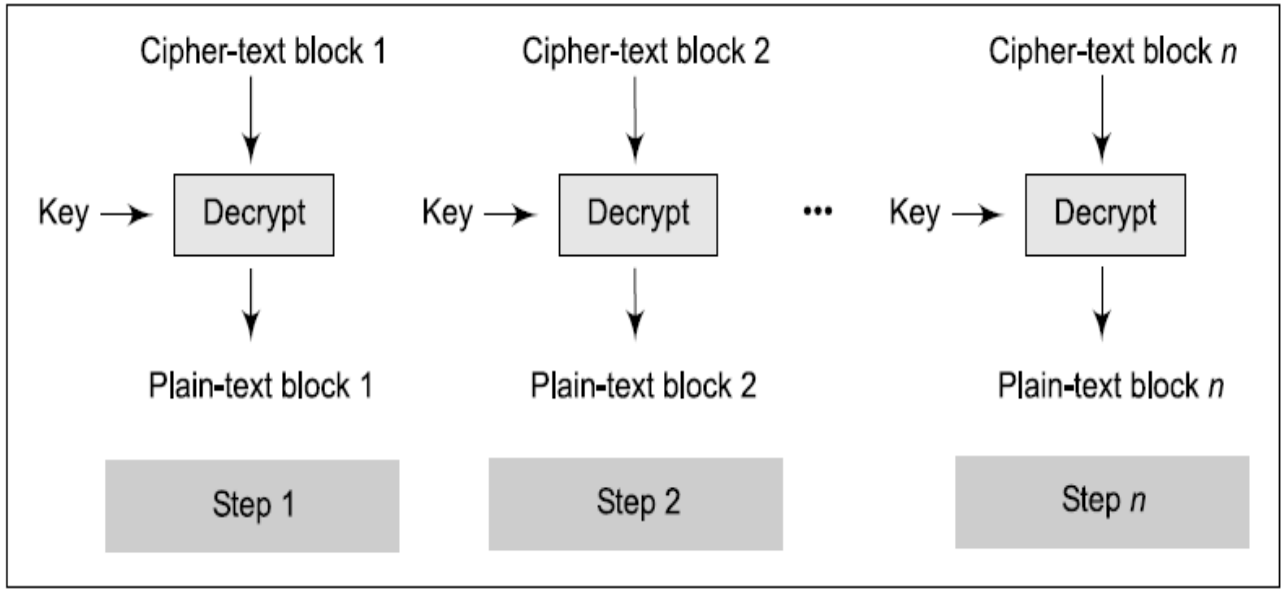



MKK

## *1. Electronic Code Book (ECB) Mode*

- Electronic Code Book (ECB) is the simplest mode of operation. Here, the incoming plain-text message is divided into blocks of 64 bits each. Each such block is then encrypted independently of the other blocks. For all blocks in a message, the same key is used for encryption.
- At the receiver's end, the incoming data is divided into 64-bit blocks, and by using the same key as was used for encryption, each block is decrypted to produce the corresponding plain-text block.



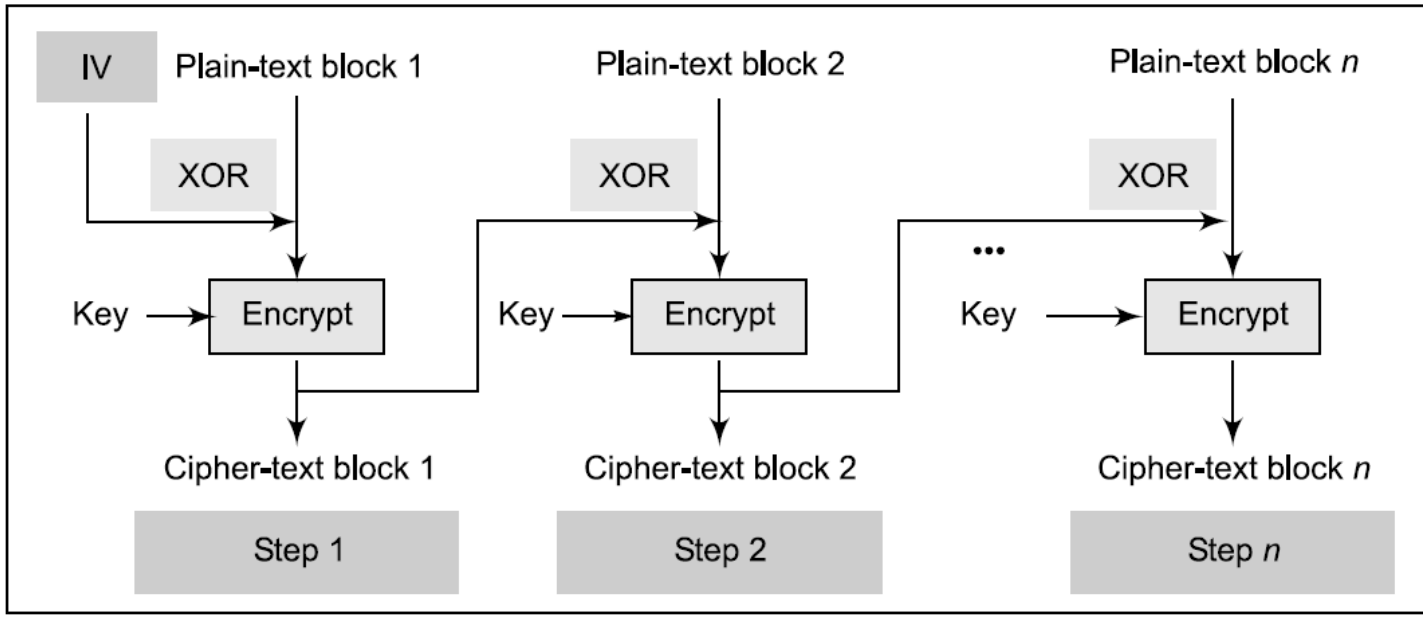




- ECB is suitable only for encrypting small messages, where the scope for repeating the same plain-text blocks is quite less.

MKK

## *2. Cipher Block Chaining (CBC) Mode*

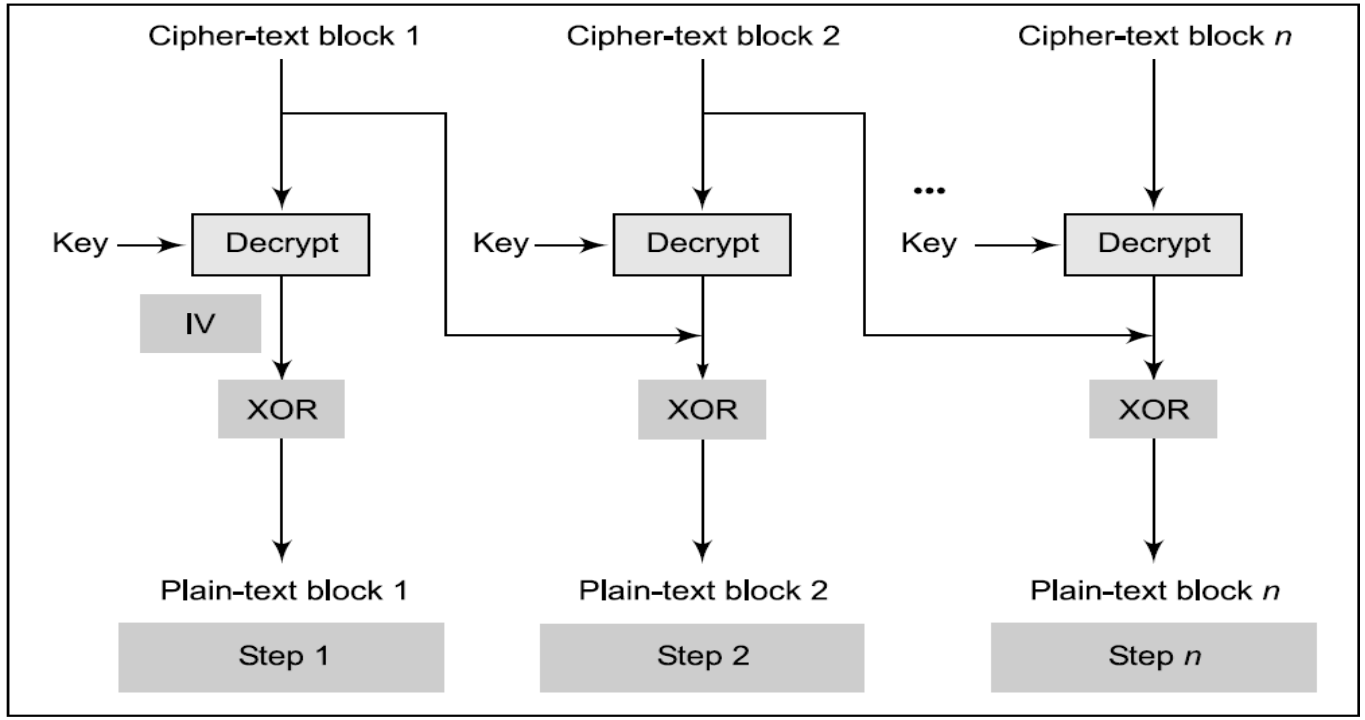
- We saw that in the case of ECB, within a given message (i.e. for a given key), a plain-text block always produces the same cipher-text block. Thus, if a block of plain text occurs more than once in the input, the corresponding cipher text block will also occur more than once in the output, thus providing some clues to a cryptanalyst.
- To overcome this problem, the Cipher Block Chaining (CBC) mode ensures that even if a block of plain text repeats in the input, these two (or more) identical plain-text blocks yield totally different cipher-text blocks in the output. For this, a feedback mechanism is used
- Chaining adds a feedback mechanism to a block cipher. In Cipher Block Chaining (CBC), the results of the encryption of the previous block are fed back into the encryption of the current block. That is, each block is used to modify the encryption of the next block. Thus, each block of cipher text is dependent on the corresponding current input plain-text block, as well as all the previous plain-text blocks.




1. As shown in the figure, the first step receives two inputs: the first block of plain text and a random block of text, called Initialization Vector (IV).
2. In the second step, the second plain-text block is XORed with the output of step 1, i.e. the first cipher-text block. It is then encrypted with the same key, as used in step 1. This produces ciphertext block 2.
3. In the third step, the third plain-text block is XORed with the output of step 2, i.e. the second cipher-text block. It is then encrypted with the same key, as used in step 1.
4. This process continues for all the remaining plain-text blocks of the original message.

Remember that the IV is used only in the first plain text block. However, the same key is used for the encryption of all plain text blocks.

MKK




The decryption process works as follows.

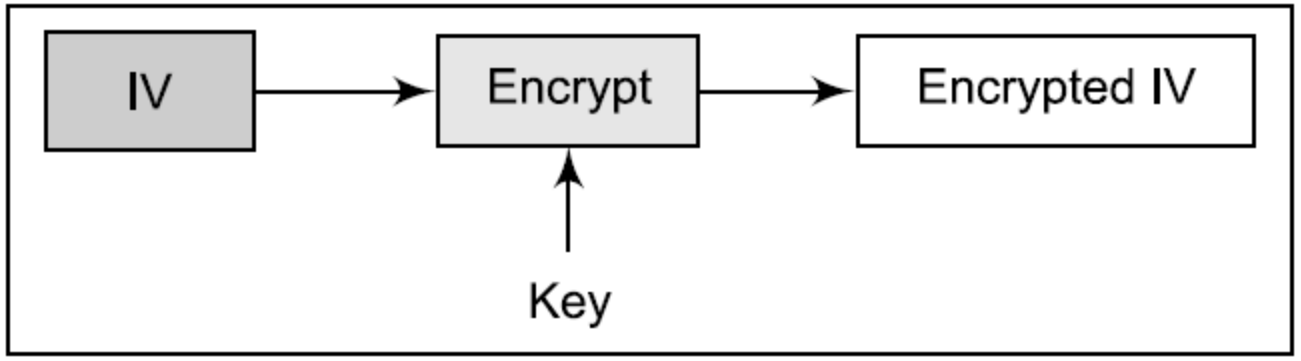
1. The cipher-text block 1 is passed through the decryption algorithm using the same key, which was used during the encryption process for all the plain-text blocks. The output of this step is then XORed with the IV. This process yields plain-text block 1.
2. In step 2, the cipher-text block 2 is decrypted, and its output is XORed with cipher-text block 1, which yields plain-text block 2.
3. This process continues for all the Cipher text blocks in the encrypted message.

MKK

### *3. Cipher Feedback (CFB) Mode*

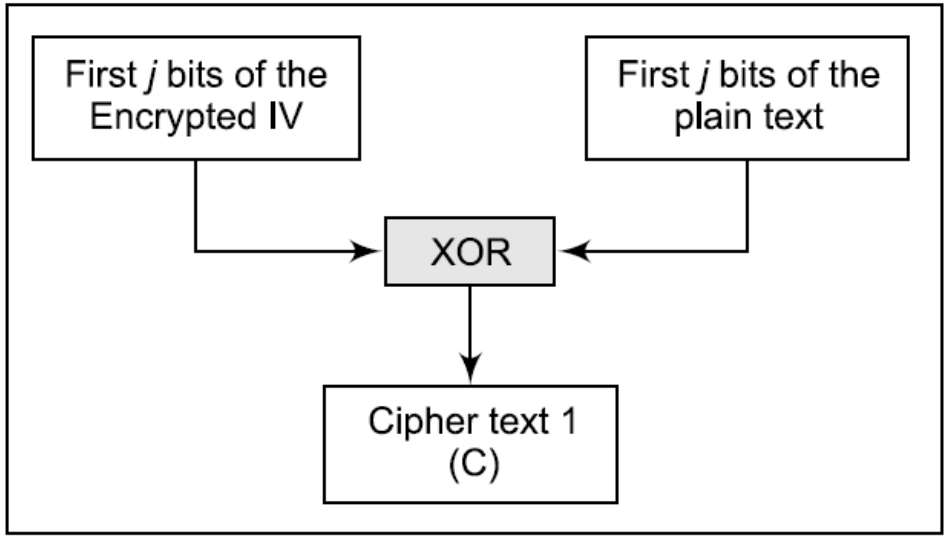
- Security is also required in applications that are character-oriented. For instance, an operator can be typing keystrokes at a terminal, which needs to be immediately transmitted across the communications link in a secure manner, i.e. by using encryption.
- In such situations, stream cipher must be used. The Cipher Feedback (CFB) mode is useful in such cases. In this mode, data is encrypted in units that are smaller (e.g. they could be of size 8 bits, i.e. the size of a character typed by an operator) than a defined block size (which is usually 64 bits).
- Let us understand how CFB mode works, assuming that we are dealing with ***j bits*** at a time

**Step 1** Like CBC, a 64-bit Initialization Vector (IV) is used in the case of CFB mode. The IV is kept in a shift register. It is encrypted in the first step to produce a corresponding 64-bit IV cipher text.

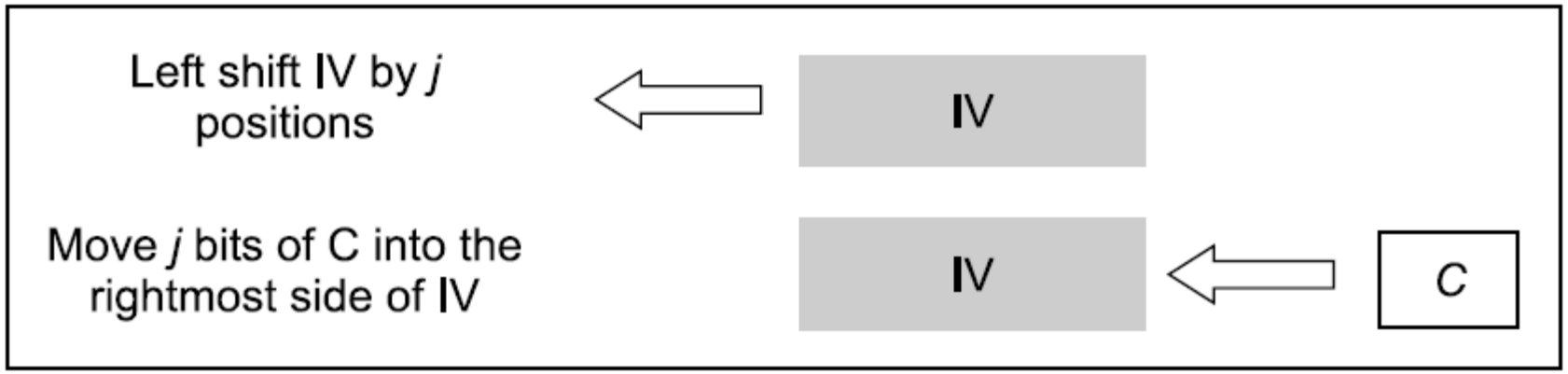



MKK

**Step 2** Now, the leftmost (i.e. the most significant) j bits of the encrypted IV are XORed with the first j bits of the plain text. This produces the first portion of cipher text (say C)




**Step 3** Now, the bits of IV (i.e. the contents of the shift register containing IV) are shifted left by j positions. Thus, the rightmost j positions of the shift register now contain unpredictable data. These rightmost j positions are now filled with C.




MKK

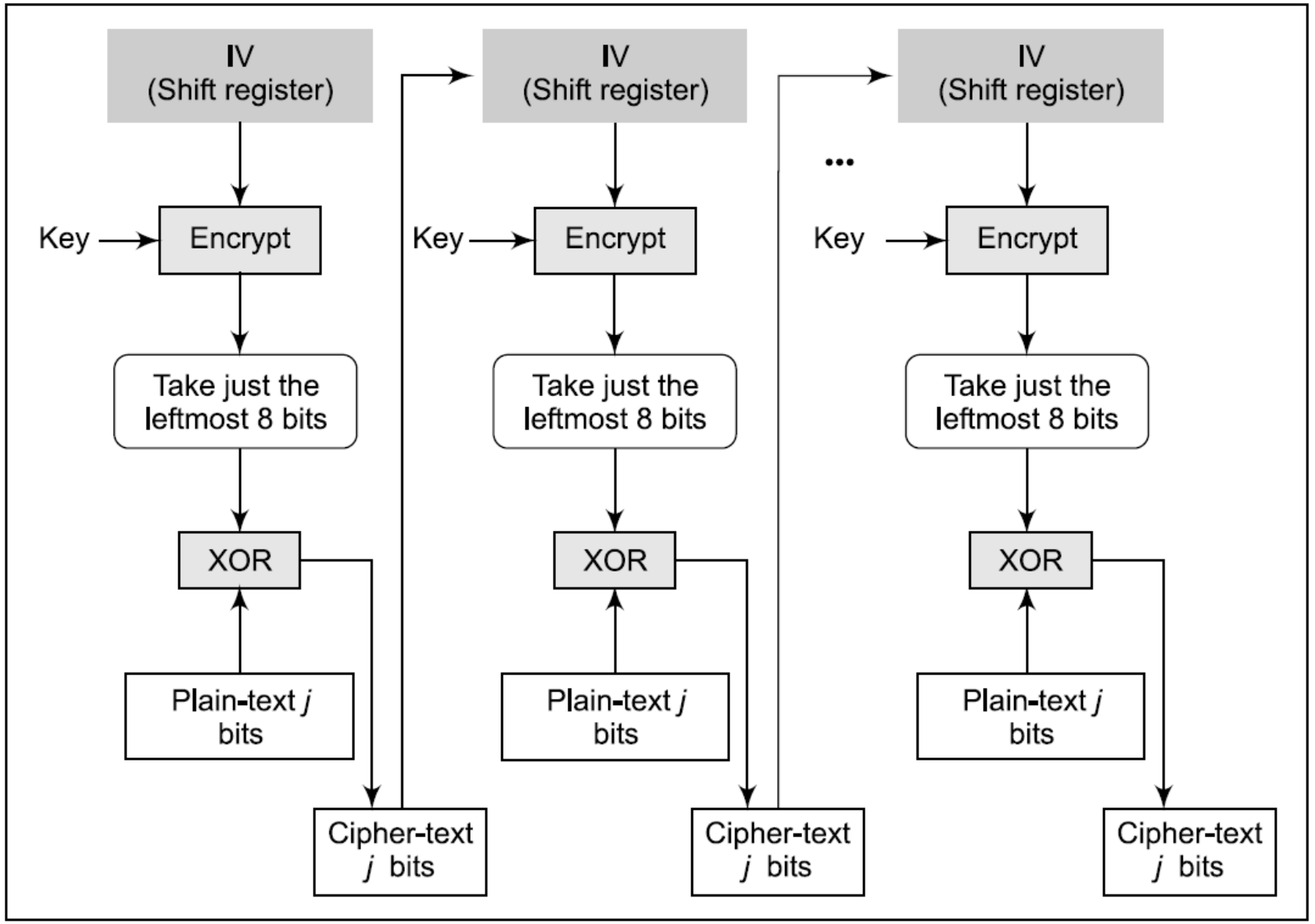
**Step 4** Now, steps 1 through 3 continue until all the plain-text units are encrypted. That is, the following steps are repeated:

- IV is encrypted.
- The leftmost j bits resulting from this encryption process are XORed with the next j bits of the plain text.
- The resulting cipher-text portion (i.e. the next j bits of cipher text) is sent to the receiver.
- The shift register containing the IV is left-shifted by j bits.
- The j bits of the cipher text are inserted from right into the shift register containing the IV.

- At the receiver's end, the decryption process is pretty similar, with minor changes.

MKK

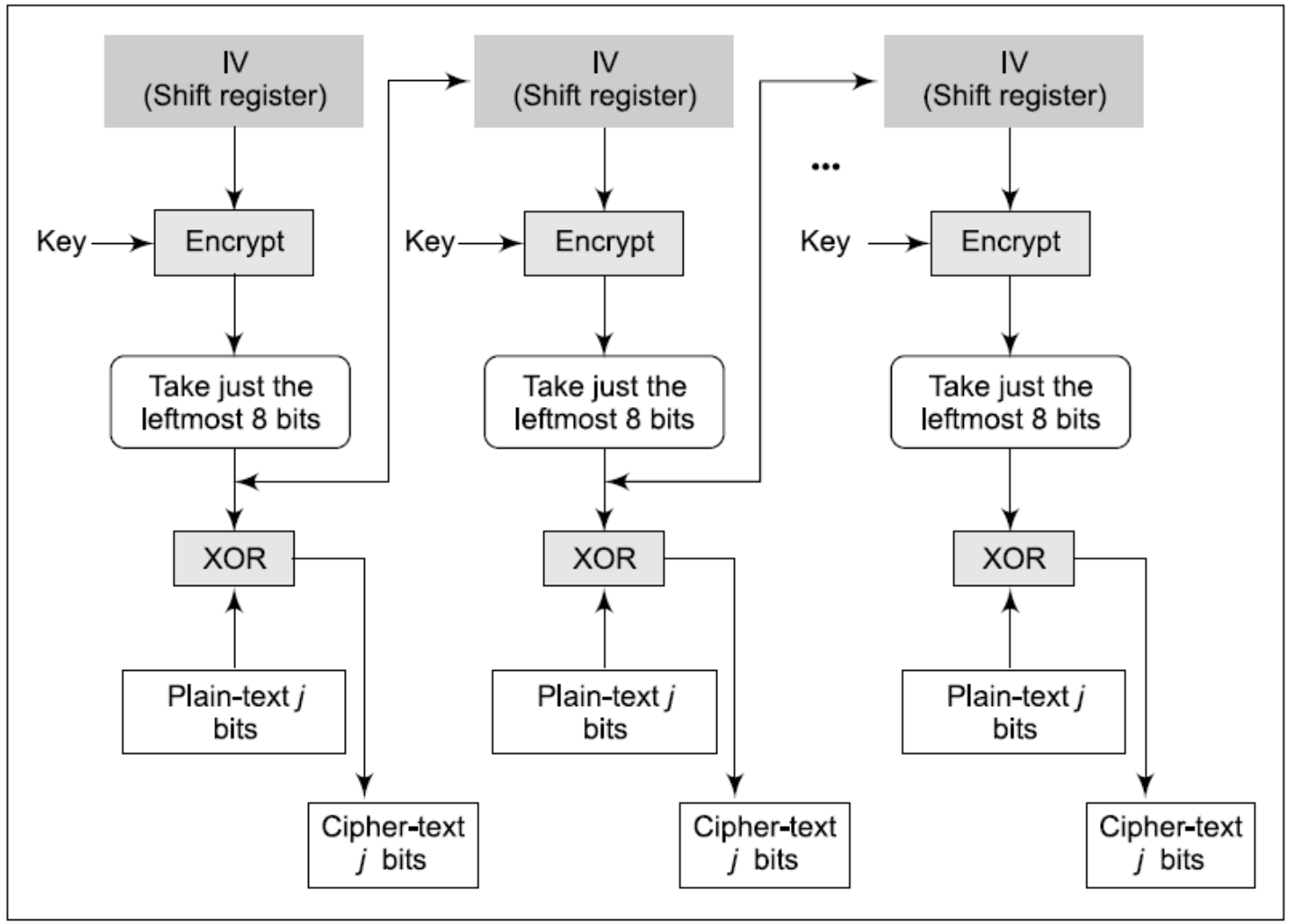
# *overall conceptual view of the CFB mode*




MKK

## *4. Output Feedback (OFB) Mode*

- The Output Feedback (OFB) mode is extremely similar to the CFB. The only difference is that in the case of CFB, the cipher text is fed into the next stage of encryption process. But in the case of OFB, the output of the IV encryption process is fed into the next stage of encryption process.



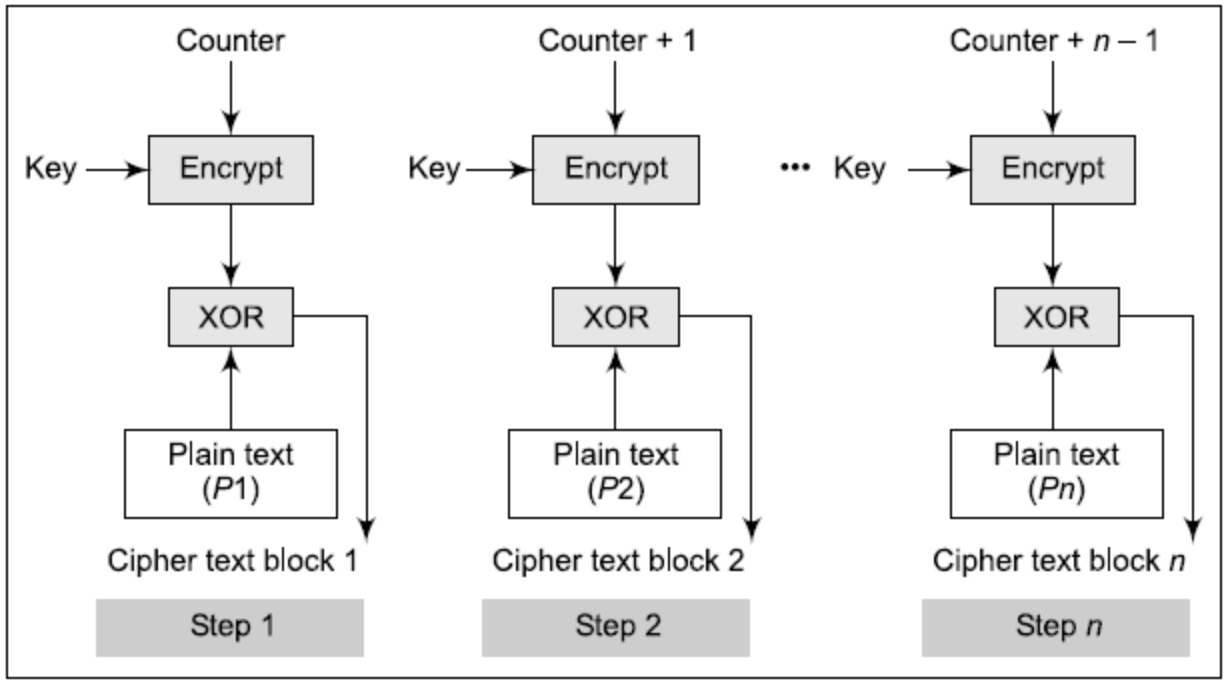

MKK

- The same details as discussed in CFB apply here, except the change, as pointed out above.
- The key advantage of the OFB mode. In simple terms, we can state that in this mode, if there are errors in individual bits, they remain errors in individual bits and do not corrupt the whole
- message. That is, bit errors do not get propagated. If a cipher-text bit $C_i$ is in error, only the decrypted value corresponding to this bit, i.e. $P_i$ is wrong. Other bits are not affected.
- Remember that in contrast to this, in the CFB mode, the cipher text bit $C_i$ is fed back as input to the shift register, and would corrupt the other bits in the message!
- The possible drawback with OFB is that an attacker can make necessary changes to the cipher text and the checksum of the message in a controlled fashion.

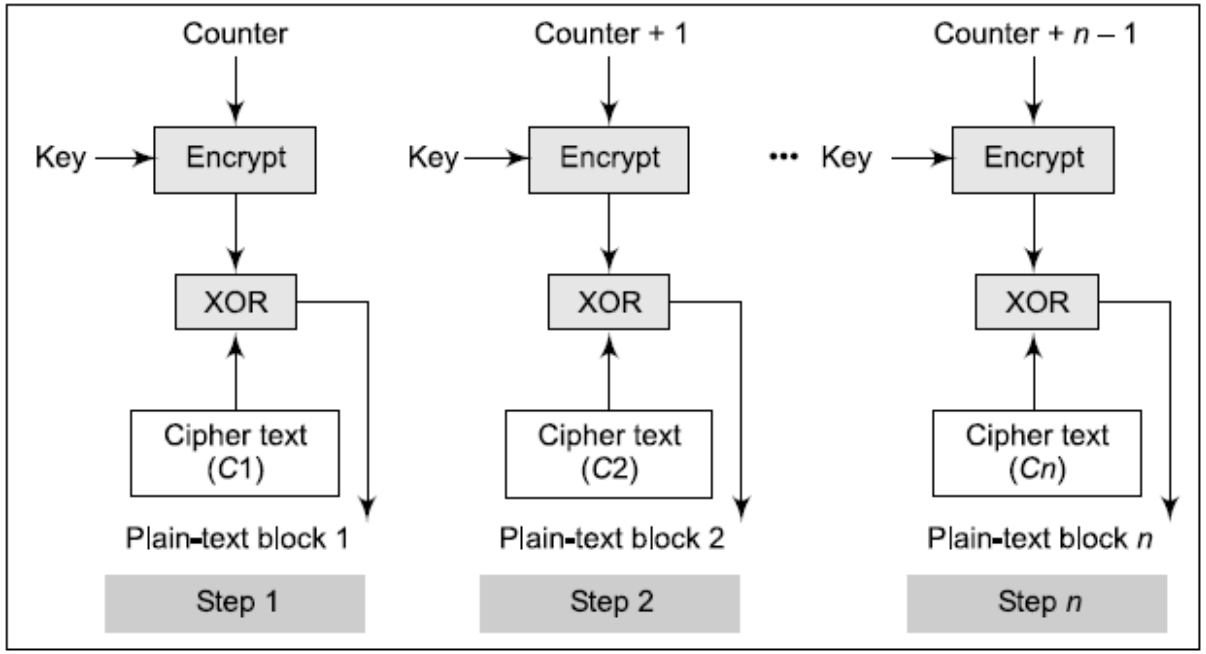
MKK

## *5. Counter (CTR) Mode*

- The Counter (CTR) mode is quite similar to the OFB mode, with one variation. It uses sequence numbers called counters as the inputs to the algorithm. After each block is encrypted, to fill the register, the next counter value is used.
- Usually, a constant is used as the initial counter value, and is incremented (usually by 1) for every iteration. The size of the counter block is the same as that of the plain-text block.




*Counter (CTR) mode: the encryption process*




*Counter (CTR) mode: the decryption process*

MKK

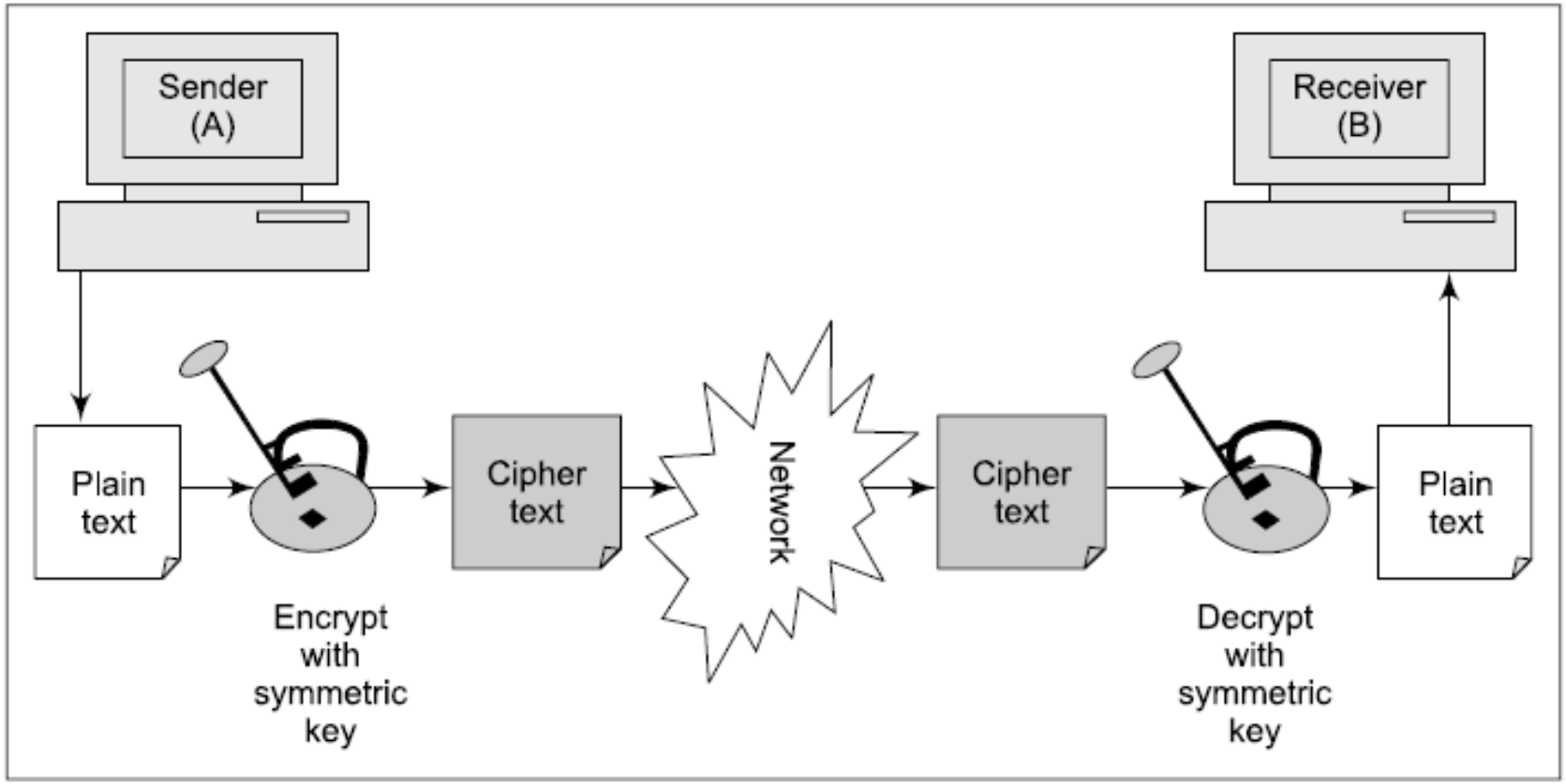
Algorithm modes: details and usage

| Algorithm mode | Details | Usage |
|---|---|---|
| ***Electronic Code Book (ECB)*** | The same key independently encrypts blocks of text, 64 bits at a time. | Transmitting a single value in a secure fashion (e.g. password or key used for encryption). |
| ***Cipher Block Chaining (CBC)*** | 64 bits of cipher text from the previous step and 64 bits of plain text of the next step are XORed together. | Encrypting blocks of text Authentication. |
| ***Cipher Feedback (CFB)*** | *K* bits of randomized cipher text from the previous step and *K* bits of plain text of the next step are XORed together. | Transmitting encrypted stream of data Authentication. |
| ***Output Feedback (OFB)*** | Similar to CFB, except that the input to the encryption step is the preceding DES output. | Transmitting encrypted stream of data. |
| ***Counter (CTR)*** | A counter and plain-text block are encrypted together, after which the counter is incremented. | Block-oriented transmissions Applications needing high speed. |

MKK

## AN OVERVIEW OF SYMMETRIC-KEY CRYPTOGRAPHY

- Symmetric-key cryptography is referred to by various other terms, such as secret-key cryptography or private-key cryptography. In this scheme, only one key is used and the same key is used for both encryption and decryption of messages.




MKK

symmetric-key cryptography has a few problems.

- The first problem here is that of key agreement or key distribution. The problem is: in the first place, how do two parties agree on a key? One solution is that somebody from the sender's end physically visits the receiver and hands over the key. Another way is to courier a paper on which the key is written. Both are not exactly very convenient. A third way is to send the key over the network to B and ask for confirmation. But then, if an intruder gets the message, he/she can interpret all the subsequent ones!
- The second problem is more serious. Since the same key is used for encryption and decryption, one key per communicating parties is required. Suppose A wants to securely communicate with B and also with C. Clearly, there must be one key for all communications between A and B; and there must be another, distinct key for all communications between A and C.
- Regardless, because these drawbacks can be overcome using intelligent solutions as we shall see, and also because symmetric-key cryptography has several advantages as well, it is widely used in practice.

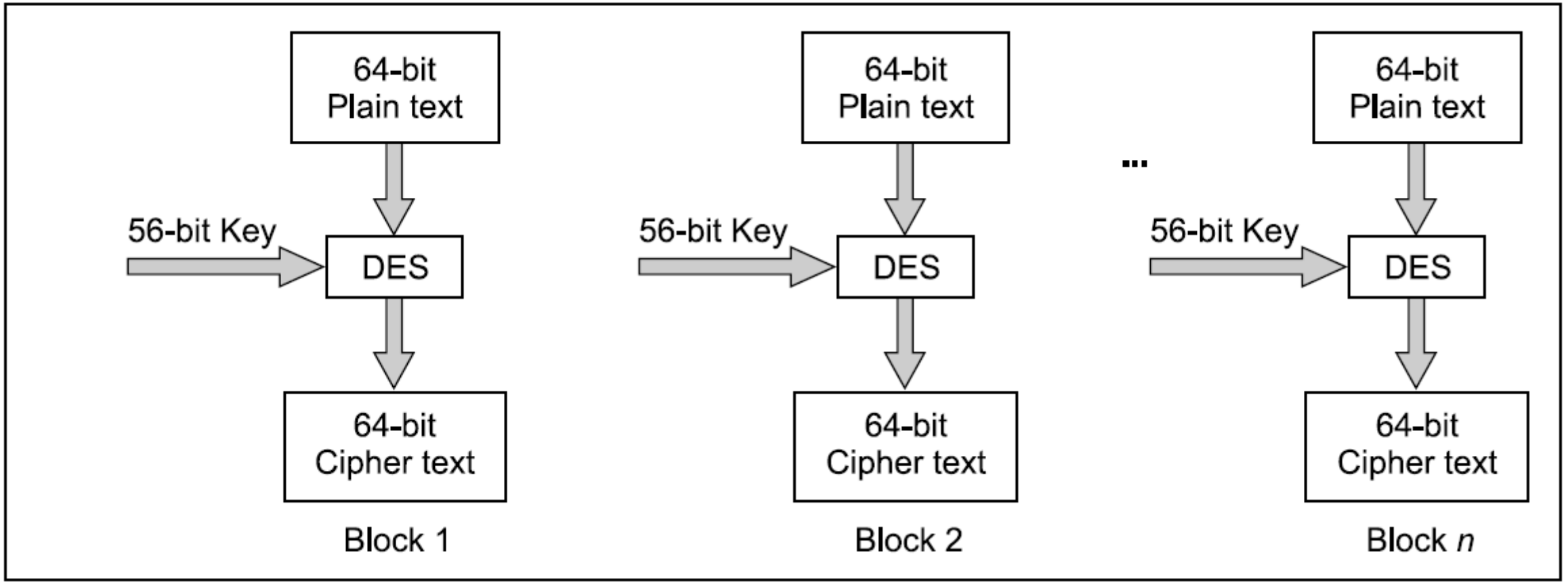
## DATA ENCRYPTION STANDARD (DES)

The Data Encryption Standard (DES), also called the Data Encryption Algorithm (DEA)

- DES is generally used in the ECB, CBC or the CFB mode.
- The origins of DES go back to 1972, when in the US, the National Bureau of Standards (NBS), now known as the National Institute of Standards and Technology (NIST), embarked upon a project for protecting the data in computers and computer communications.
- At the end of 1976, the US Federal Government decided to adopt this algorithm, and soon, it was renamed Data Encryption Standard (DES). Soon, other bodies also recognized and adopted DES as a cryptographic algorithm.

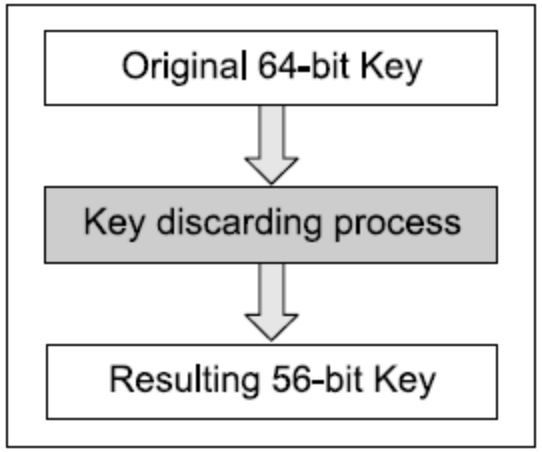
## *1. Basic Principles*

- DES is a block cipher. It encrypts data in blocks of 64 bits each. That is, 64 bits of plain text goes as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.



- We have mentioned that DES uses a 56-bit key. Actually, the initial key consists of 64 bits.
- However, before the DES process even starts, every eighth bit of the key is discarded to produce a 56-bit key. That is, bit positions 8, 16, 24, 32, 40, 48, 56 and 64 are discarded.

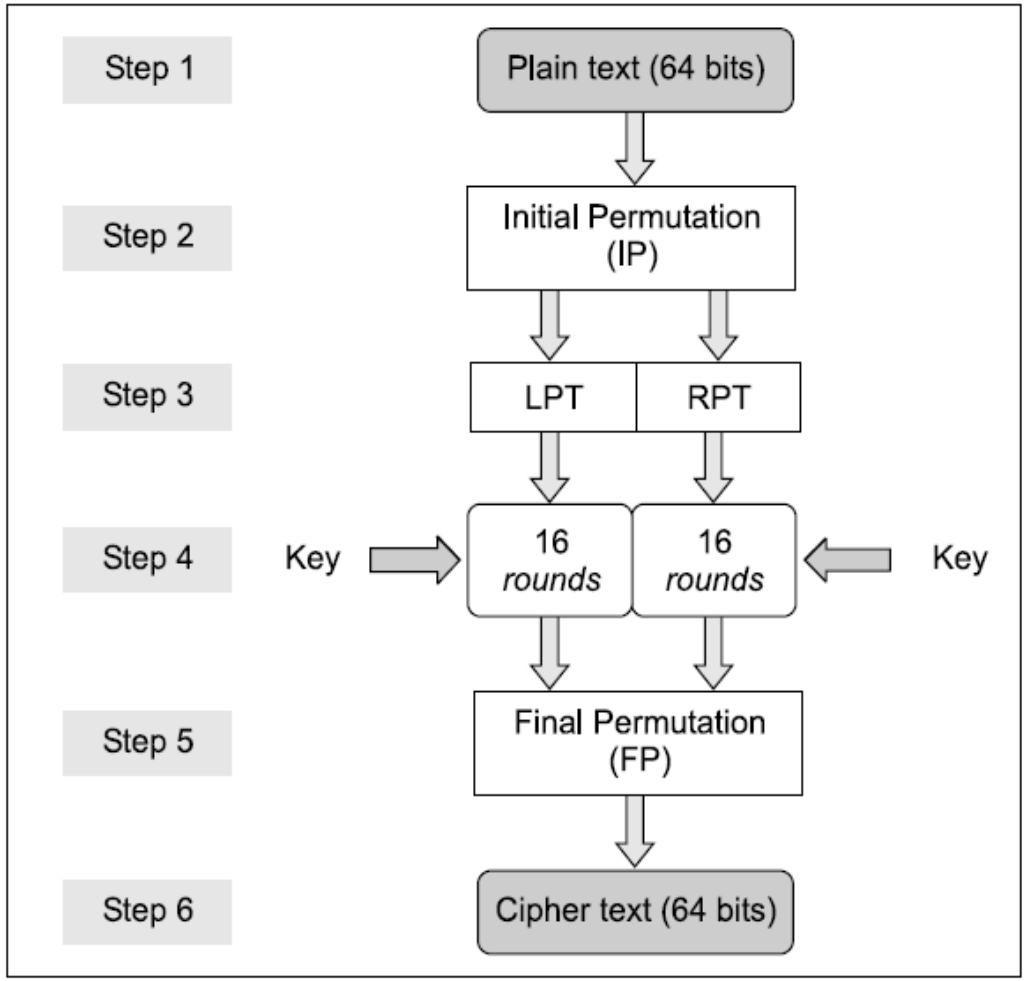
| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 |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




- DES consists of 16 steps, each of which is called a round.
- Each round performs the steps of substitution and transposition.
- Let us now discuss the broad-level steps in DES.

MKK

1. In the first step, the 64-bit plain-text block is handed over to an Initial Permutation (IP) function.
2. The initial permutation is performed on plain text.
3. Next, the Initial Permutation (IP) produces two halves of the permuted block; say Left Plain Text (LPT) and Right Plain Text (RPT).
4. Now, each of LPT and RPT go through 16 rounds of encryption process, each with its own key.
5. In the end, LPT and RPT are rejoined, and a Final Permutation (FP) is performed on the combined block.
6. The result of this process produces 64-bit cipher text.




***Broad level steps in DES***

MKK

## *2. Initial Permutation (IP)*

- The Initial Permutation (IP) happens only once, and it happens before the first round.
- It suggests how the transposition in IP should proceed, as shown in Fig.

| Bit position in the plain-text block | To be overwritten with the contents of this bit position |
|---|---|
| 1 | 58 |
| 2 | 50 |
| 3 | 42 |
| .. | .. |
| 64 | 7 |

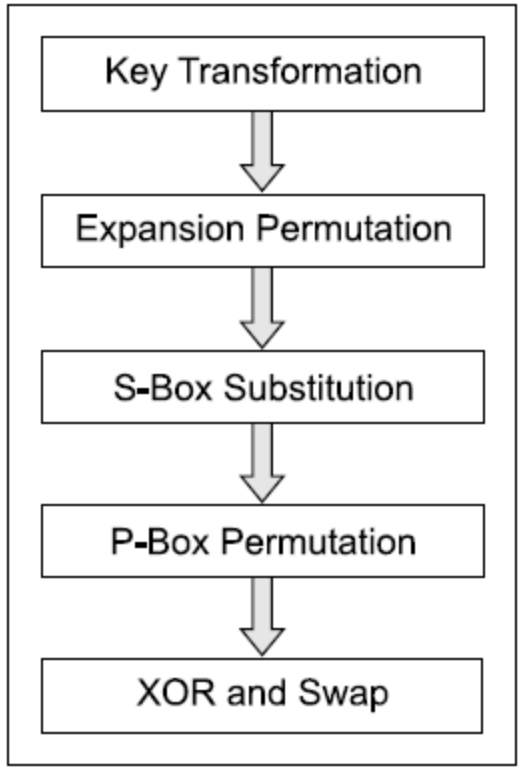
- For example, it says that the IP replaces the first bit of the original plain-text block with the 58th bit of the original plain-text block, the second bit with the 50th bit of the original plain text block, and so on. This is nothing but jugglery of bit positions of the original plain-text block.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

MKK

As we have noted, after IP is done, the resulting 64-bit permuted text block is divided into two half blocks. Each half block consists of 32 bits. We have called the left block as LPT and the right block as RPT. Now, 16 rounds are performed on these two blocks.

***3. Rounds***




Let us discuss these details step-by-step.

MKK

**Step 1: Key Transformation**

- We have noted that the initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each round, a 56-bit key is available.
- From this 56-bit key, a different 48-bit subkey is generated during each round using a process called key transformation. For this, the 56-bit key is divided into two halves, each of 28 bits.
- These halves are circularly shifted left by one or two positions, depending on the round.
- For example, if the round number is 1, 2, 9 or 16, the shift is done by only one position. For other rounds, the circular shift is done by two positions.

| Round | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of key bits shifted | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

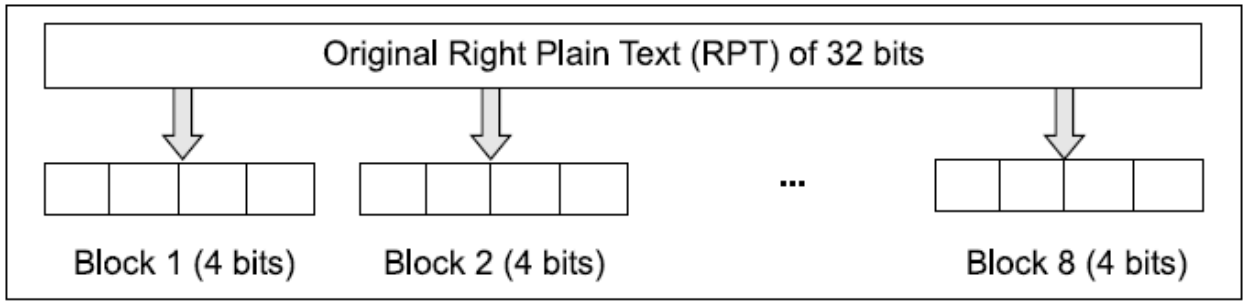
- After an appropriate shift, 48 of the 56 bits are selected.
- Since the key-transformation process involves permutation as well as selection of a 48-bit subset of the original 56-bit key, it is called compression permutation.
- Because of this compression permutation technique, a different subset of key bits is used in each round.
- That makes DES more difficult to crack.

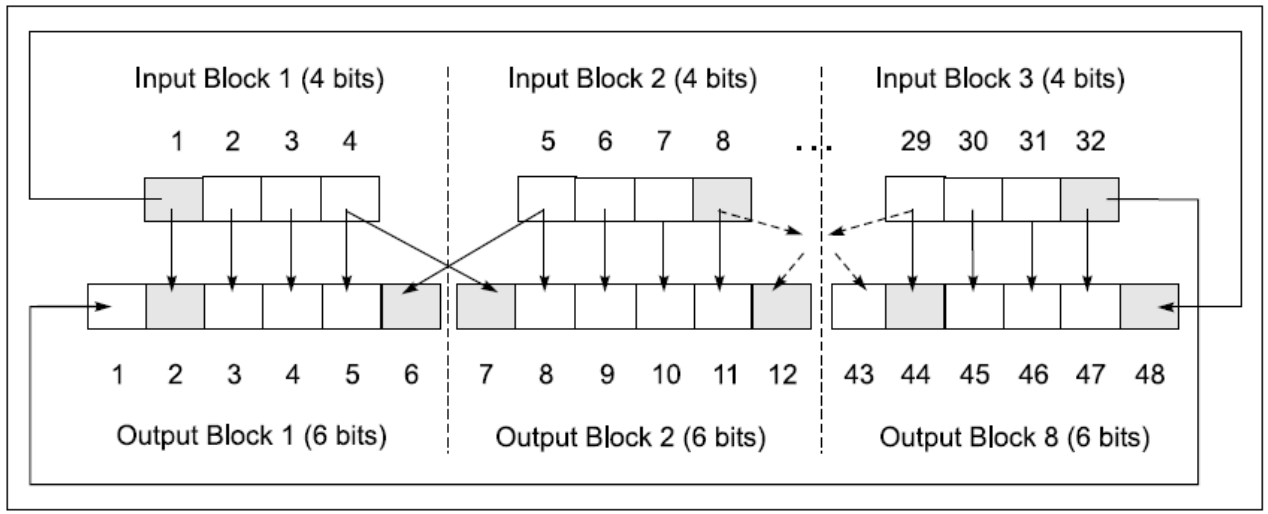
MKK

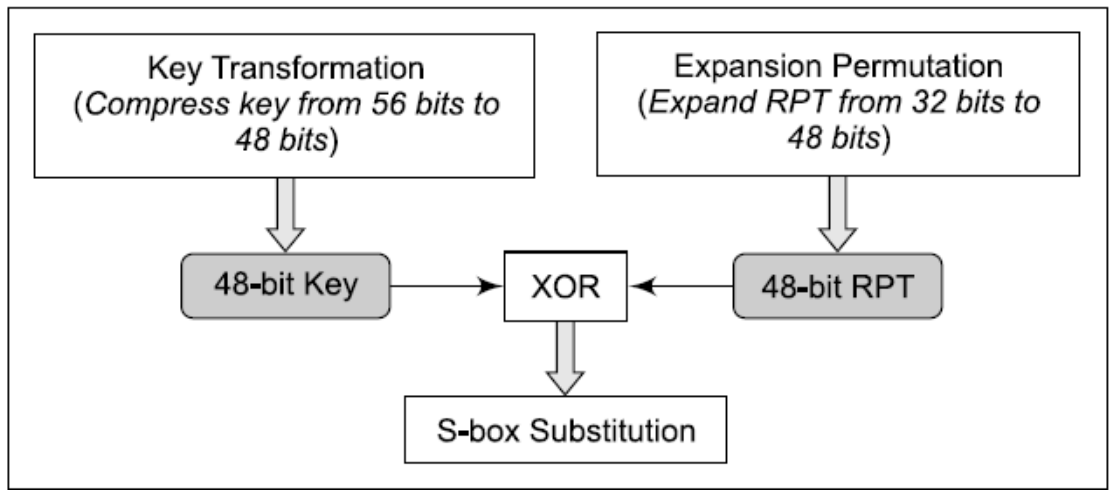
**Step 2: Expansion Permutation**

- Recall that after initial permutation, we had two 32-bit plain text areas, called Left Plain Text (LPT) and Right Plain Text (RPT). During expansion permutation, the RPT is expanded from 32 bits to 48 bits. Besides increasing the bit size from 32 to 48, the bits are permuted as well, hence the name expansion permutation. This happens as follows:
  - 1. The 32-bit RPT is divided into 8 blocks, with each block consisting of 4 bits.
  - 2. Next, each 4-bit block of the above step is then expanded to a corresponding 6-bit block. That is, per 4-bit block, 2 more bits are added. What are these two bits? They are actually the repeated first and the fourth bits of the 4-bit block. The second and the third bits are written down as they were in the input. This is shown in Fig. Note that the first bit inputted is outputted to the second output position, and also repeats in output position 48. Similarly, the 32nd input bit is found in the 47th output position as well as in the first output position.
- As we can see, the first input bit goes into the second and the 48th output positions. The second input bit goes into the third output position, and so on.
- As we have seen, firstly, the key-transformation process compresses the 56-bit key to 48 bits. Then, the expansion permutation process expands the 32-bit RPT to 48 bits. Now, the 48-bit key is XORed with the 48-bit RPT, and the resulting output is given to the next step, which is the S-box substitution

MKK


Original Right Plain Text (RPT) of 32 bits
...
Block 1 (4 bits)
Block 2 (4 bits)
Block 8 (4 bits)

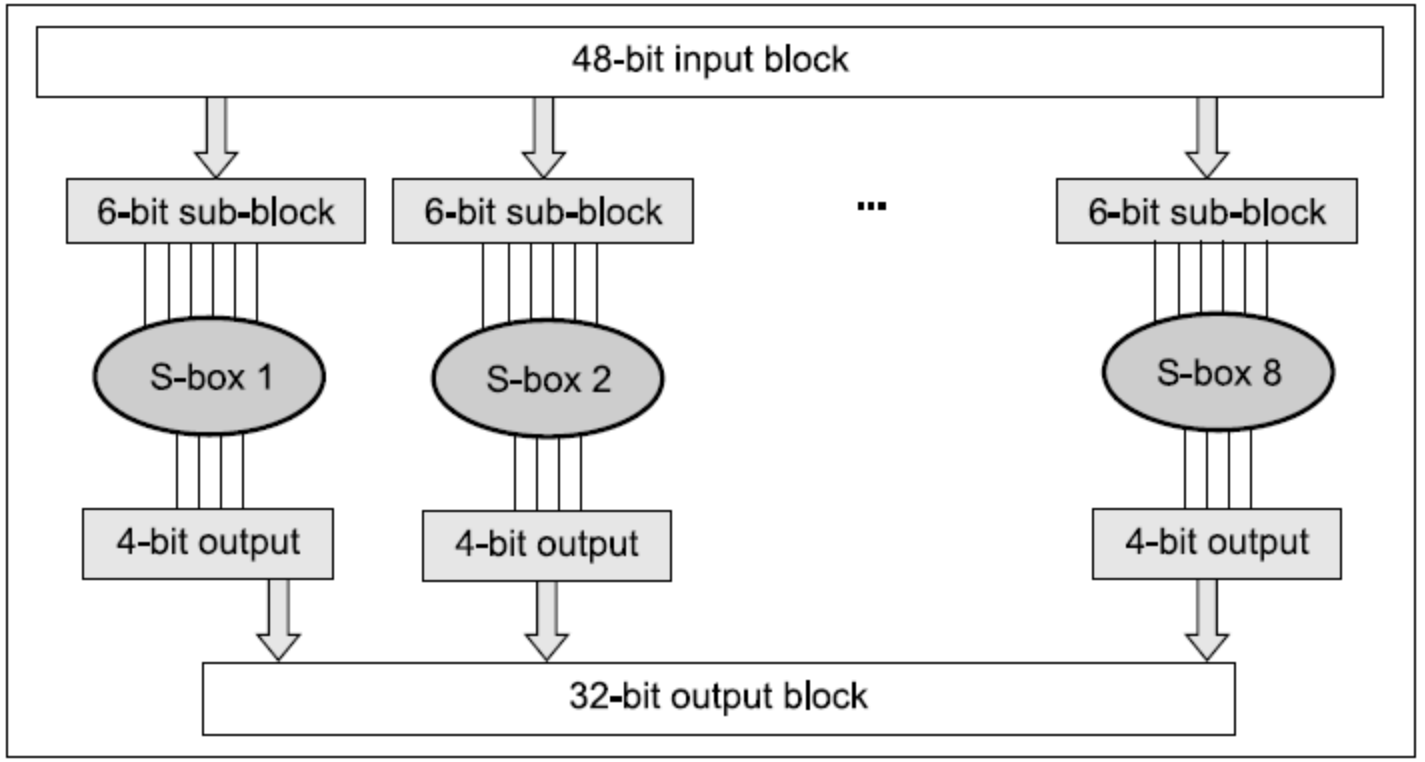

Input Block 1 (4 bits)
Input Block 2 (4 bits)
Input Block 3 (4 bits)
1 2 3 4
5 6 7 8
29 30 31 32
1 2 3 4 5 6
7 8 9 10 11 12
43 44 45 46 47 48
Output Block 1 (6 bits)
Output Block 2 (6 bits)
Output Block 8 (6 bits)


Key Transformation
(Compress key from 56 bits to 48 bits)
Expansion Permutation
(Expand RPT from 32 bits to 48 bits)
48-bit Key
XOR
48-bit RPT
S-box Substitution

MKK

## Step 3: S-box Substitution

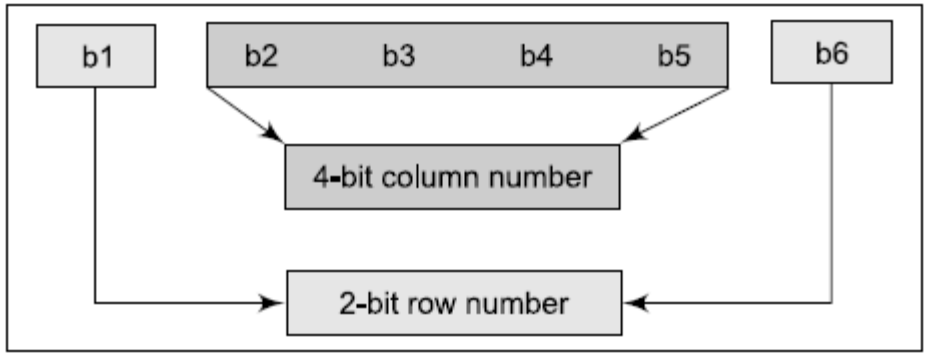
- S-box substitution is a process that accepts the 48-bit input from the XOR operation involving the compressed key and expanded RPT, and produces a 32-bit output using the substitution technique. The substitution is performed by eight substitution boxes (also called as Sboxes). Each of the eight S-boxes has a 6-bit input and a 4-bit output. The 48-bit input block is divided into 8 sub-blocks (each containing 6 bits), and each such sub-block is given to an S-box. The S-box transforms the 6-bit input into a 4-bit output




MKK

- What is the logic used by S-box substitution for selecting only four of the six bits? We can conceptually think of every S-box as a table that has 4 rows (numbered 0 to 3) and 16 columns (numbered 0 to 15). Thus, we have 8 such tables, one for each S-box. At the intersection of every row and column, a 4-bit number (which will be the 4-bit output for that S-box) is present.
- The 6-bit input indicates which row and column, and therefore, which intersection is to be selected, and thus, determines the 4-bit output.

How is it done? Let us assume that the six bits of an S-box are indicated by b1, b2, b3, b4, b5, and b6.
Now, bits b1 and b6 are combined to form a two-bit number. Two bits can store any decimal number between 0 (binary 00) and 3 (binary 11). This specifies the row number. The remaining four bits b2, b3, b4, b5 make up a four-bit number, which specifies the column number between decimal 0 (binary 0000) and 15 (binary 1111). Thus, the 6-bit input automatically selects the row number and column number for the selection of the output.




| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |

Fig.3.32(a) S-box1

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |

Fig.3.32(b) S-box2

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |

Fig.3.32(c) S-box3

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |

Fig.3.32(d) S-box4

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |

Fig.3.32(e) S-box5

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |

Fig.3.32(f) S-box6

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |

Fig. 3.32(g) S-box 7

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

Fig. 3.32(h) S-box 8

**Step 4: P-box Permutation**

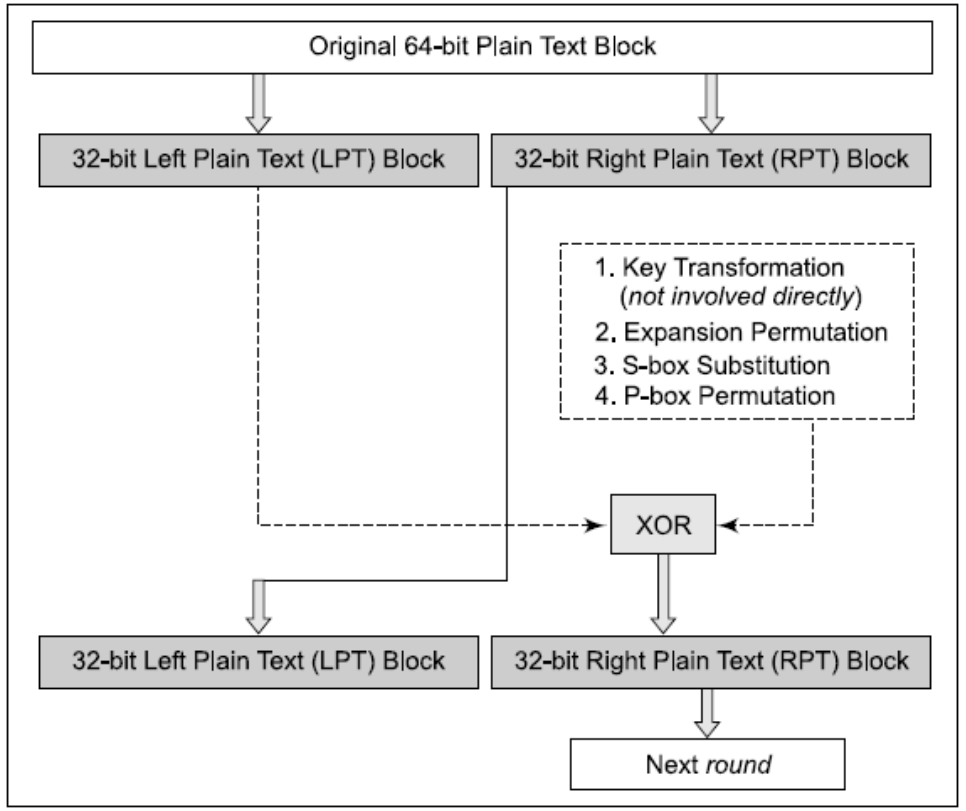
- The output of S-box consists of 32 bits. These 32 bits are permuted using a P-box. This straightforward permutation mechanism involves simple permutation (i.e. replacement of each bit with another bit, as specified in the P-box table, without any expansion or compression). This is called P-box permutation.

- The P-box is shown in Fig. For example, a 16 in the first block indicates that the bit at position 16 of the original input moves to the bit at position 1 in the output, and a 10 in the block number 16 indicates that the bit at the position 10 of the original input moves to bit at the position 16 in the output.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

## Step 5: XOR and Swap

- Note that we have been performing all these operations only on the 32-bit right half portion of the 64-bit original plain text (i.e. on the RPT). The left half portion (i.e. LPT) was untouched so far. At this juncture, the left half portion of the initial 64-bit plain text block (i.e. LPT) is XORed with the output produced by P-box permutation. The result of this XOR operation becomes the new right half (i.e. RPT). The old right half (i.e. RPT) becomes the new left half, in a process of swapping.




MKK

### *4. Final Permutation*

- At the end of the 16 rounds, the final permutation is performed (only once).The output of the final permutation is the 64-bit encrypted block.

### *5. DES Decryption*

- From the above discussion of DES, we might get a feeling that it is an extremely complicated encryption scheme, and therefore, the decryption using DES would employ a completely different approach.
- To most people's surprise, the same algorithm used for encryption in DES also works for decryption!
- The values of the various tables and the operations as well as their sequence are so carefully chosen that the algorithm is reversible.
- The only difference between the encryption and the decryption process is the reversal of key portions. If the original key K was divided into K1, K2, K3, …, K16 for the 16 encryption rounds, then for decryption, the key should be used as K16, K15, K14, …, K1.

MKK

# INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA)

- The International Data Encryption Algorithm (IDEA) is perceived as one of the strongest cryptographic algorithms. It was launched in 1990, and underwent certain changes in names
- Although it is quite strong, IDEA is not as popular as DES for two primary reasons.

| Year | Name |
|---|---|
| 1990 | Proposed Encryption Standard (PES). |
| 1991 | Improved Proposed Encryption Standard (IPES). |
| 1992 | International Data Encryption Algorithm (IDEA). |

Firstly, it is patented unlike DES, and therefore, must be licensed before it can be used in commercial applications.

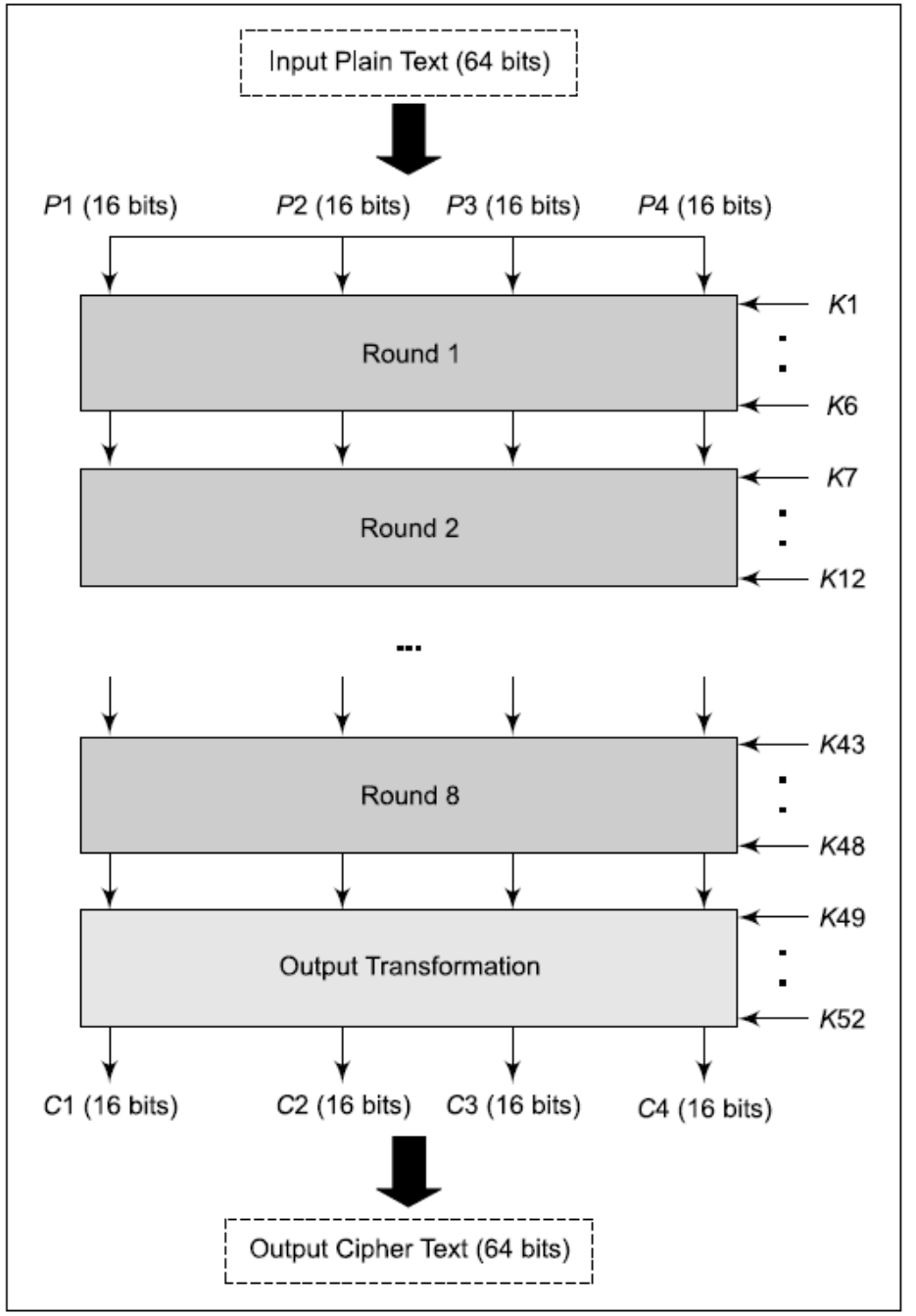
Secondly, DES has a long history and track record as compared to IDEA. However one popular email privacy technology known as Pretty Good Privacy (PGP) is based on IDEA.

MKK

## *1. Basic Principles*

- Technically, IDEA is a block cipher. Like DES, it also works on 64-bit plain-text blocks. The key is longer, however, and consists of 128 bits.
- IDEA is reversible like DES, that is, the same algorithm is used for encryption and decryption. Also, IDEA uses both diffusion and confusion for encryption. Also, IDEA uses both diffusion and confusion for encryption.
- The 64-bit input plaintext block is divided into four portions of plain text (each of size 16 bits), say P1 to P4.
- Thus, P1 to P4 are the inputs to the first round of the algorithm There are eight such rounds. As we mentioned, the key consists of 128 bits.
- In each round, six subkeys are generated from the original key. Each of the subkeys consists of 16 bits

These six subkeys are applied to the four input blocks P1 to P4.
Thus, we will have keys up to K48 for 8 rounds.

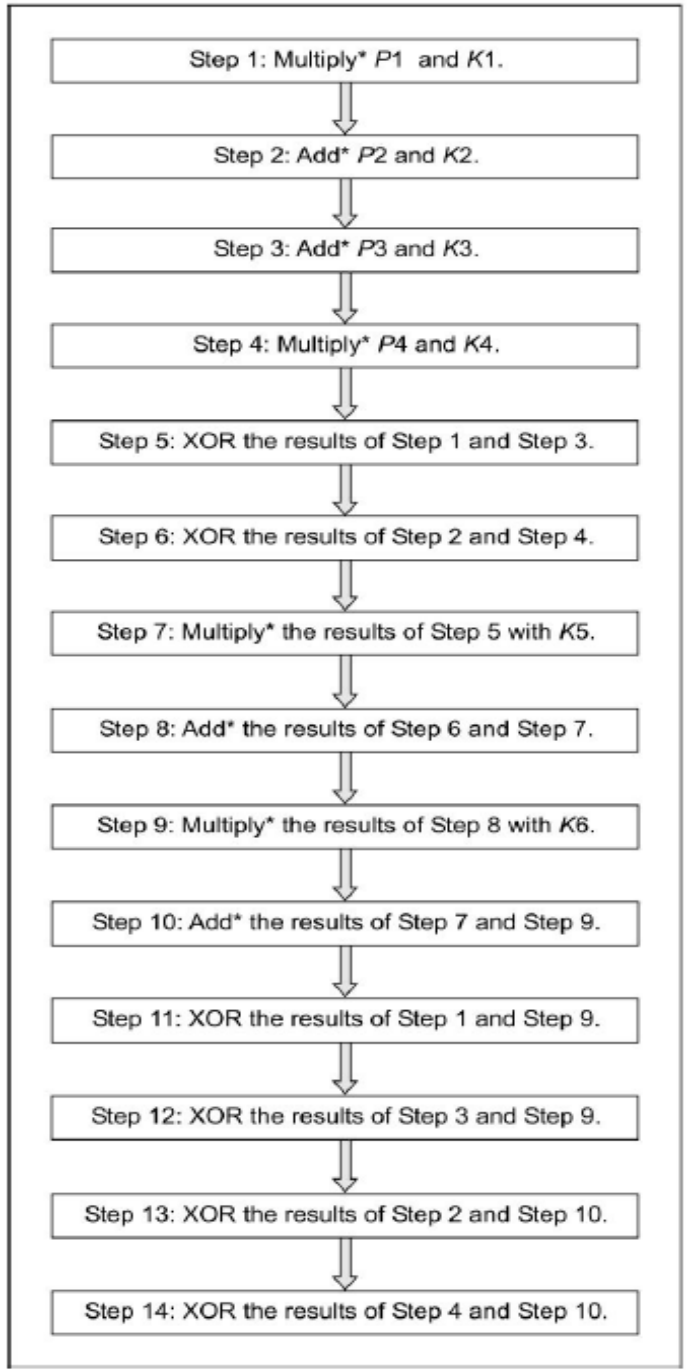


The final step consists of an output transformation, which uses just four subkeys (K49 to K52). The final output produced is the output produced by the output transformation step, which is four blocks of cipher text named C1 to C4 (each consisting of 16 bits). These are combined to form the final 64-bit cipher-text block.

MKK

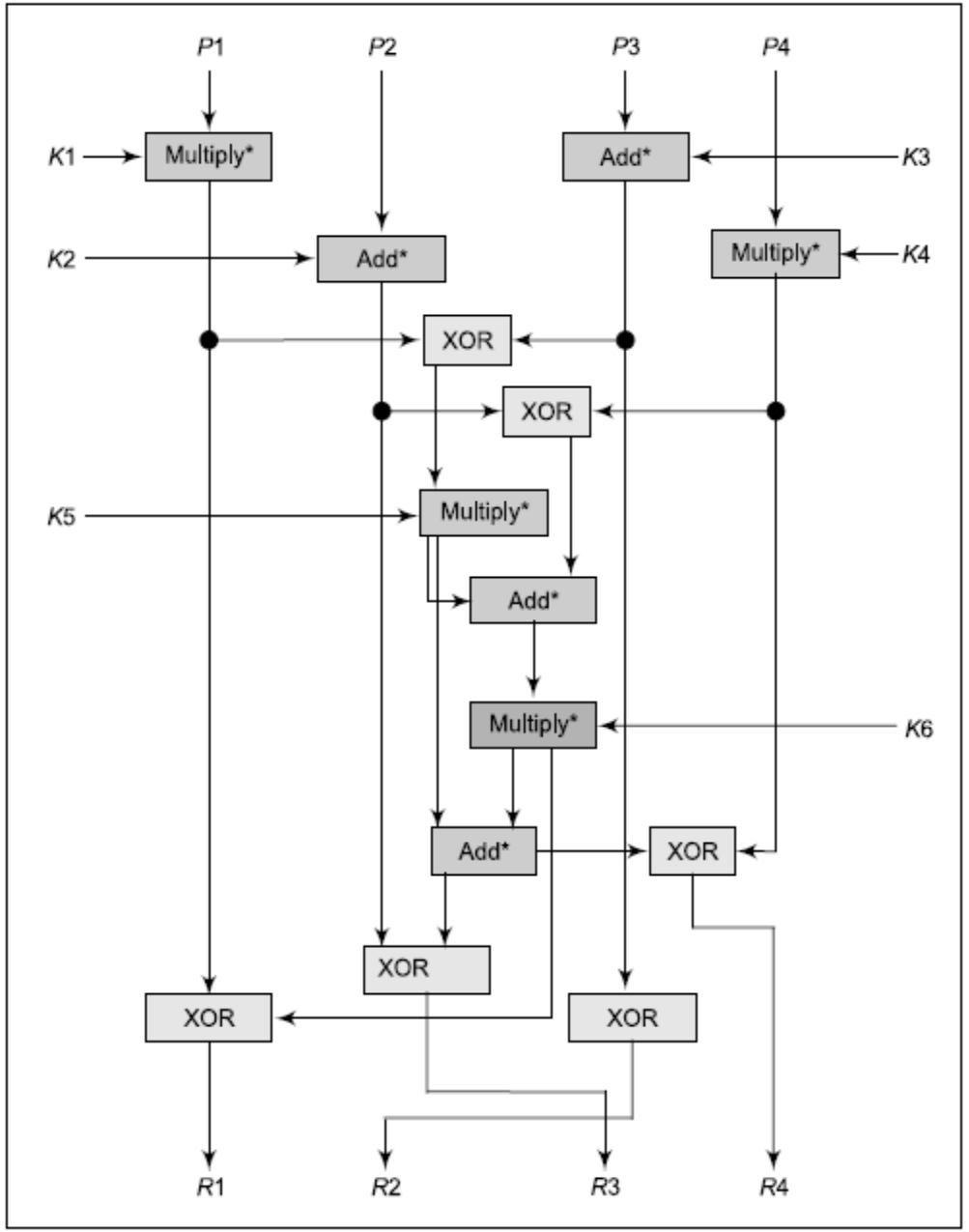
## *2. Rounds*

- We have mentioned that there are 8 rounds in IDEA. Each round involves a series of operations on the four data blocks using six keys.
- There are multiplications, additions and XOR operations.
  Note that we have put an asterisk in front of the words add and multiply, causing them to be shown as Add* and Multiply*, respectively.
- These are not mere additions and multiplications. Instead, these are addition modulo $2^{16}$ (i.e. addition modulo 65536) and multiplication modulo $2^{16} + 1$ (i.e. multiplication modulo 65537), respectively.
- Let us examine the case of the normal binary addition with an example. Suppos IDEA. Further, let us assume that P2 = 1111111100000000 and K2 = 1111111111000001.
- Then the normal addition will produce a number that consists of 17 bits (i.e.11111111011000001).
- To reduce this number (which is 130753 in decimal) to a 16-bit number. we take modulo 65536 of this.
- 130753 modulo 65536 yields 65217, which is 1111111011000001 in binary, and is a 16-bit number, which fits well in our scheme.
- This is why modulo arithmetic is required in IDEA.



Let us now re-look at the details of one round in a more symbolic fashion

- The input blocks are shown as P1 to P4, the subkeys are denoted by K1 to K6, and the output of this step is denoted by R1 to R4 (and not C1 to C4, because this is not the final cipher text—it is an intermediate output, which will be processed in further rounds as well as in the output transformation step.

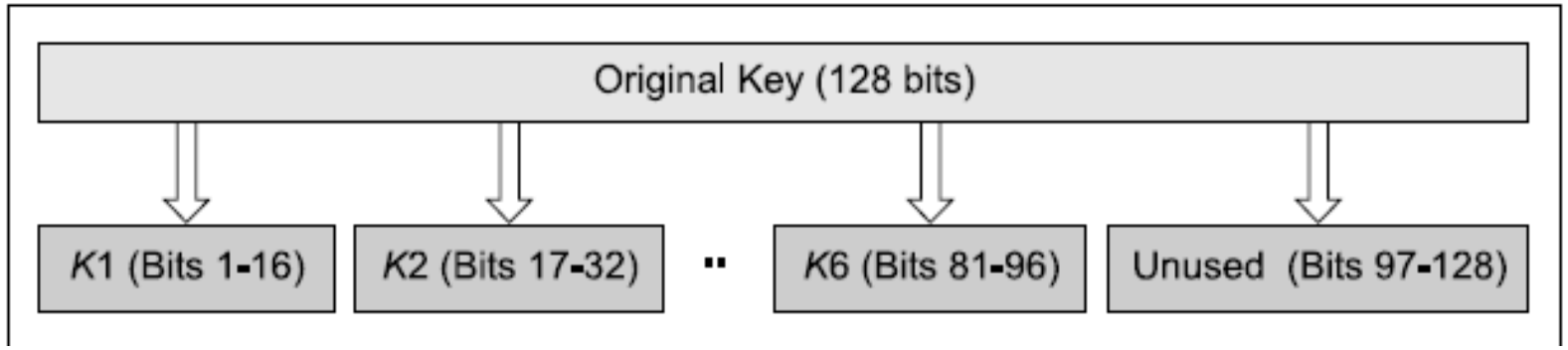


### *3. Subkey Generation for a Round*

- As we mentioned, each of the eight rounds makes use of six subkeys (so, 8 * 6 = 48 subkeys are required for the rounds), and the final output transformation uses four subkeys (making a total of 48 + 4 = 52 subkeys overall). From an input key of 128 bits, how are these 52 subkeys generated

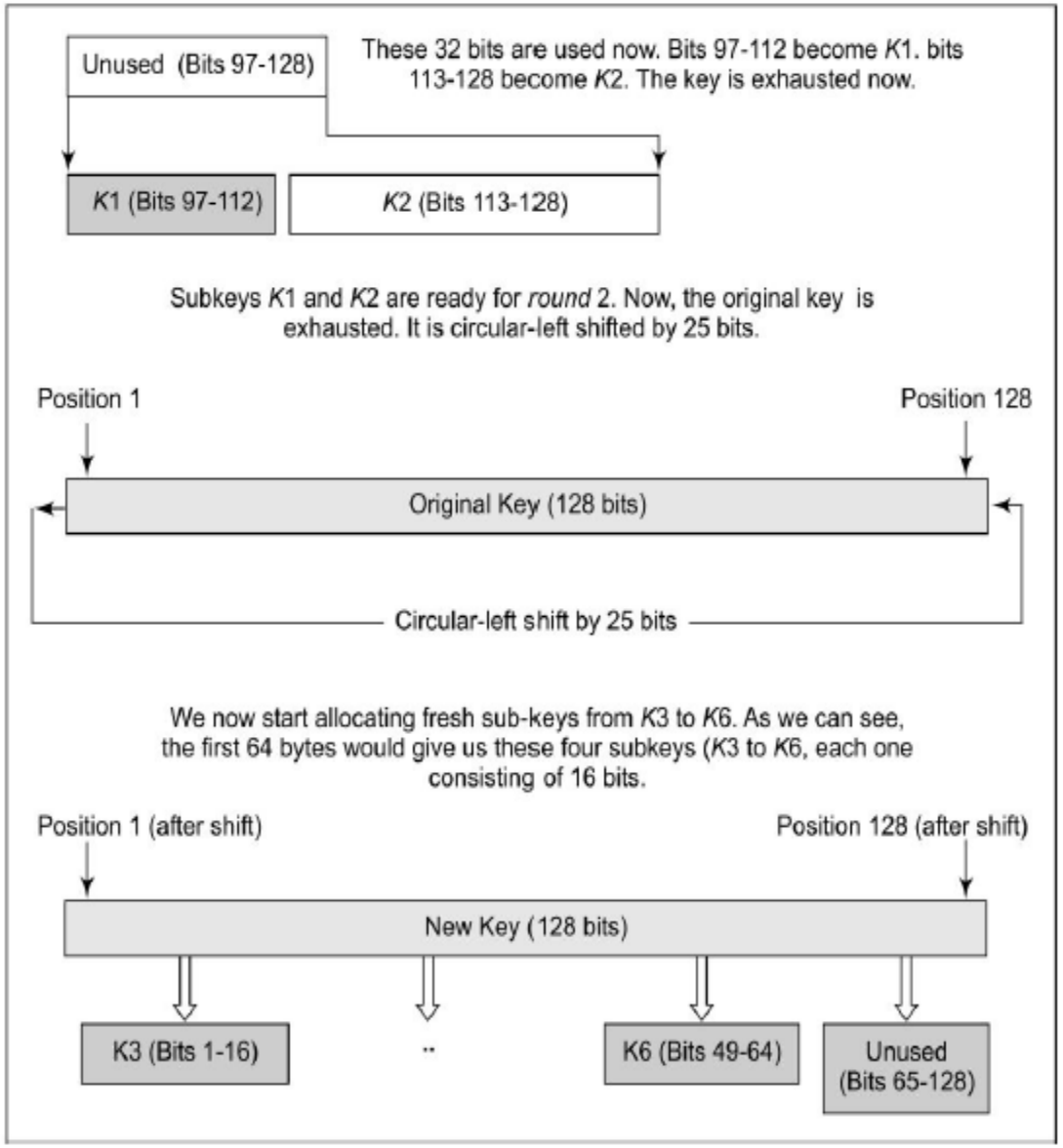
**(a) First Round**

- We know that the initial key consists of 128 bits, from which 6 subkeys K1 to K6 are generated for the first round. Since K1 to K6 consist of 16 bits each, out of the original 128 bits, the first 96 bits (6 subkeys x 16 bits per subkey) are used for the first round. Thus, at the end of the first round, bits 97–128 of the original key are unused.



**(b) Second Round**

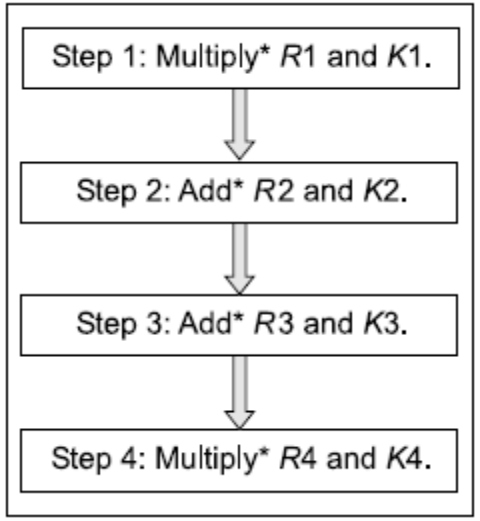
- As we can see, bits 1–96 are adequate to produce subkeys K1 to K6 for round 1. For the second round, we can utilize the 32 unused key bits at positions 97 to 128 (which would give us two subkeys, each of 16 bits).
- To get the remaining 64 bits for the second round, IDEA employs the technique of key shifting. At this stage, the original key is shifted left circularly by 25 bits.
- We can now imagine that the unused bits of the second round (i.e. bit positions 65-128) will firstly be used in round 3, and then a circular-left shift of 25 bits will be performed once again. From the resulting shifted key, we would extract the first 32 bits, which covers the shortfall in the key bits for this round. This process would go on for the remaining rounds on similar line.



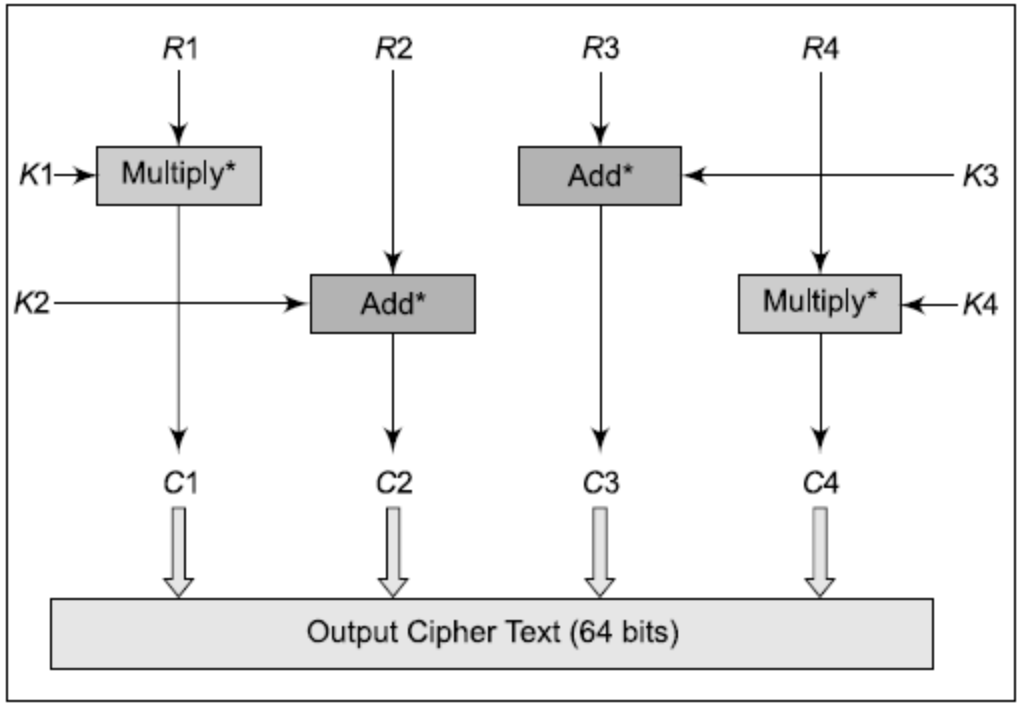

| Round | Details of the subkey generation and use |
|---|---|
| 1 | Bit positions 1-96 of the initial 128-bit key would be used. This would give us 6 subkeys *K*1 to *K*6 for round 1. Key bits 97 to 128 are available for the next round. |
| 2 | Key bits 97 to 128 make up subkeys *K*1 and *K*2 for this round. A 25-bit shift on the original key happens, as explained. Post this shifting; the first 64 bits are used as subkeys *K*3 to *K*6 for this round. This leaves bits 65 to 128 unused for the next round. |
| 3 | Unused key bits 65 to 128 are used as subkeys *K*1 to *K*4 of this round. Upon key exhaustion, another 25-bit shift happens, and bits 1 to 32 of the shifted key are used as subkeys *K*5 and *K*6. This leaves bits 33 to 128 unused for the next round. |
| 4 | Bits 33 to 128 are used for this round, which is perfectly adequate. No bits are unused at this stage. After this, the current key is again shifted. |
| 5 | This is similar to round 1. Bit positions 1-96 of the current 128-bit key would be used. This would give us 6 subkeys *K*1 to *K*6 for round 1. Key bits 97 to 128 are available for the next round. |
| 6 | Key bits 97 to 128 make up subkeys *K*1 and *K*2 for this round. A 25-bit shift on the original key happens, as explained. Post this shifting; the first 64 bits are used as subkeys *K*3 to *K*6 for this round. This leaves bits 65 to 128 unused for the next round. |
| 7 | Unused key bits 65 to 128 are used as subkeys *K*1 to *K*4 of this round. Upon key exhaustion, another 25-bit shift happens, and bits 1 to 32 of the shifted key are used as subkeys *K*5 and *K*6. This leaves bits 33 to 128 unused for the next round. |
| 8 | Bits 33 to 128 are used for this round, which is perfectly adequate. No bits are unused at this stage. After this, the current key is again shifted for the *output transformation* round. |

## *4. Output Transformation*

- The output transformation is a one-time operation. It takes place at the end of the 8th round. The input to the output transformation is, of course, the output of the 8th round.
- This is, as usual, a 64-bit value divided into four sub-blocks (say R1 to R4, each consisting of 16 bits). Also, four subkeys are applied here, and not six.




The process of the output transformation




MKK

### *5. Subkey Generation for the Output Transformation*

- The process for the subkey generation for the output transformation is exactly similar to the subkey generation process for the eight rounds. Recall that at the end of the eighth and the final round, the key is exhausted and shifted. Therefore, in this round, the first 64 bits make up subkeys K1 to K4, which are used as the four subkeys for this round.

### *6. IDEA Decryption*

- The decryption process is exactly the same as the encryption process. There are some alterations in the generation and pattern of subkeys. The decryption subkeys are actually an inverse of the encryption subkeys.

### *7. The Strength of IDEA*

- IDEA uses a 128-bit key, which is double than the key size of DES. Thus, to break into IDEA, 2128 (i.e. 1038) encryption operations would be required. As before, even if we assume that to obtain the correct key, only half of the possible keys (i.e. half of the key space) needs to be examined and tried out, a single computer performing one IDEA encryption per microsecond would require more than 5400000000000000000000000 years to break IDEA!

MKK

# RC5

RC5 is a symmetric-key block-encryption algorithm developed by Ron Rivest. The main features of RC5 are that it is quite fast as it uses only the primitive computer operations (such as addition, XOR, shift, etc). It allows for a variable number of rounds, and a variable bit-size key to add to the flexibility.

## *1. Basic Principles*

- In RC5, the word size (i.e. input plain-text block size), number of rounds and number of 8-bit bytes (octets) of the key, all can be of variable length.

| Parameter | Allowed Values |
|---|---|
| Word size in bits (RC5 encrypts 2-word blocks at a time) | 16, 32, 64 |
| Number of rounds | 0-255 |
| Number of 8-bit bytes (octets) in the key | 0-255 |

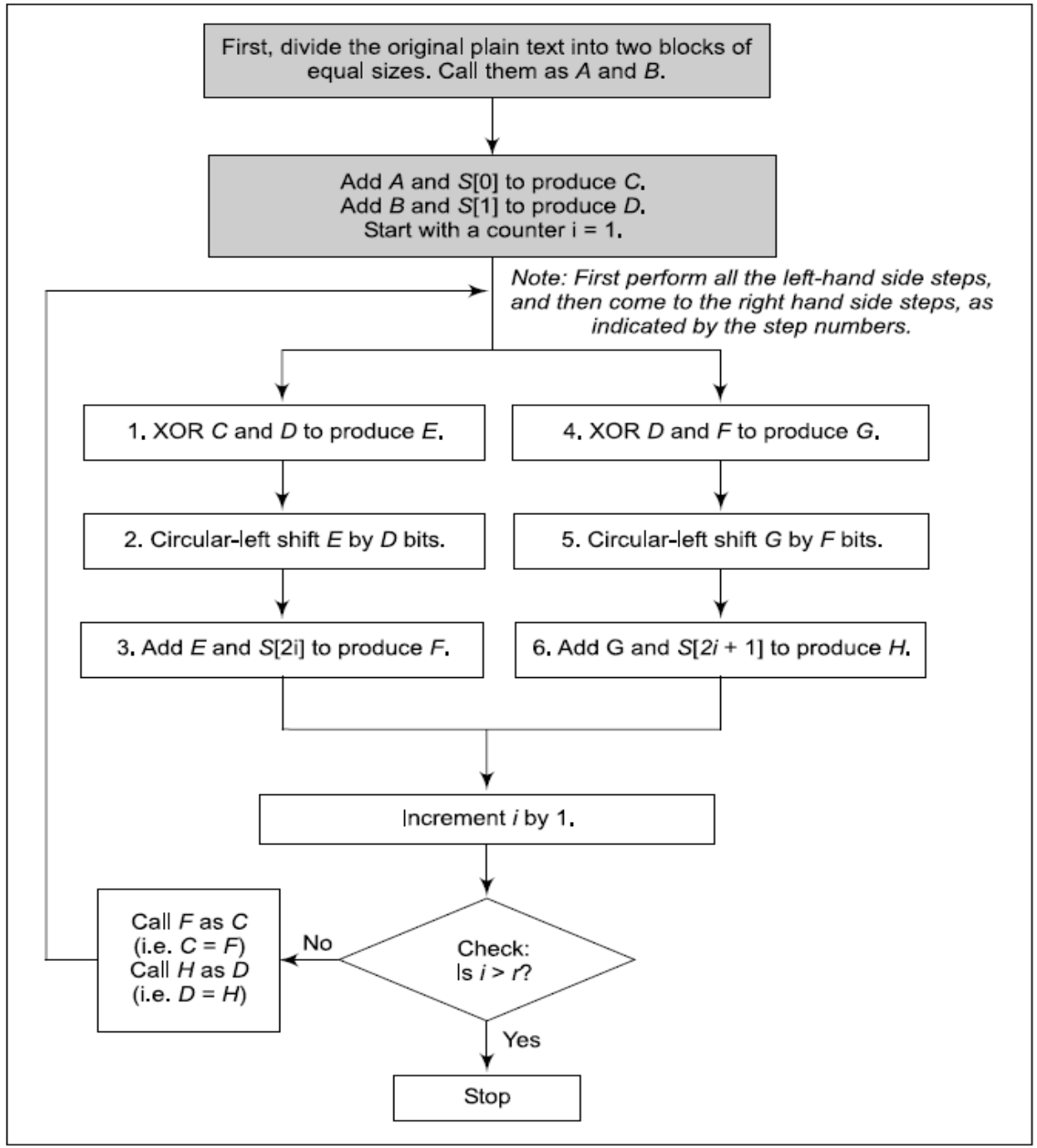
- This is unlike DES, for instance, where the block size must be of 64 bits and the key size must always be of 56 bits; or unlike IDEA, which uses 64-bit blocks and 128-bit keys.
  - The plain-text block size can be of 32, 64 or 128 bits (since 2-word blocks are used).
  - The key length can be 0 to 2040 bits (since we have specified the allowed values for 8-bit keys).

MKK

- The output resulting from RC5 is the cipher text, which has the same size as the input plain text.
- Since RC5 allows for variable values in the three parameters, as specified, a particular instance of the RC5 algorithm is denoted as ***RC5-w/r/b***, where *w = word size in bits, r = number of rounds, b = number of 8-bit bytes in the key*.
- Thus, if we have RC5-32/16/16, it means that we are using RC5 with a block size of 64 bits (remember that RC5 uses 2-word blocks), 16 rounds of encryption, and 16 bytes (i.e. 128 bits) in the key.
- *Rivest has suggested RC5-32/12/16 as the minimum safety version.*

***2. Principles of Operation***

- In RC5, there is one initial operation consisting of two steps (shown shaded in the following fig), then a number of rounds. As we have noted, the number of rounds (r) can vary from 0 to 255.
- For simplicity, we shall assume that we are working on an input plain block with size 64 bits. The same principles operation will apply to other block sizes, in general.


First, divide the original plain text into two blocks of equal sizes. Call them as A and B.
Add A and S[0] to produce C.
Add B and S[1] to produce D.
Start with a counter i = 1.
Note: First perform all the left-hand side steps, and then come to the right hand side steps, as indicated by the step numbers.
1. XOR C and D to produce E.
2. Circular-left shift E by D bits.
3. Add E and S[2i] to produce F.
4. XOR D and F to produce G.
5. Circular-left shift G by F bits.
6. Add G and S[2i + 1] to produce H.
Increment i by 1.
Check: Is i > r?
No
Call F as C (i.e. C = F) Call H as D (i.e. D = H)
Yes
Stop

- In the first two steps of the one-time initial operation, the input plain text is divided into two 32-bit blocks A and B.
- The first two subkeys (we shall later see how they are generated) S[0] and S[1] are added to A and B, respectively.
- This produces C and D respectively, and marks the end of the one-time operation.

Then, the rounds begin. In each round, there are following operations:

- Bitwise XOR
- Left circular-shift
- Addition with the next subkey, for both C and D. This is the addition operation first, and then the result of the addition mod 2w (since w = 32 here, we have $2^{32}$) is performed.

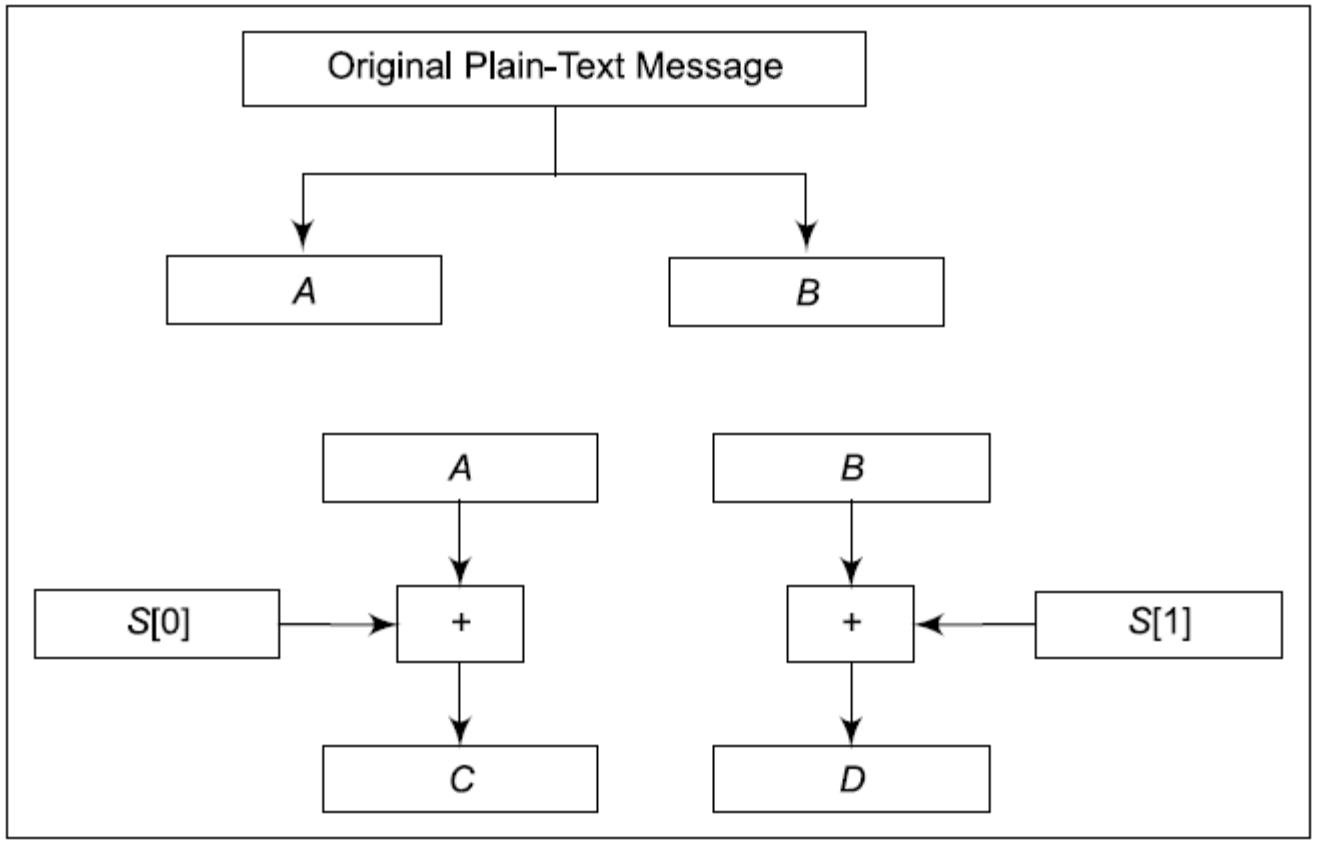
Note that the output of one block is fed back as the input to another block, making the whole logic quite complicated to decipher.

*Let us now look at the algorithm details in step-by-step fashion*

MKK

### *3. One-time Initial Operation*

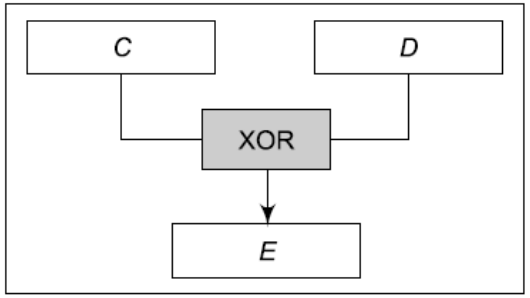
- This operation consists of two simple steps: first, the input plain text is divided into two equal-sized blocks, A and B.
- Then the first subkey, i.e. S[0] is added to A, and the second sub-key, i.e. S[1] is added to B. These operations are mod $2^{32}$, as we have noted, and produce C and D, respectively.



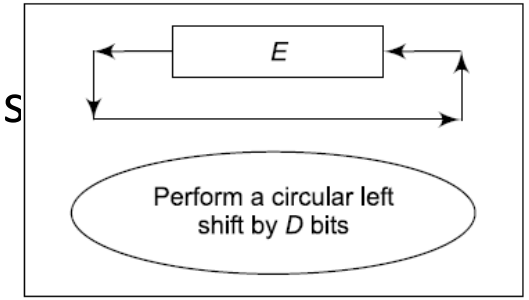
## *4. Details of One Round*

- Now, we observe the results for the first round. The process for the first round will apply for further rounds.

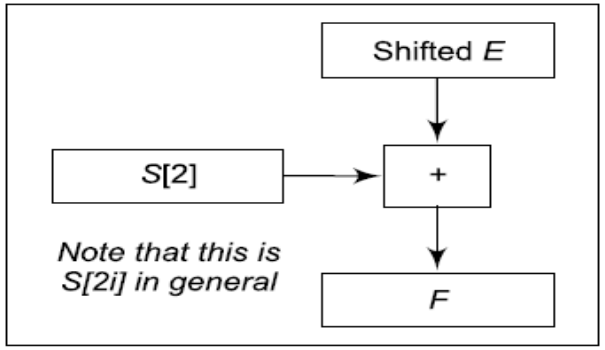
*Step 1: XOR C and D ,*In the first step of each round, C and D are XORed together to form E




*Step 2: Circular-left shift E,* Now, E is circular-left shifted by D positions



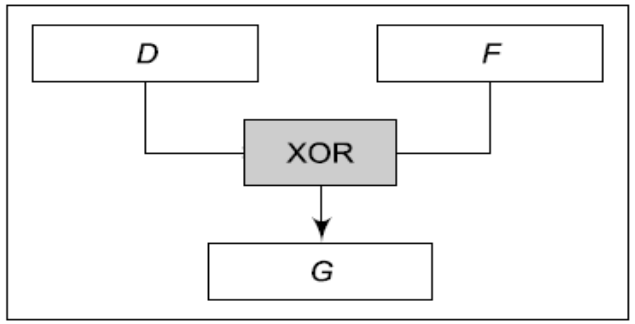

*Step 3: Add E and Next Subkey,* In this step, E is added to the next subkey (which is S[2] for the first round, and S[2i] in general, for any round, where i starts with 1. The output of this process is F.



We would now note that the operations (XOR, circular-left shift and add) in steps 4 to 6 that follow are the same as the operations in steps 1 to 3. The only difference, of course, is that the inputs to the steps 4-6 are different from those to steps 1-3.

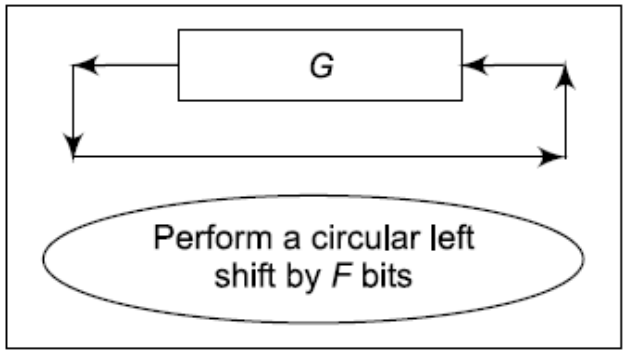
*Step 4: XOR D and F,* This step is similar to step 1.

Here, D and F are XORed to produce G.




*Step 5: Circular-left Shift G,* This step is similar to step 2.

Here, G is circular-left shifted by F Positions

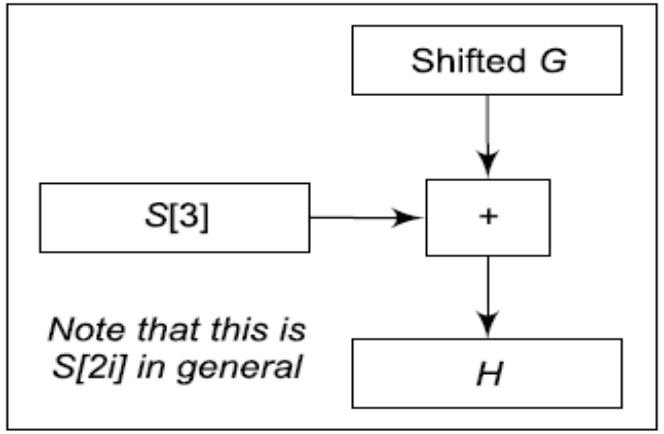



*Step 6: Add G and Next Subkey,* In this step (which is identical to step 3),

G is added to the next subkey (which is S[3] for the first round,

and S[2i + 1] in general, for any round, where i starts with 1.

The output of this process is H.



*Step 7: Miscellaneous Tasks*

In this step, we check to see if all the rounds are over or not. For this, perform the following steps:

- Increment i by 1
- Check to see if i < r

```
i = i + 1

If i < r
        Call F as C again
        Call H as D again
        Go back to step 1
 Else
         Stop
 End-if
```

## *5. Mathematical Representation*

```
A = A + S[0]
B = B + S[1]

For i = 1 to r
        A = ((A XOR B) <<< B) + S[2i]
        B = ((B XOR A) <<< A) + S[2i + 1]
 Next i
```

Mathematical representation of RC5 encryption

```
For i = r to 1 step-1 (i.e. decrement i each time by 1
     A = ((B - S[2i + 1]) >>> A) XOR A
     B = ((A - S[2i]) >>>B) XOR B
Next i
B = B - S[1]
A = A - S[0]
```

Mathematical representation of RC5 decryption

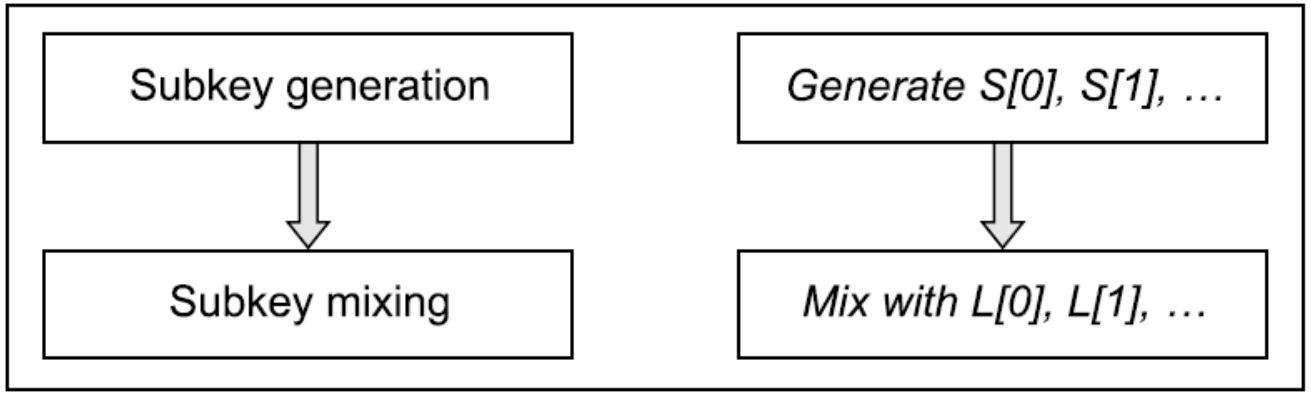
MKK

***6. Subkey Creation***

- Let us now examine the subkey creation process in RC5. This is a two-step process.

  1. In the first step, the subkeys (denoted by S[0], S[1], …) are generated.

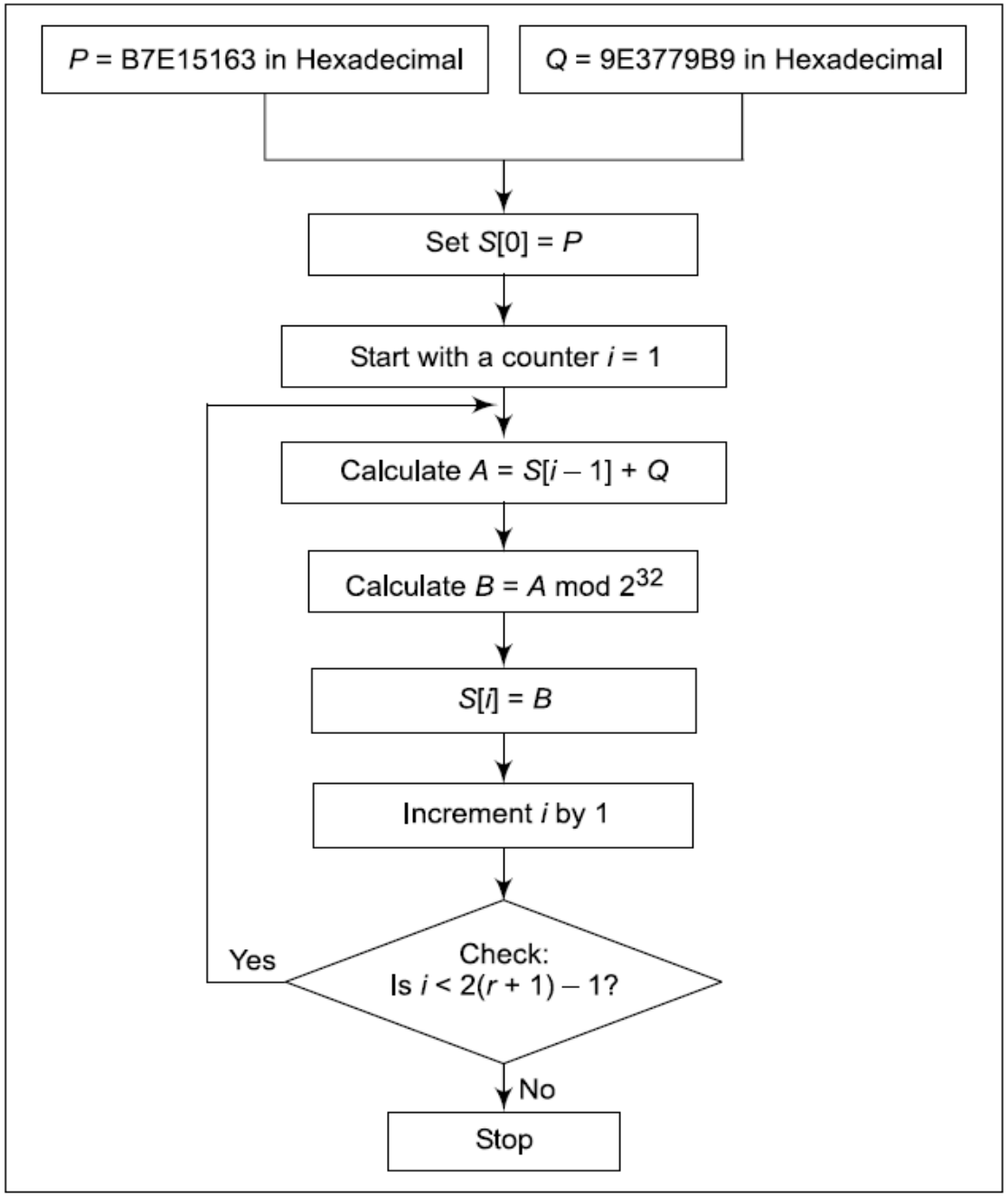
  2. The original key is called L. In the second step, the subkeys (S[0], S[1], …) are mixed with the corresponding subportions of the original key (i.e. L[0], L[1], …).




*Step 1:*

- Subkey Generation In this step, two constants P and Q are used. The array of subkeys to be generated is called as S. The first subkey S[0] is initialized with the value of P.
- Each next subkey (i.e. S[1], S[2], …) is calculated on the basis of the previous sub-key and the constant value Q, using the addition mod $2^{32}$ operations. The process is done 2 (r + 1) – 1 times, where r is the number of rounds, as before.
- Thus, if we have 12 rounds, this process will be done 2 (12 + 1) – 1 times, i.e. 2 (13) – 1 times, i.e. 25 times.
- Thus, we will generate subkeys S[0], S[1], … S[25].

# Subkey generation




```
S[0] = P

For i = 1 to 2 (r + 1) - 1
        S[i] = (S[i - 1] + Q) mod 2^32
Next i
```

Mathematical representation of subkey generation

MKK

*Step 2:*

- Subkey Mixing In the subkey mixing stage, the subkeys S[0], S[1], … are mixed with the subportions of the original key, i.e. L[0], L[1], … L[c]. Note that c is the last subkey position in the original key.

```
  i = j = 0
  A = B = 0

Do 3n times (where n is the maximum of 2 (r + 1) and c)

        A = S[i] = (S[i] + A + B) <<< 3
        B = L [i] = (L[i] + A + B) <<< (A + B)

        i = (i + 1) mod 2 (r + 1)
        j = (j + 1) mod c

    End-do
```

Mathematical representation of subkey mixing

MKK

## BLOWFISH

- Blowfish was developed by Bruce Schneier, and has the reputation of being a very strong symmetrickey cryptographic algorithm. According to Schneier, Blowfish was designed with the following objectives in mind.

(a) *Fast :* Blowfish encryption rate on 32-bit microprocessors is 26 clock cycles per byte.

(b) *Compact :* Blowfish can execute in less than 5 KB memory.

(c) *Simple :* Blowfish uses only primitive operations, such as addition, XOR and table look-up, making its design and implementation simple.

(d) *Secure :* Blowfish has a variable key length up to a maximum of 448 bits long, making it both flexible and secure.

- Blowfish suits applications where the key remains constant for a long time (e.g. communications link encryption), but not where the key changes frequently (e.g. packet switching).

MKK

**Operation**

- Blowfish encrypts 64-bit blocks with a variable-length key. It contains two parts, as follows.

(a) *Subkey Generation* - This process converts the key up to 448 bits long to subkeys totaling 4168 bits.

(b) *Data Encryption* - This process involves the iteration of a simple function 16 times. Each round contains a key-dependent permutation and key- and data-dependent substitution.

**1. Subkey Generation**

(a) The key size ranges from 32 bits to 448 bits. In other words, the key size ranges from 1 to 14 words, each comprising a word of 32 bits. These keys are stored in an array, as follows:

$$K1, K2, \ldots, Kn \quad \text{where } 1 \leq n \leq 14$$

(b) We then have the concept of a P-array, consisting of 18 32-bit sub-keys:

P1, P2… P18

Creation of the P-array is described subsequently.

(c) Four S-boxes, each containing 256 32-bit entries:

- S1,0, S1,1 …, S1,255
- S2,0, S2,1 …, S2,255
- S3,0, S3,1 …, S3,255
- S4,0, S3,1 …, S4,255

MKK

- Creation of the P-array is described subsequently.

Now let us examine how all this information is used to generate subkeys.

(a) Initialize the P-array first, followed by the four S-boxes, with a fixed string.

P1 = 243F6A88

P2 = 85A308D3

….

S4,254 = 578FDFE3

S4,255 = 3AC372E6

(b) Do a bitwise XOR of P1 with K1, P2 with K2, etc., until P18. This works fine till P14 and K14.

At this stage, the key array (K) is exhausted. Hence, for P15 to P18, reuse K1 to K4. In other words, do the following:

P1 = P1 XOR K1

P2 = P2 XOR K2

….

P14 = P14 XOR K14

P15 = P15 XOR K1

P16 = P16 XOR K2

P17 = P17 XOR K3

P18 = P18 XOR K4

MKK

(c) Now take a 64-bit block, with all the 64 bits initialized to value of 0. Use the above P-arrays and S-boxes above (the P-arrays and S-boxes are called subkeys) to run the Blowfish encryption process (described in the next section) on the 64-bit all-zero block. In other words, to generate the subkeys themselves, the Blowfish algorithm is used.

This step would produce a 64-bit cipher text. Divide this into two 32-bit blocks and replace the original values of P1 and P2 with these 32-bit block values, respectively.

(d) Encrypt the output of step (c) above using Blowfish with the modified subkeys. The resulting output would again consist of 64 bits. As before, divide this into two blocks of 32 bits each. Now, replace P3 and P4 with the contents of these two ciphertext blocks.

(e) In the same manner, replace all the remaining P-arrays (i.e. P5 through P18) and then all the elements of the four S-boxes, in order. In each step, the output of the previous step is fed to the Blowfish algorithm to generate the next two 32-bit blocks of the subkey (i.e. P5 and P6, followed by P7 and P8, etc).

## 2. Data Encryption and Decryption

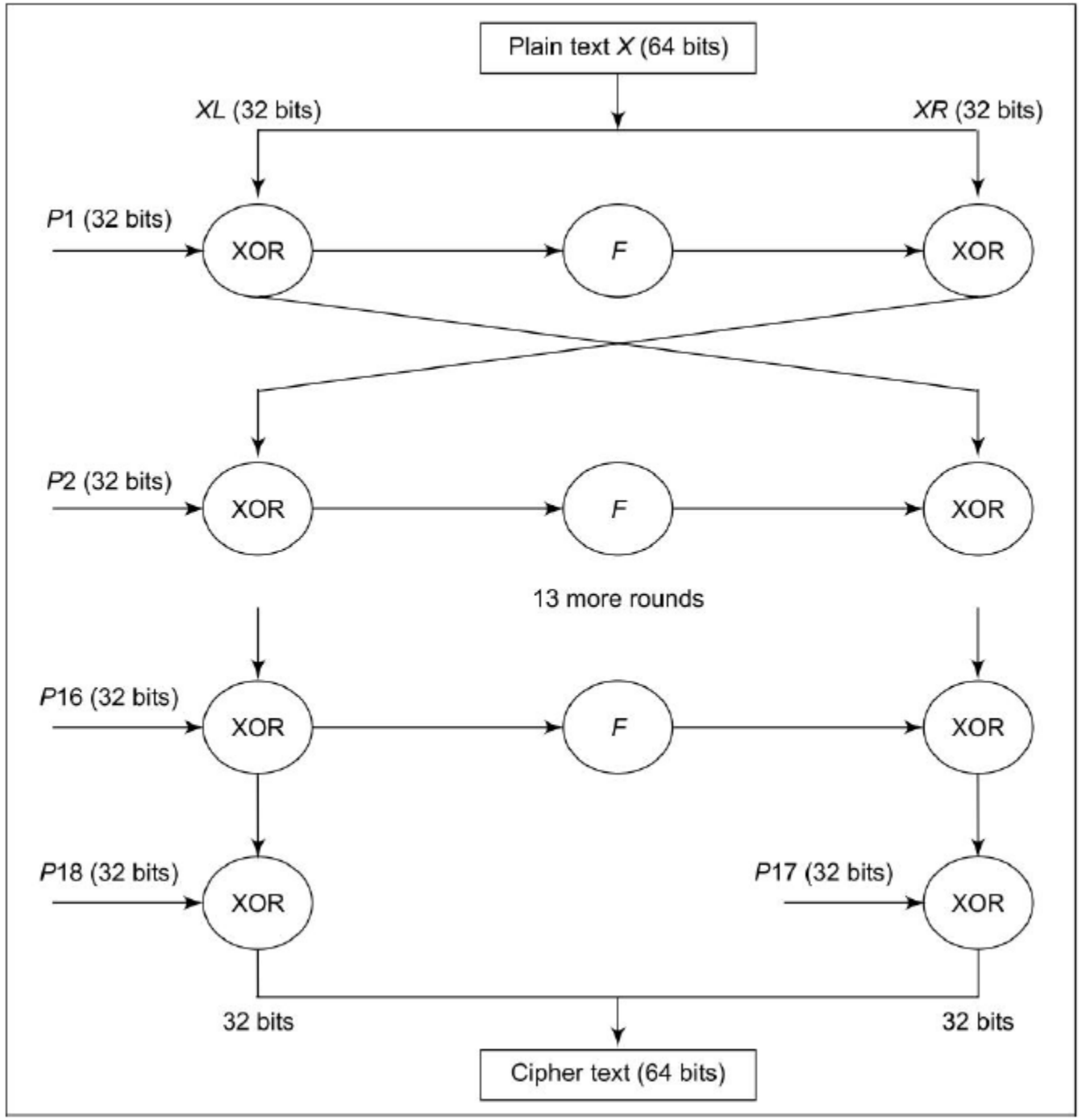
The encryption of a 64-bit block plain-text input X is shown in an algorithmic fashion in Fig. We use the P-arrays and S-boxes during the encryption and decryption processes.

```
1. Divide X into two blocks: XL and XR, of equal sizes. Thus, both XL
   and XR will consist of 32 bits each.

2. For i = 1 to 16

          XL = XL XOR Pi
          XR = F (XL) XOR XR
          Swap XL, XR
   Next i

3. Swap XL, XR (i.e. undo last swap).

4. XL = XL XOR P18.

5. Combine XL and XR back into X.
```

This process is depicted in followingFig

MKK

# Blowfish encryption



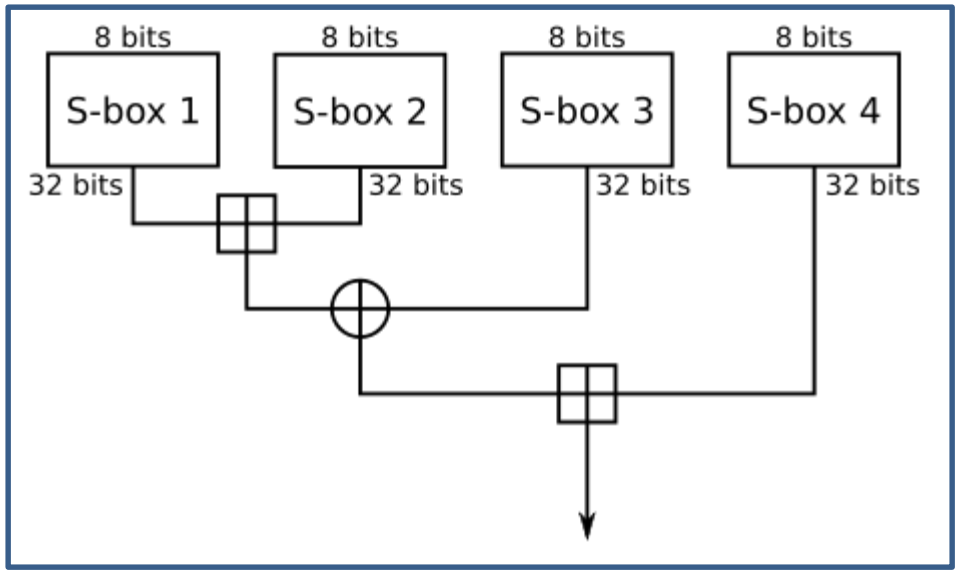

MKK

The function F is as follows:

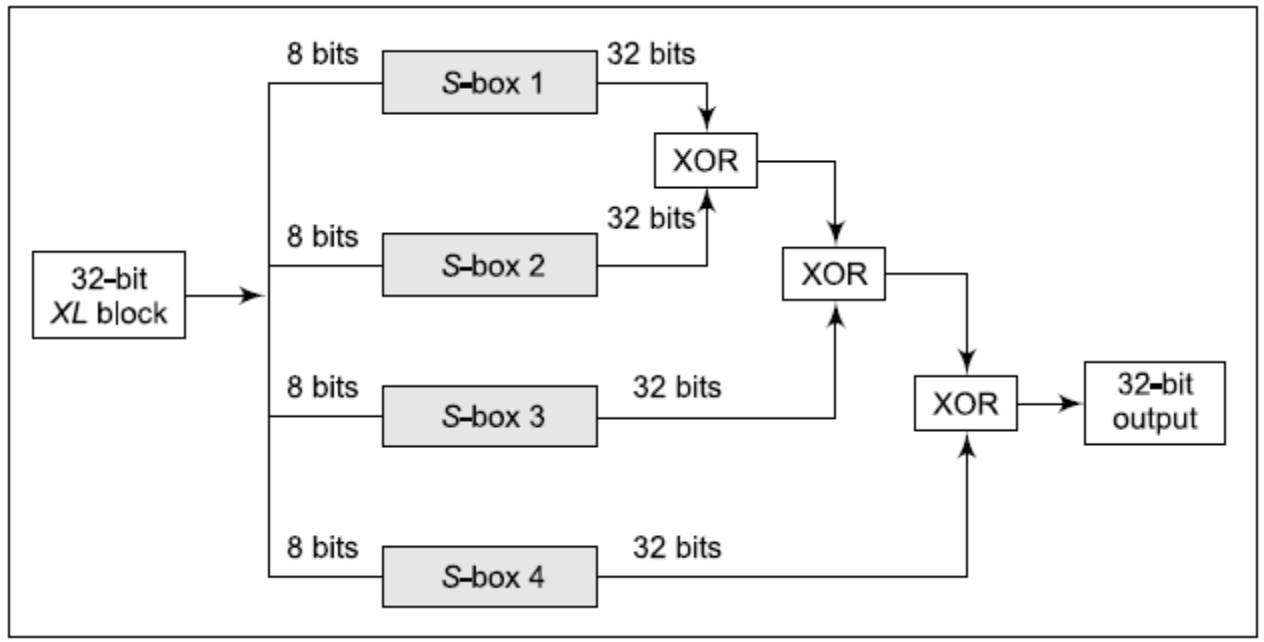
(a) Divide the 32-bit XL block into four 8-bit sub-blocks, named a, b, c, and d.

(b) Compute F [a, b, c, d] = ((S1,a + S2,b) XOR S3,c) + S4,d. For example, if a = 10, b = 95, c = 37, and d = 191, then the computation of F would be:

F [a, b, c, d] = ((S1,10 + S2,95) XOR S3,37) + S4,191




MKK


8 bits
S-box 1
32 bits
XOR
8 bits
32 bits
32-bit
XL block
S-box 2
XOR
8 bits
S-box 3
32 bits
XOR
32-bit
output
8 bits
32 bits
S-box 4

MKK