BLOCK CIPHERS AND THE DATA ENCRYPTION STANDARD BLOCK CIPHER PRINCIPLES:

Many symmetric block encryption algorithms in current use are based on a structure referred to as a Feistel block cipher. For that reason, it is important to examine the design principles of the Feistel cipher. We begin with a **comparison of stream ciphers and block ciphers.**

Stream Ciphers and Block Ciphers:

A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time.

Examples of classical stream ciphers are the autokeyed Vigenère cipher and the Vernam cipher.

block cipher is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used. A block cipher can be used to achieve the same effect as a stream cipher.

Motivation for the Feistel Cipher Structure:

A block cipher operates on a plaintext block of *n* bits to produce a ciphertext block of *n* bits. There are 2^n possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must produce a unique ciphertext block. Such a transformation is called reversible, or nonsingular. The following examples illustrate nonsingular and singular transformations for $n = 2$.

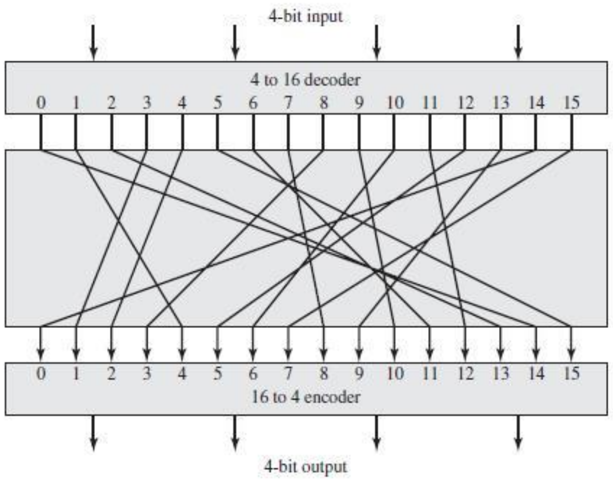
Reversible Mapping

Plaintext	Ciphertext
00	11
01	10
10	00
11	01

Irreversible Mapping

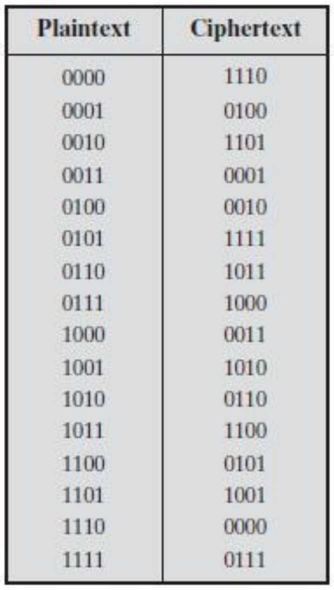
Plaintext	Ciphertext
00	11
01	10
10	01
11	01

In the latter case, a ciphertext of 01 could have been produced by one of two plaintext blocks. So if we limit ourselves to reversible mappings, the number of different transformations is $2^n!$.

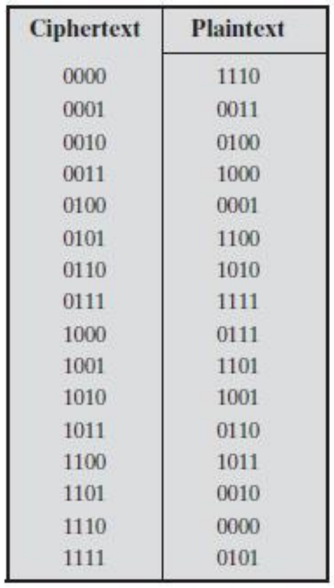


The logic of a general substitution cipher for n=4. A 4-bit input produces one of 16 possible input states, which is mapped by the substitution cipher into a unique one of 16 possible output states, each of which is represented by 4 ciphertext bits. The encryption and decryption mappings can be defined by tabulation.

This is the most general form of block cipher and can be used to define any reversible mapping between plaintext and ciphertext. Feistel refers to this as the ideal block cipher, because it allows for the maximum number of possible encryption mappings from the plaintext block.



Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111



Ciphertext	Plaintext
0000	1110
0001	0011
0010	0100
0011	1000
0100	0001
0101	1100
0110	1010
0111	1111
1000	0111
1001	1101
1010	1001
1011	0110
1100	1011
1101	0010
1110	0000
1111	0101

But there is a practical problem with the ideal block cipher. If a small block size, such as n=4, is used, then the system is equivalent to a classical substitution cipher. Such systems, as we have seen, are vulnerable to a statistical analysis of the plaintext. This weakness is not inherent in the use of a substitution cipher but rather results from the use of a small block size. If n is sufficiently large and an arbitrary reversible substitution between plaintext and ciphertext is allowed, then the statistical characteristics of the source plaintext are masked to such an extent that this type of cryptanalysis is infeasible

The Feistel Cipher:

• **diffusion** – dissipates statistical structure of plaintext over bulk of ciphertext
• **confusion** – makes relationship between ciphertext and key as complex as possible

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 L0 and R0. 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 Li-1 and Ri-1, derived from the previous round, as well as the subkey Ki, derived from the overall key K. in general, the subkeys Ki 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 ki. 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:

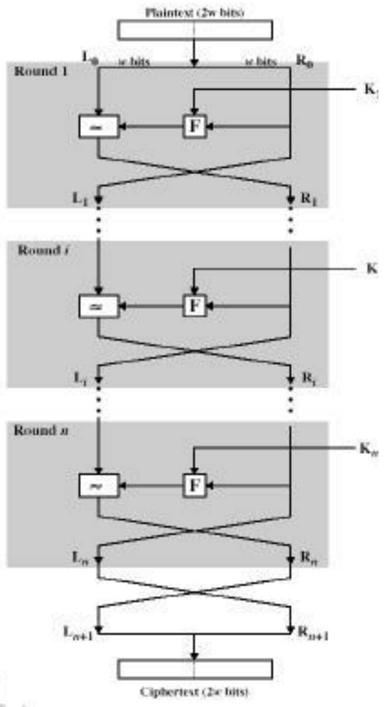
• **Block size:** Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion. Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.

• **Key size:** Larger key size means greater security but may decrease encryption/decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.

• **Number of rounds:** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.

• **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.

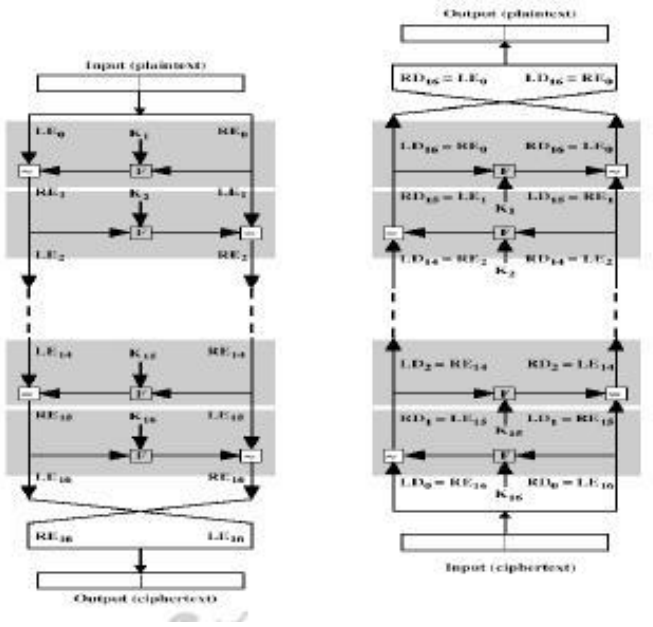
• **Round function F:** Again, greater complexity generally means greater resistance to cryptanalysis.



There are two other considerations in the design of a Feistel cipher:

• **Fast software encryption/decryption:** In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.

• **Ease of analysis:** Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze.



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 ki in reverse order. i.e., kn in the first round, kn-1 in second round and so on. For clarity, we use the notation LEi and REi 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. i.e.,

$$RE_i \,||\, LE_i \text{ (or) equivalently } RD_{16-i} \,||\, LD_{16-i}$$

After the last iteration of the encryption process, the two halves of the output are swapped, so that the cipher text is RE16 || LE16. 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 RE16 || LE16, 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,

$$LE16 = RE15$$
$$RE16 = LE15 \oplus F\ (RE15, K16)$$

On the decryption side,

$$\begin{aligned} LD1 &= RD0 = LE16 = RE15 \\ RD1 &= LD0 \oplus F(RD0, K16) \\ &= RE16 \oplus F(RE15, K16) \\ &= [LE15 \oplus F(RE15, K16)] \oplus F(RE15, K16) \\ &= LE15 \end{aligned}$$

Therefore, $LD1 = RE15$

$RD1 = LE15$

In general, for the ith iteration of the encryption algorithm,

$$LEi = REi\text{-}1$$
$$REi = LEi\text{-}1 \oplus F(REi\text{-}1, Ki)$$

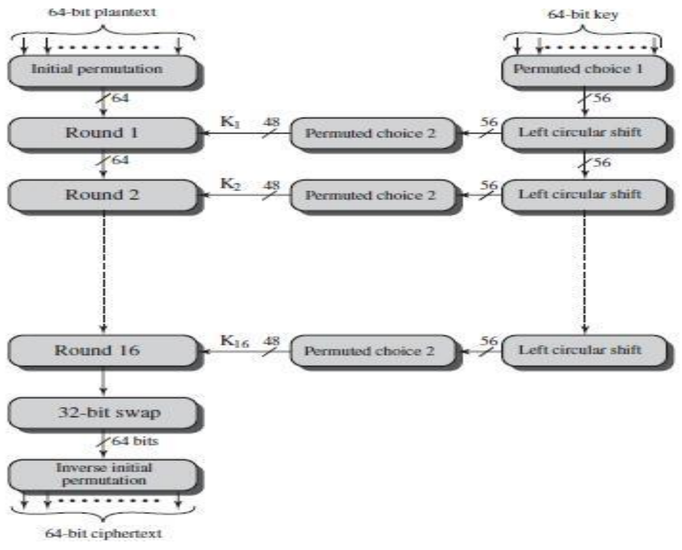
Finally, the output of the last round of the decryption process is $RE_0 \| LE_0$. A 32-bit swap recovers the original plaintext.

DATA ENCRYPTION STANDARD (DES)

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).7 For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption. The DES enjoys widespread use. It has also been the subject of much controversy concerning how secure the DES is. To appreciate the nature of the controversy, let us quickly review the history of the DES.

DES Encryption:

The overall scheme for DES encryption is illustrated in this Figure. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.

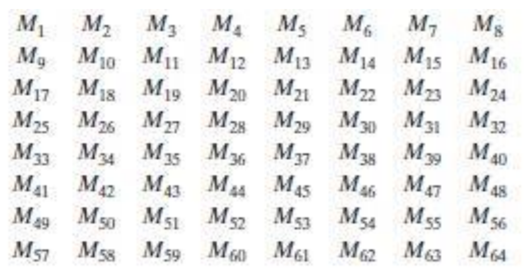


Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*. This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the **preoutput**. Finally, the preoutput is passed through a permutation [IP-1] that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.

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

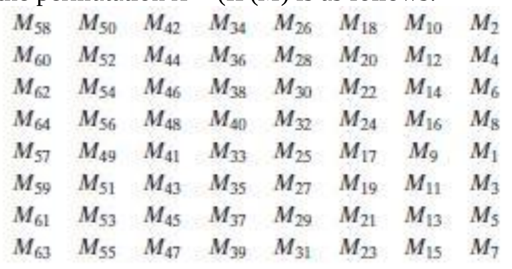
INITIAL PERMUTATION The initial permutation and its inverse are defined by tables, The tables are to be interpreted as follows. The input to a table consists of 64 bits numbered from 1 to 64. The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits.

To see that these two permutation functions are indeed the inverse of each other, consider the following 64-bit input M:



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

where M_i is a binary digit. Then the permutation *X* = (IP(*M*) is as follows:



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

If we then take the inverse permutation Y= IP-1(*X*) = IP-1(IP(*M*)), it can be seen that the original ordering of the bits is restored.

(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

• Initially the key is passed through a permutation function (PC_1)

• For each of the 16 iterations, a subkey (K_i) is produced by a combination of a left circular shift and a permutation (PC_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

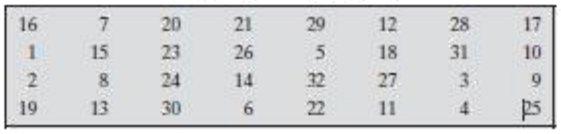
(c) Permuted Choice Two (PC-2)

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

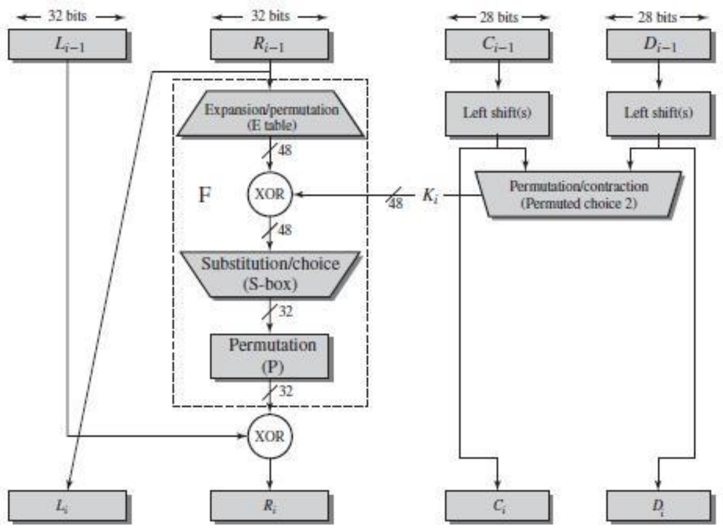
(d) Permutation Function (P)



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

DETAILS OF SINGLE ROUND Below Figure shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$



The E-box expansion permutation - here the 32-bit input data from R_{i-1} is expanded and permuted to give the 48 bits necessary for combination with the 48 bit key (defined in table 2.1). The E-box expansion permutation delivers a larger output by splitting its input into 8, 4-bit blocks and copying every first and fourth bit in each block into the output in a defined manner. The security offered by this operation comes from one bit affecting two substitutions in the S-boxes. This causes the dependency of the output bits on the input bits to spread faster, and is known as the avalanche affect.

2. The bit by bit addition modulo 2 (or exclusive OR) of the E-box output and 48 bit subkey K_i.

3. The S-box substitution - this is a highly important substitution which accepts a 48-bit input and outputs a 32-bit number (defined in table 2.3). The S-boxes are the only non-linear operation in DES and are therefore the most important part of its security. They were very carefully designed although the conditions they were designed under

has been under intense scrutiny since DES was released. The reason was because IBM had already designed a set of S-boxes which were completely changed by the NSA with no explanation why.

The input to the S-boxes is 48 bits long arranged into 8, 6 bit blocks ($b_1, b_2, . . . , b_6$). There are 8 S-boxes ($S_1, S_2, . . , S_8$) each of which accepts one of the 6 bit blocks. The output of each S-box is a four bit number. Each of the S-boxes can be thought of as a 4 × 16 matrix. Each cell of the matrix is identified by a coordinate pair (i, j), where $0 \leq I \leq 3$ and $0 \leq j \leq 15$. The value of i is taken as the decimal representation of the first and last bits of the input to each S-box, i.e. Dec (b_1b_6) =iand the value of j is take from the decimal representation of the inner four bits that remain, S-box matrices contains a 4-bit number which is output once that particular cell is selected by the input.

4. The P-box permutation - This simply permutes the output of the S-box without changing the size of the data (defined in table 2.1). It is simply a permutation and nothing else. It has a one to one mapping of its input to its output giving a 32 bit output from a 32 bit input.

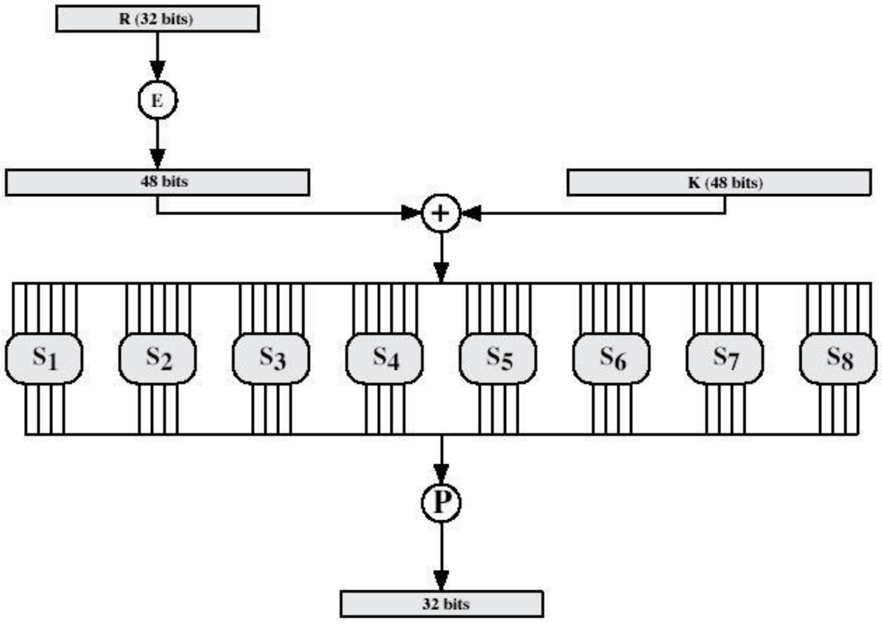
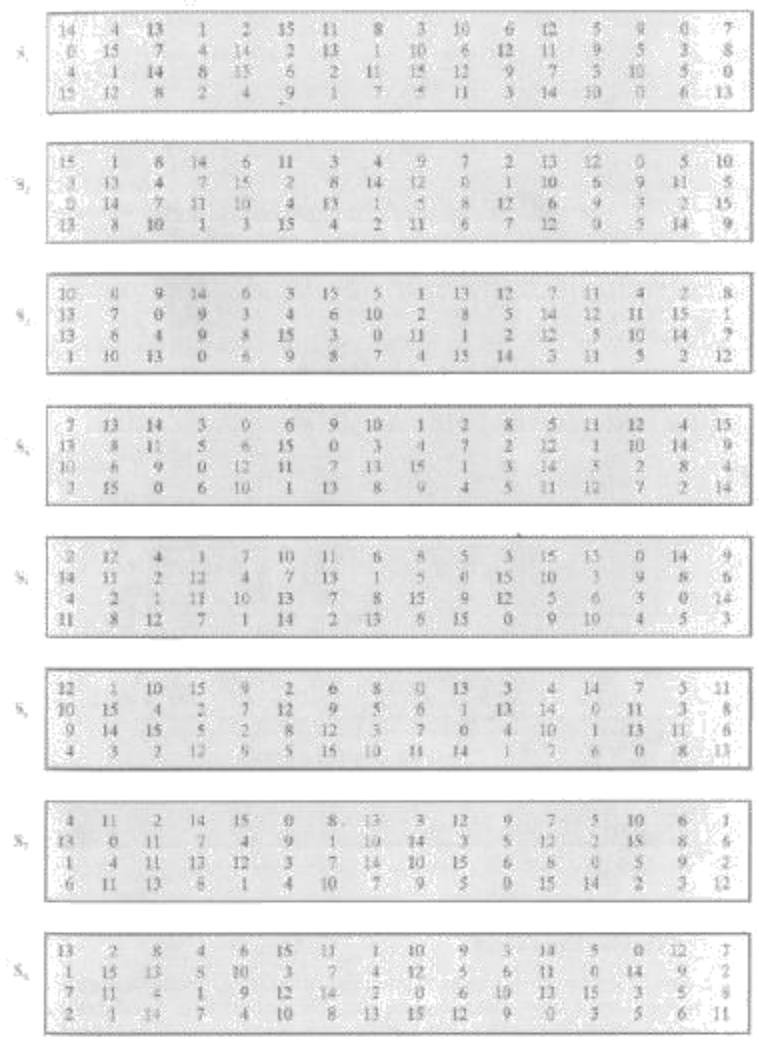


Figure 3.9 Calculation of F(R, K)

Table 3.3 Definition of DES S-Boxes



S_1	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
S_2	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
S_3	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
S_4	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
S_5	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
S_6	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
S_7	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
S_8	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

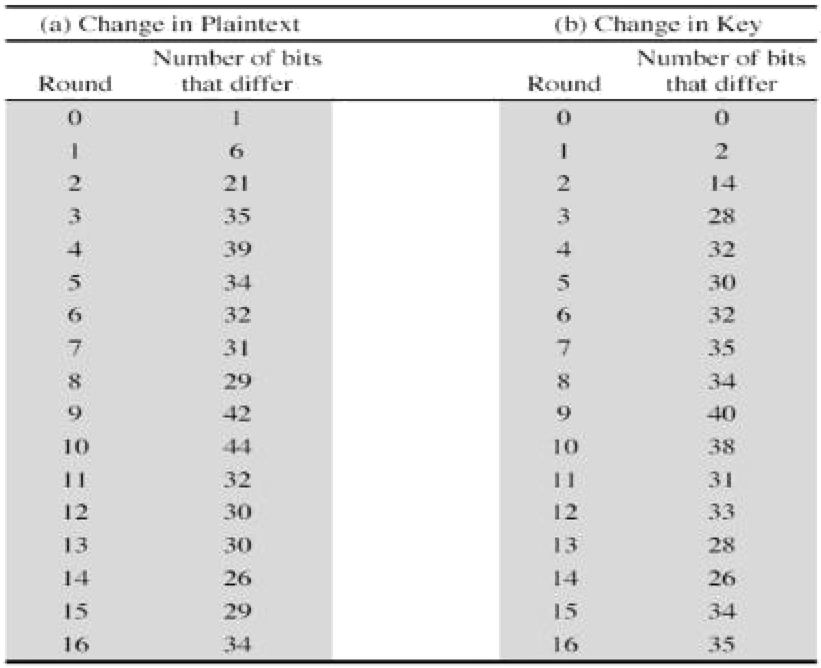
Each row of an S-box defines a general reversible substitution: middle 4 bits of each group of 6-bit input are substituted by S-box output, 1st and last 6th bits define what particular substitution out of to use.

DES decryption:

As with any feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reverse.

Avalanche effect

A desirable property of any encryption algorithm is that a small change in either plaintext or key should produce significant changes in the ciphertext. DES exhibits a strong avalanche effect.



(a) Change in Plaintext		(b) Change in Key	
Round	Number of bits that differ	Round	Number of bits that differ
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

Avalanche effect - a small change in the plaintext produces a significant change in the ciphertext.

Strength of DES

- 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
- Brute force search looks hard
- Recent advances have shown is possible
  - in 1997 on a huge cluster of computers over the Internet in a few months
  - in 1998 on dedicated hardware called "DES cracker" by EFF in a few days ($220,000)
  - in 1999 above combined in 22hrs!
- Still must be able to recognize plaintext
- No big flaw for DES algorithms

AES Evaluation

Security:

This refers to the effort required to cryptanalyze an algorithm. The emphasis in the evaluation was on the practicality of the attack. Because the minimum key size for AES is 128 bits, brute-force attacks with current and projected technology were considered impractical. Therefore, the emphasis, with respect to this point, is cryptanalysis other than a brute-force attack.

Cost:

NIST intends AES to be practical in a wide range of applications. Accordingly, AES must have high computational efficiency, so as to be usable in high-speed applications, such as broadband links.

Algorithm and implementation characteristics:

This category includes a variety of considerations, including flexibility; suitability for a variety of hardware and software implementations; and simplicity, which will make an analysis of security more straightforward

Using these criteria, the initial field of 21 candidate algorithms was reduced first to 15 candidates and then to 5 candidates. By the time that a final evaluation had been done the evaluation criteria, as described in [NECH00], had evolved. The following criteria were used in the final evaluation:

General security:

To assess general security, NIST relied on the public security analysis conducted by the cryptographic community. During the course of the three-year evaluation process, a number of cryptographers published their analyses of the strengths and weaknesses of the various candidates. There was particular emphasis on analyzing the candidates with respect to known attacks, such as differential and linear cryptanalysis. However, compared to the analysis of DES, the amount of time and the number of cryptographers devoted to analyzing Rijndael are quite limited. Now that a single AES cipher has been chosen, we can expect to see a more extensive security analysis by the cryptographic community.

Software implementations:

The principal concerns in this category are execution speed, performance across a variety of platforms, and variation of speed with key size.

Restricted-space environments:

In some applications, such as smart cards, relatively small amounts of random-access memory (RAM) and/or read-only memory (ROM) are available for such purposes as code storage (generally in ROM); representation of data objects such as S-boxes (which could be stored in ROM or RAM, depending on whether pre-computation or Boolean representation is used); and subkey storage (in RAM).

Hardware implementations:

Like software, hardware implementations can be optimized for speed or for size. However, in the case of hardware, size translates much more directly into cost than is usually the case for software implementations. Doubling the size of an encryption program may make little difference on a general-purpose computer with a large memory, but doubling the area used in a hardware device typically more than doubles the cost of the device.

Attacks on implementations:

The criterion of general security, discussed in the first bullet, is concerned with cryptanalytic attacks that exploit mathematical properties of the algorithms. There is another class of attacks that use physical measurements conducted during algorithm execution to gather information about quantities such as keys. Such attacks exploit a combination of intrinsic algorithm characteristics and implementation-dependent features. Examples of such attacks are timing attacks and power analysis. Timing attacks are described in. The basic idea behind power analysis [KOCH98,BIHA00] is the observation that the power consumed by a smart card at any particular time during the cryptographic operation is related to the instruction being executed and to the data being processed. For example, multiplication consumes more power than addition, and writing 1s consumes more power than writing 0s.

Encryption versus decryption:

This criterion deals with several issues related to considerations of both encryption and decryption. If the encryption and decryption algorithms differ, then extra space is needed for the decryption. Also, whether the two algorithms are the same or not, there may be timing differences between encryption and decryption.

Key agility:

Key agility refers to the ability to change keys quickly and with a minimum of resources. This includes both subkey computation and the ability to switch between different ongoing security associations when subkeys may already be available.

Other versatility and flexibility:

[NECH00] indicates two areas that fall into this category. Parameter flexibility includes ease of support for other key and block sizes and ease of increasing the number of rounds in order to cope with newly discovered attacks. Implementation flexibility refers to the possibility of optimizing cipher elements for particular environments.

Potential for instruction-level parallelism:

This criterion refers to the ability to exploit ILP features in current and future processors

The AES cipher

Like DES, AES is a symmetric block cipher. This means that it uses the same key for both encryption and decryption. However, AES is quite different from DES in a number of ways. The algorithm Rijndael allows for a variety of block and key sizes and not just the 64 and 56 bits of DES' block and key size. However, the AES standard states that the algorithm can only accept a block size of 128 bits and a choice of three keys - 128, 192 , 256 bits. Depending on which version is used, the name of the standard is modified to AES-128, AES-192 or AES- 256 respectively. As well as these differences AES differs from DES in that it is not a feistel structure. Recall that in a feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. In this case the entire data block is processed in parallel during each round using substitutions and permutations.

A number of AES parameters depend on the key length. For example, if the key size used is 128 then the number of rounds is 10 whereas it is 12 and 14 for 192 and 256 bits respectively. At present the most common key size likely to be used is the 128 bit key. This description of the AES algorithm therefore describes this particular implementation.

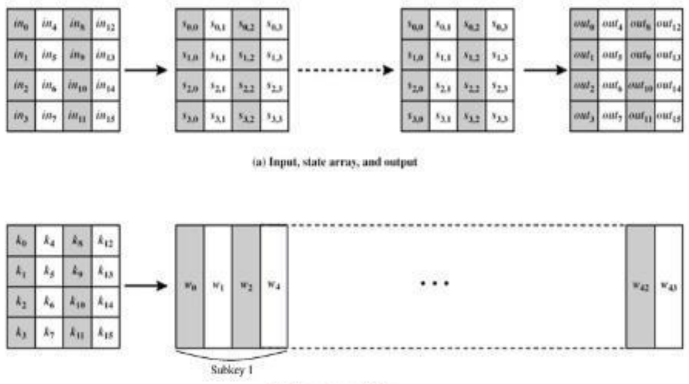
Rijndael was designed to have the following characteristics:

• Resistance against all known attacks.

•Speed and code compactness on a wide range of platforms.

•Design Simplicity.

The overall structure of AES is given below. The input is a single 128 bit block both for decryption and encryption and is known as the in matrix. This block is copied into a state array which is modified at each stage of the algorithm and then copied to an output matrix. Both the plaintext and key are depicted as a 128 bit square matrix of bytes. This key is then expanded into an array of key schedule words(the w matrix). It must be noted that the ordering of bytes within the in matrix is by column. The same applies to the w matrix.



(a) Input, state array, and output

(b) Key and expanded key

Before delving into details, we can make several comments about the overall AES structure.

1. AES structure is not a Feistel Structure

2. The key that is provided as input is expanded into an array of forty-four 32-bit words, w[i]. Four distinct words (128 bits) serve as a round key for each round.

3. Four different stages are used, one of permutation and three of substitution:
• Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block
• ShiftRows: A simple permutation
• MixColumns: A substitution that makes use of arithmetic over
• AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key

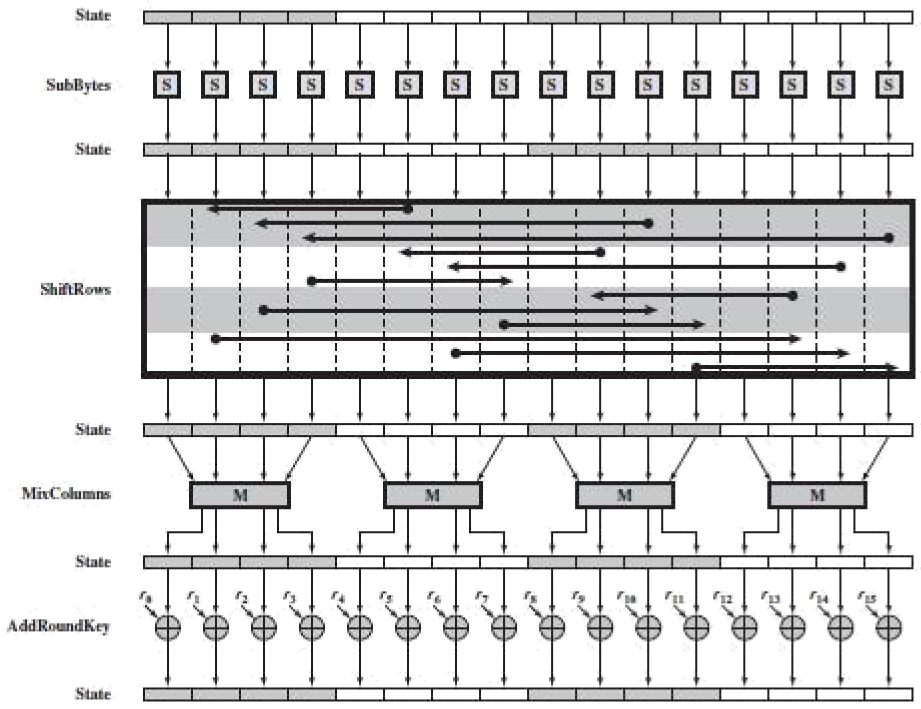
4. The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages.

5. Only the AddRoundKey stage makes use of the key.

6. The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable.

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

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


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

identical to the encryption algorithm. This is a consequence of the particular structure of AES.

9. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure 5.3 lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), State is the same for both encryption and
decryption.

10. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

Inner Workings of a Round

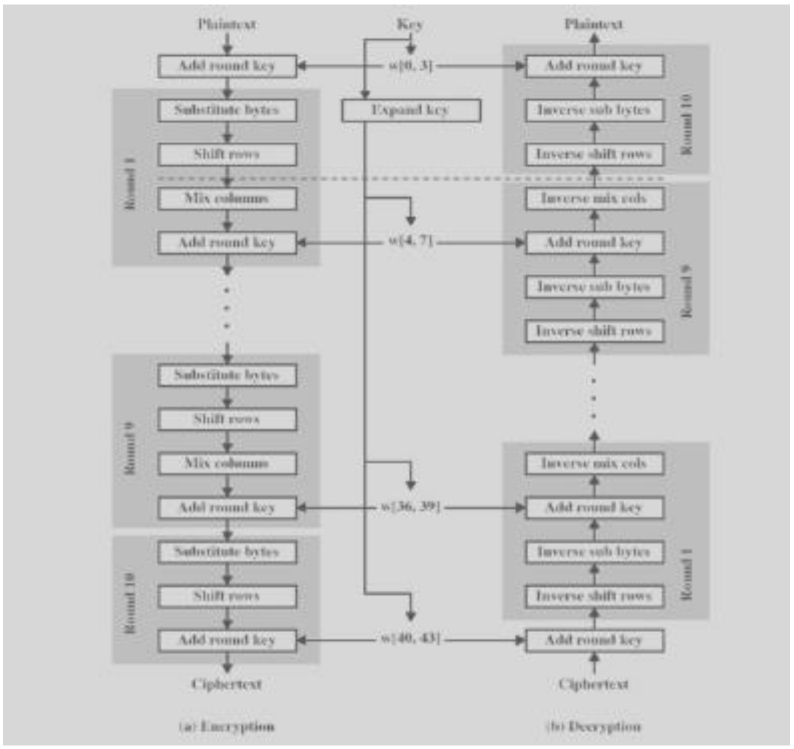
The algorithm begins with an Add round key stage followed by 9 rounds of four stages and a tenth round of three stages. This applies for both encryption and decryption with the exception that each stage of a round the decryption algorithm is the inverse of it’s counterpart in the encryption algorithm. The four stages are as follows:

1. Substitute bytes
2. Shift rows
3. Mix Columns
4. Add Round Key

The tenth round simply leaves out the Mix Columns stage. The first nine rounds of the decryption algorithm consist of the following:

1. Inverse Shift rows
2. Inverse Substitute bytes
3. Inverse Add Round Key
4. Inverse Mix Columns

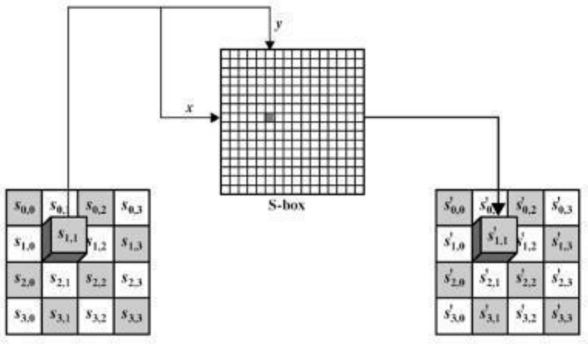
Again, the tenth round simply leaves out the Inverse Mix Columns stage. Each of these stages will now be considered in more detail.



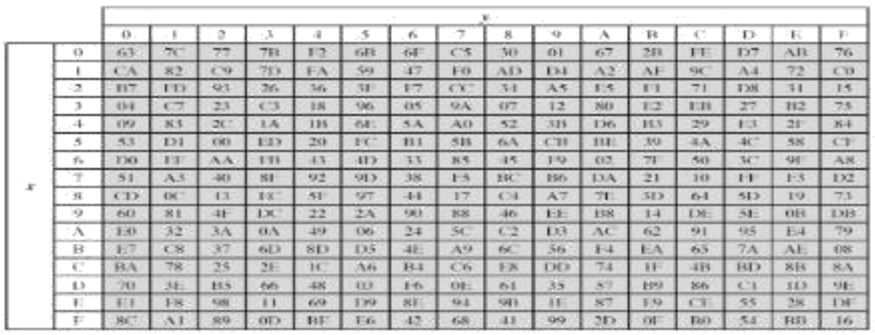
Overall structure of the AES algorithm.

AES transformation functions
Substitute Bytes
This stage (known as SubBytes) is simply a table lookup using a16×16 matrix of byte values called an s-box. This matrix consists of all the possible combinations of an 8 bit sequence (2^8= 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. We will not be too concerned here how the s-boxes are made up and can simply take them as table lookups.

Again the matrix that gets operated upon throughout the encryption is known as state. We will be concerned with how this matrix is effected in each round. For this particular

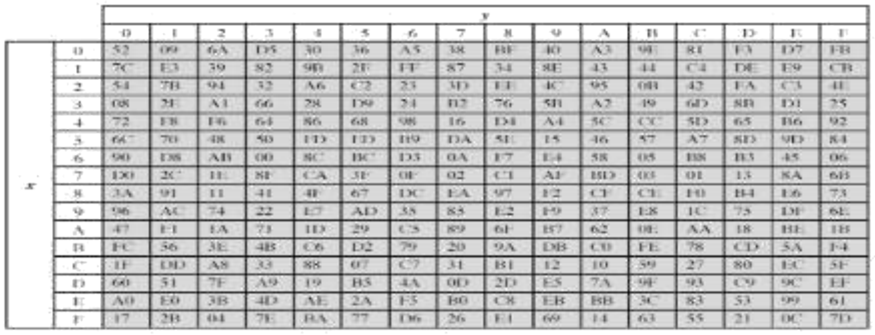

y
x
S-box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

(a) S-box



		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
x	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(b) Inverse S-box



		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
x	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

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.

The Inverse substitute byte transformation (known as InvSubBytes) makes use of an inverse s-box. In this case what is desired is to select the value {2A} and get the value {95}. Shows the two s-boxes and it can be verified that this is in fact the case.

The s-box is designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design that has a low correlation between input bits and output bits, and the property that the output cannot be described as a simple mathematical function of the input. In addition, the s-box has no fixed points (s-box (a) = a) and no opposite fixed points (s-box(a) =a) where a is the bitwise compliment of a. The s-box must be invertible if decryption is to be possible (Is-box[s-ox(a)]=a) however it should not be its self inverse i.e. s-box (a)=Is-box(a)

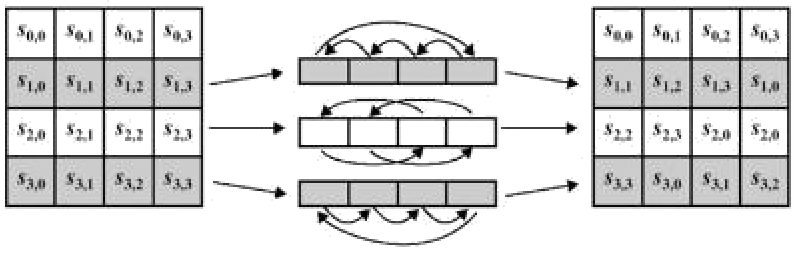
Shift Row Transformation

• 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.



$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,0}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

The Inverse Shift Rows transformation (known as InvShiftRows) performs these circular shifts in the opposite direction for each of the last three rows (the first row was unaltered to begin with).

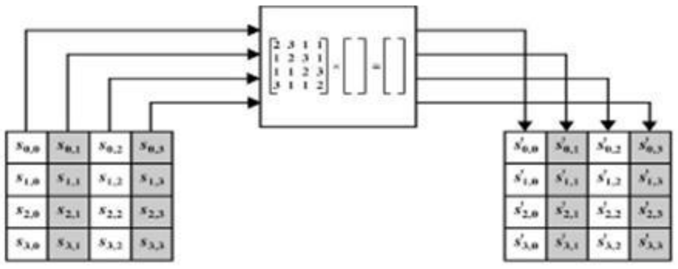
This operation may not appear to do much but if you think about how the bytes are ordered within state then it can be seen to have far more of an impact. Remember that state is treated as an array of four byte columns, i.e. the first column actually represents bytes 1 ,2 ,3 and 4. A one byte shift is herefore a linear distance of four bytes. The transformation also ensures that the four bytes of one column are spread out to four different columns.

Mix Column Transformation

This stage (known as MixColumn) is basically a substitution but it makes use of arithmetic of GF(2^8). 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.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

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 (2^8) . The MixColumns transformation of a single column j($0 \le j \le 3$)of state can be expressed as



$$\begin{aligned}
s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\
s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j} \\
s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \\
s'_{3,j} &= (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j})
\end{aligned}$$

The following is an example of MixColumns:

$$\begin{array}{|c|c|c|c|}\hline 87 & F2 & 4D & 97 \\ \hline 6E & 4C & 90 & EC \\ \hline 46 & E7 & 4A & C3 \\ \hline A6 & 8C & D8 & 95 \\ \hline\end{array} \quad \rightarrow \quad \begin{array}{|c|c|c|c|}\hline 47 & 40 & A3 & 4C \\ \hline 37 & D4 & 70 & 9F \\ \hline 94 & E4 & 3A & 42 \\ \hline ED & A5 & A6 & BC \\ \hline\end{array}$$

In particular, multiplication of a value by (i.e., by {02}) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the original value (prior to the shift) is 1. In multiplication left most bit value is 0 only perform the left shift operation.

$$\begin{array}{llllll}
(\{02\} \cdot \{87\}) & \oplus (\{03\} \cdot \{6E\}) & \oplus \{46\} & \oplus \{A6\} & = \{47\} \\
\{87\} & \oplus (\{02\} \cdot \{6E\}) & \oplus (\{03\} \cdot \{46\}) & \oplus \{A6\} & = \{37\} \\
\{87\} & \oplus \{6E\} & \oplus (\{02\} \cdot \{46\}) & \oplus (\{03\} \cdot \{A6\}) & = \{94\} \\
(\{03\} \cdot \{87\}) & \oplus \{6E\} & \oplus \{46\} & \oplus (\{02\} \cdot \{A6\}) & = \{ED\}
\end{array}$$

For the first equation, we have $\{02\} \cdot \{87\} = (0000\ 1110) \oplus (0001\ 1011) = (0001\ 0101)$ and $\{03\} \cdot \{6E\} = \{6E\} \oplus (\{02\} \cdot \{6E\}) = (0110\ 1110) \oplus (1101\ 1100) = (1011\ 0010)$. Then,

$$\begin{array}{lcl}
\{02\} \cdot \{87\} & = & 0001\ 0101 \\
\{03\} \cdot \{6E\} & = & 1011\ 0010 \\
\{46\} & = & 0100\ 0110 \\
\{A6\} & = & \underline{1010\ 0110} \\
 & & 0100\ 0111 = \{47\}
\end{array}$$

The other equations can be similarly verified.

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 .

$$\begin{array}{|c|c|c|c|}\hline 47 & 40 & A3 & 4C \\ \hline 37 & D4 & 70 & 9F \\ \hline 94 & E4 & 3A & 42 \\ \hline ED & A5 & A6 & BC \\ \hline\end{array} \oplus \begin{array}{|c|c|c|c|}\hline AC & 19 & 28 & 57 \\ \hline 77 & FA & D1 & 5C \\ \hline 66 & DC & 29 & 00 \\ \hline F3 & 21 & 41 & 6A \\ \hline\end{array} = \begin{array}{|c|c|c|c|}\hline EB & 59 & 8B & 1B \\ \hline 40 & 2E & A1 & C3 \\ \hline F2 & 38 & 13 & 42 \\ \hline 1E & 84 & E7 & D6 \\ \hline\end{array}$$

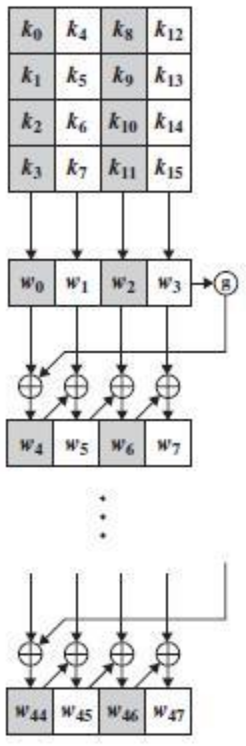
The first matrix is **State**, and the second matrix is the round key.

The **inverse add round key transformation** is identical to the forward add round key transformation, because the XOR operation is its own inverse

AES key Expansion:

Key Expansion Algorithm

The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes).This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher



1. RotWord performs a one-byte circular left shift on a word.This means that an input word [B_0, B_1, B_2, B_3] is transformed into [B_1, B_2, B_3, B_0].
2. SubWord performs a byte substitution on each byte of its input word, using the S-box .

3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j].

The round constant is a word in which the three rightmost bytes are always 0. Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word.The round constant is different for each round and is defined as Rcon[j] = (RC[j], 0, 0, 0) with RC[1] = 1 RC[j] = 2 *RC[j-1] and with multiplication defined over the field GF(2_8).The values of RC[j] in hexadecimal are

Multiple encryption and Triple DES:

- Multiple encryption is a technique in which an encryption algorithm is used multiple times. In the first instance, plaintext is converted to ciphertext using the encryption algorithm. This ciphertext is then used as input and the algorithm is applied again. This process may be repeated through any number of stages.
- Triple DES makes use of three stages of the DES algorithm, using a total of two or three distinct keys.

Double DES

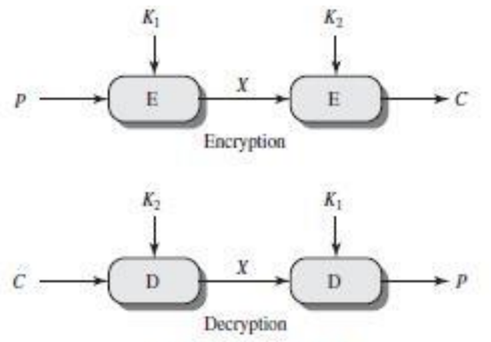
- The simplest form of multiple encryption has two encryption stages and two keys. Given a plaintext *P* and two encryption keys k_1 and k_2 , ciphertext C is generated as

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

Decryption requires that the keys be applied in reverse order:

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

For DES, this scheme apparently involves a key length of 56 * 2 = 112 bits, resulting in a dramatic increase in cryptographic strength. But we need to examine the algorithm more closely.



(a) Double encryption

MEET-IN-THE-MIDDLE ATTACK

The algorithm, known as a **meet-in-the-middle attack**, was first described in [DIFF77]. It is based on the observation that, if we have

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

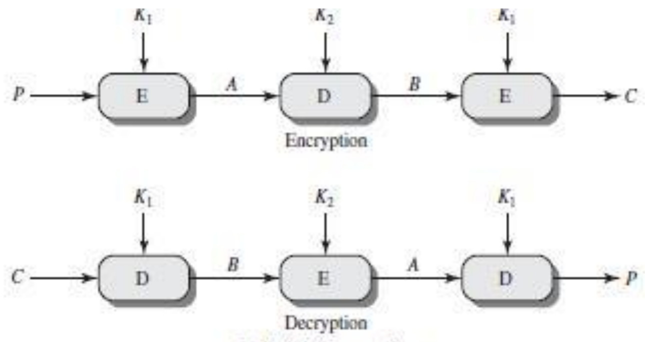
then (see Figure)

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

Given a known pair, (P, C) , the attack proceeds as follows. First, encrypt P for all possible 2^{56} values of k_1. Store these results in a table and then sort the table by the values of x. Next, decrypt C using all 2^{56} possible values of k_2 . As each decryption is produced, check the result against the table for a match. If a match occurs, then test the two resulting keys against a new known plaintext–ciphertext pair. If the two keys produce the correct ciphertext, accept them as the correct keys.

Triple DES with two keys

triple encryption method that uses only two keys [TUCH79]. The function follows an encrypt-decrypt-encrypt (EDE) sequence.



(b) Triple encryption

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

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

3DES with two keys is a relatively popular alternative to DES.

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

It is worth looking at several proposed attacks on 3DES that

A known-plaintext attack:

1. Obtain pairs n(p,c). This is the known plaintext. Place these in a table (Table 1)

sorted on the values of (Figure 6.2b).

2. Pick an arbitrary value a for A, and create a second table (Figure 6.2c) with entries defined in the following fashion. For each of the 2^{56} possible keys , k_1=i calculate the plaintext value P_i that produces :

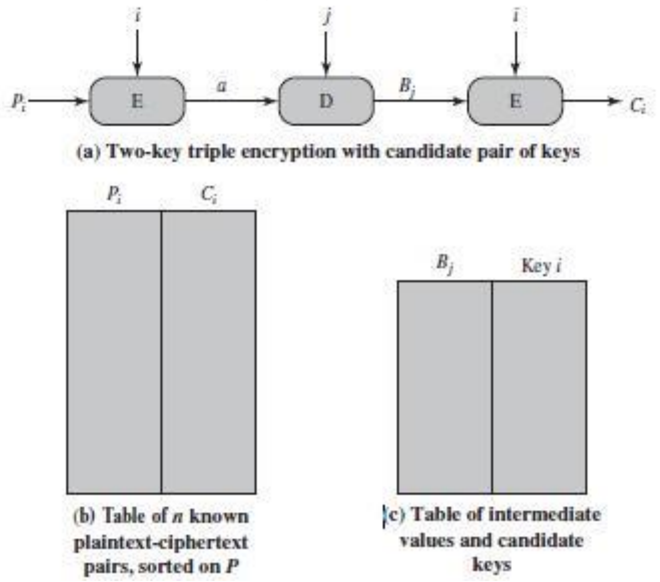
   $Pi = D(i, a)$

For each P_i that matches an entry in Table 1, create an entry in Table 2 consisting of

the k_1 value and the value of B that is produced for the (P,C) pair from Table 1, assuming that value of k_1 :

$$B = D(i, C)$$

At the end of this step, sort Table 2 on the values of *B*.



(a) Two-key triple encryption with candidate pair of keys

(b) Table of *n* known plaintext-ciphertext pairs, sorted on *P*

(c) Table of intermediate values and candidate keys

3. We now have a number of candidate values of $k_1$in Table 2 and are in a position to search for a value of k_2 . For each of the 2^{56} possible keys k_2=j, calculate the second intermediate value for our chosen value of a:

   $$B_j = D(j, a)$$

   At each step, look up B_j in Table 2. If there is a match, then the corresponding key i from Table 2 plus this value of *j* are candidate values for the unknown keys (k_1,k_2). Why? Because we have found a pair of keys (I,j) that produce a known (P,C) pair (Figure 6.2a).

4. Test each candidate pair of keys(i,j) on a few other plaintext–ciphertext pairs. If a pair of keys produces the desired ciphertext, the task is complete. If no pair succeeds, repeat from step 1 with a new value of a .

Triple DES with Three Keys

Although the attacks just described appear impractical, anyone using two-key 3DES may feel some concern.Thus, many researchers now feel that three-key 3DES is the preferred alternative (e.g., [KALI96a]).Three-key 3DES has an effective key length of 168 bits and is defined as

$$C = \mathrm{E}(K_3, \mathrm{D}(K_2, \mathrm{E}(K_1, P)))$$

Backward compatibility with DES is provided by putting $k_3=k_2$ or $k_1=k_2$.

BLOCK CIPHER MODES OF OPERATION

Four DES modes of operations have been defined (FIPS 81, http://www.itl.nist.gov/fipspubs/fip81.htm):

Mode	Description	Typical application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key	Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of the plaintext and the preceding 64 bits of the ciphertext	General-purpose block-oriented transmission Authentication
Cipher Feedback (CFB)	Input is processed s bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext	General-purpose stream-oriented transmission Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding DES output	Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block	General-purpose block-oriented transmission Useful for high-speed requirements

ELECTRONIC CODEBOOK MODE

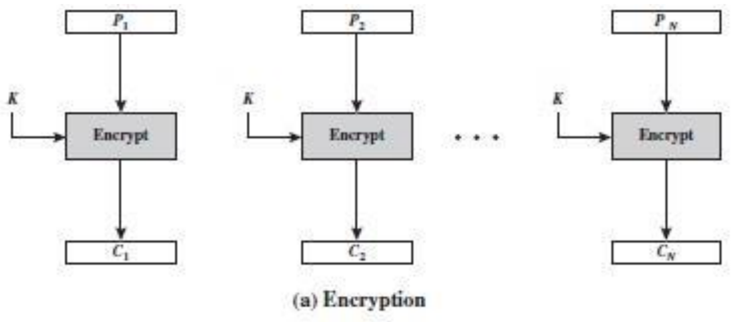
The simplest mode is the **electronic codebook** (**ECB**) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key. The term *codebook* is used because, for a given key, there is a unique ciphertext for every -bit block of plaintext.

For a message longer than bits, the procedure is simply to break the message into b-bit blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key the plaintext (padded as necessary) consists of a sequence of *b*-bit blocks, *P*1, *P*2, …, *PN*; the corresponding sequence of ciphertext blocks is *C*1, *C*2, .., *CN*.

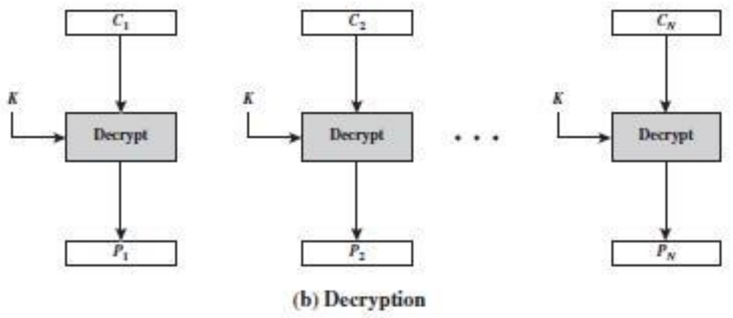
The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES or AES key securely, ECB is the appropriate mode to use.

The most significant characteristic of ECB is that if the same b-bit block of plaintext appears more than once in the message, it always produces the same ciphertext.

For lengthy messages, the ECB mode may not be secure.



(a) Encryption



(b) Decryption

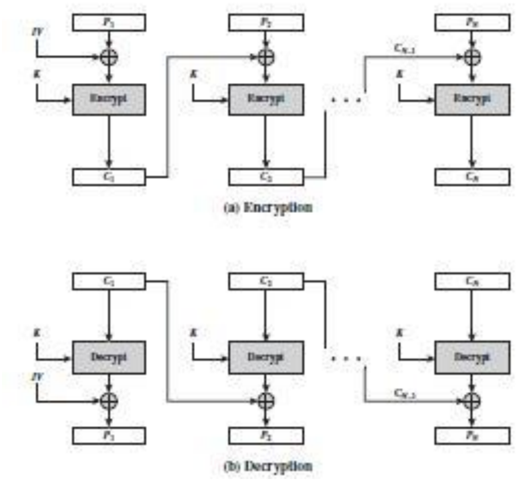
CIPHER BLOCK CHAINING MODE

To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks.A simple way to satisfy this requirement is the **cipher block chaining** (**CBC**) mode.

The input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. In effect, we have chained together the processing of the sequence of plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of bits are not exposed.

For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block. To see that this works, we can write

$$C_j = \mathrm{E}(K, [C_{j-1} \_ P_j])$$



$\mathrm{D}(K, C_j) = \mathrm{D}(K, E(K, [C_{j-1} \_ P_j]))$

$\mathrm{D}(K, C_j) = C_{j-1} \_ P_j$

$C_{j-1} \_ \mathrm{D}(K, C_j) = C_{j-1} \_ C_{j-1} \_ P_j = P_j$

To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV is a data block that is that same size as the cipher block.

The IV must be known to both the sender and receiver but be unpredictable by a third party. In particular, for any given plaintext, it must not be possible to predict the IV that will be associated to the plaintext in advance of the generation of the IV. For maximum security, the IV should be protected against unauthorized changes. This could be done by sending the IV using ECB encryption. One reason for protecting the IV is as follows: If an opponent is able to fool the receiver into using a different value for IV, then the opponent is able to invert selected bits in the first block of plaintext.To see this, consider

$C_1 = \mathrm{E}(K, [\mathrm{IV} \_ P_1])$

$P_1 = \mathrm{IV} \_ \mathrm{D}(K, C_1)$

CIPHER FEEDBACK MODE

DES is a block cipher, but it may be used as a stream cipher if to use the Cipher Feedback Mode (CFB) or the Output Feedback Mode (OFB). A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.

CFB scheme follows:

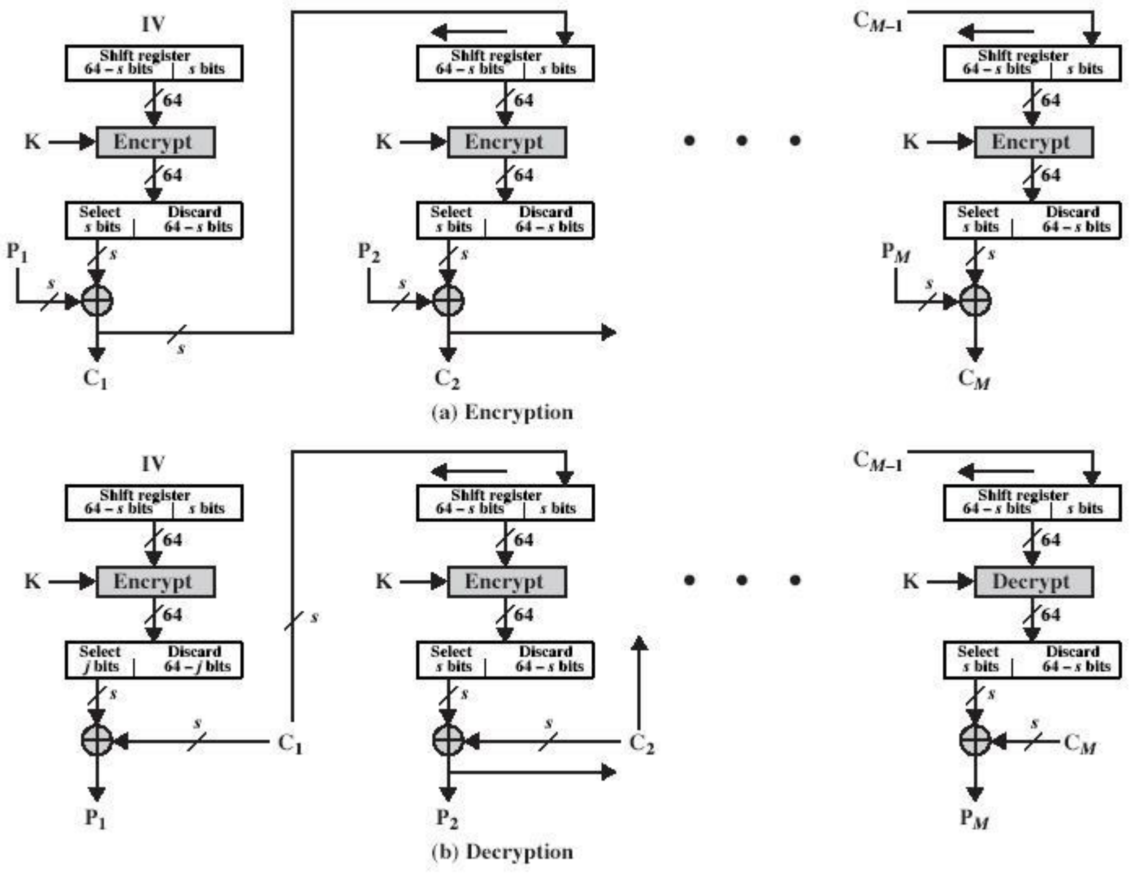


Figure 3.13 *s*-bit Cipher Feedback (CFB) Mode

In Fig. 3.13, it is assumed that the unit of transmission is s bits; usually, 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. In this case, rather than units of 64 bits, the plaintext is divided into segments of s bits.

Consider encryption. The input to the encryption function is a 64-bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant) s bits of the output of the encryption function are XORed with the first segment of plaintext P1 to produce the first unit of ciphertext C1, which is then transmitted. In addition, the contents of the shift register are shifted left by s bits and C1 is placed in the rightmost (least significant) s bits of the shift register. This process continues until all plaintext units have been encrypted.

For decryption, the same scheme is used except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit.

OUTPUT FEEDBACK MODE

The Output Feedback Mode (OFB) is similar in structure to that of CFB:

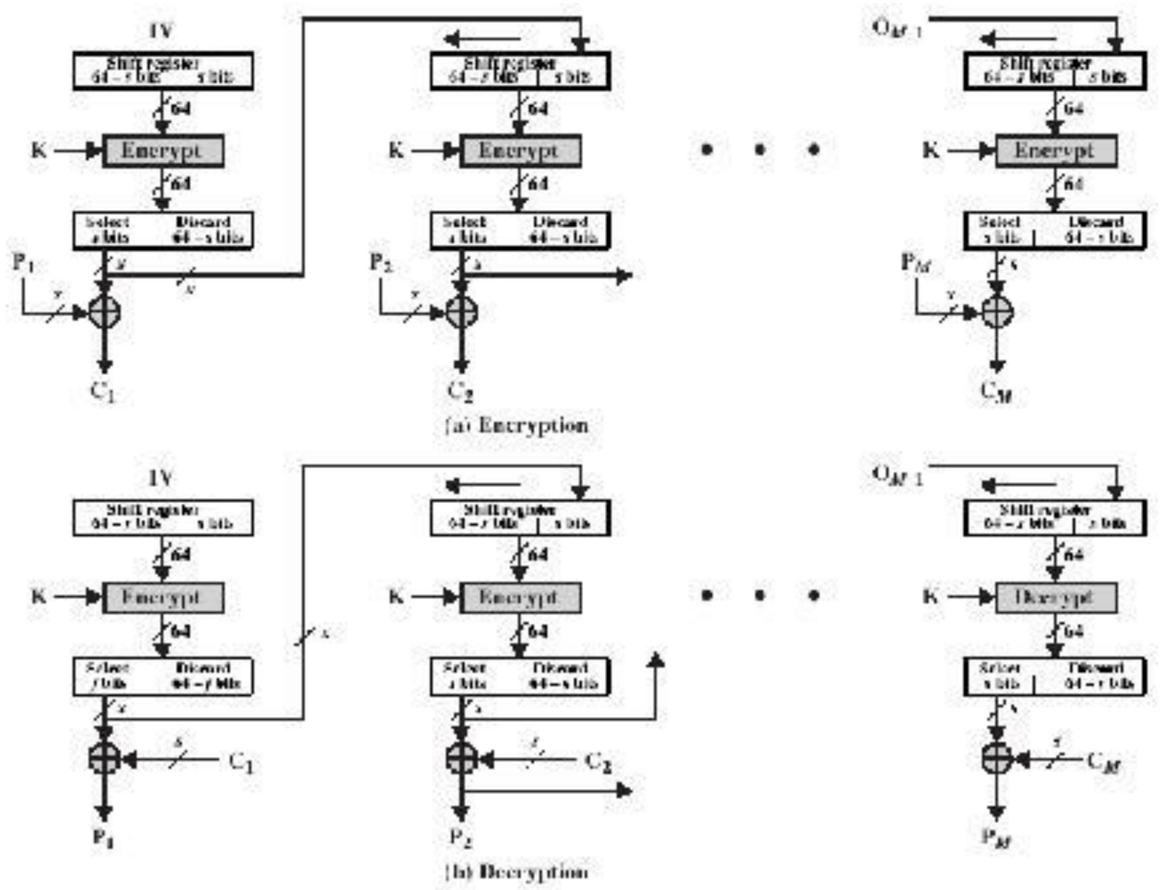


Figure 3.14 s-bit Output Feedback (OFB) Mode

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.

COUNTER MODE

A counter, equal to the plaintext block size is used. The only requirement stated in SP 800-38A NIST Special Publication 800 -38 A, 2001 Edition, Morris Dworkin, Recommendations for Block Cipher Modes of Operation) is that the counter value must be different for each plaintext block that is encrypted. This mode is with applications to ATM (asynchronous transfer mode) and IPsec (IP security) nowadays, but it was proposed in 1979.

Counter Mode works as follows:

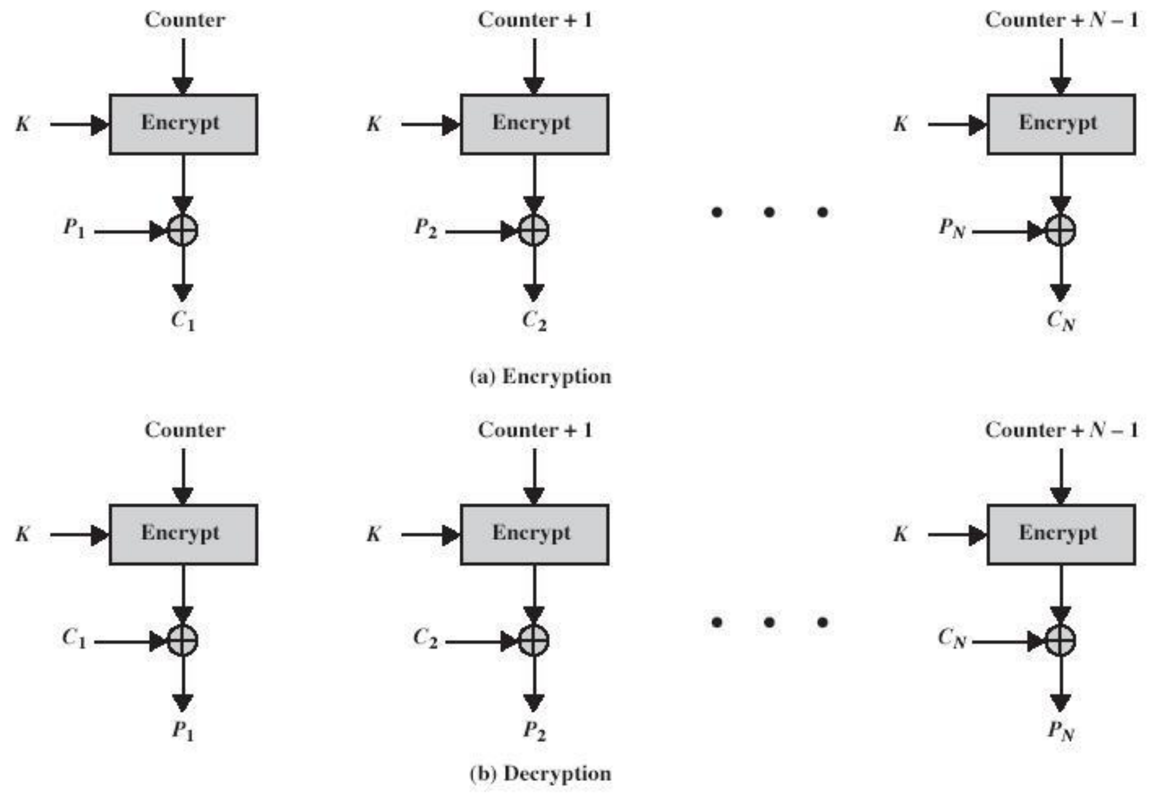


Figure 3.15 Counter (CTR) Mode

Addition is made modulo 2^b, where b is a block size. CTR mode is effective because blocks may be processed in parallel; encryption of keys may be made in advance, and only XOR will be made on-line; only necessary blocks may be decrypted; provides not less security than chaining modes but significantly simpler.