Block Cipher Principles

Stream Cipher and Block Cipher

- A **stream cipher** is one that encrypts a data stream one bit or one byte at a time. Example of stream cipher are the autokeyes vigenere cipher and vernam cipher.
- A **Block Cipher** is one in which a block of plaintext is treated as a whole and used to produce a cipher text block of equal length. Example of block cipher is DES.

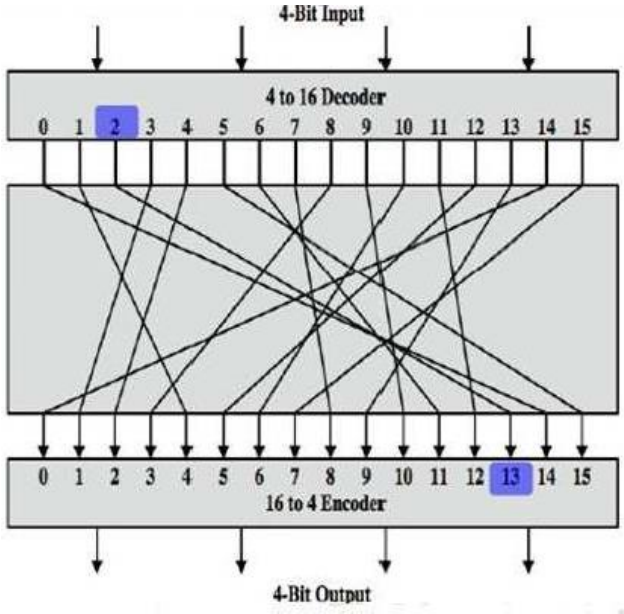
Motivation for the Feistel Cipher Structure

- 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 plain text blocks and for the encryption to be reversible each must produce unique ciphertext block.
- Reversible encryption is also called as **singular encryption**. For example singular and non singular transformation 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

- If we limit ourselves to reversible mapping the number of different transformation is $2^n!$.
- Figure below illustrates the logic of a general substitution cipher for n=4



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

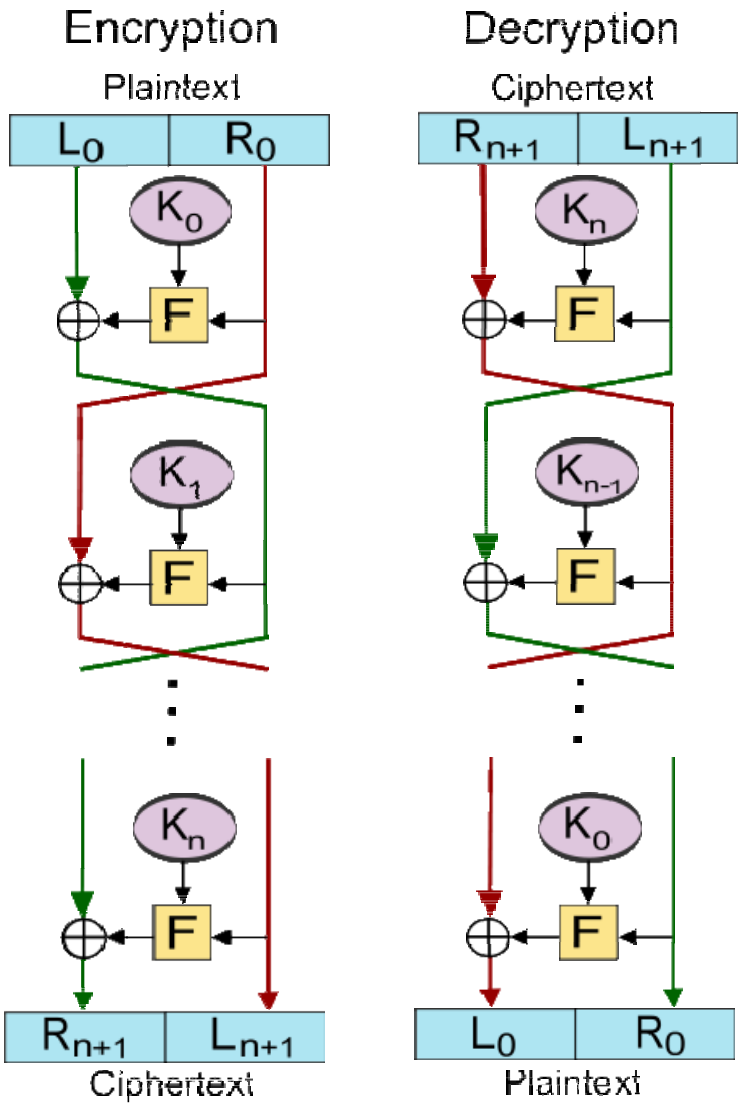
Ciphertext	Plaintext
0000	1110
0001	0011
0010	0100
0011	1000
0100	0001
0101	1100
0110	1010
0111	1111
1000	0111
1001	1101
1010	1001
1011	0110
1100	1011
1101	0010
1110	0000
1111	0101

- A 4-bit input produce one of 16 possible input states, which is mapped by substitution cipher into one of unique 16 possible output states, each of which is represented by 4-bit ciphertext.
- The encryption and decryption mapping can be defined by tabulation, as shown in table. 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.

- But there are practical problem with ideal block cipher is if we use small block size such as n=4 then it is vulnerable to statistical analysis of the plain text.
- If n is sufficiently large and an arbitrary reversible substitution between plaintext and ciphertext is allowed then statistical analysis is infeasible.
- Ideal block cipher is not practical for large block size according implementation and performance point of view.
- For such transformation mapping itself is a key and we require $nX2^n$ bits for n bit ideal block cipher which is not practical.
- In considering these difficulties, Feistel points out that what is needed is an approximation to the ideal cipher system for large n, built up out of components that are easily realizable.

The Feistel Cipher

- Feistel cipher is based on the idea that instead of using Ideal block cipher which degrades performance, a "substitution-permutation network" can be used.



Feistel Cipher Encryption

- The inputs to the encryption algorithm are a plaintext block of length b bits and a key K.

- The plaintext block is divided into two halves.
- The two halves of the data pass through rounds of processing and then combine to produce the ciphertext block.
- Each round has as inputs and derived from the previous round, as well as a subkey derived from the overall K.
- Any number of rounds could be implemented and all rounds have the same structure.
- A **substitution** is performed on the left half of the data. This is done by applying a round function F.
- The Round Function F: F takes right-half block of previous round and a subkey as input.
- The output of the function is XORed with the left half of the data.
- Left and right halves are then swapped.

Feistel Cipher Decryption

- The process of decryption with a Feistel cipher is same as the encryption process.
- The ciphertext is input to the algorithm and the subkeys are used in reverse order. That is, subkey of the last round in encryption is used in the first round in decryption, second last in the second round, and so on.

The exact realization of a Feistel network depends on the choice of the following parameters:

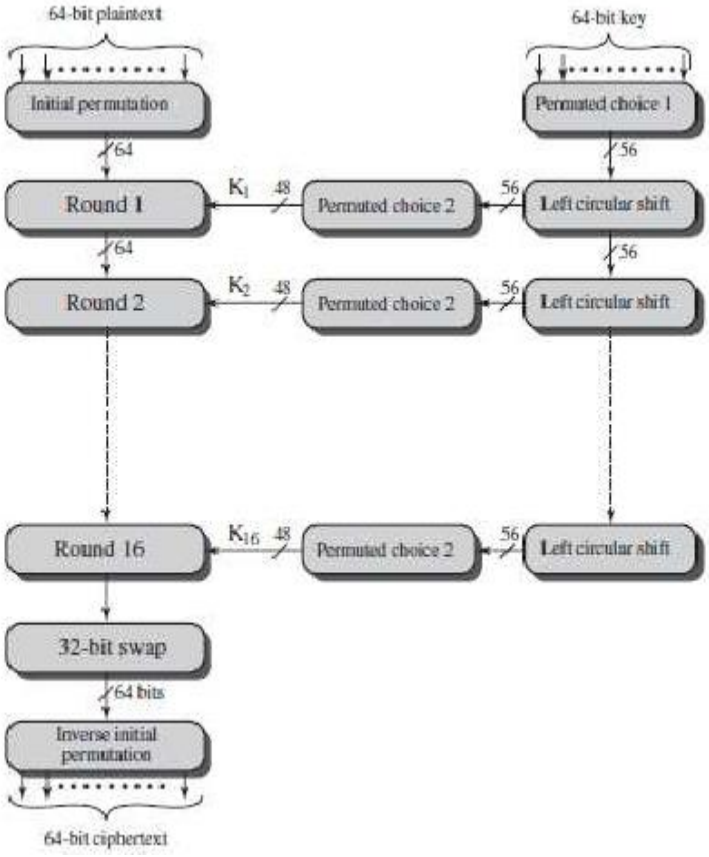
- **Block size**: Larger block sizes mean greater security but reduced encryption/decryption speed for a given algorithm. Traditionally, a block size of 64 bits is used which gives enough security without greatly affecting the speed.
- **Key size**: Larger key size means greater security but may decrease 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.
- **Sub key generation algorithm**: Greater complexity in this algorithm leads 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 implementation (as software). 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 crypt analyze, there is great benefit in making the algorithm easy to analyze. That is, if The algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a high level of assurance as to its strength.

The Data Encryption Standard

- SDES encrypts 64-bit blocks using a 56-bit key and produces a 64-bit ciphertext.
- Same steps, with the same key, are used to reverse the encryption with the order of the keys reversed.
- The DES is widely used.

DES Encryption

- The DES encryption is shown in the figure below:



- Encryption function has two inputs: the plaintext to be encrypted and the key.
- The processing of the plaintext proceeds in three phases.
  - The 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.
  - The permuted output is then passed through sixteen rounds of the same function, which involves both permutation and substitution functions. The left and right halves from the last round are swapped to produce preoutput.
  - The preoutput is passed through a permutation that is the inverse of the initial permutation function, to produce the 64-bit cipher text.
- The right-hand portion of the figure shows the way in which the 56-bit key is used.
  - Initially, the key is passed through a permutation function.
  - Then, a sub key (k_i) is produces for each of the sixteen rounds by the combination of a left circular shift and a permutation.
  - The permutation function is the same for each round, but a different sub key is produced because of the repeated shifts of the key bits.

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

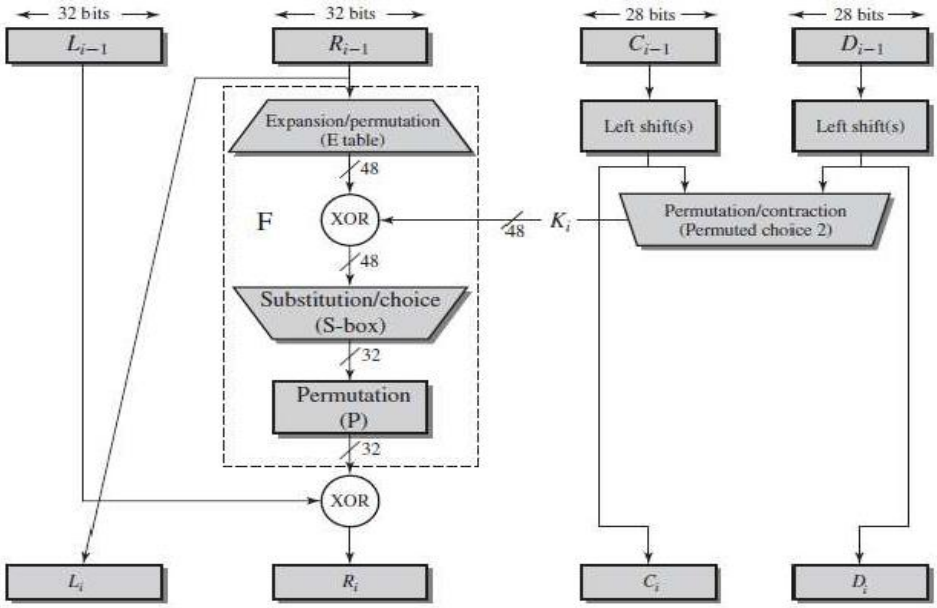
IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

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

- 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 input bit in the output.
- Inverse permutation table nullifies the effect of initial permutation.

Details of Single Round

- Figure shows the internal structure of a single round.



- The left and right halves are treated as separate 32-bit quantities, labeled L (left) and R (right).
- The overall processing at each round can be summarized as:

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

Expansion (E)

- The 32-bit input is first expanded to 48 bits.
  - Bits of input are split into groups of 4 bits.

- Each group is written as groups of 6 bits by taking the outer bits from the wo adjacent groups. For example

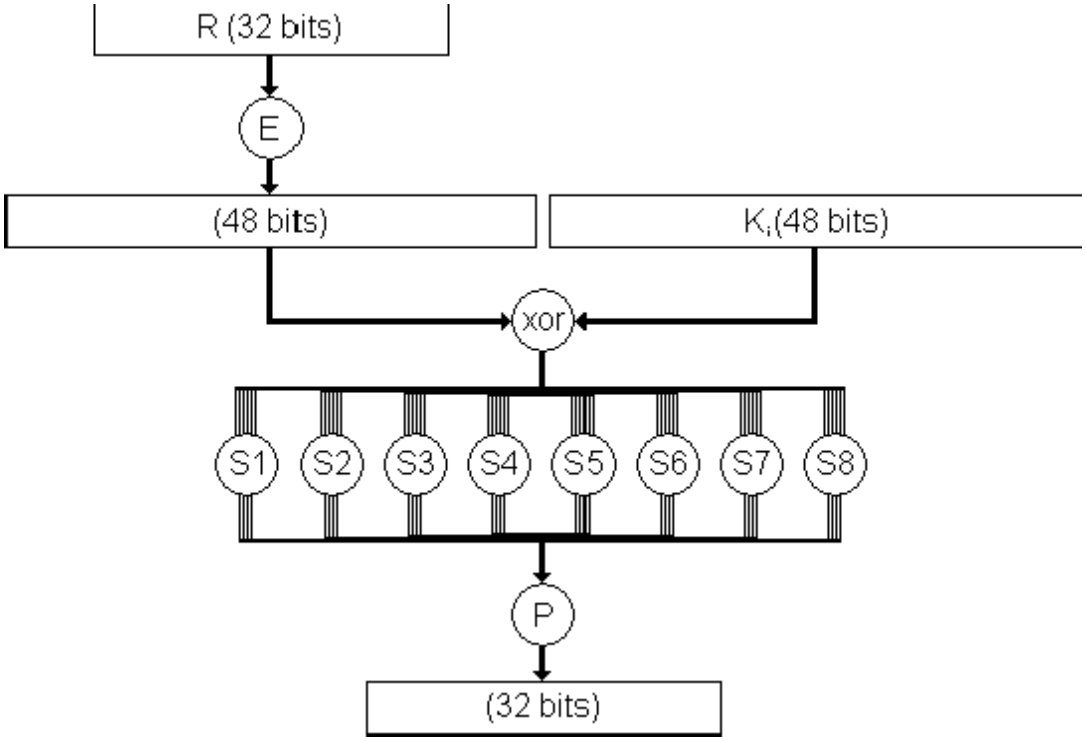
  ... efgh ijkl mnop ... is expanded to
  ... defghi hijklm lnnopq ...

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

- The resulting 48 bits are XORed with K_i.

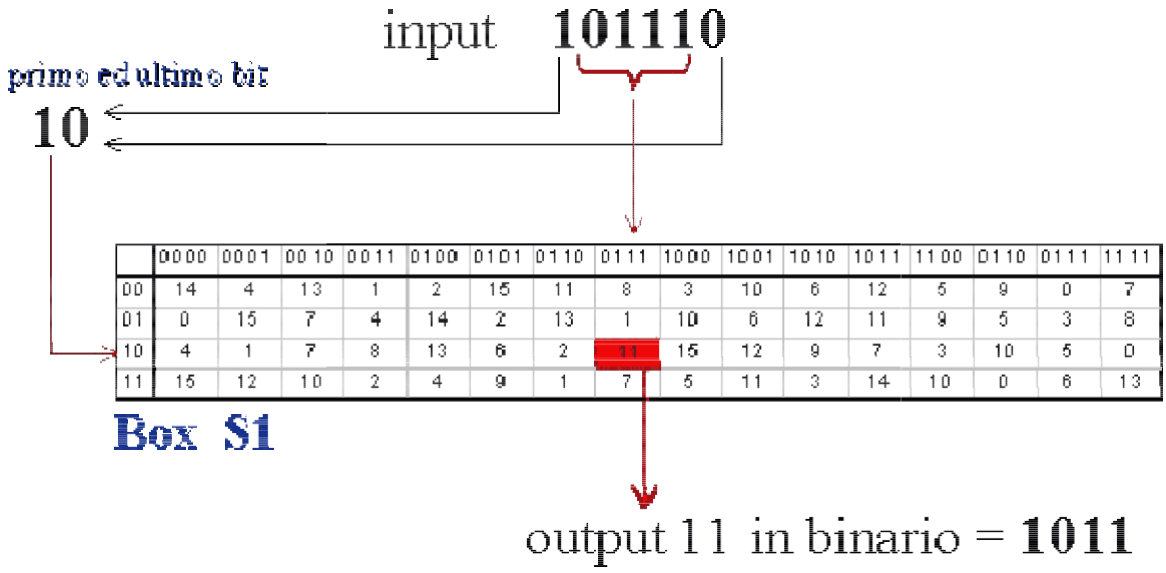
Substitution (S-Box)

- This 48-bit result is input to S-Boxes that perform a substitution on input and produces a 32-bit output.
- It is easy to understand S-Box by following figure.



- DES consists of a set of eight S-boxes.
- Each S-Box takes 6 bits as input and produces 4 bits as output.
- The first and last bits of the input to box form a 2-bit binary number which gives the binary value of row number.
- The middle four bits select one of the sixteen columns.

- The decimal value in the cell selected by the row and column is then converted to its 4-bit binary number to produce the output.
- For example, in S1, for input 101110, the row is 10 (row 2) and the column is 0111 (column 7).The value in row 2, column 7 is 11, so the output is 1011.



	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	0110	0111	1111
00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
10	4	1	7	8	13	6	2	11	15	12	9	7	3	10	5	0
11	15	12	10	2	4	9	1	7	5	11	3	14	10	0	6	13

Permutation (P)

- The result is again permuted using a permutation table.

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Key Generation

- A 64-bit key is used as input to the algorithm while only 56 bits are actually used. Every eighth bit is ignored. Sub-keys at each round are generated as given below:
  - The key is first permuted using a table named Permuted Choice One.
  - The resulting 56-bit key is divided into two 28-bit quantities, C_0 and D_0. At each round, C_{i-1} and D_{i-1} are separately subjected to a circular left shift of 1 or 2 bits, as governed by a table.
  - These shifted values are forwarded to the next round. They are also input to a permutation table-Permuted Choice Two.
  - The table produces a 48-bit output that serves as the round key k_i.

(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

DES Decryption

- Decryption in DES is same as encryption, except that the sub keys are used in reverse order.

Strength of DES

The Use of 56-Bit Keys

- DES has been developed from LUCIFER which used 128-bit keys.
- As a result, DES with only 56-bit key-length is considered insecure and devices have been proposed time and again showing that DES is no longer secure.

The Nature of the DES

- The only non-linear part of DES is the S-Boxes, design of which was not made public.
- If someone is able to find weakness in S-Box, then attack on DES is possible.
- Characteristics of the algorithm can be exploited as the algorithm is based on linear functions.

Algorithm Timing Attacks

- In this type of attack, the attacker exploits the fact that any algorithm takes different amount of time for different data.

A DES Example

- Let see example of DES and consider some of its implications. Although you are not expected to duplicate the example by hand, you will find it informative to study the hex patterns that occure from one step to the next.

Plaintext:	02468aceeca86420
Key:	0f1571c947d9e859
Ciphertext:	Da02ce3a89ecac3b

- **Result:** Above table shows plain text, key and cipher text when we apply all the steps of DES we will get cipher text as shown.

- **The Avalanche Effect:** A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in cipher text.
- In particular, a change in one bit of plaintext or one bit of the key should produce a change in many bits of the ciphertext. This is referred to as the avalanche effect.
- In DES 1 bit change in input will affect nearly 32 bit of output after all rounds.

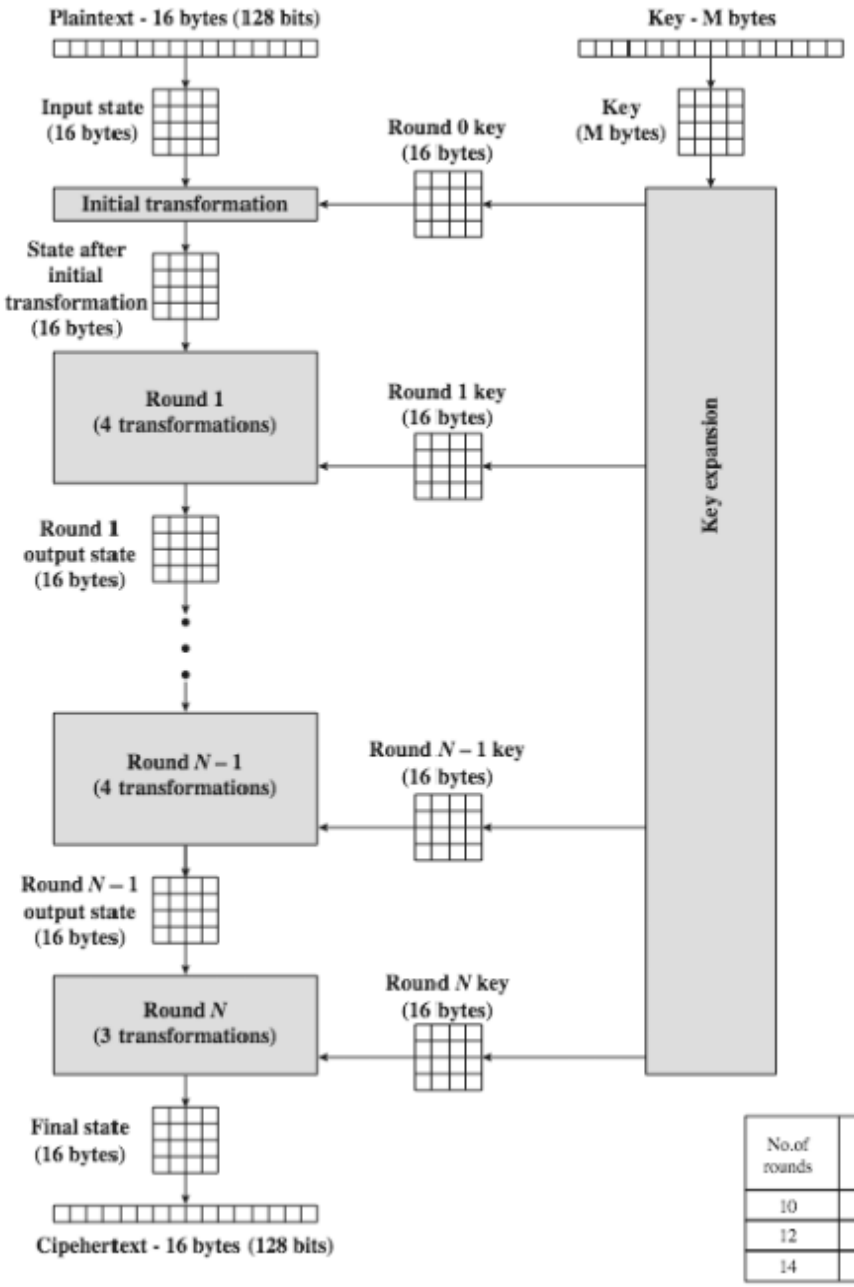
Block Cipher Design Principles

- The followed criteria need to be taken into account when designing a block cipher:
  - **Number of Rounds:** The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a weak function. The number of rounds is chosen so that efforts required to crypt analyze it becomes greater than a simple brute-force attack.
  - **Design of Function F**: F should be nonlinear and should satisfy strict avalanche criterion (SAC) and bit independence criterion (BIC).
  - **S-Box Design:** S-Box obviously should be non-linear and should satisfy SAC, BIC and Guaranteed Avalanche criteria. One more obvious characteristic of the S-box is its size. Larger S-Boxes provide good diffusion but also result in greater look-up tables. Hence, general size is 8 to 10.
  - **Key Generation Algorithm:** With any Feistel block cipher, the key is used to generate one sub key for each round. In general, sub keys should be selected such that it should be deduce sub keys from one another or main key from the sub key.

Advanced Encryption Standard (AES)

- The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six time faster than triple DES.
- A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.
- The features of AES are as follows
  - Symmetric key symmetric block cipher
  - 128-bit data, 128/192/256-bit keys
  - Stronger and faster than Triple-DES
  - Provide full specification and design details
  - Software implementable in C and Java

AES Structure

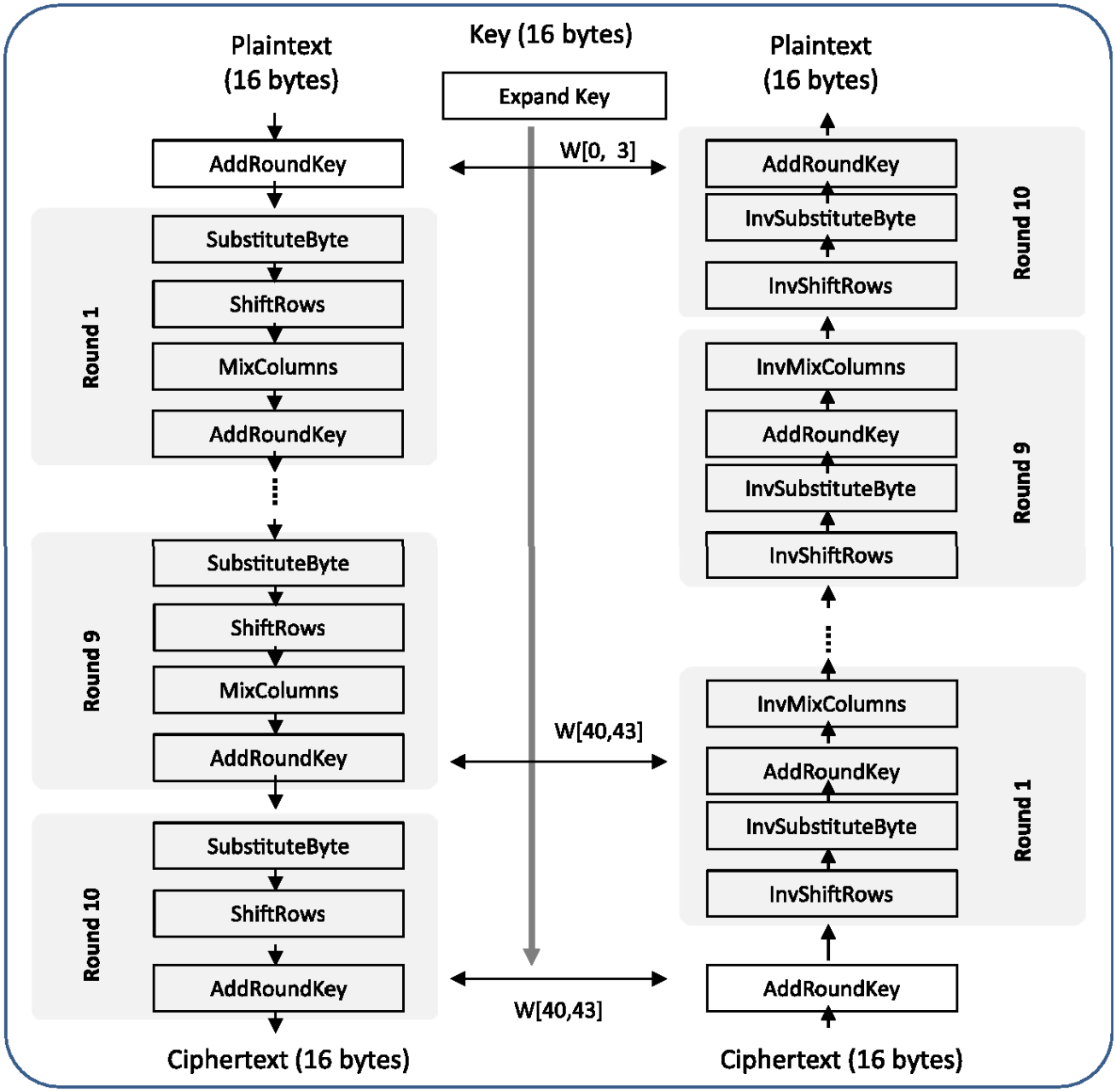


No.of rounds	Key Length (bytes)
10	16
12	24
14	32

- Figure shows the overall structure of the AES encryption process.
- Plain text block size is 128 bits (16 bytes).
- Key size is depends on number of round 128, 192, or 256 bit as shown in table.
- Based on key size AES is named as AES-128, AES-192, or AES-256.
- The input 128 bit block, this block is arranged in the form of 4 X 4 square matrix of bytes. This block is copied into the **state** array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix.
- There is a initial single transformation (AddRoundKey) before the first round which can be considered Round 0.
- The first N-1 rounds consist of four distinct transformation function: SubBytes, ShiftRows, MixColumns, and AddRoundKey,which are described subsequently.
- The final round contains only first three transformations of above round.

- Each transformation takes one or more 4 X 4 matrices as input and produces a 4 X 4 matrix as output.
- The key expansion function generates N+1 round key each of which is distinct 4 X 4 matrices. Each round key serves as one of the inputs to the AddRoundKey transformation in each round.

Detail Structure



- Figure shows detail encryption Decryption process of AES.
- Lets discuss Several comments about AES structure:
  1. It is not a Feistel structure. As we know in feistel structure half of the data block is used to modify the other half of the data block and then the halves are swapped. While in AES we use full data block as a single matrix during each round.
  2. The key is expanded into an array of fourty-four 32-bit words. And such four word (128-bit) serves as round key for each round.
  3. Four different stages are used one of permutation and three of substitution:
     - **SubBytes**: 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 bytes.
- **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 it begins with AddRoundKey, followed by nine rounds of all four stages, followed by tenth round of three stages.
5. Only AddRoundKey stage use key for this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
6. The AddRoundKey stage is in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key.
7. Each stage is easily reversible.
8. In AES decryption algorithm is not identical to encryption algorithm.
9. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plain text.
10. For making AES reversible the final round of both encryption and decryption are consists of only three stages.

AES Transformation Function

Substitute bytes Transformation (Forward & Inverse)

- Substitute bytes transformation is simple table lookup. There is separate table for forward and inverse operation.
- 16 X 16 matrix of byte value called s-box that contains the permutation of all 256 8-bit values. Each individual byte of state is mapped into a new byte in the following way.
- The left most 4-bit of the byte are used as row number and right most 4-bit are used as column number. Now row and column number serves as index into the s-box to select unique 8-bit output value.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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
	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

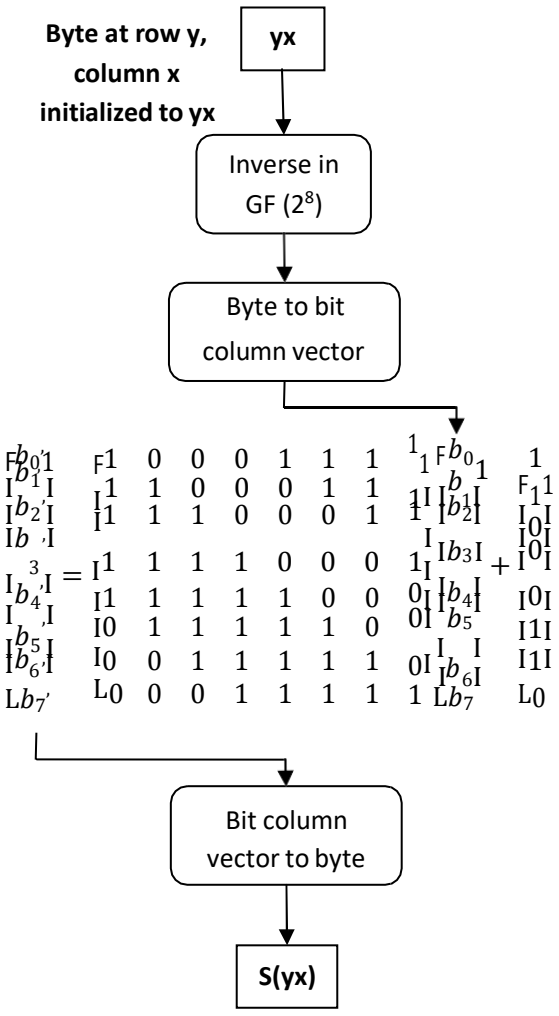
AES S-Box

- For example hexadecimal value 68 is referred to row 6 and column 8 and value in table at that position is 45 so byte value 68 is replaced with 45.
- For inverse substitute byte procedure is same but S-box is different. Reverse of above example is shown in figure.

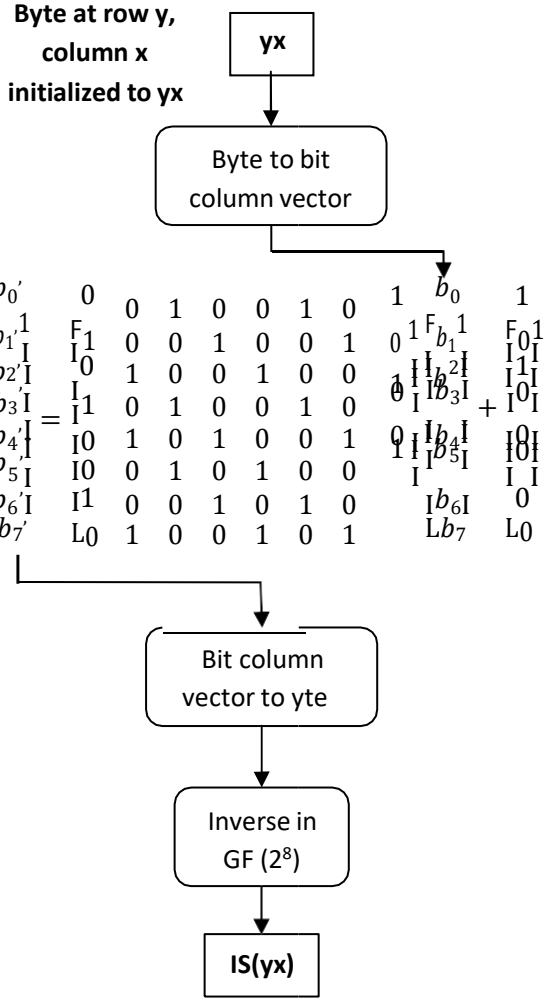
x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	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
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

AES Inverse S-Box

- S-box is constructed in the following fashion:



(a) Calculation of byte at row y, column x of S-box



(b) Calculation of byte at row y, column x of IS-

- Construction of S-box:
  1. Initialize the S-box with the byte values in ascending sequence row by row.
  2. Map each byte in the S-box to its multiplicative inverse in the finite field GF (2^8). The value {00} is mapped to itself.
  3. Consider that each byte in the S-box consist of 8 bits labeled (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0). Apply the transformation using matrix multiplication as shown in figure.
  4. Finally convert that bit column vector to byte.
- Construction of IS-box:
  1. Initialize the IS-box with the byte values in ascending sequence row by row.
  2. Consider that each byte in the IS-box consist of 8 bits labeled (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0). Apply the transformation using matrix multiplication as shown in figure.
  3. Convert that bit column vector to byte.
  4. Map each byte in the IS-box to its multiplicative inverse in the finite field GF (2^8).

ShiftRows Transformation (Forward & Inverse)

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

- The **forward shift row transformation** is performed as below:
  1. The first row of state is not altered.
  2. In second row we apply 1-byte circular left shift.
  3. In third row we apply 2-byte circular left shift.
  4. In fourth row we apply 3-byte circular left shift.
- In the **inverse shift row transformation** we apply right circular rotation.
  1. The first row of state is not altered.
  2. In second row we apply 1-byte circular right shift.
  3. In third row we apply 2-byte circular right shift.
  4. In fourth row we apply 3-byte circular right shift.

MixColumns Transformation (Forward & Inverse)

- In the forward MixColumn transformation each byte of a column is mapped into a new value that is a function of all bytes in that column.
- The transformation can be defined by the following matrix multiplication on state:

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

- In this case, the individual additions and multiplications are performed in GF (2^8).
- In the inverse MixColumn transformation procedure is same but matrix is different which is shown below.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} S_{0,0}' & S_{0,1}' & S_{0,2}' & S_{0,3}' \\ S_{1,0}' & S_{1,1}' & S_{1,2}' & S_{1,3}' \\ S_{2,0}' & S_{2,1}' & S_{2,2}' & S_{2,3}' \\ S_{3,0}' & S_{3,1}' & S_{3,2}' & S_{3,3}' \end{bmatrix}$$

AddRoundKey Transformation

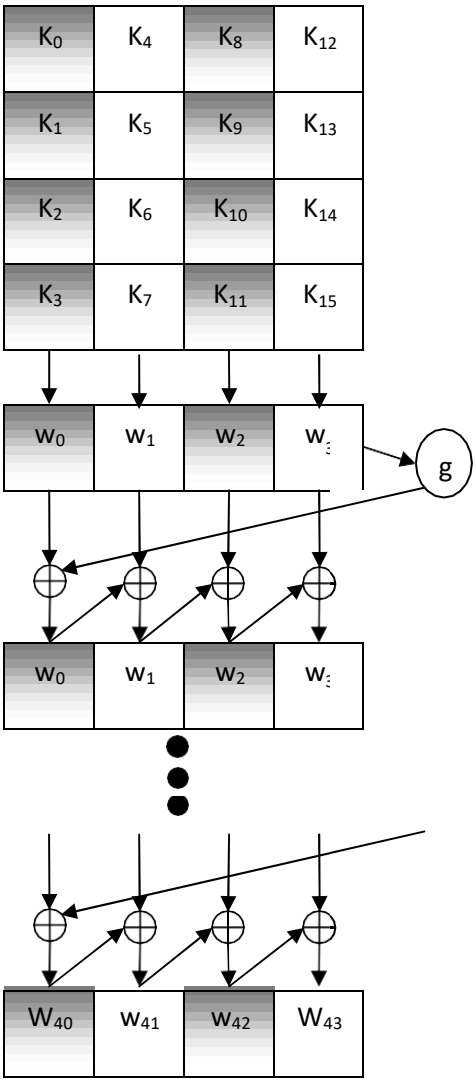
- In this transformation 128 bits state are bitwise XORed with the 128 bits of the round key.
- It is viewed as a byte level operation.
- Example

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

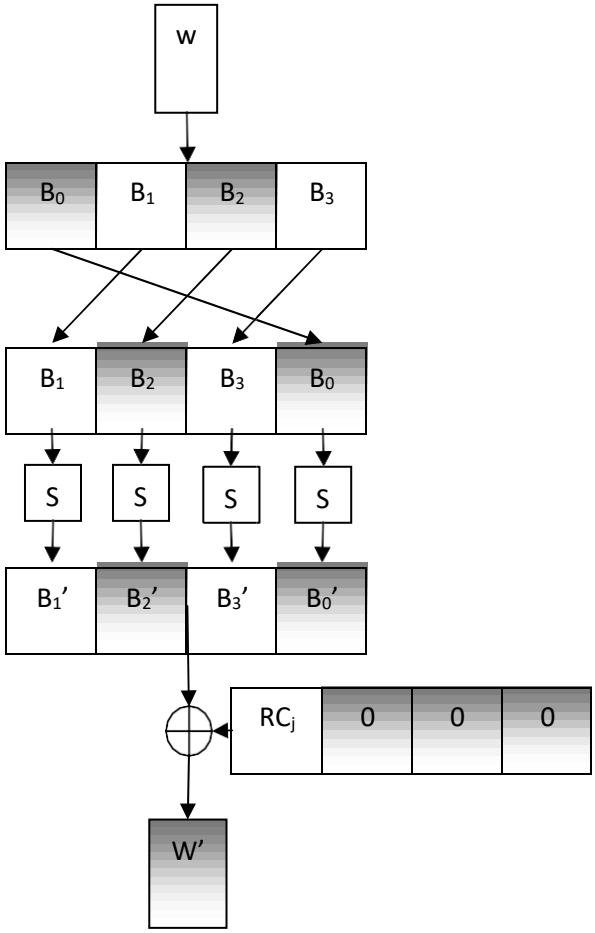
- Inverse of AddRoundKey is same because inverse of XOR is again XOR.

AES Key Expansion

- AES takes 16-byte key as input.
- As shown in figure below key expansion process is straight forward.



(a) Overall Key expansion algorithm.



(b) Function g.

- First of all key is stores in 4X4 matrix in column major matrix as shown in figure.
- Each column combines to form 32 bit word.
- Than we apply function g to every fourth word that is w3, w7, w11 etc.

- Than X-OR operation is performed as shown in figure to obtain next four word. And this process continues till generation of all words.
- As shown in figure (b) internal structure of function g.
- First we convert word to 4 byte.
- Then apply circular left shift operation.
- Then apply substitute byte operation using S-box which is same as S-box of AES encryption process.
- Than we apply X-OR operation with round constant which have least significant 3 byte as zero and most significant byte is depend on round number which is shown in table below.

Round (j)	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

- And output of this function is used for X-OR operation as shown in figure (a).

AES Example

- Let see example of AES and consider some of its implications.
- Although you are not expected to duplicate the example by hand, you will find it informative to study the hex patterns that occur from one step to the next.

Plaintext:	0123456789abcdeffedcba9876543210
Key:	0f1571c947d9e8590cb7add6af7f6798
Ciphertext:	Ff0b844a0853bf7c6934ab4364148fb9

- **Result:** Above table shows plain text, key and cipher text when we apply all the steps of AES we will get cipher text as shown.
- **The Avalanche Effect:** A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in cipher text.
- In particular, a change in one bit of plaintext or one bit of the key should produce a change in many bits of the ciphertext. This is referred to as the avalanche effect.
- In AES 1 bit change in input will affect nearly all bit of output after all rounds.

AES Implementation

Equivalent Inverse Cipher

- While implementing AES if we interchange the order of operation than it will affect the result or not is discussed here.
- If we interchange inverse shift row and inverse substitute byte operation than it will not affect and we get the same output.
- So we can obtain two equivalent decryption algorithms for one encryption algorithm.
- As inverse shift row will change position of byte and it will not affect byte value. While inverse substitute byte will change byte value by table lookup and it not concern with byte position. So we can interchange those two operations.
- If we interchange inverse mix column and add round key operation than it will affect and we do not get the same output.
- Both the operation will affect the value and so it cannot be interchange.

Implementation Aspects

- As in AES out of four three operation are byte level operation and it can be efficiently implemented on 8-bit processors.

- Only mix column operation is requiring matrix multiplication which requires some storage space and also time consuming on 8-bit processor.
- To overcome it we can use table lookup to reduce time requirement.
- Also we can implement it on 32-bit processors.
- In 32-bit processor we can use word by word operation and it much faster.