AES

➢ Introduction

The Advanced Encryption Standard (AES) is a **symmetric-key** encryption algorithm, which means that we can encrypt and decrypt using the same key. The **block size** is 128-bits (usually called state) and **key sizes** are 128,192 and 256 bits. The state array is formed from the input bytes as shown in the following figure with the relation: $s[r,c] = in[r+4c]\ for\ 0 \leq r \leq 4\ \&\ 0 \leq c \leq N_b$. Where r is the row, c is the column and N_b is the block length divided by 32.

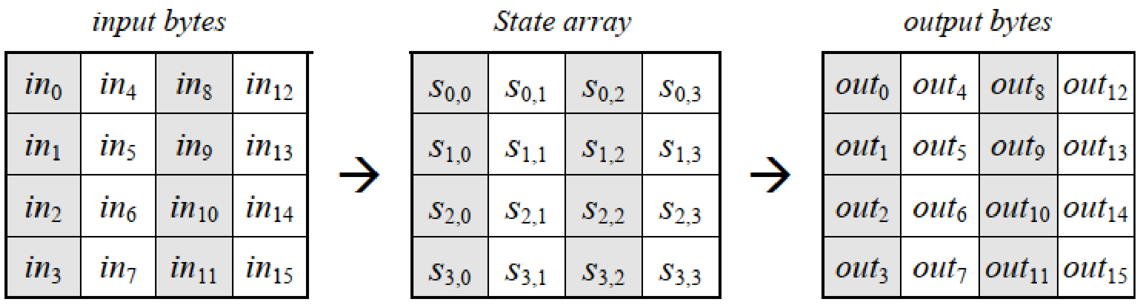

input bytes
State array
output bytes
in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 in_{11} in_{15}
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
out_0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

Figure 1:state array formation

All the arithmetic operations performed on the bytes are finite fields (galois field) operations. That is addition or subtraction are referred to bitwise XOR operation.

EX. $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \rightarrow polynomial\ notation$

$$\{01010111\}\ xor\{10000011\} = \{11010100\} \rightarrow binary\ notation$$

$$\{57\}xor\{83\} = \{d4\} \rightarrow hexadecimal\ notation$$

The multiplication is in $GF(2^8)$ which corresponds to multiplication of polynomials modulo an **irreducible polynomial** of degree 8. Which means that the degree of our polynomial should not exceed degree 8. If that happened, then we should get module of the resulted polynomial with: $x^8 + x^4 + x^3 + x + 1$ to ensure that we didn't exceed the limit of our degree which is 8. (there should be an easier way of doing this, but I can't figure it out)

➢ Operation Theory

The AES is based on substitution-permutation network which is efficient in both software and hardware. It has block size of 128 bits and key size of 128,192 or 256 (multiples of 64 bits with minimum of 128 and maximum of 256 bits). It operates on $4x4$ column-major order array of 16 bytes (number of columns may increase but the number of rows is always 4 as shown in the following matrix)

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

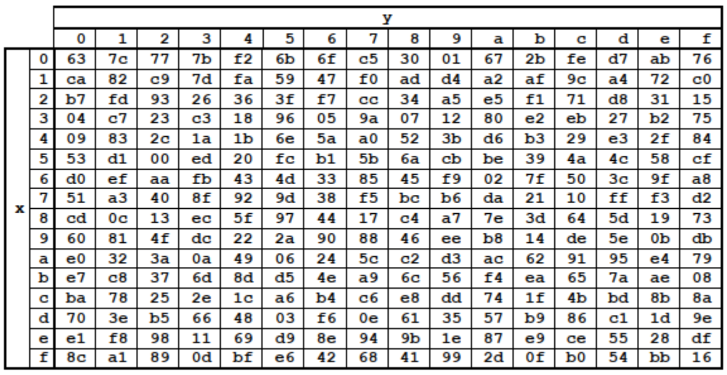
The key size specifies the number of transformation-rounds that convert the input called plaintext to the output called ciphertext. The same key is used to cipher and de-cipher the data. The number of rounds is as follows:

Table 1:number of rounds based on the key size.

10 rounds	128-bit keys
12 rounds	192-bit keys
14 rounds	256-bit keys

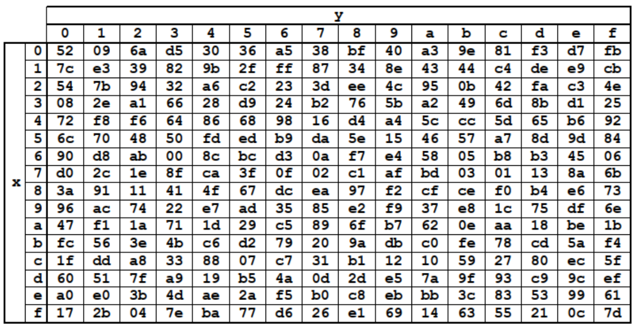
AES algorithm

1. we get the round keys from the AES key schedular which are of size 128-bits divided into 4 words for each round plus 1 more (the initial round).
2. Initial round key addition:
    a. AddRoundKey - each byte of the data is combined with a byte of the round key using bitwise XOR. Its inverse is the same value for each round.
3. For the next 9,11 or 13 rounds:
    a. SubBytes – a nonlinear substitution for each byte according to a lookup table shown in the figure below. The table below is constructed by some GF operations that I didn't understand. Its inverse is shown in the second table.



		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
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
x	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

Figure 2:SubBytes operation



		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
x	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

Figure 3:InvSubBytes lookup table

    b. ShiftRows – several circular left shifts based on the index of the row we are shifting (i.e the first row is shifted by 0, the second row is shifted by 1, the third row is shifted by 2, the fourth row is shifted by 3). The inverse of this operation is just shift right the same value for each row.

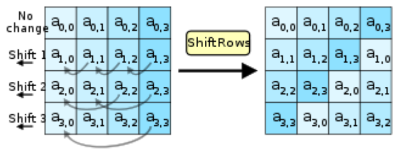

No change
Shift 1
Shift 2
Shift 3
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Figure 4:ShiftRows operation

    c. MixColumns – linear mixing operation done on each column to combine the four bytes of each column. It is simply a matrix multiplication followed by addition of a vector. This combination is called **<u>Affine transformation</u>**. Remember the multiplication mentioned in the introduction section.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \boldsymbol{Nb}. \qquad (5$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

Figure 5:MixColumns operation

The inverse for this operation is shown in the following figure. Refer to the multiplication and division algorithms for the polynomial bits in the AES_matlab file.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \boldsymbol{Nb}.$$

Figure 6:InvMixCols operation

        d. AddRoundKey
    4. Final round:
        a. SubBytes
        b. ShiftRows
        c. AddRoundKey

AES Key Schedule

The initial round (round 0) takes the main key and divides them to 32-bits words with: 4 words for AES-128 (requires 11 round keys), 6 words for AES-192 (requires13 round keys), 8 words for AES-256 (requires 15 round keys). While constructing the matrix of the main key that will be inserted to the key scheduler, which will be 4x4 for AES-128, 4x6 for AES-192 and 4x8 for AES-256, we concatenate each column to have the required words for the initial transformation. For AES-192 and AES-256, we will take only 4 words for the initial transformation and the rest of the keys will be taken for round 1. Then we generate the rest of the round keys using the following equations, but first we need to identify some notations to work with later:

- $\boldsymbol{N_k}$ $(\boldsymbol{N})$ as the length of the key in 32-bit words: 4 words for AES-128, 6 words for AES-192, and 8 words for AES-256.
- $\boldsymbol{K_0}, \boldsymbol{K_1}, \boldsymbol{K_2}, \ldots \boldsymbol{K_{N-1}}$ as the 32-bit words of the original key.
- $\boldsymbol{N_r}(\boldsymbol{R})$ as the number of rounds needed: 10 rounds for AES-128, 12 rounds for AES-192, and 14 rounds for AES-256.
- $\boldsymbol{N_b}$ the block size words, which is 4 as each word is fixed 32-bits and the block size is fixed 128-bits.
- $\boldsymbol{W_0}, \boldsymbol{W_1}, \ldots \boldsymbol{W_{4R-1}}$as the 32-bit words of the expanded key.
- **<u>RotWord </u>**is one-byte left circular shift [i.e RotWord([b0 b1 b2 b3]) = [b1 b2 b3 b0].
- **<u>SubWord </u>**is application for S-box to each of the 4 bytes of the word.

Then for $i = 0,1, \ldots 4R - 1$:

$$W_i = \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus \text{SubWord}(\text{RotWord}(W_{i-1})) \oplus rcon_{i/N} & \text{if } i \geq N \text{ and } i \equiv 0 \pmod{N} \\ W_{i-N} \oplus \text{SubWord}(W_{i-1}) & \text{if } i \geq N,\ N > 6, \text{ and } i \equiv 4 \pmod{N} \\ W_{i-N} \oplus W_{i-1} & \text{otherwise.} \end{cases}$$

The round constants ($rcon_i$) is a 32 bit word generated for each round. Its equation is as follows.

The round constant $rcon_i$ for round i of the key expansion is the 32-bit word:[note 2]

$$rcon_i = [\, rc_i \quad 00_{16} \quad 00_{16} \quad 00_{16} \,]$$

where rc_i is an eight-bit value defined as :

$$rc_i = \begin{cases} 1 & \text{if } i = 1 \\ 2 \cdot rc_{i-1} & \text{if } i > 1 \text{ and } rc_{i-1} < 80_{16} \\ (2 \cdot rc_{i-1}) \oplus 11\text{B}_{16} & \text{if } i > 1 \text{ and } rc_{i-1} \geq 80_{16} \end{cases}$$

where $\oplus$ is the bitwise XOR operator and constants such as 00_{16} and 11B_{16} are given in hexadecimal. Equivalently:

$$rc_i = x^{i-1}$$

where the bits of rc_i are treated as the coefficients of an element of the finite field $\text{GF}(2)[\text{x}]/(\text{x}^8 + \text{x}^4 + \text{x}^3 + \text{x} + 1)$, so that e.g. $rc_{10} = 36_{16} = 00110110_2$ represents the polynomial $x^5 + x^4 + x^2 + x$.

Values of rc_i in hexadecimal

i	1	2	3	4	5	6	7	8	9	10
rc_i	01	02	04	08	10	20	40	80	1B	36

➢ Description of Cyphers

Substitution-Permutation Network

The AES is based on **substitution-permutation network** which is efficient in software and hardware. The SP-network is a series of linked mathematical operations used in block cipher algorithms. It takes the plaintext and the key as inputs then applies alternating rounds of **S-boxes** and **P-boxes** (they are discussed below). These operations should be efficient to be performed on hardware such as XOR and bitwise rotations. The key is introduced in each round as the S-boxes may depend on them. The decryption is done by reversing the process.

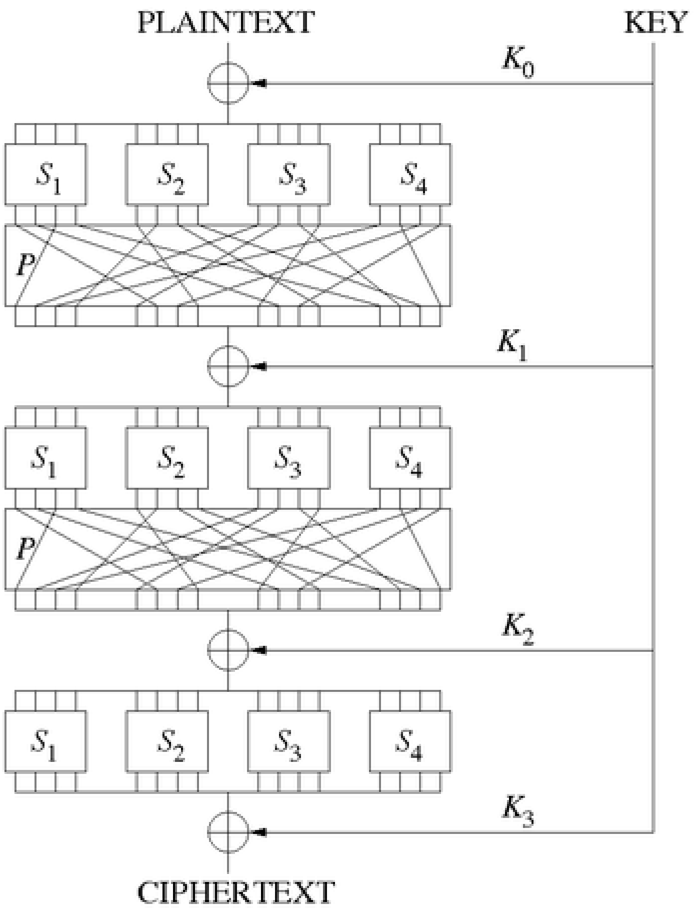
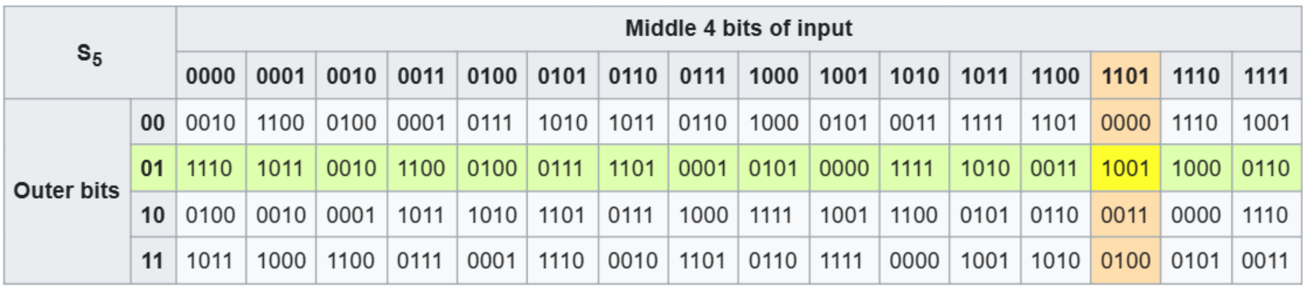


Figure 7: 3 rounds SP-network

- Substitution boxes (S-boxes)

It is a basic component of symmetric key algorithms that takes **m-bits** input and transforms them into **m-bits** output. That's why they call it a **non-linear vectorial Boolean function**. Therefore, it is called **$m * n$ s-box**. It can be implemented using look-up tables as shown in the figure below which is a **fixed table**. However, some ciphers generate the table dynamically as in the AES using a key.



S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

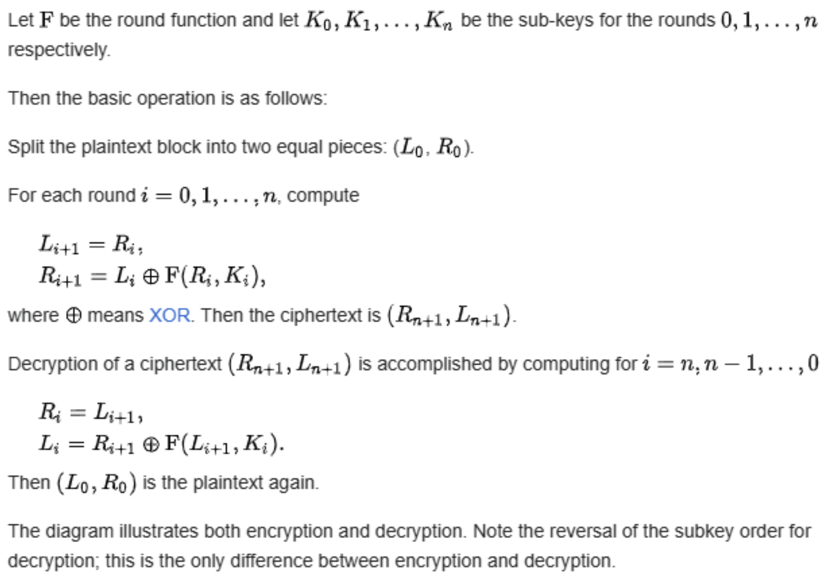
Figure 8: S-box lookup table

- Permutation Boxes (P-boxes)

It is a method of **bit shuffling** used to permute or transpose bits across the S-boxes inputs. P-boxes are classified as **compression, expansion and straight** depending on the number of output bits whether they are **less than, greater than or equal** **to the number of input bits** respectively. Only **straight boxes are invertible** (when decrypting, the output may be a result of multiple inputs, thus we should always deal with straight P-boxes).

Feistel Cipher

It is a symmetric structure used in the construction of the block cyphers that is used in the DES. Its main operation is to **divide the data into 2 halves** [Left:L, Right:R] then do operations on one of the halves then add the other half to the operated half. Its advantage is that **it is always invertible** unlike the SP-network. The figure below may illustrate the functionality clearly.



Let F be the round function and let $K_0, K_1, \ldots, K_n$ be the sub-keys for the rounds $0, 1, \ldots, n$ respectively.

Then the basic operation is as follows:

Split the plaintext block into two equal pieces: (L_0, R_0).

For each round $i = 0, 1, \ldots, n$, compute

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

where $\oplus$ means XOR. Then the ciphertext is (R_{n+1}, L_{n+1}).

Decryption of a ciphertext (R_{n+1}, L_{n+1}) is accomplished by computing for $i = n, n-1, \ldots, 0$

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

Then (L_0, R_0) is the plaintext again.

The diagram illustrates both encryption and decryption. Note the reversal of the subkey order for decryption; this is the only difference between encryption and decryption.

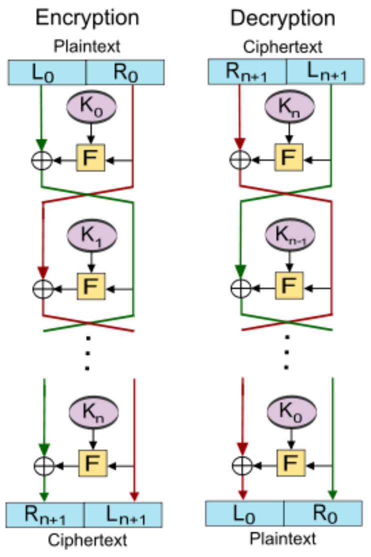


Figure 9:Feistel Cipher operation