RC4

- Remarkable characteristics of RC4 are simplicity and speed in software.
- The most important weakness of RC4 comes from the insufficient key schedule; the first bytes of output reveal information about the key.
- A number of attempts have been made to strengthen RC4……

RC4

- RC4 is not secure!
- Attacks
  - Scott R. Fluhrer, Itsik Mantin, Adi Shamir: "Weaknesses in the Key Scheduling Algorithm of RC4". Selected Areas in Cryptography 2001: 1-24.

  It is able to break the WEP encryption (that uses RC4) used in WiFi (802.11).

  - Andreas Klein: "Attacks on the RC4 stream cipher".

  Designs Codes and Cryptography 48(3): 269-286 (2008).

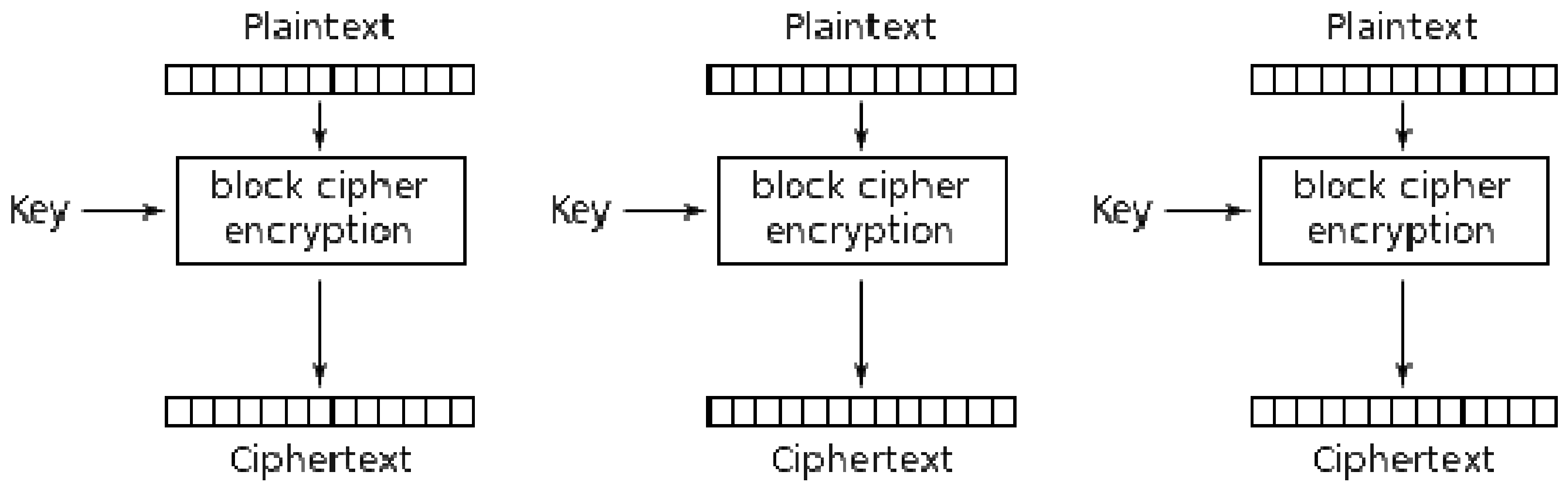
  It is able to break the WEP encryption in less than a minute (aircrack-ng tool)

Block ciphers

- Electronic codebooks (ECB) , Cipher-block chaining (CBC) , Output feedback (OFB), Counter mode (CTR).
- Substitution-permutation networks
- Feistel ciphers
- DES (overview)
- 3DES (overview)
- AES (overview)

Electronic codebook (ECB)

- The message is divided into blocks, and each block is encrypted separately.



Electronic Codebook (ECB) mode encryption

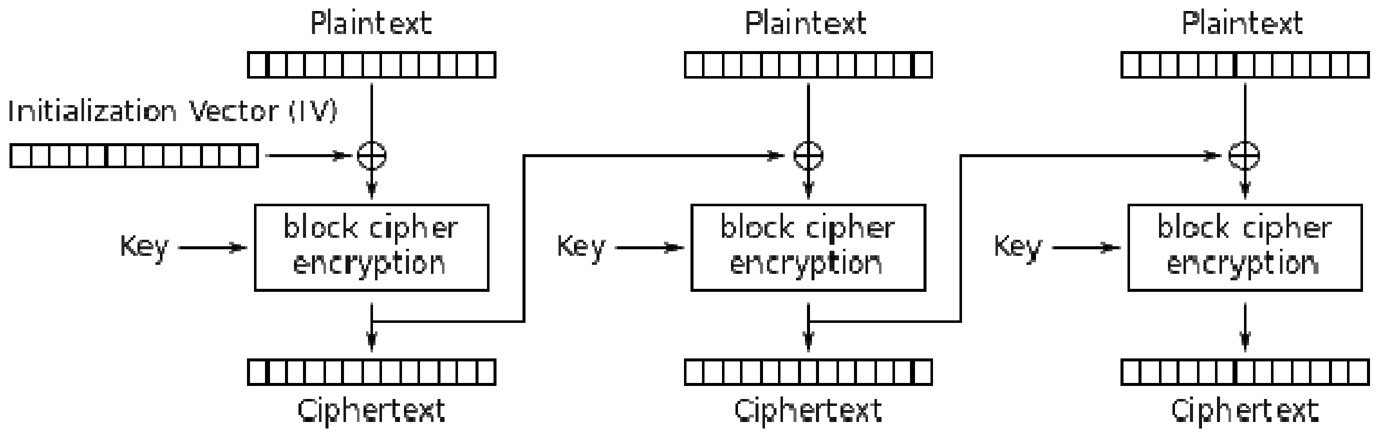
Electronic codebook (ECB)

- Pros:
  - Easy to implement
  - Parallel Encryption/decryption
- Cons:
  - Repetitions: identical plaintext blocks are encrypted into identical ciphertext blocks

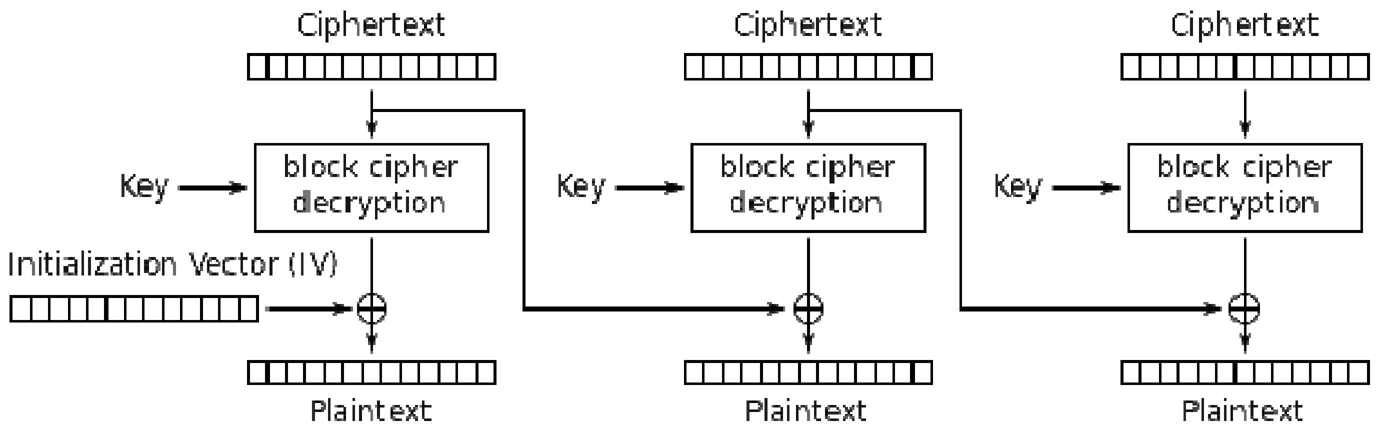
Cipher-block chaining (CBC)

- Each block of plaintext is XORed with the previous ciphertext block before being encrypted.
- Each ciphertext block depends on all plaintext blocks processed up to that point.
- An initialization vector must be used in the first block (IV or Seed).
- The seed can be communicated in plaintext because the encryption depends on the key.
- Usually the seed is a random number.

Cipher-block chaining (CBC)



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Cipher-block chaining (CBC)

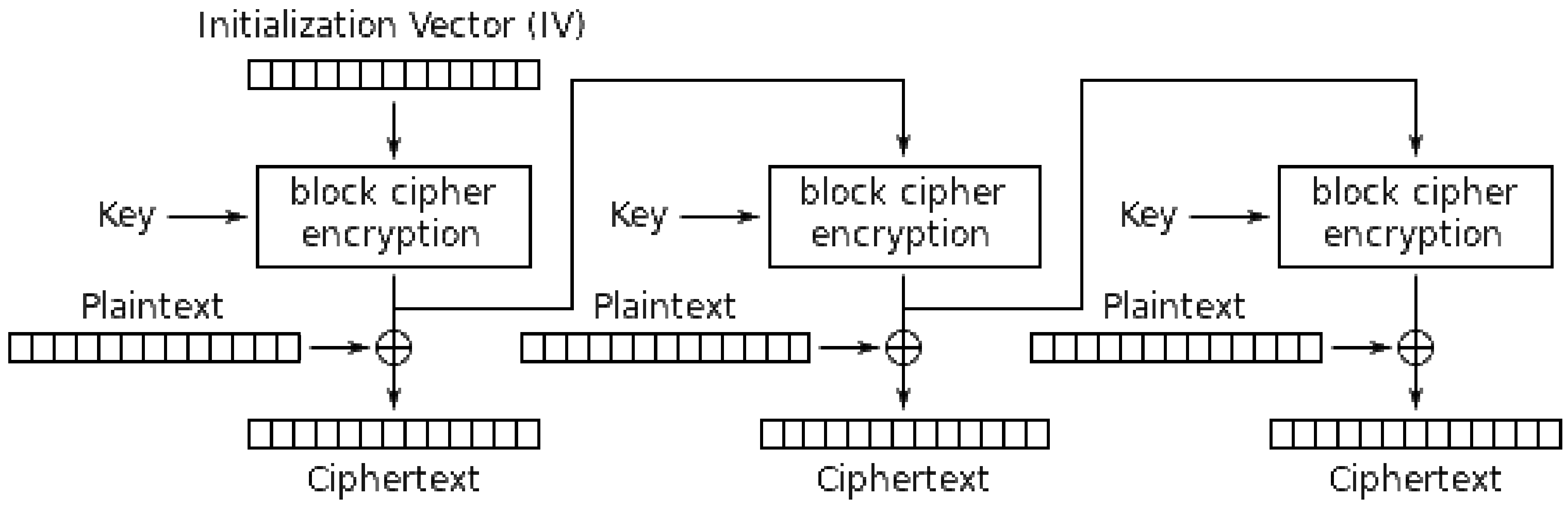
- Encryption
  - $C_0 = IV$
  - $C_i = E_K(M_i \text{ XOR } C_{i-1})$
- Decryption
  - $C_0 = IV$
  - $M_i = D_K(C_i) \text{ XOR } C_{i-1}$

Cipher-block chaining (CBC)

- Pros
  - No repetitions
- Cons
  - No parallel implementations known
  - Error propagation
    - A one-bit change in a plaintext or IV affects all following ciphertext blocks

Output feedback (OFB)

- Blocks of keystream are generated
- Such blocks are used in XOR with the plaintext to obtain the ciphertext



Output Feedback (OFB) mode encryption

Output feedback (OFB)

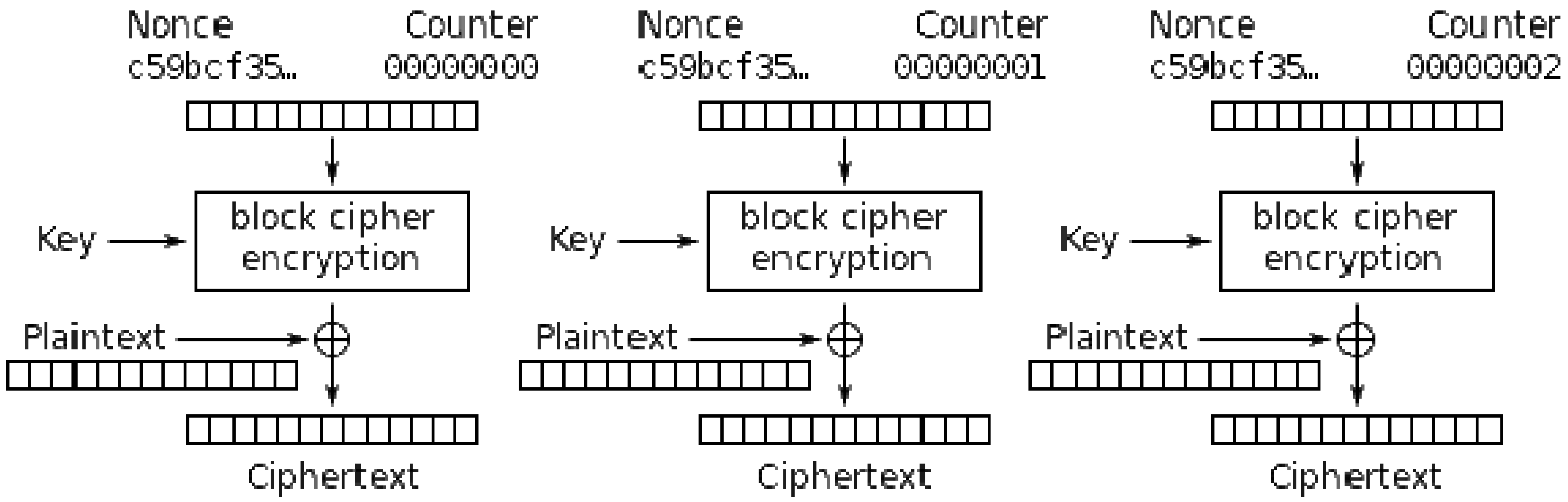
- Keystream blocks: O_j
- Encryption
  - $O_0 = IV$
  - $O_i = E_k(O_{i-1})$
  - $C_i = M_i$ XOR O_i
- Decryption is symmetric
  - $M_i = C_i$ XOR O_i

Output feedback (OFB)

- Pros
  - The block cipher operations may be performed in advance, and the final step can be performed in parallel once the plaintext or ciphertext is available
- Cons
  - The keystream blocks cannot be computed in parallel

Counter mode (CTR)

- Similar to OFB but it generates the keystreams block independently by using a counter.
- Therefore it allows parallelization.
- A **Nonce** is an arbitrary number used only once in a cryptographic communication.
- Typically the Nonce is random, and it is combined together with the counter using any lossless operation (concatenation, addition, or XOR) to produce the actual unique counter block for encryption.



Counter (CTR) mode encryption

Substitution-permutation (SP) networks

- There are two kinds of simple transformations one might imagine on a block of data: substitutions and permutations.

Let's assume we are encrypting k-bit blocks:

- A *substitution* specifies, for each of the 2^k possible values of the input, the k-bit output.
- A *permutation* specifies, for each of the k input bits, the output position to which it goes. For instance, the 1st bit might become the 13th bit of output, the 2nd bit would become the 61st bit of output, and so on.

- Notice that a permutation is a special case of a substitution in which each bit of the output gets its value from exactly one of the bits of the input.

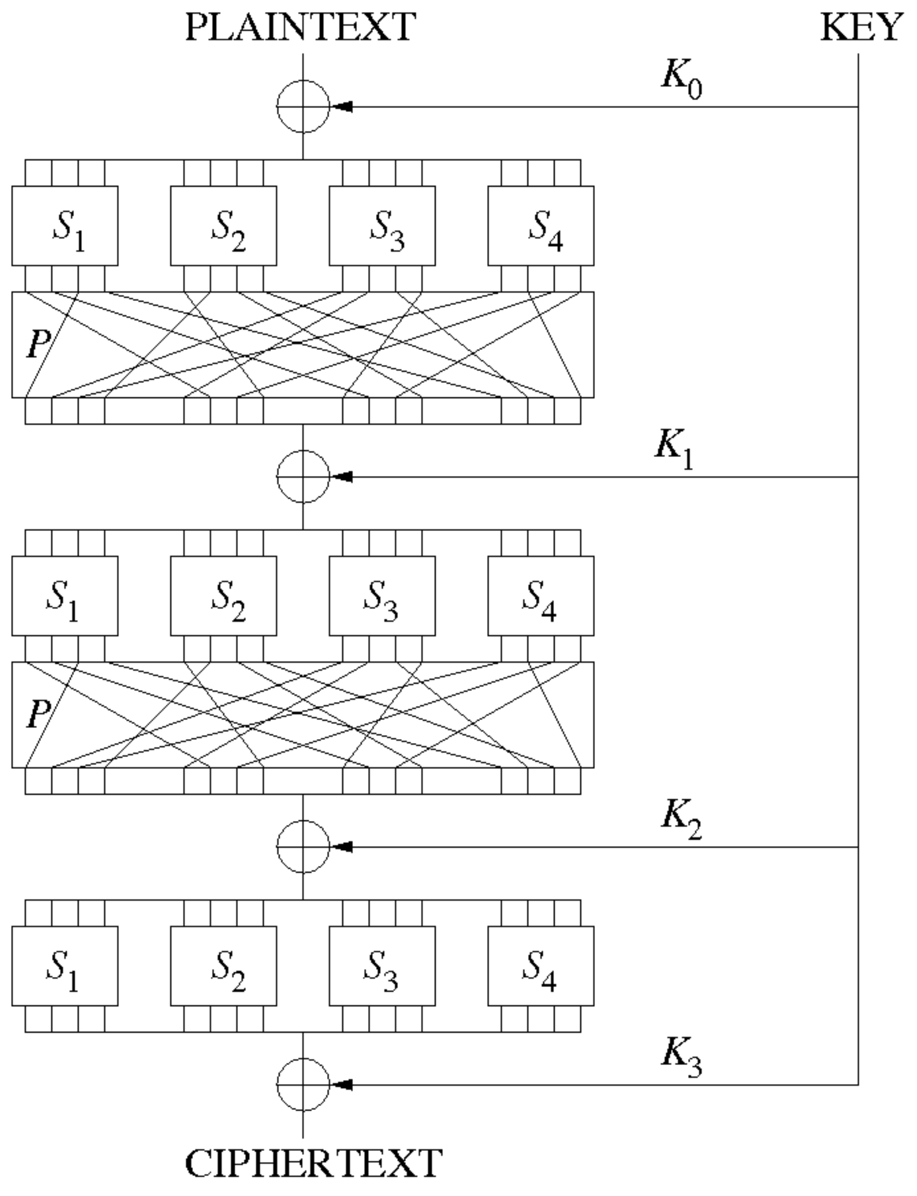
Substitution-permutation (SP) networks

- A substitution-permutation network applies several alternating "rounds" or "layers" of substitution boxes (S-boxes) and permutation boxes (P-boxes).

Substitution-permutation (SP) networks

- An S-box substitutes a (small) block of bits by another block of bits.
  - The substitution should be one-to-one, to ensure invertibility (hence decryption).
  - The length of the output should be the same as the length of the input.
  - A “good” S-box should have the following two properties:
    - Changing one input bit will change about half of the output bits.
    - Each output bit will depend on every input bit.
- A P-box is a permutation of all the bits: it takes the outputs of all the S-boxes of one round, permutes the bits, and feeds them into the S-boxes of the next round.
- At each round, the round key (obtained from the key with some simple operations, for instance, using S-boxes and P-boxes) is combined using some group operation, typically XOR.

Substitution-permutation (SP) networks



Substitution-permutation (SP) networks

- A well-designed SP network with several alternating rounds of S- and P-boxes satisfies Shannon’s *confusion and diffusion* properties.
- *Confusion*: each character of the ciphertext should depend on several parts of the key.
- *Diffusion*: if we change a character of the plaintext, then several characters of the ciphertext should change, and similarly, if we change a character of the ciphertext, then several characters of the plaintext should change

Feistel ciphers

- Similar to substitution permutation-networks but it uses an internal function called a round function which is not necessarily invertible

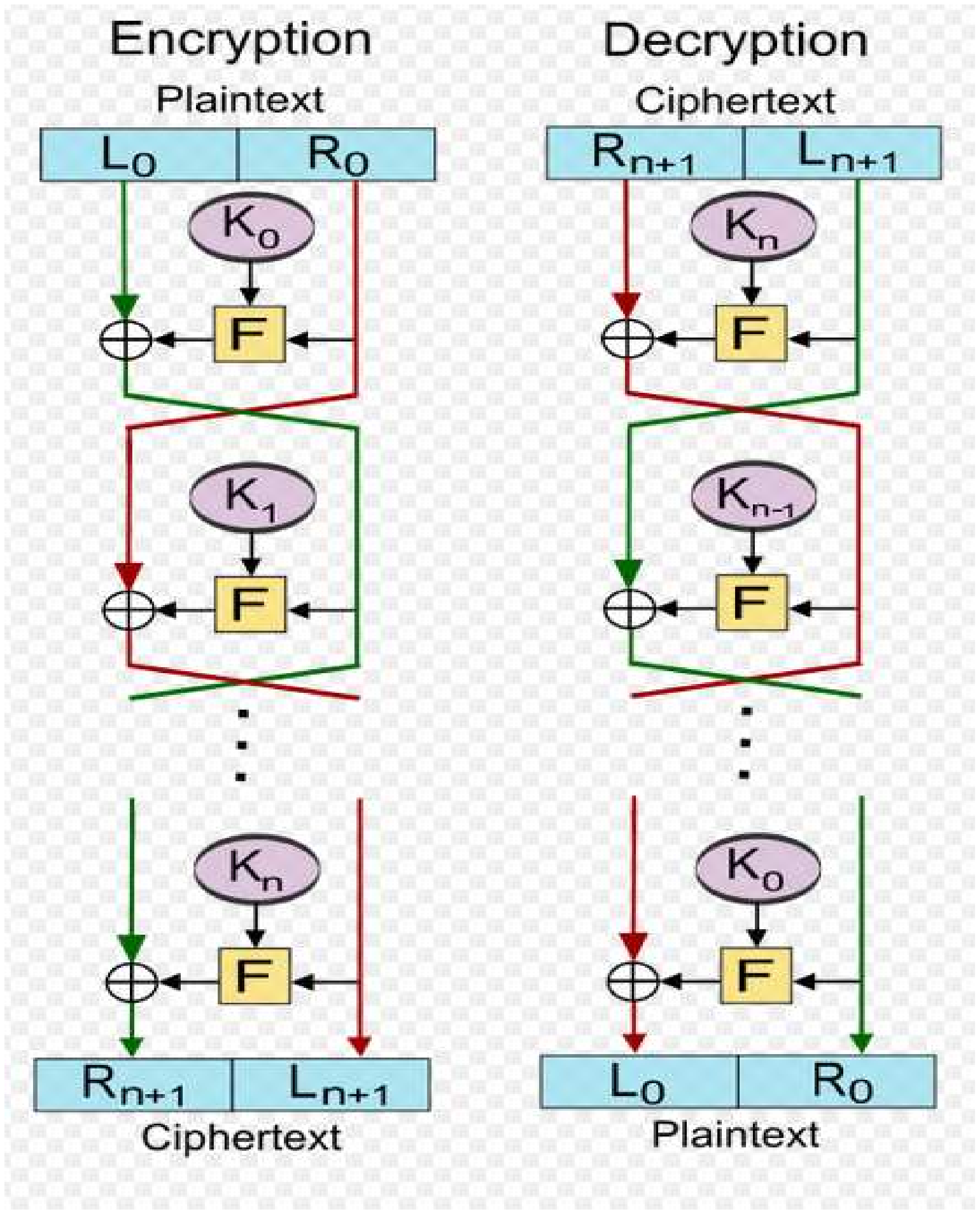
- Let F be the round function
- $K_0, K_1, \dots, K_n$ are the keys for the rounds $0, 1, \dots n$, resp.
- Encryption algorithm:

```
Split the plaintext block into two equal pieces, (L0, R0)
For each round i =0,1…,n.
    $L_{i+1} = R_i$,
    $R_{i+1} = L_i$ XOR $F(R_i, K_i)$.
C = $(R_{n+1}, L_{n+1})$.
```

- Decryption algorithm: the block is (R_{n+1}, L_{n+1})

```
For each round i =n,n-1…,0.
    $R_i = L_{i+1}$,
    $L_i = R_{i+1}$ XOR $F(L_{i+1}, K_i)$
M=$(L_0,R_0)$
```

Feistel ciphers



Feistel ciphers

- Pros:
  - The decryption operation is identical to encryption, but you just feed in the data and round keys in the opposite order.
  - It works with any round function F. It can be
    - noninvertible,
    - fast to compute,
    - hard to analyze,
    - easy to implement in software or hardware,
    - …

Data Encryption Standard (DES)

- Published in 1977 by the National Bureau of Standards for use in commercial and unclassified U.S. Government applications.
- It was designed by IBM and NSA.
- DES uses a 56-bit key, and maps a 64-bit input block into a 64-bit output block.
- The key actually looks like a 64-bit quantity, but one bit in each of the 8 octets is used for odd parity on each octet. Therefore, only 7 of the bits in each octet are actually meaningful as a key.

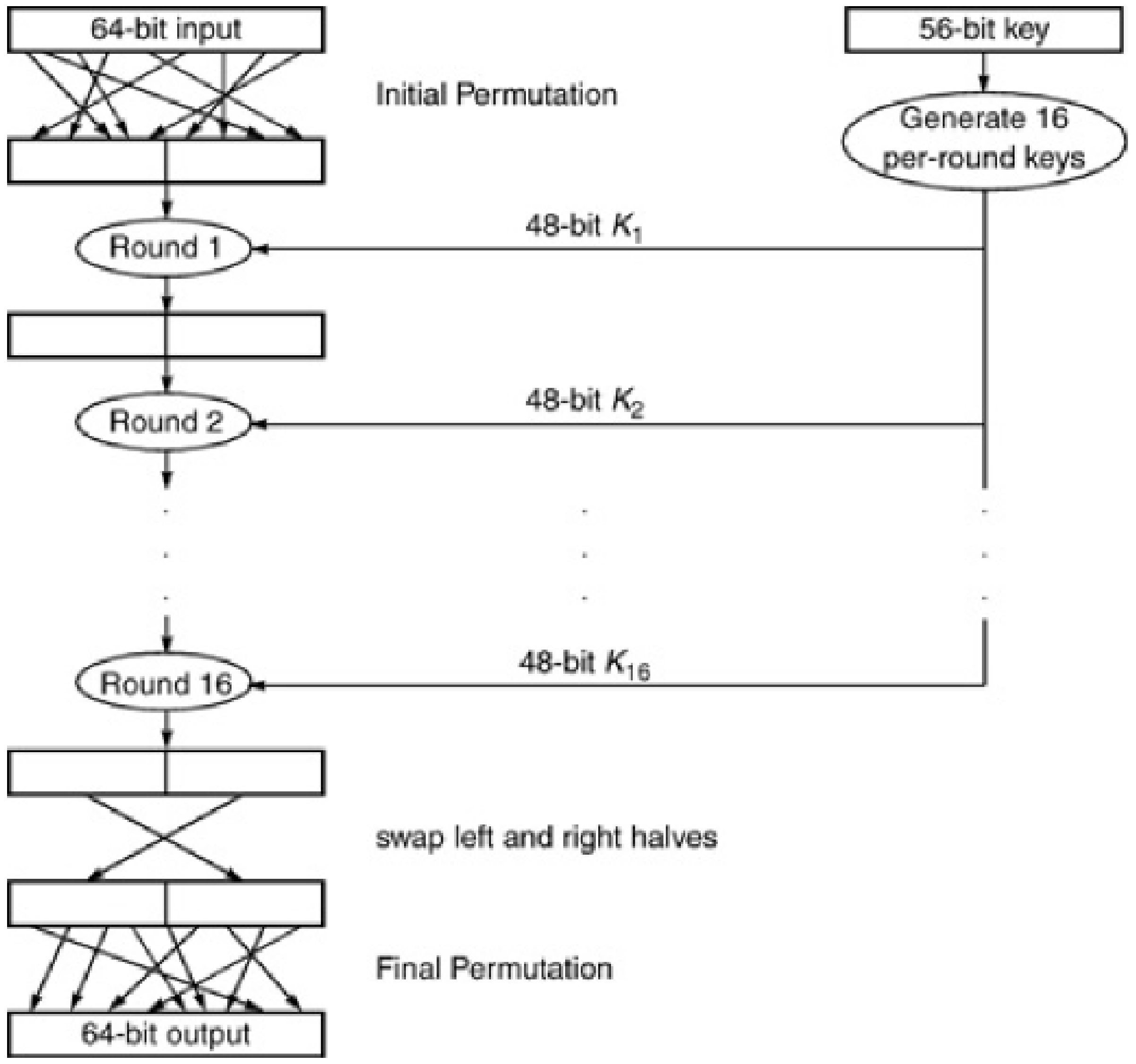
Data Encryption Standard (DES) Overview

- The 56-bits key is used to produce 16 round-keys of 48 bits by taking different subsets of the initial 56 bits.
- The input (64 bits plaintext) is initially subject to a permutation.
- Each round takes as input the 64-bits output of the previous round and the 48-bits round key, and produces a 64 bits output.
- After 16 rounds, the output is
  - split into two halves,
  - the two halves are swapped,
  - another permutation is applied which is the inverse of the first one.
- Decryption uses the same algorithm, backwards →

Data Encryption Standard (DES) decryption

- Decryption works by essentially running DES backwards.
- To decrypt a block, you'd first run it through the initial permutation to undo the final permutation (the initial and final permutations are inverses of each other).
- You'd do the same key generation, though you'd use the keys in the opposite order (first use K_{16}, the key you generated last).
- Then you run 16 rounds just like for encryption. Why this works will be explained when we explain what happens during a round.
- After 16 rounds of decryption, the output has its halves swapped and is then subjected to the final permutation (to undo the initial permutation).

Data Encryption Standard (DES) Overview



Data Encryption Standard (DES) remarks

- The design process of DES is a secret.
- Probably, some details have been chosen at random, some other details have been chosen to prevent specific attacks, finally some other details have been chosen for fitting the hardware and software means available when DES has been developed (about 1970).
- We are going to see an overview of it.

Data Encryption Standard (DES) Initial and final permutations

- The numbers specify the bit numbers of the input to the permutation. The order of the numbers in the tables corresponds to the output bit position.
- E.g. the initial permutation says: the first bit of the output is taken from the 58th bit of the input, the second bit from the 50th bit, and so on, with the last bit of the output taken from the 7th bit of the input.
- permutation is not a random-looking permutation and there is no security significance to this particular permutation.

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

Final 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

Data Encryption Standard (DES) Generating round keys

- The DES key looks like it's 64 bits long, but 8 of the bits are parity.
- Let's number the bits of the DES key from left to right as 1, 2,...64.
- Bits 8, 16,...64 are the parity bits.
- DES generates sixteen 48-bit keys, K_1, K_2,...,K_{16}.
- Initialization: permutation on the 56 useful bits of the key
  - The output of the permutation is divided into two 28-bit values, called C_0 and D_0.

C_0

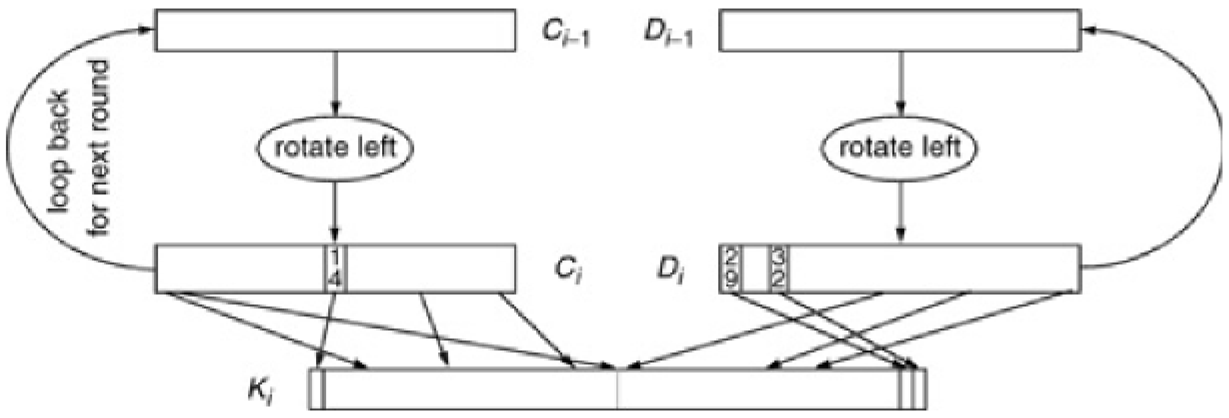
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36

D_0

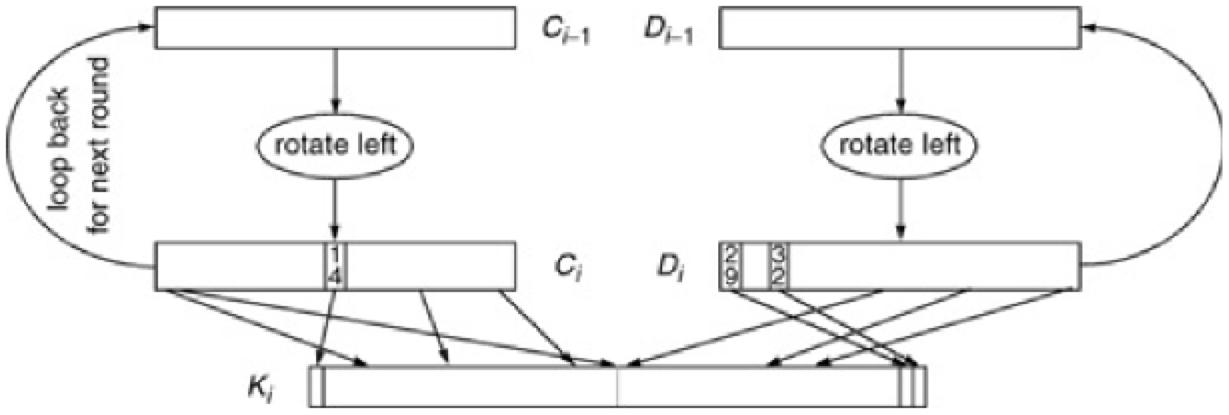
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

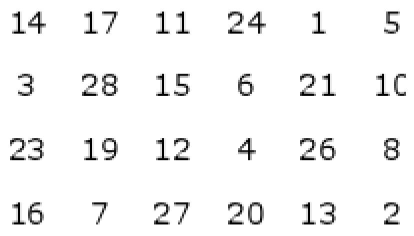
Data Encryption Standard (DES) Generating round keys

- The bits of C_{i-1} and D_{i-1} are shifted. The number of bits shifted is different in the different rounds.
  - In rounds 1, 2, 9, and 16, it is a single-bit rotate left
  - In the other rounds, it is a two-bit rotate left.
- Then another permutation is performed where some bits are discarded.



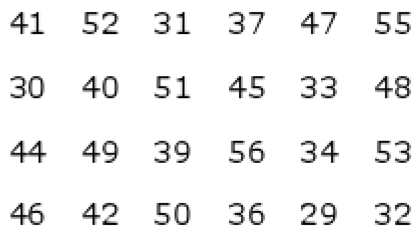
Data Encryption Standard (DES) Generating round keys





14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2

Permutation of the left part of K_i

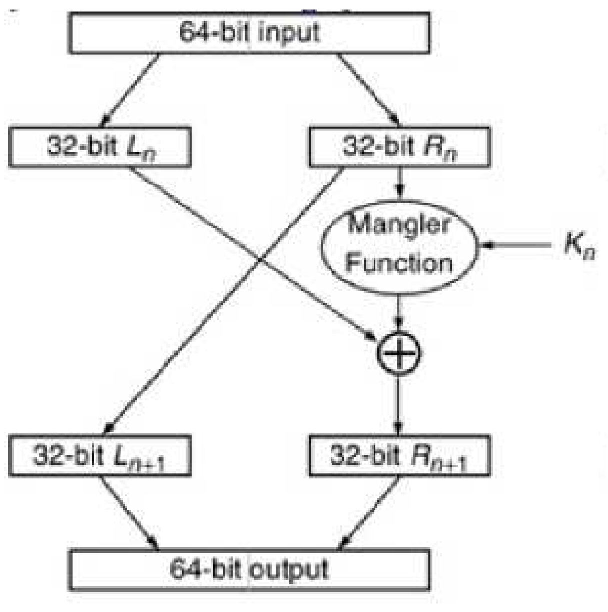


41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

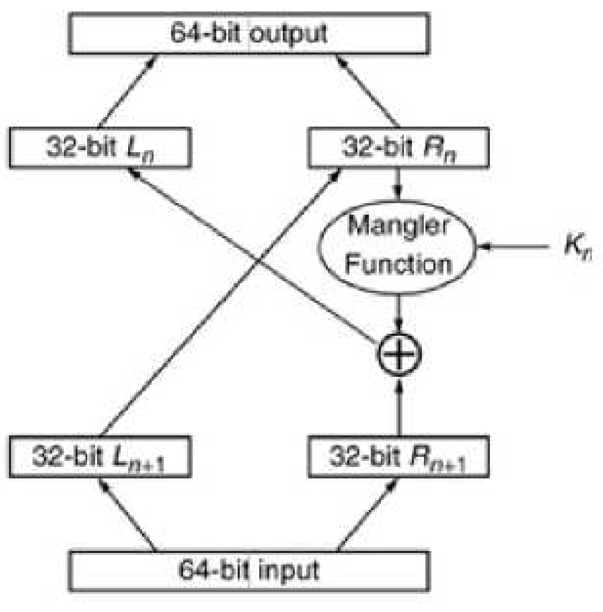
Permutation of the right part of K_i

Data Encryption Standard (DES)
DES rounds

- Encryption:
  - The 64-bit input is divided into two 32-bit halves called L_n and R_n.
  - L_{n+1} is equal to R_n
  - R_{n+1} is obtained as follows.
    - R_n and K_n are input to a **Mangler function** which takes as input 32 bits of the data plus 48 bits of the key to produce a 32-bit output.
    - The output is XORed with $L_{n.}$
  - The concatenation of L_{n+1} and R_{n+1} is the 64-bit output of the round.
- Decryption is symmetric →



Encryption

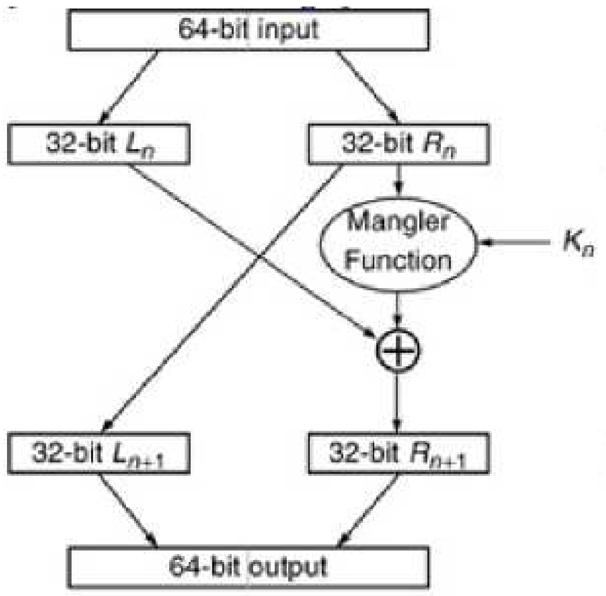


Decryption

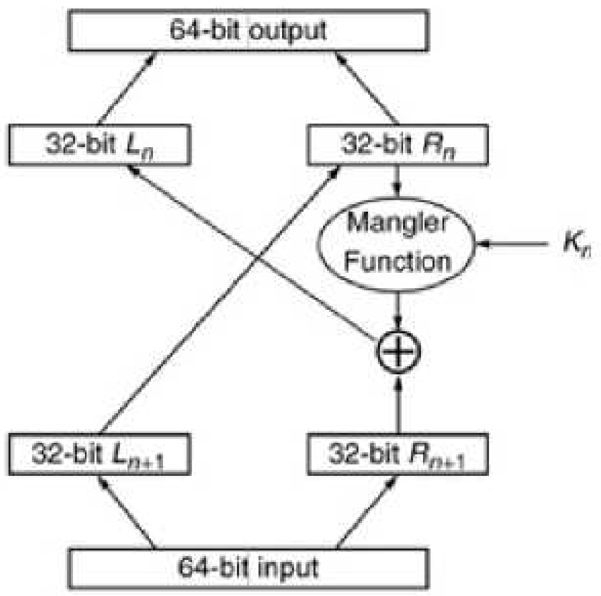
Data Encryption Standard (DES)
DES rounds

Decryption

- Suppose you know L_{n+1} and R_{n+1}. How do you get L_n and R_n?
- Well, R_n is just L_{n+1}. Now you know R_n, L_{n+1}, R_{n+1} and K_n. You also know that R_{n+1} equals L_n XOR mangler(R_n, K_n).
- You can compute mangler(R_n, K_n), since you know R_n and K_n. Now XOR it with R_{n+1}. The result will be L_n. Note that the mangler is never run backwards. DES is elegantly designed to be reversible without constraining the mangler function to be reversible. This design is due to Feistel.



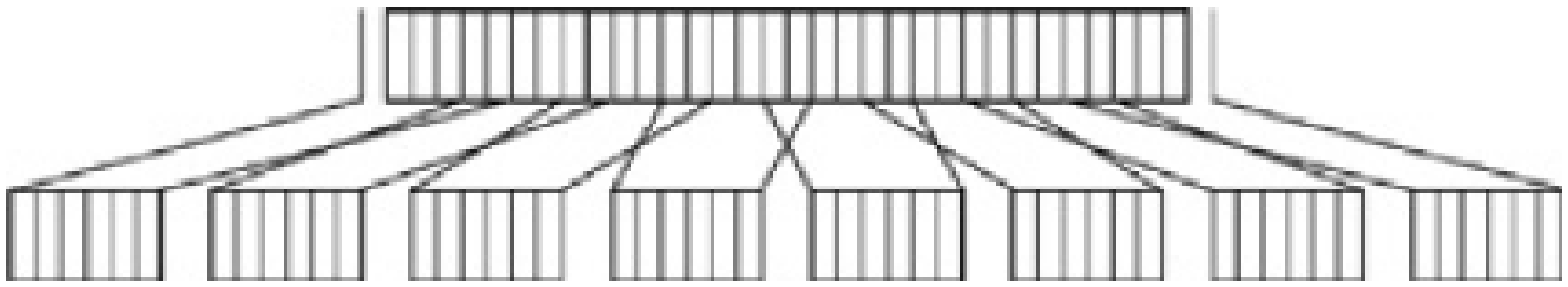
Encryption



Decryption

Data Encryption Standard (DES) Mangler function

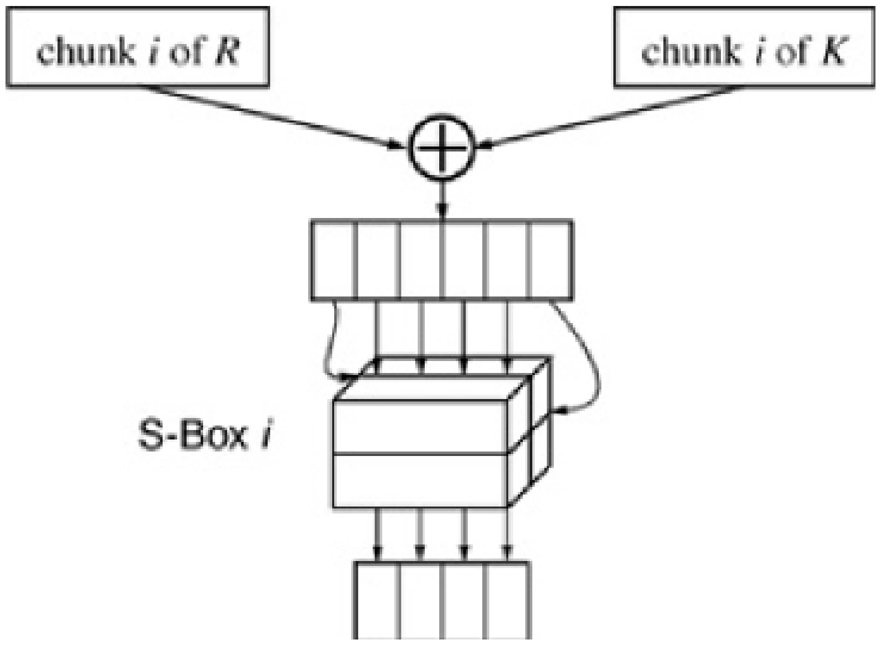
- It is like the round function F in Feistel ciphers
- To simplify notation: R:=R_n, K:=K_n
- The Mangler function first expands R from a 32-bit value to a 48-bit value.
  - It breaks R into eight 4-bit chunks and then expanding each of those chunks to 6 bits by taking the adjacent bits and concatenating them to the chunk. The leftmost and rightmost bits of R are considered adjacent.



- The 48-bits K is broken into 6-bits chunks

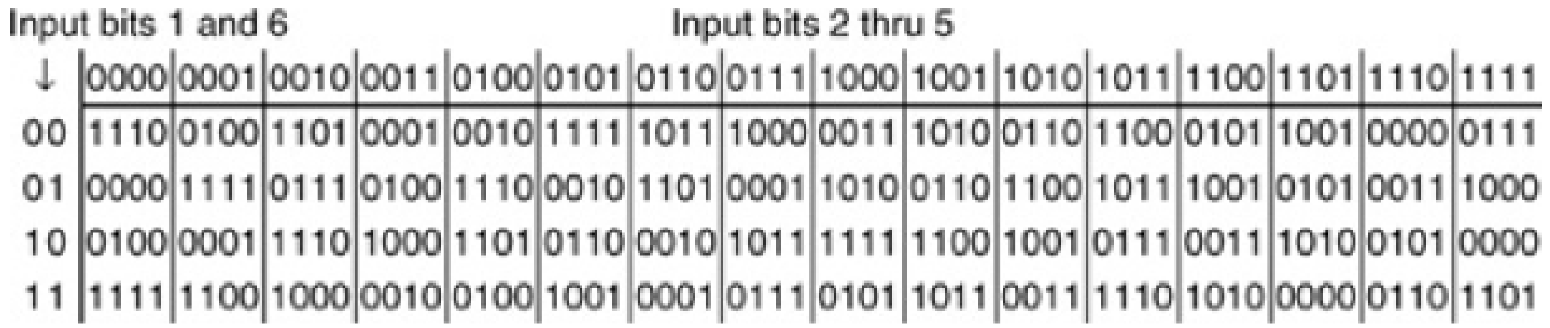
Data Encryption Standard (DES) Mangler function

- Chunks i of R and K are XORed
- The result is used as input to an S-box
- The S-box produces 4-bits from an input of 6 bits (i.e., it is not one-to-one!)
- The S-box maps several input to the same output
  - 6-bits input -> 64 possible inputs
  - 4-bits output -> 16 possible outputs
  - For each output value there are 4 possible input values



Data Encryption Standard (DES) Mangler function

- For each output value there are 4 possible input values
- Example of S-Box (1st chunk)



Input bits 1 and 6	Input bits 2 thru 5															
↓	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

- The 4-bits outputs of the 8 chunks are combined into 32-bits
- Finally, another permutation is performed.