Q1:

① $S_1(x_1) = 1110$

$S_1(x_2) = 0000$

1 2

8 4 2 1

So $S_1(x_1) \oplus S_1(x_2) = \boxed{1110}$

$x_1 \oplus x_2 = 000001$

So $S_1(x_1 \oplus x_2) = \boxed{0000}$

→ $S_1(x_1) \oplus S_1(x_2) \neq S_1(x_1 \oplus S_2)$

② $S_1(x_1) = 1101$

$S_1(x_2) = 0100$

→ $S_1(x_1 \oplus S_2) = \boxed{1001}$

$x_1 \oplus x_2 = 011111$

$S_1(x_1 \oplus x_2) = \boxed{1000}$

So $S_1(x_1) \oplus S_1(x_2) \neq S_1(x_1 \oplus S_2)$

③ $S_1(x_1) = 0110$

$S_1(x_2) = 1100$

→ $S_1(x_2) \oplus S_1(x_2) = \boxed{1010}$

$x_1 \oplus x_2 = 111111$

$S_1(x_1 \oplus x_2) = \boxed{1101}$

So $S_1(x_1) \oplus S_1(x_2) \neq S_1(x_1 \oplus S_2)$

P2

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: 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

According to the table:

$IP(1) = 58$

$IP(2) = 50$

$IP(3) = 42$

$IP(4) = 34$

$IP(5) = 26$

$IP^{-1}(IP(1)) = IP^{-1}(58) = 1$

$IP^{-1}(IP(2)) = IP^{-1}(50) = 2$

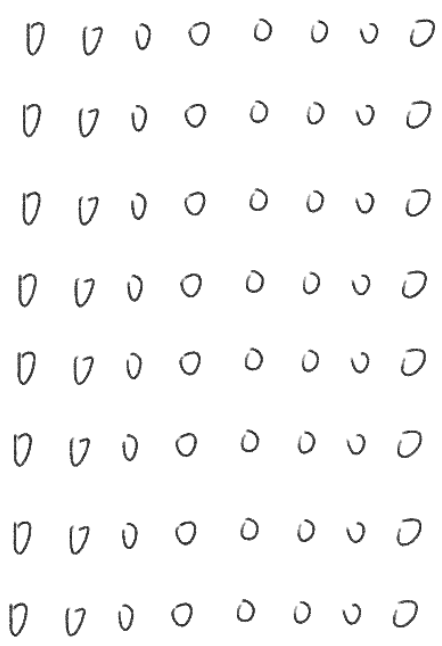
$IP^{-1}(IP(3)) = IP^{-1}(42) = 3$

$IP^{-1}(IP(4)) = IP^{-1}(34) = 4$

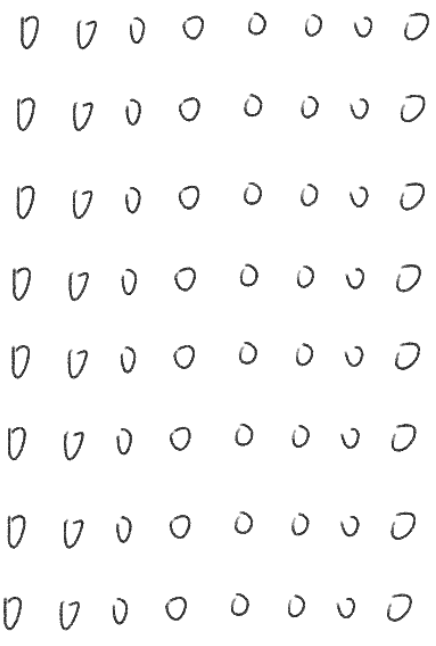
$IP^{-1}(IP(5)) = IP^{-1}(26) = 5$

P3

plaintext = 64 bits



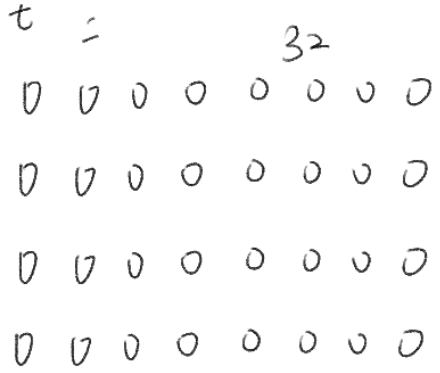
key = 64 bits



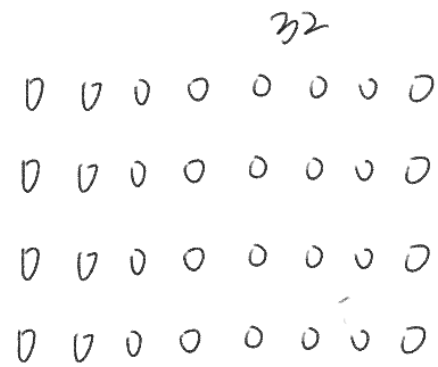
Input → IP: ↓ 64 bits

Input Split :

left :



Right :



E BIT-SELECTION TABLE

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

E-expansion to Right half :

Now the key after pc-1 =

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

→

left shift once :

omitted the left - right split of the key, because it's gonna be the same.

after 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

→

now use the input after E-expansion XOR the key we obtain from the last page.

48

0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

⊕

48

0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

=

48

0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

now use S-box:

8 4 2 1

14	1 1 1 0
15	1 1 1 1
10	1 0 1 0
7	0 1 1 1
2	0 0 1 0
12	1 1 0 0
4	0 1 0 0
13	1 1 0 1

→

Apply P-box

right =

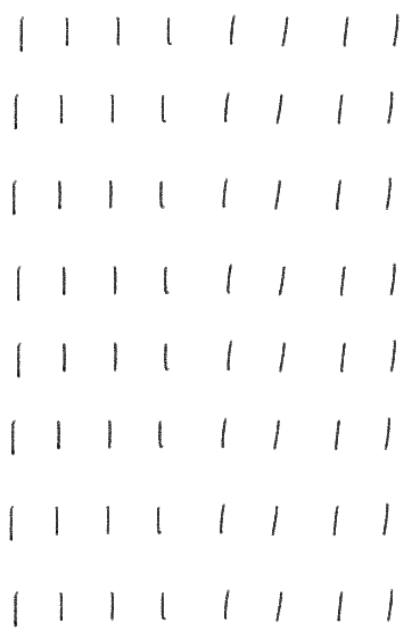
next right

next left ⟶

Q4:

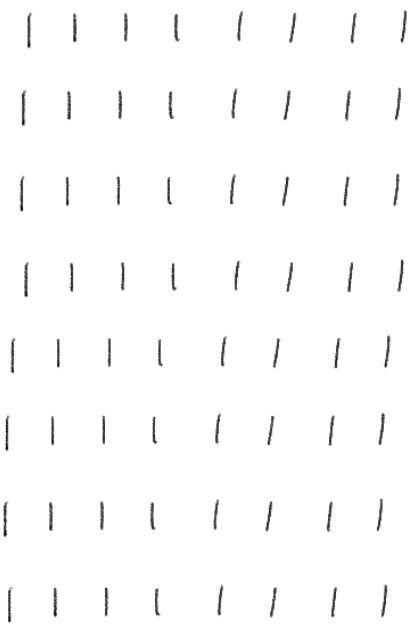
Input:

64



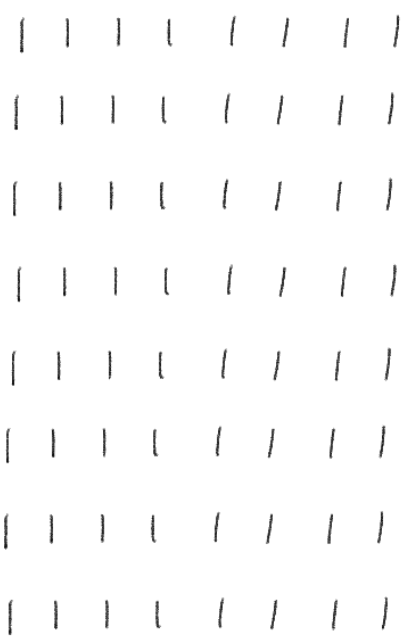
key:

64



① Input + Ip:

64

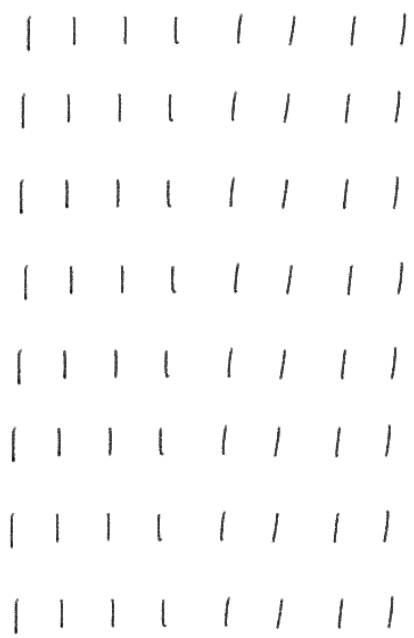


+ Ip table

64

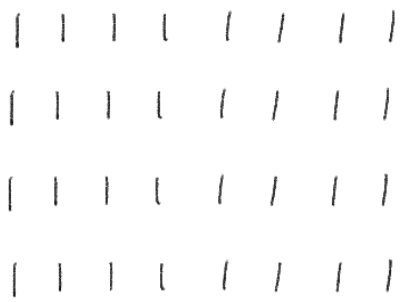
=

64

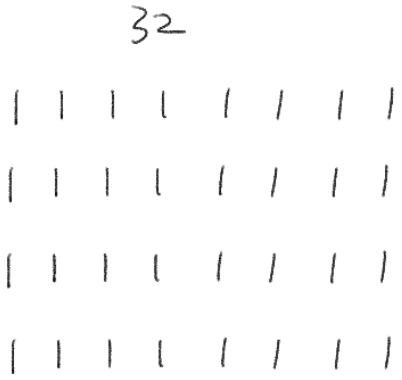


② Input split :

right :



left :



apply E-table to right half:

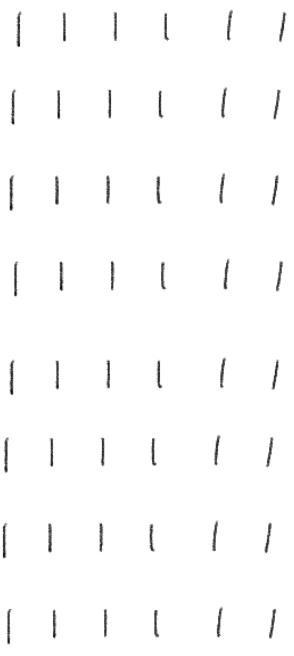
32

Right + E-table

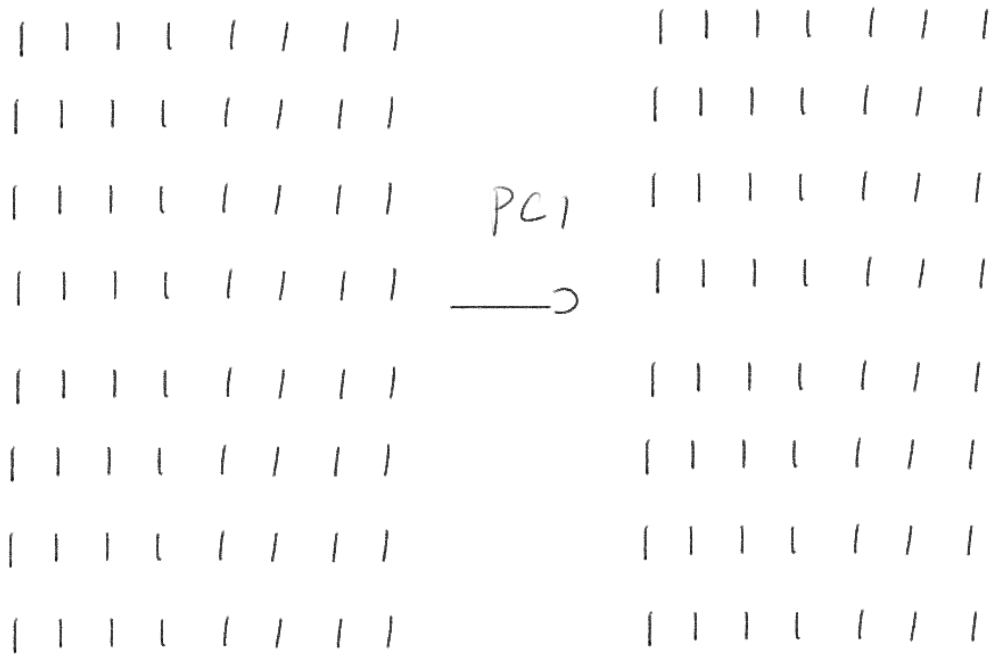
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

48

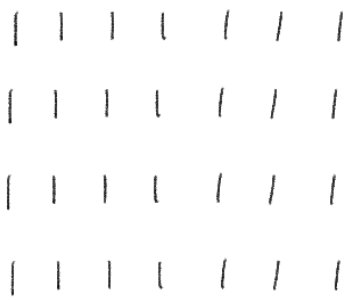
⇒



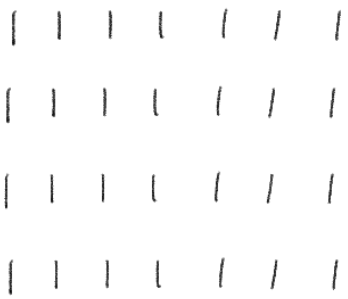
key:



split: right =



left:



Both Shift left once, and Combine:

56

PC 2

48

Final Key

Now

48

Input after E-table ⊕ final key

⟶

0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

now apply S-box

8 4 2 1

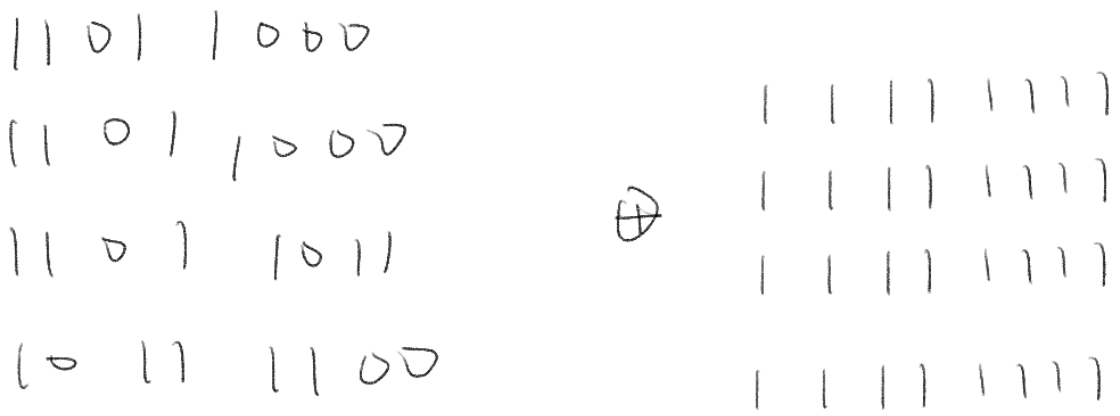
14
15
10
7
2
12
4
13

S-box ⟶

1 1 1 0
1 1 1 1
1 0 1 0
0 1 1 1
0 0 1 0
1 1 0 0
0 1 0 0
1 1 0 1

P-box ⟶

1 1 0 1 1 0 0 0
1 1 0 1 1 0 0 0
1 1 0 1 1 0 1 1
1 0 1 1 1 1 0 0

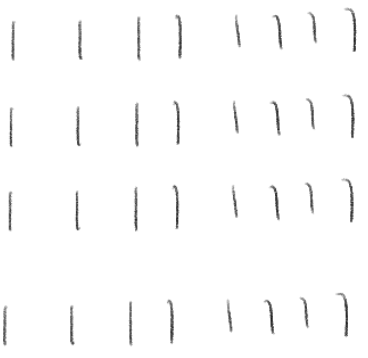


$$= \quad 0\,0\,1\,0 \quad 0 \quad 1\,1\,1$$

$$0\,0\,1\,0 \quad 0\,1\,1\,1 \longrightarrow \text{new right half}$$

$$0\,0\,1\,0 \quad 0 \quad 1\,0\,0$$

$$0\,1\,0\,0 \quad 0\,0\,1\,1$$



Q5:

Input :

key:

we know already that the key after pc-2 is : 48

Input +

The 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

—>

Split Input:

32

left = 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

32

right 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

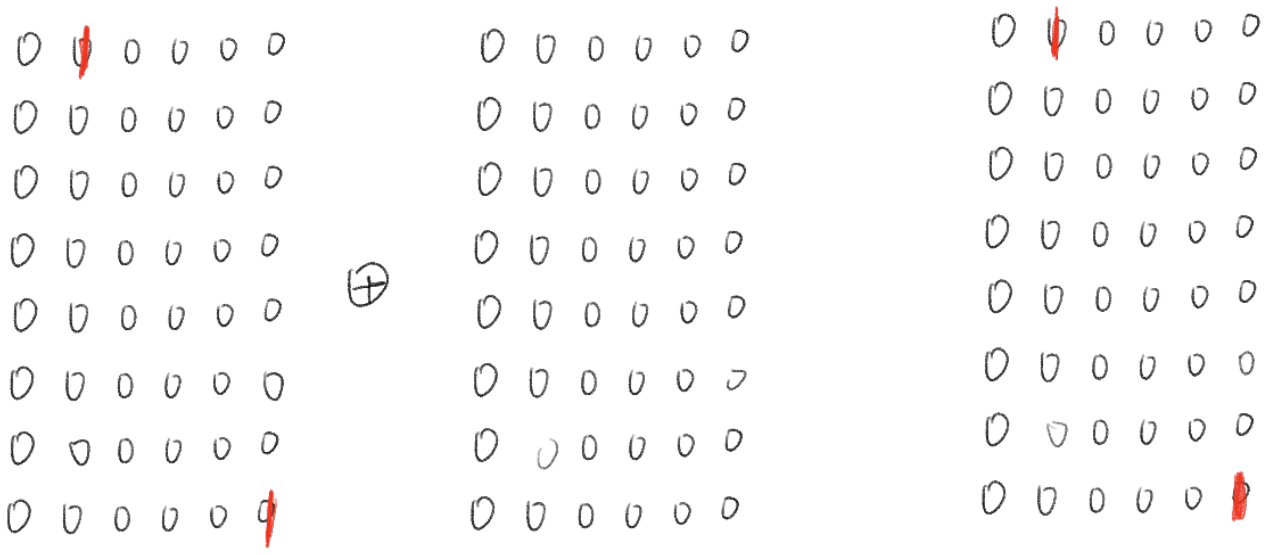
E-table performed upon the right half

E BIT-SELECTION TABLE

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

—> 0 0 0 0 0 0 48
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

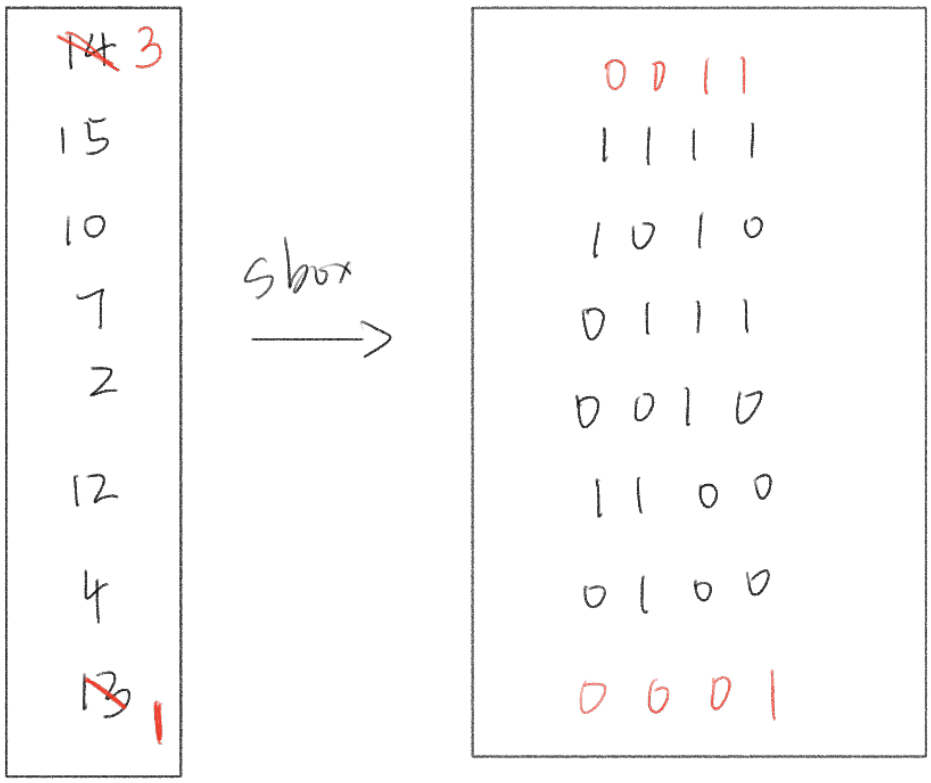
Now Use the 48 bit Input to XOR the 48 bit key



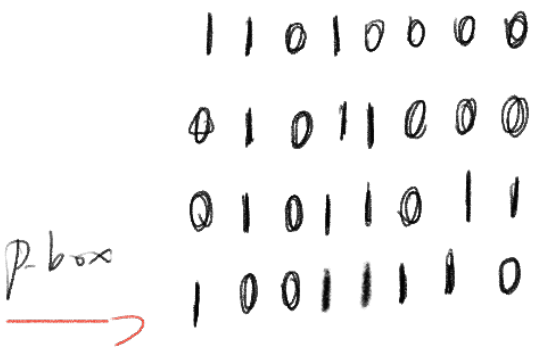
① 2 S-boxes get different

8 4 2 1

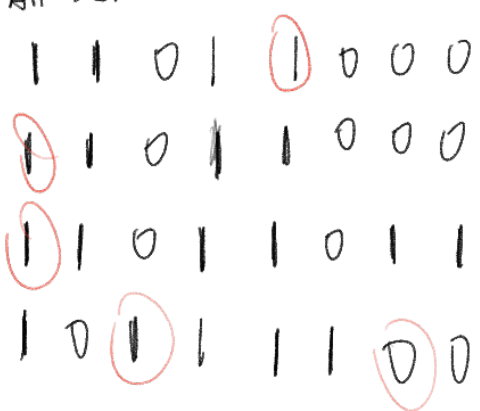
S-box:



new right:



All zeros:



(2.) 1 bit change in input ⟶ at least 2 bits change in output. So in this case at least 4

(3) as illustrated, the output is

L: 1000 0000 0000 0000 0000 0000 0000 0000

R: 1101 0000 0101 1000 0101 1011 1001 1110

(4). change in L_1 : 1

change in L_2 : 5

total : 6

Q6:

In the DES, the 64 bits keys are converted to 56 bits through PC-1, and a 48 bits subkey is generated during each round.

Therefore, find the 56-bits key is important.

In worst-case-scenario, we have to exhaustively search for each bit ——> $\boxed{2^{56}}$.

On average, we're expected to find all the key after getting half of it

——> $2^{56} / 2 = \boxed{2^{55}}$

P7:

①

plaintext: 0000 0000 0000 0000

Round key: BBBB 5555 5555 EEEE 64 bit

KeyAdd (xor) BBBB 5555 5555 EEEE

S-box: 8888 0000 0000 1111

To binary: 1000 1000 1000 1000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0001 0001 0001

64bits

P-box:

1000 0000 0000 0001 1000 0000 0000 0001 1000 0000 0000 0001 1000 0000 0000 0001

Hex: 8001 8001 8001 8001

②

key schedule:

key: BBBB 5555 5555 EEEE FFFF

round key for Round 1: BBBB 5555 5555 EEEE

↓

rotation: DFFF F777 6AAA AAAA BDDD

S-box: DFFF F777 6AAA AAAA BDDD

→ 7FFF F777 6AAA AAAA BDDD only left 4 bits

Counter Add with (00010): 7FFF E777 6AAA AAAA BDDD

P7:

```python
def step_forward(seed, tap_pos):
    # step 1: calculate the bit being generated
    # make seed str, so be able to fetch each digit
    bit_generated = str(int(seed[tap_pos]) ^ int(seed[-1]))
    # move right,discard the last bit
    seed = bit_generated + seed[:-1]

    return seed


def LFSR(n, init_seed, tap_pos, bits_num):
    seed = init_seed
    result = []
    for i in range(bits_num):
        seed = step_forward(seed, tap_pos)
        result.append(seed[0])
    return "".join(result)


n = 4
seed = "1101"
tap_pos = 1

print(LFSR(4, init_seed=seed, tap_pos=1, bits_num=40))
```

test:

result:

```
/usr/bin/pythonw /Users/apple/PycharmProjects/
0110110110110110110110110110110110110110
```