Gift Cipher

https://youtu.be/YInBFvhLsQU

Gift 암호란?

Gift 암호 기본 개념 설명

Gift 암호 구현

Gift 암호란?

Gift 암호란?

SPN구조의 경량 암호

Gift 암호 기본 개념 설명

2 Specifications

In this work, we propose two versions of GIFT, GIFT-64-128 is a 28-round SPN cipher and GIFT-128-128 is a 40-round SPN cipher, both versions have a key length of 128-bit. For short, we call them GIFT-64 and GIFT-128 respectively.

GIFT can be perceived in three different representations. In this paper, we adopt the classical 1D representation, describing the bits in a row like PRESENT. It can also be described in bitslice 2D, a rectangular array like RECTANGLE [48] (see Appendix A), and even in 3D cuboid like 3D [34] (see Appendix B).

Round function. Each round of GIFT consists of 3 steps: SubCells, PermBits, and AddRoundKey, which is conceptually similar to wrapping a gift:

1. Put the content into a box (SubCells);
2. Wrap the ribbon around the box (PermBits);
3. Tie a knot to secure the content (AddRoundKey).

Figure 1 illustrates 2 rounds of GIFT-64.

Gift 암호 기본 개념 설명

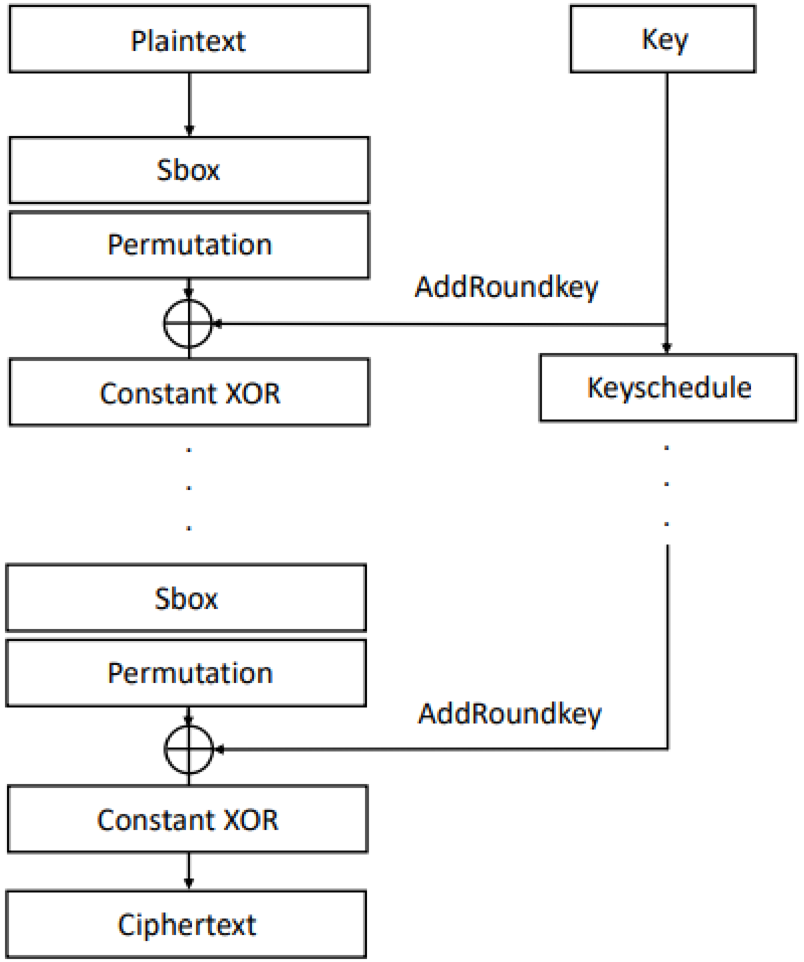


Figure 2. Encryption process of GIFT block cipher.

Gift 암호 기본 개념 설명

2.2. *GIFT Block Cipher*

The GIFT block cipher is a symmetric key cryptography using the Substitution Permutation Network (SPN) method. There are GIFT-64/128 (64-bit block and 128-bit key) and GIFT-128/128 (128-bit block and 128-bit key). In the GIFT block cipher, each round performs four steps: Sbox, Permutation, AddRoundKey and Constant XOR. The encryption operation of GIFT block cipher is described in Figure 2.

2.2.1. Sbox of GIFT Block Cipher

The n-bit block ($n = 64, 128$) is split into 4 bits and becomes the input value of the 4-bit Sbox. The Sbox of GIFT block cipher is given in Table 3.

Table 3. Sbox of GIFT block cipher.

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

Gift 암호 기본 개념 설명

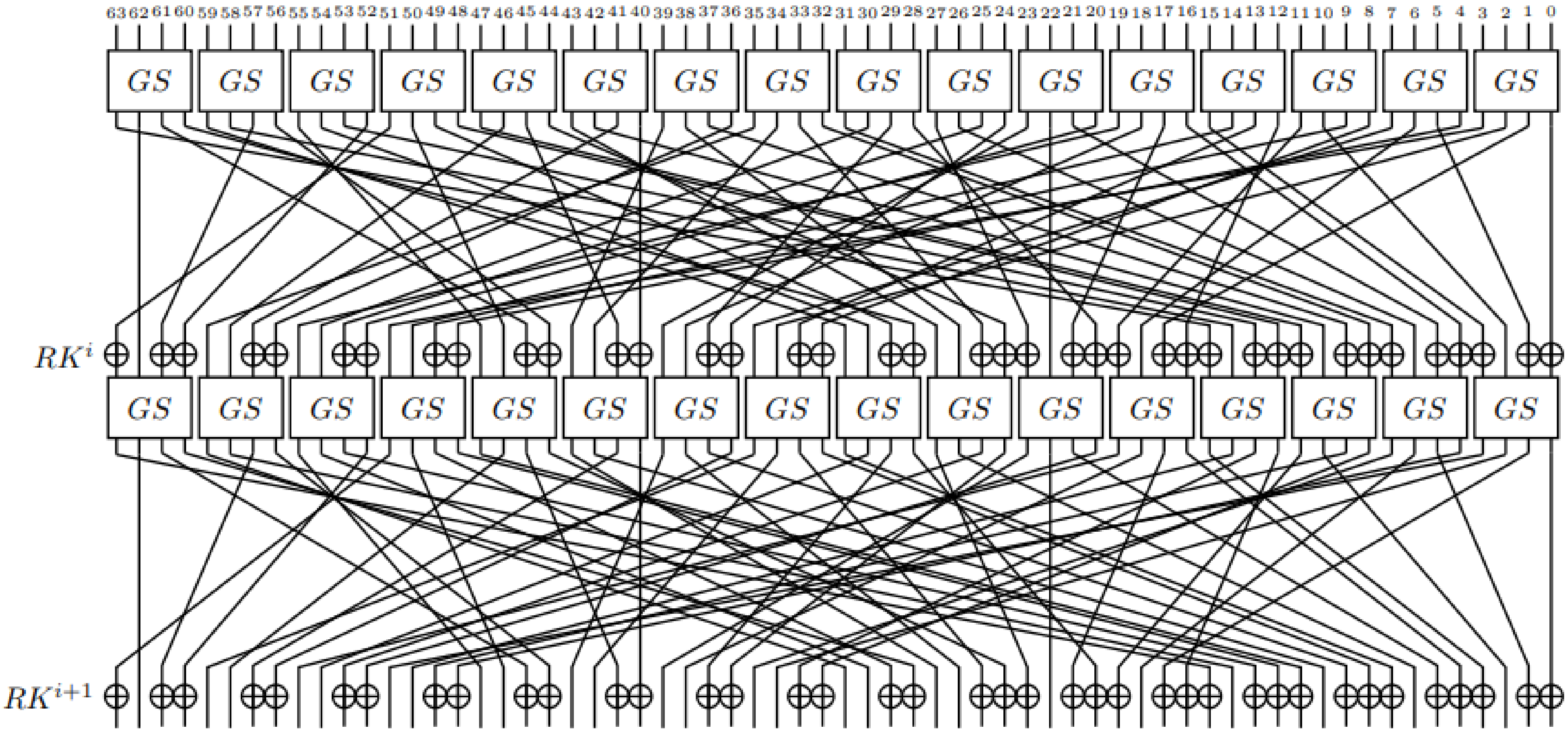


Fig. 1. 2 Rounds of `GIFT-64`.

Gift 암호 기본 개념 설명

2.2.2. Permutation of GIFT Block Cipher

In the permutation, GIFT-64/128 replaces the $P_{64}(i)$-th bit of block B with the i-th bit of block B. Details on the permutation of GIFT-64/128 are shown in Table 4. In this paper, detailed Table on permutation of GIFT-128/128 is omitted. Permutation Table of GIFT-128/128 can be found in [4].

Table 4. Permutation of GIFT-64 bit.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{64}(i)$	0	17	34	51	48	1	18	35	23	49	2	19	16	33	50	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_{64}(i)$	4	21	38	55	52	5	22	39	36	53	6	23	20	37	54	7
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P_{64}(i)$	8	25	42	59	56	9	26	43	40	57	10	27	24	41	58	11
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P_{64}(i)$	12	29	46	63	60	13	30	47	44	61	14	31	28	45	62	15

2.2.3. AddRoundkey of GIFT Block Cipher

In the GIFT-64/128 block cipher, k_0 and k_1 (32-bit total) are selected from the key ($K = k_7, ..., k_0$). k_0 and k_1 are used as U and V of the round key as follows, $RK = U||V = u_{15}...u_0||v_{15}...v_0$ ($U = k_1, V = k_0$). The round key is exclusive-ored with the block B, where U is XORed to b_{4i+1} and V is XORed to b_{4i}.

$$b_{4i+1} \leftarrow b_{4i+1} \oplus u_i, \;\; b_{4i} \leftarrow b_{4i} \oplus v_i, \;\; i = 0, ..., 15 \tag{6}$$

Gift 암호 기본 개념 설명

2.2.4. Constant XOR of GIFT Block Cipher

Round constants C given in Table 5 are used in GIFT-64/128 and GIFT-128/128 block ciphers. Single bit and round constants ($C = c_5c_4c_3c_2c_1c_0$) are XORed to block B as in Equation (8).

$$
\begin{aligned}
&b_{n-1} \leftarrow b_{n-1} \oplus 1, \\
&b_{23} \leftarrow b_{23} \oplus c_5,\ b_{19} \leftarrow b_{19} \oplus c_4,\ b_{15} \leftarrow b_{15} \oplus c_3, \\
&b_{11} \leftarrow b_{11} \oplus c_2,\ b_7 \leftarrow b_7 \oplus c_1,\ b_3 \leftarrow b_3 \oplus c_0.
\end{aligned} \tag{8}
$$

Table 5. Round constants C.

Rounds	Constants C															
1 to 16	01	03	07	0F	1F	3E	3D	3B	37	2F	1E	3C	39	33	27	0E
17 to 32	1D	3A	35	2B	16	2C	18	30	21	02	05	0B	17	2E	1C	38
33 to 48	31	23	06	0D	1B	36	2D	1A	34	29	12	24	08	11	22	04

2.2.5. Keyschedule of GIFT Block Cipher

In GIFT-64/128 and GIFT-128/128 block ciphers, the Keyschedule updates key ($K = k_7, ..., k_0$) and extracts the round key from the updated key K. The Keyschedule is shown in Equation (9). The notation ($\ggg i$) denotes a right rotation operation (i-bit).

$$
k_7||k_6||...||k_1||k_0 \leftarrow k_1 \ggg 2||k_0 \ggg 12||...||k_3||k_2, \tag{9}
$$

Gift 암호 기본 개념 설명

2.2.4. Constant XOR of GIFT Block Cipher

Round constants C given in Table 5 are used in GIFT-64/128 and GIFT-128/128 block ciphers. Single bit and round constants ($C = c_5c_4c_3c_2c_1c_0$) are XORed to block B as in Equation (8).

$$\begin{aligned} &b_{n-1} \leftarrow b_{n-1} \oplus 1, \\ &b_{23} \leftarrow b_{23} \oplus c_5,\ b_{19} \leftarrow b_{19} \oplus c_4,\ b_{15} \leftarrow b_{15} \oplus c_3, \\ &b_{11} \leftarrow b_{11} \oplus c_2,\ b_7 \leftarrow b_7 \oplus c_1,\ b_3 \leftarrow b_3 \oplus c_0. \end{aligned} \tag{8}$$

Table 5. Round constants C. $b(3*i+3+i) = b(3*i+3+i) \oplus C_i$

Rounds	**Constants *C***															
1 to 16	01	03	07	0F	1F	3E	3D	3B	37	2F	1E	3C	39	33	27	0E
17 to 32	1D	3A	35	2B	16	2C	18	30	21	02	05	0B	17	2E	1C	38
33 to 48	31	23	06	0D	1B	36	2D	1A	34	29	12	24	08	11	22	04

2.2.5. Keyschedule of GIFT Block Cipher

In GIFT-64/128 and GIFT-128/128 block ciphers, the Keyschedule updates key ($K = k_7, \ldots, k_0$) and extracts the round key from the updated key K. The Keyschedule is shown in Equation (9). The notation ($\ggg i$) denotes a right rotation operation (i-bit).

$$k_7||k_6||\ldots||k_1||k_0 \leftarrow k_1 \ggg 2||k_0 \ggg 12||\ldots||k_3||k_2, \tag{9}$$

$k_0 = 16$

$\downarrow$

$k_7k_6k_5k_4k_3k_2k_1k_0 = k_1k_0k_7k_6k_5k_4k_3k_2$

$$\boxed{\begin{aligned} k_7 &= k_1 \ggg 2 \\ k_6 &= k_0 \ggg 12 \end{aligned}}$$

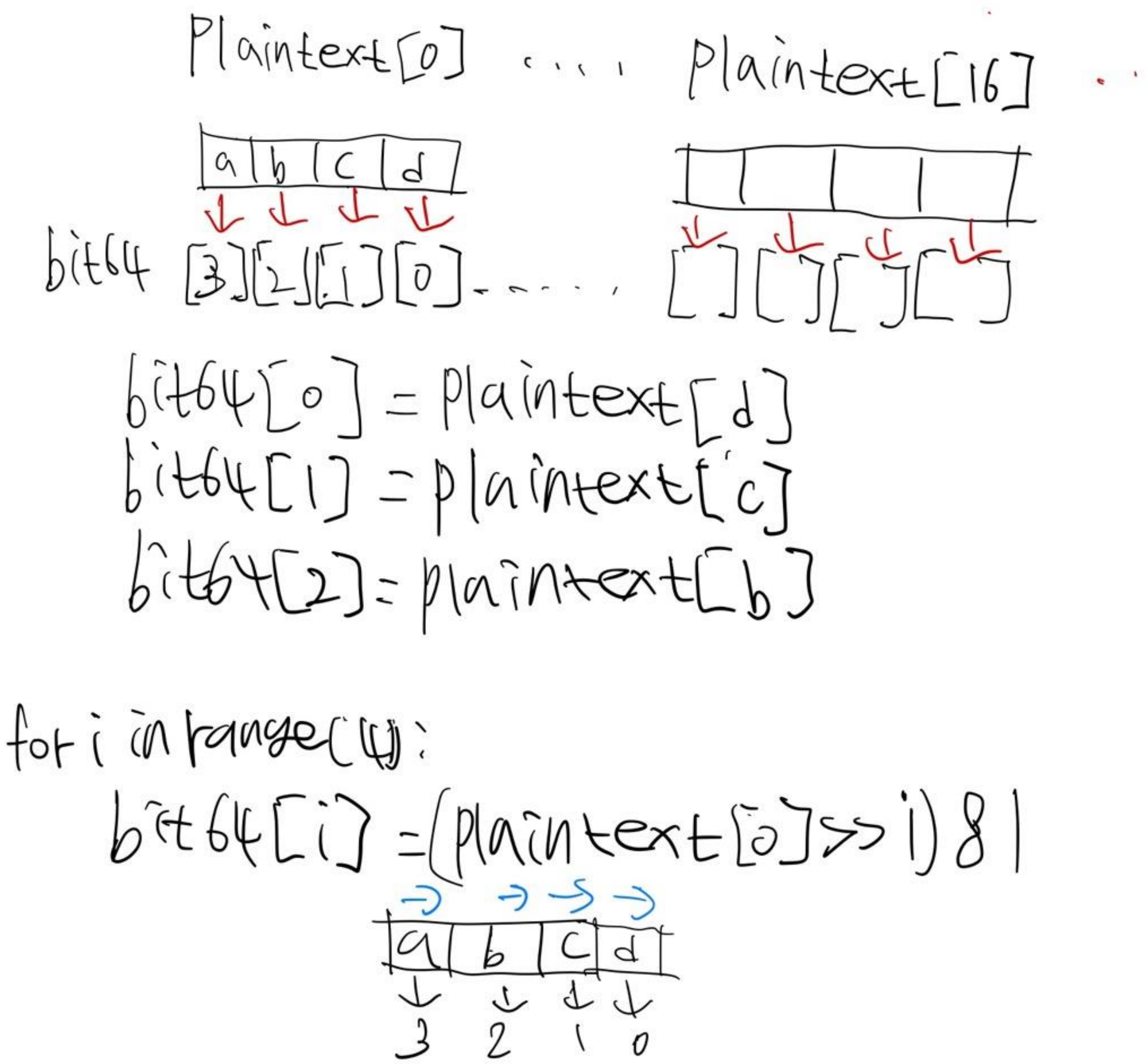
Gift 암호 구현

```python
GIFT_S=[1, 10, 4, 12, 6, 15, 3, 9, 2, 13, 11, 7, 5, 0, 8, 14]  # 16

GIFT_P=[
    0, 17, 34, 51, 48, 1, 18, 35, 32, 49, 2, 19, 16, 33, 50, 3,
    4, 21, 38, 55, 52, 5, 22, 39, 36, 53, 6, 23, 20, 37, 54, 7,
    8, 25, 42, 59, 56, 9, 26, 43, 40, 57, 10, 27, 24, 41, 58, 11,
    12, 29, 46, 63, 60, 13, 30, 47, 44, 61, 14, 31, 28, 45, 62, 15
]

GIFT_RC =[0x01, 0x03, 0x07, 0x0F, 0x1F, 0x3E, 0x3D, 0x3B, 0x37, 0x2F,
    0x1E, 0x3C, 0x39, 0x33, 0x27, 0x0E, 0x1D, 0x3A, 0x35, 0x2B,
    0x16, 0x2C, 0x18, 0x30, 0x21, 0x02, 0x05, 0x0B, 0x17, 0x2E,
    0x1C, 0x38, 0x31, 0x23, 0x06, 0x0D, 0x1B, 0x36, 0x2D, 0x1A,
    0x34, 0x29, 0x12, 0x24, 0x08, 0x11, 0x22, 0x04]

plaintext = [0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xa,0xb,0xc,0xd,0xe,0xf]
key = [0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xa,0xb,0xc,0xd,0xe,0xf,0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xa,0xb,0xc,0xd,0xe,0xf]


def Sbox(plaintext):
    for i in range(16):
        plaintext[i] = GIFT_S[plaintext[i]]

def print_state(plaintext):
    print("state : 0x",end='')
    for i in range(16):
        temp = hex(plaintext[15-i])
        y = temp.replace("0x","")
        print(y, end='')
    print()

def print_state_bit(plaintext):
    print("state: 0b", end='')
    for i in range(64):
        print(plaintext[63-i],end='')
    print()

def print_key(plaintext):
    print("key : 0x", end='')
    for i in range(32):
        temp = hex(plaintext[15 - i])
        y = temp.replace("0x", "")
        print(y, end='')
    print()
```

Gift 암호 구현

```python
def hex_to_key(key):

    bit128 = [0 for i in range(128)]

    for i in range(32):
        for j in range(4):
            bit128[4*i+j] = (key[i]>>j) & 1

    return bit128

def print_key_bit(key):
    print("key: 0b", end='')
    for i in range(128):
        print(key[127 - i], end='')
    print()

def hex_to_bit(plaintext):

    bit64 = [0 for i in range(64)]

    for i in range(16):
        for j in range(4):
            bit64[4*i+j] = (plaintext[i] >> j) & 1

    return bit64

def permutation(bit64):
    new_64 = [0 for i in range(64)]
    for i in range(64):
        new_64[GIFT_P[i]] = bit64[i]

    return new_64

def AddRoundkey(plaintext, roundkey): # plaintext = new_64
    v = roundkey[0:16]
    u = roundkey[16:32]

    for i in range(16):
        plaintext[4*i+1] = plaintext[4*i+1]^u[i]
    for i in range(16):
        plaintext[4*i] = plaintext[4*i]^v[i]
```

Gift 암호 구현



Plaintext[0] .... Plaintext[16] ..
a b c d
bit64 [3][2][1][0] .....
bit64[0] = plaintext[d]
bit64[1] = plaintext[c]
bit64[2] = plaintext[b]
for i in range(4):
bit64[i] = (plaintext[0]>>i) & 1
a b c d
3 2 1 0

Gift 암호 구현

```python
def keyschedule(key):

    k = [0 for i in range(128)]  # k = k[127] k[126] k[125] ... k[4] k[3] k[2] k[1] k[0]
    k[0:16] = key[32:48]
    k[16:32] = key[48:64]
    k[32:48] = key[64:80]
    k[48:64] = key[80:96]
    k[64:80] = key[96:112]
    k[80:96] = key[112:128]
    k[96:112] = key[0:16]
    k[112:128] = key[16:32]

    temp = [0 for i in range(128)] #temp[15, 14, 13, 12, ... 0]
    temp[96:112] = k[96:112]
    k[96] = temp[108]
    k[97] = temp[109]
    k[98] = temp[110]
    k[99] = temp[111]

    for i in range(12):
        k[100+i] = temp[96+i]
    temp[112:128] = k[112:128]

    for i in range (14):
        k[112+i] = temp[114+i]
    k[126] = temp[112]
    k[127] = temp[113]

    return k
```

Gift 암호 구현

```python
def bit_to_hex(plaintext):
    out = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    #for i in range(16):
        #out[0] = plaintext[0] * 1 + plaintext[1] * 2 + plaintext[2] * 4 + plaintext[3] * 8
        #out[1] = plaintext[4] * 1 + plaintext[5] * 2 + plaintext[6] * 4 + plaintext[7] * 8
        #out[2] = plaintext[8] * 1 + plaintext[9] * 2 + plaintext[10] * 4 + plaintext[11] * 8
    for i in range(16):
        out[i] = out[i] + plaintext[4*i] * 1
        out[i] = out[i] + plaintext[4*i+1] * 2
        out[i] = out[i] + plaintext[4*i+2] * 4
        out[i] = out[i] + plaintext[4*i+3] * 8

    return out




def AddconstantXOR(plaintext,GIFT_RC, count):
    constants = GIFT_RC[count]
    c = [0, 0, 0, 0, 0, 0, 0, 0]
    for i in range(8):
        c[i] = (constants >> i) & 1
    #for i in range()
    #c = constants[i] >> 2 &1
    for i in range(6):
        plaintext[3*i+3+i] = plaintext[3*i+3+i] ^ c[i]
    plaintext[63] = plaintext[63]^1
```

Gift 암호 구현

```python
def GIFT_Enc(plaintext, key):

    count = 0

    print_key(key)
    print_state(plaintext)
    bit128 = hex_to_key(key)

    for i in range(28):
        # S-box
        Sbox(plaintext)
        print_state(plaintext)

        # hex to bit
        bit64 = hex_to_bit(plaintext)
        # print_state_bit(bit64)

        new_64 = permutation(bit64)
        # print_state_bit(new_64)
        print_hex(new_64)

        # bit128 = hex_to_key(key)
        # print_key_bit(bit128)

        AddRoundkey(new_64, bit128[0:32])
        print_hex(new_64)

        # print_key_bit(bit128)
        k = keyschedule(bit128)
        # print_key_bit(k)
        bit128 = k
        # print_hex(new_64)

        AddconstantXOR(new_64,GIFT_RC, count)
        count = count+1
        print_hex(new_64)

        out = bit_to_hex(new_64)
        plaintext = out

        print()

GIFT_Enc(plaintext, key)
```

Gift 암호 구현

```
state: 0xa90a9213dc912a97

state : 0xbd1bd4ac05da4bd9
state: 0x9c9f3c99b16cde41
state: 0x9e8c3e89b34fdc61
state: 0x1e8c3e89b3c7d469

state : 0xa825c82d7c59063d
state: 0x0cdb642488d0bb57
state: 0x3ec85634baf38977
state: 0xbec85634ba7309ff

state : 0x7852f3c67b9c1dee
state: 0x56f82ffdf79f0806
state: 0x76ea2dfef79d2a15
state: 0xf6ea2dfef7952295

state : 0xe38b40e8e9df44df
state: 0xecddb8fc24e69295
state: 0xdcefaadf34c4a0a6
state: 0x5cefaadf344428a6

state : 0xf58ebb0ec66642b3
state: 0xca67f7483b62c8e3
state: 0xea75f54b3b60eaf0
state: 0x6a75f54b3b686af0

state : 0x3b9fef67c7323be1
state: 0xb672f71bbe2d1fe3
state: 0x8640e538ae0f2dd0
state: 0x0640e538ae872dd0

state : 0x13618fc2b8294001
state: 0x228419205e903cb1
state: 0x20a51b225c813e83
state: 0xa0a51b225c013e8b

state : 0xb1bfa744f51ac827
state: 0x92d4f734343abccb
state: 0xa0f5c536062b8ef9
state: 0x20f5c536062b8e79

state : 0x41ef5fc31347289d
state: 0xc629d95e6f783511
state: 0xc408db5c6d693723
state: 0x4408db5c6d693f2b

state : 0x661207f5303dce47
state: 0x73962d64241e473d
state: 0x41b71f66160f750f
state: 0xc1b71f66160ff587


종료 코드 0(으)로 완료된 프로세스
```

Q & A