# DEFAULT

송민호

유튜브: https://youtu.be/iSyGotjngOE

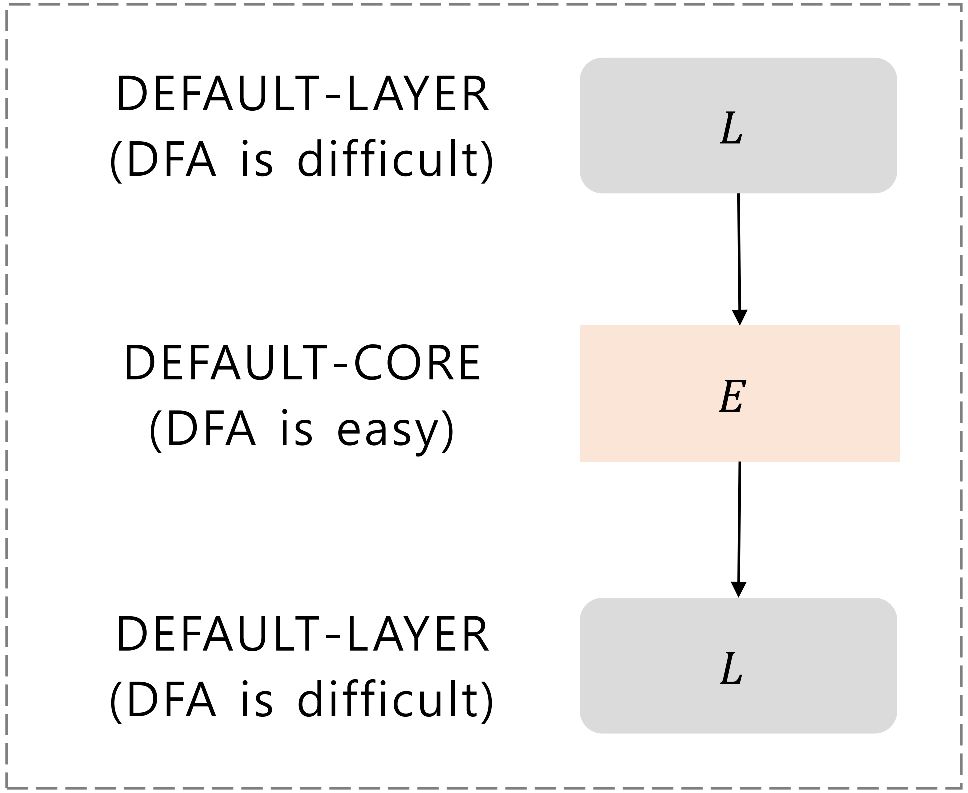
# DEFAULT 개요

- GIFT의 기본 구조를 따르는 경량 대칭키 암호

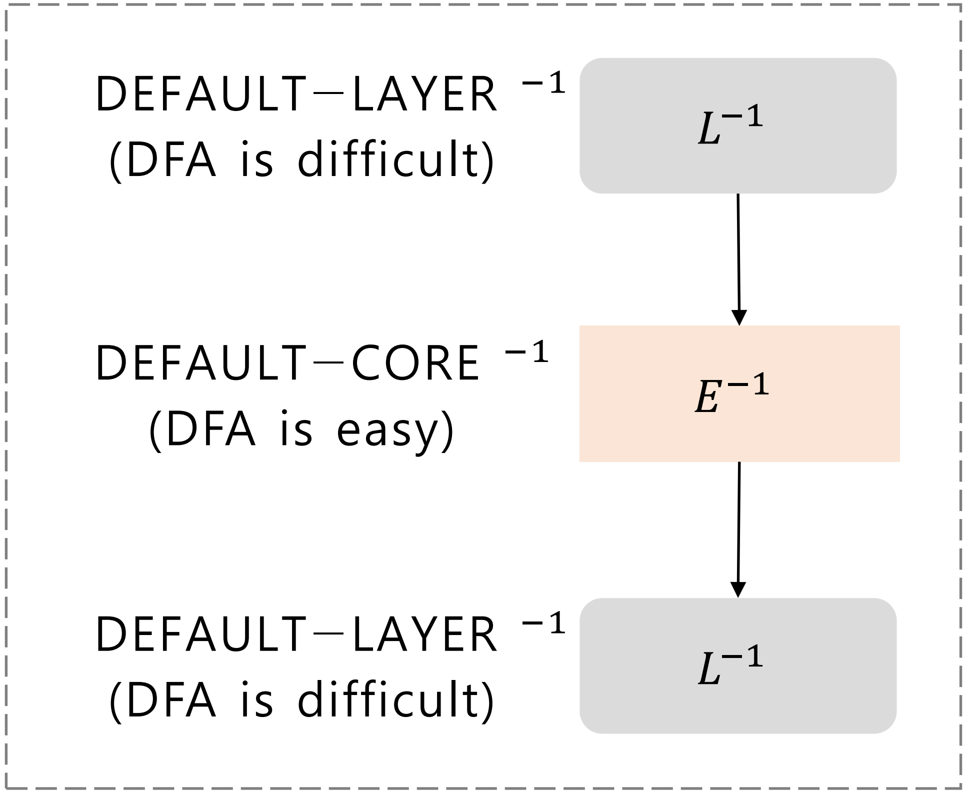
- DFA(Differential Fault Analysis) 공격에 저항할 수 있도록 설계됨
  - 선형 구조의 Sbox 사용

- DEFAULT-LAYER, DEFAULT-CORE 두 개로 이루어져 있음
  - LAYER → CORE → LAYER 순으로 진행

# DEFAULT 전체 구조

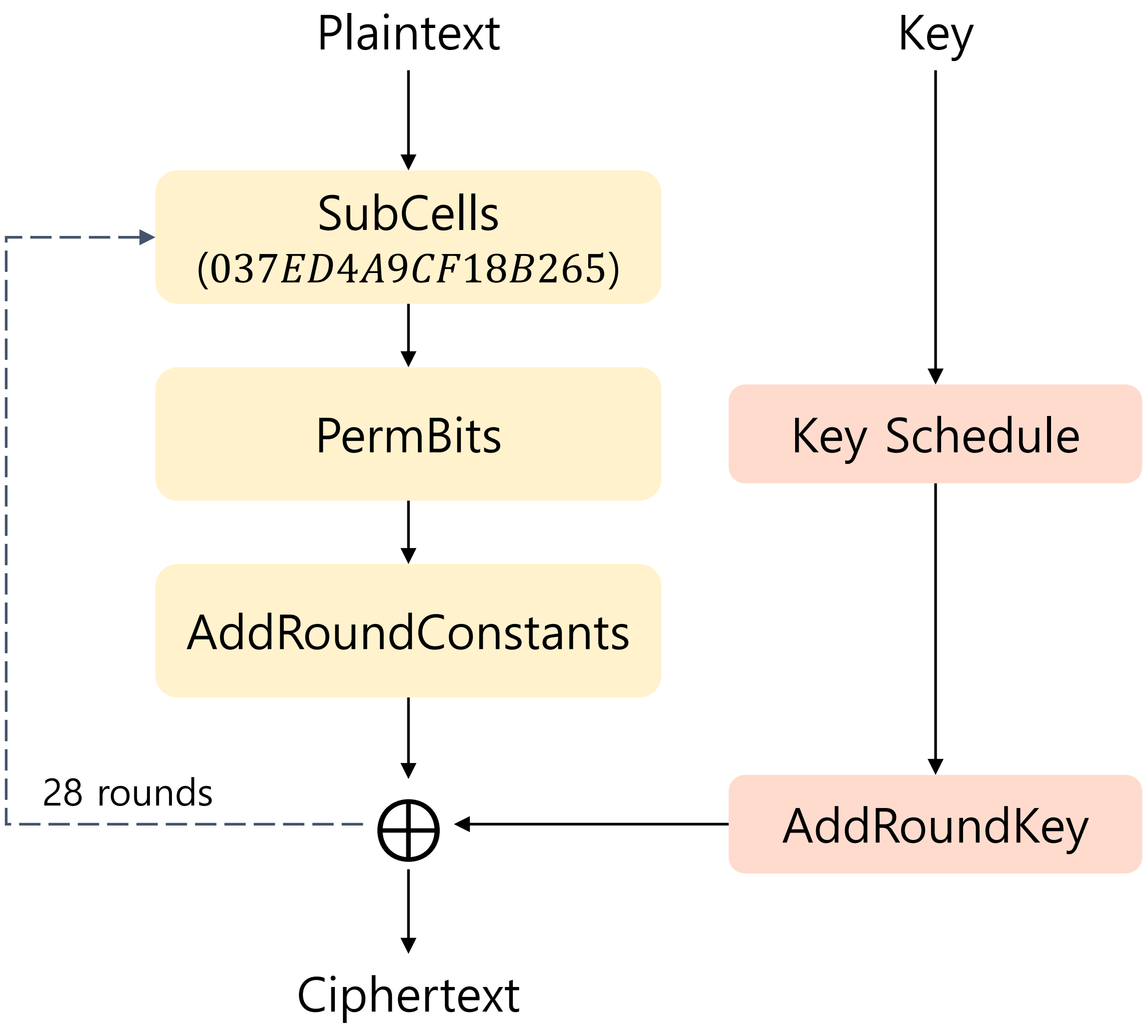



Encryption

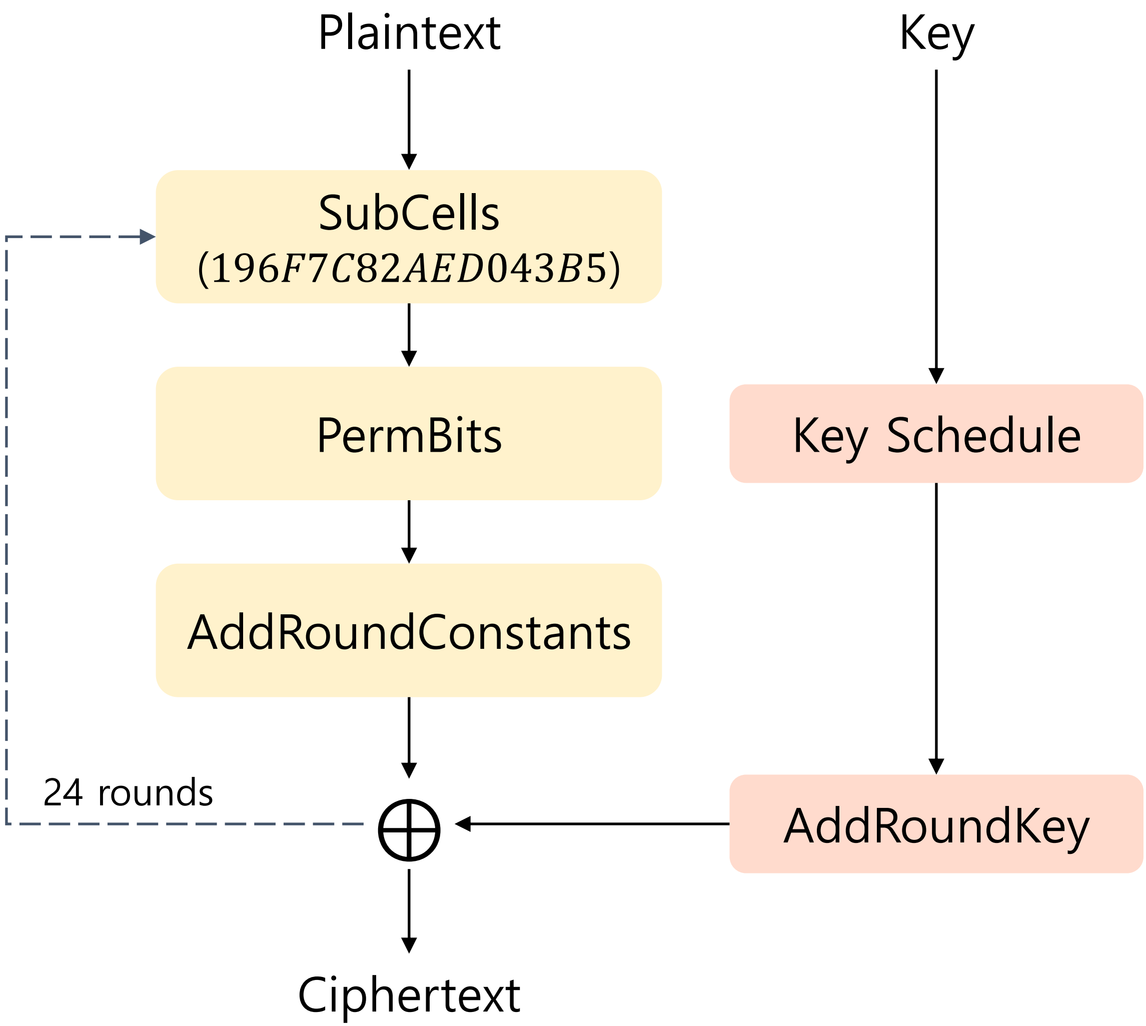



Decryption

# DEFAULT 전체 구조 - LAYER, CORE




**DEFAULT-LAYER**




**DEFAULT-CORE**

# DEFAULT 구조 - SubCells

- DEFAULT-LAYER
  - 4-bit LS Sbox 사용 (S = 037ED4A9CF18B265)
  - $w_i \leftarrow S(w_i),\ \forall_i \in \{0, \ldots, 31\}$
- DEFAULT-CORE
  - 4-bit non_LS Sbox 사용 (S = 196F7C82AED043B5)
  - $w_i \leftarrow S(w_i),\ \forall_i \in \{0, \ldots, 31\}$
- LS Sbox를 통해 DFA 공격에 내성을 가짐

# DEFAULT 구조 - Permutation

- Permutation에 사용되는 $P_{128}$은 GIFT-128과 같음

$$P_{128}(i):\ b_{P_{128}(i)} \leftarrow b_i, \qquad \forall_i \in \{0, \dots, 127\}$$

**Table 13:** Specifications of GIFT-128 Bit Permutation.

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $P_{128}(i)$ | 0 | 33 | 66 | 99 | 96 | 1 | 34 | 67 | 64 | 97 | 2 | 35 | 32 | 65 | 98 | 3 |
| $i$ | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| $P_{128}(i)$ | 4 | 37 | 70 | 103 | 100 | 5 | 38 | 71 | 68 | 101 | 6 | 39 | 36 | 69 | 102 | 7 |
| $i$ | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| $P_{128}(i)$ | 8 | 41 | 74 | 107 | 104 | 9 | 42 | 75 | 72 | 105 | 10 | 43 | 40 | 73 | 106 | 11 |
| $i$ | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| $P_{128}(i)$ | 12 | 45 | 78 | 111 | 108 | 13 | 46 | 79 | 76 | 109 | 14 | 47 | 44 | 77 | 110 | 15 |
| $i$ | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| $P_{128}(i)$ | 16 | 49 | 82 | 115 | 112 | 17 | 50 | 83 | 80 | 113 | 18 | 51 | 48 | 81 | 114 | 19 |
| $i$ | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| $P_{128}(i)$ | 20 | 53 | 86 | 119 | 116 | 21 | 54 | 87 | 84 | 117 | 22 | 55 | 52 | 85 | 118 | 23 |
| $i$ | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| $P_{128}(i)$ | 24 | 57 | 90 | 123 | 120 | 25 | 58 | 91 | 88 | 121 | 26 | 59 | 56 | 89 | 122 | 27 |
| $i$ | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| $P_{128}(i)$ | 28 | 61 | 94 | 127 | 124 | 29 | 62 | 95 | 92 | 125 | 30 | 63 | 60 | 93 | 126 | 31 |

# DEFAULT 구조 - AddRoundConstant

- Round Constants도 GIFT-128과 같음

  - A single bit “1” and a 6-bit Round Constant $C = c_5c_4c_3c_2c_1c_0$ are XORed into the cipher state at bit position 127, 23, 19, 15, 11, 7 and 3.

$$w_{127} = w_{127} \oplus 1, \qquad w_{23} = w_{23} \oplus c_5, \qquad w_{19} = w_{19} \oplus c_4$$

$$w_{15} = w_{15} \oplus c_3, \qquad w_{11} = w_{11} \oplus c_2, \qquad w_7 = w_7 \oplus c_1, \qquad w_3 = w_3 \oplus c_0$$

**Table 2:** Round constants for DEFAULT

| | Round Constants | ♯ |
|---|---|---|
| DEFAULT-CORE | 1, 3, 7, 15, 31, 62, 61, 59, 55, 47, 30, 60, 57, 51, 39, 14, 29, 58, 53, 43, 22, 44, 24, 48, 33, 2, 5, 11 | 28 |
| DEFAULT-LAYER | 1, 3, 7, 15, 31, 62, 61, 59, 55, 47, 30, 60, 57, 51, 39, 14, 29, 58, 53, 43, 22, 44, 24, 48 | 24 |

# DEFUALT 구조 - Key

- KeySchedule
  - 128-bit master key $K$를 사용하여 4개의 128-bit subkey 생성($K_0, K_1, K_2, K_3$)

  - $K_0 = K$
  - $K_{i+1} = R'(R'(R'(R'(K_i))))$ for $i \in \{0, 1, 2\}$

  - $R'$은 라운드 함수(no AddRoundKey, changed AddRoundConstant)
  - $R' = SubCells \rightarrow Permutation \rightarrow K_i[127] \oplus 1$

- AddRoundKey
  - $b_i \leftarrow b_i \oplus k_i^j, \forall_i \in \{0, \ldots, 127\}$

# 구현

default-cipher-public-code / **default_128_enc_dec_ref.py**

Edit •••

```python
# Default cipher reference source code
# Written by Anubhab Baksi <anubhab001@e.ntu.edu.sg>
# Hosted at https://bitbucket.org/anubhab001/default-cipher-public-code

LS_sbox = 0, 3, 7, 14, 13, 4, 10, 9, 12, 15, 1, 8, 11, 2, 6, 5
LS_inv_sbox = 0, 10, 13, 1, 5, 15, 14, 2, 11, 7, 6, 12, 8, 4, 3, 9
non_LS_sbox = 1, 9, 6, 15, 7, 12, 8, 2, 10, 14, 13, 0, 4, 3, 11, 5
non_LS_inv_sbox = 11, 0, 7, 13, 12, 15, 2, 4, 6, 1, 8, 14, 5, 10, 9, 3


player = 0, 33, 66, 99, 96, 1, 34, 67, 64, 97, 2, 35, 32, 65, 98, 3, 4, 37, 70, 103, 100, 5, 38, 71, 68, 101, 6, 39, 36, 69, 102, 7, 8, 41, 74, 107, 104, 9, 42, 75, 72, 105, 10, 43, 40, 73, 106, 11, 12, 45, 7

inv_player = 0, 5, 10, 15, 16, 21, 26, 31, 32, 37, 42, 47, 48, 53, 58, 63, 64, 69, 74, 79, 80, 85, 90, 95, 96, 101, 106, 111, 112, 117, 122, 127, 12, 1, 6, 11, 28, 17, 22, 27, 44, 33, 38, 43, 60, 49, 54, 59,

RC = 1, 3, 7, 15, 31, 62, 61, 59, 55, 47, 30, 60, 57, 51, 39, 14, 29, 58, 53, 43, 22, 44, 24, 48, 33, 2, 5, 11, 1, 3, 7, 15, 31, 62, 61, 59, 55, 47, 30, 60, 57, 51, 39, 14, 29, 58, 53, 43, 22, 44, 24, 48, 1,

def sbox_layer(pt, sbox):
    ct = [None]*len(pt)
    for i in range(len(pt)):
        ct[i] = sbox[pt[i]]
    return ct[:]


def bits_to_nibble(x):
    assert len(x) == 4
    y = x[0]*1 + x[1]*2 + x[2]*4 + x[3]*8
    return y

def bits_to_nibbles(x):
    y = []
    for i in range(len(x)//4):
        z = x[4*i : 4*i + 4]
        y.append(bits_to_nibble(z))
    return y
def nibble_to_bits(x):
    y = []
    z = [zz == '1' for zz in (bin(x)[2:]).zfill(4)][::-1]
    y.extend(z)
    return y

```

# 구현

```python
def nibbles_to_bits(x):
    y = map(nibble_to_bits, x)
    return [item for sublist in y for item in sublist]

def nibble_encoding(a):
    assert [type(aa) == type(True) for aa in a]
    a_nibbles = bits_to_nibbles(a)

    b = []
    for aa in a_nibbles:
        b.append(bits_to_nibble(nibble_to_bits(aa)[::-1]))

    return nibbles_to_bits(b)

def key_update(key):
    ## key update

    # Right rotate the entire key 20 steps
    temp_key = nibbles_to_bits(key)
    temp_key = temp_key[20:] + temp_key[:20]
    # Right rotate most significant 16 bits 1 step
    temp_key_16bit = temp_key[-16:]
    temp_key_16bit = nibble_encoding(temp_key_16bit)
    temp_key_16bit = temp_key_16bit[1:] + temp_key_16bit[:1]
    temp_key_16bit = nibble_encoding(temp_key_16bit)
    temp_key = temp_key[:-16] + temp_key_16bit

    assert len(temp_key) == 128
    key = bits_to_nibbles(temp_key)

    return key

```

# 구현

```python
def add_round_constants(state, rc):
    state[3] ^= (rc & 1)
    state[7] ^= ((rc >> 1) & 1)
    state[11] ^= ((rc >> 2) & 1)
    state[15] ^= ((rc >> 3) & 1)
    state[19] ^= ((rc >> 4) & 1)
    state[23] ^= ((rc >> 5) & 1)
    state[127] ^= True
    return state

def perm_layer(x, player):
    assert len(x) == 128
    y = [None]*len(x)
    for i in range(len(x)):
        y[player[i]] = x[i]
    return y[:]

def add_key(pt, key):
    return [p^k for p, k in zip(pt, key)]

def encryptNonLS(round_keys, pt, no_of_rounds=28, round_starting=0):

    for r in range(no_of_rounds):

        # sbox
        pt = sbox_layer(pt=pt, sbox=non_LS_sbox)

        # permutation
        pt = nibbles_to_bits(pt)
        pt = perm_layer(x=pt, player=player)

        # round constants
        pt = add_round_constants(pt, rc=RC[r + round_starting])
        pt = bits_to_nibbles(pt)

        # round key add
        round_key = round_keys[r + round_starting + 1]
        pt = add_key(pt, round_key)

    return pt
```

# Q & A