Cryptanalysis
(암호분석)

Chapter 7 – Part 2

2020.6

Contents

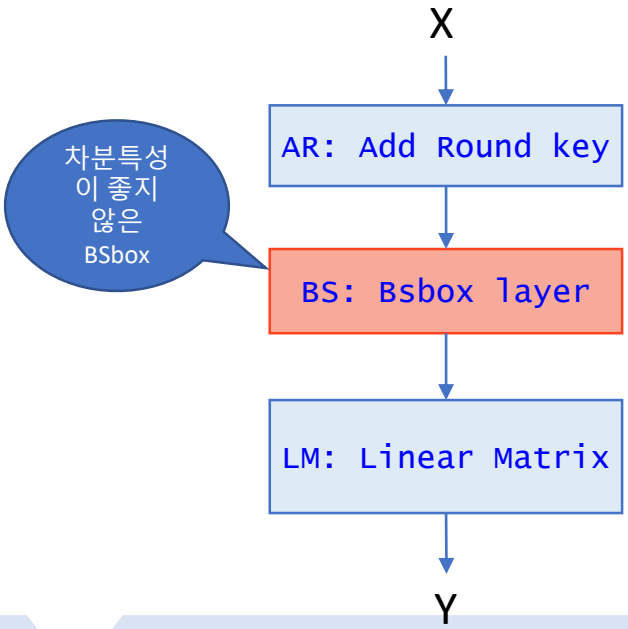
- Differential probability
- Differential Cryptanalysis (DC)
- Variants of DC

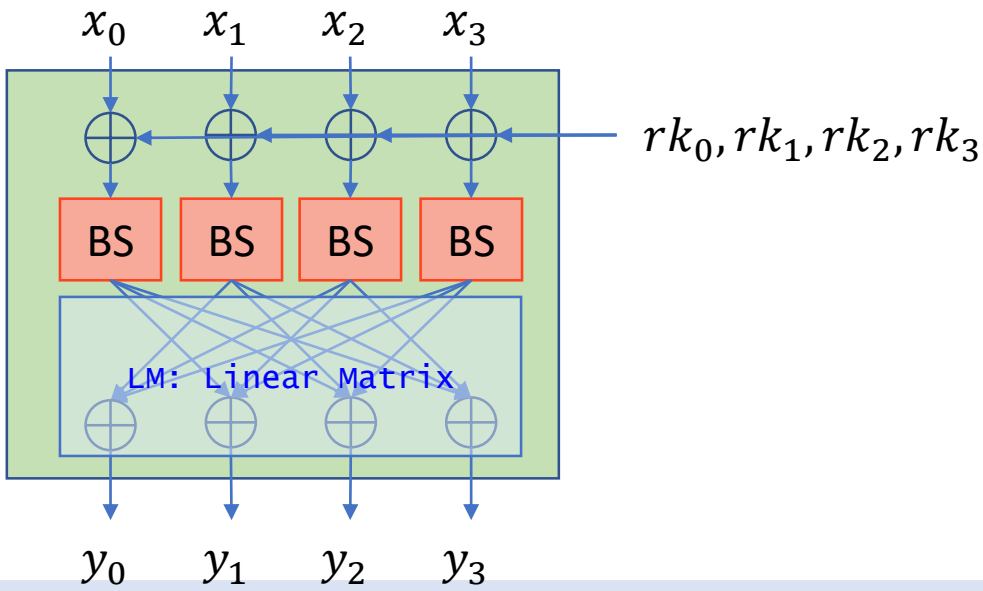
이번
슬라이드

블록암호 BC20R

Bad Cipher 2020
(Round reduced)

- BC20R의 라운드 함수
  - AR: 라운드 키 XOR (TC20과 동일함)
  - BS_Layer: 비선형 Bsbox (8비트) 4개 적용
  - LM_Layer: 선형 함수 (TC20과 동일함)





블록암호 BC20R

- BSbox 정의

```
BSbox = [
  0x13,0x31,0xba,0x03,0xa9,0xb8,0x32,0x88,0x23,0xa8,0x33,0x9b,0xb9,0x28,0x91,0x98,
  0x29,0x3b,0x3a,0x38,0x09,0x89,0x01,0x80,0x83,0x10,0xb0,0x19,0xab,0x12,0x02,0x90,
  0xb3,0x30,0x08,0x11,0xbb,0x81,0x9a,0xa3,0xb2,0xa2,0x22,0x8b,0x20,0x1b,0x2a,0x82,
  0x8a,0x2b,0x0a,0x1a,0xaa,0x93,0x00,0x21,0x99,0xb1,0x18,0xa0,0x0b,0xa1,0x39,0x92,
  0x53,0x71,0xfa,0x43,0xe9,0xf8,0x72,0xc8,0x63,0xe8,0x73,0xdb,0xf9,0x68,0xd1,0xd8,
  0x69,0x7b,0x7a,0x78,0x49,0xc9,0x41,0xc0,0xc3,0x50,0xf0,0x59,0xeb,0x52,0x42,0xd0,
  0xf3,0x70,0x48,0x51,0xfb,0xc1,0xda,0xe3,0xf2,0xe2,0x62,0xcb,0x60,0x5b,0x6a,0xc2,
  0xca,0x6b,0x4a,0x5a,0xea,0xd3,0x40,0x61,0xd9,0xf1,0x58,0xe0,0x4b,0xe1,0x79,0xd2,
  0x44,0x37,0x0e,0x6c,0x3f,0x1f,0x8e,0x76,0x7d,0x26,0x2f,0x94,0x5c,0x0c,0x66,0x17,
  0x1d,0x97,0x14,0xb6,0xac,0xcf,0x87,0x06,0x6f,0xae,0xc7,0x5f,0x24,0xc6,0x96,0xe4,
  0xc4,0xe5,0xec,0x34,0x4e,0x0f,0x74,0xbe,0xff,0x0d,0x9d,0xf5,0xa6,0x84,0x2e,0x4d,
  0xdf,0x05,0x6d,0x45,0x54,0xde,0x5e,0x95,0xbc,0x3e,0xad,0x46,0x47,0x7e,0x7f,0x36,
  0xd4,0xf7,0x9f,0xbd,0x7c,0x56,0x1c,0x3d,0x27,0xa7,0x25,0x67,0xaf,0xed,0xa4,0x57,
  0x8d,0x4f,0xf6,0xfd,0x85,0x1e,0xb5,0x65,0xa5,0x6e,0x77,0xe6,0xee,0x8f,0xd6,0x3c,
  0x55,0xcd,0x07,0xb7,0xe7,0x64,0xcc,0x2d,0x75,0xb4,0x5d,0x2c,0x35,0x8c,0x9e,0x16,
  0x9c,0xc5,0x86,0x15,0xfc,0x04,0xd7,0x4c,0xdd,0xce,0xd5,0xfe,0xdc,0xbf,0xf4,0xef ]
```

최대차분확률은?

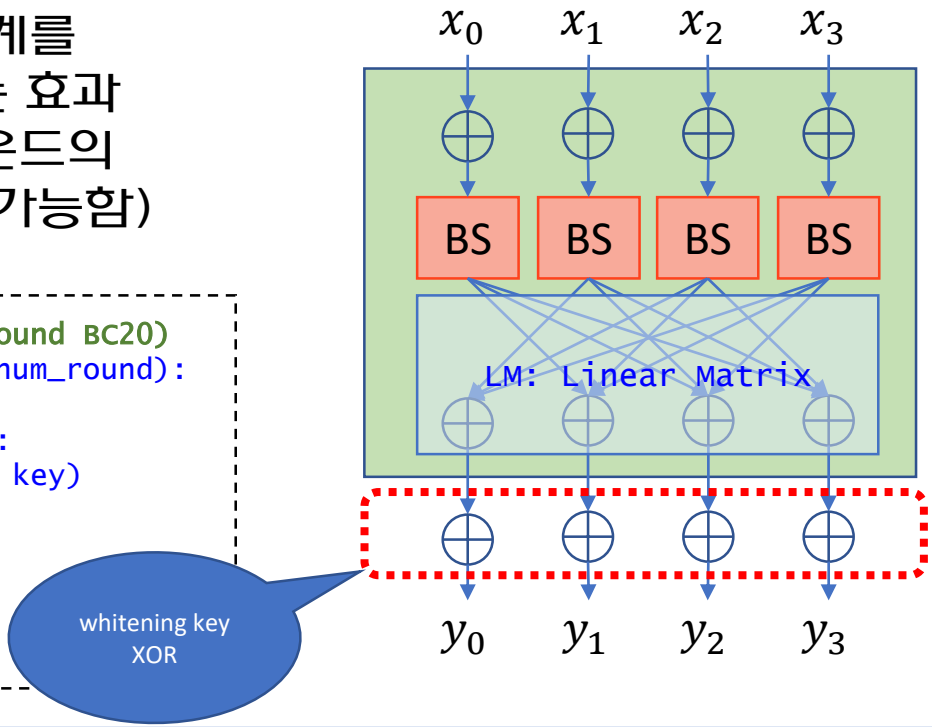
$$\mathrm{P}[64 \rightarrow 64] = DP(64,64) = \frac{\delta(64,64)}{2^8} = \frac{132}{256} = 0.52$$

차분특성이 좋지 않은 BSbox

블록암호 BC20R

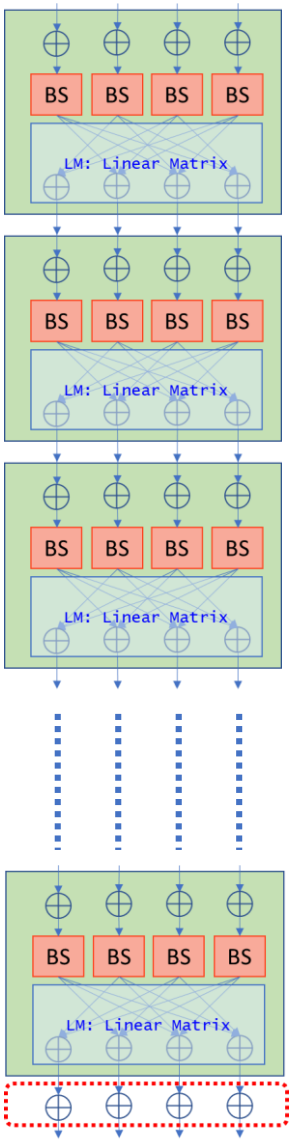
- 마지막 라운드에 추가되는 요소
  - Whitening Key XOR
  - 마지막 연산으로 key XOR를 넣는 이유:
    암호문으로부터 복호화 단계를
    전혀 진행할 수 없도록 하는 효과
    (없으면 누구나 마지막 라운드의
    LM, BS 단계까지 복호가 가능함)

```
#--- BC20R Encryption (Reduced Round BC20)
def BC20R_Enc(input_state, key, num_round):
    state = input_state
    for i in range(0, num_round):
        state = Enc_Round(state, key)

    #== Whitening Key (추가)
    state = AR(state, key)

    return state
```



블록암호 BC20R

- 전체 구조
  - 입력/출력 블록: 4바이트 (32비트)
  - 암호키: 4바이트 (32비트)
  - 라운드 수: 10라운드
    (마지막 라운드는 whitening key XOR 포함)
  - 부분 라운드(reduced-round) 알고리즘
    공격대상을 설정하기 위해 작은 라운드 알고리즘으로 설정하는 경우에도 마지막 whitening은 포함됨
  - 키스케줄: 사용하지 않음 (암호키 = 라운드키)
    (물론 바람직하지 않은 설계임)



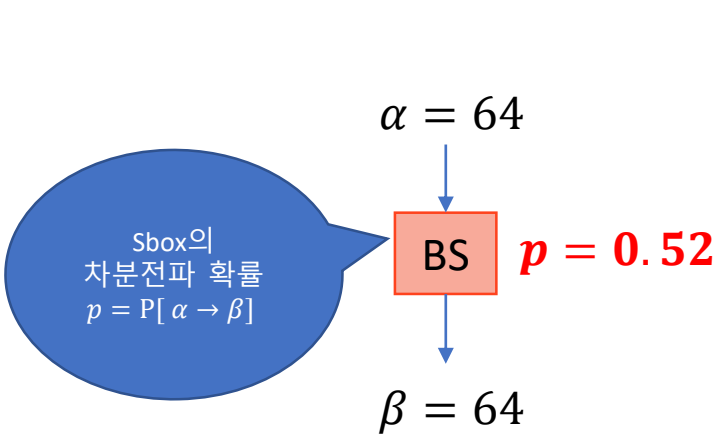
BC20R의 1라운드 차분 특성

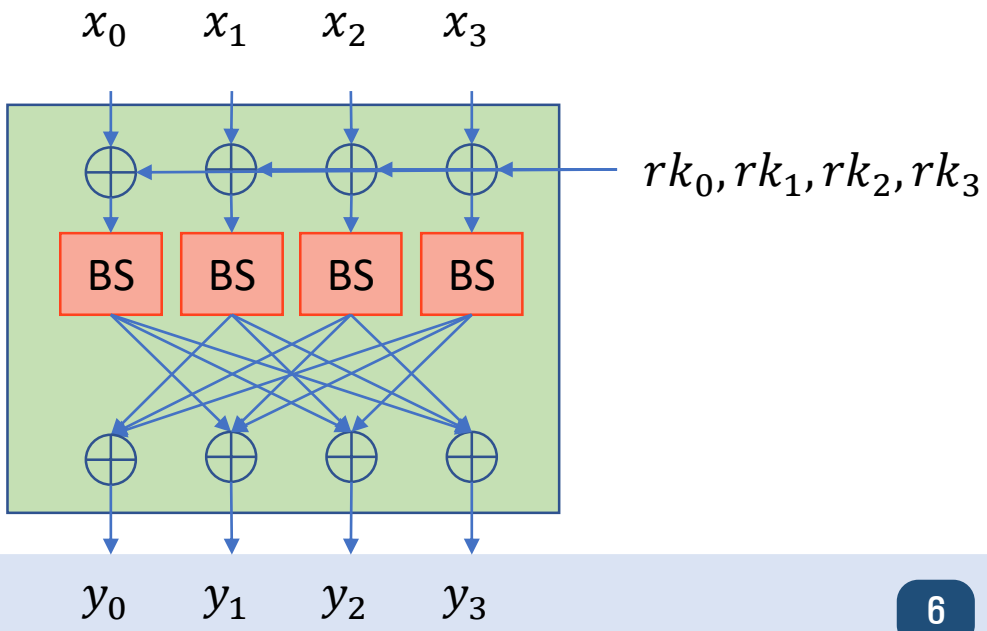
- 라운드 함수의 입력: $X1 \oplus X2 = [\alpha, 0, 0, 0]$
  - AR(Add Round Key) 직후 차분: $[\alpha, 0, 0, 0]$
  - Sbox를 통과 후 차분: $[\beta, 0, 0, 0]$
  - 선형함수 LM 후: $[0, \beta, \beta, \beta]$

  라운드 차분특성 $[\alpha, 0, 0, 0]$ → $[0, \beta, \beta, \beta]$ (확률 p)

차분특성:
차분의 성질을
이용하여 랜덤과
알고리즘을 구별하는
특성

최대 차분확률 $[64, 0, 0, 0]$ → $[0, 64, 64, 64]$ (확률 $p = 0.52$)





BC20R의 1라운드 차분 특성

```
dx = 64
num_iteration = 100
counter = 0
diff_dic = {}
for i in range(num_iteration):
    P1 = [random.randint(0,255), random.randint(0,255), \
          random.randint(0,255), random.randint(0,255)]
    P2 = [P1[0]^dx , P1[1], P1[2], P1[3]]
    key = [1, 2, 3, 4]
    C1 = BC20R.BC20R_Enc(P1, key,1)
    C2 = BC20R.BC20R_Enc(P2, key,1)
    dy = [ C1[i]^C2[i] for i in range(4) ]
    dy_int = Common.list2int(dy)
    if dy_int in diff_dic :
        diff_dic[dy_int].append(P1)
    else:
        diff_dic[dy_int] = P1

    expected_dy = [0, 64, 64, 64]
    expected_int = Common.list2int(expected_dy)
    if expected_int == dy_int:
        counter += 1

print('prob. =', counter/num_iteration)
print(diff_dic[expected_int])
```

```
prob. = 0.52
[22, 191, 228, 108, [117, 12, 197, 225], [7, 246, 177, 191], [15, 144,
48, 211], [92, 37, 94, 17], [104, 224, 99, 124], [61, 44, 51, 43], [44,
44, 191, 73], [59, 145, 34, 176], [11, 151, 55, 144], [125, 173, 6, 83],
[64, 39, 101, 32], [126, 22, 182, 214], [16, 65, 172, 186], [42, 105,
109, 13], [107, 167, 158, 96], [4, 254, 14, 98], [115, 209, 229, 54],
[54, 227, 144, 76], [55, 0, 115, 92], [24, 84, 93, 215], [79, 94, 215,
160], [115, 23, 16, 161], [71, 37, 28, 173], [56, 28, 197, 214], [29, 13,
8, 104], [67, 243, 90, 7], [20, 56, 234, 235], [223, 203, 140, 81], [75,
91, 83, 156], [23, 16, 198, 77], [41, 240, 222, 89], [127, 129, 167,
116], [124, 176, 222, 160], [89, 94, 225, 1], [77, 190, 98, 137], [109,
254, 205, 134], [78, 221, 134, 102], [92, 165, 155, 127], [159, 147, 101,
137], [20, 4, 181, 37], [73, 109, 0, 225], [69, 184, 48, 32], [17, 117,
85, 159], [23, 129, 165, 95], [4, 194, 148, 86], [99, 27, 34, 24], [159,
208, 158, 198], [52, 136, 144, 215], [69, 97, 170, 52], [30, 84, 8, 12],
[0, 130, 77, 192]]
```

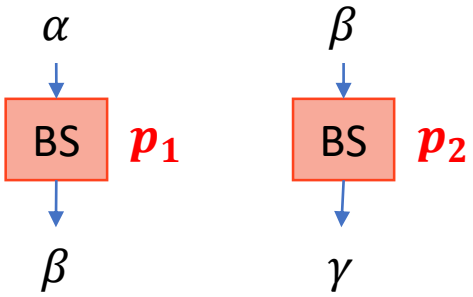
BC20R의 2라운드 차분 특성

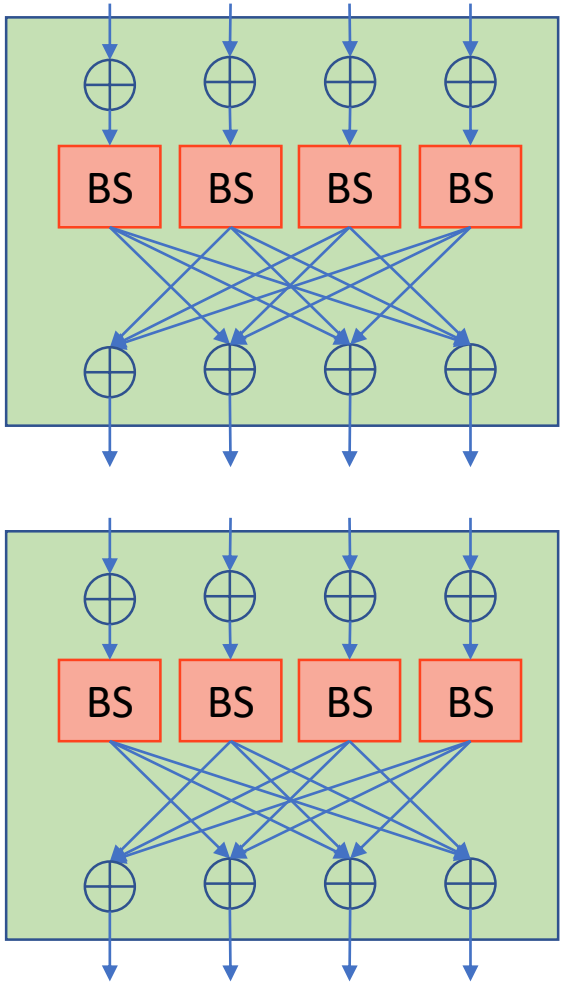
- 1라운드 함수의 입력: $X1 \oplus X2 = [\alpha, 0, 0, 0]$
  - AR(Add Round Key) 직후 차분: $[\alpha, 0, 0, 0]$
  - Sbox를 통과 후 차분: $[\beta, 0, 0, 0]$
  - 선형함수 LM 후: $[0, \beta, \beta, \beta]$

  라운드 차분특성 $[\boldsymbol{\alpha}, \mathbf{0}, \mathbf{0}, \mathbf{0}]$ → $[\mathbf{0}, \boldsymbol{\beta}, \boldsymbol{\beta}, \boldsymbol{\beta}]$ **(확률** $\boldsymbol{p_1}$**)**

- 2라운드 함수의 입력: $X1 \oplus X2 = [0, \beta, \beta, \beta]$
  - AR(Add Round Key) 직후 차분: $[0, \beta, \beta, \beta]$
  - Sbox를 통과 후 차분: $[0, \gamma, \gamma, \gamma]$
  - 선형함수 LM 후: $[\gamma, 0, 0, 0]$

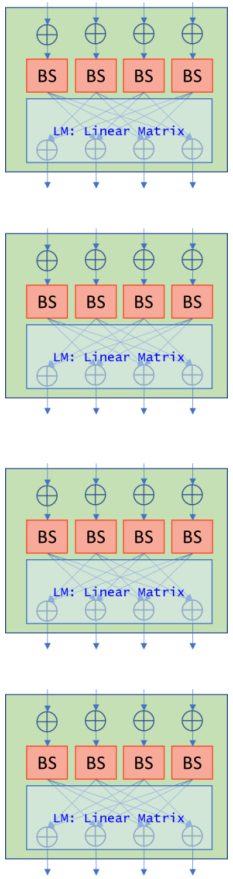
  라운드 차분특성 $[0, \beta, \beta, \beta]$ → $[\gamma, 0, 0, 0]$**(확률** $\boldsymbol{p_2^3}$**)**





최대 차분확률 $[64, 0, 0, 0]$ → $[64, 0, 0, 0]$ **(확률** $p^4 = 0.52^4 = 0.07$**)**

BC20R의 다중 라운드 차분 특성



- 반복 차분특성
  - 라운드 함수의 차분특성을 결합하여 다중 라운드와 랜덤을 구별하는 방법

- 다중 라운드 차분 특성 예제
  - 1라운드: $[\alpha, 0, 0, 0]$ → $[0, \beta, \beta, \beta]$ (확률 p_1)
  - 2라운드: $[0, \beta, \beta, \beta]$ → $[\gamma, 0, 0, 0]$ (확률 p_2^3)
  - 3라운드: $[\gamma, 0, 0, 0]$ → $[0, \delta, \delta, \delta]$ (확률 p_3)
  - 4라운드: $[0, \delta, \delta, \delta]$ → $[\epsilon, 0, 0, 0]$ (확률 p_4^3)

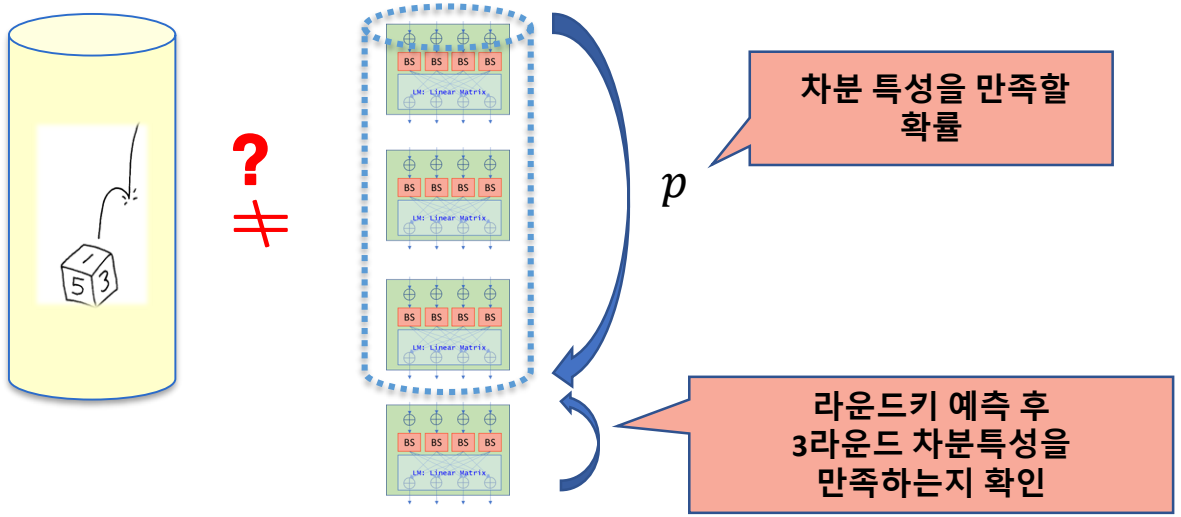
  → 2라운드 단위로 $[\alpha, 0, 0, 0]$ → $[\gamma, 0, 0, 0]$ 패턴이 반복된다!

4라운드 최대 차분확률

$[64, 0, 0, 0]$ → $[64, 0, 0, 0]$ (확률 $(p^4)^2 = 0.52^8 = 0.004$)

BC20R의 차분 공격법(DC)

- 4라운드 BC20R에 대한 차분 공격 개요
  - 3라운드 BC20R과 랜덤함수를 구별하는 차분특성을 사용하고,
  - 4라운드 키의 일부를 찾는 공격법



BC20R의 차분 공격법(DC)

- 선택 (평문, 암호문) 쌍 획득(만들기)

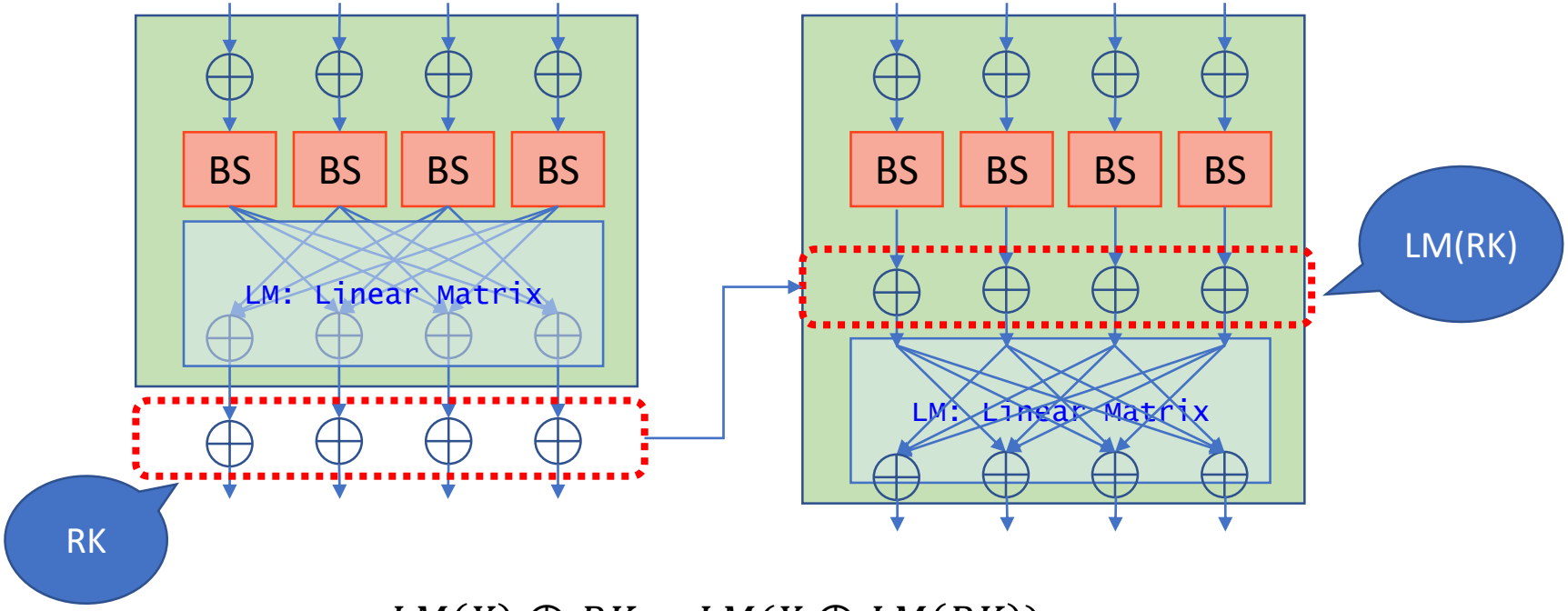
공격자는
이 암호키는 모르고
원하는
(평문, 암호문) 쌍만
얻는다고 가정한다.

```
# 선택 (평문, 암호문) 만들기
dx = 64
num_ptct_paris = 1<<8
num_round = 4   # 3라운드 특성을 이용한 4라운드 공격법
ptct_list = []
key = [ 1, 2, 3, 4]  # 공격자가 찾아야 하는 암호키
for i in range(num_ptct_paris):
    P1 = [random.randint(0,255), random.randint(0,255), \
          random.randint(0,255), random.randint(0,255)]
    P2 = [P1[0]^dx , P1[1], P1[2], P1[3]]
    C1 = BC20R.BC20R_Enc(P1, key, num_round)
    C2 = BC20R.BC20R_Enc(P2, key, num_round)
    ptct_pair = copy.deepcopy([P1, P2, C1, C2])
    ptct_list.append(ptct_pair)

ptct_file = '4R_ptct.var'
#== 생성한 (평문, 암호문) 쌍을 파일에 저장
Common.save_var_to_file(ptct_list, ptct_file)
```

공격에 필요한 (평문,
암호문)쌍을 파일에 저장한다.
공격에서는 이 파일만
사용하면 된다.

BC20R의 차분 공격법(DC)

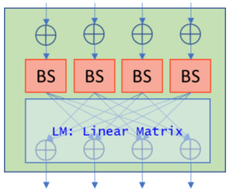
- 마지막 라운드(4라운드) 동치 변형

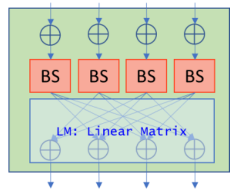


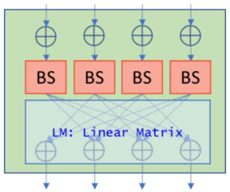
$$LM(X) \oplus RK = LM(X \oplus LM(RK))$$

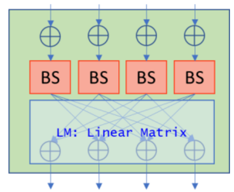
LM: involution
($LM^2 = I$)

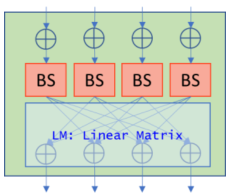
직접 그려봅시다!

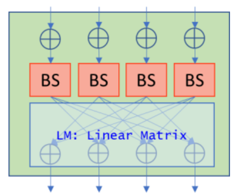

BS BS BS BS
LM: Linear Matrix

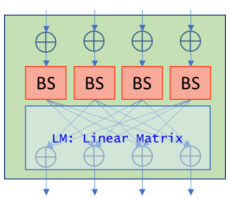

BS BS BS BS
LM: Linear Matrix

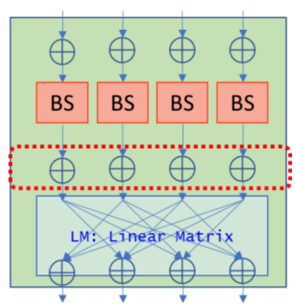

BS BS BS BS
LM: Linear Matrix


BS BS BS BS
LM: Linear Matrix


BS BS BS BS
LM: Linear Matrix

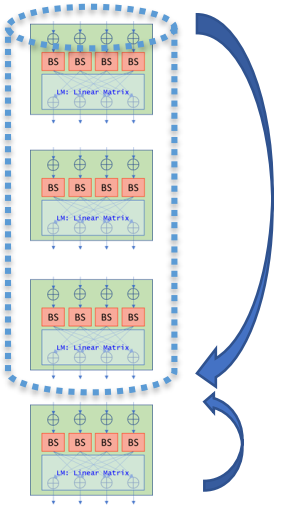

BS BS BS BS
LM: Linear Matrix

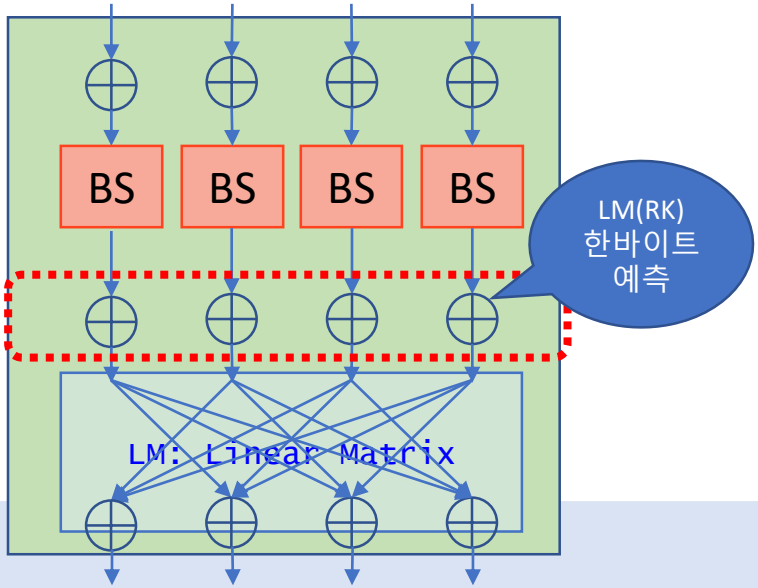

BS BS BS BS
LM: Linear Matrix


BS BS BS BS
LM: Linear Matrix

BC20R의 차분 공격법(DC)

- 마지막 라운드의 복호화
  - (동치) 라운드키 한바이트 예측하면,
  - 암호문으로부터 해당 바이트의 3라운드 출력을 얻는다.
  - 3라운드 차분을 계산해보고 차분특성과 일치하면 예측한 키를 올바른 키후보로 간주한다.
  - 많은 암호문에 대하여 반복하고 가장 많이 추천된 키후보를 올바른 키로 정한다.





BC20R의 차분 공격법(DC)

- 공격 알고리즘

```
#== 파일에서 (평문, 암호문) 쌍을 읽어 변수에 저장
ptct_list = Common.load_var_from_file(ptct_file)
dx = 64
num_round = 4   # 3라운드 특성을 이용한 4라운드 공격법
rkey_dic = {}
for i in range(len(ptct_list)):
    C1 = copy.deepcopy(ptct_list[i][2])
    C2 = copy.deepcopy(ptct_list[i][3])
    state1 = BC20R.LM_Layer(C1)
    state2 = BC20R.LM_Layer(C2)
    for rk in range(256):  # Key 후보 전수조사
        byte1 = BC20R.IBSbox[ state1[1] ^ rk ]
        byte2 = BC20R.IBSbox[ state2[1] ^ rk ]
        if (byte1 ^ byte2) == dx :
            if rk in rkey_dic:
                rkey_dic[rk] += 1
            else:
                rkey_dic[rk] = 1
max_count = 0
max_rk = 0
for rk in rkey_dic:
    if rkey_dic[rk] > max_count:
        max_count = rkey_dic[rk]
        max_rk = rk
print(max_rk, rkey_dic[max_rk])
```

차분특성을 만족하면 후보키 사전에 넣는다.

가장 많이 후보로 선택된 값을 암호키로 정한다.