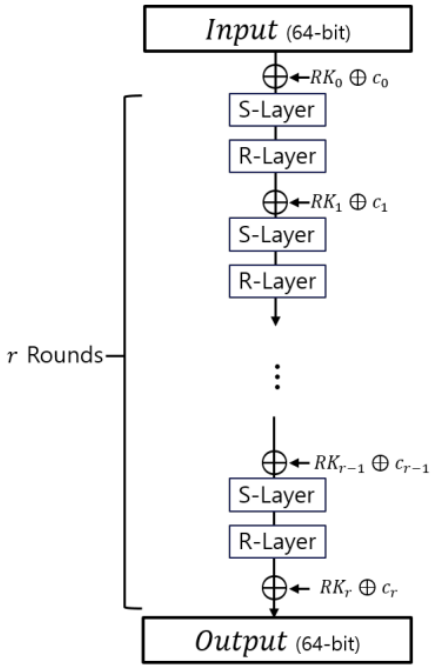
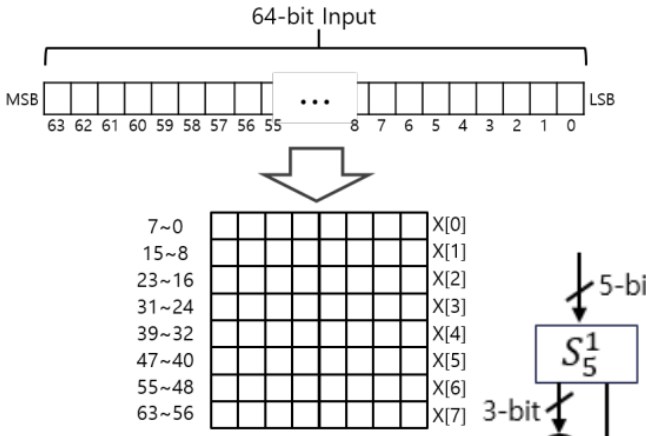
PIPO CPA 공격

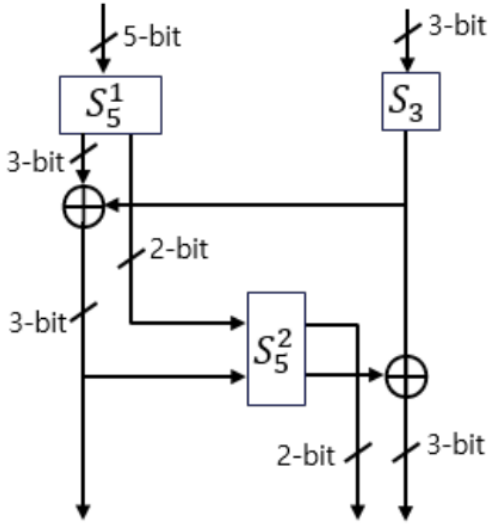
https://youtu.be/e0C5A4tzbXM

PIPO

- SPN 구조
- 더 작은 SBOX를 조합한 unbalanced-Bridge 구조의 S-Layer








```
==MASTER_KEY==
0x6DC416DD,     0x779428D2,     0x7E1D20AD,     0x2E152297,
==ROUND_KEY==
0x7E1D20AD,     0x2E152297,
0x6DC416DD,     0x779428D3,
0x7E1D20AD,     0x2E152295,
0x6DC416DD,     0x779428D1,
0x7E1D20AD,     0x2E152293,
0x6DC416DD,     0x779428D7,
0x7E1D20AD,     0x2E152291,
0x6DC416DD,     0x779428D5,
0x7E1D20AD,     0x2E15229F,
0x6DC416DD,     0x779428DB,
0x7E1D20AD,     0x2E15229D,
0x6DC416DD,     0x779428D9,
0x7E1D20AD,     0x2E15229B,
0x6DC416DD,     0x779428DF,
```

```
==MASTER_KEY==
0x009A3AA4,     0x76A96DB5,     0x54A71206,     0x26D15633,     0x6DC416DD,     0x779428D2,     0x7E1D20AD,     0x2E152297,
==ROUND_KEY==
0x7E1D20AD,     0x2E152297,
0x6DC416DD,     0x779428D3,
0x54A71206,     0x26D15631,
0x009A3AA4,     0x76A96DB6,
0x7E1D20AD,     0x2E152293,
0x6DC416DD,     0x779428D7,
0x54A71206,     0x26D15635,
0x009A3AA4,     0x76A96DB2,
0x7E1D20AD,     0x2E15229F,
0x6DC416DD,     0x779428DB,
0x54A71206,     0x26D15639,
0x009A3AA4,     0x76A96DBE,
0x7E1D20AD,     0x2E15229B,
0x6DC416DD,     0x779428DF,
0x54A71206,     0x26D1563D,
0x009A3AA4,     0x76A96DBA,
0x7E1D20AD,     0x2E152287,
0x6DC416DD,     0x779428C3,
```

PIPO 구현

```
void convert(u8* X)
{
    int i,j;
    u8 T[8] = { 0, };
    for (i = 0; i < 8; i++)
        for (j = 0; j < 8; j++)
            T[i] |= (((X[j] & (1<<i))>>i) << j);

    for (i = 0; i < 8; i++)
        X[i] = T[i];
}

void sbox_TLU(u8 *X) {
    int i;
    convert(X);
    for (i = 0; i < 8; i++)
        X[i] = Sbox[X[i]];
    convert(X);
}
```

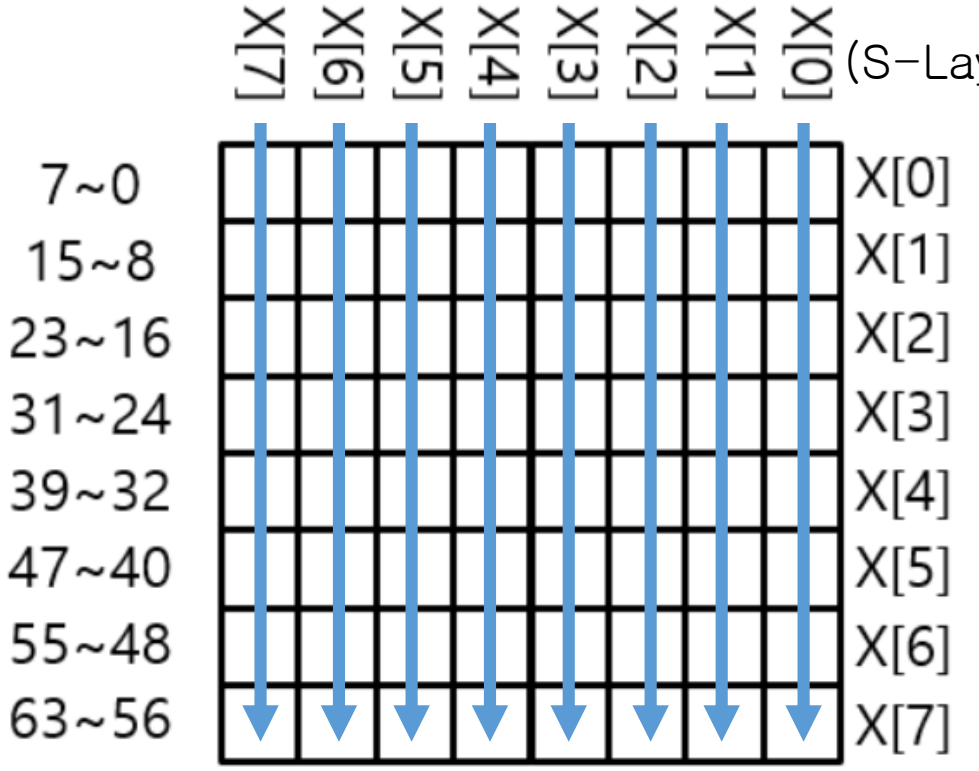
```
//left rotation: (0,7,4,3,6,5,1,2)
void pbox(u8* X)
{
    X[1] = ((X[1] << 7)) | ((X[1] >> 1));
    X[2] = ((X[2] << 4)) | ((X[2] >> 4));
    X[3] = ((X[3] << 3)) | ((X[3] >> 5));
    X[4] = ((X[4] << 6)) | ((X[4] >> 2));
    X[5] = ((X[5] << 5)) | ((X[5] >> 3));
    X[6] = ((X[6] << 1)) | ((X[6] >> 7));
    X[7] = ((X[7] << 2)) | ((X[7] >> 6));

}
```

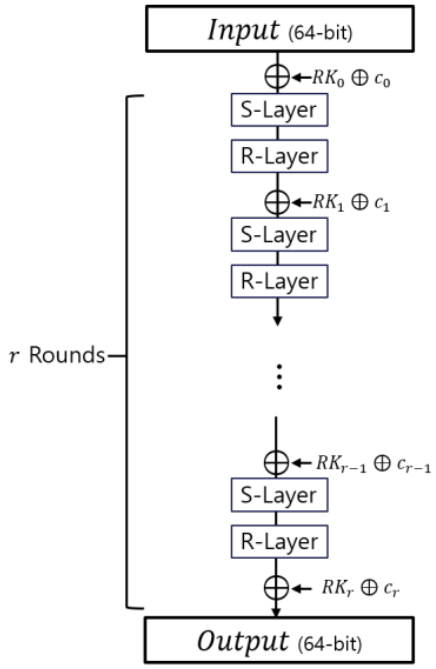
```
// S5_1
Mask_refreshing(X[7]);
ISW_AND(T[3], X[7], X[6]);
for (i = 0; i < SHARES; i++) X[5][i] ^= T[3][i];
Mask_refreshing(X[3]);
ISW_AND(T[3], X[3], X[5]);
for (i = 0; i < SHARES; i++)
{X[4][i] ^= T[3][i]; X[7][i] ^= X[4][i]; X[6][i] ^= X[3][i];}
Mask_refreshing(X[4]);
ISW_OR(T[3], X[4], X[5]);
for (i = 0; i < SHARES; i++) {X[3][i] ^= T[3][i]; X[5][i] ^= X[7][i];}
Mask_refreshing(X[5]);
ISW_AND(T[3], X[5], X[6]);
for (i = 0; i < SHARES; i++) X[4][i] ^= T[3][i];
```

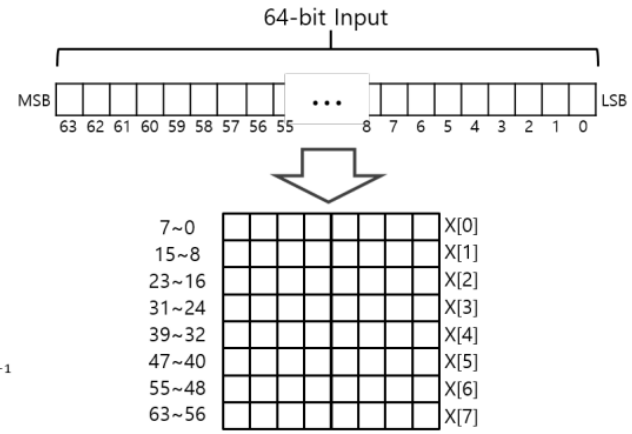
```
u8 T[3] = { 0, };
//(MSB: x[7], LSB: x[0])
// Input: x[7], x[6], x[5], x[4], x[3], x[2], x[1], x[0]
//S5_1
X[5] ^= (X[7] & X[6]);
X[4] ^= (X[3] & X[5]);
X[7] ^= X[4];
X[6] ^= X[3];
X[3] ^= (X[4] | X[5]);
X[5] ^= X[7];
X[4] ^= (X[5] & X[6]);
//S3
X[2] ^= X[1] & X[0];
X[0] ^= X[2] | X[1];
X[1] ^= X[2] | X[0];
X[2] = ~X[2];
// Extend XOR
X[7] ^= X[1];   X[3] ^= X[2];   X[4] ^= X[0];
//S5_2
T[0] = X[7];    T[1] = X[3];    T[2] = X[4];
X[6] ^= (T[0] & X[5]);
T[0] ^= X[6];
X[6] ^= (T[2] | T[1]);
T[1] ^= X[5];
X[5] ^= (X[6] | T[2]);
T[2] ^= (T[1] & T[0]);
// Truncate XOR and bit change
X[2] ^= T[0];   T[0] = X[1] ^ T[2]; X[1] = X[0]^T[1];   X[0] = X[7];    X[7] = T[0];
T[1] = X[3];    X[3] = X[6];    X[6] = T[1];
T[2] = X[4];    X[4] = X[5];    X[5] = T[2];
// Output: (MSb) x[7], x[6], x[5], x[4], x[3], x[2], x[1], x[0] (LSb)
```

CPA 공격

- 테이블의 SBOX 연산 후 값를 중간 값으로 공격, 1000개 파형 수집
- 공격 이후 전환 필요

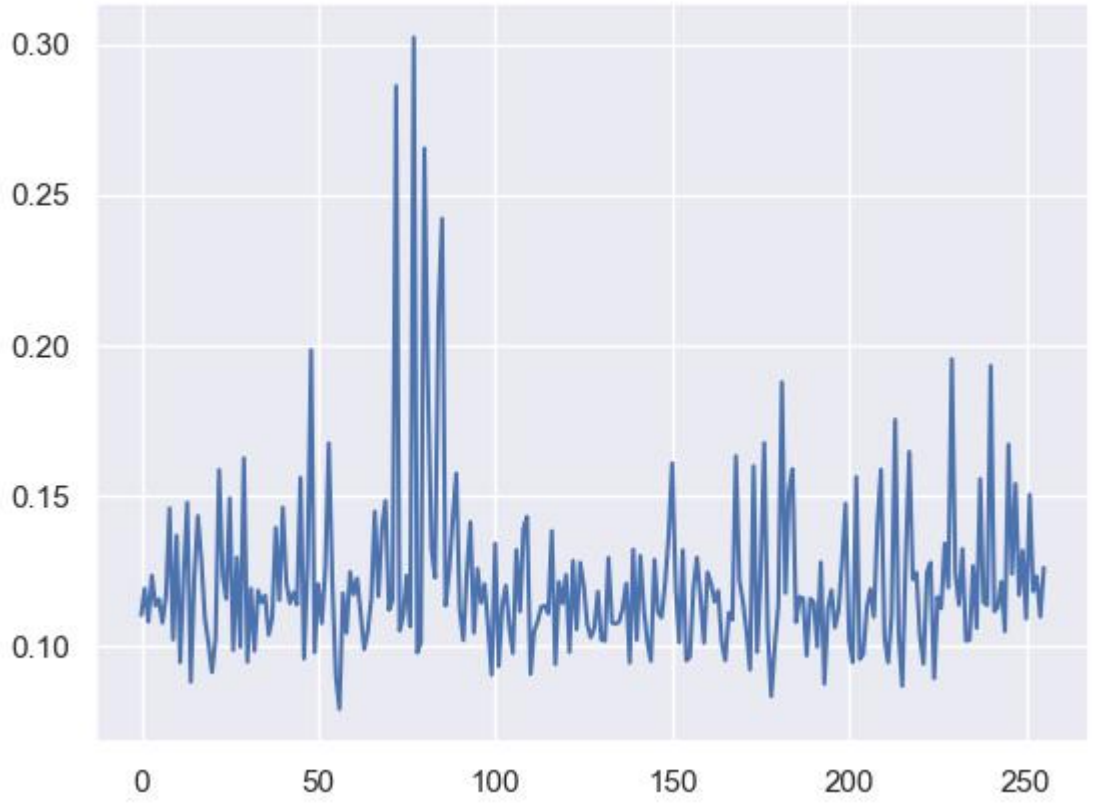


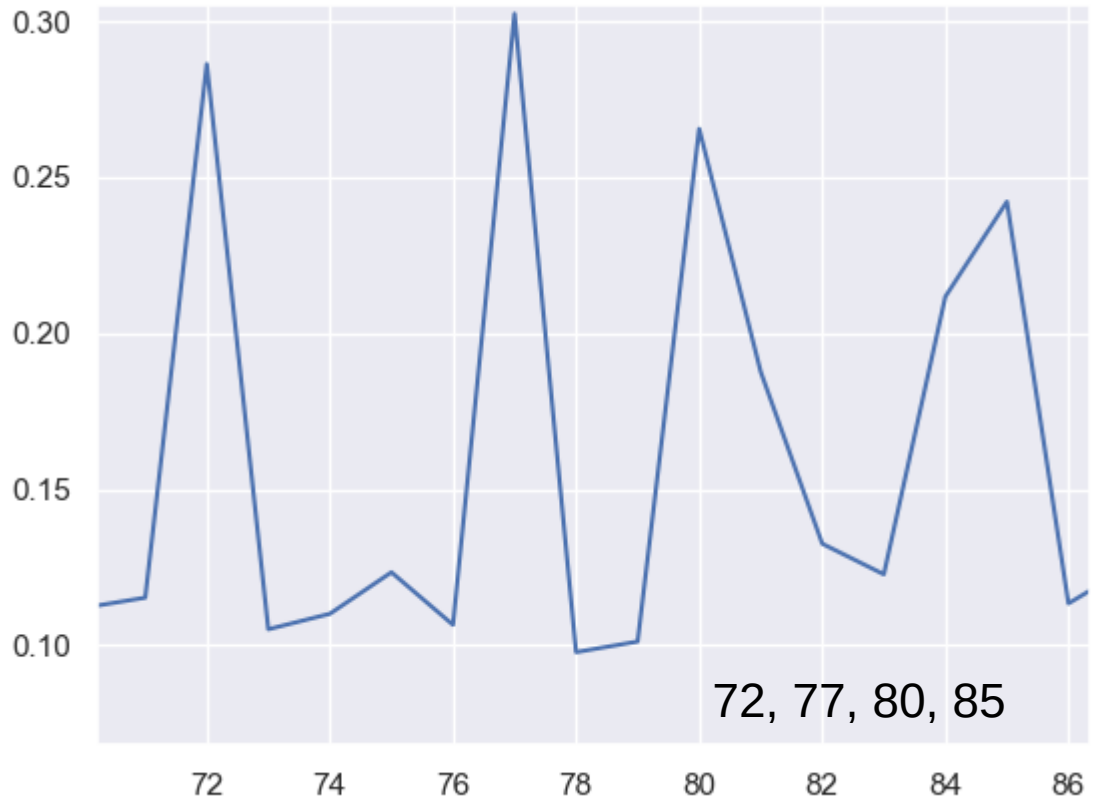




CPA 공격 8BIT

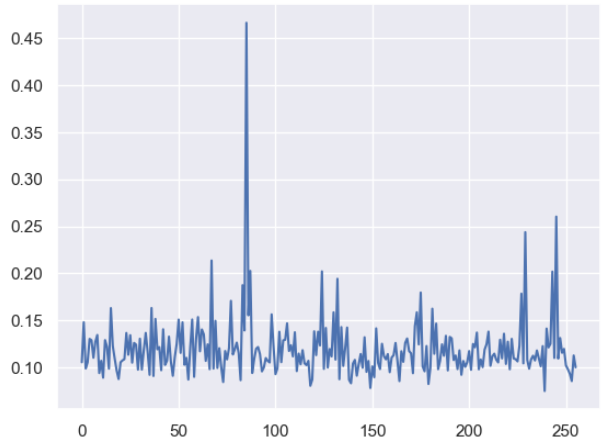
- 중간 값의 최하위 비트 공격 결과 (알려진 값 : 0X55, 85)

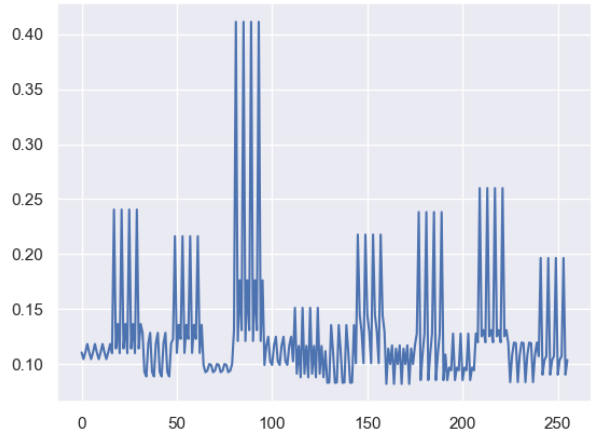


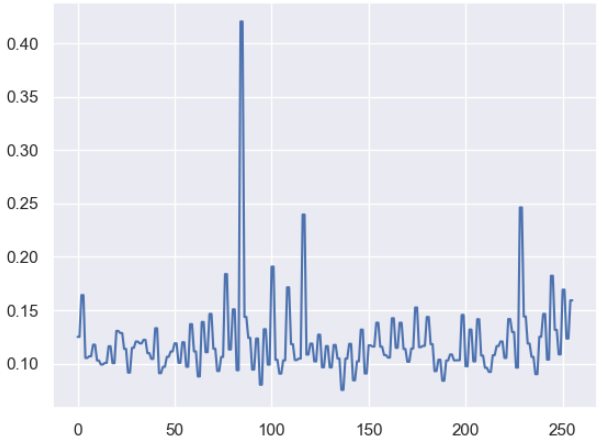


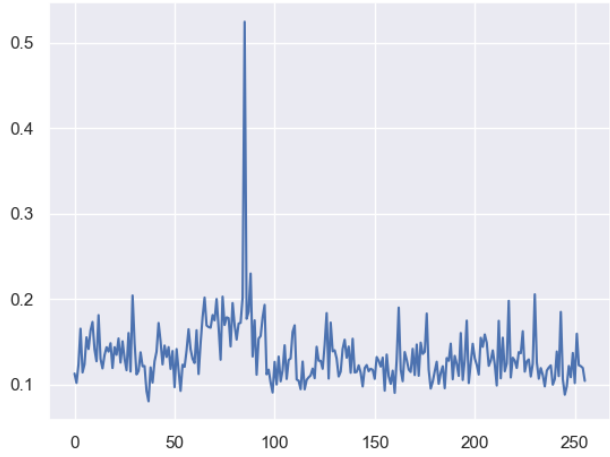
CPA 공격 1BIT

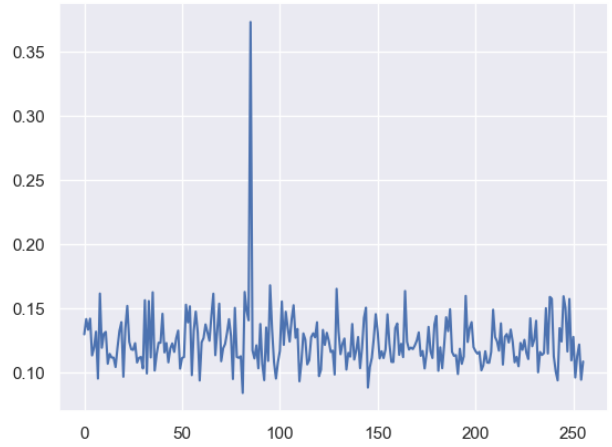
- 중간 값 8비트에서 1비트 씩 공격 수행

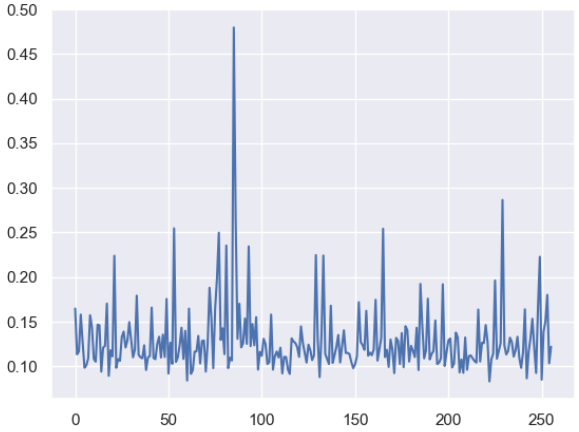


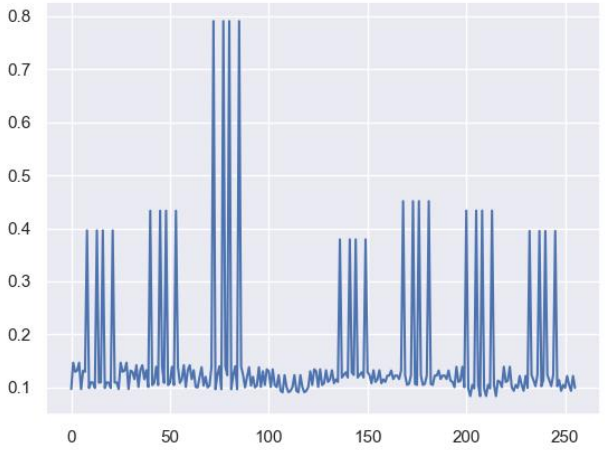


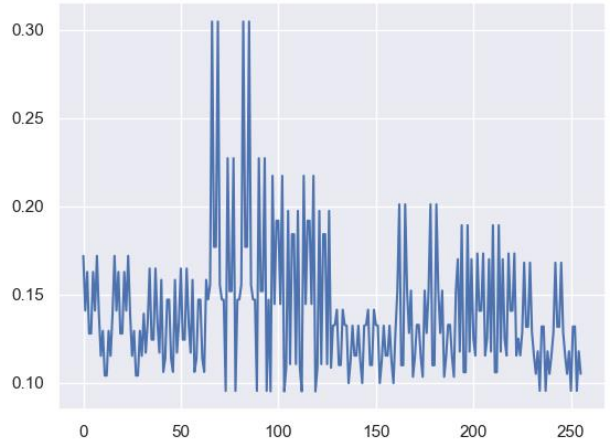






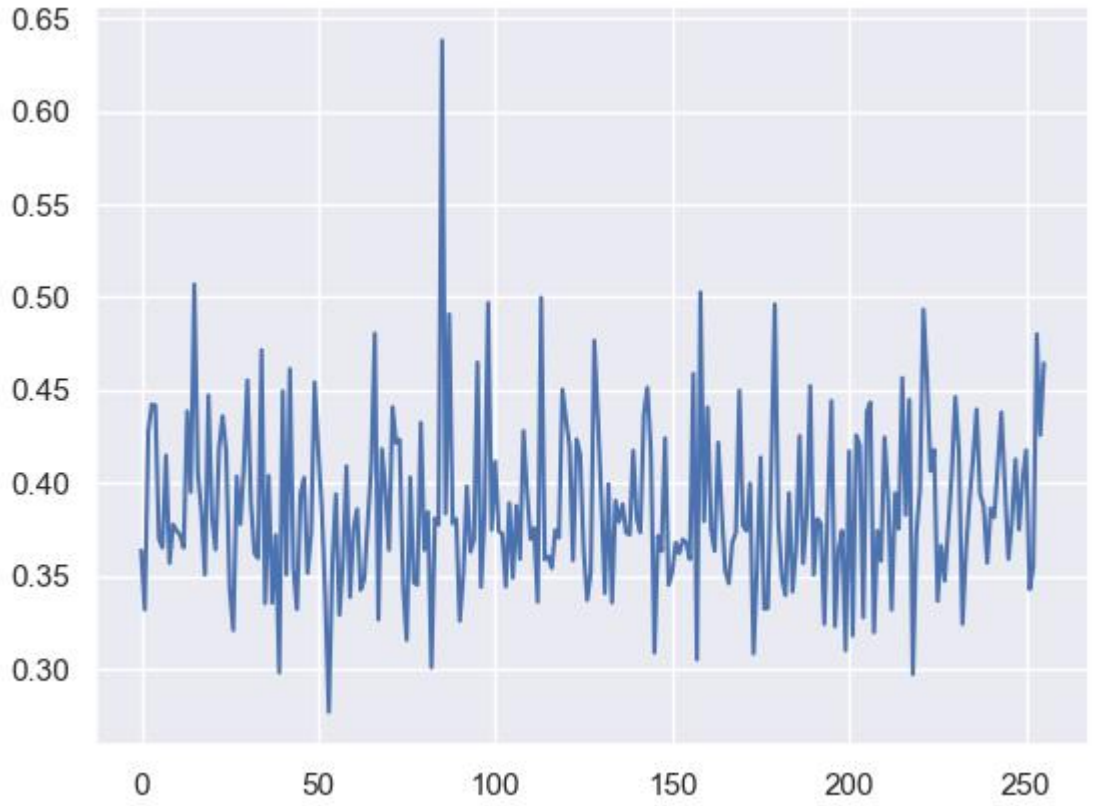






효율적인 공격

- 하위 4번째 비트에서 70개의 파형파형으로 공격가능



Q & A