2017320132 장연석

[Midterm Project_Problem2_Linear Cryptanalysis]

[1] Solution (need to show all the processes step by step)

1) Get Linear Approximation Table for S-box, and Inverse S-box

- toy_cipher_2017320132/linear_approx.c 소스코드

<S-Box>

Input Sum \ Output Sum	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	+8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	-2	-2	0	0	-2	+6	+2	+2	0	0	+2	+2	0	0
2	0	0	-2	-2	0	0	-2	-2	0	0	+2	+2	0	0	-6	+2
3	0	0	0	0	0	0	0	0	+2	-6	-2	-2	+2	+2	-2	-2
4	0	+2	0	-2	-2	-4	-2	0	0	-2	0	+2	+2	-4	+2	0
5	0	-2	-2	0	-2	0	+4	+2	-2	0	-4	+2	0	-2	-2	0
6	0	+2	-2	+4	+2	0	0	+2	0	-2	+2	+4	-2	0	0	-2
7	0	-2	0	+2	+2	-4	+2	0	-2	0	+2	0	+4	+2	0	+2
8	0	0	0	0	0	0	0	0	-2	+2	+2	-2	+2	-2	-2	-6
9	0	0	-2	-2	0	0	-2	-2	-4	0	-2	+2	0	+4	+2	-2
A	0	+4	-2	+2	-4	0	+2	-2	+2	+2	0	0	+2	+2	0	0
B	0	+4	0	-4	+4	0	+4	0	0	0	0	0	0	0	0	0
C	0	-2	+4	-2	-2	0	+2	0	+2	0	+2	+4	0	+2	0	-2
D	0	+2	+2	0	-2	+4	0	+2	-4	-2	+2	0	+2	0	0	+2
E	0	+2	+2	0	-2	-4	0	+2	-2	0	0	-2	-4	+2	-2	0
F	0	-2	-4	-2	-2	0	+2	0	0	-2	+4	-2	-2	0	+2	0

<Inverse S-Box>

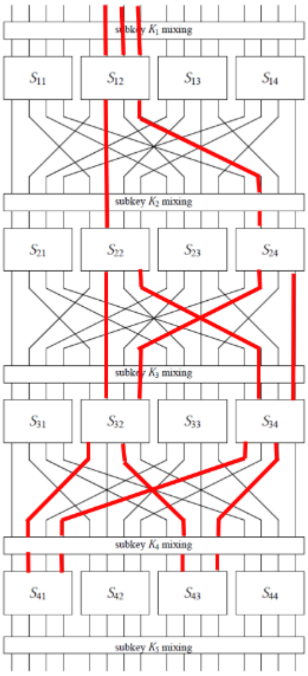
IN \ OUT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	2	-2	2	-2	0	0	4	4	-2	2	2	-2
2	0	-2	-2	0	0	-2	-2	0	0	-2	-2	0	4	2	2	-4
3	0	-2	-2	0	-2	0	4	2	0	-2	2	-4	-2	0	0	-2
4	0	0	0	0	-2	-2	2	2	0	0	-4	4	-2	-2	-2	-2
5	0	0	0	0	-4	0	0	-4	0	0	0	0	0	4	-4	0
6	0	-2	-2	0	-2	4	0	2	0	-2	2	4	2	0	0	2
7	0	6	-2	0	0	2	2	0	0	-2	-2	0	0	2	2	0
8	0	2	0	2	0	-2	0	-2	-2	-4	2	0	2	-4	-2	0
9	0	2	0	-6	-2	0	-2	0	2	0	2	0	0	-2	0	-2
10	0	0	2	-2	0	-4	2	2	2	-2	0	0	2	2	0	4
11	0	0	2	-2	2	2	4	0	-2	2	0	0	4	0	-2	-2
12	0	2	0	2	2	0	-2	4	2	0	2	0	0	2	-4	-2
13	0	2	0	2	-4	-2	0	2	-2	4	2	0	2	0	2	0
14	0	0	-6	-2	2	-2	0	0	-2	2	0	0	0	0	-2	2
15	0	0	2	-2	0	0	-2	2	-6	-2	0	0	-2	2	0	0

```c
// Make Linear Approximation Table of S-Box
void make_approx_table(){
    //Linear Approximation Table
    BIT approx_table[16][16];
    // Input Sum a
    for(int a = 0; a < 16; a++){
        BIT* a_bin = dec_to_bin(a, 4);
        // Output Sum B
        for(int b = 0; b < 16; b++){
            approx_table[a][b] = -8;
            BIT* b_bin = dec_to_bin(b, 4);
            for(int X_in = 0; X_in < 16; X_in++){
                int Y_out = s_box[X_in];
                // Input X
                BIT* X_in_bin = dec_to_bin(X_in, 4);
                // Output Y
                BIT* Y_out_bin = dec_to_bin(Y_out, 4);
                BIT result = 0;
                for(int i = 0; i < 4; i++){
                    if(a_bin[i] == 1){
                        result ^= X_in_bin[i];
                    }
                }
                for(int j = 0; j < 4; j++){
                    if(b_bin[j] == 1){
                        result ^= Y_out_bin[j];
                    }
                }
                if(result == 0){
                    approx_table[a][b]++;
                }
                free(X_in_bin);
                free(Y_out_bin);
            }
            free(b_bin);
        }
        free(a_bin);
    }
    for(int i = 0; i < 16; i++){
        for(int j = 0; j < 16; j++){
            printf("%4d,", approx_table[i][j]);
        }
        printf("\n");
    }
}
```

```c
// Make Linear Approximation Table of Inverse S-Box
void make_inv_approx_table(){
    //Linear Approximation Table
    BIT approx_table[16][16];
    // Input Sum a
    for(int a = 0; a < 16; a++){
        BIT* a_bin = dec_to_bin(a, 4);
        // Output Sum B
        for(int b = 0; b < 16; b++){
            approx_table[a][b] = -8;
            BIT* b_bin = dec_to_bin(b, 4);
            for(int X_in = 0; X_in < 16; X_in++){
                int Y_out = inv_s_box[X_in];
                // Input X
                BIT* X_in_bin = dec_to_bin(X_in, 4);
                // Output Y
                BIT* Y_out_bin = dec_to_bin(Y_out, 4);
                BIT result = 0;
                for(int i = 0; i < 4; i++){
                    if(a_bin[i] == 1){
                        result ^= X_in_bin[i];
                    }
                }
                for(int j = 0; j < 4; j++){
                    if(b_bin[j] == 1){
                        result ^= Y_out_bin[j];
                    }
                }
                if(result == 0){
                    approx_table[a][b]++;
                }
                free(X_in_bin);
                free(Y_out_bin);
            }
            free(b_bin);
        }
        free(a_bin);
    }
    for(int i = 0; i < 16; i++){
        for(int j = 0; j < 16; j++){
            printf("%4d,", approx_table[i][j]);
        }
        printf("\n");
    }
}
```

*2) Look for route that has largest linear bias and find active S-box with bias using **linear approximation table of 1)***

[For last round's key (K5,1..4 / K5,9..12)]



Linear approximation: U4,2⊕U4,4⊕U4,10⊕U4,12⊕P6⊕P7⊕P8=0

Active S-box: S12, S22, S24, S32, S34

Linear Bias: $2^4 \times (\frac{4}{16})^5 = 0.015625$

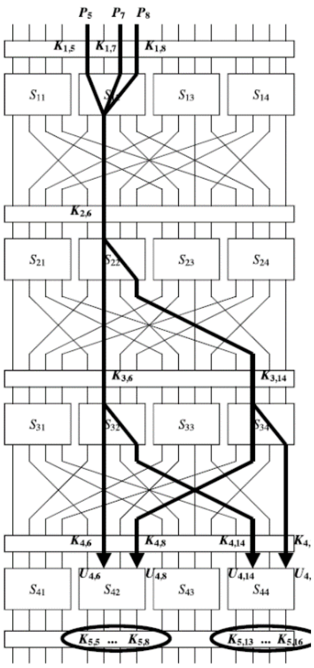
V = partial_decrypt(C, K)

U = Inv_S_box(V)

Active S-box의 수는 적고,

각 S-box의 bias 가 가능한 큰 Route를 선택

[For last round's key (K5,5..8 / K5,13..16)]



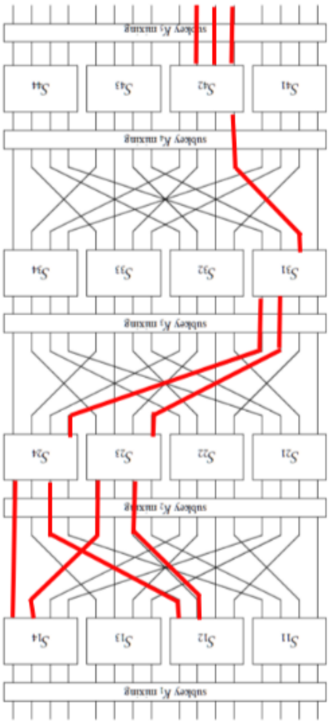
Linear approximation: U4,6⊕U4,8⊕U4,14⊕U4,16⊕P5⊕P7⊕P8=0

Active S-box: S12, S22, S32, S34

Linear Bias: $2^3 \times (\frac{4}{16})^4 = 0.03125$

논문에서 예시로 소개된 Route

[For first round's key (K1,1..4 / K1,9..12)]



Linear approximation: U4,1⊕U4,2⊕U4,9⊕U4,10⊕C10⊕C11⊕C12=0

Active S-box: S13, S24, S31, S32

Linear Bias: $2^3 \times (\frac{6}{16}) \times (\frac{4}{16})^3 = 0.046875$

V = partial_encrypt(P, K)

U = S_box(V)

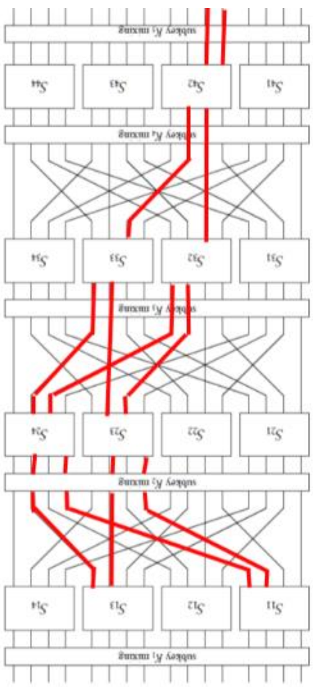
Last round key를 구할 때와 동일한 방식으로 Route를 찾되,

S-box 대신 Inv S-box의 linear approximation table을 이용

Active Inv S-box의 수는 적고,

각 Inv S-box의 bias 가 가능한 큰 Route를 선택

[For first round's key (K1,5..8 / K1,13..16)]



Linear approximation: U4,4⊕U4,5⊕U4,13⊕U4,14⊕C11⊕C12=0

Active S-box: S13, S22, S23, S31, S32

Linear Bias: $2^4 \times (\frac{4}{16})^5 = 0.015625$

3) *Calculate bias approximation value for each 8bit subkey &*

Select a subkey by comparing bias approximation value with active S-box bias

[Last round의 Subkey extraction과정]

2)에서 구한 route의 linear expression 식 E를 이용

8bit의 subkey로 가능한 값 0x00부터 0xFF 각각에 대해

각 plaintext P, ciphertext C 쌍에 대해

subkey 값을 이용해 C를 partially decrypt (V)

Partially decrypt한 결과를 Inverse S-box를 이용해 치환 (U)

U의 비트 중 E에 포함된 비트들을 추출 (Ui), P의 비트 중 E에 포함된 비트들을 추출 (Pi)

Ui 값들과 Pi 값들을 XOR 연산한 결과가 0인 경우 count를 증가

얻은 count 값을 이용해 해당 subkey의 bias를 구함

가장 큰 bias 값을 얻은 subkey를 반환

[First round의 Subkey extraction과정]

2)에서 구한 route의 linear expression 식 E를 이용

8bit의 subkey로 가능한 값 0x00부터 0xFF 각각에 대해

각 plaintext P, ciphertext C 쌍에 대해

subkey 값을 이용해 P를 partially encrypt (V)

Partially encrypt한 결과를 S-box를 이용해 치환 (U)

U의 비트 중 E에 포함된 비트들을 추출 (Ui), C의 비트 중 E에 포함된 비트들을 추출 (Ci)

Ui 값들과 Ci 값들을 XOR 연산한 결과가 0인 경우 count를 증가

얻은 count 값을 이용해 해당 subkey의 bias를 구함

가장 큰 bias 값을 얻은 subkey를 반환

[For last round's key]

```
K5,1..4 / K5,9..12   ->
 [ 0] count: 4910, bias: 0.009000
 [ 1] count: 4906, bias: 0.009400
 [ 2] count: 5233, bias: 0.023300
 [ 3] count: 5116, bias: 0.011600
 [ 4] count: 4977, bias: 0.002300
 [ 5] count: 5021, bias: 0.002100
 [ 6] count: 4887, bias: 0.011300
 [ 7] count: 4960, bias: 0.004000
 [ 8] count: 5001, bias: 0.000100
 [ 9] count: 5074, bias: 0.007400
 [ A] count: 5018, bias: 0.001800
 [ B] count: 5062, bias: 0.006200
 [ C] count: 5071, bias: 0.007100
 [ D] count: 4954, bias: 0.004600
 [ E] count: 4907, bias: 0.009300
 [ F] count: 4903, bias: 0.009700
[10] count: 5011, bias: 0.001100
[11] count: 4989, bias: 0.001100
[12] count: 5034, bias: 0.003400
[13] count: 5019, bias: 0.001900
[14] count: 4998, bias: 0.000200
[15] count: 5038, bias: 0.003800
[16] count: 4994, bias: 0.000600
[17] count: 4925, bias: 0.007500
[18] count: 5068, bias: 0.006800
[19] count: 4999, bias: 0.000100
[1A] count: 4969, bias: 0.003100
[1B] count: 5009, bias: 0.000900
[1C] count: 4970, bias: 0.003000
[1D] count: 4955, bias: 0.004500
[1E] count: 5022, bias: 0.002200
[1F] count: 5000, bias: 0.000000
```

```
K5,5..8 / K5,13..16   ->
 [ 0] count: 5057, bias: 0.005700
 [ 1] count: 4999, bias: 0.000100
 [ 2] count: 4953, bias: 0.004700
 [ 3] count: 4874, bias: 0.012600
 [ 4] count: 4961, bias: 0.003900
 [ 5] count: 5029, bias: 0.002900
 [ 6] count: 5058, bias: 0.005800
 [ 7] count: 5005, bias: 0.000500
 [ 8] count: 5038, bias: 0.003800
 [ 9] count: 4985, bias: 0.001500
 [ A] count: 4928, bias: 0.007200
 [ B] count: 4996, bias: 0.000400
 [ C] count: 5007, bias: 0.000700
 [ D] count: 4928, bias: 0.007200
 [ E] count: 5120, bias: 0.012000
 [ F] count: 5062, bias: 0.006200
[10] count: 5271, bias: 0.027100
[11] count: 5243, bias: 0.024300
[12] count: 4729, bias: 0.027100
[13] count: 4646, bias: 0.035400
[14] count: 4953, bias: 0.004700
[15] count: 4943, bias: 0.005700
[16] count: 5104, bias: 0.010400
[17] count: 4963, bias: 0.003700
[18] count: 5086, bias: 0.008600
[19] count: 4945, bias: 0.005500
[1A] count: 5008, bias: 0.000800
[1B] count: 4998, bias: 0.000200
[1C] count: 4869, bias: 0.013100
[1D] count: 4786, bias: 0.021400
[1E] count: 5242, bias: 0.024200
[1F] count: 5214, bias: 0.021400
```

```
K5,1..4 / K5,9..12   ->  [02] bias: 0.023300 (active S-box bias: 0.015625)
K5,5..8 / K5,13..16  ->  [13] bias: 0.035400 (active S-box bias: 0.031250)
>> Last round's key:     / 0 0 0 0 / 0 0 0 1 / 0 0 1 0 / 0 0 1 1
```

[For first round's key]

```
K1,1..4 / K1,9..12   ->
 [ 0] count: 4694, bias: 0.030600
 [ 1] count: 5023, bias: 0.002300
 [ 2] count: 5525, bias: 0.052500
 [ 3] count: 5006, bias: 0.000600
 [ 4] count: 5249, bias: 0.024900
 [ 5] count: 4987, bias: 0.001300
 [ 6] count: 4714, bias: 0.028600
 [ 7] count: 5028, bias: 0.002800
 [ 8] count: 4961, bias: 0.003900
 [ 9] count: 5289, bias: 0.028900
 [ A] count: 5024, bias: 0.002400
 [ B] count: 4748, bias: 0.025200
 [ C] count: 5029, bias: 0.002900
 [ D] count: 4768, bias: 0.023200
 [ E] count: 4942, bias: 0.005800
 [ F] count: 5013, bias: 0.001300
```

```
[10] count: 5010, bias: 0.001000
[11] count: 5019, bias: 0.001900
[12] count: 5015, bias: 0.001500
[13] count: 5142, bias: 0.014200
[14] count: 4995, bias: 0.000500
[15] count: 5053, bias: 0.005300
[16] count: 4948, bias: 0.005200
[17] count: 4908, bias: 0.009200
[18] count: 5113, bias: 0.011300
[19] count: 5021, bias: 0.002100
[1A] count: 4926, bias: 0.007400
[1B] count: 5036, bias: 0.003600
[1C] count: 4815, bias: 0.018500
[1D] count: 4974, bias: 0.002600
[1E] count: 5024, bias: 0.002400
[1F] count: 5001, bias: 0.000100
```

```
K1,5..8 / K1,13..16   ->
 [ 0] count: 5023, bias: 0.002300
 [ 1] count: 4870, bias: 0.013000
 [ 2] count: 4978, bias: 0.002200
 [ 3] count: 5045, bias: 0.004500
 [ 4] count: 4932, bias: 0.006800
 [ 5] count: 5005, bias: 0.000500
 [ 6] count: 5024, bias: 0.002400
 [ 7] count: 4909, bias: 0.009100
 [ 8] count: 5076, bias: 0.007600
 [ 9] count: 4959, bias: 0.004100
 [ A] count: 5010, bias: 0.001000
 [ B] count: 5085, bias: 0.008500
 [ C] count: 5049, bias: 0.004900
 [ D] count: 5086, bias: 0.008600
 [ E] count: 5036, bias: 0.003600
 [ F] count: 4913, bias: 0.008700
```

```
[10] count: 4968, bias: 0.003200
[11] count: 4917, bias: 0.008300
[12] count: 5085, bias: 0.008500
[13] count: 5256, bias: 0.025600
[14] count: 4967, bias: 0.003300
[15] count: 4988, bias: 0.001200
[16] count: 4931, bias: 0.006900
[17] count: 5076, bias: 0.007600
[18] count: 5143, bias: 0.014300
[19] count: 5044, bias: 0.004400
[1A] count: 4793, bias: 0.020700
[1B] count: 5058, bias: 0.005800
[1C] count: 4952, bias: 0.004800
[1D] count: 5021, bias: 0.002100
[1E] count: 4875, bias: 0.012500
[1F] count: 4926, bias: 0.007400
```

```
K1,1..4 / K1,9..12    ->  [02] bias: 0.052500 (active S-box bias: 0.046875)
K1,5..8 / K1,13..16   ->  [13] bias: 0.025600 (active S-box bias: 0.015625)
>> First round's key:     / 0 0 0 0 / 0 0 0 1 / 0 0 1 0 / 0 0 1 1
```

[2] Appendix: Source code (with comment)

- toy_cipher_2017320132/linear_cryptanalysis.c

```
BIT last_P_idx[] = {5, 6, 7};
BIT last_U_idx[] = {1, 3, 9, 11};
BIT last_K[]     = {0, 2};
printf("K5,1..4 / K5,9..12   ->  ");
int last_maxIdx = get_partial_subkey(last_P_idx, last_U_idx, last_K, sizeof(last_P_idx), sizeof(last_U_idx), sizeof(last_K));
printf(" (active S-box bias: %f)\n", 16.0 * 4 * 4 * 4 * 4 * 4 / (16 * 16 * 16 * 16 * 16));
temp = dec_to_bin(last_maxIdx, 8);
for(int j = 0; j < 2; j++){
    for(int i = 0; i < 4; i++){
        k5[4 * last_K[j] + i] = temp[4 * j + i];
    }
}
free(temp);

BIT last_P_idx_2[] = {4, 6, 7};
BIT last_U_idx_2[] = {5, 7, 13, 15};
BIT last_K_2[]     = {1, 3};
printf("K5,5..8 / K5,13..16  ->  ");
int last_maxIdx_2 = get_partial_subkey(last_P_idx_2, last_U_idx_2, last_K_2, sizeof(last_P_idx_2), sizeof(last_U_idx_2), sizeof(last_K_2));
printf(" (active S-box bias: %f)\n", 8.0 * 4 * 4 * 4 * 4 / (16 * 16 * 16 * 16));
temp = dec_to_bin(last_maxIdx_2, 8);
for(int j = 0; j < 2; j++){
    for(int i = 0; i < 4; i++){
        k5[4 * last_K_2[j] + i] = temp[4 * j + i];
    }
}
free(temp);
printf(">> Last round's key:      ");
show_bits(k5, 16);
puts("");
```

get_partial_subkey 함수

- 모든 plaintext-ciphertext pair를 이용해 각 key의 bias를 계산, last round's key 값을 반환

```
BIT first_C_idx[] = {9, 10, 11};
BIT first_U_idx[] = {0, 1, 8, 9};
BIT first_K[]     = {0, 2};
printf("K1,1..4 / K1,9..12   ->  ");
int first_maxIdx = get_partial_subkey_2(first_C_idx, first_U_idx, first_K, sizeof(first_C_idx), sizeof(first_U_idx), sizeof(first_K));
printf(" (active S-box bias: %f)\n", 8.0 * 4 * 4 * 4 * 6 / (16 * 16 * 16 * 16));
temp = dec_to_bin(first_maxIdx, 8);
for(int j = 0; j < 2; j++){
    for(int i = 0; i < 4; i++){
        k1[4 * first_K[j] + i] = temp[4 * j + i];
    }
}
free(temp);

BIT first_C_idx_2[] = {10, 11};
BIT first_U_idx_2[] = {4, 5, 12, 13};
BIT first_K_2[]     = {1, 3};
printf("K1,5..8 / K1,13..16  ->  ");
int first_maxIdx_2 = get_partial_subkey_2(first_C_idx_2, first_U_idx_2, first_K_2, sizeof(first_C_idx_2), sizeof(first_U_idx_2), sizeof(first_K_2));
printf(" (active S-box bias: %f)\n", 16.0 * 4 * 4 * 4 * 4 * 4 / (16 * 16 * 16 * 16 * 16));
temp = dec_to_bin(first_maxIdx_2, 8);
for(int j = 0; j < 2; j++){
    for(int i = 0; i < 4; i++){
        k1[4 * first_K_2[j] + i] = temp[4 * j + i];
    }
}
free(temp);
printf(">> First round's key:     ");
show_bits(k1, 16);
```

get_partial_subkey_2 함수

- 모든 plaintext-ciphertext pair를 이용해 각 key의 bias를 계산, first round's key 값을 반환