INTRODUCTION TO CRYPTOGRAPHY – PROJECT 3

B.Tech. Computer Science and Engineering (Cybersecurity)

Name: Anish Sudhan Nair	Roll No.: K041
Batch: K2/A2	Date of submission: 02/03/2022

Code:

Language: C

Editor: Atom

Compiler: clang/ZSH

Consider the simple substitution permutation network shown in Figure 1 on the second page. Assume that the S-box is as given below:

input	000	001	010	011	100	101	110	111
ouput	110	101	001	000	011	010	111	100

In terms of hexadecimal notation, the S-box is given by

input	0	1	2	3	4	5	6	7
ouput	6	5	1	0	3	2	7	4

1. Create the normalized linear approximation table for the given S-box similar to the table given in the slide 18 of sec4.3 class notes.

   Remark: Your input sum and output sum should be from 0 to 7 in hexadecimal notation. So your table should be an 8×8 table and the formula for the entries are $N_L(a, b) - 4$.

2. Find a linear approximation trail with nonzero bias, analogous to slides 22 of section 4.3 in class notes, which relates the plaintext bits P1, P2, P4, and P5 to the bit $H1$. You should sketch the trail and attach as a pdf file.

3. What is the **total bias** of the linear approximation trail you found?

   Note: Remember that $n = 4$ for the original SPN discussed in the class notes. Therefore for each input sum a and the output sum b, the bias was calculated by the formula

   $$\epsilon(a, b) = \frac{N_L(a, b)}{2^n} - \frac{1}{2} = \frac{N_L(a, b)}{16} - \frac{1}{2} = \frac{N_L(a, b) - 8}{16}.$$

   But $n = 3$ for the Simple SPN given in this project, and so

   $$\epsilon(a, b) = \frac{N_L(a, b)}{2^n} - \frac{1}{2} = \frac{N_L(a, b)}{8} - \frac{1}{2} = \frac{N_L(a, b) - 4}{8}.$$

4. Suppose you are given the following known plaintext and ciphertext pairs for this cipher, all encrypted with the same (unknown) key:

Plaintext	Ciphertext
100111	100100
000111	110010
001100	111001
011000	011101
001000	001101
011010	101001

   Using the linear approximation trail from part (2), determine the first and third bits of the subkey K_4.

   Remark: This problem has been specifically constructed so that a very small number of plaintexts and ciphertexts is sufficient to determine two subkey bits.

5. Why is this information insufficient to determine the second bit of the subkey K_4?.

1. Question 1

```
Project — -zsh — 80×29
(base) anish@PotatoBook project % clang linear_approx_table.c -o lat
(base) anish@PotatoBook project % ./lat
Input and Output:
------------------------------------------------------------
x1      x2      x3      y1      y2      y3
------------------------------------------------------------
0       0       0       1       1       0
0       0       1       1       0       1
0       1       0       0       0       1
0       1       1       0       0       0
1       0       0       0       1       1
1       0       1       0       1       0
1       1       0       1       1       1
1       1       1       1       0       0
------------------------------------------------------------
LINEAR APPROXIMATION OF BIAS TABLE
------------------------------------------------------------
|X|0    1       2       3       4       5       6       7
------------------------------------------------------------
|0|4    0       0       0       0       0       0       0
|1|0    -2      -2      0       0       -2      2       0
|2|0    0       -2      -2      0       0       -2      2
|3|0    2       0       2       0       -2      0       2
|4|0    0       2       -2      0       0       2       2
|5|0    2       0       -2      0       -2      0       -2
|6|0    0       0       0       -4      0       0       0
|7|0    2       -2      0       0       2       2       0
------------------------------------------------------------
(base) anish@PotatoBook project %
```

2. Question 2

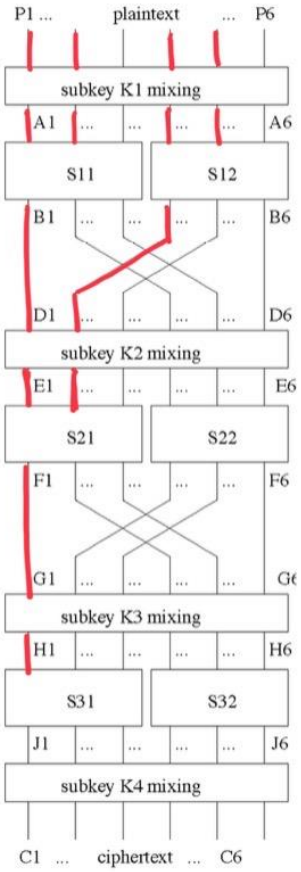


Figure 1: A very simple SPN network

3. Question 3
   ε(a,b)=e(110,100)= (4-8)/8 = -1/2
   Since, n=3 and no.s of s boxes equals 3,
   Total bias = $2^2(-1/2)^3 = -1/2$

4. Question 4

```
Project — -zsh — 80×12
(base) anish@PotatoBook project % clang key_guesser.c -o kg
(base) anish@PotatoBook project % ./kg
The counter values of all keys:
Key 1: 6
Key 2: 2
Key 3: 0
Key 4: 4
Key 5: 4
Key 6: 2
Key 7: 2
Key 8: 4
(base) anish@PotatoBook project %
```

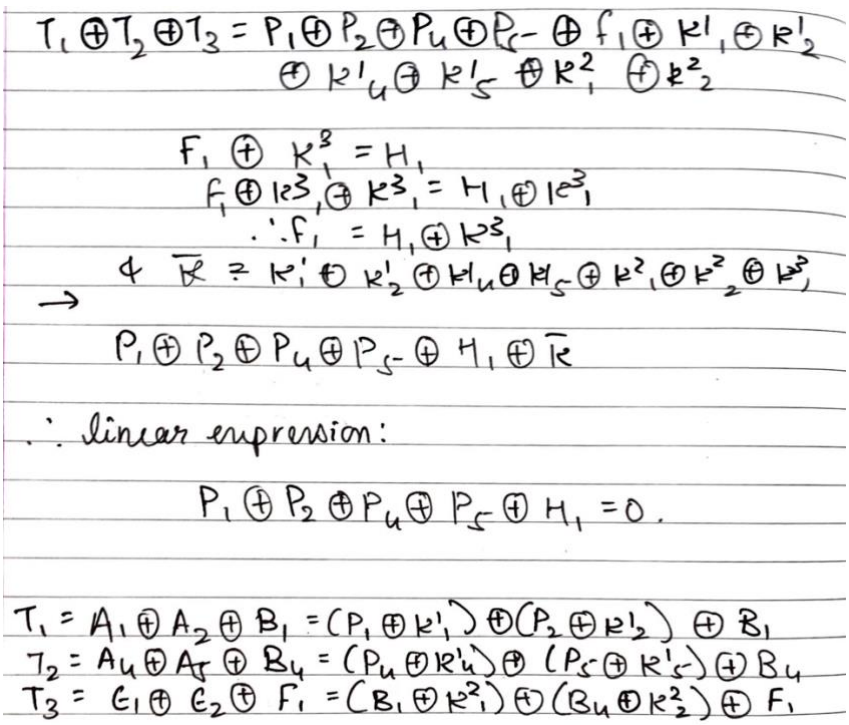
From the counter values, 6 & 0 are applicable as possible key values considering (6/2 +or- 6/2)=0 or 6. These two counter values are analogues to keys 010 and 000.
Hence, Key chosen -> 010 or 000
Therefore, first bit=0
Third bit=0

5. Question 5

As stated in the previous question, due to the limited set of plaintext-ciphertext pairs, the counter values when put through the formula for choosing the appropriate keys yields two possible keys for whom the value of the second bit differ. Hence, it's not possible to determine the second key bit.

CALCULATION:

For Linear Expression:



CODES:

For Normalised Linear Approximation Table:

```c
#include <stdio.h>
int x1[8]={0,0,0,0,1,1,1,1};
int x2[8]={0,0,1,1,0,0,1,1};
int x3[8]={0,1,0,1,0,1,0,1};
int y1[8]={1,1,0,0,0,0,1,1};
int y2[8]={1,0,0,0,1,1,1,0};
int y3[8]={0,1,1,0,1,0,1,0};
int inputRows[8][3]={{0,0,0},{0,0,1},{0,1,0},{0,1,1},{1,0,0},{1,0,1},{1,1,0},{1,1,1}};
int outputRows[8][3]={{1,1,0},{1,0,1},{0,0,1},{0,0,0},{0,1,1},{0,1,0},{1,1,1},{1,0,0}};
int inputModAdds[8];
int outputModAdds[8];

int modCalculator(int positions[3], int row[8][3], int rowNum){

  int value=0;

  for (int i = 0; i < 3; i++) {
    if (positions[i]>=0) {
      value=(value+row[rowNum][i])%2;
    }
  }

  return value;
}

int biasCalculator(int a1, int a2, int a3, int b1, int b2, int b3){

  int count=0;
  int positionsNumerator[3]={a1,a2,a3};
  int positionsDenominator[3]={b1,b2,b3};

  for (int i = 0; i < 3; i++) {
    if (positionsNumerator[i]==0) {
      positionsNumerator[i]=-1;
    }
    if (positionsDenominator[i]==0) {
       positionsDenominator[i]=-1;
    }
  }

  for (int i = 0; i < 8; i++) {
    inputModAdds[i]=modCalculator(positionsNumerator,inputRows, i); //i is row number
    outputModAdds[i]=modCalculator(positionsDenominator,outputRows, i); //i is row number
    if (inputModAdds[i]==outputModAdds[i]) {
      count++;
    }
  }

  int bias;
  bias = count-4;
  return bias;
}
```

```
int main() {

  int c1[8],c2[8],c3[8],c4[8],c5[8],c6[8],c7[8],c8[8];

  printf("Input and Output: \n");
  printf("-------------------------------------------------------------\n" );
  printf("x1\tx2\tx3\ty1\ty2\ty3\n");
  printf("-------------------------------------------------------------\n" );
  for (int i = 0; i < 8; i++) {
    printf("%d\t%d\t%d\t%d\t%d\t%d\n",x1[i],x2[i],x3[i],y1[i],y2[i],y3[i]);
  }
  printf("-------------------------------------------------------------\n" );
  printf("LINEAR APPROXIMATION OF BIAS TABLE\n" );

  int i=0;
  printf("-------------------------------------------------------------\n" );
  printf("|X|0\t1\t2\t3\t4\t5\t6\t7\n");
  printf("-------------------------------------------------------------\n" );

  for (int a1 = 0; a1 <= 1; a1++) {
    for (int a2 = 0; a2 <= 1; a2++) {
      for (int a3 = 0; a3 <= 1; a3++) {
        c1[i]=biasCalculator(a1,a2,a3,0,0,0);
        c2[i]=biasCalculator(a1,a2,a3,0,0,1);
        c3[i]=biasCalculator(a1,a2,a3,0,1,0);
        c4[i]=biasCalculator(a1,a2,a3,0,1,1);
        c5[i]=biasCalculator(a1,a2,a3,1,0,0);
        c6[i]=biasCalculator(a1,a2,a3,1,0,1);
        c7[i]=biasCalculator(a1,a2,a3,1,1,0);
        c8[i]=biasCalculator(a1,a2,a3,1,1,1);

        printf("|%d|%d\t%d\t%d\t%d\t%d\t%d\t%d\n",i,c1[i],c2[i],c3[i],c4[i],c5[i],c6[i],c7[i],c8[i]);
        i++;

      }
    }
  }
  printf("-------------------------------------------------------------\n" );


  return 0;
}
```

For Counter Values (of keys):

```
#include <stdio.h>
char plainTxts[6][6]={"100111","000111","001100","011000","001000","011010"};
char cipherTxts[6][6]={"100100","110010","111001","011101","001101","101001"};
char sBoxInput[8][3]={"000","001","010","011","100","101","110","111"};
char sBoxOutput[8][3]={"110","101","001","000","011","010","111","100"};
char inverse[3];
int keyCount[8];

int expressionSummation(int index){

  int sum=0;
  for (int i = 0; i < 6; i++) {
    sum=(((((((plainTxts[index][0]+plainTxts[index][1])%2)+plainTxts[index][3])%2)+plainTxts[index][4])%2)+inverse[0])%2;
    //linear expression derived via manual calculation
  }
  return sum;
}

void inverseCalculator(char value[]){

  for (int i = 0; i < 8; i++) {
    int count=0,outputIndex=0;
    for (int index = 0; index < 3; index++) {
      // printf("Working, Value=%c, So=%c\n", value[0],sBoxOutput[i][index]);
      if (value[index]==sBoxOutput[i][index]) {
```

```
            count++;
            outputIndex=i;
            // printf("Running for i=%d\n", i);
          }
        }
        if (count==3) {
          // printf("\nInverse \n:");
          for (int i = 0; i < 3; i++) {
            inverse[i]=sBoxInput[outputIndex][i];
            // printf("%c",inverse[i] );
          }
        }
      }
    }

    int main(){

      int i=0,counter,sum;

      for (int a1 = 0; a1 <= 1; a1++) {
        for (int a2 = 0; a2 <= 1; a2++) {
          for (int a3 = 0; a3 <= 1; a3++) {

            char key[3]={a1,a2,a3};
            char value[3];
            counter=0,sum=0;
            for (int indexOuter = 0; indexOuter < 6; indexOuter++) {
              for (int indexInner = 0; indexInner < 3; indexInner++) {
                int temp_val=((key[indexInner]+cipherTxts[indexOuter][indexInner])%2);
                value[indexInner]=temp_val+'0';
              }
              inverseCalculator(value);
              sum=expressionSummation(indexOuter);
              if (sum==0) {
                counter++;
              }
            }
            keyCount[i]=counter;
            i++;

          }
        }
      }

      printf("The counter values of all keys: \n");
      for (int i = 0; i < 8; i++) {
        printf("Key %d: %d\n",i+1,keyCount[i]);
      }

      return 0;
    }
```