

Report for the assignment of Linear Cryptanalysis of Feal 4 Cipher

Name: Vikrant Singh, Student No: 21262315, Module name: CA642 Cryptography & Number Theory,

Methodology Used:

Step1. The first task is to read the file provided 'known.txt'. So, the function in *read_file()* is doing that, it reads the data from the file and store it in an array list called *text*.

Step2. Then function *stripdata()* is created, which reads the array list text. Looking at the file created a small condition that separates data and strips unwanted text and then stores the data into two separate array lists called *plaintext* and *ciphertext*.

Step3. Then there are functions in code that are helpful in the main algorithm of linear cryptanalysis and the description of these functions are as follows:

- ***btoh()***: This function takes a byte array as input and converts it into hex value so that we can assure the radix 16 values we have are correct and also at the end this function is used to convert key values in byte array to hex values and then we store them.
- ***htob()***: This function takes a string and makes it into a real hex value to make sure that the string we got from the file is hexadecimal values.
- Then some functions are required for the calculations used in linear cryptanalysis and these functions are *F()* which takes a byte array as an input and then calculate *y0*, *y1*, *y2* and *y3* using round functions *g0()*, *g1()* and *rot()* and then there is a function called *concat_word()* which combines all the results means *y0*, *y1*, *y2* and *y3* and returns full hex string of 32 bit.
- ***Xor()*** function is used to find the xor of the two-byte array because we can't use direct xor operations as we do with an integer so using a function to return a xor of the two-byte array.
- ***Convertbytearraytoint2()*** is a function used to unpack our byte array of hex values to its respected integer values so that we can perform shift operations to get particular bits we required.

Step4. *main()* is phase 1 for the algorithm where we are trying to guess the middle 16 bits for the key and for those calculations we have to compare the value of our constant for all known plaintext and ciphertext values which are 200 for us in both the cases.

The equation for the constant is

$$S5,13(L0 \oplus R0 \oplus L4) \oplus S21(L0 \oplus R0 \oplus L4) \oplus S15(L0 \oplus L4 \oplus R4) \oplus S15(F(L0 \oplus R0 \oplus K \sim 0))$$

Let's start collecting elements required to calculate this equation:

Firstly two loops in code generate all possible values for middle bits of this type:

```
byte[] K0_prime = new byte[]{0, a0, a1, 0};
```

then there is another loop that collects our 200 combinations of plaintext and ciphertext and inside that loop there we use our *htob()* function first and convert our plaintext and ciphertext from two different lists to hex values and then divide them into 32-bit parts L0, R0 for plaintext and L4, R4 for ciphertext as follows:

```
byte L0[] = Arrays.copyOfRange(full_plaintext, 0, 4);
byte R0[] = Arrays.copyOfRange(full_plaintext, 4, 8);
byte L4[] = Arrays.copyOfRange(full_ciphertext, 0, 4);
byte R4[] = Arrays.copyOfRange(full_ciphertext, 4, 8);
```

Now our next task is to find all the xor's required for the calculation of the constant using the *xor()* function defined above in the code as follows:

```
byte[] xor_1_1 = xor(L0,R0);
byte[] xor_1_final = xor(xor_1_1,L4);
byte[] xor_2_1 = xor(L0,L4);
byte[] xor_2_final = xor(xor_2_1,R4);
```

Also, xor in which K0_prime is there and pass its *F()* for round calculations as follows:

```
byte[] xor_roundfunction_1 = xor(L0,R0);
byte[] xor_roundfunction_final = F(xor(xor_roundfunction_1,K0_prime));
```

Now let's calculate the particular bits we required for the value of our constant and we do it ((xor value>>(31-location of bit we want))&1) and before doing these operations convert the values to their respective hex integer value using function *convertbytearraytoint2()*:

```
int bit_5 = (convertByteArrayToInt2(xor_1_final)>>26)&1;
int bit_13 = (convertByteArrayToInt2(xor_1_final)>>18)&1;
int bit_21 = (convertByteArrayToInt2(xor_1_final)>>10)&1;
int bit_15_1 = (convertByteArrayToInt2(xor_roundfunction_final)>>16)&1;
```

```
int bit_15_2 = (convertByteArrayToInt2(xor_2_final)>>16)&1;
```

All bits required for our constant calculation is there so now let's calculate the constant:

```
int constant = (bit_5^bit_13^bit_21^bit_15_1^bit_15_2);
```

Compare the value of all 200 constant values and if it's 0 or 1 200 times end-use that case and pass them for phase operations.

Step5. Now we have all possible K0_prime which are middle 16 bits, so let's now recover the remaining 16 bits of the key, here call *phase2()* function with last passed case values of middle 16 bits.

The equation used for the second round is as follows:

$$a = S_{23,29}(L_0 \oplus R_0 \oplus L_4) \oplus S_{31}(L_0 \oplus L_4 \oplus R_4) \oplus S_{31}(F(L_0 \oplus R_0 \oplus K_0))$$

Key in this round looks like following:

```
byte[] K = new byte[] {b0, (byte) (a0^b0), (byte) (a1^b1), b1};
```

Similarly, as the previous face, we have two loops to generate all possible values of a key, this time loops enumerate all possible for the first and last byte and middle bits obtained by xor then-current values and values from the last phase.

After that similarly, divide data into required parts and calculate all the required xor operations as follows:

```
byte L0[] = Arrays.copyOfRange(full_plaintext, 0, 4);
byte R0[] = Arrays.copyOfRange(full_plaintext, 4, 8);
byte L4[] = Arrays.copyOfRange(full_ciphertext, 0, 4);
byte R4[] = Arrays.copyOfRange(full_ciphertext, 4, 8);

byte[] xor_1_1 = xor(L0,R0);
byte[] xor_1_final = xor(xor_1_1,L4);
byte[] xor_2_1 = xor(L0,L4);
byte[] xor_2_final = xor(xor_2_1,R4);

byte[] xor_roundfunction_1 = xor(L0,R0);
byte[] xor_roundfunction_final = F(xor( xor_roundfunction_1,K));
```

Then calculate bits according to the equation of phase 2 as follows:

```
int bit_23 = (convertByteArrayToInt2(xor_1_final)>>8)&1;
```

```

int bit_29 = (convertByteArrayToInt2(xor_1_final)>>2)&1;

int bit_31_1 = (convertByteArrayToInt2(xor_roundfunction_final)>>0)&1;

int bit_31_2 = (convertByteArrayToInt2(xor_2_final)>>0)&1;

```

And then calculate our constant using all these bits and compare its values again 200 times if count_0 or count_1 is 200 then save the value of that key and here I have 1020 values which are nearly 2^{10} and these values are then converted into their respective hexadecimal values using *both()* function and stored in an array list called a key.

```

int constant = (bit_23^bit_29^bit_31_1^bit_31_2);

key.add(btoh(K));

```

After that, I used another equation to find another set of possible values of K0 and I got 2040 values for that equation, store them in a another array list called *keyguess2* < > and below is the code used to find the values of the second equation :

$$S_{5,15}(L_0 \oplus R_0 \oplus L_4) \oplus S_7(L_0 \oplus L_4 \oplus R_4) \oplus S_7(F(L_0 \oplus R_0 \oplus K_0))$$

```

int bit_5 = (convertByteArrayToIntger(xor_1_final)>>26)&1;

int bit_15 = (convertByteArrayToIntger(xor_1_final)>>16)&1;

int bit_7_1 = (convertByteArrayToIntger(xor_2_final)>>24)&1;

int bit_7_2 = (convertByteArrayToIntger(xor_roundfunction_final)>>24)&1;

int constant2 = bit_5^bit_15^bit_7_1^bit_7_2;

```

Then just take the intersection of array list results generated with 1020 and 2040 results in array lists *key* < > *and* *keyguess2* < > and then we have narrow results of possible K0 values which are 16 possible results.

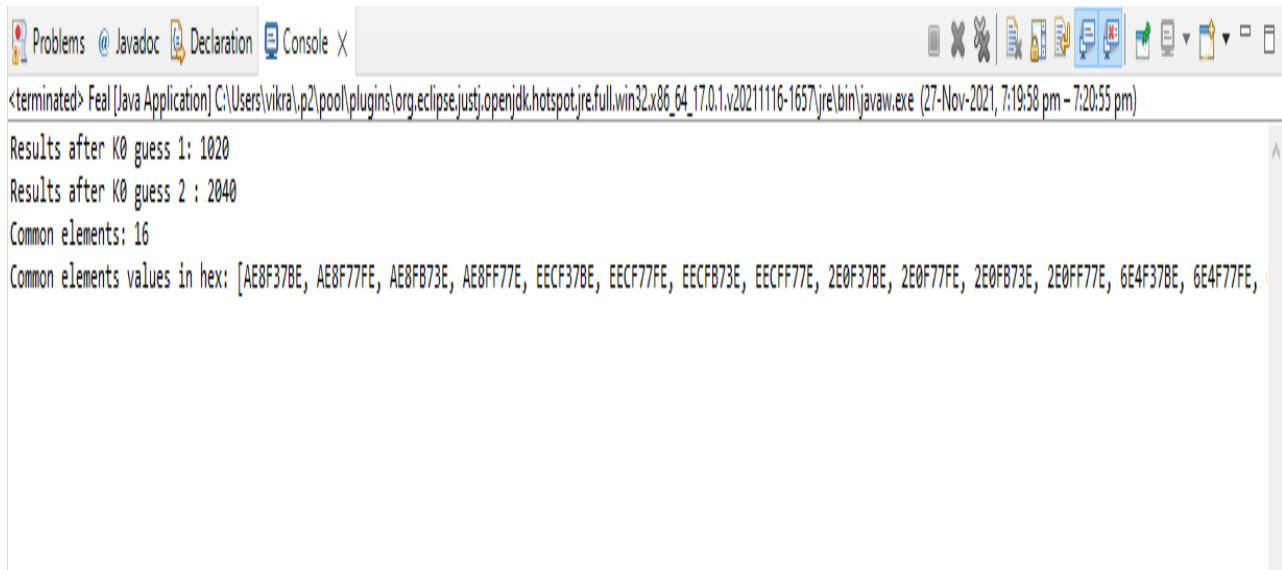
```

key.retainAll(keyguess2);

System.out.println("Common elements: "+ key.size());

```

Output



```
<terminated> Feal [Java Application] C:\Users\vikraip2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe (27-Nov-2021, 7:19:58 pm - 7:20:55 pm)
Results after K0 guess 1: 1020
Results after K0 guess 2 : 2040
Common elements: 16
Common elements values in hex: [AE8F37BE, AE8F77FE, AE8FB73E, AE8FF77E, EECF37BE, EECF77FE, EECFB73E, EECFF77E, 2E0F37BE, 2E0F77FE, 2E0FB73E, 2E0FF77E, 6E4F37BE, 6E4F77FE, ]
```

Getting Results for K1 Possible values

Phase1_k1 () and *Phase2_k1* () functions defined in the code are used for finding K1_prime and K1.

Equation used for the generation of K1_prime which produce 256 results:

$$a = S_{15}(F(X1 \text{ xor } K1_prime) \text{ xor } S_{15}(L0 \text{ xor } R0) \text{ xor } S_{5, 13, 21}(L4 \text{ xor } R4 \text{ xor } X1))$$

Equations used for K1 possible values produce 4080 and 8160 results respectively:

$$a = S_{31}(F(X1 \text{ xor } K1)) \text{ xor } S_{31}(L0 \text{ xor } R0) \text{ xor } S_{23, 29}(L4 \text{ xor } R4 \text{ xor } X1)$$

$$a = S_7(F(X1 \text{ xor } K1)) \text{ xor } S_7(L0 \text{ xor } R0) \text{ xor } S_{5, 15}(L4 \text{ xor } R4 \text{ xor } X1)$$

$$\text{Where } X1 = ((F(R0 \wedge L0 \wedge K0)) \wedge L0);$$

Intersection narrow down results are 64.

Output

```

Searching for K0 possible values.....
Results after K0 guess 1: 1020
Results after K0 guess 2 : 2040
Common elements: 16
Common elements values in hex: [AE8F37BE, AE8F77FE, AE8FB73E, AE8FF77E, EECF37BE, EECF77FE, EECFB73E, EECFF77E, 2E0F37BE, 2E0F77FE, 2E0FB73E, 2E0FF77E, 2E0FB73E, 2E0FF77E, 2E0FB73E, 2E0FF77E]
Searching for K1 possible values.....
Results after K1_prime guess : 256
Results after K1 guess 1 : 4080
Results after K1 guess 2 : 8160
Common elements: 64
Common elements values in hex: [B1EC90B8, B1ECD0F8, B1EC1038, B1EC5078, F1AC90B8, F1ACD0F8, F1AC1038, F1AC5078, 316C90B8, 316CD0F8, 316C1038, 316C5078, 316C90B8, 316CD0F8, 316C1038, 316C5078]

```

Getting Results for K2 Possible values

Phase1_k2 () and *Phase2_k2* () functions defined in the code are used for finding K1_prime and K1.

Equation used for k2_prime:

$$a = S_{15}(F(X2 \text{ xor } K2) \text{ xor } S_{15}(X1) \text{ xor } S_{5,13,21}(L4 \text{ xor } X2))$$

Equations used for final K2 values:

$$a = S_{31}(F(X2 \text{ xor } K2)) \text{ xor } S_{31}(X1) \text{ xor } S_{23,29}(L4 \text{ xor } X2)$$

$$a = S_7(F(X2 \text{ xor } K2)) \text{ xor } S_7(X1) \text{ xor } S_{5,15}(L4 \text{ xor } X2)$$

Where $X2 = (F(X1 \wedge K1) \wedge L0 \wedge R0)$;

Results for K2 are taking very long time and I think that happens because my code is bit inefficient.

Work done on equations of K1, K2 and K3

Tryout 1 for K1

Following Equation is used as a first tryout for the equation of k1 this approach produce 1040400 results for K1_prime, which makes it very time ineffective for K1.

K1 equation with 32-bit dependency (Test 1)

$$(1) S_{23,29}(L4) = S_{23,29}(X0) \oplus S_{3,29}(Y3) \oplus S_{23,29}(Y1) \oplus S_{23,29}(K4)$$

using equations from future

$$S_{23,29}(X0) = S_{23,29}(L0 \oplus R0)$$

$$S_{23,29}(Y3) = S_{3,29}(L4 \oplus R4) \oplus S_{3,29}(K4 \oplus K5 \oplus K3) \oplus 1$$

$$S_{23,29}(Y1) = S_{3,29}(K1) \oplus S_{3,29}(Y0) \oplus S_{3,29}(L0) \oplus 1$$

$$S_{3,29}(Y0) = S_{3,29}(F(L0 \oplus R0 \oplus K0))$$

using all these equations - in (1)

$$a = S_{23,29}(L0 \oplus R0 \oplus L4) \oplus S_{3,29}(L0 \oplus L4 \oplus R4) \oplus S_{3,29}(F(L0 \oplus R0 \oplus K0)) \oplus S_{3,29}(K1)$$

$$\text{where } a = S_{3,29}(K3 \oplus K4 \oplus K5) \oplus S_{23,29}(K4)$$

K1 equation with middle 16-bit dependency

$$(1) S_{13}(Y) = S_{7,15,23,31}(X) \oplus 1 \quad (2) S_{5,15}(Y) = S_7(X) \quad (3) S_{5,21}(Y) = S_{23,31}(X)$$

eliminate common elements S_7, S_{15}, S_{23} and S_{31}

$$S_{5,13,21}(Y) = S_{15}(X) \oplus 1$$

$$S_{5,13,21}(L4) = S_{5,13,21}(X0) \oplus S_{5,13,21}(Y3) \oplus S_{5,13,21}(Y1) \oplus S_{5,13,21}(K4)$$

As in above part fill all values.

$$S_{5,13,21}(X0) = S_{5,13,21}(L0 \oplus R0)$$

$$S_{5,13,21}(Y3) = S_{5,13,21}(L4 \oplus R4) \oplus S_{5,13,21}(K4 \oplus K5 \oplus K3) \oplus 1$$

$$S_{5,13,21}(Y1) = S_{5,13,21}(K1) \oplus S_{15}(Y0) \oplus S_{15}(L0) \oplus 1$$

$$S_{5,13,21}(Y0) = S_{15}(F(L0 \oplus R0 \oplus K0))$$

Combining these equations.

$$a = S_{5,13,21}(L0 \oplus R0 \oplus L4) \oplus S_{5,13,21}(L0 \oplus L4 \oplus R4) \oplus S_{15}(F(L0 \oplus R0 \oplus K0)) \oplus S_{15}(K1)$$

Output

```
Problems @ Javadoc Declaration Console X
<terminated> Feal [Java Application] C:\Users\vikra.p2\pools\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe (28-Nov-
Searching for K0 possible values.....
Results after K0 guess 1: 1020
Results after K0 guess 2 : 2040
Common elements: 16
Common elements values in hex: [AE8F37BE, AE8F77FE, AE8F873E, AE8FF77E, EECF37BE, EECF77FE, EECF873E, EECFF77E, 2E0F37BE, 2E0F77FE,
Searching for K1 possible values.....
Results after K1_pime guess : 1040400
```

Tryout 2 for K1:

Then after reading some comments on future learn, I figured out that for round 2 we have new text say L1 and R1 which are the result after the first round, I worked on a new equation that depends on L1 and R1 and K1 and I am attaching my work on finding this equation:

Text 2 of equation for K1 according to comments mention on future learn

K1 equation with 32-bit dependency

Suppose after first round we have new L1 and R1 for next round.

$$S_{23,29}(L4) = S_{23,29}(Y1) \oplus S_{23,29}(Y3) \oplus S_{23,29}(Y4) \quad \text{--- (1)}$$

Now start collecting all values.

$$S_{23,29}(Y1) = S_{23,29}(F(X1 \oplus K1))$$

And $X1 = R1$

$$S_{23,29}(Y1) = S_{23,29}(F(R1 \oplus K1))$$

$$S_{23,29}(Y3) = S_{31}(L4 \oplus R4) \oplus S_{31}(K4 \oplus K5 \oplus K3) \oplus 1$$

at values in (1)

$$a = S_{23,29}(L4) \oplus S_{31}(L4 \oplus R4) \oplus S_{23,29}(F(R1 \oplus K1))$$

$$a = S_{23,29}(L4) \oplus S_{23,29}(F(R1 \oplus K1)) \oplus S_{31}(L4 \oplus R4)$$

K1 equation with dependency on Middle 16-bits

$$a = S_{5,13,21}(L4) \oplus S_{5,13,21}(F(R1 \oplus K1)) \oplus S_{15}(L4 \oplus R4)$$

where $a = (K4 \oplus K5 \oplus K3)$

Similarly, I also did some work to find the equations for K2 and K3 which depends on L2, R2 and L3 and R3 respectively and these are as follows:

K2 equations : Try out 1

Tryout for K2 equations
K2 equation for 32-bit dependency

$$S_{23,24}(R_4) = (Y_2 \oplus Y_3 \oplus K_4)$$

Suppose that after second round we have L2 and R2 as text for the next round.

$$S_{23,24}(Y_2) = S_{23,24}(F(X_2 \oplus K_2)) = S_{23,24}(F(R_2 \oplus K_2))$$

where $Y_2 = R_2$

$$S_{23,24}(Y_3) = S_{23,24}(F(X_3 \oplus K_3))$$

$$= S_{31}(X_3 \oplus K_3) \oplus 1$$

$$= S_{31}(L_4 \oplus K_4 \oplus R_4 \oplus K_5 \oplus K_3) \oplus 1$$

$$= S_{31}(L_4 \oplus R_4) \oplus S_{31}(K_4 \oplus K_5 \oplus K_3) \oplus 1$$

Put values in ① and rearrange

$$a = S_{23,24}(R_4) \oplus S_{23,24}(F(R_2 \oplus K_2)) \oplus S_{31}(L_4 \oplus R_4)$$

where $a = S_{31}(K_4 \oplus K_5 \oplus K_3)$

K2 equation with 16-bit dependency

$$a = S_{5,13,21}(R_4) \oplus S_{5,13,21}(F(R_2 \oplus K_2)) \oplus S_{15}(L_4 \oplus R_4)$$

K2: Tryout 2

Tryout for K3 values (another)

$$S(L4) = S(K4) \oplus S(Y3) \oplus S(X2)$$

$$\begin{aligned} S(Y3) &= S(F(X3 \oplus K3)) \\ &= S'(X3) \oplus S'(K3) \end{aligned}$$

$$\begin{aligned} &= S'(Y2 \oplus X1) \oplus S'(K3) \\ &= S'(X1) \oplus S'(F(X2 \oplus K2)) \oplus S'(K3) \end{aligned}$$

$$a = S(L4) \oplus S'(X1) \oplus S(X2) \oplus S'(F(X2 \oplus K2))$$

$$a = S(L4 \oplus X2) \oplus S'(X1) \oplus S'(F(X2 \oplus K2))$$

$$a = S_{15}(F(X2 \oplus K2)) \oplus S_{15}(X1) \oplus S_{15}(F(L4 \oplus X2))$$

32 bit dependant equation

$$a = S_{31}(F(X2 \oplus K2)) \oplus S_{31}(X1) \oplus S_{23,29}(L4 \oplus X2)$$

$$\text{Where } X2 = (F(X1 \oplus K1) \oplus L0 \oplus R0)$$

K3 equation :

Tryout for K3

$$\begin{aligned} S_{23,29}(L4) &= S_{23,29}(Y3) \oplus S_{23,29}(K4) \\ &= S_{23,29}(F(X3 \oplus K3)) \oplus S_{23,29}(K4) \\ &= S_{23,29}(F(R3 \oplus K3)) \oplus S_{23,29}(K4) \end{aligned}$$

$$\text{Where } X3 = R3$$

$$a = S_{23,29}(L4) \oplus S_{23,29}(F(R3 \oplus K3))$$

Where $a = S_{23,29}(K4)$

16 bit - dependent equation

$$a = S_{5,13,21}(L4) \oplus S_{5,13,21}(F(R3 \oplus K3))$$