Declaration on Plagiarism

Assignment Submission Form

This form must be filled in and completed by the student(s) submitting an assignment

Name(s)	Vinit Saini
Programme	MSc in Computing - Blockchain
Module Code	CA642I
Assignment Title	Differential Cryptanalysis of the FEAL-4 Cipher
Submission Date	27 November 2022
Module Coordinator	Geoff Hamilton

Name(s): _____Vinit Saini______________ Date: ___27 November 2022_______

Differential Cryptanalysis of FEAL-4

Java implementation

Introduction

Fast data Encipherment Algorithm (FEAL), is a block-cipher that was proposed as an alternative to the standard Data Encryption Standard (DES) algorithm and designed to be much faster implemented in software. However, The algorithm is prone to various kinds of attacks. Differential and Linear attacks are the most common ones. The structure of FEAL is similar to the DES algorithm with a modified F-function and key scheduling algorithm. The effect of permutations (P-boxes) and substitutions (S-boxes) of DES cipher are achieved through byte level rotation and addition.

FEAL uses a round function F that consists of $G_i(a,b)$ sub-functions which are further responsible for byte manipulations to achieve the permutation and substitution effect. *(a, b)* are two bytes inputs to the G_i function which returns the single-byte output.

Figure 1 denotes the implementation detail of $G_i(a,b)$ sub-functions

$$G_0(a,b) = (a + b \pmod{256}) <\!<\!< 2$$

$$G_1(a,b) = (a + b + 1 \pmod{256}) <\!<\!< 2$$

Figure 1

Figure 2 below denotes the round function F in a schematical view where 4 input bytes correspond to 32-bit input being passed through round function F to generate the 32-bit output.

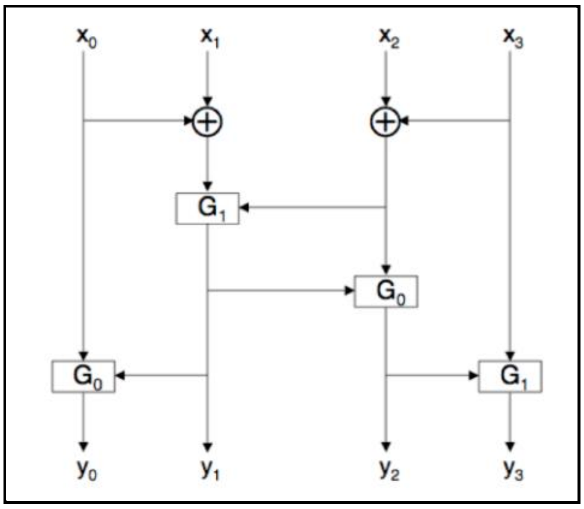


Figure 2

Initially, FEAL was suggested as a four-round cipher, called FEAL-4. However, after the cryptanalysis of FEAL-4 the eight-round version FEAL-8 was also introduced. Again, another form having N number of rounds FEAL-N was also introduced later.

FEAL-4

FEAL-4 is a 4-round cipher that uses a Feistel structure with a 64-bit block and a 64-bit key. A key scheduler expands the 64-bit key into twelve 16-bit round keys. Some of these subkeys are used for key whitening, in which they are combined with portions of the data using XOR, before the first round and after the last round. Figure 3 represents a block diagram for the FEAL-4 structure. Subkeys K_4 and K_5 are being used for key whitening purposes in the diagram whereas subkeys from K_0 to K_3 are being used in relation to each round respectively.

Round function F is the function that breaks the linearity in FEAL-4. It takes a 4 bytes input and produces a 4 bytes output. In all 4 rounds, it is being used to encrypt the input. The behaviour of round function F defines the strength of the FEAL-4 cipher against statistical attacks like differential cryptanalysis.

In each round, the data block is split into two halves, the left and right portions. In the first round, Input to round function F is determined by XORing the left and right portion of the block with the round subkey K_0. i.e.

$$K_0 \oplus P_{left} \oplus P_{right}$$

The output of function F is then XORed with P_{left} (left portion of block) to make the new left portion. Then these resulting portions are swapped to form new P_{left} and P_{right} for the next round. The key whitening process happens before the first round of encryption, XORing K_4 and K_5 with the original block.

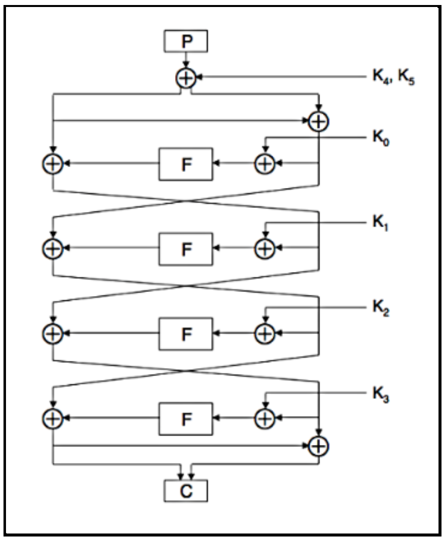


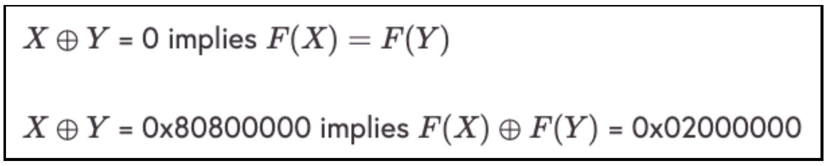
Figure 3

Differential Cryptanalysis

Differential cryptanalysis is a general form of cryptanalysis applicable primarily to block ciphers. It is the study of how differences in information input can affect the resultant difference in the output. In the case of a block cipher, it refers to a set of techniques for tracing differences through the network of transformation, discovering where the cipher exhibits behaviour and exploiting such properties to recover the secret key (Wikipedia, 2022).

Some specific differences in inputs which provide the advantage in such cryptanalysis are called characteristics. These differentials are known to provide specific differences in their outputs when passed through the function F. For example, in figure 4, X and Y are two inputs such that X ⊕ Y = 0x80800000 will always produce the F(X) ⊕ F(Y) = 0x02000000. Hence, characteristic 0x80800000 can be utilised to make differential cryptanalysis possible.

Differential cryptanalysis depends on two basic properties which are as follows



$X \oplus Y$ = 0 implies $F(X) = F(Y)$

$X \oplus Y$ = 0x80800000 implies $F(X) \oplus F(Y)$ = 0x02000000

Figure 4

Differential properties for FEAL-4

Firstly, we choose two plaintexts P_0 and P_1 such that

$$\boldsymbol{P_0 \oplus P_1 = 0x8080000080800000}$$

This is known as the input difference or characteristic. The reason for choosing 0x80800000 is that this differential has a very high probability of producing 0x02000000 when passed through function F. This can be verified easily by passing it into function F.

0x80800000 → 0x02000000

Another property of function F is if the input difference is zero i.e. 0x00000000 It is certain to produce the output difference of 0x00000000

0x00000000 → 0x00000000

Figure 5 below highlights how these differential characteristics flow down to the FEAL-4 structure.

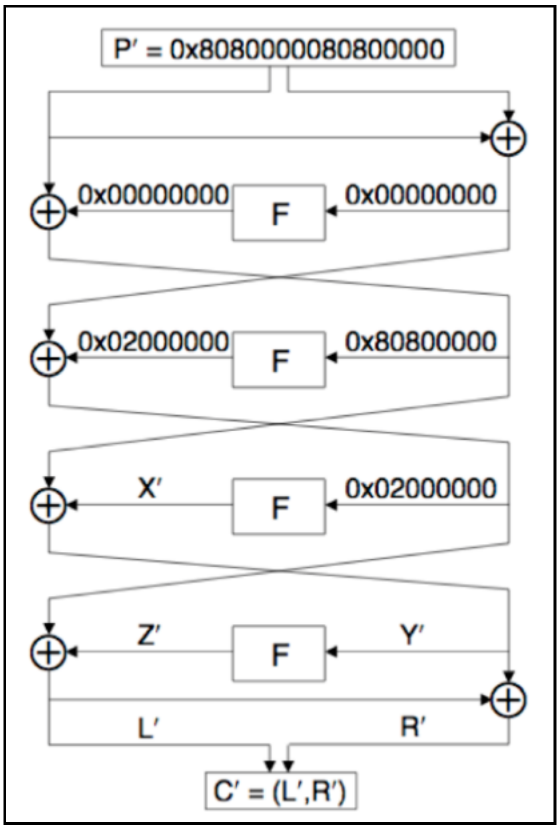


Figure 5

Algorithm

Prerequisites

1. The prerequisite is to generate plaintext pairs with some differential characteristics e.g. ***0x8080000080800000*** for round 4th such as $\boldsymbol{P_0 \oplus P_1 = 0x8080000080800000}$
2. Compute their corresponding ciphertexts and store them as $\boldsymbol{C_0}$ and $\boldsymbol{C_1}$.
3. Similarly, for round 3rd we compute pairs of plaintext and ciphertexts with differential ***0x0000000080800000*** and for 2nd round, we use differential ***0x0000000002000000.***

The algorithm starts with finding the K_3 (last round subkey). We exploit the generated ciphertext pairs as this is a chosen ciphertext attack.

1. The algorithm computes the left and right portions of each ciphertext pair to compute the **Z'** by XORing the difference of the ciphertext pair's left halves with a known differential characteristic such as ***0x02000000.***

$$\boldsymbol{Z' = \Delta(\,cipherLeft0\,,\,cipherLeft1) \wedge 0x02000000}$$

2. Then, the Algorithm searches for a potential key in a $\mathbf{2^{32}}$ key space. It picks a putative key value from 0x0L to 0xFFFFFFFFL range and validates if it satisfies the given ciphertext pair. To check if the key is a valid key, we compute Z_0 and Z_1 using the function ***F*** which takes the input value of XORing the key with the corresponding right halves of the ciphertext.

   Z_0 ***= F(cipherR0 ^ putativeKey)***
   Z_1 ***= F(cipherR1 ^ putativeKey)***

3. At last, the algorithm compares the XOR of Z_0 and Z_1 with **Z'**. If matches, this key is a potential candidate for the round subkey. This key is validated through all of the given ciphertext pairs to find the maximum satisfaction score. If the key satisfies all of the ciphertext pairs, we consider it a valid subkey for the round.

   Z' = = Δ(Z_0 ***,*** Z_1***) for all ciphertext pairs; if true, key is a potential candidate***

Note – There can be many keys that hold true for this condition. However, the algorithm just returns the very first key found that has a score equal to the number of chosen ciphertext pairs in favour of the speed optimization.

Appendix 1 lists all the valid subkeys for each round when 12 plaintext-ciphertext pairs were used.

4. Once the K_3 subkey is found, this is used to substitute in round 4 to solve it. After solving it we get the output ciphertext pairs for the previous round which will be used further to find out the K_2 now. Similarly, it works for all rounds going upward in reverse order from finding K_3 first followed by K_2, K_1, K_0, K_4 and K_5.

5. Input differentials that have been used for each round are as follows.
   - ***Round 4: 0x8080000080800000 → 0x02000000***
   - ***Round 3: 0x0000000080800000 → 0x02000000***
   - ***Round 2: 0x0000000002000000 → 0x02000000***

6. Finding K_0, K_4 and K_5 is a little bit different but quite straightforward. The algorithm chooses a putative key from 0x0L to 0xFFFFFFFF range. Then using this putative key we find a $temp_{left}$ variable with the help of the ***F*** function as –

   $\boldsymbol{temp_{left}}$ ***= F(cipherR0 ^ putativeKey) ^ cipherL0***

   Once we have this $temp_{left}$ variable, it can be simply used to find out the K_4 and K_5. Since, as per the FEAL-4 structure, K_4 and K_5 were XORed initially with left and right portions of plaintext only. Finding K_4 and K_5 is pretty easy using plaintext and ciphertexts.

$$K_4 = temp_{left} \wedge Plaintext_{left}$$
$$K_5 = tempLeft \wedge Ciphertext_{right} \wedge Plaintext_{right}$$

However, This value of K_0 must hold true for all plaintext pairs. Therefore we verify this K_0 to return identical K_4 and K_5 for all pairs.

This is how the algorithm identifies the correct K_0, K_4 and K_5.

Programme Output

```
6 Discovered keys are as follows:
Key0 :  0x15BD591A
Key1 :  0x1C2059A1
Key2 :  0x604D51F6
Key3 :  0x73487836
Key4 :  0xE8324AF4
Key5 :  0x15D0AF18
```

Developer notes

Coding language

```
Java (v8)
```

Java classes

```
Solution.java // main program class
TextProvider.java // inner class implementation as data provider
```

`Solution.java` needs a command line argument input from the user, this input suggests the number of pairs to do the cryptanalysis of FEAL-4.

Possible values are from `4` to `12` where `4` is the minimum number of chosen plaintext-ciphertext pairs required and `12` is the maximum supported by the `TextProvider.java,` data provider class.

Command to compile the program

```
javac solution.java
```

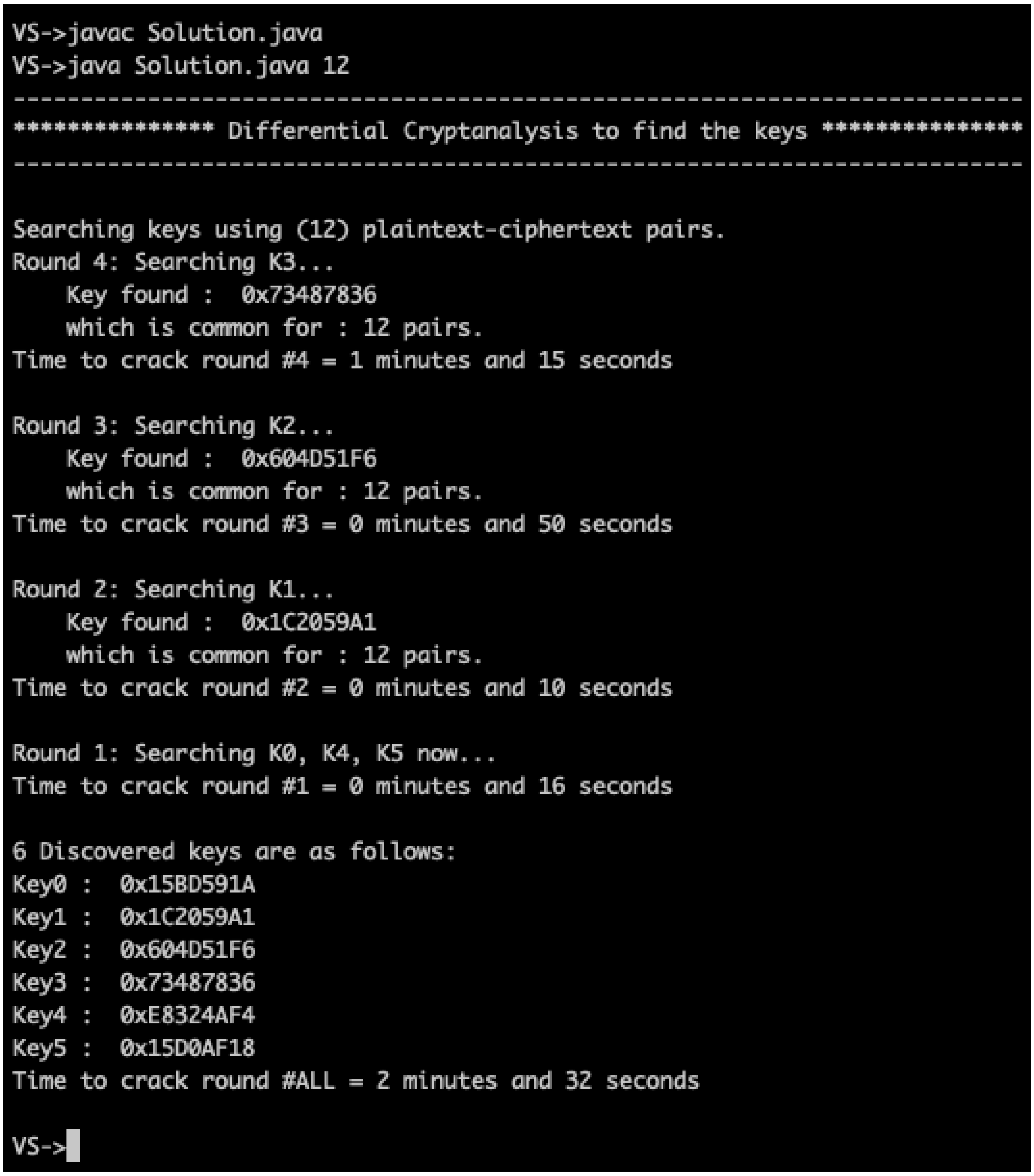
Command to run the program

```
java solution.java 12
```

Example output when run with 12 pairs of plaintext.

Hardware Specification

MacbookPro [Processor (2.4 GHz 8-Core Intel Core i9) Memory (32 GB 2667 MHz DDR4)]



```
VS->javac Solution.java
VS->java Solution.java 12
---------------------------------------------------------------------------
*************** Differential Cryptanalysis to find the keys ***************
---------------------------------------------------------------------------

Searching keys using (12) plaintext-ciphertext pairs.
Round 4: Searching K3...
    Key found :  0x73487836
    which is common for : 12 pairs.
Time to crack round #4 = 1 minutes and 15 seconds

Round 3: Searching K2...
    Key found :  0x604D51F6
    which is common for : 12 pairs.
Time to crack round #3 = 0 minutes and 50 seconds

Round 2: Searching K1...
    Key found :  0x1C2059A1
    which is common for : 12 pairs.
Time to crack round #2 = 0 minutes and 10 seconds

Round 1: Searching K0, K4, K5 now...
Time to crack round #1 = 0 minutes and 16 seconds

6 Discovered keys are as follows:
Key0 :  0x15BD591A
Key1 :  0x1C2059A1
Key2 :  0x604D51F6
Key3 :  0x73487836
Key4 :  0xE8324AF4
Key5 :  0x15D0AF18
Time to crack round #ALL = 2 minutes and 32 seconds

VS->
```

Appendix 1

The below table lists all valid keys for each round.
Highlighted ones are those which were returned by the algorithm.

K_3	**K_2**	**K_1**	**K_0**	**K_4**	**K_5**
0x73487836,	0x604D51F6,	0x1C2059A1,	0x15BD591A	0xE8324AF4	0x15D0AF18
0x7348F8B6,	0x604DD176,	0x1C20D921,	0x15BDD99A	0xE8324AF6	0x15D0AF1A
0xF3C87836,	0xE0CD51F6,	0x9CA059A1,	0x953D591A	0xEA324AF4	0x17D0AF18
0xF3C8F8B6	0xE0CDD176	0x9CA0D921	0x953DD99A	0xEA324AF6	0x17D0AF1A

Bibliography

Differential_cryptanalysis (2022) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Differential_cryptanalysis (Accessed: 26, November 2022).

Differential Cryptanalysis Tutorial (no date) *The Amazing King*. Available at: http://www.theamazingking.com/crypto-diff.php (Accessed: 26, November 2022).

Differential Cryptanalysis of FEAL (no date) *The Amazing King*. Available at: http://www.theamazingking.com/crypto-feal.php (Accessed: 26, November 2022).

Future Learn (2022) *Differential cryptanalysis of FEAL-4* [MOOC]. Available at: https://www.futurelearn.com/courses/hash-functions-cryptanalysis/4/steps/1600749 (Accessed: 26, November 2022)

Stamp, M. and Low, R. M. (2007) *Applied Cryptanalysis.* San Jose, CA: A John Wiley & Sons, Inc., Publication.