Differential Cryptanalysis of FEAL-4

Implementation of the Algorithm in C++.

A summer project by

**Shubhankar Gaikwad**

Mentored By

**Dr.. Rishiraj Bhattacharya**

**School of Computer Sciences,**

**National Institute of Science Education and Research**

**Bhubaneswar , Odisha 752050**

**CERTIFICATE**

*This is to certify that the work contained in this report entitled “***Cryptanalysis of FEAL-4”** *is a bonafide work of* **Shubhankar Gaikwad***, carried out in the School of Computer Sciences, National Institute of Science Education and Research , Bhubaneshwar under my supervision.*

Dr. Rishiraj Bhattacharya

July 2019 READER F, School of Computer Sciences Bhubaneswar National Institute of Science Education and Research, Bhubaneswar

**Acknowledgment**

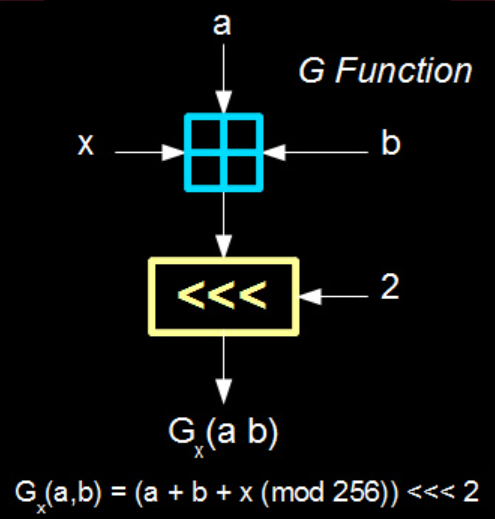
I would like to express our deepest gratitude to **Dr. Rishiraj Bhattacharya** , Professor, School of Computer Science, National Institute of Science Education and Research, Bhubaneswar, for his guidance and support throughout this project work. The summer Project opportunity,that I had with the National Institute of Science Education and Research, Bhubaneswar was a great chance for learning and professional development. Therefore, I consider myself very lucky individual that I was provided with an opportunity to be a part of it. I will strive to use gained skills and knowledge in the best possible way in my future endeavors.

Contents

Page

1. Introduction to FEAL 5
2. Structure of FEAL-4 8
3. Differential Cryptanalysis Basics 12
4. Differentials for cryptanalysis of FEAL-4 14
5. Algorithm for differential Cryptanalysis of FEAL-4 17
   1. Cryptanalysis of last rounds 17
   2. Cryptanalysis of first round 19
6. Implementation in C++ 20
7. Conclusion 30
8. Bibliography 31
9. Introduction to Feal

**F**EAL stands for Fast data Encipherment Algorithm, 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.The algorithm is susceptible to various forms of attacks.The structure of FEAL is very similar to the DES with a modified F-function, initial and final permutations and key scheduling algorithm.In the F-function the permutation P and the S boxes of DES are replaced by byte rotation and addition denoted by a function denoted by Gx(a,b), where a and b are the two inputs to the G-box and in this thesis.G box are possible in two variants denoted by x, which can take values 0 or 1. Gx(a,b)=Rotl2(a+b+x) mod 256, where Rotl2 denotes the cyclic rotation of the input bits by two positions.



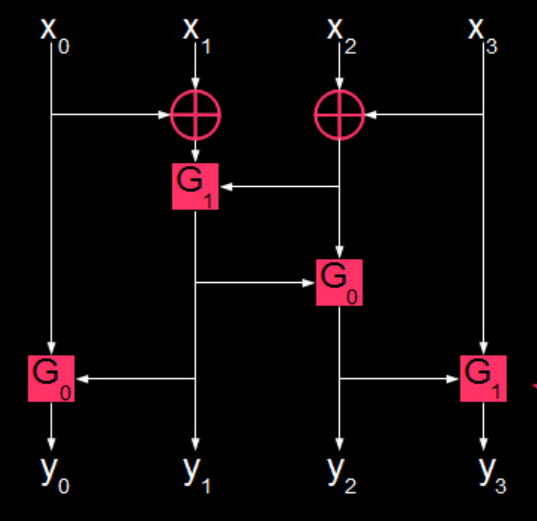
**Figure 1.1** G box. Gx(a,b)=Rotl2(a+b+x) mod 256

<<< refers to the left rotation of the input bits by 2 position.

Figure 1.1 gives details of G box. The F function gets 32-bits input and a 16-bit subkey and calculates a 32-bit output.The initial and final permutations are replaced by initial and final transformations, initial permutation in which the whole 64-bit input is XORed with 64-bit subkey and the left half XORed with the right and the final transformation in which the left half is XORed with the right half.The key scheduling algorithm is replaced by a key processing algorithm, which makes the subkeys depend on the key in more complex fashion.

However we will not be discussing the key processing algorithm in this work.

Figure 1.2 gives insights of the F function of the FEAL.



**Figure 1.2** The F box used in FEAL

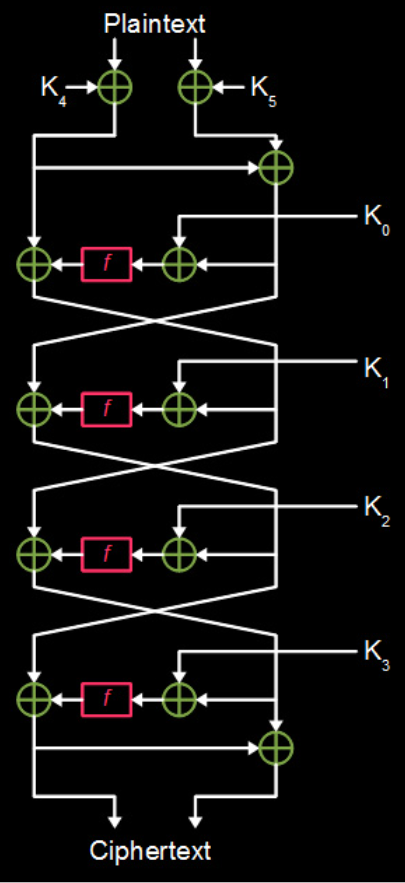
Originally, FEAL was suggested as a four-round cryptosystem, called FEAL-4. After the cryptanalysis of FEAL-4 the eight round variant FEAL-8 was introduced. Later FEAL-N with an arbitrary number of Rounds and FEAL-NX with increased size of 128-bit key were introduced.

2. Structure of FEAL-4

FEAL-4 is a Feistel network with a 64-bit block size This means that the algorithm to encrypt and decrypt data takes in 64-bit chunks. The Feistel structure means that the blocks are actually split in half for processing. These halves are mixed together using the XOR operations throughout the encryption process.

The non-linear component is called the F function.It is a one-way/trapdoor function that takes a 32 bit input and produces a 32 bit output. This function is used 4 times during encryption: once for each round. The strength of FEAL-4 against statistical attacks like differential cryptanalysis is dependent on the behavior of this F function.

The 64 bit key is expanded to six 32-bit subkeys. The somewhat separate process that achieves this is known as the key schedule. In the case of FEAL, this process is designed to be one-way.In other words, if we are able to recover one of the subkeys, it should be impossible to use that information to recover part of the "real" 64-bit key. Luckily, we only need all of the subkeys that are generated by the key schedule (or our own cryptanalysis efforts) to decrypt goodies. This means that recovering the original 64-bit key is not required. Ignoring the key scheduling process for now and pretending that the six 32-bit subkeys are all generated randomly/independently. In other words, we are behaving as if FEAL-4 has a 192-bit key.

The overall layout of FEAL is that of a Feistel cipher. It splits the data being encrypted into left and right halves. The right half is mixed with the round's 

**Figure 2.1** FEAL-4 internal structure, K0 ,K1, K2, K3, K4 and K5 each represent a 32-bit subkey. F-function is being represented by f.

subkey, then it is run through an F function. and combined with the left half using the XOR operation. The resulting halves are then swapped and the process is repeated for the other rounds (all four rounds ). Before any of these rounds happen, subkeys 4 and 5 are used to XOR the incoming plaintext.

Figure 2.1 gives insights of the structure of FEAL-4.

**Initial Preprocessing** **:**First, its left half is XOR'd with subkey K4 and its right half is XOR'd with subkey K5. Next, this new left half is XOR'd with this new right half to produce an even newer right half. The right half is ready to continue with the next operation. The preparation is done now and this state is the beginning of a round of FEAL.

**Each Round:**The right half splits off and becomes the final left half for this round.The original right half is XOR'd with this round's subkey to produce an input for the F-function. The F-function processes this input and spits out a number.The left half is XORed with the number spit out by the F-function. This XOR result becomes the final right half for this round. The next round of the cipher takes these left and right final halves and uses them for its input left and right halves, respectively.

**Final Round** **:**In the last round is a little funky because the output halves don't swap like they do for other rounds. Instead the final round's left output half becomes the final ciphertext's left half. The final ciphertext's right half is produced by XORing that left half with the final round original right half.Figure 2.1 gives insights of the FEAL-4 structure.

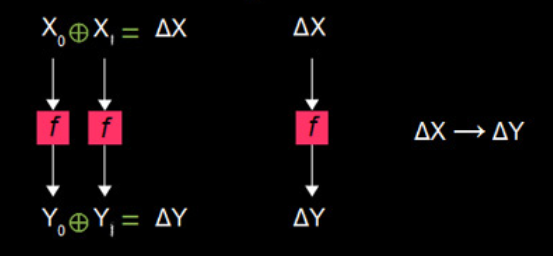
3.Differential cryptanalysis basics

Ciphers, and specifically their nonlinear components like F-functions and S-boxes, are meant to behave like pseudorandom number generators.A good nonlinear function would take an input X0, change some bits of X0 to produce X1, and the corresponding outputs (Y0 and Y1) should not have a predictable pattern of bit changes. This behavior should extend to how other input properties map to output properties.If there is a property that maps with a probability other than what a random function would (50% for example), it can be exploited to discover information about the key used in the cipher.

The ways in which these patterns map properties of the inputs to properties to the corresponding outputs is called a **characteristic**.The type of characteristics that are taken advantage of in this attack are called a "differential characteristic".Two inputs (X0 and X1) are taken to target the non-linear function that differ by certain bits. This difference is called a **differential**. To find the differential of two numbers, just XOR them. Next, both of these inputs are fed through the function. Then we take the differential of the resulting outputs (Y0 and Y1). This mapping of input differentials resulting in output differentials is called a differential characteristic.

The probability that any pair of inputs that form a particular input differential leads to a particular output differential should be the same as in a random function. Any deviation from this distribution can be leveraged against the security of the algorithm.

Figure 3.1 shows flow of differential through a F-function.



**Figure 3.1** Flow of differential characteristic through a F-function.

🛆X corresponds to input differential and 🛆Y corresponds to the output differential.

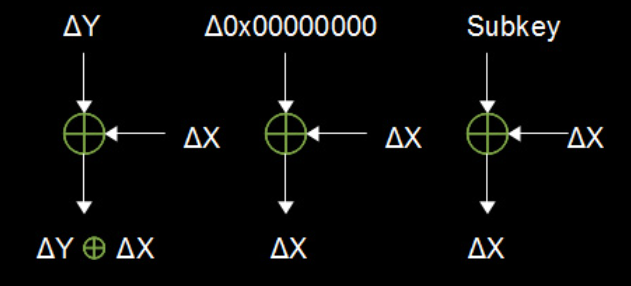
4.Differentials for cryptanalysis of FEAL-4

If a differential passes through the function and changes into a predictable output differential; tracing efforts can continue.One example of high probability differential characteristic that occurs through FEAL's F-function is ***0x80800000*** → ***0x02000000***. This characteristic actually occurs with a probability of 1 (100% of the time).

Another property of any nonlinear function is that input differentials of 0x00000000 always lead to output differentials of 0x00000000 i.e. ***0x00000000 → 0x00000000*** . Sending the same data through the function twice will lead to the same output both times. Which in turn will make XOR of the output 0, hence 0 differential.

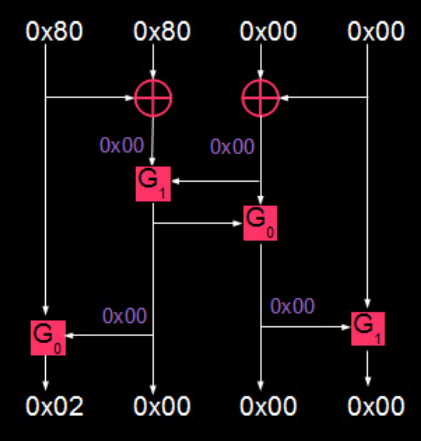
Key mixing is interesting because it has no effect on the differentials. It does affect the actual data not but the difference between them. The reason is that the key remains the same even though you are operating on pairs of data. This means that the key input of the mixing XOR operation has a differential of 0x00000000. XORing a differential 🛆X by 0 will always result in the original differential 🛆X.

Figure 4.1 shows the effect of key mixing on the differential cryptanalysis.



**Figure 4.1** Effect of key mixing on differential cryptanalysis

Figure 4.2 shows the actual flow of differential in a G-box.



**Figure 4.2** Flow of differential in G-box.

0x80800000 → 0x02000000

The conclusion that we can come up after analysis of FEAL-4 can be summarised using two results.

* **0x80800000 --> 0x02000000**
* **0x00000000 --> 0x00000000**

5.Algorithm for differential Cryptanalysis of FEAL-4

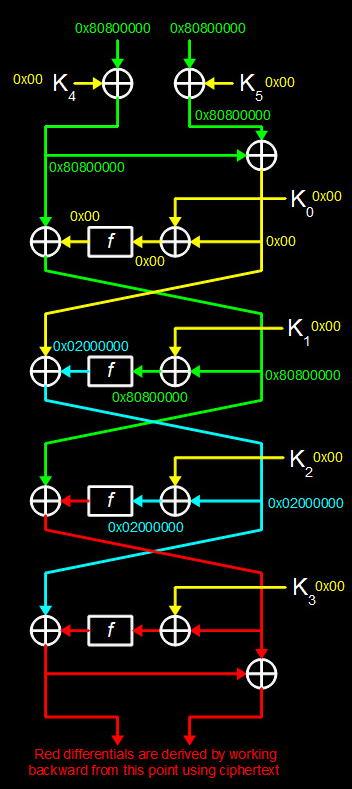
5.1 Cryptanalysis of last rounds

Perform an XOR on the left ciphertext against the right ciphertext. This gives the right input texts to the last round. Now for each possible subkey starting from 0x00000000 all the way through to 0xFFFFFFFF. On each iteration, XOR this candidate subkey with the round 4 right input that you just found. The result of this operation is the pair of input texts that went into the last round function during encryption.

Input differentials to be used for each round are as follows

* Round 4: 0x8080000080800000 → 0x02000000
* Round 3: 0x0000000080800000 → 0x02000000
* Round 2: 0x0000000002000000 → 0x02000000

Figure 5.1 shows complete flow of differentials through FEAL-4.



**Figure 5.1** Complete flow of differentials through FEAL-4

5.2 Cryptanalysis of first round

Differential cryptanalysis cannot be used to get its subkey. It also has two more subkeys that mix in right before it. Our goal is to recover all three of these subkeys in one go. To do this, we guess K0 and decrypt through round 1 like usual. Next, use that along with a chosen-plaintext pair to calculate K4 and K5. Repeat this process and keep track of which K0 guesses lead to identical ,K4’s and K5’s with every chosen-plaintext pair. The K0 guess that keeps the corresponding K4 and K5 consistent is the correct K0 (this also gives us K4 and K5). Now we have the full 192-bits of subkey.

6.Implementation in C++

# **ALL THE SYNTAX USED IN THE CODE BELOW IS AS PER STANDARD C++14**

*#include<bits/stdc++.h>*

*using namespace std;*

*#define MAX\_CHOSEN\_PAIRS 10000*

*//32-bit sub-keys, 6 in all required for encryption in FEAL-4*

*unsigned long subkey[6];*

*//cyclically rotates the bits by two positions to the left*

*unsigned char rotl2(unsigned char a) {return ((a << 2) | (a >> 6));}*

*//function that separates the left half of the plain text*

*unsigned long leftHalf(unsigned long long a) {return (a >> 32LL);}*

*//function that separates the right half of the plain text*

*unsigned long rightHalf(unsigned long long a) {return a;}*

*//function to separate the bytes of text*

*unsigned char sepByte(unsigned long a, unsigned char index) {return a >> (8 \* index);}*

*//function to combine bytes of the text*

*unsigned long combineBytes(unsigned char b3, unsigned char b2, unsigned char b1, unsigned char b0)*

*{*

*return b3 << 24L | (b2 << 16L) | (b1 << 8L) | b0;*

*}*

*//function to combine the left and the right halves of the*

*//cipher text*

*unsigned long long combineHalves(unsigned long leftHalf, unsigned long rightHalf)*

*{*

*return (((unsigned long long)(leftHalf)) << 32LL) | (((unsigned long long)(rightHalf)) & 0xFFFFFFFFLL);*

*}*

*//design of g-box*

*//Gx(a,b)=(a+b+x)%MOD <<<2*

*//<<< implies left cyclic rotation by two positions of the bits*

*//only two variants of g-box possible*

*//with x(mode here) taking value 0 & 1*

*unsigned char gBox(unsigned char a, unsigned char b, unsigned char mode)*

*{*

*return rotl2(a + b + mode);*

*}*

*//design of f-box*

*unsigned long fBox(unsigned long plain)*

*{*

*//separate the bytes of the plain text plainText=x3x2x1x0 into namely x0,x1,x2 and x3*

*unsigned char x0 = sepByte(plain, 0);*

*unsigned char x1 = sepByte(plain, 1);*

*unsigned char x2 = sepByte(plain, 2);*

*unsigned char x3 = sepByte(plain, 3);*

*//t0 is one of the inputs to the g-box corresponding to the x1 input*

*unsigned char t0 = (x2 ^ x3);*

*//t1 is one of the inputs to the g-box corresponding to the x1 input*

*unsigned char t1 = (x0 ^ x1);*

*//3 representing the MSB and 0 representing the LSB*

*//of the output of the f-box*

*unsigned char y1 = gBox(t1, t0, 1);*

*unsigned char y0 = gBox(x0, y1, 0);*

*unsigned char y2 = gBox(t0, y1, 0);*

*unsigned char y3 = gBox(x3, y2, 1);*

*//combine the bytes of the output text y0,y1,y2,y3 and return as 32-bit y3y2y1y0*

*return combineBytes(y3, y2, y1, y0);*

*}*

*//design implementation of FEAL-4*

*unsigned long long encrypt(unsigned long long plain)*

*{*

*//initial left and right halves of the plain Text*

*unsigned long left = leftHalf(plain);*

*unsigned long right = rightHalf(plain);*

*left = left ^ subkey[4];*

*right = right ^ subkey[5];*

*//Round 1*

*unsigned long round2Left = left ^ right;*

*unsigned long round2Right = left ^ fBox(round2Left ^ subkey[0]);*

*//Round 2*

*unsigned long round3Left = round2Right;*

*unsigned long round3Right = round2Left ^ fBox(round2Right ^ subkey[1]);*

*//Round 3*

*unsigned long round4Left = round3Right;*

*unsigned long round4Right = round3Left ^ fBox(round3Right ^ subkey[2]);*

*//Round 4*

*unsigned long cipherLeft = round4Left ^ fBox(round4Right ^ subkey[3]);*

*unsigned long cipherRight = cipherLeft ^ round4Right;*

*//combine both the halves*

*return combineHalves(cipherLeft, cipherRight);*

*}*

*//generate the 32-bit sub-keys*

*void generateSubkeys(int seed)*

*{*

*//seed for using seeding the rand() function*

*srand(seed);*

*//i corresponding to the i th indexed sub-key*

*for(int i = 0; i < 6; i++)*

*{*

*subkey[i] = (rand() << 16L) | (rand() & 0xFFFFL);*

*//bound the sub-key*

*subkey[i]=subkey[i]%(unsigned long)10000;*

*}*

*}*

*//number of plain text pairs*

*int numPlain;*

*unsigned long long plain0[MAX\_CHOSEN\_PAIRS];*

*unsigned long long cipher0[MAX\_CHOSEN\_PAIRS];*

*unsigned long long plain1[MAX\_CHOSEN\_PAIRS];*

*unsigned long long cipher1[MAX\_CHOSEN\_PAIRS];*

*//function to undo the last operation of XOR of the*

*//left and right half of the cipher text*

*void undoFinalOperation()*

*{*

*for(int i = 0; i < numPlain; i++)*

*{*

*//separating and undoing the last operation of the cipherText0 series*

*unsigned long cipherLeft0 = leftHalf(cipher0[i]);*

*unsigned long cipherRight0 = rightHalf(cipher0[i]) ^ cipherLeft0;*

*//separating and undoing the last operation of the cipherText1 series*

*unsigned long cipherLeft1 = leftHalf(cipher1[i]);*

*unsigned long cipherRight1 = rightHalf(cipher1[i]) ^ cipherLeft1;*

*//reconstructing the cipherText0 series*

*cipher0[i] = combineHalves(cipherLeft0, cipherRight0);*

*//reconstructing the cipherText1 series*

*cipher1[i] = combineHalves(cipherLeft1, cipherRight1);*

*}*

*}*

*//function to crack last round*

*unsigned long crackLastRound(unsigned long outdiff)*

*{*

*printf(" Using output differential of 0x%08x\n", outdiff);*

*printf(" Cracking...");*

*unsigned long fakeK;*

*for(fakeK = 0x00000000L; fakeK < 0xFFFFFFFFL; fakeK++)*

*{*

*//count when the calculated differential matches the actual differential*

*int score = 0;*

*for(int c = 0; c < numPlain; c++)*

*{*

*//cout<<"c "<<c<<" key checked "<<fakeK<<endl;*

*unsigned long cipherLeft = (cipher0[c] >> 32LL);*

*cipherLeft ^= (cipher1[c] >> 32LL);*

*unsigned long cipherRight = cipher0[c] & 0xFFFFFFFFLL;*

*cipherRight ^= (cipher1[c] & 0xFFFFFFFFLL);*

*unsigned long Y = cipherRight;*

*unsigned long Z = cipherLeft ^ outdiff;*

*unsigned long fakeRight = cipher0[c] & 0xFFFFFFFFLL;*

*unsigned long fakeLeft = cipher0[c] >> 32LL;*

*unsigned long fakeRight2 = cipher1[c] & 0xFFFFFFFFLL;*

*unsigned long fakeLeft2 = cipher1[c] >> 32LL;*

*unsigned long Y0 = fakeRight;*

*unsigned long Y1 = fakeRight2;*

*unsigned long fakeInput0 = Y0 ^ fakeK;*

*unsigned long fakeInput1 = Y1 ^ fakeK;*

*unsigned long fakeOut0 = fBox(fakeInput0);*

*unsigned long fakeOut1 = fBox(fakeInput1);*

*unsigned long fakeDiff = fakeOut0 ^ fakeOut1;*

*if (fakeDiff == Z) score++; else break;*

*}*

*if (score == numPlain)*

*{*

*printf("found subkey : 0x%08lx\n", fakeK);*

*return fakeK;*

*}*

*}*

*cout<<"failed"<<endl;*

*return 0;*

*}*

*//function to generate plain text pairs*

*//randomized generation to avoid false*

*void chosenPlaintext(unsigned long long diff)*

*{*

*cout<<"Generating "<<numPlain<<" chosen-plaintext pairs"<<endl;*

*printf("Using input differential of 0x%016llx\n", diff);*

*srand(time(NULL));*

*for(int c = 0; c < numPlain; c++)*

*{*

*plain0[c] = (rand() & 0xFFFFLL) << 48LL;*

*plain0[c] += (rand() & 0xFFFFLL) << 32LL;*

*plain0[c] += (rand() & 0xFFFFLL) << 16LL;*

*plain0[c] += (rand() & 0xFFFFLL);*

*//encrypting the plain text*

*cipher0[c] = encrypt(plain0[c]);*

*//generating the other buddy of plain text using the input differential*

*plain1[c] = plain0[c] ^ diff;*

*//encrypting the plain text*

*cipher1[c] = encrypt(plain1[c]);*

*}*

*}*

*void undoLastRound(unsigned long crackedSubkey)*

*{*

*for(int c = 0; c < numPlain; c++)*

*{*

*//getting the left and right halves of the cipher text*

*unsigned long cipherLeft0 = leftHalf(cipher0[c]);*

*unsigned long cipherRight0 = rightHalf(cipher0[c]);*

*unsigned long cipherLeft1 = leftHalf(cipher1[c]);*

*unsigned long cipherRight1 = rightHalf(cipher1[c]);*

*//updating the left halves*

*cipherLeft0 = cipherRight0;*

*cipherLeft1 = cipherRight1;*

*//updating the right halves*

*cipherRight0 = fBox(cipherLeft0 ^ crackedSubkey) ^ (cipher0[c] >> 32LL);*

*cipherRight1 = fBox(cipherLeft1 ^ crackedSubkey) ^ (cipher1[c] >> 32LL);*

*//combining both the halves*

*cipher0[c] = combineHalves(cipherLeft0, cipherRight0);*

*cipher1[c] = combineHalves(cipherLeft1, cipherRight1);*

*}*

*}*

*int main()*

*{*

*cout<<"--------------------FEAL-4 DIFFERENTIAL CRYPTANALYSIS DEMO--------------------"<<endl;*

*cout<<endl;*

*generateSubkeys(time(NULL));*

*numPlain = 10000;*

*//uncomment to see the generated sub-keys*

*//for(int i=0;i<6;i++)*

*//cout<<"0x"<<std::hex<<subkey[i]<<endl;*

*//first input differential to be used for cracking round 4*

*unsigned long long inputDifferential1 = 0x8080000080800000LL;*

*//second input differential to be used for cracking round 3*

*unsigned long long inputDifferential2 = 0x0000000080800000LL;*

*//third input differential to be used for cracking round 2*

*unsigned long long inputDifferential3 = 0x0000000002000000LL;*

*//common output differential*

*unsigned long outDiff = 0x02000000L;*

*//initial start time*

*unsigned long fullStartTime = time(NULL);*

*//Operations on Round 4*

*cout<<"ROUND 4"<<endl;*

*//generating the chosen plain texts for the given input differential*

*chosenPlaintext(inputDifferential1);*

*//undoing the last operation of XOR*

*undoFinalOperation();*

*//local start time*

*unsigned long startTime = time(NULL);*

*//the cracked sub-key for the round*

*unsigned long crackedSubkey3 = crackLastRound(outDiff);*

*//local end time*

*unsigned long endTime = time(NULL);*

*cout<<"Time to crack round #4 ="<<endTime - startTime<<" seconds"<<endl;*

*cout<<endl<<endl;*

*//CRACKING ROUND 3*

*printf("ROUND 3\n");*

*//generating the chosen plain texts for the given input differential*

*chosenPlaintext(inputDifferential2);*

*//undoing the last operation of XOR*

*undoFinalOperation();*

*//undoing the last round using the sub-key cracked previously*

*undoLastRound(crackedSubkey3);*

*//local start time*

*startTime = time(NULL);*

*//sub-key cracked in this round*

*unsigned long crackedSubkey2 = crackLastRound(outDiff);*

*//local end time*

*endTime = time(NULL);*

*cout<<"Time to crack round #3 ="<<endTime - startTime<<" seconds"<<endl;*

*cout<<endl<<endl;*

*//CRACKING ROUND 2*

*printf("ROUND 2\n");*

*//generating the chosen plain texts for the given input differential*

*chosenPlaintext(inputDifferential3);*

*//undoing the last operation of XOR*

*undoFinalOperation();*

*//undoing the last round using the sub-key cracked previously*

*undoLastRound(crackedSubkey3);*

*//undoing the last round using the sub-key cracked previously*

*undoLastRound(crackedSubkey2);*

*//local start time*

*startTime = time(NULL);*

*//sub-key cracked in this round*

*unsigned long crackedSubkey1 = crackLastRound(outDiff);*

*//local end time*

*endTime = time(NULL);*

*cout<<"Time to crack round #2 ="<<endTime - startTime<<" seconds"<<endl;*

*cout<<endl<<endl;*

*//CRACK ROUND 1*

*printf("ROUND 1\n");*

*//since the last round cannot be cracked using differential*

*//plain text is not generated for this round the previous generated*

*//plain text is manipulated*

*//undo the previous rounds using the sub-key cracked previously*

*undoLastRound(crackedSubkey1);*

*unsigned long crackedSubkey0 = 0;*

*unsigned long crackedSubkey4 = 0;*

*unsigned long crackedSubkey5 = 0;*

*printf(" Cracking...");*

*//local start time*

*startTime = time(NULL);*

*//guessing the local sub-key K0*

*unsigned long guessK0;*

*for(guessK0 = 0; guessK0 < 0xFFFFFFFFL; guessK0++)*

*{*

*//guessing the local sub-key K4*

*unsigned long guessK4 = 0;*

*//guessing the local sub-key K5*

*unsigned long guessK5 = 0;*

*for(int c = 0; c < numPlain; c++)*

*{*

*unsigned long plainLeft0 = leftHalf(plain0[c]);*

*unsigned long plainRight0 = rightHalf(plain0[c]);*

*unsigned long cipherLeft0 = leftHalf(cipher0[c]);*

*unsigned long cipherRight0 = rightHalf(cipher0[c]);*

*unsigned long tempy0 = fBox(cipherRight0 ^ guessK0) ^ cipherLeft0;*

*if (guessK4 == 0)*

*{*

*guessK4 = tempy0 ^ plainLeft0;*

*guessK5 = tempy0 ^ cipherRight0 ^ plainRight0;*

*}*

*else if (((tempy0 ^ plainLeft0) != guessK4) || ((tempy0 ^ cipherRight0 ^ plainRight0) != guessK5))*

*{*

*guessK4 = 0;*

*guessK5 = 0;*

*break;*

*}*

*}*

*if (guessK4 != 0)*

*{*

*crackedSubkey0 = guessK0;*

*crackedSubkey4 = guessK4;*

*crackedSubkey5 = guessK5;*

*endTime = time(NULL);*

*printf("found subkeys : 0x%08lx 0x%08lx 0x%08lx\n", guessK0, guessK4, guessK5);*

*cout<<" Time to crack round #1 ="<<(endTime - startTime)<<" seconds"<<endl;*

*break;*

*}*

*}*

*cout<<endl<<endl;*

*printf("0x%08lx - ", crackedSubkey0); if (crackedSubkey0 == subkey[0]) printf("Subkey 0 : GOOD!\n"); else printf("Subkey 0 : BAD\n");*

*printf("0x%08lx - ", crackedSubkey1); if (crackedSubkey1 == subkey[1]) printf("Subkey 1 : GOOD!\n"); else printf("Subkey 1 : BAD\n");*

*printf("0x%08lx - ", crackedSubkey2); if (crackedSubkey2 == subkey[2]) printf("Subkey 2 : GOOD!\n"); else printf("Subkey 2 : BAD\n");*

*printf("0x%08lx - ", crackedSubkey3); if (crackedSubkey3 == subkey[3]) printf("Subkey 3 : GOOD!\n"); else printf("Subkey 3 : BAD\n");*

*printf("0x%08lx - ", crackedSubkey4); if (crackedSubkey4 == subkey[4]) printf("Subkey 4 : GOOD!\n"); else printf("Subkey 4 : BAD\n");*

*printf("0x%08lx - ", crackedSubkey5); if (crackedSubkey5 == subkey[5]) printf("Subkey 5 : GOOD!\n"); else printf("Subkey 5 : BAD\n");*

*cout<<endl;*

*unsigned long fullEndTime = time(NULL);*

*cout<<"Total crack time = "<<fullEndTime - fullStartTime<<" seconds"<<endl;*

*cout<<"FINISHED"<<endl;*

*//to stop at the display screen*

*while(1){}*

*return 0;*

*}*

7.Conclusion

Exhaustive search of the keyspace from the perspective of subkeys means brute forcing every permutation of 192 bits. This comes out to testing approximately the following number of keys: ***6,300,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000***

However by finding the subkeys using the differential cryptanalysis, we end up testing only about ***171,798,691,840*** keys.

7. Bibliography

* Biham, Eli, and Adi Shamir. “Differential Cryptanalysis of the Data Encryption Standard.” 1993, doi:10.1007/978-1-4613-9314-6.
* “Differential Cryptanalysis of FEAL.” *The Amazing King - Differential Cryptanalysis of FEAL*, www.theamazingking.com/crypto-feal.php.
* “FEAL.” *Wikipedia*, Wikimedia Foundation, 28 July 2016, en.wikipedia.org/wiki/FEAL.
* Heys, Howard M. “A Tutorial On Linear And Differential Cryptanalysis.” *Cryptologia*, vol. 26, no. 3, 2002, pp. 189–221., doi:10.1080/0161-110291890885.