Linear Cryptanalysis of the FEAL-4 Cipher

Name: SNEHA DECHAMMA MALLENGADA SURESH
Student Number: 23262168

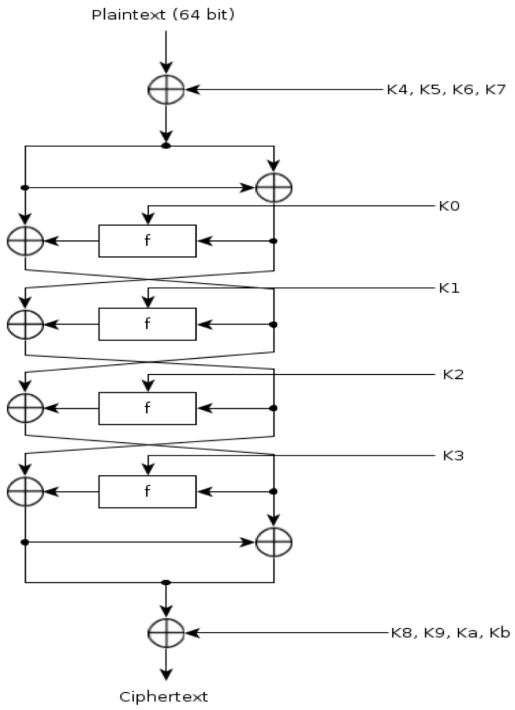
What is Cryptanalysis?
The mathematical study of cryptographic security systems is known as cryptanalysis, and its goal is to identify vulnerabilities that can be used to obtain plaintext information from ciphertext without the need for a key.
The majority of cryptanalytic methods emerge from two primary categories: linear and differential cryptanalysis. Differential-linear, non-linear, CPA-linear, partial/truncated differential, and higher order differential cryptanalytic techniques are a few common expansions of these two.

Linear Cryptanalysis
One type of known plaintext attack (KPA) is linear cryptanalysis, which makes use of linear relationships that hold true with a specific probability between input plaintexts and output ciphertexts. These relations can be utilized to obtain some information about the key(s) used in the cryptographic system, in conjunction with a set of plaintexts and matching ciphertexts. Block cipher encryption techniques are typically linear, with the exception of the S-box-related portions. Substitution boxes, also known as S-boxes, are PRFs that are rendered non-linear by hiding the connection between the plaintext and the ciphertext. Finding approximately equal correlations between the input and output bits of the S-boxes that the cryptosystem uses is the main goal of linear cryptanalysis. In the case of an S-box mapping n bit inputs X to m bit outputs Y, the number of possible linear relations is $(2n - 1)(2m - 1)$, with certain options being more likely than others. They take the following form:
$Xi1 \oplus Xi2 \oplus \cdots \oplus Xia \oplus \cdots \oplus Yj1 \oplus Yj2 \oplus \cdots \oplus Yjb = 0$ where $i1,i2,...,ia \in [n]$ are bits of X, and likewise $j1,j2,...,jb \in [m]$ for Y.

These linear relationships across different S-boxes are combined to create a single most/least likely linear approximation that connects the plaintext, the ciphertext, and the key.

Structure of FEAL-4
The FEAL-4 (Fast Data Encipherment Algorithm) is a symmetric key block cipher that operates on 64-bit blocks of data. It consists of four rounds, each involving several key-dependent operations, including bitwise XOR, modular addition, and non-linear functions. The FEAL-4 algorithm utilizes a 128-bit key, which is divided into six subkeys.

The key schedule generates these subkeys, which are then used in each round to modify the data. The rounds are designed to provide confusion and diffusion to enhance the security of the cipher.



It is known that the scheme consists of four rounds, six subkeys with a total size of 32 bits each, and 64 bits of input plaintext and 64 bits of output ciphertext. The graphic on illustrates the internal structure of the smaller f-boxes, or round functions. The definition of the functions Gi is:
Gi(a, b) = ((a + b + i) % 256) << 2
for i = 0 and i = 1.

Implementation
We have implemented the attack described , on the FEAL-4 cipher, in C++.
Compile and execute the C++ file to execute the code, either with or without a command line argument specifying the quantity of plaintexts to be utilized in each round of the attack. Following the generation of random 32-bit subkeys, the FEAL-4 cipher is compromised using the linear cryptanalytic attack described previously.

Key Steps in Linear Cryptanalysis:

1. Random Key Generation: The process begins by generating a random 128-bit key.
2. Plaintext-Ciphertext Pair Generation: A specified number of plaintext-ciphertext pairs are generated using the FEAL-4 encryption algorithm. These pairs are obtained by encrypting random plaintexts and creating corresponding ciphertexts.
3. Differential Cryptanalysis: Differential cryptanalysis involves examining the differences (differentials) between pairs of plaintexts and their corresponding ciphertexts. In this implementation, a differential is introduced by XORing a constant value with one set of plaintexts.
4. Round Decryption: The last round of the FEAL-4 algorithm is decrypted to obtain the intermediate values.

5. Key Recovery: Linear equations are set up to represent the relationships between the differentials and the key bits. A systematic search is conducted to find a key that satisfies the linear equations, thereby recovering the subkey for that particular round.
6. Iterative Process:The process is repeated for each round of the FEAL-4 algorithm, starting from the last round and moving towards the first.

Code

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <iomanip>

#define MAX_CHOSEN_PAIRS 10000

typedef unsigned long long ull;
typedef unsigned uint;
typedef unsigned char byt;

int num_plaintexts;
uint key[6];

ull plaintext0[MAX_CHOSEN_PAIRS];
ull ciphertext0[MAX_CHOSEN_PAIRS];
ull plaintext1[MAX_CHOSEN_PAIRS];
ull ciphertext1[MAX_CHOSEN_PAIRS];

inline uint getLeftHalf(ull x) {
    return static_cast<uint>(x >> 32);
}

inline uint getRightHalf(ull x) {
    return static_cast<uint>(x & 0xFFFFFFFFULL);
}

inline ull getCombinedHalves(uint a, uint b) {
    return (static_cast<ull>(a) << 32) | (static_cast<ull>(b) & 0xFFFFFFFFULL);
}

void createRandomKeys() {
    std::srand(static_cast<unsigned>(std::time(nullptr)));

    for (int i = 0; i < 6; ++i)
        key[i] = (std::rand() << 16) | (std::rand() & 0xFFFFU);
}

byt g(byt a, byt b, byt x) {
    byt tmp = a + b + x;
    return (tmp << 2) | (tmp >> 6);
}
uint f(uint input) {
```

```
    byt x[4], y[4];
    for (int i = 0; i < 4; ++i) {
        x[3 - i] = static_cast<byt>(input & 0xFF);
        input >>= 8;
    }

    y[1] = g(x[0] ^ x[1], x[2] ^ x[3], 1);
    y[0] = g(x[0], y[1], 0);
    y[2] = g(x[2] ^ x[3], y[1], 0);
    y[3] = g(x[3], y[2], 1);

    uint output = 0;
    for (int i = 0; i < 4; ++i)
        output += (static_cast<uint>(y[i]) << (8 * (3 - i)));

    return output;
}

ull encrypt(ull plaintext) {
    uint initialLeft = getLeftHalf(plaintext) ^ key[4];
    uint initialRight = getRightHalf(plaintext) ^ key[5];

    uint round1Left = initialLeft ^ initialRight;
    uint round1Right = initialLeft ^ f(round1Left ^ key[0]);

    uint round2Left = round1Right;
    uint round2Right = round1Left ^ f(round1Right ^ key[1]);

    uint round3Left = round2Right;
    uint round3Right = round2Left ^ f(round2Right ^ key[2]);

    uint round4Left = round3Left ^ f(round3Right ^ key[3]);
    uint round4Right = round4Left ^ round3Right;

    return getCombinedHalves(round4Left, round4Right);
}

void generatePlaintextCiphertextPairs(ull inputDiff) {
    std::cout << "Generating " << num_plaintexts << " plaintext-ciphertext
pairs\n";
    std::cout << "Using input Linear 0x" << std::hex << inputDiff << std::dec <<
"\n";

    std::srand(static_cast<unsigned>(std::time(nullptr)));

    for (int i = 0; i < num_plaintexts; ++i) {
        plaintext0[i] = (static_cast<ull>(std::rand() & 0xFFFFULL) << 48) |
                        (static_cast<ull>(std::rand() & 0xFFFFULL) << 32) |
                        (static_cast<ull>(std::rand() & 0xFFFFULL) << 16) |
                        (std::rand() & 0xFFFFULL);
```

```
        ciphertext0[i] = encrypt(plaintext0[i]);
        plaintext1[i] = plaintext0[i] ^ inputDiff;
        ciphertext1[i] = encrypt(plaintext1[i]);
    }
}

void decryptLastOperation() {
    for (int i = 0; i < num_plaintexts; ++i) {
        uint cipherLeft0 = getLeftHalf(ciphertext0[i]);
        uint cipherRight0 = getRightHalf(ciphertext0[i]) ^ cipherLeft0;
        uint cipherLeft1 = getLeftHalf(ciphertext1[i]);
        uint cipherRight1 = getRightHalf(ciphertext1[i]) ^ cipherLeft1;

        ciphertext0[i] = getCombinedHalves(cipherLeft0, cipherRight0);
        ciphertext1[i] = getCombinedHalves(cipherLeft1, cipherRight1);
    }
}

uint crackHighestRound(uint differential) {
    std::cout << "  Using output Linear of 0x" << std::hex << differential <<
std::dec << "\n";
    std::cout << "  Processing...\n";

    for (uint tmpKey = 0x00000000U; tmpKey <= 0xFFFFFFFFU; ++tmpKey) {
        int score = 0;

        for (int i = 0; i < num_plaintexts; ++i) {
            uint cipherRight0 = getRightHalf(ciphertext0[i]);
            uint cipherLeft0 = getLeftHalf(ciphertext0[i]);
            uint cipherRight1 = getRightHalf(ciphertext1[i]);
            uint cipherLeft1 = getLeftHalf(ciphertext1[i]);

            uint cipherLeft = cipherLeft0 ^ cipherLeft1;
            uint fOutDiffActual = cipherLeft ^ differential;

            uint fInput0 = cipherRight0 ^ tmpKey;
            uint fInput1 = cipherRight1 ^ tmpKey;
            uint fOut0 = f(fInput0);
            uint fOut1 = f(fInput1);
            uint fOutDiffComputed = fOut0 ^ fOut1;

            if (fOutDiffActual == fOutDiffComputed)
                ++score;
            else
                break;
        }

        if (score == num_plaintexts) {
            std::cout << "found key : 0x" << std::hex << tmpKey << std::dec <<
"\n";
            std::cout << std::flush;
```

```
            return tmpKey;
        }
    }

    std::cout << "failed\n";
    return 0;
}

void decryptHighestRound(uint crackedKey) {
    for (int i = 0; i < num_plaintexts; ++i) {
        uint cipherLeft0 = getRightHalf(ciphertext0[i]);
        uint cipherLeft1 = getRightHalf(ciphertext1[i]);

        uint cipherRight0 = f(cipherLeft0 ^ crackedKey) ^
getLeftHalf(ciphertext0[i]);
        uint cipherRight1 = f(cipherLeft1 ^ crackedKey) ^
getLeftHalf(ciphertext1[i]);

        ciphertext0[i] = getCombinedHalves(cipherLeft0, cipherRight0);
        ciphertext1[i] = getCombinedHalves(cipherLeft1, cipherRight1);
    }
}

int main(int argc, char **argv) {
    std::cout << "Linear Cryptanalysis of FEAL-4\n\n\n";

    if (argc == 1)
        num_plaintexts = 12;
    else if (argc == 2)
        num_plaintexts = std::atoi(argv[1]);
    else {
        std::cout << "Usage: " << argv[0] << " [Number of chosen plaintexts]\n";
        return 0;
    }

    createRandomKeys();
    uint startTime = static_cast<uint>(std::time(nullptr));

    // Round 4
    std::cout << "Round 4: To find K3\n\n";
    generatePlaintextCiphertextPairs(0x8080000080800000ULL);
    decryptLastOperation();

    uint roundStartTime = static_cast<uint>(std::time(nullptr));
    uint crackedKey3 = crackHighestRound(0x02000000U);
    uint roundEndTime = static_cast<uint>(std::time(nullptr));
    std::cout << "  Time to crack round #4 = " << int(roundEndTime -
roundStartTime) << " seconds\n\n";

    // Round 3
    std::cout << "Round 3: To find K2\n";
```

```
    generatePlaintextCiphertextPairs(0x0000000080800000ULL);
    decryptLastOperation();
    decryptHighestRound(crackedKey3);

    roundStartTime = static_cast<uint>(std::time(nullptr));
    uint crackedKey2 = crackHighestRound(0x02000000U);
    roundEndTime = static_cast<uint>(std::time(nullptr));
    std::cout << "  Time to crack round #3 = " << int(roundEndTime -
roundStartTime) << " seconds\n\n";

    // Round 2
    std::cout << "Round 2: To find K1\n";
    generatePlaintextCiphertextPairs(0x207f5bc585764709);
    decryptLastOperation();
    decryptHighestRound(crackedKey3);
    decryptHighestRound(crackedKey2);

    roundStartTime = static_cast<uint>(std::time(nullptr));
    uint crackedKey1 = crackHighestRound(0x02000000U);
    roundEndTime = static_cast<uint>(std::time(nullptr));
    std::cout << "  Time to crack round #2 = " << int(roundEndTime -
roundStartTime) << " seconds\n\n";

    // Round 1
    std::cout << "Round 1: To find K0\n";
    decryptHighestRound(crackedKey1);
    std::cout << "Processing... \n";

    roundStartTime = static_cast<uint>(std::time(nullptr));

    uint crackedKey0 = 0;
    uint crackedKey4 = 0;
    uint crackedKey5 = 0;

    for (uint tmpK0 = 0; tmpK0 < 0xFFFFFFFFL; ++tmpK0) {
        uint tmpK4 = 0;
        uint tmpK5 = 0;

        for (int i = 0; i < num_plaintexts; ++i) {
            uint plainLeft0 = getLeftHalf(plaintext0[i]);
            uint plainRight0 = getRightHalf(plaintext0[i]);
            uint cipherLeft0 = getLeftHalf(ciphertext0[i]);
            uint cipherRight0 = getRightHalf(ciphertext0[i]);

            uint temp = f(cipherRight0 ^ tmpK0) ^ cipherLeft0;
            if (tmpK4 == 0) {
                tmpK4 = temp ^ plainLeft0;
                tmpK5 = temp ^ cipherRight0 ^ plainRight0;
            } else if (((temp ^ plainLeft0) != tmpK4) || ((temp ^ cipherRight0 ^
plainRight0) != tmpK5)) {
                tmpK4 = 0;
```

```
                tmpK5 = 0;
                break;
            }
        }

        if (tmpK4 != 0) {
            crackedKey0 = tmpK0;
            crackedKey4 = tmpK4;
            crackedKey5 = tmpK5;
            break;
        }
    }

    std::cout << "found key K0: 0x" << std::hex << crackedKey0 << std::dec << "\n";
    std::cout << "found key K4: 0x" << std::hex << crackedKey4 << std::dec << "\n";
    std::cout << "found key K5: 0x" << std::hex << crackedKey5 << std::dec << "\n";
    uint endTime = static_cast<uint>(std::time(nullptr));
    std::cout << "Total time taken = " << int(endTime - startTime) << " seconds\n";

    std::cout << "\n\n\n";

    generatePlaintextCiphertextPairs(0xaea129c37cc07d12);

    key[0] = crackedKey0;
    key[1] = crackedKey1;
    key[2] = crackedKey2;
    key[3] = crackedKey3;
    key[4] = crackedKey4;
    key[5] = crackedKey5;

    for (int i = 0; i < num_plaintexts; ++i) {
        ull a = encrypt(plaintext0[i]);
        ull b = encrypt(plaintext1[i]);
        if (a != ciphertext0[i] || b != ciphertext1[i]) {
            std::cout << "Failed " << std::hex << a << " " << b << " " <<
ciphertext0[i] << " " << ciphertext1[i]
                      << std::dec;
            return 0;
        }
    }

    std::cout << "Each ciphertext generated using the keys obtained above matches
the ciphertext generated by the actual encryption algorithm!\n";
    std::cout << "Finished successfully.\n";
    return 0;
}
```

Output

```
Round 4: To find K3

Generating 12 plaintext-ciphertext pairs
Using input Linear 0x8080000080800000
  Using output Linear of 0x2000000
  Processing...
found key : 0x8891871
  Time to crack round #4 = 10 seconds

Round 3: To find K2
Generating 12 plaintext-ciphertext pairs
Using input Linear 0x80800000
  Using output Linear of 0x2000000
  Processing...
found key : 0x2ecc372b
  Time to crack round #3 = 57 seconds

Round 2: To find K1
Generating 12 plaintext-ciphertext pairs
Using input Linear 0x2000000
  Using output Linear of 0x2000000
  Processing...
found key : 0x24024249
  Time to crack round #2 = 43 seconds

Round 1: To find K0
Processing...
```

```
Plaintext=  a7f1d92a82c8d8fe - - K1, K2, K3, K4
Ciphertext= 150e2afb5daa0829

Plaintext=  434d98558ce2b347 - - K2, K3, K1, K4
Ciphertext= 42f0b42b04e989d1

Plaintext=  171198542f112d05 - - K1, K2, K4, K3
Ciphertext= 0a0311feb6564f4a

Plaintext=  58f56bd688079992 - - K4, K1,K2, K3
Ciphertext= 3b27f3cca5174c46

Plaintext=  48336241f30d23e5 - - K0, K2, K3, K4
Ciphertext= 2a2a66634ed5d691

Plaintext=  5f30d1c8ed610c4b - - K2, K4, K1, K3
Ciphertext= 0a3a0048885f4b17

Plaintext=  0235398184b814a2 - - K3, K1, K2, K4
Ciphertext= ad33592df0f21dcf

Plaintext=  9cb45a672acae548 - - K3, K1,K4, K0
Ciphertext= dcf32b9dd19e09a4

Plaintext=  e9c5f1b0c4158ae5 - - K4, K0, K1, K2
Ciphertext= cac0ce52a0950361

Plaintext=  9b4d39f6f7e8a105 - - K1, K4, K2, K0
Ciphertext= a8344725f5d01d18

Plaintext=  d3feeda5d5f3d9e4 - - K2, K1, K3, K4
Ciphertext= 6c89973c13cc8729

Plaintext=  5bfa6cc351e220ae
Ciphertext= bded36e0c6d22846

Plaintext=  0ce106986d61ff34
Ciphertext= bd2c874760ccf8ae
```