Advanced Topics in Cybersecurity (650.050) (22S)

Project: Linear Cryptoanalysis on DES

July 5, 2022

Teacher: Arnab Roy *Student: Luca Campa*

1 The Project: Introduction

By following the paper written by Matsui[1], the project describes my own implementation of the attack against 8-round DES. As done by Matsui in his paper, I considered all the 8 S-Boxes and I omitted the initial permutation and the final permutation (they do not make changes in the attack structure, the performances would remain the same). The notation used within this paper is the one indicated by Matsui:

- The right-most bit is indicated by the index 0
- P: The 64 bit plaintext
- C: the corresponding 64-bit ciphertext
- P_H: the left 32-bit of P
- P_L: the right 32-bit of P
- C_H: the left 32-bit of C
- C_L: the right 32-bit of C
- X_i: the 32-bit intermediate value in the i-th round
- K_i: the 48-bit key in the i-th round
- $F_i(X_i, K_i)$: the i-th round function F
- $A[i]$: the bit of A indicated by index i
- $A[i, j, .., k]$: $A[i] \oplus A[j] \oplus ... \oplus A[k]$

SBOX analysis The first step of my project was dedicated to the analysis of the 8 S-Boxes used by DES. In particular, I studied, for each S-Box, the probability that the xored value of input bits masked by α coincides with the xored value of the output bits masked by β. In practice, I follow Matsui **Definition 1**. The distribution tables of the S-Boxes can be generated with the **sbox_distribution_table.c** code (the input parameter is the index of the table).
These results were useful to deeply understand the discovery and the reasoning behind the linear approximations given by Matsui. Moreover, before to deal with the 8-round DES, I tried the attack against 5-round DES by using the approximation supplied by Matsui and an other one found by me.

[1]Linear Cryptoanalysis Method for DES Cipher, 1994, Mitsuru Matsui

2 Linear Approximations

In order to discover some bits of K_1 and K_8 I used the linear approximations for 7 and 6 rounds adapted to deal with 8 rounds. This two approximations are:

$$P_H[7, 18, 24] \oplus P_L[12, 16] \oplus C_H[7, 18, 24, 29] \oplus C_L[15] = K_1[19, 23] \oplus L_3 \oplus K_7[22] \quad (1)$$

$$P_L[7, 18, 24] \oplus C_H[7, 18, 24, 29] \oplus C_L[15] = L_2 \oplus K_6[22] \quad (2)$$

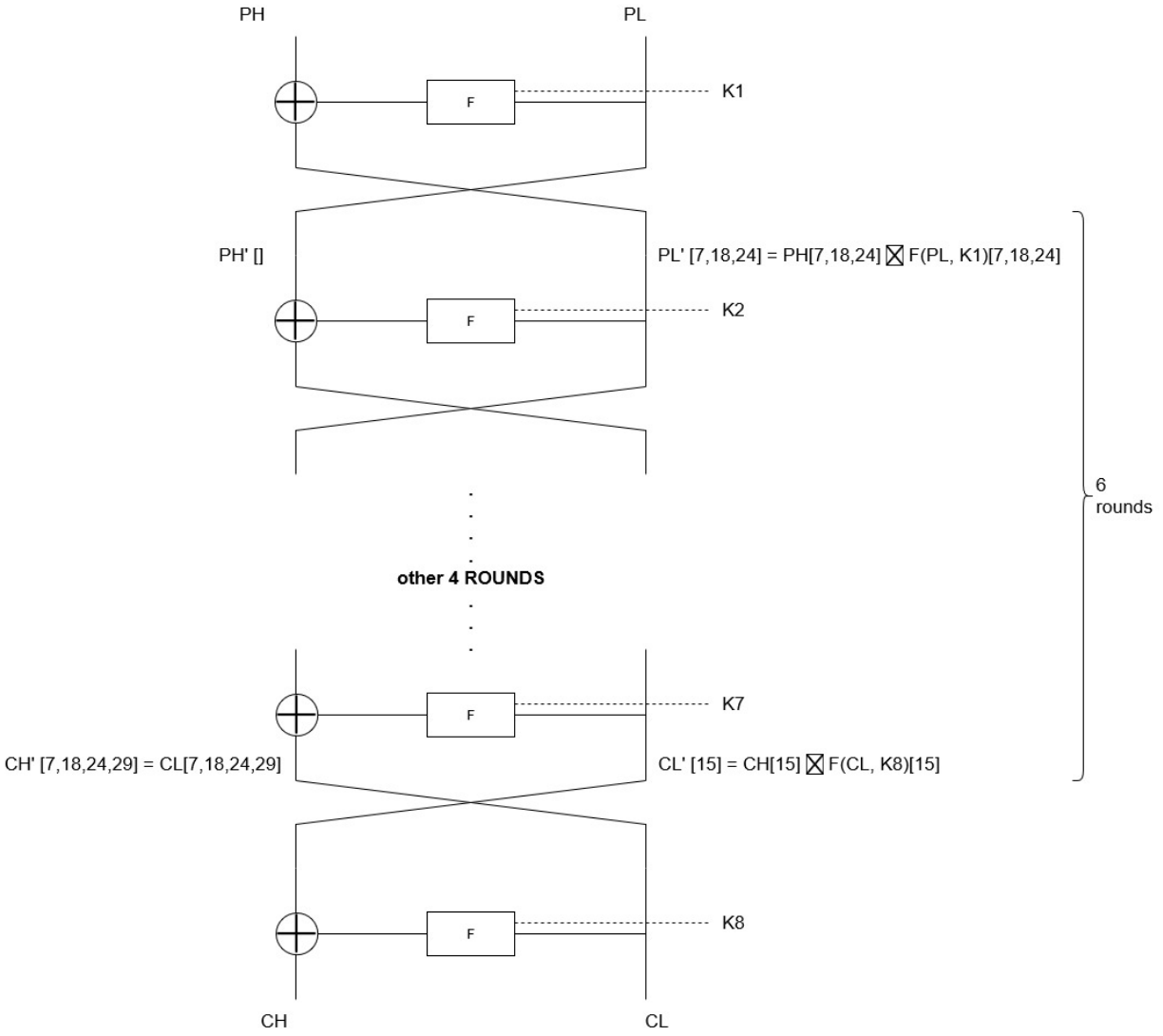


Figure 1: 6 round approximation to 8 rounds conversion

In order to adapt those approximations to 8 rounds we need to consider the fact that the DES round swaps the two 32-bit parts of the output. The transformation (an example is shown in Figure 1)

raises to the following two approximations that can be used to guess many bits of the two target sub-keys:

$$P_H[7,18,24] \oplus P_L[12,16] \oplus C_L[7,18,24,29] \oplus F_8(C_L,K_8)[15] \oplus C_H[15] \\ = K_1[19,23] \oplus K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22] \quad (3)$$

$$P_H[7,18,24] \oplus F(P_L,K_1)[7,18,24] \oplus C_L[7,18,24,29] \oplus C_H[15] \oplus F(C_L,K_8)[15] \\ = L_3 \oplus K_7[22] \quad (4)$$

You can notice that:

- For 7 rounds:
  - $C_L[15] \rightarrow F_8(C_L,K_8)[15] \oplus C_H[15]$
  - $C_H[7,18,24,29] \rightarrow C_L[7,18,24,29]$
- For 6 rounds
  - $P_L[7,18,24] \rightarrow P_H[7,18,24] \oplus F(P_L,K_1)[7,18,24]$
  - $C_H[7,18,24,29] \rightarrow C_L[7,18,24,29]$
  - $C_L[15] \rightarrow C_H[15] \oplus F(C_L,K_8)[15]$

An other interesting property is that DES has got a lot of symmetries that we can exploit in order to discover more bits by using the two approximations above with very few changes. Let us see what are the symmetries:

- The decryption of DES is the encryption of the ciphertext with the subkeys applied in the reverse order. Then we can consider the ciphertexts as plaintexts and viceversa.
- The chosen approximations are symmetric, than from the two above we can retrive other 2 approximations by simply swapping C_L, C_H with P_L, P_H and substituting K_i with K_{N-i+1} ($N = 8$ rounds). In order to perform the attack we can consider only the symmetric version of equation 3. The other one could be useful if the exhaustive search is too complex yet.

$$P_L[7,18,24,29] \oplus F_8(P_L,K_1)[15] \oplus P_H[15] \oplus C_H[7,18,24] \oplus C_L[12,16] \\ = K_8[19,23] \oplus K_6[22] \oplus K_5[44] \oplus K_4[22] \oplus K_2[22] \quad (5)$$

3 The Attack

The following python style code shows how to use Algorithm 2 (by Matsui) together with equation 3 in order to find the key bits of K_8 from index 42 to index 47 (6 bits - 64 possibilities). How did I decide to search for these key bits? The procedure to discover what are the bits we are interested in is:

- In equation 3, we are interested in the bit index 15 of $F_8(C_L,K_8)$

- We need to reverse the round function in order to discover what are the bits that activate bit index 15.
- The inverse permutation tells us that the bit 15 is activated by bit 30.
- Bit 30 is activated by the 6 bits of S-Box 1
- The indexes of S-Box 1 are 42,43,44,45,46,47. Then, a wrong choice for these 6 bits in K_8 would not satisfy equation 3.

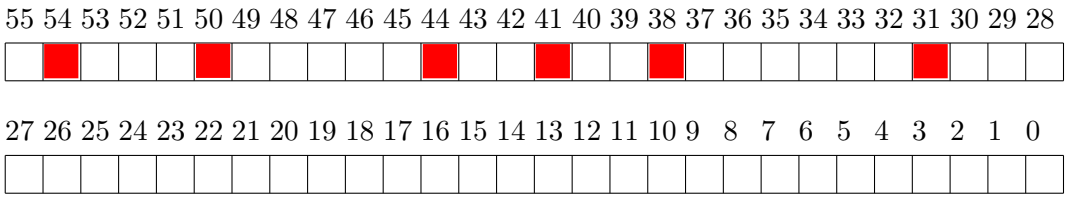
```
def attack(plaintexts, ciphertexts):
    counter_keys[64]
    for possible_bits in range(0,64):
        for i in range(0, len(plaintexts)):
            P = plaintexts[i]
            C = ciphertexts[i]
            PH = P[0:32]
            PL = P[32:]
            CH = C[0:32]
            CL = C[32:]

            K8 = (possible_bits << 42)
            # from now on I will use the notation by Matsui in order to deal with
            ↪  xor values in the equation: X[1,2,3] means X[1] ^ X[2] ^ X[3]

            if ( (PH[7,18,24] ^ PL[12,16] ^ CL[7,18,24,29] ^ CH[15] ^ F(CL, K8)[15])
            ↪  == 0 ):
                counter_keys[i]++

    Tmax = max(counter_keys)
    Tmin = min(counter_keys)

    samples = len(plaintexts)

    if ( |Tmax - samples/2| > |Tmin - samples/2| ):
        return indexOf(Tmax) # returns the key guess which has Tmax as counter
    else:
        return indexOf(Tmin) # return the key guess which has Tmin as counter
```

Depending on the chosen key guess and the probability associated with the chosen approximation we obtain a different equation: $0 = K[k_i, k_j, ...]$ or $1 = K[k_i, k_j, ...]$ that we can use to discover more bits. By combining the results of equation 3 and 4 we can obtain an equation with 2 key bits thanks to which we reduce the search space. Anyway, with 8 rounds, we don't need to use it because we discovered such bits by using other approximations.

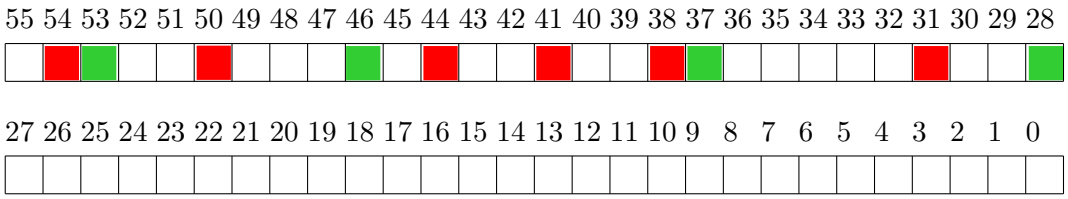
3.1 Steps of the attack and master key recovery step by step

At each step we update the actual knowledge of Master Key.

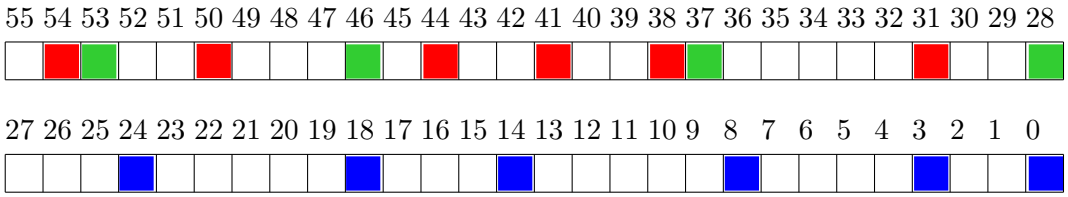
1 Application of Algorithm 2 with equation 3



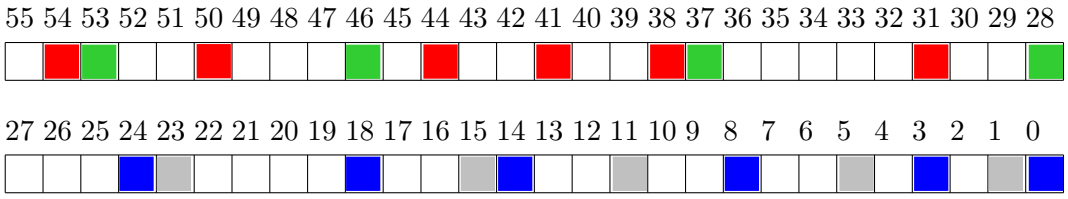
2 Application of Algorithm 2 with equation 3 with few changes (keys are used in the opposite direction, plaintexts treated as ciphertexts and ciphertexts treated as plaintexts)



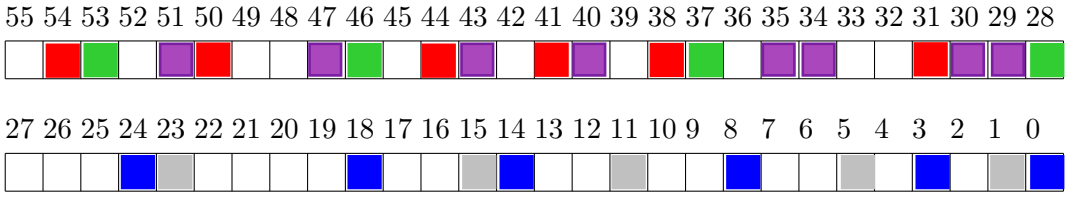
3 Application of Algorithm 2 with equation 4



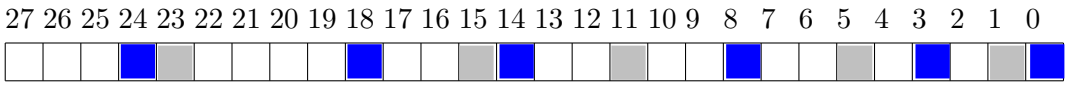
4 Application of Algorithm 2 with equation 4 with few changes (keys are used in the opposite direction, plaintexts treated as ciphertexts and ciphertexts treated as plaintexts)



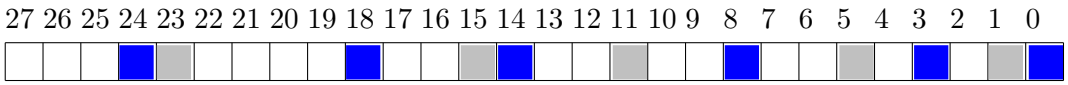
5 Application of Algorithm 2 with equation 5



6 Application of Algorithm 2 with equation 5 with few changes (keys used in the opposite direction, plaintexts treated as ciphertexts and ciphertexts treated as plaintexts)



7 Exhaustive search on the remaining 22 key bits: $O(2^{22})$. At each step, the generated key is tried against 3 plaintext-ciphertext couples in order to find the correct one.



4 Performance

The code was tested on a Ubuntu Virtual Machine with 8gb RAM and 6 CPU cores. For 8 rounds DES, The execution time was more or less 330 seconds (5 minutes and 30 seconds). The performance could be improved by running the code on a physical machine and by using the power of GPUs. As for 5 rounds DES, the execution time was more or less 40 seconds on the same virtual machine.

5 Usage

Open the file **config.h** with your preferred editor and change the key as you want (64 bit key). The used DES implementation[2] uses a 64 bit key in order to generate the 56 bit key. The attack will find the 56 bit key. It is very simple to go from it to one of the possible 64 bit keys which generate the 56 bit one.

Compile

```
1 > g++ main.cpp -o des_key_recovery
```

Running

```
1 > ./des_key_recovery
```

[2]https://github.com/fffaraz/cppDES/tree/master/cppDES