Persistent Fault Analysis on DES Encryption Scheme

Term Project Report (CS 417/617)

Cryptography and Network Security

Submitted By-

Shashank (160001054)

Ashish (160002008)

Sahil (160002051)

Rahul (160001047)

Report Summary -

1) Introduction

In this attack, our objective is to recover the round 16 key bits of DES Block Cipher by performing a Persistent Fault Attack on the last round s-boxes in the encryption scheme. We will be mounting the PFA attack by perturbing the [0][0] entry of each of the S-boxes present in the last round.

1.1) Overview of the Attack

Consider that the attack operates in known Plaintext-Ciphertext mode ***(we will focus on this mode in our attack)*** or chosen Plaintext mode, which can further reduce the number of plaintext-Ciphertext pairs required for the attack. Specifically, for this attack on DES Cipher or any other feistel cipher for that matter, this fault attack will happen in a differential setting, i.e., exploiting the difference of correct and faulty ciphertexts for a fixed input. Later in the attack, we exploit the statistical biases introduced due to fault injection. These biases could be exploited in a differential setting, i.e. with the correct and faulty ciphertext pairs. Later in the time complexity section, we will see approximately how many of these pairs will be needed to successfully recover a 48-bit 16th round key.

There are 3 types of fault attacks:

- ***Transient:***
  Here, a fault is injected during only a target computation, for example, in our case, this involves the round 16 key bits. In this type, the fault doesn't persist from one encryption to another. For our attack, this requires more computation and perturbing bits everytime a new encryption is made on the adversary's part.
- ***Permanent:***
  Here, the fault remains for lifetime, and thus the statistical bias can be studied nicely. However, we also try to find out the key recovery success rate of our attack. Therefore, it needs to switch back-and-forth to correct and faulty modes of operation a couple of times (#Number of trials).
- ***Persistent:***
  This approach is used in our attack. It lies in between the above two mentioned models. The term "persistent" refers to the characteristic of a new type of faults

whose duration may not be permanent and typically can last for several encryptions, for example, a few minutes or up to a few hours. Sometimes it might be persistent till the device is reset. An example of such fault is a modification of a stored constant, like an S-box entry, using rowhammer injection techniques.
In our code, we have simulated the fault to persist during a particular trail, which consists of an adequate number of correct-faulty Ciphertext pairs.

1.2) Fault model:

1. Due to known plaintext-ciphertext scenario, the adversary can already produce the Correct ciphertexts for a given set of plaintexts {p1, p2,....pN} on its end.
2. Then the attacker can inject the fault by perturbing [0][0] bit of each of the S-boxes of last round in the victim's machine. This fault remains till that particular encryption trail.
3. The attacker can then monitor the incorrect ciphertexts for the same plain text input (assuming the master key is not changed) and use them for the PFA attack.

Here, we have tried to inject the fault only in the last round S-boxes. Due to this, none of the previous rounds is affected by the fault. Once we recover the corresponding key bits and the *Second-last round Ciphertexts,* the same approach could be used to recover previous round-keys of the cipher.

1.3) DES Block Cipher:

DES is a feistel cipher of 16 rounds. Each block of 64 input bits after an initial permutation sequence is divided into two blocks of 32 bits each, a left half block **L** and a right half **R.**

There are 16 round keys (each of 48 bits), each of which is generated using the 64-bit **master key** (which includes 8 parity bits). The **keygen() algorithm** can be referred to in the report.

We now proceed through 16 iterations, for 1<=n<=16, using a function f which operates on two blocks--a data block of 32 bits and a key K_n of 48 bits--to produce a block of 32 bits. ***Let ^ denote XOR operation, (bit-by-bit addition modulo 2).*** Then for n going from 1 to 16 we calculate

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} \wedge f(R_{n-1}, K_n)$$

This results in a final block, for n = 16, of $L_{16}R_{16}$.

The output ciphertext Y would be InverseInitialPermutaion($R_{16}L_{16}$)

1.3.1) Function f:

The function f is made up of 4 sub units-

a) Expansion Box - It first expands each block R_{n-1} from 32 bits to 48 bits. This is done by using a selection table that repeats some of the bits in R_{n-1} . We'll call the use of this selection table the function **E**. Thus **E**(R_{n-1}) has a 32 bit input block, and a 48 bit output block.

b) Key Mixing - Here the output of Expansion Box is xored with the round key K_n:

$$K_n \wedge \mathbf{E}(R_{n-1}).$$

c) S-Box layer - We now have 48 bits, or eight groups of six bits. Each group of six bits will give us an address in a different **S** box. Located at that address will be a 4 bit number. This 4 bit number will replace the original 6 bits. The net result is that the eight groups of 6 bits are transformed into eight groups of 4 bits (the 4-bit outputs from the **S** boxes) for 32 bits total.

With the previous result, which is 48 bits, in the form:

$$K_n \wedge \mathbf{E}(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8,$$

where each B_i is of six bits. We now calculate

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$$

where $S_i(B_i)$ referred to the output of the i-th **S** box.

d) Permutation Box - The final stage in the calculation of ***f*** is to do a permutation **P** of the **S**-box output to obtain the final value of ***f*** = **P**($S_1(B_1)S_2(B_2)...S_8(B_8)$)

**(The full implementation of DES encryption can be found in the des.cpp file in the DES folder in the attachment.)

1.3.2) DES S-Box Characteristics: How the Attack Works?

In the DES cipher, every s-box maps a 6-bit value to a 4-bit value. Clearly this is not an injective mapping. However we can take advantage of the way in which this mapping is done. We know that s-box lookup tables are of size 64 (4-*-16 table). **A value-value mapping isn't unique, however, the position in the table where this lookup is made is unique.**

For example, lookup at position [0][0] can be made only when all the bits are 0.

For the sake of simplicity, we are perturbing the [0][0] value of 4*16 lookup table.

2) ATTACK (Step By Step)

We will exploit the property of S-boxes mentioned in the last section in mounting the attack.
The attack can be made as follows: *(We are focusing only on the last round).*

Given - We have N correct and faulty pairs of ciphertext corresponding to the same plaintext. We will prove a more appropriate approximation of the expected value of N for the attack to be successful in the time complexity section.

1. Move up the DES round iteration and calculate the initial permutation of the ciphertext pairs. Both ciphertext were produced from the same plaintext, the only difference being the S-boxes were perturbed while producing one while it was untouched while producing another.

2. Flip the left and right subpart of the obtained result in step one for both the ciphertexts. Note that after flipping , the left 32 bits for both the ciphertexts will be the same. (**But in the implementation the ciphertexts pairs were collected just after the 16th round of encryption of the plaintext ,hence step1 and step2 were not implemented in code).**

3. ***[Attack Code starts from here]:*** XOR the 32 right sub-bits of the correct and faulty ciphertexts **Let a particular xor value be = X.** If the value of X is zero that means both the ciphertexts are exactly the same hence it would not produce any desirable results - Move to step 1 with a different pair of ciphertexts. Else move to step 4.
   Formally, let the pairs of ciphertext were $\{ <C_1 \square C_{1r}>, < C_2 \square C_{2r}> \}$
   $C_1 \square = C_2 \square$
   $C_{1r} \wedge C_{2r} = X$, (the ciphertexts would differ only in last 32 bits)
   $\Rightarrow C_{1r} = L_{15} \wedge f(R_{15})$, $C_{2r} = L_{15} \wedge f'(R_{15})$
   $\Rightarrow f(R_{15}) \wedge f'(R_{15}) = X$
   That is xor of output of round functions on R_{15} with perturbed S-box
   entry and correct S-box entry is X.

4. ***Moving up the DES f-function, we find out the inverse permutation*** of the 32 bit output value X.

5. The value obtained in step 4 is the XOR value of the corresponding bits of the mapped outputs of the s-box for the correct-faulty ciphertext pair.
   All but some of these bits should be non-zero.
   This is because the S-box used for producing those bits was [0][0], and as the faulty ciphertext was produced by perturbing the [0][0] entries in the S-boxes the difference in xors arised, otherwise the parity of bits for both ciphertexts would have been the same.

2.1) Statistical Analysis

Now we focus on the case where we get a non-zero value. Note that we aren't interested in knowing what this XOR value is. We just want to know that at which ciphertext (L_{16}), does this value become non-zero, as mentioned in the above point. **We know that this difference came because of [0][0] lookup in the S-box layer for producing that bit. Had any other S-box entry been used there would be no difference.**

Continuing after step 5 from above,

6. We know the value of L_{16} which is same as R_{15} of last round. Apply Expansion box on this value.
7. The key mixing layer should produce XOR value zero for the [0][0] S-box entry to be used for lookup for filling the bits corresponding to the location where the difference was observed.
8. For XOR to be zero, the Key at that position is the same as the output of the Expansion box with L_{16} as input. Thus a particular sub-byte(of len 6 bits) of the key is found.

More formally,

Key length K = 48 bits.

Divide K into sub bytes of 6 bits each. That is, K= < k0, k1, k2, k3, k4, k5, k6, k7>

Now suppose $E(L_{16})$ = <x0, x1, x2, x3, x4, x5, x6, x7>

Y = inv_permutation(X) ------ (the round function inverse permutation)

If the bit i of Y is non zero (where i ranges from 0 - 32) ,

Suppose □ = i/4 (□= 0,1,2,3...7)

=> k□ ^ E(Y□) = 0

=> k□ = E(Y□)

Using same method depending upon the values of i all the values of k□ could be found and hence the key K.

2.2) Pseudo Codes

- **DES Round function (**In des.cpp file in DES folder**)**

```
1.  Round_function(int round, int mode, bool faulty=false) {
2.          Expansion();                                    // Expansion of right sub-bits
3.          xor_key(round, mode);                           // XOR with Round Key
4.          if(mode == ENC){
                                // Perturb S-box only during encryption
                                // Perturbation only for the last round.....
5.                  if (round == 16 && faulty)
6.                          Perturb();              // Perturbing Bits……
7.
8.                  substitution();                 // Performing substitution lookup..
9.
10.                 if (round == 16 && faulty)
11.                         AntiPerturb();          // Reversing Changes back.....
12.
13.         }
14.         else {
15.                 substitution();
16.         }
17.         permutation();                          // DES f-function permutation...
18.         xor_left();                             // XORing with left sub-bit.....
19. }
```

- **Storing Correct and Faulty Ciphertexts (**In des.cpp file**)**

```
if(round==16 && type == ENC){
        if(!faulty){
                CorrectL.push_back(left);
                CorrectR.push_back(right);
        }
        else{
                FaultyL.push_back(left);
                FaultyR.push_back(right);
        }
}
```

- **Recovering the Key (**In pfa.cpp file**)**

```
1. int totalPairs = CorrectL.size();                    // Total pairs for comparison = N.
2. int xors[32];                                        // XORs of faulty and Correct R's
3. int cnts[32];                                        //** This 0/1 value checks if any of the
                                                        //** 4-bits of 8 S-boxes is set or not

4. for(int i=0;i<32;i++)
        d4.sub[i] = i;                          // Customising substitution values in such a way that
                                                // inverseIP permutation could be found out very easily.

5. for(int i=0; i < totalPairs; i++){
      for(int j=0;j<32;j++){                            // Taking all 32 Right sub-bits of last round

        xors[j] = CorrectR[i][j] ^ FaultyR[i][j]; // XORing correct and Faulty R's
        cnts[d4.p[j]] = xors[j];                // InverseIP permutation sets the respective cnts
                                                // of outputs of S-boxes.

        d4.right[j] = CorrectL[i][j];           // Left Output is unchanged and equal
                                                    // to Previous Round R output.
   }

6. d4.Expansion();                                      // Left Output Expanded.....

7. for(int j=0;j<8;j++){                                // Each of 8 sub-bits
        for(int k=0 ; k<=3 ; k++){
            if(cnts[4*j + k]){                          // Check if any of the 4 bits is 1.
                vector<int> result;                 // Store expanded key bits in the result.
                for(int ind = 0; ind<6; ind++){
                    result.push_back(d4.expansion[6*j + ind]);      // KEY FOUND!!!
                }
                // ------------------------------------------------
                // Store or output the Round Key found!!----------
```

3) Space - Time Complexity Analysis

3.1) Space Complexity

Let's first analyze the space complexity of the attack.
The attack is only perturbing S-boxes entries at specific locations without using any additional space. Hence the space complexity of the attack is the same as the space complexity of Des algorithm which is the sum of space required for storing S-boxes.

Number of bits used for each S-box entry = 4
No. of S-boxes entry in one S-box= 4*16 =64
No. of S-boxes in each round - 8
Total number of rounds = 16

Total number of bits used for storage of all the S-boxes values over all the rounds = 16*8*64*4 = 32.7 bytes (approx.)

3.2) Time Complexity

Let's first analyze the time complexity of Des encryption scheme for a 64 bit plain text.
It's equal to the complexity of each feistel round * 16.
Complexity of a feistel round is the same as the complexity of f function.
Each operation of round function from expansion to permutation is just made up of simple O(1) lookups and wire arrangements to permute the bits and 32 bits Xors.
So the average time complexity of Des is **O(length of plain text).**

Now the question is to find the average number of ciphertext pairs which should be used so that all the 8 sub bytes of the last round key can be found.

Let's only consider for simplicity the average number of ciphertext pairs required to predict the subkey k0 of K= < k0, k1, k2, k3, k4, k5, k6, k7>

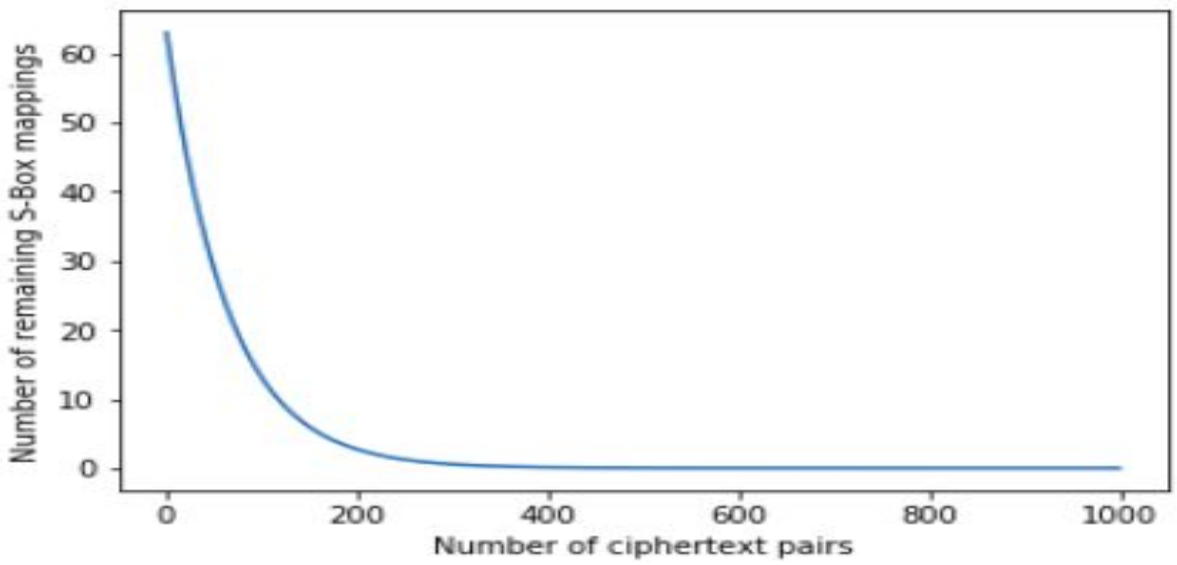
The total number of possible key set size for k0 = (2^6) = 64
Also assuming the ciphertext pairs are drawn from a uniform random distribution. We need to find the average number of ciphertexts pair to be drawn such that all the inputs are covered.

Let N be the total number of ciphertexts that are available and n is the number ciphertexts already used for analysis. Let θ_n denote the average number of different inputs of S-box lookups using N ciphertexts (considering only one entry in the lookup table is perturbed). When n is large θ_n will converge to $\eta = 2^b$ (here, b = 6).
After computing for θ_n we have,

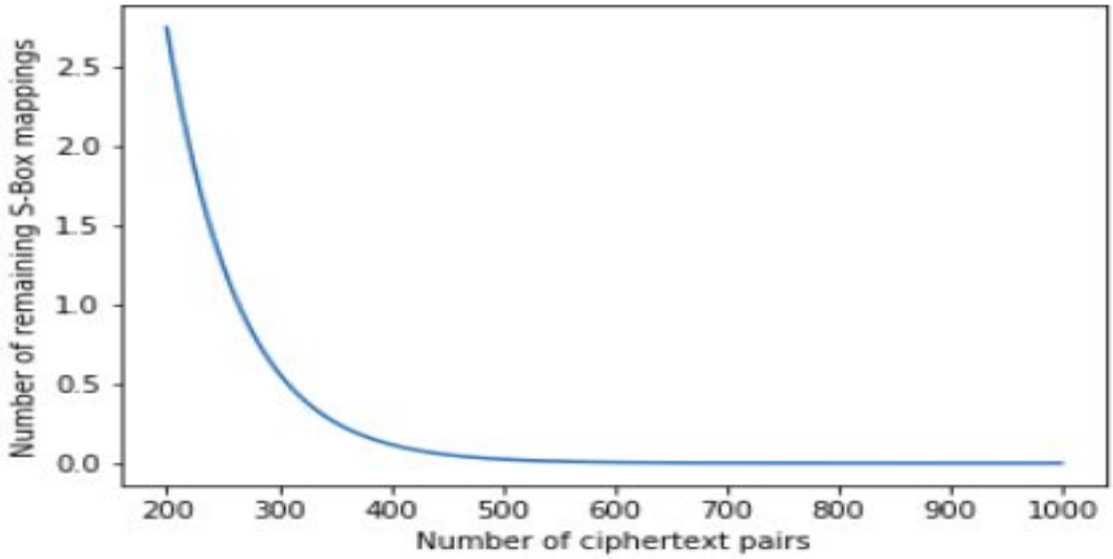
$$\theta_n = \frac{1 - q^n}{1 - q}, \quad \text{where} \quad q = \frac{\eta - 1}{\eta}$$

(The proof for above is provided in the paper, except for the case that in the paper, there's one key which is never bound to appear and that's the correct key. Whereas for our case, we just want to find out the worst case number of ciphertext pairs such that all 64 possible sbox positions are mapped, which ensures that 0,0 position will be mapped in any case. Therefore we just need to take $\eta = 2^b$ instead of $2^b - 1$ and wait till epsilon, i.e. the number of remaining mappings left becomes equal 0).

Let $\in_n$ denote the average number of remaining possible lookups/mappings after using n ciphertexts.
We have $\in_n = 2^b - \theta_n$.
Figure below describes the relationship between $\in_n$ and n where b = 6 (for DES).



More precisely:



The exact value of Expected number of Ciphertext pairs required (N) can be estimated by Coupon's Collector problem. Here, the number of coupons to be collected are 64.
So N = 64 * (1/1 + 1/2 + ... + 1/64) = 384.

So, when N is close to ~384, ϵ_n can be reduced to 0, which gives the required key candidate in the worst case time complexity.

So, approximately, 384 * 8 = **3072** (since we have to find 8 sub parts of key K) **ciphertext pairs** are required to recover the full last round key (48 bits).

Therefore, if we use a single string to encrypt, it's length must be ~3072 for a single sub-bit recovery (Because, each character takes 1 byte, so we need 8 characters to have 64-bits input plaintext).

4) Results

The results obtained were satisfactory using approximately 384 correct-faulty ciphertext pairs for 100 trials.
The attack was mounted with randomly generated plain text strings of this length. This plain text string was used to generate both correct and faulty ciphertexts. These ciphertexts were analysed using the techniques mentioned in the previous section and the key bits were guessed.
For every hundred randomly generated strings of this length , in about 15 strings it was possible to correctly guess ***at least one of the sub-keys*** of the last round key.

* (The detailed results can be found at output.txt file in IO_files folder)

Is the attack performance invariant of sbox mapping?
Yes, it's invariant because the attack exploits the fact that the fault has been injected at a particular position, ie. [0][0] of the S-box table. Changing the mapping of the S-box doesn't have any effect on the fact that the key is recovered only when the XOR of E(A) and ROUND KEY equals to {0,0,0,0,0,0}.

4.1) Important Notes:

1. *In the attack, we have perturbed the [0][0] entry for each of the 8 S-Boxes all at once. Since the permutation applied at the end of DES f-function gives a unique mapping and doesn't interfere with the outputs of each-other's S-boxes, this can*

be done. ***Experiments show that this approach proved to be useful in some trials, where we recovered more than 1 sub-bytes of the 16th round key .***

2. *Also, we have used 100 trials, to determine the success rate of our attack, Experiments show that this is between 10% to 20%, which is pretty convincing.*

3. ***About the strings used in attack:*** *Experiments showed that increasing the diversity of characters used in our string increased the chances of key recovery. For our case, we have used readable characters of English Text, ie. the ones which have their ASCII values between 32 and 126 (both inclusive). Therefore, diversity of our strings is = 95.* ***Note that this doesn't make the attack fall under the 'chosen plaintext category', because we aren't using patterns in our input plaintext, they are generated completely randomly using srand() and rand() functions.***

4.2) Implemented Code for the Attack:

1. DES Implementation in ***"des.cpp"*** *file in DES folder*
2. PFA mounting code in ***"pfa.cpp"*** file.
3. Lookup tables for initial permutation, E-box, S-box, etc in the ***"lookup.h"*** file in the DES folder.
4. Actual 16th round key in ***"actualKey.txt"*** in IO_files folder
5. Recovered output keys in ***"output.txt"*** in IO_files folder.