Cryptography and Network Security

(CS 417/617)

Persistent fault analysis on DES block cipher

Instructor: Dr. Bodhisatwa Mazumdar

Submitted By:

Surya Teja Togaru - 160001059

Sai Kumar Reddy - 160001043

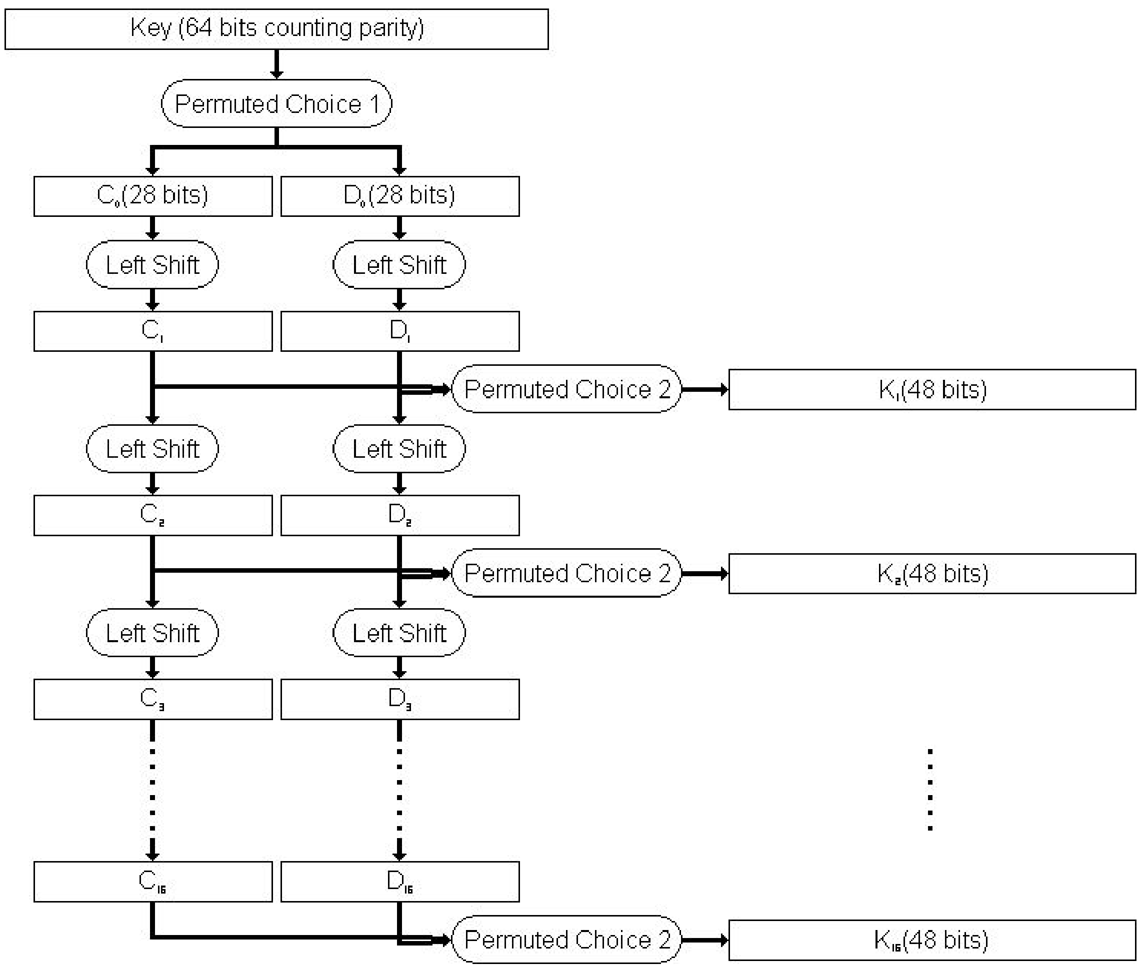
Kovidh Vengala - 160001030

Vandana Varakantham - 160001060

1. DES Implementation:

1.1. Key Generation Algorithm:

The initial key used in DES is of 64 bits. Later in the key generation algorithm, only 56 bits of the complete key are used. The 56 bits key is generated by eliminating 8 bits of parity (8,16,24,32,40,48,56,64 corresponding positions) from the complete 64 bit key.



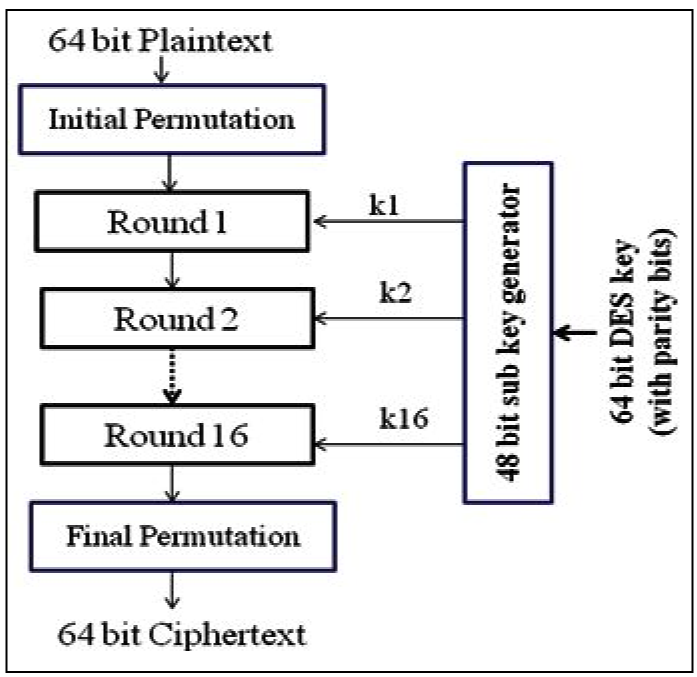
For each and every round in the DES, a distinct 48-bit key is generated from the 56-bit key generated. Firstly, the 56 bit is divided into two equal parts and each part is shifted based on the round number. After the shift, the two halves are combined and passed into the compression P-box to reduce the key to 48-bit.

The number of bits shifted is 1 (to the left) for the rounds 1,2,9,16 and shifted by 2 for the remaining rounds. The permutation which the compression box uses is shown below.

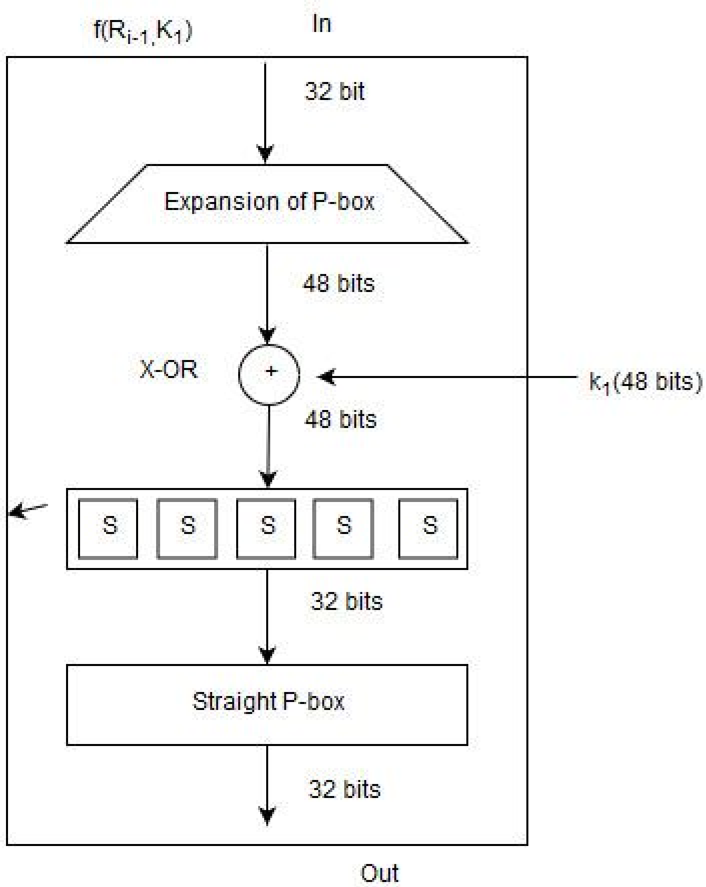
14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

1.2. DES:

The DES takes in 64-bit plain text and using a 56-bit key generates 64-bit ciphertext. Initially, it uses the initial permutation to permute the PT and later sends the output to 16 round functions and finally uses the inverse permutation to generate the 64-bit ciphertext.



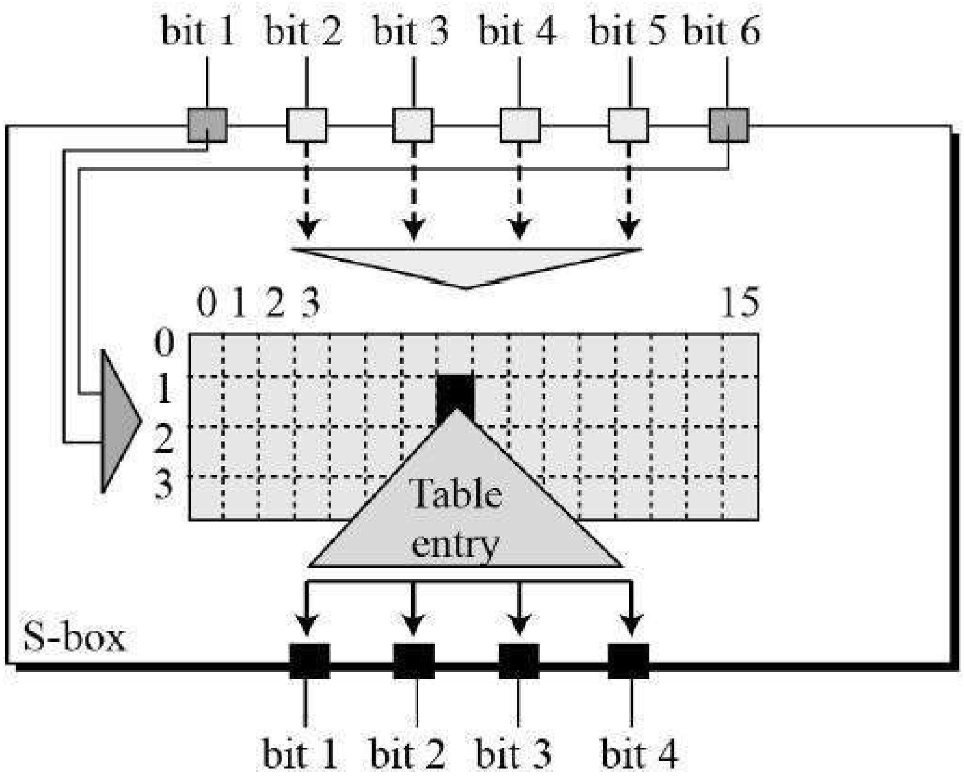
Round Function consists of expansion Pbox, SBoxes and straight PBox. The initially permuted plain text is divided into two parts left and right. For each part (32 bits) is sent to the expansion Pbox which converts it into 48 bits. Later the 48-bit output is xored with the corresponding round key.



The xored output (divided into 8 parts of 6 bits each) is sent to SBoxes. In each Sbox the 6-bit output is converted into 4-bit output. All combined makes it 32 bits, which is later sent into Straight P-Box. The final output (32 bits) is sent to subsequent round functions.

S-Box:

A 6x4 substitution box is used in these round functions. This SBox comprises a 4x16 lookup table. Each entry consists of a 4-bit value. The input 6 bits corresponds to a unique entry in the lookup table and returns the corresponding 4-bit value of the entry.



1.3 Code:

Main Function:

```
string des(string pt, string key, int s[8][4][16]){

  vector<string> rkb(16); //RoundKeys in binary
  rkb = keygen(key);

  string cipher = encrypt(pt, rkb, s);
  return cipher;
}
```

Key generation Function:

```
vector<string> keygen(string key)
{
  // getting 56 bit key from 64 bit by dropping the parity bits
  key = permute(key, keyp, 56); // key without parity

  key56 = key;
  // Splitting
  string left = key.substr(0, 28);
  string right = key.substr(28, 28);

  vector<string> rkb(16); // rkb for RoundKeys in binary
  for (int i = 0; i < 16; i++) {
    // Shifting
    left = shift_left(left, shift_table[i]);
    right = shift_left(right, shift_table[i]);

    // Combining
    string combine = left + right;

    // Key Compression
    string RoundKey = permute(combine, key_comp, 48);

    rkb[i] = RoundKey;
  }
  for(int i = 0; i<16; ++i)
  {
    rk[i] = bin2hex(rkb[i]);// Hexadecimal Keys
  }
  return rkb;
}
```

Encryption Function:

```
string encrypt(string pt, vector<string> rkb, int s[8][4][16])
{
	// Hexadecimal to binary
	pt = hex2bin(pt);
	// Initial Permutation
	pt = permute(pt, initial_perm, 64);
	// Splitting
	string left = pt.substr(0, 32);
	string right = pt.substr(32, 32);
	for (int i = 0; i < 16; i++) {
		// Expansion D-box
		string right_expanded = permute(right, exp_d, 48);
		// XOR RoundKey[i] and right_expanded
		string x = xor_(rkb[i], right_expanded);
		// S-boxes
		string op = "";
		for (int i = 0; i < 8; i++) {
		int row = 2 * int(x[i * 6] - '0') + int(x[i * 6 + 5] - '0');
		int col = 8 * int(x[i * 6 + 1] - '0') + 4 * int(x[i * 6 + 2]
- '0') + 2 * int(x[i * 6 + 3] - '0') + int(x[i * 6 + 4] - '0');
		int val = s[i][row][col];
		op += char(val / 8 + '0');
		val = val % 8;
		op += char(val / 4 + '0');
		val = val % 4;
		op += char(val / 2 + '0');
		val = val % 2;
		op += char(val + '0');
	}
	// Straight D-box
	if(i<15)
	op = permute(op, per, 32);
	// XOR left and op
	x = xor_(op, left);
	left = x;
	// Swapper
	if (i != 15) {
		swap(left, right);
	}
}
```

Output:

```
Plain Text: 123456ABCD132536

Encryption:

After initial permutation: 14A7D67818CA18AD
After splitting: L0=14A7D678 R0=18CA18AD

         Left_part Right_part Round_key
Round  1 18CA18AD  5A78E394   194CD072DE8C
Round  2 5A78E394  4A1210F6   4568581ABCCE
Round  3 4A1210F6  B8089591   06EDA4ACF5B5
Round  4 B8089591  236779C2   DA2D032B6EE3
Round  5 236779C2  A15A4B87   69A629FEC913
Round  6 A15A4B87  2E8F9C65   C1948E87475E
Round  7 2E8F9C65  A9FC20A3   708AD2DDB3C0
Round  8 A9FC20A3  308BEE97   34F822F0C66D
Round  9 308BEE97  10AF9D37   84BB4473DCCC
Round 10 10AF9D37  6CA6CB20   02765708B5BF
Round 11 6CA6CB20  FF3C485F   6D5560AF7CA5
Round 12 FF3C485F  22A5963B   C2C1E96A4BF3
Round 13 22A5963B  387CCDAA   99C31397C91F
Round 14 387CCDAA  BD2DD2AB   251B8BC717D0
Round 15 BD2DD2AB  CF26B472   3330C5D9A36D
Round 16 19BA9212  CF26B472   181C5D75C66D

Cipher Text: C0B7A8D05F3A829C

Decryption

After initial permutation: 19BA9212CF26B472
After splitting: L0=19BA9212 R0=CF26B472

         Left_part Right_part Round_key
Round  1 CF26B472  BD2DD2AB   181C5D75C66D
Round  2 BD2DD2AB  387CCDAA   3330C5D9A36D
Round  3 387CCDAA  22A5963B   251B8BC717D0
Round  4 22A5963B  FF3C485F   99C31397C91F
Round  5 FF3C485F  6CA6CB20   C2C1E96A4BF3
Round  6 6CA6CB20  10AF9D37   6D5560AF7CA5
Round  7 10AF9D37  308BEE97   02765708B5BF
Round  8 308BEE97  A9FC20A3   84BB4473DCCC
Round  9 A9FC20A3  2E8F9C65   34F822F0C66D
Round 10 2E8F9C65  A15A4B87   708AD2DDB3C0
Round 11 A15A4B87  236779C2   C1948E87475E
Round 12 236779C2  B8089591   69A629FEC913
Round 13 B8089591  4A1210F6   DA2D032B6EE3
Round 14 4A1210F6  5A78E394   06EDA4ACF5B5
Round 15 5A78E394  18CA18AD   4568581ABCCE
Round 16 14A7D678  18CA18AD   194CD072DE8C

Plain Text: 123456ABCD132536
```

2. Persistent Fault Analysis on DES:

The standard Persistent Fault Analysis cannot be applied on DES. This is due to the fact that the round functions abstract the two halves of the output to the subsequent round function. One of the advantages of the DES is it does not require the S-box to be injective. So in order to mount the attack, we need to reconsider the ciphertext only requirement.

2.1. Recovery Algorithm for 16th round key:

In Persistent Fault Analysis (PFA), in order to recover a particular part of a round key, we need to attack the corresponding S-Box. DES comprises 8 S-Boxes which are of 6x4 tables.

Let the S-Boxes be S1,S2….S8. Let the input and output for the last rounds be (xl, xr) and (yl, yr) respectively. Since there are eight Boxes we divide the input and output to eight parts of 4 bits each.

To retrieve the last part of the round key, we need to change the entry in S_8. Suppose let us say the correct S-box to be S_8 and faulty Sbox to be S_8*. Using the correct S_8, let the output be yl and yr and the corresponding faulty output to be yl* and yr*.

Let 'a' be the XORed result of the correct output and the faulty output. By doing the XOR operation, we can find the corresponding input that accessed the fault entry in the last round. This gives us the last 6 bits of the round key.

Similarly, in this way, we can retrieve the remaining part of the 16th round key.

2.2. Algorithm:

$p, cl, cr \leftarrow [\cdot]$ // Initialize empty lists
for $i \leftarrow 0;\ i < n;\ i \leftarrow i + 1$ **do**
    $p(i) \leftarrow$ random plaintext
    $y_l || y_r \leftarrow E(p(i))$
    $cl(i), cr(i) \leftarrow y_l, y_r$
end
Overwrite element e of S_t in E
for $i \leftarrow 0;\ i < n;\ i \leftarrow i + 1$ **do**
    $y_l' || y_r' \leftarrow E(p(i))$
    $|a_1, \ldots, a_b| \leftarrow y_l \oplus y_l'$
    if $a_j = 0,\ j \in \{1, \ldots, b\} \setminus \{t\} \wedge a_t \neq 0$ **then**
        return $y_r^t \oplus e$
    end
end

```
//Store the round 16,15 and the master keys.
vector<string> recoveredKey(3, "");
int numTrials = 0;
cout<<endl<<"Recovering Round 16 key..."<<endl;
//Round 16 key recovery
for(int i = 0; i<8; ++i)
{
    // cout<<"Attacking S-box "<<i<<" and entry is
"<<fault[i][0][0]<<endl;
    // Overwrite 0,0 element of S-box
    fault[i][0][0] = 0; // change o,o to FAULT_ROW adn FAULT_COLUMN
    do {
      //Run the faultless and faulty DES encryption.
      string pt = genRandomHexString(16);
      string correct_cipher = des(pt,key,s);
      string wrong_cipher = des(pt,key,fault);

      //Reverse the Final Permutation
      correct_cipher = bin2hex(permute(hex2bin(correct_cipher),
initial_perm, 64));
      wrong_cipher = bin2hex(permute(hex2bin(wrong_cipher), initial_perm,
64));
```

```
      //Get the right part of the cipher
      string yr = hex2bin(correct_cipher).substr(32,32);
      string yr_ = hex2bin(wrong_cipher).substr(32,32);
      uint64_t right = strtouint64(correct_cipher);
      uint64_t wrong = strtouint64(wrong_cipher);

      //a = xor(yl,yl_) --> xor the left parts
      uint32_t a = (right^wrong)>>32;
      numTrials++;
      //check if the right parts of the cipher are equal and validate the
      //xor of the left parts.
      if(correct_cipher.compare(wrong_cipher) && !(yr.compare(yr_)) &&
checkProperty(a, i))
      {
        //Expand the right part of the cipher to 48-bits
        string yrExpanded = permute(yr, exp_d, 48);
        //Add the 6 bits (ith word of yrExpanded) to recover the key
        recoveredKey[0] += yrExpanded.substr(6*i,6);
        cout<<"Using S-box "<<i<<":\t"<<"Key bits recovered: "<<
          yrExpanded.substr(6*i,6)<<"\tTrials:"<<numTrials<<endl;
        break;
      }
  } while(true);
  numTrials = 0;
  //Reset the S-box to correct entry
  fault[i][0][0] = s[i][0][0];
}
```

Recovering the 56-bit key:

After recovering the last round key, we decrypt the ciphertext using the key to generate the output of the 15th round. Now using this output, we can similarly mount the attack to generate the previous round key after decrypting i.e. 15th round key.

We reverse the recovereds round key using expansion P-Box which gives us 56 bits (some bits will be missing). We use the fact that the missing bits in different round keys will be at different positions. Thus, by reverting the key scheduling algorithm i.e.,expanding the 48-bit keys, shifting one of them and OR'ing the two components, we can extract the final key.

56-Bit Key Recovery Function:

```
string reverseKeySchedule(string key1, string key2)
{
    string key1Expanded = permute(key1, key_exp, 56);
    string key2Expanded = permute(key2, key_exp, 56);
    //Splitting the keys
    string left1 = key1Expanded.substr(0, 28);
    string right1 = key1Expanded.substr(28, 28);
    string left2 = key2Expanded.substr(0, 28);
    string right2 = key2Expanded.substr(28, 28);
    //Tracing back
    left2 = shift_left(left2,1);
    right2 = shift_left(right2,1);
    string last = left1+right1;
    string  onelasttime = left2+right2;
    return bin2hex(or_(last, onelasttime));
}
```

Output:

```
ayrus@XAPYRUS:/mnt/c/Users/Surya/Documents/Edu/Undergrad - IIT Indore/Crypto$ ./main
Round 00 Key: D064A070A885
Round 01 Key: 54A82622A0CD
Round 02 Key: A2AC22A2B583
Round 03 Key: E826262E0723
Round 04 Key: E096185E4942
Round 05 Key: 44927244C158
Round 06 Key: A6D852C1B448
Round 07 Key: 2E6342E89628
Round 08 Key: 2713551BD428
Round 09 Key: 0F50C1085D24
Round 10 Key: 1B41F88868B4
Round 11 Key: 9C4189E14A91
Round 12 Key: 130B0D93021B
Round 13 Key: 093885971304
Round 14 Key: 112CEC1023E4
Round 15 Key: 408C9C5460C5

Recovering Round 16 key...
Using S-box 0:  Key bits recovered: 010000      Trials:52
Using S-box 1:  Key bits recovered: 001000      Trials:106
Using S-box 2:  Key bits recovered: 110010      Trials:65
Using S-box 3:  Key bits recovered: 011100      Trials:2
Using S-box 4:  Key bits recovered: 010101      Trials:59
Using S-box 5:  Key bits recovered: 000110      Trials:93
Using S-box 6:  Key bits recovered: 000011      Trials:35
Using S-box 7:  Key bits recovered: 000101      Trials:51

Recovering Round 15 key...
Using S-box 8:  Key bits recovered: 000100      Trials:11
Using S-box 8:  Key bits recovered: 010010      Trials:11
Using S-box 8:  Key bits recovered: 110011      Trials:21
Using S-box 8:  Key bits recovered: 101100      Trials:5
Using S-box 8:  Key bits recovered: 000100      Trials:66
Using S-box 8:  Key bits recovered: 000010      Trials:7
Using S-box 8:  Key bits recovered: 001111      Trials:8
Using S-box 8:  Key bits recovered: 100100      Trials:18

Recovering 56-bit key...
Round 16 Key: 408C9C5460C5
Round 15 Key: 112CEC1023E4
Extracted Key: 0202F6E1FC081A
Original Key:  0202F6E1FC081A
```

3. Analysis:

Attacking a particular SBox will produce 6 bits of the corresponding round key. As we recover the different blocks of the round key, the residual key space reduces correspondingly. The residual key space is calculated based on the number of bits left to recover. Suppose the number of bits left to recover is ‘r’, then the residual key space will be: $2^r - 1$.

After the complete recovery, the residual space will be 0.

The table contains the data of the number of queries performed to retrieve the sub key by mounting the attack on each S-box.

Faulty S-box	Round 16 Key bits recovered	Number of Queries	Round 15 Key Bits Recovered	Number of Queries
S_1	010000	52	000100	11
S_2	001000	106	010010	11
S_3	110010	65	110011	21
S_4	011100	2	101100	5
S_5	010101	59	000110	66
S_6	000110	93	000010	7
S_7	000011	35	001111	8
S_8	000101	51	100100	18

The attack mounted on the DES is performance invariant on the S-box mapping. The change in the fault entry position will only change the number of executions required to retrieve the key (can be observed from the table). The number of executions is also dependent on the input 64 bit (PT). Since the PT are generated at random the number of executions vary widely. The S-box mapping does not alter the performance of the attack.

4. Complexity Analysis of the attack:

Consider a Fiestel cipher which has an 'r' number of rounds in the encryption process. In each round let us say it uses a ‘b’ number of S-boxes. We assume all the S-boxes are different from each other.

Suppose we inject a fault in one of the S-box, the probability of that entry being hit is $p = \frac{1}{2^n}$ (n is the size of the input to the S-box). The probability that this entry is not hit is $p' = 1 - \frac{1}{2^n}$. The probability that this entry not being hit in all the rounds is $(1 - \frac{1}{2^n})^{r}$.

In our algorithm, in order to retrieve the partial key, the faulty entry must be hit only in the last round. Finally, the probability of ‘e’ being hit only at the last round is $\left(\frac{1}{2^n}\right)(1 - \frac{1}{2^n})^{r-1}$. The number of ciphertexts required will be the inverse of the probability i.e. $(2^n)(1 - \frac{1}{2^n})^{-r+1}$

For the attack we implemented on the DES, r=16, b=8, n=6, and m=4. So, the probability that the faulty element will only be hit in the last round is $\left(\frac{1}{2^6}\right)(1 - \frac{1}{2^6})^{15} \approx 0.0123.$

The expected number of required ciphertexts is $0.0123^{-1} \approx 82$

Approximately 82 ciphertexts are required to retrieve the sub part of the last round key which is of 6 bit. For the complete last round key (48 bit) we need 656 ciphertexts.

References:

1. Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. Persistent fault analysis on block ciphers. IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 150{172, 2018.
2. Caforio, Andrea and Subhadeep Banik. “A Study of Persistent Fault Analysis.” IACR Cryptology ePrint Archive 2019 (2019): 1057.