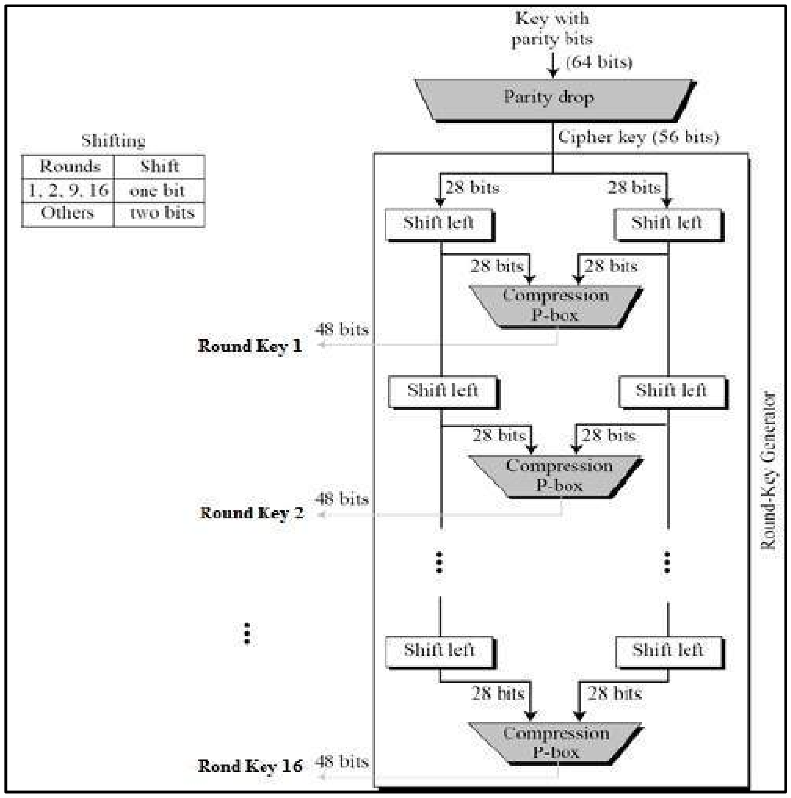
Cryptography and Network Security (CS 417/617)
Term Project Report
Spring Semester 2019-2020
Instructor: Dr. Bodhisatwa Mazumdar

Team Members:
Arushi Jain (160001008)   Mounika Mukkamalla (160001036)
Neha Nagendra (160001042)   Priyanka Rotte (160001051)

1. Data Encryption Standard (DES)

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).
DES is an implementation of a Feistel Cipher. It uses a 16 round Feistel structure. The block size is 64-bit. Though the key length is 64-bit, DES has an effective key length of 56 bits.

1.1 Key generation algorithm



The initial key consists of 64 bits. However, before the DES process starts, every 8th bit of the key is discarded to produce a 56-bit key i.e. bit position 8, 16, 24, 32, 40, 48, 56 and 64 are discarded.

1	2	3	4	5	6	7	~~8~~	9	10	11	12	13	14	15	~~16~~
17	18	19	20	21	22	23	~~24~~	25	26	27	28	29	30	31	~~32~~
33	34	35	36	37	38	39	~~40~~	41	42	43	44	45	46	47	~~48~~
49	50	51	52	53	54	55	~~56~~	57	58	59	60	61	62	63	~~64~~

From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called key transformation. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of bits to be shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

After an appropriate shift, 48 of the 56 bit are shuffled according to the fixed permutation table(given below) and selected. So each round key is 48-bits long.

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

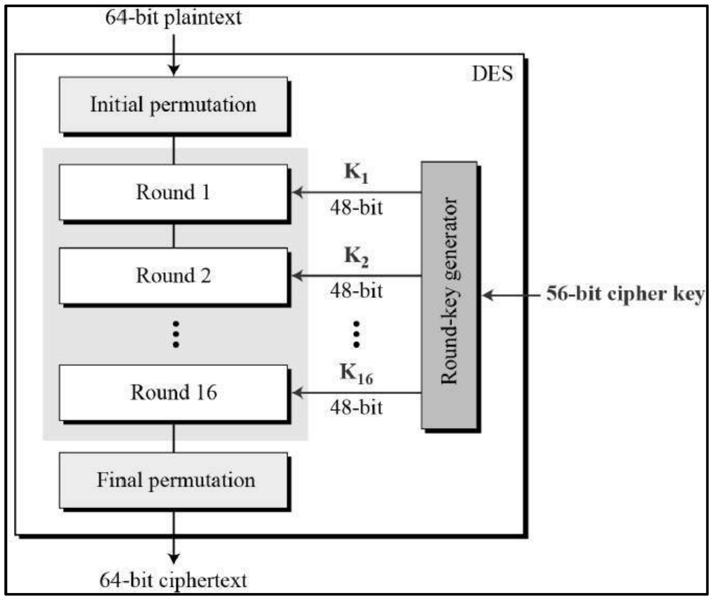
DES is based on substitution (confusion) and transposition (diffusion). DES consists of 16 steps, each of which is called “round”. Each round performs the steps of substitution and transposition.

Pseudo-code for DES

```
1. PT - 64-bit plain text
2. PT' = Initial Permutation(PT)
3. $X_l$, $X_r$ = divide_in_middle(PT')
4. For 16 rounds:
      a. $X_r'$ = Expand($X_r$)
      b. $Z_r$ = $X_r'$ XOR key
      c. $Y_r$ = $X_l$ XOR $X_r'$
      d. $Y_l$ = $X_r$
5. Switch $Y_l$ and $Y_r$
6. CT' = join($Y_l$,$Y_r$)
7. CT = Final Permutation(CT')
```

CT is the 64-bit ciphertext.

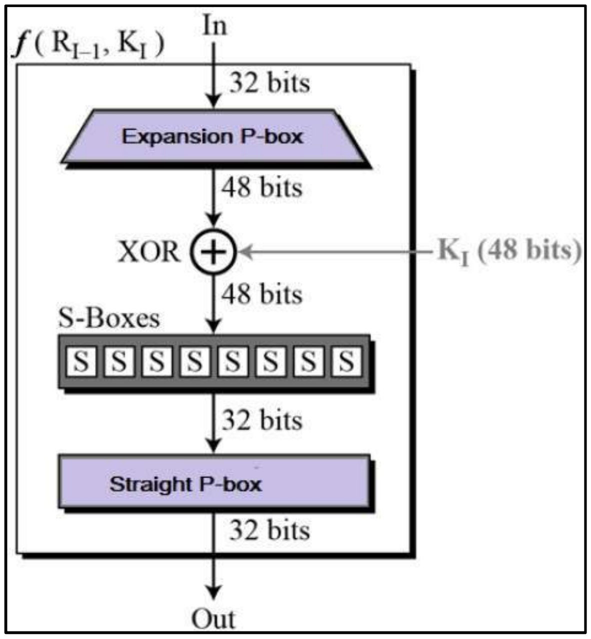
DES structure is given below. DES is specified by initial and final permutations, round function and key generator



The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES.

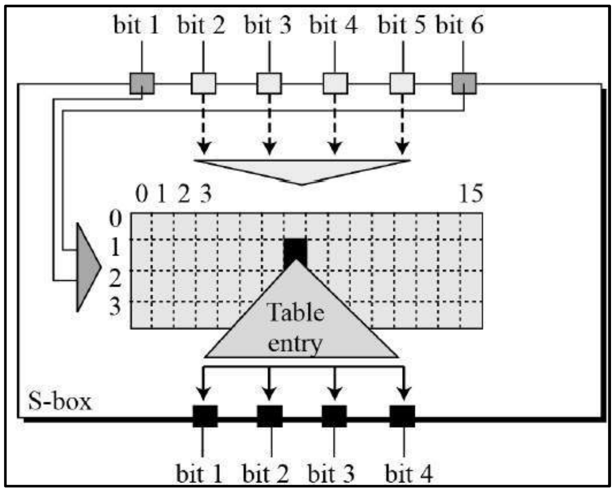
Round Function:

The heart of this cipher is the DES function, *f*. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.



Since the right input is 32-bit and the round key is 48-bit, we first need to expand the right input to 48 bits. For this, an expansion permutation box is used which converts 32 bit to 48-bit output.

Each S-Box has an internal structure as illustrated below.



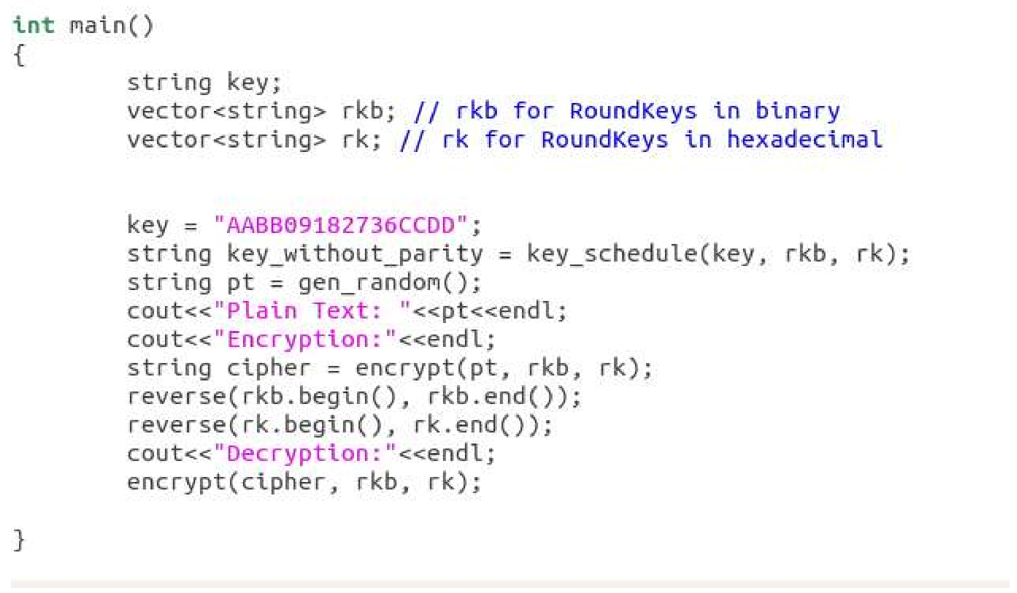
The S-Boxes here are 6*4 substitution boxes. In each S-box, we have a 4*16 lookup table where each block of the table is a 4-bit value. When a 6-bit input is given, it points out a block in the lookup table and this is outputted from the s-box.

Block ciphers are designed with reversibility in mind. So the same logic can be used to decrypt the ciphertext. Having the right key, the receiver of CT can use the round keys in the reverse order on the same system to get the plain text back.

1.3 Implementation

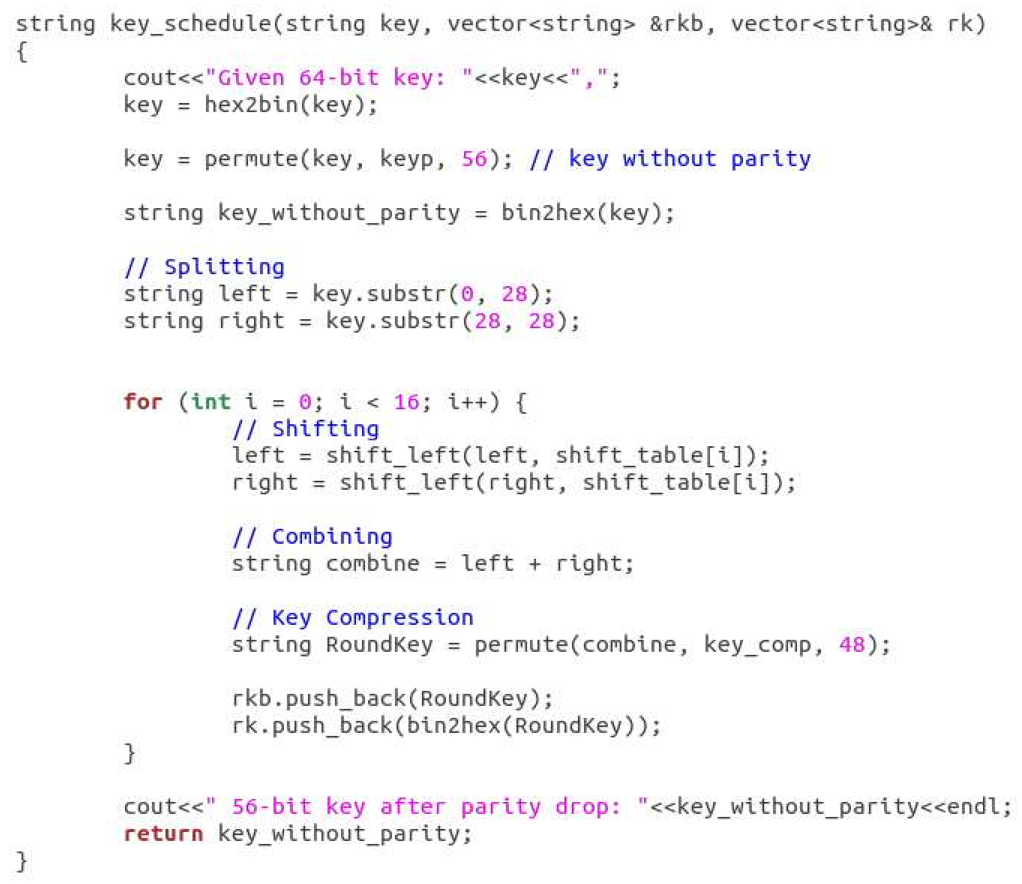
The encryption algorithm is shown here.

a. The main()



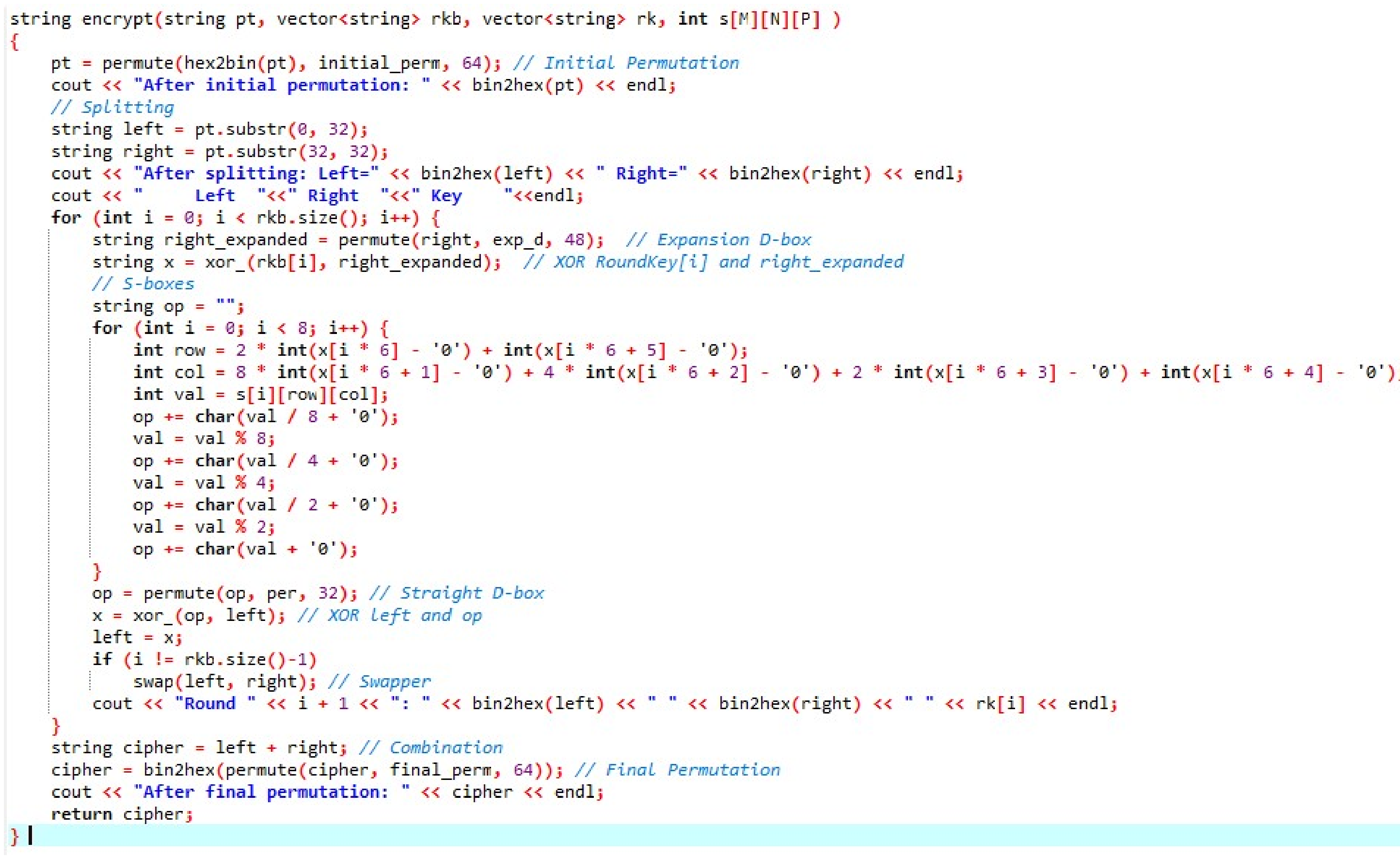
```
int main()
{
        string key;
        vector<string> rkb; // rkb for RoundKeys in binary
        vector<string> rk; // rk for RoundKeys in hexadecimal


        key = "AABB09182736CCDD";
        string key_without_parity = key_schedule(key, rkb, rk);
        string pt = gen_random();
        cout<<"Plain Text: "<<pt<<endl;
        cout<<"Encryption:"<<endl;
        string cipher = encrypt(pt, rkb, rk);
        reverse(rkb.begin(), rkb.end());
        reverse(rk.begin(), rk.end());
        cout<<"Decryption:"<<endl;
        encrypt(cipher, rkb, rk);

}
```

b. Round key generation function()



```
string key_schedule(string key, vector<string> &rkb, vector<string>& rk)
{
        cout<<"Given 64-bit key: "<<key<<",";
        key = hex2bin(key);

        key = permute(key, keyp, 56); // key without parity

        string key_without_parity = bin2hex(key);

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


        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.push_back(RoundKey);
                rk.push_back(bin2hex(RoundKey));
        }

        cout<<" 56-bit key after parity drop: "<<key_without_parity<<endl;
        return key_without_parity;
}
```

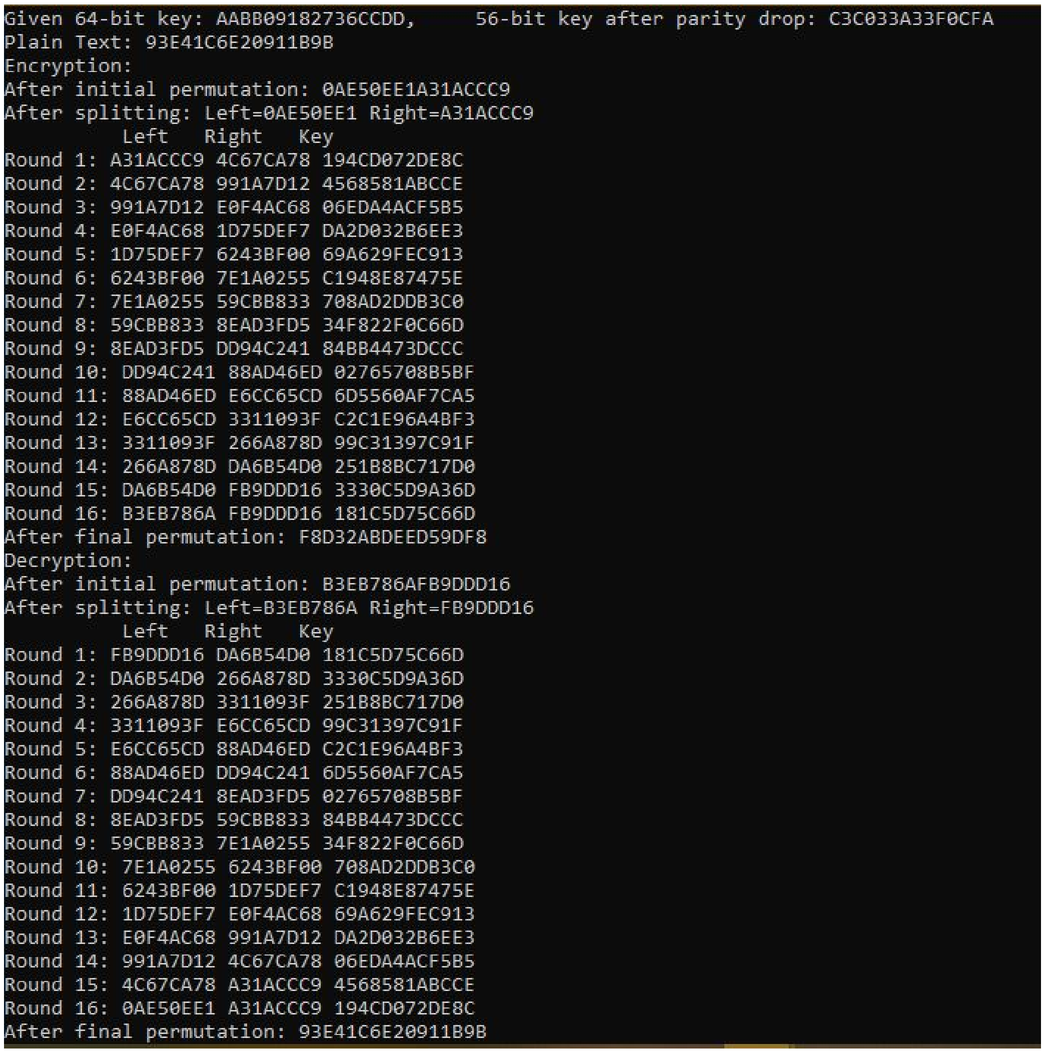
c. The encryption and decryption algorithm
   This same algorithm can perform the decryption operation as well. If reverse the key vector rkb and send ciphertext in place of plain text this function will decrypt it and produce the plain text back. This happens because reversibility is an inherent property of a des cipher.



```
string encrypt(string pt, vector<string> rkb, vector<string> rk, int s[M][N][P] )
{
    pt = permute(hex2bin(pt), initial_perm, 64); // Initial Permutation
    cout << "After initial permutation: " << bin2hex(pt) << endl;
    // Splitting
    string left = pt.substr(0, 32);
    string right = pt.substr(32, 32);
    cout << "After splitting: Left=" << bin2hex(left) << " Right=" << bin2hex(right) << endl;
    cout << "     Left  "<<" Right  "<<" Key    "<<endl;
    for (int i = 0; i < rkb.size(); i++) {
        string right_expanded = permute(right, exp_d, 48);  // Expansion D-box
        string x = xor_(rkb[i], right_expanded);  // XOR RoundKey[i] and 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');
        }
        op = permute(op, per, 32); // Straight D-box
        x = xor_(op, left); // XOR left and op
        left = x;
        if (i != rkb.size()-1)
            swap(left, right); // Swapper
        cout << "Round " << i + 1 << ": " << bin2hex(left) << " " << bin2hex(right) << " " << rk[i] << endl;
    }
    string cipher = left + right; // Combination
    cipher = bin2hex(permute(cipher, final_perm, 64)); // Final Permutation
    cout << "After final permutation: " << cipher << endl;
    return cipher;
}
```

The complete code for the des algorithm is attached to this report.

1.4 Output of DES



```
Given 64-bit key: AABB09182736CCDD,     56-bit key after parity drop: C3C033A33F0CFA
Plain Text: 93E41C6E20911B9B
Encryption:
After initial permutation: 0AE50EE1A31ACCC9
After splitting: Left=0AE50EE1 Right=A31ACCC9
          Left   Right   Key
Round 1: A31ACCC9 4C67CA78 194CD072DE8C
Round 2: 4C67CA78 991A7D12 4568581ABCCE
Round 3: 991A7D12 E0F4AC68 06EDA4ACF5B5
Round 4: E0F4AC68 1D75DEF7 DA2D032B6EE3
Round 5: 1D75DEF7 6243BF00 69A629FEC913
Round 6: 6243BF00 7E1A0255 C1948E87475E
Round 7: 7E1A0255 59CBB833 708AD2DDB3C0
Round 8: 59CBB833 8EAD3FD5 34F822F0C66D
Round 9: 8EAD3FD5 DD94C241 84BB4473DCCC
Round 10: DD94C241 88AD46ED 02765708B5BF
Round 11: 88AD46ED E6CC65CD 6D5560AF7CA5
Round 12: E6CC65CD 3311093F C2C1E96A4BF3
Round 13: 3311093F 266A878D 99C31397C91F
Round 14: 266A878D DA6B54D0 251B8BC717D0
Round 15: DA6B54D0 FB9DDD16 3330C5D9A36D
Round 16: B3EB786A FB9DDD16 181C5D75C66D
After final permutation: F8D32ABDEED59DF8
Decryption:
After initial permutation: B3EB786AFB9DDD16
After splitting: Left=B3EB786A Right=FB9DDD16
          Left   Right   Key
Round 1: FB9DDD16 DA6B54D0 181C5D75C66D
Round 2: DA6B54D0 266A878D 3330C5D9A36D
Round 3: 266A878D 3311093F 251B8BC717D0
Round 4: 3311093F E6CC65CD 99C31397C91F
Round 5: E6CC65CD 88AD46ED C2C1E96A4BF3
Round 6: 88AD46ED DD94C241 6D5560AF7CA5
Round 7: DD94C241 8EAD3FD5 02765708B5BF
Round 8: 8EAD3FD5 59CBB833 84BB4473DCCC
Round 9: 59CBB833 7E1A0255 34F822F0C66D
Round 10: 7E1A0255 6243BF00 708AD2DDB3C0
Round 11: 6243BF00 1D75DEF7 C1948E87475E
Round 12: 1D75DEF7 E0F4AC68 69A629FEC913
Round 13: E0F4AC68 991A7D12 DA2D032B6EE3
Round 14: 991A7D12 4C67CA78 06EDA4ACF5B5
Round 15: 4C67CA78 A31ACCC9 4568581ABCCE
Round 16: 0AE50EE1 A31ACCC9 194CD072DE8C
After final permutation: 93E41C6E20911B9B
```

2. PFA on DES

We cannot apply the standard Persistent Fault Analysis technique to Feistel ciphers like DES as outputs of the previous round functions mask the left and right side of the outputs. Any Feistel round is a permutation over bit strings of length equal to the cipher's block size. It does not require the S-box to be bijective. Hence, the skewed distribution of a faulty S-box does not appear in the collected ciphertexts. If, however, we loosen the ciphertext-only requirement, persistent fault can be mounted [1].

2.1 Recovering Final Round Key

In DES we have 8 6x4 S Boxes which are pairwise different. In Persistent Fault Analysis (PFA), we inject fault one by one in each of the S-Boxes to recover different parts of the round key.

Consider DES whose substitution layer consists of 8 different 6x4 S-boxes S_1, S_2, …, S_8. Let the input for the last round of the DES is $x_l||x_r$ and output ciphertext be $y_l||y_r$.
We can further split these into 8 blocks of 4 bits each:

$$= |^{1},^{2},\ldots,^{8}| \qquad = |^{1},^{2},\ldots,^{8}|$$
$$= |^{1},^{2},\ldots,^{8}| \qquad = |^{1},^{2},\ldots,^{8}|$$

For illustration, consider that we alter an entry e of an S-box S_8 i.e. the last S-box. The faulty S-box is S'_8.
Let the ciphertext from a faultless device having S-box S_8 be y which can be split into y_l and y_r. Similarly, y'_l and y'_r be the ciphertext generated from the same plaintext on a faulty device having S-box S'_8 and that accesses the faulty element only in the last round.

$$' \quad = |_{1},_{2},\ldots,_{8}|$$

Here, each a_i is a block of 4 bits with $a_i = 0$ for $0 \leq i < 8$ and $_8 = {}_8(^8\ {}_8)\ '_8(^8\ {}_8)$
From the above equation, we can see that if the correct ciphertext is given, incorrect ciphertext that accessed the faulty entry e of S'_8 in only the last round can be found.
In other words, we can recover k_8 using $_8 = {}^8$.
K_8 gives us the last 6 bits of the last round key. To recover the remaining parts of the last round key, we'll add injections in other S-boxes in the same manner.

<u>Algorithm for DES PFA Key Recovery</u>

```
for i<-0; i<8; i<-i+1 do
     s'(i)(e) <- v
     while(true) do
     p <- random plaintext
     $y_l||y_r$ <- encrypt(p, s, key)
     $y_l'||y_r'$ <- encrypt(p, s', key)
     $|a_1,....,a_b|$ <- $y_l$ xor $y_l'$
     if ($a_j$ = 0, j in {1,.., b}\{i} ^ $a_i$ = v xor s(i)(e)) then
               $key_i$ = expand($y_r'$) xor e
               break
     endwhile
     s'(i)(e) <- s(i)(e)
endfor
```

Example:

Suppose we change `S-box 8`.

We set `S[0][0] = 0 (0000)` which was initially `13 (1101)`.

Now, when the input to `S-box 8` in the final round is `(000000)`, we get the output as `(0000)` instead of `(1101)`. Thus, for all remaining S-boxes, the 6x4 mapping is correct, while for S-box 8, if the 6-bit input is `(000000)`, the output is incorrect.

a. Ensure the fault has been accessed only in the final round

   If the faulty entry in S-box 8 is accessed in the final round, the output bits will match for all the 4-bit blocks output by the faultless S-boxes, while for the faulty S-box, the output blocks will be different and their xor will be equal to the xor of correct and faulty value in S-box 8. Therefore, if the xor of the faulty and the faultless cipher is the same as the xor of correct and incorrect value in the S-box, it implies that the fault has been accessed only in the last round. If the xor value is 0, it means the fault has not been accessed i.e. input to S-box 8 was not `000000`. If the xor value is neither 0 nor the xor of correct and faulty value in the S-box (`1101` in this case), it means the faulty element was accessed in some other round and got confused and diffused subsequently before reaching the final round.

b. Find the key from the fault

   The `input to the S-box` is given by `key xor expanded (xr)`. Thus,

   `000000 = key[42..48] xor expanded(`x_r`[42..48])`

   For the final round, $y_r = x_r$.

   `000000 = key[42..48] xor expanded(yr[42..48])`

   Thus, `key[42..48] = (000000) xor expanded(yr[42..48])`

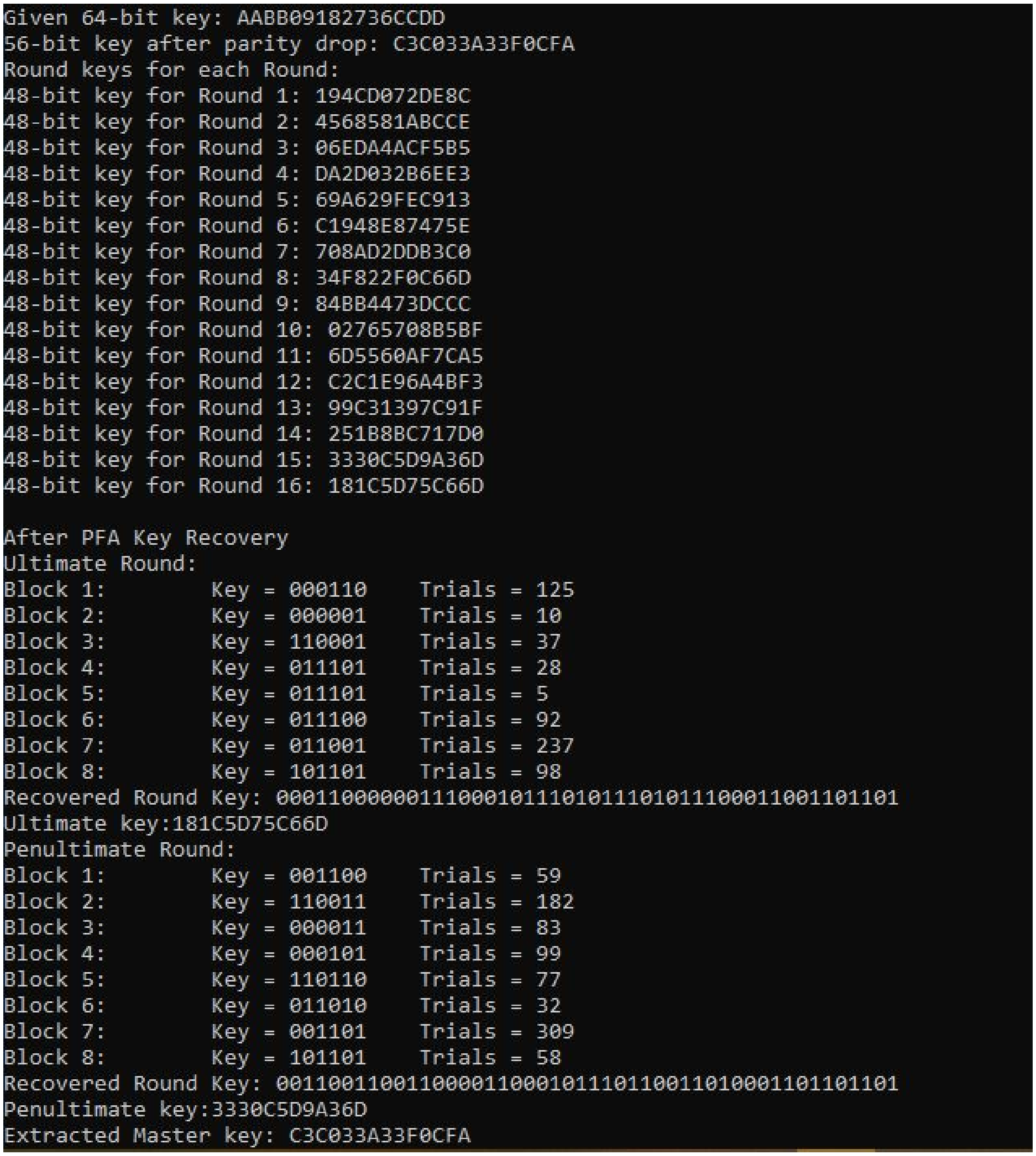
   Similarly, we can recover `key[0..42]`, by adding faults to each of the remaining 7 S-boxes one-by-one.

2.2 Recovering the Master Key

Once we recover the key for the last round, we use that key and the ciphertext generated to get the output of the penultimate i.e. 15th round of DES using decryption. Then, we apply the same key recovery algorithm for the 15th round to get the 15th round key.

Now, we have the keys for two consecutive rounds of DES i.e for the last two rounds. Using the key generation algorithm and compression box, we know the position of the missing bits in our last round key. Also, it is possible to generate the master key from two consecutive round keys as no bits are missing in the same positions for two round keys. We first expand the keys using the inverse of the compression box. Then, we circularly left-shift the right and left halves of the expanded penultimate key by one bit as done in the key generation algorithm and join the two halves. Finally, we OR the generated 56-bit string with the expanded last round key to recover the master key.

2.3 Output of PFA on DES



```
Given 64-bit key: AABB09182736CCDD
56-bit key after parity drop: C3C033A33F0CFA
Round keys for each Round:
48-bit key for Round 1: 194CD072DE8C
48-bit key for Round 2: 4568581ABCCE
48-bit key for Round 3: 06EDA4ACF5B5
48-bit key for Round 4: DA2D032B6EE3
48-bit key for Round 5: 69A629FEC913
48-bit key for Round 6: C1948E87475E
48-bit key for Round 7: 708AD2DDB3C0
48-bit key for Round 8: 34F822F0C66D
48-bit key for Round 9: 84BB4473DCCC
48-bit key for Round 10: 02765708B5BF
48-bit key for Round 11: 6D5560AF7CA5
48-bit key for Round 12: C2C1E96A4BF3
48-bit key for Round 13: 99C31397C91F
48-bit key for Round 14: 251B8BC717D0
48-bit key for Round 15: 3330C5D9A36D
48-bit key for Round 16: 181C5D75C66D

After PFA Key Recovery
Ultimate Round:
Block 1:        Key = 000110    Trials = 125
Block 2:        Key = 000001    Trials = 10
Block 3:        Key = 110001    Trials = 37
Block 4:        Key = 011101    Trials = 28
Block 5:        Key = 011101    Trials = 5
Block 6:        Key = 011100    Trials = 92
Block 7:        Key = 011001    Trials = 237
Block 8:        Key = 101101    Trials = 98
Recovered Round Key: 000110000001110001011101011101011100011001101101
Ultimate key:181C5D75C66D
Penultimate Round:
Block 1:        Key = 001100    Trials = 59
Block 2:        Key = 110011    Trials = 182
Block 3:        Key = 000011    Trials = 83
Block 4:        Key = 000101    Trials = 99
Block 5:        Key = 110110    Trials = 77
Block 6:        Key = 011010    Trials = 32
Block 7:        Key = 001101    Trials = 309
Block 8:        Key = 101101    Trials = 58
Recovered Round Key: 001100110011000011000101110110011010001101101101
Penultimate key:3330C5D9A36D
Extracted Master key: C3C033A33F0CFA
```

3. Analysis of the Attack

Following table shows the last round key bits recovered from every attack on a S-box. Each attack recovers 6 bits of the 48-bit last round key. For example, attack on the first S-box recovers the first 6 bits of the key. Hence, the residual key space will be 2^{42}-1. After all the bits are recovered, the residual key space will be zero.

Faulty S-Box	Recovered SubKey Bits	Number of SubKey Bits Left to Recover	Residual Key Space
S_1	000110	42	$2^{42}-1$
S_2	000001	36	$2^{36}-1$
S_3	110001	30	$2^{30}-1$
S_4	011101	24	$2^{24}-1$
S_5	011101	18	$2^{18}-1$
S_6	011100	12	$2^{12}-1$
S_7	011001	6	$2^{6}-1$
S_8	101101	0	$2^{0}-1 = 0$

The table below shows the number of trials required to get the SubKey bits for each attack on a S-box for recovering the ultimate and penultimate round keys:

Faulty S-Box	Number of Trials for Ultimate Round Key	Number of Trials for Penultimate Round Key	Average Number of Trials Required
S_1	125	59	92
S_2	10	182	96
S_3	37	83	60
S_4	28	99	63.5
S_5	5	77	41
S_6	92	32	62
S_7	237	309	273
S_8	98	58	78

The Persistent Fault mounted on the DES is performance invariant of the S-box mapping. The same can be seen from the output of the PFA code and the table above. We are using the key recovery algorithm twice to recover keys for the last and the second-last rounds of the DES respectively. We can see that even if the fault is injected in the same S-box, different numbers of trials are required to access the faulty element of the S-box in that round of the DES and there is a wide range of variation. For example, an injection of fault in the 2nd S-box requires only 10 trials to access the faulty element while recovering the last round key bits. However, the same fault requires 182 trials while recovering the key bits for the penultimate round. The number of trials required is dependent on the input plain texts to the DES which are generated at random. The performance of the attack is invariant to the S-box mapping.

4. Complexity Analysis

In a general case, let's assume b be the number of s boxes used in each round, r be the number of rounds in an encryption scheme, n be the number of bits given to each s box as input and m be the number of bits taken as an output from each s box.

We consider the case where $S_1, S_2, ..., S_b$ are pairwise different substitution boxes. The key observation is that in the presence of persistent faults some ciphertexts remain uncorrupted.

Suppose a single fault has been injected into one box. The probability that this element is accessed in an s box is $p = 1/2^n$. And the probability that this element is not accessed is $p', (1-1/2^n)$. Furthermore, the probability that this element is not accessed in each round lies at $p'^{(r)}$ that is $(1-1/2^n)^r$.

To obtain the last round partial key, the faulty element must be accessed only in the last round. So, the probability to obtain the last round partial key is $((p)*(p'^{(r-1)}))$ that is $(1/2^n)(1-1/2^n)^{(r-1)}$ The expected number of required ciphertexts pairs is given by the reciprocal $(2^n)(1-1/2^n)^{-(r-1)}$.

So, in the case of the data encryption standard(des), there are sixteen rounds and eight pairwise different 6×4 s-boxes in the substitution layer of each round. So,,
$b = 8 \quad r = 16 \quad n = 6 \quad m = 4$
The probability that a faulty element in one box is not accessed in all rounds stands at $(1-1/2^6)^{16} \approx 0.777$. Additionally, the probability that the faulty element is only accessed in the last round is given by $1/2^6(1-1/2^6)^{15} \approx 0.0123$. That is the probability that a partial key of 6 bit is obtained in one attack is 0.0123. And the expected number of required ciphertexts pairs is $2^6(1-1/2^6)^{-15} \approx 82.$

So, the average complexity in one attack is 82 encryptions. As we know we need 16 attacks to get the complete key without parity. So the overall complexity is `82*16 = 1312` encryptions.

5. References

[1] Caforio, Andrea and Subhadeep Banik. “A Study of Persistent Fault Analysis.” *IACR Cryptology ePrint Archive 2019* (2019): 1057.