Assignment – 8

Q1. Write a program to encrypt and decrypt the message using

the Substitution and permutation network.

A:

Pseudocode and Explanation –

1. **Key Scheduling (key_scheduling function):**
    - The key_scheduling function generates round keys by dividing the input key into smaller blocks.
    - The length of each subkey is determined by the plaintext length (l), and the number of blocks is specified.
    - These subkeys are used in different encryption and decryption rounds.
2. **Substitution Function (substitution_func and inverse_substitution_func):**
    - The substitution_func replaces each 4-bit block of the plaintext with a corresponding value according to a predefined substitution table (S-box).
    - The inverse_substitution_func reverses this process during decryption.
3. **Permutation Function (permutation_func):**
    - The permutation_func rearranges the bits of the input text based on the block structure. Each bit is moved to a different position, shuffling the input.
    - During decryption, the same function is used to reverse the permutation.

Encryption Process:

1. **Initialization:**

- The plaintext, number of blocks, and number of rounds are provided by the user.
- The key scheduling process splits the key into subkeys for each round.

2. **Rounds:**
   - For each round, except the last one, the following operations are performed:
     1. **XOR**: The plaintext is XORed with the current round key.
     2. **Substitution**: The resulting bits are substituted using the S-box.
     3. **Permutation**: The substituted bits are permuted (except for the second-last round).
3. **Final Round:**
   - In the last round, only XOR is performed between the plaintext and the final round key to generate the **cipher text**.

Decryption Process:

1. **Initialization:**
   - Decryption starts by using the cipher text obtained from encryption.
2. **Inverse Rounds:**
   - The decryption is essentially the reverse of encryption:
     1. **Inverse Permutation**: The cipher text undergoes inverse permutation (skipped for the last two rounds).
     2. **Inverse Substitution**: The result is then substituted back using the inverse S-box.
     3. **XOR**: The result is XORed with the corresponding round key to reverse the XOR operation.
3. **Final Result:**

- After all rounds are reversed, the **decrypted text** should match the original plaintext.

Code –

```
// SPN
// For m rounds,
// Apply XOR, substitution and permutation for all m-2 rounds
// for 2nd last round, only perform XOR and substitution
// in last round, perform XOR with the final subkey
#include<bits/stdc++.h>
using namespace std;
vector<string> key_scheduling(string key, int l, int block_size, int rounds){
    vector<string> keys;

    int j = 0;
    int i = 0;
    while(j<rounds){
        string s = key.substr(i,l);
        keys.push_back(s);
        j++;
        i+=block_size;
    }

    return keys;
}
string substitution_func(string pt){
    unordered_map<string,string> map;
    map["0000"] = "1110";
    map["0001"] = "0100";
    map["0010"] = "1101";
    map["0011"] = "0001";
    map["0100"] = "0010";
    map["0101"] = "1111";
    map["0110"] = "1011";
    map["0111"] = "1000";
    map["1000"] = "0011";
    map["1001"] = "1010";
    map["1010"] = "0110";
    map["1011"] = "1100";
    map["1100"] = "0101";
    map["1101"] = "1001";
    map["1110"] = "0000";
    map["1111"] = "0111";

    string ans;
    for(int i=0;i<pt.length();i+=4){
```

```
        string s = pt.substr(i,4);
        string q = map[s];
        ans+=q;
    }

    return ans;
}
string inverse_substitution_func(string pt){
    unordered_map<string,string> map;
    map["1110"] = "0000";
    map["0100"] = "0001";
    map["1101"] = "0010";
    map["0001"] = "0011";
    map["0010"] = "0100";
    map["1111"] = "0101";
    map["1011"] = "0110";
    map["1000"] = "0111";
    map["0011"] = "1000";
    map["1010"] = "1001";
    map["0110"] = "1010";
    map["1100"] = "1011";
    map["0101"] = "1100";
    map["1001"] = "1101";
    map["0000"] = "1110";
    map["0111"] = "1111";

    string ans;
    for(int i=0;i<pt.length();i+=4){
        string s = pt.substr(i,4);
        string q = map[s];
        ans+=q;
    }

    return ans;
}
string permutation_func(string pt, int num_of_blocks, int pt_l){

    string ans;

    for(int i=0;i<num_of_blocks;i++){
        for(int j=i;j<pt_l;j+=num_of_blocks){
            ans+=pt[j];
        }
    }

    return ans;
}
```

```
int main()
{
    string plain_text;
    cout<<"Enter the plain text: "<<endl;
    cin>>plain_text;

    int num_of_blocks;
    cout<<"Enter the number of blocks: "<<endl;
    cin>>num_of_blocks;

    int plain_text_length = plain_text.length();
    int block_size = plain_text_length/num_of_blocks;

    int rounds;
    cout<<"Enter the number of rounds: "<<endl;
    cin>>rounds;

    string key = "00111010100101001101011000111111";
    // cout<<"Enter the key: "<<endl;
    // cin>>key;

    vector<string> keys =
key_scheduling(key,plain_text_length,num_of_blocks,rounds);


    // Encryption
    for(int i=0;i<rounds-1;i++){
        string key_to_use = keys[i];
        string new_pt;

        // XOR
        for(int j=0;j<plain_text_length;j++){
            if(key_to_use[j] == plain_text[j]){
                new_pt+='0';
            }else{
                new_pt+='1';
            }
        }

        string after_subs = substitution_func(new_pt);

        string after_permut;

        if(i<(rounds-2)){
            after_permut = permutation_func(after_subs, num_of_blocks,
plain_text_length);
        }else{
            after_permut = after_subs;
        }
```

```
        plain_text = after_permut;
    }

    string cipher_text;
    string qw = keys[keys.size()-1];
    for(int j=0;j<plain_text_length;j++){
        if(qw[j] == plain_text[j]){
            cipher_text+='0';
        }else{
            cipher_text+='1';
        }
    }


    cout<<"Cipher Text: "<<cipher_text<<endl;



    // DECRYPTION
    string decrypted_text = cipher_text;
    cout << "Cipher Text: " << decrypted_text << endl;

    for(int i=rounds-1; i>=0; i--){
        string key_to_use = keys[i];
        string temp;


        if(i < (rounds - 2)){
            decrypted_text = permutation_func(decrypted_text, num_of_blocks,
plain_text_length);
            // cout << "After Inverse Permutation (Round " << i+1 << "): " <<
decrypted_text << endl;
        }


        if(i < (rounds-1)){
            decrypted_text = inverse_substitution_func(decrypted_text);
            // cout << "After Inverse Substitution (Round " << i+1 << "): " <<
decrypted_text << endl;
        }


        for(int j=0; j<decrypted_text.length(); j++){
            if(key_to_use[j] == decrypted_text[j]){
                temp += '0';
            } else {
                temp += '1';
            }
        }
```

```
        decrypted_text = temp;
    }

    cout<< "Decrypted Text: " << decrypted_text << endl;


}
```

Output –

```
cd "c:\Users\arinr\Desktop\Crypto_Lab\Lab_8\" ; if ($?) { g++ SPN.cpp -o SPN } ; if ($
?) { .\SPN }
Enter the plain text:
0010011010110111
Enter the number of blocks:
4
Enter the number of rounds:
5
Cipher Text: 1011110011010110
Cipher Text: 1011110011010110
Decrypted Text: 0010011010110111
```

Q2. Write a program to implement the Feistel Cipher.

A:

Pseudocode and Explanation –

1. **Feistel Function (apply_func)**:
   - This function performs a basic operation on the right half of the data.
   - It shifts a 3-bit chunk of the right half by rotating the last character to the front.
   - The function is simplistic and is meant to represent the round function in Feistel ciphers.
2. **Encryption Process**:
   - The plaintext is divided into two equal halves: left and right.
   - For each round:
     1. The right half is passed through the Feistel function.

2. The result is XORed with the left half to produce the new right half.
3. The halves are swapped.

- After completing the rounds, the final cipher text is the concatenation of the left and right halves.

3. **Decryption Process**:

- The cipher text is again divided into two halves: left1 and right1.
- Decryption follows the same process as encryption but in reverse:
    1. The left half is passed through the Feistel function.
    2. The result is XORed with the right half to recover the original left half.
    3. The halves are swapped.
- After completing all rounds, the final decrypted text is the concatenation of the halves.

.

Code –

```
// Feistel Cipher
#include<bits/stdc++.h>
using namespace std;
string apply_func(string r){
    string ans;
    for(int i=0;i<r.length();i+=3){
        char ch = r[i+2];
        string temp = r.substr(i,3);
        temp = ch+temp;
        temp.pop_back();
        ans+=temp;
    }

    return ans;
}
int main()
{
    string plain_text;
    cout<<"Enter the plain text: "<<endl;
    cin>>plain_text;
```

```
int rounds;
cout<<"Enter the number of rounds: "<<endl;
cin>>rounds;

int l = plain_text.length();

string left = plain_text.substr(0,l/2);
string right = plain_text.substr(l/2,l);

// ENCRYPTION
// Let function be right shift operation
for(int i=0;i<rounds;i++){
    string new_r = apply_func(right);
    string after_xor;
    for(int j=0;j<new_r.length();j++){
        if(left[j] == new_r[j]){
            after_xor+='0';
        }else{
            after_xor+='1';
        }
    }

    left = right;
    right = after_xor;

}
string cipher_text = left+right;
cout<<"Encrypted Text: "<<cipher_text<<endl;


// DECRYPTION
string left1 = cipher_text.substr(0,l/2);
string right1 = cipher_text.substr(l/2,l);
for(int i=0;i<rounds;i++){
    string new_l = apply_func(left1);
    string after_xor;
    for(int j=0;j<new_l.length();j++){
        if(right1[j] == new_l[j]){
            after_xor+='0';
        }else{
            after_xor+='1';
        }
    }

    right1 = left1;
    left1 = after_xor;

}
```

```
    string decrypted_text = left1+right1;
    cout<<"Decrypted Text: "<<decrypted_text<<endl;

}
```

Output –

```
PS C:\Users\arinr\Desktop\Crypto_Lab\Lab_8> cd "c:\Users\arinr\Desktop\Crypto_Lab\Lab_8\" ; if ($?) { g++ Feistel.cpp -o Feistel
 ; if ($?) { .\Feistel }
Enter the plain text:
011110100001
Enter the number of rounds:
1
Encrypted Text: 100001001010
Decrypted Text: 011110100001
```