NIVAAN KRISHUNDUTT
214551467

SECURITY AND ENCRYPTION FEISTEL PRACTICAL

INTRODUCTION

The aim of this practical was to implement the Feistel Cipher where the user will determine the number of rounds the Feistel Cipher will execute. The user will also decide whether the entered text must be encrypted or decrypted.

IMPLEMENTATION

The Feistel Cipher was chosen to be implemented using C++.

The user text is firstly entered by the user. It is then processed to remove all whitespaces between the text and if the length of the text is an odd number, a character 'X' is added at the end. This ensures that the text can be evenly split into two halves.

The user is then prompted to enter the number of rounds (which is stored in an integer variable *rounds*) and is also asked to enter a choice of whether the text should be encrypted or decrypted (if encryption is chosen, a Boolean variable *enc* will be true. False indicates decryption should be performed).

A 2-Dimensional array of integers (*keys[][]*) is dynamically created to store the keys for each round. The number of rows in the array is determined by the number of rounds chosen and the number of columns in the array is calculated as *0.5 * usertext.length() * 8* (since the key should be half the length of the usertext and multiply by 8 because the ASCII value of each character has 8 bits).

If Encryption is required, the keys are randomly generated using the *rand()* function.

If Decryption is required, the user has to enter the keys in text form. This will then be converted to their ASCII forms and stored into *keys[][]* bit by bit.

The usertext is then split into two halves, namely, *lusertxt* and *rusertxt*. These two strings are then converted to their ASCII forms and stored in integer arrays *leftASCII[]* and *rightASCII[]* respectively.

A temporary array of integers *newR[]* is used to temporarily store the new value of *rightASCII[]*. To perform encryption, the following was done:

newR[i] = (leftASCII[i] + keys[r][i]) % 2;	This performs an XOR with the left text from the previous round and the key for each specific round.
leftASCII = rightASCII;	The new leftASCII value takes on the value of the previous rightASCII value.
rightASCII = newR	The new rightASCII value was previously calculated and stored in newR.

After the required amount of rounds, the answer is then converted from its ASCII form to text form and the result is then displayed.

If an encryption process was performed, the Keys used are then displayed. This is done to ensure that the encrypted data can later be decrypted using these Keys.

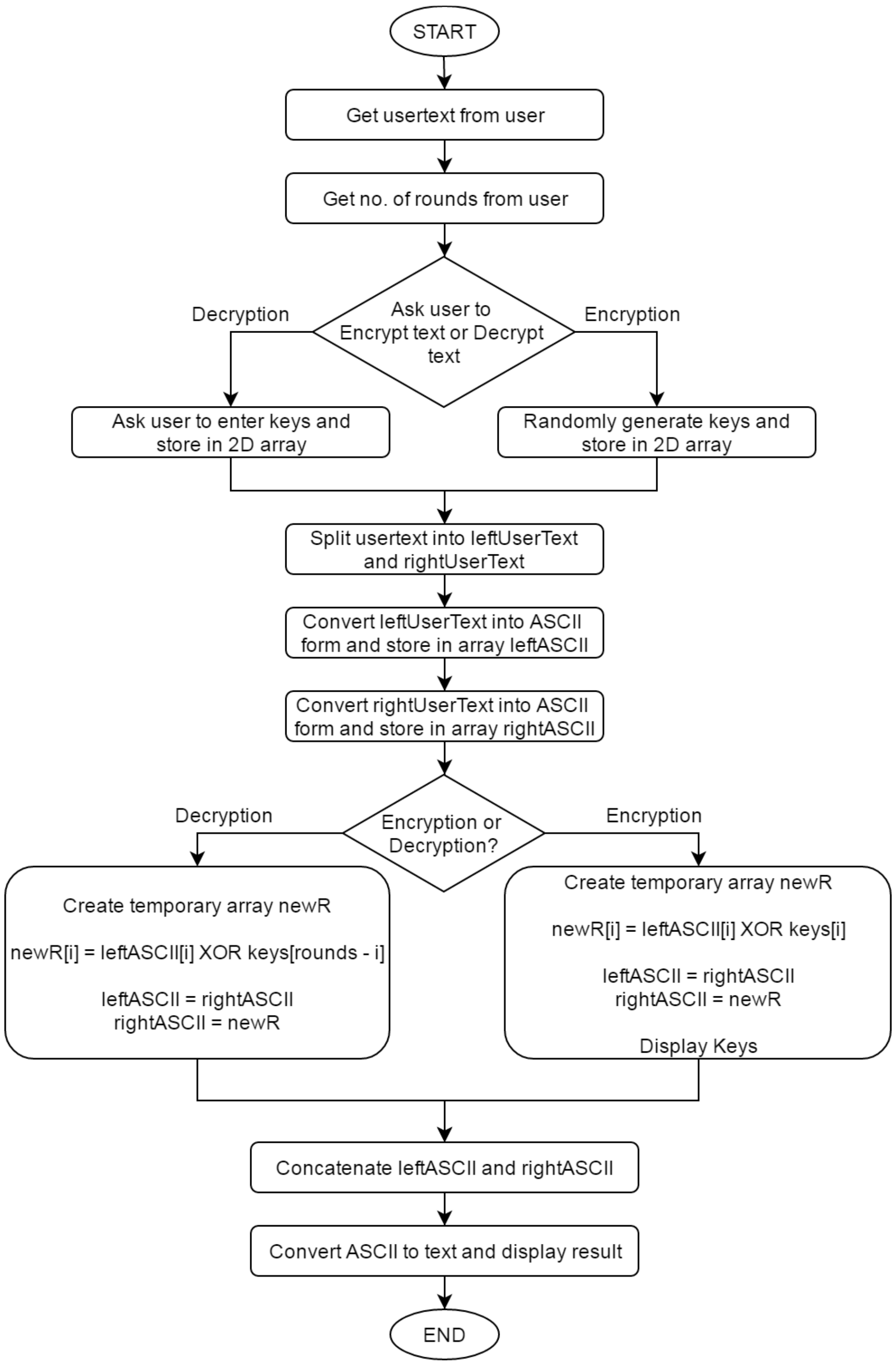
START
Get usertext from user
Get no. of rounds from user
Ask user to Encrypt text or Decrypt text
Decryption
Encryption
Ask user to enter keys and store in 2D array
Randomly generate keys and store in 2D array
Split usertext into leftUserText and rightUserText
Convert leftUserText into ASCII form and store in array leftASCII
Convert rightUserText into ASCII form and store in array rightASCII
Encryption or Decryption?
Decryption
Encryption
Create temporary array newR
newR[i] = leftASCII[i] XOR keys[rounds - i]
leftASCII = rightASCII
rightASCII = newR
Create temporary array newR
newR[i] = leftASCII[i] XOR keys[i]
leftASCII = rightASCII
rightASCII = newR
Display Keys
Concatenate leftASCII and rightASCII
Convert ASCII to text and display result
END

Figure 1: Flow chart of the program

CODE:

```
#include <stdio.h>
#include <iostream>
#include <string>
#include <math.h>
#include <time.h>
#include <bitset>

using namespace std;

void convertToArrayStream(int arr[], string txt)       //Function used to convert a
{                                                      //string into the ASCII version
                                                //Stores result in an array of integers
       string stream;
       for (int i = 0; i < txt.length(); i++)
              stream += std::bitset<8>((int)(txt.at(i))).to_string();
              //Converts string to ASCII stream of bits

       for (int i = 0; i < (txt.length() * 8); i++)
              arr[i] = stream[i] - '0';
              //Stores either a 1 or 0 in each element of the array
}

int binaryToBase10(int n)   //Function used to convert a binary number to base 10 number
{
       int ans = 0;
       for (int i = 0; n > 0; i++)
       {
              if (n % 10 == 1)
                     ans += (1 << i);
              n /= 10;
       }
       return ans;
}

string asciiToText(string txt)            //Converts ASCII to text and returns a string
{
       string ans;
       for (int j = 0; j < txt.length(); j = j + 8)
       {
              char x = binaryToBase10(atoi((txt.substr(j, 8)).c_str()));
              ans += x;
       }
       return ans;
}

int main()
{
       int rounds;
       string usertext = "";

       cout << "Please enter text : ";
       getline(cin, usertext);                         //Gets a sentence from the user

       for (int i = 0; usertext[i] != '\0'; i++)       //Removes whitespaces from text
              if (usertext[i] == ' ')
```

```
                for (int j = i; usertext[j] != '\0'; j++)
                        usertext[j] = usertext[j + 1];

        if ((usertext.length() % 2) != 0)  //Ensures length of text is an even number
                usertext += "X";

        cout << "Enter the number of rounds : ";
        cin >> rounds;                     //Gets number of rounds from user

        bool enc;
        cout << "\n\nWould you like to Encrypt the text or Decrypt it ? (1 = Encrypt, 0 =
Decrypt) ";
        cout << "\n** For Encryption, the Keys will be randomly generated.";
        cout << "\n** For Decryption, you will have to enter the keys manually in the form
of text.\n";
        cin >> enc;

        int **keys = new int*[rounds];     //Dynamically create 2D array to store keys
        for (int i = 0; i < rounds; i++)
                keys[i] = new int[usertext.length() * 4];

        if (enc)
        {
                srand(time(NULL));   //Randomly generates keys for encryption operation
                for (int i = 0; i < rounds; i++)
                        for (int j = 0; j < (usertext.length() * 4); j++)
                                keys[i][j] = rint(rand() % 2);
        }
        else
        {
                string inpt;
                for (int i = 0; i < rounds; i++)          //Gets keys from user as text
                {
                        cout << "Enter key used in round " << (i + 1) << " of Encryption :";
                        cin >> inpt;
                        convertToArrayStream(keys[i], inpt);
                        //Converts the text ASCII and stores in array
                }
        }

        //Splits user text into two halves
        string lusertxt = usertext.substr(0, (usertext.length() / 2));
        string rusertxt = usertext.substr(usertext.length() / 2);

        //Array used to store ASCII form of left half of the user text
        int *leftASCII = new int[lusertxt.length() * 8];
        //Array used to store ASCII form of right half of the user text
        int *rightASCII = new int[rusertxt.length() * 8];
        //Array used to store temporarily store the new value of the right half
        int *newR = new int[rusertxt.length() * 8];

        //Converts string to ASCII and stores in respective array
        convertToArrayStream(leftASCII, lusertxt);
        convertToArrayStream(rightASCII, rusertxt);
```

```
	for (int r = 0; r < rounds; r++)
	{
		if (enc)		//Performs Feistel Encryption
		{
			//Calculates the new right half of text
			for (int i = 0; i < (lusertxt.length() * 8); i++)
				newR[i] = (leftASCII[i] + keys[r][i]) % 2;
				//leftASCII[i] XOR keys[r][i]
		}
		else		//Performs Feistel Decryption
		{
			//Calculates the new right half of text
			for (int i = 0; i < (lusertxt.length() * 8); i++)
				newR[i] = (leftASCII[i] + keys[rounds - r - 1][i]) % 2;
				//leftASCII[i] XOR keys[r][i]
				//Keys are used in reverse order for Decryption
		}
		//leftASCII = rightASCII
		memcpy(leftASCII, rightASCII, (lusertxt.length() * 8 * sizeof(int)));
		//rightASCII = newR
		memcpy(rightASCII, newR, (rusertxt.length() * 8 * sizeof(int)));
	}

	string ans;

	//Converting the Result to one string variable
	for (int i = 0; i < (lusertxt.length() * 8); i++)
		ans += to_string(leftASCII[i]);
	for (int i = 0; i < (rusertxt.length() * 8); i++)
		ans += to_string(rightASCII[i]);

	//Converting the result from ASCII to text form
	string C = asciiToText(ans);

	if (enc){						//Displaying results
		cout << "\n\nCiphertext : " << C << "\n\nKeys: \n";

		string tmp = "";					//Displays Keys used
		for (int j = 0; j < rounds; j++)
		{
			for (int i = 0; i < (usertext.length() * 4); i++)
				tmp += to_string(keys[j][i]);
			cout << "Round " << (j + 1) << " : " << asciiToText(tmp) << endl;
			tmp = "";
		}
	}
	else{
		cout << "\n\nPlaintext : " << C << "\n";
	}	system("pause");
	return 0;
}
```

CONCLUSION

A notable challenge faced during the implementation of this Cipher technique was found when performing a decryption process, where the Keys of each round would have to be known. Due to the fact that the Keys for each round are generated randomly during encryption and are erased when the program terminates, this made it impossible to decrypt unless the Keys used during encryption were displayed to the user before termination.

Another challenge that was found was that in this program, input data is recorded as string, it is then converted to the American Standard Code for Information Interchange (ASCII) code of the data and each bit stored in an array of integers. That array is then processed (encrypted or decrypted) and thereafter converted from the ASCII code to plaintext (which is stored as a string). This amount of data type conversions could lead to a possible loss of data. Also, all this processing is time consuming and could result in a long processing time for a large number of rounds.

In contrast to the weaknesses and challenges faced, the Feistel Cipher has many advantages. One being that the Feistel Cipher adopts a Substitution-Permutation (SP) structure, which is largely advantageous as it incorporates confusion and diffusion systematically. In addition to this, the Keys for encryption in this program are randomly generated, which further increases the integrity and thus making this cipher almost impossible to break.

The decryption process is similar to the encryption process, with the only difference being that the keys are used in reverse order for the decryption process (i.e. K_n is used in the first round and K_1 used in the last round). Due to this fact, similar code was used for both processes with minor changes made. This leads to reduced number of line of code which makes the program more efficient; which is advantageous in many ways.