Implementation and Analysis of Text and Image Encryption-Decryption Algorithm

Assuring the security and confidentiality of sensitive information is crucial in the current digital era. Data security depends heavily on encryption technologies to prevent unauthorized access or interception. To offer reliable security, the approach combines symmetric key encryption and data transformation methods. In order to provide security in file sharing and chatting encryption and decryption technique is very useful. To secure the sensitive data, the algorithm combines symmetric key encryption and data transformation methods. The code illustrates how to encrypt and decode inputs that are both text and images. In this report there is the code flows detail, discusses implementation flaws, and explains the algorithm.

The overall implementation is for both text and image encryption and decryption algorithm. Text encryption and image encryption make up the two main components of the implemented code. The user is given the option of encrypting either text or an image.

1- The algorithm reads the user's input for text encryption and determines whether it is longer than the permitted 16 characters. The user is asked to enter a proper message if the input is lengthier. The program then generates a key depending on the user's input or one automatically if the input is incorrect. To make sure the key adheres to the necessary format, it is verified and updated. Based on the original key, the Round_Key function creates a series of round keys. Using the created round keys, the encryption function encrypts the text, and the decryption function decrypts it.
2- The code asks the user to choose an image file in order to encrypt images. The chosen image is reduced in size to 256x256 pixels and made grayscale. The image data is then changed into a format that can be encrypted. A key that must be entered by the user is then validated and changed in a manner akin to how text is encrypted. The Encryption function encrypts the image data using a set of round keys that were generated by the Round_Key function. In order to recover the original image, the decryption function turns back the encryption process.

Overall flow of code is given below:

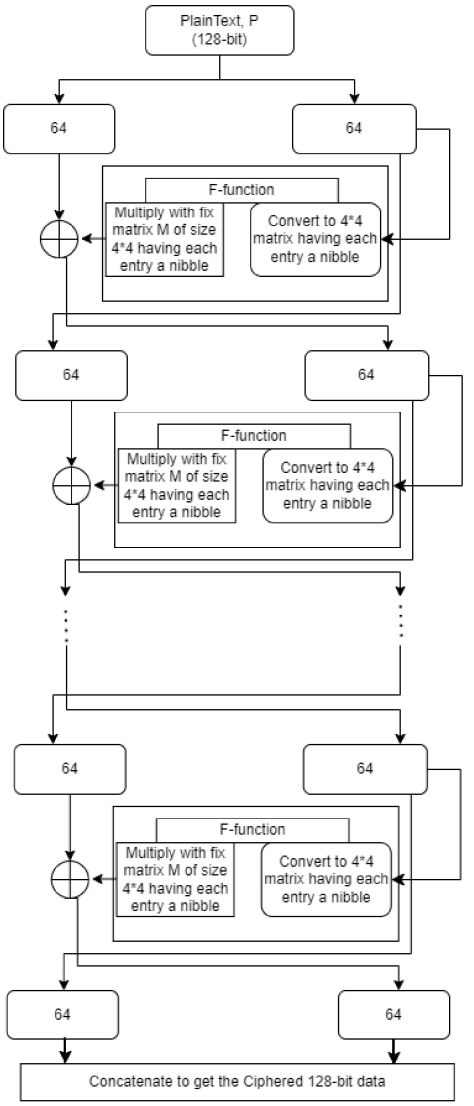
- Start from Repeat
- Ask the user whether they want "text" or "image" encryption
- When "text" is selected:
    - ask the user to input a text message
    - check to see if the message is longer than 16 characters
    - Ask the user to write a legitimate message.
    - Create or verify the encryption key
    - Encrypt the text
    - Decrypt the text
- Suppose "image" is picked:
    - A prompt asking the user to choose an image file
    - resizing and grayscale conversion
    - Create or verify an encryption key
    - Transform picture data for encryption
    - Encrypt and decrypt images, and perform image decryption.
- Request another encryption attempt from the user
- End if "no" is selected.

Feistel Cipher:

A symmetric encryption method that works with data blocks is called a Feistel cipher. Horst Feistel, a cryptographer who created this idea in the early 1970s, received credit for giving it its name. The well-known Data Encryption Standard (DES) and other block ciphers frequently employ the Feistel cipher scheme.

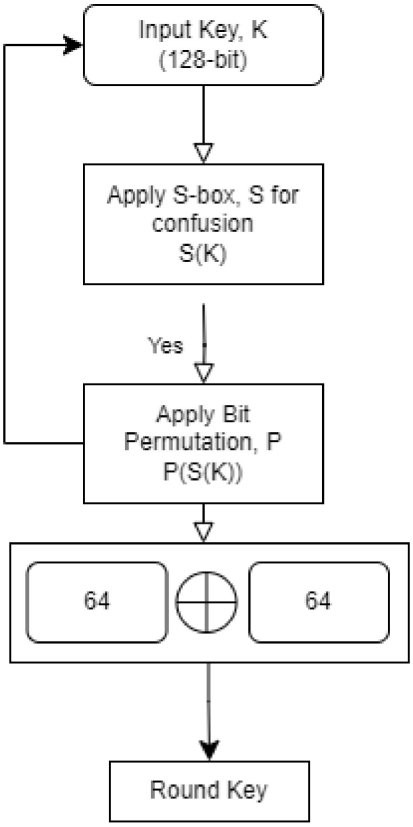
A Feistel cipher's basic operation entails splitting the input data block into two halves and carrying out a series of rounds, each of which manipulates one half of the data block based on the other half and a round key. After that, each round's output is swapped with the opposite half, and the process is repeated a predetermined number of times. The ciphertext is then created by combining the two halves.

Proposed Cipher:



FlowChart of the Proposed Fiestel Cipher

In this encryption above explained flowchart is implemented. Here is proposed key generation flow chart:



Key Schedule of the Proposed Fiestel Cipher

Overview of whole process:

Here is the step by step overview of implementation and explanation of function which is in the code:

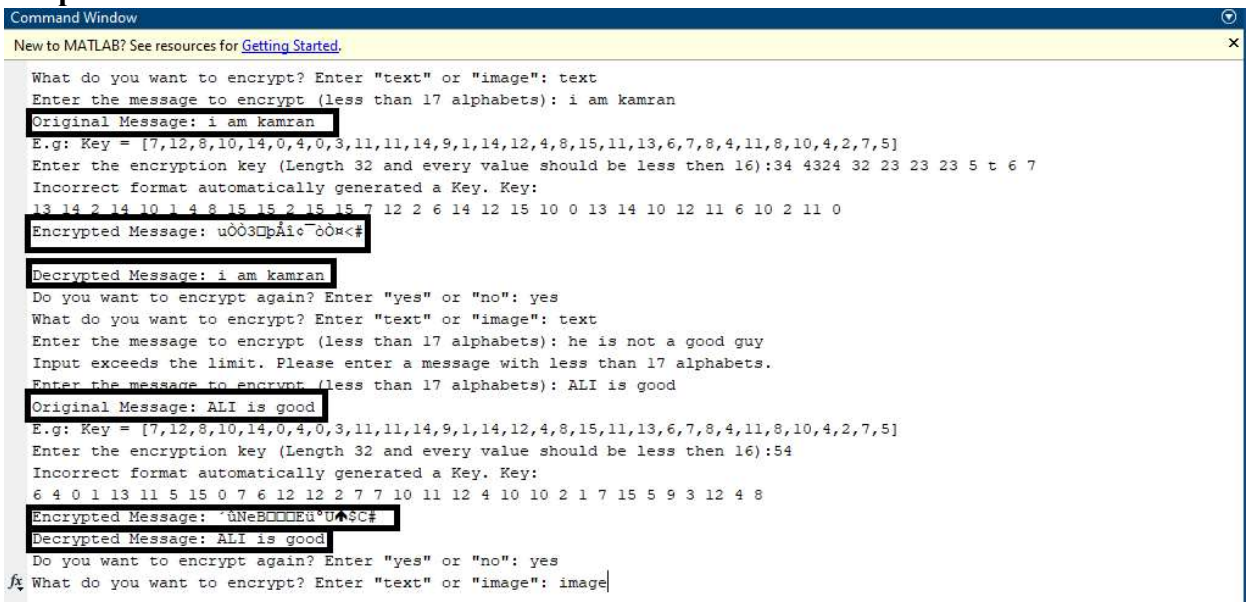
- **User Input:** Input is divided into left and right halves, respectively, the input plaintext block is divided into two equal parts known as the left half (L0) and the right half (R0).
- **Key Generation:** Using a longer encryption key, a key schedule algorithm creates a collection of round keys, each of which is unique to a particular round of encryption. The round keys are created from the original key and frequently go through different changes to increase security.
- **Round Function:** The right half (Ri-1) of the data block and the current round key (Ki) are both passed via a round function during each round of encryption. The purpose of the round function is to muddle and diffuse the encryption process. It often contains bitwise operations, permutation operations, and substitution boxes (S-boxes) for nonlinear substitutions.
- **XOR and Swap:** The output is XORed with the left half (Li-1) following the round function. The result is then switched with the right half (Ri-1) to become the left half (Li) for the following round. The prior right half (Ri-1) is still present.
- **Iteration:** For the desired number of rounds, steps 3 and 4 (round function and XOR & swap) are repeated. The specific Feistel cipher implementation's security needs will determine the number of rounds.

- **Output:** Once all rounds have been completed, the left half (Ln) and the right half (Rn) are produced. The ciphertext, or encrypted version of the original plaintext, is created by concatenating these two halves (Ln || Rn).

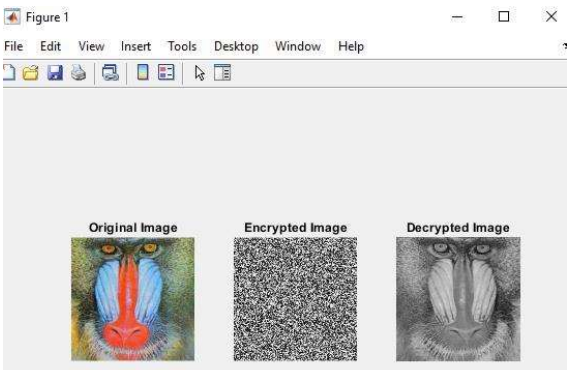
Limitations:

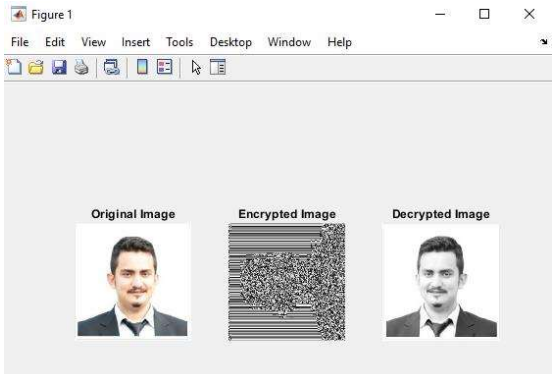
- Regarding the key generation procedure, the code contains restrictions. If the user's key input is incorrect, the code creates a key on its own. This automatic generation might not, however, always result in safe or random keys.
- It's possible that the encryption algorithm built into the code won't work for extremely sensitive or important data. It's critical to evaluate the security requirements and, if necessary, take various encryption methods into account that provide higher security.
- Errors or exceptions that can arise during the encryption procedure are not handled by the programming. The robustness and error handling of the code could be improved to increase its dependability.
- The code does not handle other formats and only supports a small number of image file types, including jpg, png, and bmp. It might be improved.

Output Screenshots:



Output for Image encryption and decryption:





Conclusion:

The findings and contributions of the applied encryption method for text and image data are summarized in the report's conclusion. There are suggestions for more enhancements, highlighting the significance of ongoing study and advancement in the area of data security.

Note: This code is implemented in MATLAB. The .m files are also attached with the report.

Code:

```
close all;
clear all;
clc
repeat = true;

while repeat
    choice = input('What do you want to
encrypt? Enter "text" or "image": ',
's');

    if strcmpi(choice, 'text')
        % Get input from the user
        Text = input ("Enter the
message to encrypt (less than 17
alphabets): ", 's');

        % Check if the input exceeds
the limit
        while numel(Text) > 16
            disp("Input exceeds the
limit. Please enter a message with less
than 17 alphabets.");
            Text = input("Enter the
message to encrypt (less than 17
alphabets): ", 's');
        end

        % Add spaces if the input
length is less than 16
        if numel(Text) < 16
            Text = [Text, repmat(' ',
1, 16 - numel(Text))];
        end

        disp("Original Message: " +
Text);
        disp('E.g: Key =
[7,12,8,10,14,0,4,0,3,11,11,14,9,1,14,1
2,4,8,15,11,13,6,7,8,4,11,8,10,4,2,7,5]
')
        kk = input('Enter the
encryption key (Length 32 and every
value should be less then 16):', 's');
        Key = str2num(kk);

        % Check if the length is less
than 32
        if length(Key) < 32
            % Generate random numbers
to fill the remaining length
            remainingLength = 32 -
length(Key);
            randomNumbers = randi([0,
15], 1, remainingLength);

            % Concatenate the random
numbers to the input array
            Key = [Key, randomNumbers];
        end
        disp("Incorrect format
automatically generated a Key. Key:");
        % Define the modulo value
        moduloValue = 16;

        % Check each index of the array
        for i = 1:length(Key)
            if Key(i) > 15
                Key(i) = mod(Key(i),
moduloValue);
            end
        end
        Text_Encryption(Text,Key);

    elseif strcmpi(choice, 'image')
        clear
        % Prompt the user to input the
image
        disp('Please select an image
file:');
        [fileName, filePath] =
uigetfile({'*.jpg;*.png;*.bmp'},
'Select Image File');
        imageFilePath =
fullfile(filePath, fileName);
        resized_image =
imread(imageFilePath);
        original_image =
imresize(resized_image, [256, 256]);
        %original_image =
imread(imageFilePath);
        Image =
rgb2gray(original_image);
        Image = Image';
        Image = Image(:);
        l = length(Image);

        Data = double(reshape(Image,
16, l/16)');

        disp('E.g: Key =
[7,12,8,10,14,0,4,0,3,11,11,14,9,1,14,1
2,4,8,15,11,13,6,7,8,4,11,8,10,4,2,7,5]
')
        kk = input('Enter the
encryption key (Length 32 and every
value should be less then 16):', 's');
        Key = str2num(kk);
```

```
        % Check if the length is less
than 32
        if length(Key) < 32
            % Generate random numbers
to fill the remaining length
            remainingLength = 32 -
length(Key);
            randomNumbers = randi([0,
15], 1, remainingLength);

            % Concatenate the random
numbers to the input array
            Key = [Key, randomNumbers];
        end
        disp("Incorrect format.
Automatically generated a Key. Key:");
        % Define the modulo value
        moduloValue = 16;

        % Check each index of the array
        for i = 1:length(Key)
            if Key(i) > 15
                Key(i) = mod(Key(i),
moduloValue);
            end
        end

        fprintf('%d ', Key);
        fprintf('\n');
        RoundKeys = Round_Key(Key);

        i = 1;
        while i < (l/16)+1
            Encryt(i,:) =
Encryption(Data(i,:),Key);
            i = i + 1;
        end

        Encrypt = Encryt';
        Encrypt = Encrypt(:);
        Encrypted_image =
uint8(reshape(Encrypt ,256,256));

        % Decryption
        Decrypted = [];
        j = 1;
        while j < (l/16)+1
            Decryt(j,:) =
Decryption(Encryt(j,:),Key);
            j = j + 1;
        end

        Decrypt = Decryt';
        Decrypt = Decrypt(:);
        Decrypted_image =
uint8(reshape(Decrypt, 256, 256));

        % Display images
        subplot(1, 3, 1);
        imshow(original_image);
        title('Original Image');

        subplot(1, 3, 2);
        imshow(Encrypted_image);
        title('Encrypted Image');

        subplot(1, 3, 3);
        originalImage =
Decrypted_image;

        % Get the dimensions of the
image
        [rows, columns, ~] =
size(originalImage);

        % Create a new matrix for the
rotated image
        rotatedImage = zeros(columns,
rows, 3, class(originalImage));

        % Perform the right 90-degree
rotation
        for row = 1:rows
            for col = 1:columns
                rotatedImage(col, rows
- row + 1, :) = originalImage(row, col,
:);
            end
        end

        % Display the rotated image
        imshow(rotatedImage);
        title('Decrypted Image');



    else
        disp('Invalid choice.');
    end

    % Ask if the user wants to repeat
    repeat_choice = input('Do you want
to encrypt again? Enter "yes" or "no":
', 's');
    if strcmpi(repeat_choice, 'no')
        repeat = false;
    end
end
function Output = Round_Key(input)
```

```
    i = 1;
    while i < 17
        input = Key_Scadulae(input);
        Round_Key(i,:) =
bitxor(input(1:16),input(17:32));
        i = i + 1;
    end
    Output = Round_Key;
end

function Output = Key_Scadulae(input)
    binaryInput = de2bi(input, 4,
'left-msb')';
    binaryInput = binaryInput(:)';
    Permutation =
[8,54,11,66,73,38,112,95,85,124,114,119
,92,33,45,115,93,34,17,98,104,103,55,10
0,27,49,32,36,67,80,41,39,60,106,58,123
,9,25,72,122,79,105,69,2,37,84,94,97,3,
86,75,102,10,64,23,12,28,71,121,126,52,
13,6,51,77,89,44,116,68,96,1,61,18,24,1
13,81,48,109,15,120,40,83,118,42,4,16,1
9,108,21,63,128,50,70,107,22,30,110,111
,127,31,101,14,46,91,90,99,43,117,87,74
,35,82,88,59,78,47,125,5,7,26,65,76,62,
57,56,29,20,53];
    permuted_data =
binaryInput(Permutation);
    foubits =
bi2de(reshape(permuted_data,4,32)','lef
t-msb')';
    sBox = [9 4 10 11;
            13 1 8 5;
            6 2 0 3;
            12 14 15 7];
    Output = sub_bytes(foubits ,sBox);
end

function Output = f_function(Data,
RoundK)
    Key_add = bitxor(Data, RoundK);
    sBox = [9 4 10 11;
            13 1 8 5;
            6 2 0 3;
            12 14 15 7];
    Substitute =
reshape(sub_bytes(Key_add, sBox), 4,
4);
    Mat = [9   10    4    9;
            2    4    6    4;
            15    7   13    8;
            2    5   15    8];
    Mat = gf(Mat, 4);
    Output = Mat * Substitute;
    Output = Output.x;
    Output = double(Output(:)');
end

function Encrypted = Encryption(Data,
Key)
    Data = de2bi(Data, 8);
    Data = Data';
    Data = Data(:)';
    input = bi2de(reshape(Data, 4,
32)')';
    RoundKeys = Round_Key(Key);
    LB = input(1:16);
    RB = input(17:32);
    disp('Please wait it will take few
moments.......')
    for i = 1:16
        disp('Please wait it will take
few moments')
        x = RB;
        RB = bitxor(f_function(RB,
RoundKeys(i, :)), LB);
        LB = x;
    end
    Encrypt4 = [LB, RB];
    X = de2bi(Encrypt4, 4)';
    X = X(:);
    X = bi2de(reshape(X, 8, 16)')';
    Encrypted = X;
end

function Decrypted = Decryption(Data,
Key)
    Data = de2bi(Data, 8);
    Data = Data';
    Data = Data(:)';
    input = bi2de(reshape(Data, 4,
32)')';
    RoundKeys = Round_Key(Key);
    LB = input(1:16);
    RB = input(17:32);
    for i = 16:-1:1
        x = LB;
        LB = bitxor(f_function(LB,
RoundKeys(i, :)), RB);
        RB = x;
    end
    Decrypt4 = [LB, RB];
    X = de2bi(Decrypt4, 4)';
    X = X(:);
    X = bi2de(reshape(X, 8, 16)')';
    Decrypted = X;
end
function bytes_out = sub_bytes
(bytes_in, s_box)
bytes_out = s_box (bytes_in + 1);
```

```
end
function Text_Encryption(Text,Key)


Data = double(char(Text));
Data = de2bi(Data, 8);
Data = Data';
Data = Data(:)';

input = bi2de(reshape(Data, 4, 32)')';

fprintf('%d ', Key);
fprintf('\n');
RoundKeys = Round_Key(Key);
% Encryption
LB = input(1:16);
RB = input(17:32);

for i = 1:16
    x = RB;
    RB = bitxor(f_function(RB,
RoundKeys(i, :)), LB);
    LB = x;
end

Encrypted4 = [LB, RB];

X = de2bi(Encrypted4, 4)';
X = X(:);
X = bi2de(reshape(X, 8, 16)')';

Encrypted = char(X);
disp("Encrypted Message: " +
Encrypted);

% Decryption
input = Encrypted4;

% Generate round keys in reverse order
RoundKeys = flipud(RoundKeys);

LB = input(1:16);
RB = input(17:32);

for i = 1:16
    x = LB;
    LB = bitxor(f_function(LB,
RoundKeys(i, :)), RB);
    RB = x;
end

Decrypted4 = [LB, RB];

X = de2bi(Decrypted4, 4)';
X = X(:);
X = bi2de(reshape(X, 8, 16)')';

Decrypted = char(X);
disp("Decrypted Message: " +
Decrypted);

function Output = Round_Key(input)
i = 1;
while i < 17
    input = Key_Scadulae(input);
    Round_Key(i,:) =
bitxor(input(1:16), input(17:32));
    i = i + 1;
end
Output = Round_Key;
end

function Output = f_function(Data,
RoundK)
Key_add = bitxor(Data, RoundK);
sBox = [9 4 10 11;
        13 1 8 5;
        6 2 0 3;
        12 14 15 7];
Substitute = reshape(sub_bytes(Key_add,
sBox), 4, 4);
Mat = [9   10    4    9;
        2    4    6    4;
       15    7   13    8;
        2    5   15    8];
Mat = gf(Mat, 4);
Output = Mat * Substitute;
Output = Output.x;
Output = double(Output(:)');
end
function Output = Key_Scadulae(input)
binaryInput = de2bi(input, 4, 'left-
msb')';
binaryInput = binaryInput(:)';
Permutation =
[8,54,11,66,73,38,112,95,85,124,114,119
,92,33,45,115,93,34,17,98,104,103,55,10
0,27,49,32,36,67,80,41,39,60,106,58,123
,9,25,72,122,79,105,69,2,37,84,94,97,3,
86,75,102,10,64,23,12,28,71,121,126,52,
13,6,51,77,89,44,116,68,96,1,61,18,24,1
13,81,48,109,15,120,40,83,118,42,4,16,1
9,108,21,63,128,50,70,107,22,30,110,111
,127,31,101,14,46,91,90,99,43,117,87,74
,35,82,88,59,78,47,125,5,7,26,65,76,62,
57,56,29,20,53];
permuted_data =
binaryInput(Permutation);
```

```
foubits =
bi2de(reshape(permuted_data,4,32)','lef
t-msb')';
    sBox = [9 4 10 11;
            13 1 8 5;
            6 2 0 3;
            12 14 15 7];
Output = sub_bytes(foubits ,sBox);
end
% Add the implementation of the
sub_bytes function here if available
function bytes_out = sub_bytes
(bytes_in, s_box)
bytes_out = s_box (bytes_in + 1);
end
end
```