Concepts of Parallel and Distributed Systems (CSCI-251)
Symmetric Encryption: Stream Ciphers

I. Goal

A stream cipher is one of many methods for performing symmetric encryption. In this project, we will focus on using the Linear Feedback Shift Registers (LFSR)[1] as a major component in a stream cipher to generate some pseudorandom sequence of bits known as a keystream. We want to also use this algorithm for the encryption and decryption of simple messages and images.

II. Overview

Write a command line application that encrypts and decrypts messages and images with a stream cipher that uses the LFSR key generator. This program will use .NET version 8 and C# programming language.

III. Stream Cipher-Linear Feedback Shift Registers

Recall from your lecture notes that a stream cipher is a symmetric key cipher that encrypts plain text one digit at a time.
A stream cipher with LFSR has the following operations:

- **Key Initialization**: The LFSR is initialized with an initial seed value (secret key)
- **LFSR keystream Generation[1]**: The LFSR creates a pseudorandom bit sequence (keystream) by shifting the bits to the left. The vacant position is then filled by an Exclusive OR (XOR) operation between the shifted-out bit and the bit from a specified tap position in the register. **Take a look at the example in the lecture slides and write your program based on the example.**
- **Encryption**: The keystream generated by the LFSR is combined with the plaintext using a bitwise XOR operation. Each bit of the plaintext is XORed with a corresponding bit from the pseudorandom sequence.
- **Decryption**: To decrypt the ciphertext, a similar keystream is generated using the LFSR initialized with the seed value and tap position. The keystream is XORed with the ciphertext to recover the original plaintext.

IV. Requirements

Your program must be a .net core command line program:

dotnet run <option> <other arguments>

For "option", your program will accept any of the following:

1. Cipher
2. GenerateKeystream
3. Encrypt
4. Decrypt
5. TripleBits
6. EncryptImage
7. DecryptImage

[1] Canteaut, A. (2011). Linear Feedback Shift Register. In: van Tilborg, H.C.A., Jajodia, S. (eds) Encyclopedia of Cryptography and Security. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-5906-5_357

Each of these options will accomplish a basic task, as detailed below (along with the extra command line options):

1. **Cipher** *seed tap*: This option takes the initial seed and a tap position from the user and simulates one step of the LFSR cipher. This returns and prints the new seed and the recent rightmost bit (which may be 0 or 1).

2. **GenerateKeystream** *seed tap step*: This option will accept a seed, tap position, and the number of steps (let's say n: a positive integer). For each step, the LFSR cipher simulation prints the new seed and the rightmost bit (see sample run). At the end of the iteration, save the keystream (a string consisting of "n" rightmost bits) in a file called "**keystream**" in the same directory where your program is.

3. **Encrypt** *plaintext:* This option will accept plaintext in bits; perform an XOR operation of the plaintext with the saved "keystream"; and return a set of encrypted bits (ciphertext).

4. **Decrypt** *ciphertext:* This option will accept ciphertext in bits; perform an XOR operation with the retrieved keystream from the file; and return a set of decrypted bits (plaintext).

5. **TripleBit** *seed tap step iteration*: This option will accept an initial *seed, tap*, *step* - a positive integer (let's say p) and perform 'p' steps of the LFSR cipher simulation. It will also accept *iteration*- a positive integer (let's say w). After each iteration *i (0 ≤ i <w)*, it returns a new seed, and accumulated integer value.

   At each step, it performs the cipher simulation; gets the recent rightmost bit and performs an arithmetic with this bit value.

   How to get the accumulated Integer Value for each Iteration

   At each iteration, this method **prints only the seed at the end of the last step and prints an accumulated integer value obtained after the arithmetic operation is done at the last step.**

   To get the accumulated integer value for each iteration: Initialize a variable to one and perform the following 'p' times (p= step):

   a. Multiply the value of the variable by 3;
   b. Add the recent rightmost bit returned by each step to the variable to get a new variable value.

   For instance, if p =5, and we have 1,1,1,1,1; each number being the rightmost bit value for all the 5 steps; the variable takes on the values 4, 13, 40, 121, and 364. The accumulated integer value is 364 for the first iteration in this instance.

6. **EncyrptImage** *imagefile seed tap:* Given an image with a seed and a tap position , generate a row-encrypted image. You should install a nugget package called **SkiaSharp.Views** and include the namespace: "SkiaSharp" in your program. **Do not use any other nugget packages or libraries for images.** You will need this namespace to convert the image found in the path to a bitmap image. You will use one of its classes - "SkiColor" to access each pixel in the bitmap image. https://learn.microsoft.com/en-us/dotnet/api/skiasharp.skcolor?view=skiasharp-2.88.

To encrypt an image, follow the steps below:

a. Generate a new-seed using the seed and tap. Do this using the cipher option.
b. Generate a row-encrypted bitmap image.

For each pixel in each row of the original bitmap, do the following for its red, blue and green color components:

- Use the new-seed to generate a random unsigned 8-bit integer
- Perform an XOR on the random 8-bit integer and the value of the color component of the pixel; the result of the XOR will be the new value to replace the color component.

Create a new color object using the new red, new green, and new blue components. Then, set the new color to the pixel.

c. Encode the row-encrypted bitmap image to an image file and save it in the same directory where your program is. Name the image: "File_NameENCRYPTED"

7. **DecyrptImage** *imagefile seed tap.* Given an encrypted image, a seed and tap position, generate the original image and save it with a different name in the same directory. Name the image: "File_NameNEW".
8. If the user specifies invalid user arguments for any of the options, your program should provide an appropriate error message indicating the problem.

V. Design

- The program must be command-line driven.
- The output (minus error messages) must match the write-up.
- Command line help must be provided. You should generate your help message.
- The program must be designed using object-oriented design principles as appropriate.
- The program must make use of reusable software components as appropriate.
- Each class and interface must include a comment describing the overall class or interface.

VI. Important Notes

- Make sure you include these two nugget packages in your program so that your program can run on any platform: "**SkiaSharp.NativeAssets.Linux"** for Linux and "**SkiaSharp.NativeAssets.macOS**" for Mac.
- Skiasharp works well with encoding and decoding most image file extensions.
- It is not required for you to include transparency (alpha) when creating a new SkColor object.
- Note that the shift of the bits is one position from right to left.
- After each LFSR simulation, the new rightmost bit is the pseudorandom bit for that step/iteration.
- Do not remove leading zeros when printing any output to the console.

VII. Grading

Your project will be graded over 100 points. I will grade your project on the following criteria:

- **Cipher** (10 points) - Generating a new seed and pseudorandom bit based on the LFSR algorithm.
- **GenerateKeystream** (5 points) - Generating each seed and a (new rightmost)pseudorandom bit after each simulation and a keystream (a sequence of pseudorandom bits).
- **Encrypt** (12 points) - generating a ciphertext in bits from a given plaintext (in bits) using the keystream in 3 test cases (see sample run). Each test case is 4 points:
  * Plaintext length less than the length of the keystream
  * Plaintext length greater than the length of the keystream
  * Plaintext length the same as the length of the keystream
- **Decrypt** (3 points) - generating a plaintext in bits from a given ciphertext (in bits) using the keystream with the above test cases. Each test case is 1 point
- **TripleBits** (20 points) – generating each seed at the end of the last step, and an accumulated integer value gotten from the arithmetic operation per iteration.
- **EncryptImage** (30 points)-generating an encrypted image from an image file
- **DecryptImage** (20 points)-generating the original image from an encrypted image

VIII. Submission Requirements

Zip up your solution in a file called project3.zip. The zip file should contain at least the following files, with the csproj being in the root of the zip.

- Program.cs
- Lfsr.csproj
- Text file if you used generative AI

I will be testing your program on either a Mac or Windows machine, with secondary tests being run on Linux. You should ensure your program works on multiple platforms.

IX. Sample Runs (for Output Formatting)

- **Cipher**:

```
dotnet run cipher 0110010010011110 8
0110010010011110 – seed
1100100100111101 1
```

- **GenerateKeystream***:*

```
dotnet run GenerateKeystream 0110010010011110 8 12
0110010010011110 – seed
1100100100111101 1
1001001001111011 1
0010010011110111 1
0100100111101111 1
1001001111011111 1
0010011110111110 0
0100111101111101 1
1001111011111010 0
0011110111110100 0
0111101111101001 1
1111011111010011 1
1110111110100110 0
The Keystream: 111110100110
```

- **Encrypt** *:* This includes the 3 test cases for encryption.

Case 1	dotnet run Encrypt 0100010000 The ciphertext is: 111010110110
Case 2	dotnet run Encrypt 011001001000101101 The ciphertext is: 011001110110001011
Case 3	dotnet run Encrypt 111010110110 The ciphertext is: 000100010000

- **Decrypt**: It can be generated using the same test cases. However, we will have "dotnet run Decrypt". Descriptive comments should be different from the above. e.g. "The ciphertext is " should be replaced with "The plaintext is".

- **TripleBits**: if p =5 and w = 12; we have:

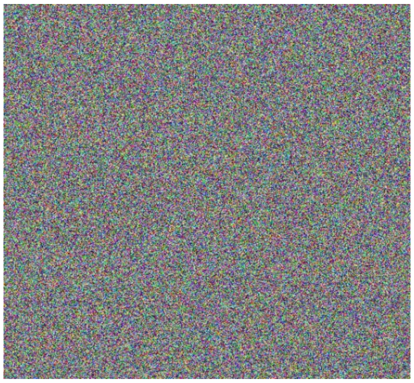
```
dotnet run TripleBits 0110010010011110 8 5 12
0110010010011110 – seed
1001001111011111  364
0111101111101001  271
0111110100110010  327
1010011001001001  271
1100100100111101  361
0010011110111110  363
1111011111010011  328
1111101001100100  252
0100110010010011  328
1001001001111011  355
0100111101111101  361
1110111110100110  255
```

- **EncryptImage:**

Although the aspect ratio in these results is not the same here due to “copy and paste” and the inclusion of text in each box; **Please make sure that the aspect ratio of all your images/results is the same.**

flower.png

```
dotnet run EncryptImage "flower.png" 01101000010100010000 15
```



flowerENCRYPTED.png

- **DecryptImage**:

```
dotnet run DecryptImage "flowerEncrypted.png" 01101000010100010000 15
```

flowerNEW.png