Encrypting Image using Stream Cipher A5/1

Author

Youssef Mahmoud Abdelkader

Mazen Khaled

Date

19/5/2024

Under The Supervision of:

Lamiaa Gaber

Table of Contents:

1. Introduction

Brief Description

This project demonstrates the encryption of images using the A5/1 stream cipher. The A5/1 algorithm, commonly used in GSM (Global System for Mobile communication) encryption, employs Linear Feedback Shift Registers (LFSRs) to generate a pseudorandom keystream which is then used to encrypt and decrypt image data.

Objective

The objective of this project is to illustrate the process of encrypting images using the A5/1 stream cipher, showcasing the security and efficiency of stream ciphers in cryptographic applications using Verilog Language to implement this algorithm.

---

2. Background

Stream Ciphers

Stream ciphers encrypt plaintext digits individually, often by combining them with a pseudorandom cipher digit stream. They are typically faster and have lower hardware complexity compared to block ciphers.

A5/1 Algorithm

The A5/1 algorithm is a stream cipher used for securing GSM (Global System for Mobile communication) cellular communications. It relies on three LFSRs to generate a keystream based on a secret key and a frame number, and we used this algorithm to generate a pseudorandom keystream which is then used to encrypt and decrypt image data.

LFSR (Linear Feedback Shift Register)

LFSRs are shift registers with feedback paths. They are used to generate pseudorandom sequences in stream ciphers. Each bit in the register is shifted to the next position, and the new bit is a linear function of the previous bits.

3. Project Overview

Scope:

This project covers the encryption of images using the A5/1 stream cipher. It includes key generation, keystream generation, and XOR-based encryption and decryption processes. The A5/1 algorithm is implemented using the Verilog hardware description language (HDL).

Workflow

1. Input an image.
2. Generate a keystream using the A5/1 algorithm implemented in **Verilog**.
3. XOR the image data with the keystream to encrypt the image.
4. Use the same keystream to decrypt the image, verifying the encryption process.

---

4. Prerequisites

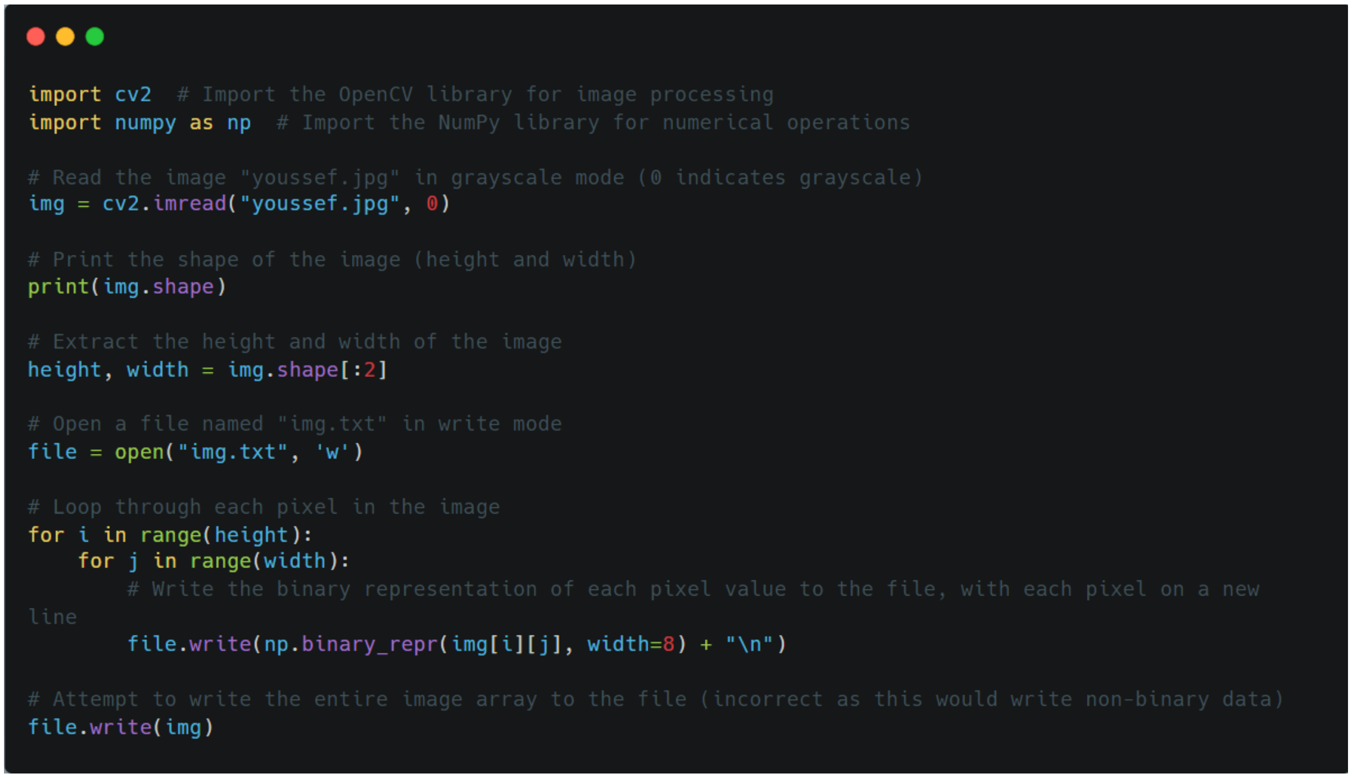
Software:

- Python 3.x
- Libraries: NumPy, OpenCV (cv2)
- Verilog simulator (Xilinx ISE USING Model Sim)

5. Usage

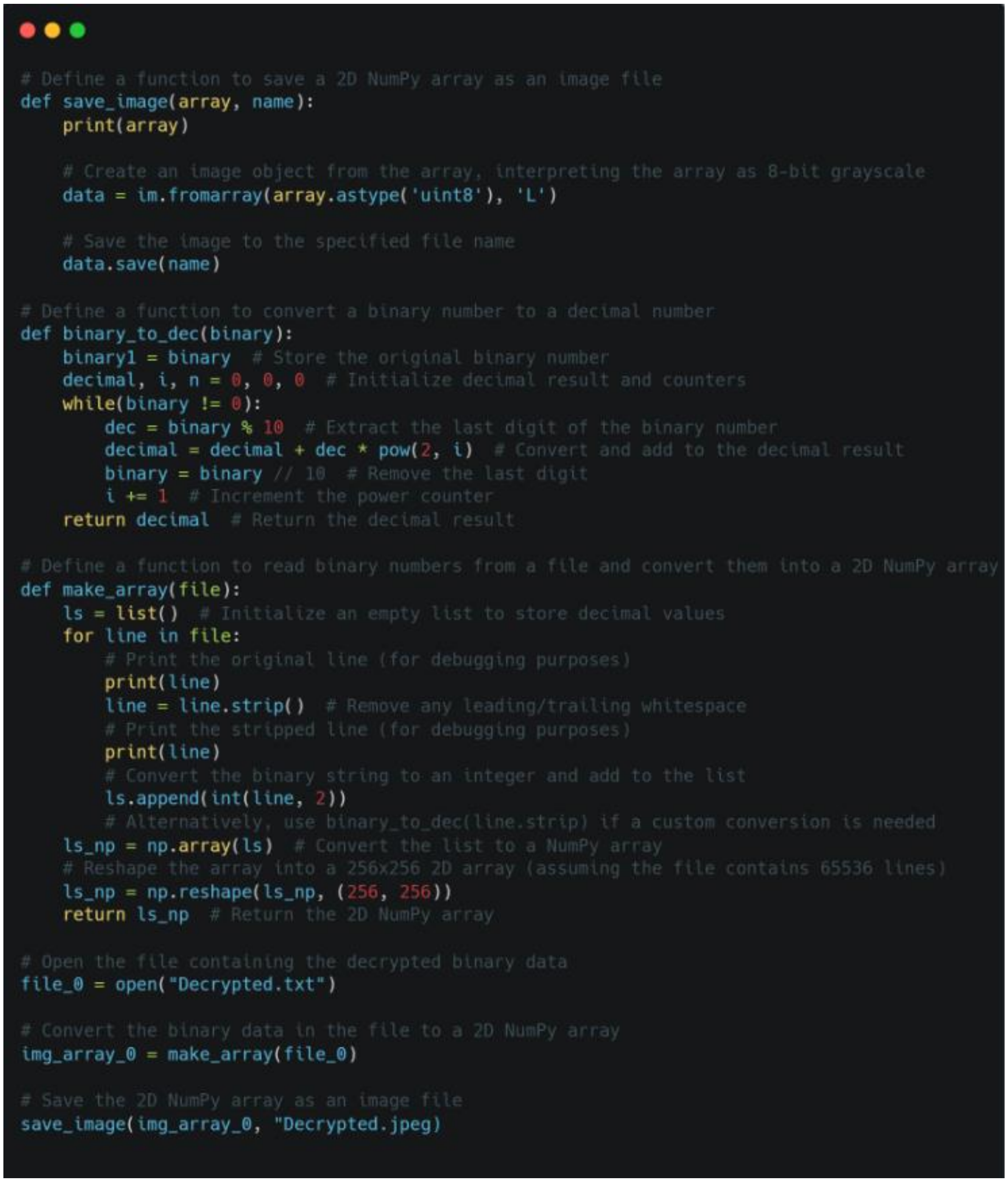
Python Codes

To encrypt an image using *Python:*



```python
import cv2  # Import the OpenCV library for image processing
import numpy as np  # Import the NumPy library for numerical operations

# Read the image "youssef.jpg" in grayscale mode (0 indicates grayscale)
img = cv2.imread("youssef.jpg", 0)

# Print the shape of the image (height and width)
print(img.shape)

# Extract the height and width of the image
height, width = img.shape[:2]

# Open a file named "img.txt" in write mode
file = open("img.txt", 'w')

# Loop through each pixel in the image
for i in range(height):
    for j in range(width):
        # Write the binary representation of each pixel value to the file, with each pixel on a new
line
        file.write(np.binary_repr(img[i][j], width=8) + "\n")

# Attempt to write the entire image array to the file (incorrect as this would write non-binary data)
file.write(img)
```

To encrypt an image using *Python:*



```python
# Define a function to save a 2D NumPy array as an image file
def save_image(array, name):
    print(array)

    # Create an image object from the array, interpreting the array as 8-bit grayscale
    data = im.fromarray(array.astype('uint8'), 'L')

    # Save the image to the specified file name
    data.save(name)

# Define a function to convert a binary number to a decimal number
def binary_to_dec(binary):
    binary1 = binary  # Store the original binary number
    decimal, i, n = 0, 0, 0  # Initialize decimal result and counters
    while(binary != 0):
        dec = binary % 10  # Extract the last digit of the binary number
        decimal = decimal + dec * pow(2, i)  # Convert and add to the decimal result
        binary = binary // 10  # Remove the last digit
        i += 1  # Increment the power counter
    return decimal  # Return the decimal result

# Define a function to read binary numbers from a file and convert them into a 2D NumPy array
def make_array(file):
    ls = list()  # Initialize an empty list to store decimal values
    for line in file:
        # Print the original line (for debugging purposes)
        print(line)
        line = line.strip()  # Remove any leading/trailing whitespace
        # Print the stripped line (for debugging purposes)
        print(line)
        # Convert the binary string to an integer and add to the list
        ls.append(int(line, 2))
        # Alternatively, use binary_to_dec(line.strip) if a custom conversion is needed
    ls_np = np.array(ls)  # Convert the list to a NumPy array
    # Reshape the array into a 256x256 2D array (assuming the file contains 65536 lines)
    ls_np = np.reshape(ls_np, (256, 256))
    return ls_np  # Return the 2D NumPy array

# Open the file containing the decrypted binary data
file_0 = open("Decrypted.txt")

# Convert the binary data in the file to a 2D NumPy array
img_array_0 = make_array(file_0)

# Save the 2D NumPy array as an image file
save_image(img_array_0, "Decrypted.jpeg)
```

Input/Output

- **Input Image:** An image file in formats like PNG or JPEG.
- **Output Image:** The encrypted/decrypted image file.

6. Algorithm Description

Detailed Steps：

A5/1 Stream Cipher

1. LFSRs Initialization:

- **LFSR1**: Length 19, feedback polynomial: x^19 + x^18 + x^17 + x^14 + 1
- **LFSR2**: Length 22, feedback polynomial: x^22 + x^21 + 1
- **LFSR3**: Length 23, feedback polynomial: x^23 + x^22 + x^21 + x^8 + 1

2. Majority Circuit (3-input):

The majority circuit is a critical part of the A5/1 algorithm. It determines which LFSRs to clock based on their current states. The majority function works as follows:

- **Inputs:** The middle bits of each LFSR (**LFSR1[8]**, **LFSR2[10]**, **LFSR3[10]**).
- **Function:** The majority value is the bit value (0 or 1) that appears in at least two of the three input bits.

Majority $(a, b, c) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$

This function ensures that at least two LFSRs will be clocked in each cycle, providing synchronization while allowing some variability.

3. Key Stream Generation:

- **Clocking LFSRs:** Each LFSR is clocked based on the majority circuit's output. If the LFSR's middle bit matches the majority bit, it is clocked.
- **Feedback Mechanism:** Each LFSR produces a feedback bit based on its feedback polynomial. The feedback bit is shifted into the LFSR, and the oldest bit is discarded.
- **Output Bit:** The output bit of the keystream is generated by **XORing** the output bits of all three LFSRs **(LFSR1[18] $\oplus$ LFSR2[21] $\oplus$ LFSR3[22])**.

4. Image Encryption:

- **Bit-wise XOR**: Each bit of the image is encrypted by XORing its value with the corresponding keystream bit. This process is repeated for all pixels, effectively transforming the original image into an encrypted image.

5. Image Decryption:

- **Regenerate Keystream**: To decrypt the image, the same keystream used for encryption is regenerated using the same secret key and frame number.
- **Bit-wise XOR**: The encrypted image is decrypted by XORing each bit value with the regenerated keystream bit, restoring the original image.

7. Implementation Details

Programming Language

The project is implemented using **Python** for image processing and **Verilog** for the A5/1 algorithm.

Code Structure:

- **Image to text.py:** Contains the Python code to create a .txt file which contains pixel values of an image in 8-bit binary format (Encryption).
- **txt to image.py:** Contains the Python code to file is used to create .jpeg file from the text file obtained after simulation (Decryption).
- **Main. v:** Contains the Verilog implementation of the A5/1 algorithm which have instantiations from the following modules:
    - **X-reg:** a LFSR with size of 19 bit
    - **Y-reg:** a LFSR with size of 22 bit
    - **Z-reg:** a LFSR with size of 23 bit
    - **Maj:** generate the enable signal for shifting the LFSR.
- **A5_1_testbench:** Contains the testbench for simulating the **A5/1** Verilog module. This testbench reads a .txt file generated by the **image_to_text.py** script, iterates over each bit, and XORs it with the encryption bit generated by the main A5/1 circuit. The output is written to another .txt file, which can be used as input for the **text_to_image.py** script to generate the **encrypted** image.

Key Functions:

- **Save-image:** function to save a 2D NumPy array as an image file (Python).
- **Make-array:** function to read binary numbers from a file and convert them into a 2D NumPy array.

8. Results

Examples:

- Original Image:

Path: Image_Encrypt\Image_Encrypt\python_codes\youssef.jpg

- Encrypted Image:

Path: Image_Encrypt\Image_Encrypt\python_codes\output.jpeg

- Decrypted Image:

Path: Image_Encrypt\Image_Encrypt\python_codes\Decrypted.jpeg

Performance

The implementation efficiently encrypts and decrypts images, demonstrating the effectiveness of the A5/1 stream cipher for this purpose.

---

9. Conclusion

Summary

This project successfully demonstrates the use of the A5/1 stream cipher for encrypting and decrypting images. The implementation showcases the simplicity and effectiveness of stream ciphers in cryptographic applications. Using Verilog for the A5/1 algorithm provides a hardware-accurate simulation, which can be extended to real FPGA implementations.

---

10. References

- A5/1: [Wikipedia] (https://en.wikipedia.org/wiki/A5/1)

- Linear Feedback Shift Register: [Wikipedia] (https://en.wikipedia.org/wiki/Linear-feedback_shift_register)