**Report on LFSR Implementation**

Mousa Mohamed Mousa **20235042**

Mohamed Tamer Younes **20235027**

**S2** Cryptography – IBCU Year 2

**1. How the LFSR Works**

A **Linear Feedback Shift Register (LFSR)** is a shift register where the input bit is a linear function of its previous state. It is commonly used to generate pseudorandom sequences. Here’s how it works:

* **Shift Register**: The LFSR consists of a series of bits (the seed) that shift one position to the left or right with each step.
* **Feedback**: The new bit that enters the register is calculated using a feedback function. This function XORs specific bits (called **tap positions**) from the current state.
* **Pseudorandom Sequence**: By repeatedly shifting and applying the feedback function, the LFSR generates a sequence of bits that appears random.

In our implementation:

* The **seed** is the initial state of the LFSR.
* The **tap positions** are specified by the user and determine which bits are used for feedback.
* The step() method generates the next bit in the sequence by XORing the tap positions and shifting the register.

For example, Taps are s0 and s1

1101

?110 -> 1

?111-> 0

**2. Choice of Polynomial Feedback**

The feedback polynomial is determined by the **tap positions** chosen by the user. These positions define which bits are XORed to produce the feedback bit. For example:

* If the tap positions are [0, 2], the feedback polynomial is x0+x2*x*0+*x*2.
* The choice of tap positions affects the randomness and period of the generated sequence.

In our implementation:

* The user specifies the tap positions (e.g., 0,2).
* The feedback bit is calculated by XORing the bits at the tap positions.

**3. Generated Sequence**

The LFSR generates a pseudorandom sequence of bits. In our implementation:

* The user provides an initial seed (e.g., 10110).
* The LFSR generates a sequence of at least **100 bits** using the generate() method.

Generated 100-bit sequence: 1001110100111010011101001110100111010011101001110100111010011101001110100111010011101001110100111010

**4. Example of Encryption and Decryption**

The LFSR is used to implement a simple **stream cipher** for encryption and decryption. Here’s how it works:

1. **Encryption**:
   * The plain text is converted to binary using the toBinary() method.
   * A keystream is generated using the LFSR.
   * The plaintext is XORed with the keystream to produce the ciphertext.

Example:

Plaintext: mousa

Remember the 100 pseudo generated bits? The main keystream is now generated to match the length of the plaintext, by iterating through each bit of the plaintext, the LFSR produces a keystream of the same size.

This keystream is then XORed with the plaintext to produce the encrypted text, ensuring that the encrypted output has the same number of bits as the plaintext. This process guarantees consistency and correctness in both encryption and decryption.

Keystream: 0011011101010000100101100111110001101110

Encrypted text (using keystream): 0101101000111111111000110000111100001111

**Decryption**:

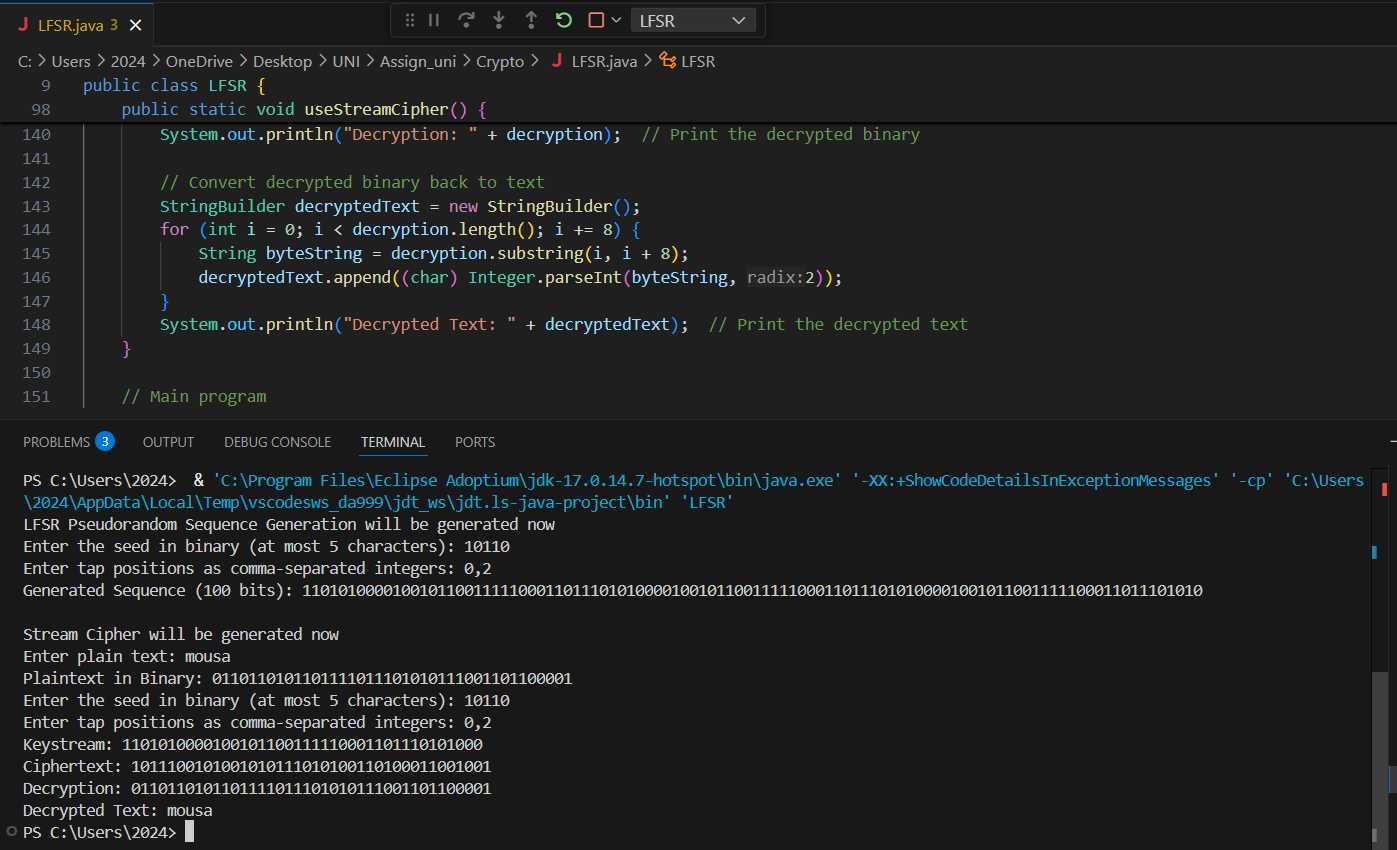
-The ciphertext is XORed with the same keystream to recover the plaintext.

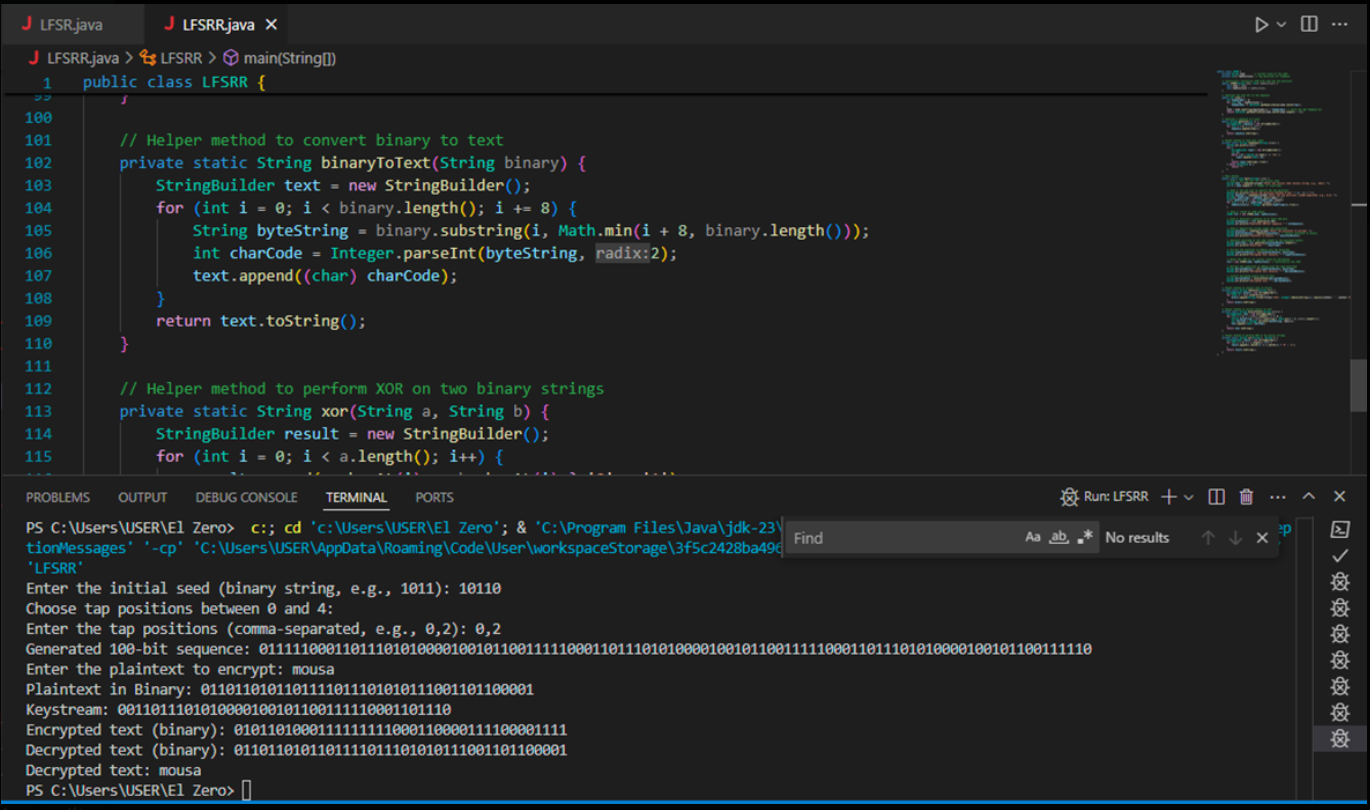
-The binary result is converted back to text using the binaryToText() method.

Decrypted text by XORing the encrypted text 00011000100000001011100011001000 with the emitted keystream as clarified previously will give us 0110110101101111011101010111001101100001, in which translated to a human form:

mousa

-> (successfully decrypted and got the right plain text before encryption)





**Conclusion**

The LFSR is a simple and effective way to generate pseudorandom sequences, which can be used for cryptographic applications like stream ciphers. By choosing appropriate tap positions and seeds, we can ensure the generated sequence is sufficiently random for encryption and decryption.