Simulation of Stream Ciphers using Linear Feedback Shift Registers

Theory of Automata & Formal Languages Semester Project

Group Members

Sr No.	Name	CMS
1	Muhammad Sunaam	393223
2	Abdul Basit	367949
3	Hafiz Ahmad Raza Khan	371502

Submitted To

Dr. Farzana Jabeen

Contents

Introduction

Linear Feedback Shift Registers (LFSRs) are fundamental components in digital systems, widely used for generating pseudorandom sequences in applications such as **cryptography**, **error detection**, and **data compression**. An LFSR operates by shifting bits in a register and using a feedback function, typically an XOR operation on specific tapped bits, to produce the next bit in the sequence. Despite their simplicity, LFSRs demonstrate powerful capabilities in streamlining computational processes and generating deterministic, yet seemingly random, outputs.

This project explores the simulation of a 4-bit LFSR using automata theory, a foundational area in computer science that provides formal models to represent computational processes. By leveraging a **Turing Machine**, the project models the iterative shifting and feedback operations of an LFSR. The choice of a Turing Machine is pivotal, as it allows the representation of both the sequential state transitions and the memory manipulation necessary to simulate the LFSR's behavior.

The objective of this project is to demonstrate how automata theory can be applied to real world systems, bridging the gap between theoretical computation and practical applications. The project involves designing a Turing Machine in **JFLAP**, implementing the LFSR logic, and generating a keystream for encryption and decryption. Through this simulation, the project not only showcases the computational power of automata but also highlights their relevance in modern cryptographic systems.

Problem Description

In modern cryptographic systems, stream ciphers play a critical role in securing data through encryption and decryption. The **A5/1 cipher**, widely used in GSM communication, is one such stream cipher that relies on **Linear Feedback Shift Registers (LFSRs)** to generate pseudo random keystreams for secure data transmission. An LFSR operates by sequentially shifting bits in a register and computing feedback through an XOR operation on specific tapped bits, resulting in deterministic, yet seemingly random, outputs. This project focuses on simulating the core mechanisms of the A5/1 cipher using automata theory principles.

To explore the theoretical foundations of cryptography and automata, we implemented the A5/1 cipher in Python for practical encryption and decryption tasks. Additionally, we modelled individual LFSRs—the building blocks of A5/1—using **Turing Machines in JFLAP**. The Turing Machine was specifically designed to handle the following operations:

1. **Bit Shifting:** Simulating the sequential update of LFSR registers.

2. **Feedback Calculation:** Using XOR operations on predefined taps to compute the feedback bit.

3**. Keystream Generation:** Producing pseudorandom sequences for encryption and decryption.

The use of a **Turing Machine** was crucial to effectively simulate the complex, parallel nature of LFSRs for taking **XOR** with the keystream and plain text. By leveraging multiple tapes:

- One tape contained the Pseudo Random key stream generated by the **LFSR**.
- The second tape contains the **plaintext** which is encrypted.

The problem posed two main challenges:

1. **Modelling Encryption and Decryption:** Demonstrating how LFSRs can generate a keystream to XOR with plaintext for encryption and with ciphertext for decryption.
2. **Simulating the LFSR Mechanism:** Accurately implementing the shifting and feedback logic within the constraints of a Turing Machine.

This project successfully bridges the gap between theoretical computation and real-world cryptography by simulating the A5/1 cipher's core mechanisms. The Python implementation and the Turing Machine simulation collectively illustrate the practical and theoretical aspects of encryption and decryption using LFSRs.

Theoretical Background

Automata theory provides a foundational framework for modelling computational processes, enabling the representation and analysis of algorithms, systems, and devices in a formalized way. Among its central models are the **Deterministic Finite Automata (DFA)**, **Pushdown Automata (PDA)**, and **Turing Machines (TM)**. Each model has unique capabilities and limitations, making them suitable for specific types of computational problems.

Automata Models

- **Deterministic Finite Automata (DFA):** A DFA is a state machine with no memory other than its current state. It is suitable for recognizing regular languages but lacks the capacity to perform complex operations requiring intermediate data storage or manipulation.
- **Push Down Automata (PDA):** A PDA extends the capabilities of a DFA by incorporating a stack, enabling it to process context free languages. While it introduces memory in the form of a stack, its operations are limited to LIFO (last in, first out) behavior, making it unsuitable for tasks involving random access or simultaneous operations.

- **Turing Machine (TM):** A TM is the most powerful model of computation, capable of simulating any algorithm. It operates on an infinite tape that acts as both input and memory, allowing read/write operations and bidirectional movement of the tape head. The Turing Machine further enhances this model by allowing parallel processing across multiple tapes.

For this project, the **Turing Machine** was chosen due to its ability to perform the sequential bit shifts and XOR operations required for simulating LFSRs.

Linear Feedback Shift Register (LFSR)

An **LFSR** is a digital circuit used to generate pseudorandom binary sequences. It consists of:

1. **Register:** A sequence of bits (states) that shifts with each iteration.
2. **Feedback Function:** A combination of specific bits (taps) from the register, typically XORed together, to produce a new bit.

The LFSR starts with an initial seed, and with each clock cycle:

- The bits shift to the right.
- A new bit is calculated using the feedback function and placed at the leftmost position.

LFSRs are widely used in cryptography (e.g., A5/1 cipher), error detection (e.g., CRC), and random number generation due to their efficiency and simplicity.

A5/1 Cipher

The A5/1 cipher is a stream cipher used in **GSM encryption**. It relies on three LFSRs of varying lengths:

- **LFSR 1:** 19 bits
- **LFSR 2:** 22 bits
- **LFSR 3:** 23 bits

The LFSRs operate in parallel, and their outputs are combined to produce a keystream. A majority voting mechanism determines which LFSRs should shift in each iteration.

XOR Operation

The exclusive OR (XOR) is a binary operation crucial in cryptography. It has the following properties:

Input A	Input B	Exclusive OR
0	0	0
1	0	1
0	1	1
1	1	0

Table 1: Exclusive OR

In this project, XOR is used in two ways:

1. As the feedback function in the LFSR to compute new bits.
2. For encryption and decryption, where the plaintext/ciphertext is XORed with the keystream.

Significance

The theoretical concepts described above form the backbone of this project. By implementing the A5/1 cipher in Python and simulating LFSRs using a Turing Machine, the project demonstrates how automata theory can be applied to cryptographic systems, bridging the gap between theoretical computation and real world applications.

Design & Implementation

This project focuses on simulating the **A5/1 cipher** using automata theory principles. It involves implementing the cipher in Python and simulating the core component, **Linear Feedback Shift Registers (LFSRs)**, using Turing Machines in **JFLAP**. This section describes the design considerations, implementation details, and the integration of these systems.

Design Considerations

The project design aims to address the following requirements:

- **LFSR Simulation:** Accurately simulate the operation of a 4-bit LFSR, including bit shifting, feedback computation, and keystream generation.
- **A5/1 Cipher Implementation:** Combine multiple LFSRs to generate a keystream for encrypting and decrypting messages.
- **Automata Simulation:** Leverage a Turing Machine to implement the LFSR operations within the constraints of automata theory.

The implementation consists of two parts:

- **Python Implementation:** A practical implementation of the A5/1 cipher to demonstrate encryption and decryption.
- **JFLAP Simulation:** A theoretical simulation of a single LFSR using a multi-tape Turing Machine to perform the necessary operations.

A5/1 Implementation

These are the files of project with their functionality:

- **main.py**: Entry point for the application.
- **src/analysis/statistics.py**: Tracks state transitions and their frequency during encryption.
- **src/analysis/visualizer.py**: Visualizes state transitions using a directed graph.
- **src/core/cipher.py**: Implements the core encryption algorithm.
- **src/core/registers.py**: Contains the Linear Feedback Shift Registers (LFSRs) used in the cipher.
- **src/gui/interface.py**: Provides a GUI interface for user interaction.

Module Description:

1. **main.py**
    - **Purpose:** Entry point of the application.
    - **Functionality:** Provides CLI options to initialize the A5/1 cipher, encrypt/decrypt messages, generate state transition graphs, or launch a GUI interface.
2. **src/analysis/statistics.py**
    - **Purpose:** Handles statistical data for state transitions.
    - **Functionality:** Records state transitions and calculates the frequency of each state.
3. **src/analysis/visualizer.py**
    - **Purpose:** Visualizes state transitions.
    - **Functionality:** Uses NetworkX to create a graph of state transitions and saves it as an image.
4. **src/core/cipher.py**
    - **Purpose:** Core implementation of the A5/1 stream cipher.
    - **Functionality:** Initializes the cipher, generates keystreams, encrypts/decrypts data, records state transitions, and visualizes state history.
5. **src/core/registers.py**
    - **Purpose:** Manages Linear Feedback Shift Registers (LFSRs) for the cipher.
    - **Functionality:** Implements the majority rule-based clocking mechanism, initializes LFSRs with key and frame, and generates output bits.
6. **src/gui/interface.py**
    - **Purpose:** Provides a GUI for the A5/1 cipher simulator.

- **Functionality:** Allows users to input keys, frames, and messages, encrypt/decrypt messages, and visualize state transition graphs interactively.

Interface:

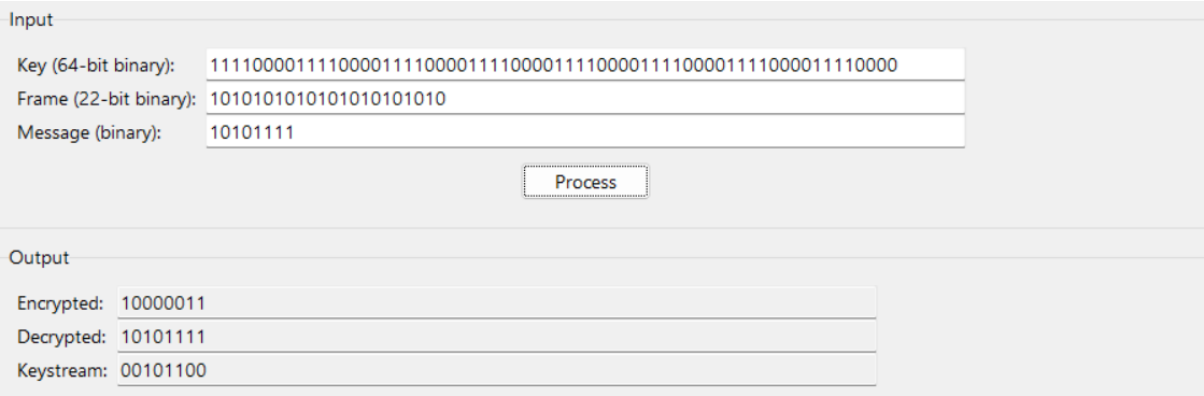

Figure 1: User Interface for the A5/1 Encryption

JFLAP Implementation

There are two parts to the JFLAP implementation. A **Standard Turing Machine** simulates the **4-bit Linear Feedback Register** while a **Multi-Tape Turing Machine** implements the XOR operations for encryption and decryption.

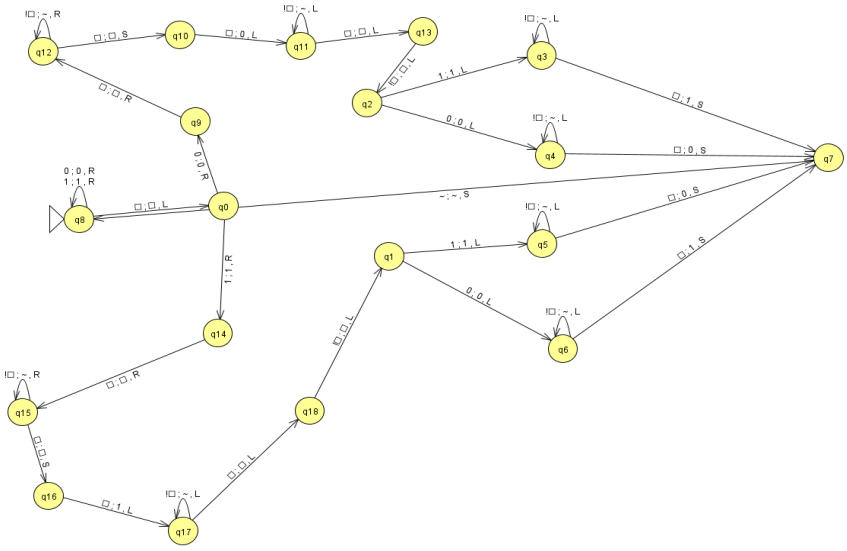

Figure 2: Turing Machine for 4-bit LFSR Implementation

Linear Feedback Shift Register

The LFSR's initial state (seed) is written on the Tape.

Shifting Bits

The Turing Machine:

- Read the bits on Tape 1.
- Shift the bits one position to the right.
- Writes a placeholder for the feedback bit in the leftmost position.

Computing Feedback

The machine moves to back and forth to:

- Read the tap positions.
- Performs the XOR operation on the corresponding bits from Taps.
- Writes the computed feedback bit to Tapes leftmost position.

Keystream Generation

The machine appends the rightmost shifted bit to the generated pseudo random sequence.

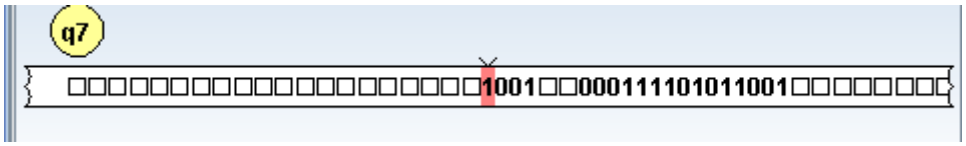

Figure 3: Output results of 4-bit LFSR

Iterative Process

The process repeats for n iterations, where n is the desired keystream length. The Turing Machine stops after completing n shifts.

Challenges Faced

Modelling the **LFSR** on a single taped Turing Machine was a monumental task that took the bulk of our time.

Results and Validation

The results of the project demonstrate the successful implementation of the **A5/1 cipher** using Python and the accurate simulation of **LFSRs in JFLAP** with a Turing Machine. This section outlines the outcomes, validation methods, and insights derived from the project.

Python Implementation Results

- **Encryption:** The plaintext is encrypted by taking the XOR of each plaintext bit with the corresponding bit of the keystream (generated by the A5/1 cipher). The

keystream is derived based on the initialized state of the LFSRs, which are synchronized using a session key and a frame number. This process ensures that the ciphertext is unique for each frame, even if the same session key is reused.

Formula: `Ciphertext = Plaintext ⊕ Keystream`

- **Decryption:** The ciphertext is decrypted in a similar manner by XORing the ciphertext with the same keystream used during encryption. This reverses the encryption operation, retrieving the original plaintext.

  Formula: `Plaintext = Ciphertext ⊕ Keystream`

- **Key Role:** The keystream acts as a one-time pad for each bit of data, ensuring that the security relies on the unpredictability and integrity of the keystream generation process. If the keystream is compromised, both encryption and decryption can be trivially reversed.

This XOR-based mechanism is efficient and forms the foundation of stream cipher encryption.

Figure 4: Results of the A5/1 Implementation

JFLAP Simulation Results

The Turing Machine in JFLAP successfully simulated the behavior of a 4-bit LFSR. The following results were observed:

- **Shifting Operations:** The Turing Machine correctly shifted the bits of the register with each iteration.
- **Feedback Computation:** XOR operations on the designated tap positions produced accurate feedback bits.
- **Keystream Output:** The Turing Machine generated a keystream consistent with the expected output for the given seed and feedback configuration.

Insights

- The Python implementation is efficient and scalable, making it suitable for practical cryptographic applications.
- The JFLAP simulation, while constrained by the manual design of the Turing Machine, effectively demonstrates the theoretical underpinnings of LFSRs and their integration into cryptographic systems.

- The Turing Machine simplifies the complex operations of an LFSR by delegating tasks to separate tapes, highlighting the power of automata theory in simulating computational systems.

Conclusion

This project successfully combines the theoretical foundations of automata theory with practical cryptographic applications by implementing the A5/1 cipher and simulating Linear Feedback Shift Registers (LFSRs) using Turing Machines. It demonstrates how concepts from computational models can be utilized in real-world encryption systems while providing insights into their limitations and capabilities.

Key Achievements

Python Implementation:

- The A5/1 cipher was implemented and validated through encryption and decryption processes.
- The generated keystream was pseudo-random, and its integration into XOR-based encryption ensured secure transformation of plaintext into ciphertext.
- Decryption confirmed the integrity of the ciphertext-to-plaintext process.

Turing Machine Simulation:

- LFSRs were accurately simulated using a multi-tape Turing Machine in JFLAP.
- Complex operations such as bit-shifting, feedback computation, and keystream generation were modelled effectively within the constraints of automata theory.
- The simulation highlighted the theoretical feasibility of cryptographic systems within the Turing Machine framework.

This project successfully bridges the gap between theoretical computation models and practical cryptographic systems. By implementing the A5/1 cipher in Python and simulating LFSRs in JFLAP using Turing Machines, it underscores the versatility of automata theory and its relevance to modern computer science applications.