Fall 2024 2024-11-06

Project Progress Report

Indronil Bhattacharjee *CS 380/525: Intro to Cryptography*

Cryptography Algorithms Implementation: RSA and ElGamal

1 Overview

This project involves the implementation of two fundamental public-key cryptography algorithms: RSA and ElGamal. The implementation is carried out in C++, utilizing the Number Theory Library (NTL) to handle large integers and modular arithmetic, which are crucial for the encryption and decryption processes. The project aims to provide a practical understanding of cryptographic principles, encryption efficiency, and secure data handling.

2 Project Objectives

- **Implement RSA and ElGamal algorithms** to understand their operations involving key generation, encryption, and decryption.
- **Measure and compare the performance** of both algorithms in terms of execution times for encryption and decryption.
- **Securely manage data** by reading plaintext from and writing ciphertext to files, ensuring data integrity throughout the process.

3 Environment Setup

3.1 Development Tools

- **Compiler**: GNU GCC Compiler
- **IDE**: Any preferred IDE that supports C++ (e.g., Visual Studio Code, CLion, or simply a text editor with a command line terminal)
- **Libraries**: NTL (Number Theory Library) for handling large number operations, and GMP (GNU Multiple Precision Arithmetic Library) as it is often used by NTL for better performance.

3.2 Library Installation

- **NTL**: Download and install from NTL's official website. Configure the library with GMP support to enhance performance.
- **GMP**: Install using your system's package manager or compile from source available on the GNU website.

4 Implementation Details

4.1 File Structure

- **RSA.h / RSA.cpp**: These files contain the class and methods for RSA algorithm operations.
- **ElGamal.h / ElGamal.cpp**: These files handle the ElGamal algorithm functionalities.
- **FileIO.h / FileIO.cpp**: Responsible for file operations, including reading input messages and writing encrypted or decrypted messages.
- **main.cpp**: Drives the program, orchestrating the encryption and decryption processes, handling file operations, and timing the encryption and decryption operations for performance analysis.

4.2 Key Features

- **Modular Design:** Each cryptographic algorithm is encapsulated in its own class, allowing for easy expansion or modification.
- **File-Based Input/Output:** To simulate real-world applications, all messages are read from and written to files, separating the data handling from the core cryptographic operations.
- **Performance Measurement:** Execution times for each operation (encryption and decryption) are measured and output to the console, providing insights into the efficiency of the implemented algorithms.

5 Usage

- **Input**: The program expects a plaintext file named input.txt for the encryption process. This file should be prepared and placed in the appropriate directory before running the program.
- **Output**: Encrypted and decrypted outputs are written to separate files, named according to the encryption method used (e.g., encrypted_rsa.txt, decrypted_rsa.txt).
- **Execution**: Run the compiled executable in a terminal or command prompt. The program will read the input, perform encryption and decryption, and then write the results to the output files while displaying timing information in the console.

6 Conclusion

This project serves as a practical exploration of RSA and ElGamal encryption algorithms, emphasizing the use of libraries for handling complex arithmetic and the importance of performance in cryptographic implementations. Future enhancements could include implementing additional algorithms, integrating more robust error handling, and expanding the user interface for easier interaction and configuration.