TRIBHUWAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Proposal

On

"FILE ENCRYPTION AND DECRYPTION"

Submitted by:

Amrita Regmi (THA081BEI005)

Arjun Paudel (THA081BEI007)

Shirish Dhami (THA081BEI040)

Swastika Kharel (THA081BEI047)

Submitted to:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

February 2025

ABSTRACT

Data security represents a significant challenge in the contemporary digital landscape, where sensitive data is routinely shared and stored. This initiative aims to develop a file encryption and decryption system utilizing the C programming language to safeguard data confidentiality and integrity. The system incorporates cryptographic methods to transform plain text files into an encrypted state, rendering them inaccessible without the appropriate decryption key. The encryption phase employs algorithms such as AES (Advanced Encryption Standard) or XOR-based encryption, while the decryption phase reverses this process to retrieve the original file. The implementation encompasses key generation, secure storage solutions, and efficient encryption and decryption functions. The anticipated result is a lightweight, rapid, and secure tool designed to protect files from unauthorized access. This project plays a vital role in bolstering data security for both personal and organizational applications, showcasing the practical use of cryptographic principles within C programming.

Key words:Data security,encryption and decryption,cryptographic algorithms

Table of Contents

List of Figures

List of Abbreviations

· **AES** – Advanced Encryption Standard

· **RSA** – Rivest-Shamir-Adleman

· **XOR** – Exclusive OR

· **SPN** – Substitution-Permutation Network

· **SSL/TLS** – Secure Sockets Layer / Transport Layer Security

· **CLI** – Command-Line Interface

· **GUI** – Graphical User Interface

· **GCC** – GNU Compiler Collection

· **OS** – Operating System

· **DES** – Data Encryption Standard

1. INTRODUCTION

1.1 BACKGROUND INTRODUCTION

The exponential increase in digital data necessitates robust security measures. Sensitive data, including personal, financial, and corporate information, is routinely stored and transmitted electronically. This prevalence exposes such data to significant risks from cyber threats like unauthorized access, breaches, and hacking. Encryption, a common technique for transforming readable data into an unreadable format decipherable only with a specific key, offers a solution by restricting access to authorized users and preventing data manipulation. Algorithms like AES, RSA, and XOR provide varying levels of security and performance. This project develops a lightweight and efficient file encryption and decryption system in C as a means of bolstering data security and mitigating cyber threats. It allows users to secure files so decryption and access requires the correct key and will offer a practical application.

1.2 MOTIVATION

Completing this project provides a strong foundation in cryptography and secure coding practices. It enhances problem-solving skills, deepens knowledge of encryption algorithms, and opens doors to cybersecurity careers, research, and advanced encryption projects. This experience can lead to opportunities in ethical hacking, secure software development, and open-source contributions. Additionally, it strengthens technical skills in C programming, file handling, and algorithm implementation, making it valuable for both academic and professional growth. Ultimately, this project serves as a stepping stone for further exploration in cybersecurity, advanced encryption techniques, and real-world security applications.

1.3 PROBLEM DEFINITION

This project aims to develop a file encryption and decryption system using the C programming language, ensuring data confidentiality and integrity.

1.4 OBJECTIVES

The primary objectives of this project are:

- To implement a secure and efficient file encryption and decryption system using C.
- To use cryptographic techniques such as AES (Advanced Encryption Standard), XOR-based encryption, or RSA for securing files.
- To ensure user-friendly interaction, allowing encryption and decryption through a simple command-line interface.

2. LITERATURE REVIEW

Encryption and Decryption are the fundamental aspects of cyber security, ensuring data confidentiality and integrity. Various encryption techniques have been implemented in different programming languages including C, which provides low-level control over data manipulation and file handling. The research and development of encryption algorithms have significantly contributed to securing sensitive information in diverse fields, including banking, communications and digital forensics.

Existing works on Encryption and Decryption

Several encryption algorithms and methodologies have been developed to provide secure data transmission and storage. The following sections discuss some of the prominent encryption techniques and their implementations.

2.1 Caesar Cipher Implementation in C

The Caesar Cipher is one of the earliest encryption techniques, relying on shifting characters in the plain text by a fixed number of positions in the alphabet.

It is implemented in C by reading a file character by character, shifting each letter forward (for encryption) or backward (for decryption), and writing the modified text into another file. This approach utilizes basic ASCII operations.

Importance & Applications

- Useful for learning fundamental cryptographic principles.
- Simple to implement and understand.
- Used is basic encoding applications.

Drawbacks & Limitations

- Weak security due to predictability.

- Easily breakable using frequency analysis.
- Not suitable for modern security applications.

Criticism & Link to Motivation

Caesar Cipher lacks complexity and is highly vulnerable to brute-force attacks. This limitation highlights the need for stronger encryption techniques in modern applications.

2.2 Advanced Encryption Standard(AES) in C

AES is a widely used symmetric encryption standard that provides high security and is adopted by the U.S. government for secure communications.

AES in C is implemented using a substitution-permutation network (SPN) that processes data in fixed-size blocks (128-bit). The algorithm involves key expansion, initial rounds of substitution, shift rows, mix columns, and final key addition.

Importance & Applications

- Provides strong security and is used in government and financial institutions.
- Suitable for encrypting files, databases, and network communications.
- Resistant to brute-force attacks due to its complex structure.

Drawbacks & Limitations

- Computationally intensive for low-power devices.
- Key management is critical and complex.
- Susceptible to side-channel attacks if improperly implemented.

Criticism & Link to Motivation

While AES is robust, its computational demands may hinder performance in embedded systems and file-handling applications. Our project aims to implement an efficient file encryption-decryption mechanism using optimized methods.

2.3 RSA encryption in C

RSA (Rivest-Shamir-Adleman) is an asymmetric encryption algorithm widely used for secure data transmission.

RSA in C is implemented using modular exponentiation, where encryption and decryption are performed using a public-private key pair. Large prime number generation and key pair computation are critical steps in the implementation.

Importance & Applications

- Used for secure email communication, digital signatures, and SSL/TLS encryption.
- Provides high security due to its reliance on number factorization problems.
- Supports non-repudiation in digital transactions.

Drawbacks & Limitations

- Computationally expensive, making it slower than symmetric encryption methods.
- Requires significant processing power for key generation and encryption.
- Not ideal for encrypting large files due to performance overhead.

Criticism & Link to Motivation

While RSA provides excellent security, its performance limitations make it less suitable for large-scale file encryption. This motivates the exploration of hybrid encryption methods to balance security and efficiency.

3. PROPOSED SYSTEM ARCHITECTURE

3.1 System Architecture

The proposed encryption-decryption system consists of the following components:

Component	Function	Input From	Output To
User Interface	Inputs plaintext/files; selects algorithm (AES/RSA)	None	Encryption Module
Encryption Module	Encrypts using AES/RSA with key from KMS	User Interface	File Handling System
File Handling	Securely reads/writes encrypted files	Encryption/Decryption Modules	Decryption/Encryption Modules
Decryption Module	Decrypts using AES/RSA with key	File Handling	Output Interface
Key Management	Generates/distributes/stores encryption/decryption keys	None	Enc./Decrypion Modules
Output Interface	Displays/saves decrypted output	Decrypiontation	

Table 3.1 System Architect

3.2 Data Flow Diagram

Step 1: The user inputs plain text or uploads a file.

Step 2: The encryption module processes the data using AES or RSA.

Step 3: The encrypted file is stored securely in the file system.

Step 4: When decryption is requested, the file is retrieved and decrypted.

Step 5: The decrypted content is displayed to the user or saved.

3.3 Tools and Environment

To develop and implement this system, the following tools and environments will be used:

- **Programming Language**: C (for core encryption and file handling operations).
- **Development Environment**: Code::Blocks,Visual Studio Code with GCC.
- **Operating System**: Windows.
- **Version Control**: Git for source code management, hosted on Github.

4. METHODOLOGY

The methodology for implementing file encryption and decryption using C follows a structured approach, ensuring efficiency, security, and reliability. We have divided the process is into several key phases:

4.1. Requirement Analysis:

- Identification of encryption and decryption requirements, including security standards and algorithms.
- Determination of file handling mechanisms and ensuring compatibility with Windows OS.
- Defining the scope and limitations of the encryption system.
- Researching potential threats and vulnerabilities associated with file encryption.
- Identifying user requirements and expected performance benchmarks.

4.2. Algorithm Selection:

- Choosing a suitable encryption algorithm such as:
    - **Advanced Encryption Standard (AES):** A strong encryption algorithm offering high security.
    - **Data Encryption Standard (DES):** A traditional method with moderate security.
    - **XOR-based encryption:** A simpler, lightweight encryption technique.
- Evaluating encryption strength, key size, and computational efficiency.
- Implementing a secure key generation and management strategy to prevent unauthorized access.

4.3. System Design:

- Defining the overall architecture of the encryption and decryption system, including:
    - User input handling.
    - File I/O operations.
    - Encryption and decryption module integration.
- Designing data structures and file-handling mechanisms for reading, encrypting, writing, and decrypting files.
- Establishing error-handling procedures to manage invalid input, corrupted files, or unauthorized access attempts.
- Developing a user-friendly interface for interacting with the encryption system.

4.4. Implementation:

- Developing core encryption and decryption functionalities using the C programming language, ensuring:
    - Efficient memory management.
    - Secure handling of encryption keys.
    - Optimized file reading and writing processes.
- Implementing file handling operations, including:
    - Reading plain text files and storing them in memory.
    - Applying encryption algorithms and writing the output to a new file.
    - Decrypting encrypted files back to plain text format.
- Utilizing Code::Blocks and Visual Studio Code for coding, debugging, and testing.

- Applying modular programming principles to enhance code maintainability and reusability.

4.5. Testing and Debugging:

- Performing unit testing to validate encryption and decryption functions under different scenarios.
- Conducting integration testing to ensure seamless communication between system components.
- Testing with different file formats (e.g., text, binary) and sizes to evaluate robustness and performance.
- Identifying and fixing potential security vulnerabilities, memory leaks, and runtime errors using debugging tools.
- Implementing boundary testing to check for data corruption and unauthorized access attempts.

4.6. Version Control and Documentation:

- Using Git and GitHub for source code management and version control, ensuring:
    - Proper commit messages.
    - Branching strategies for feature development.
    - Collaborative contributions and code reviews.
- Maintaining detailed documentation for:
    - System architecture and workflow.
    - Algorithm implementation and encryption logic.

- User guidelines and troubleshooting steps.
    - Code comments and inline documentation for future development.

4.7. Deployment and Evaluation:

- Deploying the encryption system on Windows OS and conduct final system checks.
- Evaluating system performance based on:
    - Encryption and decryption speed.
    - Security level against brute force and cryptographic attacks.
    - Ease of use and accessibility.
- Gathering user feedback to assess practical usability and identify areas for improvement.
- Plan future enhancements such as GUI integration, cross-platform compatibility, and additional security features.

By following this detailed methodology, we will be developing the file encryption and decryption system using C efficiently, ensuring robustness, security, and user-friendliness while maintaining best coding practices and efficient software management.

5. SCOPE AND APPLICATIONS

The scope of this project includes:

- Developing a lightweight encryption and decryption tool that can be used to secure files.
- Implementing different encryption techniques based on security needs.
- Creating a command-line interface (CLI) tool for file encryption and decryption.

This encryption and decryption system has a wide range of applications. Some of them are as follows:

1. Data Protection for Personal & Business Use:
    - Encrypts sensitive files like financial records and private documents.
    - Secures business data, contracts, and customer information from breaches.
2. Secure Communication:
    - Encrypts messages and documents for safe transmission over networks.
    - Ensures only intended recipients can access the information.
3. Cyber security & Ethical Hacking:
    - Helps security professionals test encryption methods.
    - Assists ethical hackers in understanding low-level cryptography.
4. Educational & Learning Purposes:
    - Useful for students and developers learning encryption in C.
    - Provides hands-on experience with cryptographic techniques.
5. Secure File Storage & Backup:
    - Prevents unauthorized access to stored files.

- Helps organizations comply with data security regulations.

6. TIME ESTIMATION

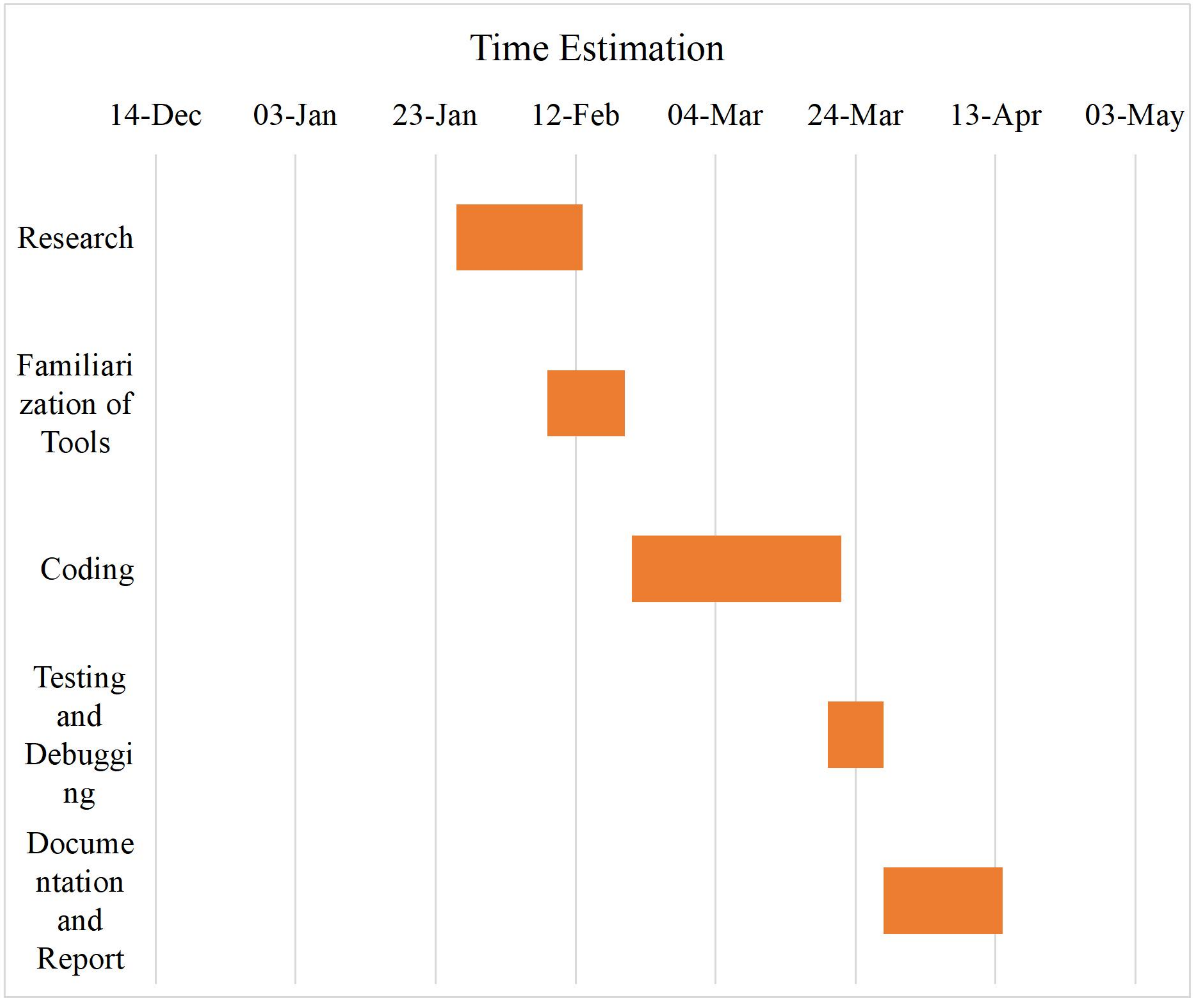

Table 6 Gantt Chart with project activities and timeline

7. FEASIBILITY ANALYSIS

This project aims to secure sensitive data by encrypting and decrypting files using C.

7.1 Technical Feasibility

- Language & Platform: C, Windows.
- Tools: GCC, file handling.
- Algorithms: AES, XOR-based or RSA.
- Requirements: Intel i3, 2GB RAM, 100MB storage.

7.2 Operational Feasibility

- CLI-based user interaction.
- Workflow: User selects mode → Provides file → Encryption/Decryption → Output file generated.

7.3 Economic Feasibility

- Low cost: Uses free tools.
- Minimal maintenance.

7.4 Legal & Ethical Feasibility

- Compliance with data privacy laws.
- Ethical use encouraged.

The project is feasible, cost-effective, and implementable within schedule.

References

1. NIST (National Institute of Standards and Technology) Publications (AES standard)
2. William Stallings, Cryptography and Network Security: Principles and Practice, Pearson Education.
3. .Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, Wiley.
4. Rivest, R. (1978). "The MD5 Message-Digest Algorithm." MIT Technical Report.
5. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. CRC Press.
6. Wikipedia