SOFTWARE REQUIREMENT SPECIFICATION(SRS)
FOR
LIGHT WEIGHT CRYPTOGRAPHIC ALGORITHM

Submitted By

ANAND GUGULOTH - 160121749037
CHANAKYA KUSUMA - 160121749044

Branch: Department of Computer Engineering and Technology

Submitted To

Smt. KAVITHA AGARWAL,
Assistant Professor, Dept. of CET,
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY(A)

TABLE OF CONTENTS

1.INTRODUCTION

The ever-growing realm of resource-constrained devices, such as those in the Internet of Things (IoT), wearable electronics, and sensor networks, necessitates the development of secure communication protocols. These devices often have limited processing power, memory, and battery life, posing a challenge for traditional cryptographic algorithms that can be computationally expensive. To address this gap, lightweight cryptography emerges as a crucial field dedicated to designing cryptographic primitives specifically tailored for resource-constrained environments.

This project introduces a novel lightweight cryptographic algorithm, aiming to provide secure communication for resource-constrained devices. Our proposed algorithm, [Name of your algorithm], is designed to be efficient in terms of [mention specific aspects like low memory footprint, low power consumption, fast execution speed, etc.]. We believe New Light Weight Cryptography Algorithm offers a promising solution for securing communication in resource-constrained environments by achieving a balance between security and efficiency.

1.1 PURPOSE

This Software Requirements Specification (SRS) document outlines the complete requirements for a new lightweight cryptographic algorithm. This document serves as a guide for the development and implementation of the algorithm, facilitating clear communication and understanding among developers, testers, and stakeholders.
The purpose of the New lightweight cryptographic algorithm is to provide a secure and efficient method for encrypting and decrypting data, particularly targeting resource-constrained environments or applications where performance is a critical factor. The algorithm is designed to protect the confidentiality of data by transforming plaintext into ciphertext, making it unintelligible to unauthorized parties.
This SRS document will detail the algorithm's design, including its key generation process, encryption and decryption procedures, padding scheme, and data encoding format. It will also address performance considerations, security requirements, and any relevant constraints on the algorithm's implementation.

1.2 SCOPE

The New lightweight cryptographic algorithm is a symmetric-key block cipher designed for efficient and secure data encryption. The algorithm operates on 128-bit blocks of data and uses a 128-bit key for encryption and decryption.
Specifically, this algorithm will cover the following aspects:
Key Generation: A secure process for generating a master key from a user-provided password and deriving 8 sub-keys for use in the encryption and decryption rounds.
Encryption: A detailed description of the encryption process, including block splitting, round function operations (using a Feistel network structure), S-box substitution, bit rotation, key mixing, master key XOR (whitening-like), padding, and Base64 encoding of the ciphertext.
Decryption: The inverse operations of encryption, including ciphertext decoding, sub-key reversal, inverse round function, master key XOR, and padding removal, to recover the original plaintext.

1.3 DEFINTIONS, ACRONYMS AND ABBREVIATIONS

- Plaintext: The original, unencrypted data that is input to the encryption algorithm.
- Ciphertext: The encrypted data output from the encryption algorithm, making the original data unintelligible.
- Encryption: The process of transforming plaintext into ciphertext using a cryptographic algorithm and a key.
- Decryption: The process of transforming ciphertext back into plaintext using a cryptographic algorithm and the correct key.
- Symmetric-key Encryption: A type of encryption where the same key is used for both encryption and decryption.
- Block Cipher: A type of symmetric-key encryption that operates on fixed-size blocks of data.
- Key: A secret value used by a cryptographic algorithm to perform encryption and decryption.
- Feistel Network: A common block cipher structure where the data block is split into two halves, and a round function is applied iteratively to one half using sub-keys.
- SPN (Substitution-Permutation Network): Another block cipher structure that utilizes substitution (S-boxes) and permutation (P-boxes) to achieve confusion and diffusion.
- S-box (Substitution Box): A non-linear lookup table used in SPN structures to substitute bytes of data for confusion.
- Round Function: A function applied in each round of the Feistel network, incorporating substitution, permutation, and key mixing operations.
- Master Key: The initial key derived from the user's password using PBKDF2.
- Sub-keys: Keys derived from the master key and used in individual rounds of encryption and decryption.
- Key Schedule: The algorithm for generating sub-keys from the master key.
- Padding: The process of adding extra bytes to the plaintext to ensure it is a multiple of the block size.
- Base64 Encoding: A method for encoding binary data as text using a 64-character alphabet, making it suitable for storage or transmission in text-based formats.
- SRS: Software Requirements Specification
- PBKDF2: Password-Based Key Derivation Function 2
- HMAC: Hash-based Message Authentication Code
- SHA-256: Secure Hash Algorithm 256-bit

1.4 REFERENCES

Lightweight Cryptography - an overview | ScienceDirect Topics

General Structure of Feistel network | Download Scientific Diagram (researchgate.net)

Substitution-Permutation Network - an overview | ScienceDirect Topics

Performance Evaluation and Review of Lightweight Cryptography in an Internet-of-Things Environment | IEEE Conference Publication | IEEE Xplore

1.5 OVERVIEW

This document outlines the Software Requirements Specification (SRS) for a new, custom-designed symmetric-key encryption software. The software will be able to encrypt and decrypt data using a hybrid cryptographic algorithm that combines elements of Feistel network and Substitution-Permutation Network (SPN) principles. The goal of this software is to provide a reasonable level of security while maintaining efficiency for data protection.

The software will consist of two main components:

Key Generation Module: This module will be responsible for generating a secure master key and a set of sub-keys derived from a user-provided password. This process will involve using a robust key derivation function (PBKDF2) and incorporating elements of randomness (salt) and key stretching for enhanced security. The generated keys will be stored in a text file for later use in encryption and decryption.

Encryption/Decryption Module: This module will handle the encryption and decryption of data using the generated keys. The encryption algorithm will operate on 128-bit blocks, utilizing 8 rounds of a hybrid Feistel-SPN structure. Each round will involve S-box substitution, bit rotation, XOR operations with sub-keys, and swapping of data halves. The ciphertext will be encoded using Base64 for easier handling and storage. The decryption module will reverse the encryption process using the same keys and operations to recover the original plaintext.

This SRS document will detail the specific requirements for both modules, including functional requirements, performance requirements, security considerations, and user interface specifications. The document will serve as a guide for the design, development, and testing of the encryption software, ensuring that it meets the desired security and functionality goals.

2. GENERAL DESCRIPTION

The general description section of the Software Requirements Specification (SRS) document provides a comprehensive overview of the symmetric-key encryption software, highlighting its purpose, target users, functionalities, and underlying assumptions. This software is envisioned as a user-friendly application designed for individuals seeking a reasonable level of security for their personal data. It prioritizes ease of use and employs a robust hybrid cryptographic algorithm that draws upon the strengths of both Feistel network and Substitution-Permutation Network (SPN) principles.

This encryption software aims to empower users to protect their sensitive information from unauthorized access or casual snooping. It offers core functionalities such as generating encryption keys from user-provided passwords, encrypting data using these generated keys, and decrypting ciphertext to recover the original plaintext. The target audience for this software is individuals with basic computer literacy who may not possess extensive knowledge of cryptography or advanced security concepts.

The software will be developed using Python, ensuring compatibility with widely used operating systems like Windows, macOS, and Linux. It will rely solely on the standard

Python library, eliminating the need for specialized hardware or external libraries. This design choice prioritizes accessibility and ease of installation for the intended user base.

It's crucial to acknowledge certain assumptions and limitations. The software relies on users choosing strong passwords and securely storing their generated keys. Users must understand that this software is not intended for high-security applications or environments where data is subject to attacks from sophisticated adversaries. Its primary focus is to offer a practical and user-friendly solution for safeguarding personal data against common security threats.

2.1 PRODUCT PERSPECTIVE

This symmetric-key encryption software is designed as a standalone application, offering a user-friendly and practical solution for individuals seeking to protect their personal data with a reasonable level of security. It focuses on providing a straightforward and intuitive interface for encrypting and decrypting sensitive information. The software employs a hybrid cryptographic algorithm that blends elements of the Feistel network and Substitution-Permutation Network (SPN) principles, providing a balanced approach to security and efficiency.

The target audience for this software is individuals who need to protect their personal data but may not have extensive technical expertise in cryptography or security. The software aims to empower these users with a tool that is easy to understand and use, without requiring complex configurations or specialized knowledge.

This software is not intended for high-security applications or environments where data is susceptible to attacks from sophisticated adversaries or requires adherence to strict regulatory standards, such as HIPAA or PCI DSS. Its primary focus is to provide a practical and accessible security solution for everyday users concerned about protecting their personal information.

The core functionality of the software revolves around three key aspects: key generation, encryption, and decryption. Key generation allows users to create a secure master key and sub-keys derived from a user-provided password. This process employs the robust PBKDF2 algorithm with SHA-256, incorporating a random salt and allowing users to specify the number of iterations for key stretching, enhancing the resilience against brute-force attacks.

The encryption process utilizes the generated keys and a hybrid Feistel-SPN algorithm with 8 rounds. This approach combines the strengths of both structures, offering both confusion and diffusion properties. The Feistel structure provides a framework for iterative encryption with efficient decryption, while the SPN elements, such as the S-box substitution and bit rotation, contribute to the cryptographic strength of each round.

Decryption reverses the encryption process, utilizing the same keys and operations in reverse order to recover the original plaintext. The software handles padding, ensuring that the data is correctly aligned for encryption and decryption, and utilizes Base64 encoding for easier storage and handling of the ciphertext.

The software's design prioritizes simplicity and user-friendliness. The interface is designed to guide users through the encryption and decryption process with clear instructions and prompts. The software relies solely on the standard Python library, eliminating the need for any additional installations or dependencies, making it accessible to a wider range of users.

It's essential to acknowledge the software's limitations. Users are responsible for selecting strong passwords for key generation and for securely storing the generated key files. The software assumes that it will be used in environments where the primary threats are casual snooping, unauthorized access by individuals with limited technical skills, or accidental data loss.

2.2 PRODUCT FEATURES

This section details the features of the symmetric-key encryption software, outlining its functionalities in key generation, encryption, and decryption.

1. Key Generation Features:

The Key Generation Module will provide the following features:

- Password-Based Key Derivation:
  - The software will employ the PBKDF2 (Password-Based Key Derivation Function 2) algorithm with HMAC-SHA256 to derive a secure master key from the user-supplied password.
  - This ensures that even weak passwords are transformed into strong cryptographic keys.
- Salt Incorporation:
  - A 16-byte random salt will be generated using a cryptographically secure random number generator (CSPRNG) for each key generation process.
  - The salt will be combined with the password during key derivation, preventing pre-computation attacks and ensuring unique keys even for identical passwords.
- Key Stretching:
  - Users will be able to specify the number of iterations for PBKDF2, allowing for key stretching.
  - Key stretching increases the computational effort required to brute-force the password, significantly enhancing security.
- Sub-key Generation:
  - The software will generate 8 sub-keys from the master key using a custom key schedule algorithm that incorporates SHA-256 for round constant generation.
  - These sub-keys will be utilized in different rounds of the encryption algorithm, further enhancing security and complexity.
- Key Storage:
  - The generated master key and sub-keys will be saved to a text file with a unique filename based on a timestamp.
  - This ensures that each set of keys is stored separately and prevents accidental overwriting.

2. Encryption Features

- Block Cipher Operation:
  - The encryption algorithm will operate on 128-bit (16-byte) blocks of data.
  - This block size is widely used in modern cryptographic algorithms and provides a good balance between security and efficiency.
- Padding:
  - The software will implement a padding scheme to ensure that the input data length is a multiple of the block size.
  - Padding bytes will be added with a value equal to the padding length, providing a deterministic and reversible padding method.
-
- Hybrid Feistel-SPN Algorithm:
  - The encryption process will utilize a hybrid algorithm that combines elements of both Feistel network and SPN structures.
  - This approach leverages the strengths of both structures, enhancing both confusion and diffusion properties.
- Eight Encryption Rounds:
  - The algorithm will perform 8 rounds of encryption for each data block, utilizing different sub-keys in each round for added security.
- Round Function:
  - The round function will include the following operations:
  - Substitution: An S-box will be used for byte substitution, introducing non-linearity.
  - Rotation: Bit rotation will be applied to the substituted bytes for diffusion.
  - Key Mixing: The rotated bytes will be XORed with the sub-key for key-dependent diffusion.
- Master Key XOR:
  - After the 8 rounds of encryption, the ciphertext will be XORed with the master key for an additional layer of security, similar to key whitening techniques.
- Base64 Encoding:
  - The resulting ciphertext will be encoded using Base64 to ensure safe and convenient storage and transmission.
- Ciphertext Output:
  - The Base64 encoded ciphertext will be saved to a text file with a unique filename based on a timestamp.
  - The filename will also include the key filename for easy association with the corresponding keys.

3. Decryption Features

- Ciphertext Input: The software will allow users to select a ciphertext file for decryption.
- Base64 Decoding: The Base64 encoded ciphertext will be decoded back to its original binary form.
- Key Import: The corresponding key file will be loaded to retrieve the master key and sub-keys.
- Reverse Sub-key Order: The order of the sub-keys will be reversed to perform the decryption rounds correctly.

- Decryption Rounds: The decryption process will follow the same 8 rounds as encryption but in reverse order, using the reversed sub-keys and the inverse of the round function.
- Master Key XOR: The decrypted plaintext will be XORed with the master key to reverse the XOR operation performed during encryption.
- Padding Removal: The padding bytes will be removed from the decrypted plaintext to recover the original data.
- Plaintext Output: The decrypted plaintext will be displayed to the user.

2.3 User Classes and Characteristics

Target User: Individual User

- The primary target user for this software is the Individual User. This user class encompasses individuals who:
- Need to protect personal data: They have sensitive information on their computers that they want to keep confidential, such as financial records, personal documents, or private photos.
- Have basic computer literacy: They are comfortable with navigating file systems, opening and saving files, and entering text into applications.
- May not have extensive technical knowledge: They may not possess in-depth understanding of cryptography or security principles.
- Desire a simple and user-friendly solution: They prefer software that is easy to use and understand, without requiring complex configuration or specialized knowledge.

Characteristics:

- Operating System: Likely using common operating systems such as Windows, macOS, or Linux.
- Technical Skills: Basic understanding of file management, opening and saving files, and text input.
- Security Awareness: May have a general awareness of data security but may not be familiar with advanced cryptographic concepts.
- Motivation: Driven by a desire to protect their personal data from unauthorized access, accidental loss, or casual snooping.

Excluded User Classes

- The following user classes are not the intended audience for this software:
- Advanced Users: Users with strong technical expertise in cryptography or security who might require more sophisticated encryption solutions with customizable algorithms and key management options.
- Organizations: Businesses or institutions that handle sensitive data requiring compliance with strict regulatory standards (e.g., HIPAA, PCI DSS). These users typically require enterprise-level encryption solutions with advanced features and management capabilities.

2.4 OPERATING ENVIRONMENT

The software is intentionally designed to be lightweight and efficient, ensuring compatibility with a broad range of commonly available computers and operating systems. It aims to function seamlessly on typical consumer-grade hardware without demanding excessive processing power or memory resources.

1. Hardware Infrastructure

- Processor: Any modern processor, including those found in most laptops and desktops, with a clock speed of 1 GHz or higher is sufficient to run the software effectively.
- Memory (RAM): The software requires a minimum of 512 MB of RAM to ensure smooth operation, a standard feature in most contemporary computers.
- Storage: The software, including its executable files, along with the generated key files and ciphertext files, will require a modest amount of disk space. The exact storage requirements will depend on the size of the data being encrypted, but typical usage should not pose any storage limitations on modern computers.

2. Software Environment

- Windows: The software will run on Windows 7 and all subsequent versions of the Windows operating system, encompassing a large user base.
- macOS: Users running macOS version 10.13 or later will be able to use the software without compatibility issues.
- Linux: The software is designed to function seamlessly on any modern Linux distribution that has a compatible Python environment installed.

3. Network Infrastructure

- A notable advantage of this encryption software is its ability to function completely offline. It does not have any specific network requirements, allowing users to encrypt and decrypt their data without an active internet connection. While users may choose to transfer key files or ciphertext files over a network if desired, this is not essential for the software's core encryption and decryption operations.

4. Other Environmental Considerations

- File System Access: The software necessitates access to a file system to read and write key files and ciphertext files. The user should have appropriate permissions to interact with the file system and choose directories for storing these files.
- User Interface: The software will employ a simple command-line interface (CLI) for user interaction. This approach is chosen for its minimal resource requirements and its compatibility across different operating systems.

2.5 DESIGN AND IMPLEMENTATION CONSTRAINTS

Design and implementation constraints define the limitations and boundaries within which the light weight cryptography algorithm must be developed. These constraints can stem from technological, operational, legal, or environmental factors and must be carefully considered to ensure the system's success. This section outlines the key design and implementation constraints for the light weight cryptography algorithm.

Programming Language and Libraries

- The software will be implemented using the Python programming language (version 3.6 or higher). Python is chosen for its readability, ease of use, and extensive standard library.

- The software will rely solely on the standard Python library. No external third-party libraries will be used. This constraint ensures portability, reduces dependencies, and simplifies installation for users.

User Interface

- The software will have a command-line interface (CLI). This provides a simple and efficient way for users to interact with the software, minimizing resource consumption and maintaining compatibility across different operating systems.
- No graphical user interface (GUI) will be developed.

Cryptographic Algorithm

- The encryption algorithm will be a custom-designed hybrid algorithm combining elements of Feistel network and SPN structures.
- The specific details of the algorithm, including the S-box, key schedule, and round function, are provided in the detailed design documentation.

Security Considerations

- The software is designed for individual users seeking a reasonable level of security for their personal data. It is not intended for high-security applications or environments with sophisticated threats.
- The security of the encryption relies heavily on the strength of the user-provided password. The software will encourage users to choose strong and unique passwords.
- The software will not implement advanced security features such as:
- Key management systems: Users are responsible for securely storing the generated key files.
- Hardware security modules (HSMs): Keys will be stored in software.
- Side-channel attack countermeasures: The implementation will not explicitly address side-channel attacks.

Performance

- The software should perform encryption and decryption operations with reasonable speed and efficiency on typical consumer-grade hardware.
- Specific performance benchmarks will be defined and tested during development.

Maintenance and Extensibility

- The code will be written with clarity and maintainability in mind, using well-documented functions and modular design.
- The software is not intended to be highly extensible. However, the design should allow for potential future updates or improvements to the cryptographic algorithm or other features.

2.6 ASSUMPTIONS AND DEPENDENCIES

Assumptions and dependencies are conditions or elements that must be in place for the Light weight cryptography algorithm to function as intended. Assumptions are accepted as true or as certain to happen without concrete proof, while dependencies are aspects that rely on other systems, processes, or external factors. Identifying these elements is crucial for understanding the context in which the system is developed and deployed. This section outlines the key assumptions and dependencies for the Light weight cryptography algorithm.

Assumptions

- The development and functionality of the software rely on the following key assumptions:

- Strong Password Selection: It is assumed that users will choose strong and unique passwords for key generation. The software's security heavily depends on the strength of these passwords. While the software will encourage the use of strong passwords, it will not enforce password complexity rules.
- Secure Key Storage: The software assumes that users will be responsible for securely storing the generated key files. The software itself will not provide mechanisms for secure key storage or management.
- Appropriate Use Case: The software is intended for individual users seeking a reasonable level of security for their personal data. It is assumed that the software will not be used in high-security applications or environments where data is subject to attacks from sophisticated adversaries.
- User Understanding of Limitations: It is assumed that users understand the software's limitations and will not rely on it for protecting extremely sensitive data in high-risk environments.

Dependencies

- Python Environment: The software is dependent on a compatible Python environment (version 3.6 or higher) being installed on the user's system.
- File System Access: The software requires access to a file system to read and write key files and ciphertext files. The user must have appropriate permissions to interact with the file system.
- Operating System Support: The software's functionality is dependent on the underlying operating system (Windows, macOS, or Linux) providing the necessary system calls and libraries for file I/O and other operations.

3. SYSTEM FEATURES AND REQUIREMENTS

The system features and requirements section outlines the key functionalities, performance standards, security measures, and other essential attributes for Light weight Cryptography algorithm. This section details the specific features and requirements of the symmetric-key encryption software, encompassing both functional and non-functional aspects.

Functional Requirements

Functional requirements define what the software should do. They describe the specific actions and behaviors the software must perform.

Key Generation Module

- FR-KEYGEN-01: Password Input: The software shall allow the user to enter a password as a string.
- FR-KEYGEN-02: Salt Generation: The software shall generate a 16-byte random salt using a cryptographically secure random number generator.
- FR-KEYGEN-03: Key Derivation: The software shall use the PBKDF2-HMAC-SHA256 algorithm to derive a 128-bit master key from the password and salt.
- FR-KEYGEN-04: Iteration Control: The software shall allow the user to specify the number of iterations for PBKDF2 (for key stretching).
- FR-KEYGEN-05: Sub-key Generation: The software shall generate 8 128-bit sub-keys from the master key using a custom key schedule algorithm incorporating SHA-256.
- FR-KEYGEN-06: Key File Output: The software shall save the master key and sub-keys to a text file with a unique filename based on a timestamp.
- FR-KEYGEN-07: File Format: The key file shall store the keys in the following format:

Master Key: <hexadecimal representation of master key>
Sub-key 1: <hexadecimal representation of sub-key 1>
Sub-key 2: <hexadecimal representation of sub-key 2>
...
Sub-key 8: <hexadecimal representation of sub-key 8>

Encryption Module

- FR-ENC-01: Data Input: The software shall allow the user to input the data to be encrypted.
- FR-ENC-02: Data Encoding: The software shall convert the input data to bytes using UTF-8 encoding.
- FR-ENC-03: Padding: The software shall pad the data to a multiple of the block size (16 bytes) using padding bytes equal to the padding length.
- FR-ENC-04: Key File Input: The software shall allow the user to select the key file containing the master key and sub-keys.
- FR-ENC-05: Encryption Algorithm: The software shall encrypt the data using the custom hybrid Feistel-SPN algorithm with 8 rounds.
- FR-ENC-06: Master Key XOR: The software shall XOR the ciphertext with the master key.
- FR-ENC-07: Base64 Encoding: The software shall encode the ciphertext using Base64.
- FR-ENC-08: Ciphertext Output: The software shall save the Base64 encoded ciphertext to a text file.
- FR-ENC-09: Ciphertext Filename: The ciphertext filename shall include a timestamp and the key filename used for encryption.

Decryption Module

- FR-DEC-01: Ciphertext Input: The software shall allow the user to select the ciphertext file to be decrypted.
- FR-DEC-02: Key File Input: The software shall allow the user to select the corresponding key file used for encryption.
- FR-DEC-03: Base64 Decoding: The software shall decode the Base64 encoded ciphertext.
- FR-DEC-04: Decryption Algorithm: The software shall decrypt the ciphertext using the same hybrid algorithm and keys used for encryption but with reversed operations and sub-key order.
- FR-DEC-05: Master Key XOR: The software shall XOR the decrypted data with the master key.
- FR-DEC-06: Padding Removal: The software shall remove the padding bytes from the decrypted data.
- FR-DEC-07: Plaintext Output: The software shall display the decrypted plaintext to the user.

Non-Functional Requirements

Non-functional requirements specify how the software should perform. They describe aspects of the software's quality and behavior.

Performance Requirements

- NFR-PERF-01: Encryption/Decryption Speed: The software shall encrypt and decrypt data at a reasonable speed, such that the processing time does not cause noticeable delays for typical data sizes used by individual users.

Security Requirements

- NFR-SEC-01: Algorithm Strength: The encryption algorithm shall provide a reasonable level of security against common attacks.
- NFR-SEC-02: Key Derivation Security: The key derivation process shall utilize a strong key derivation function (PBKDF2) with sufficient iterations for key stretching to protect against brute-force attacks.
- NFR-SEC-03: Salt Randomness: The salt used in key derivation shall be generated using a cryptographically secure random number generator.

Usability Requirements

- NFR-USAB-01: Ease of Use: The software shall be simple and straightforward to use, with clear instructions and prompts guiding the user through the key generation, encryption, and decryption processes.

Maintainability Requirements

- NFR-MAINT-01: Code Clarity: The software shall be written with clarity and maintainability in mind, using well-documented functions and a modular design.
- This detailed description of the system features and requirements provides a comprehensive guide for the development of the symmetric-key encryption software.

3.1 SYSTEM FEATURES

Key Generation: The system generates a strong key using a proven technique like PBKDF2, which combines a user's password with a random salt. This ensures the key is unique and difficult to guess.
Encryption and Decryption: The system encrypts data using a lightweight algorithm you've chosen. This algorithm should be specified, detailing its key size, block size, and mode of operation. The system can handle various data types (text files, binary data, etc.) by padding them to a specific size for proper encryption. Decryption utilizes the same algorithm and key to reverse the process, removing the padding and returning the original data. Error handling mechanisms are implemented to address issues like invalid keys or corrupted data.
An optional user interface could be developed to simplify interaction. Users could provide data, passwords, and select files for encryption or decryption, with the system displaying the encrypted/decrypted data and relevant information.

Security is paramount. The system prioritizes secure key management, potentially using user password management or secure key derivation techniques. Cryptographically secure random number generation ensures strong salts, and input validation helps prevent vulnerabilities.

Performance is also considered. Benchmarking might compare your algorithm's speed to established ones like AES. If targeting resource-constrained devices, monitoring memory and processing power usage during encryption/decryption is crucial.

3.2 EXTERNAL INTERFACES

For your lightweight cryptographic algorithm project, the external interfaces will depend on whether you're implementing a user interface (UI) or not. Here's a breakdown for both scenarios:

1. With User Interface (UI):

- **User Input:**
    - The UI will provide mechanisms for users to enter:
        - Data to be encrypted/decrypted (text box, file selection dialog).
        - Password for key generation (password field).
- **User Output:**
    - The UI will display the results of encryption/decryption operations:
        - Encrypted/decrypted data (text box, file download option).
        - Additional information (e.g., filename, key length).

2. Without User Interface (No UI):

- **Input:**
    - The system will likely require code-level integration for inputting data and passwords. This could involve:
        - Function calls to provide data bytes or file paths.
        - Function calls to provide password strings.
- **Output:**
    - The system will provide functions to retrieve the results:
        - Function calls to obtain encrypted/decrypted data bytes.
        - Function calls to access additional information (e.g., key length) as needed.

Regardless of UI:

- **Error Handling:** The system should provide mechanisms (function calls or error messages) to indicate issues like:
    - Invalid input (e.g., incorrect password, corrupted data).
    - Encryption/decryption errors.

By clearly defining these external interfaces in your SRS document, you ensure proper integration with any external components (UI or other systems) that might interact with your lightweight cryptographic algorithm.

3.3 PERFORMANCE REQUIREMENTS

The performance of this lightweight cryptographic algorithm system will be evaluated based on its efficiency and resource usage.

- **Encryption/Decryption Speed:** We'll measure the speed of the encryption and decryption processes. This might involve defining a target throughput (e.g., megabytes per second) or encryption/decryption time for specific data sizes. Additionally, we'll benchmark the performance against a standard algorithm like AES to understand the efficiency gains achieved by the lightweight approach.
- **Resource Constraints:** If targeting resource-constrained devices, memory footprint and processing power consumption are crucial considerations. We'll establish acceptable memory usage limits during encryption/decryption and potentially outline constraints on processing power usage to ensure efficient operation on these devices.

- **Trade-offs Acknowledged:** It's important to recognize the inherent trade-off between performance and security. While lightweight algorithms prioritize speed, they might have a lower security ceiling compared to robust algorithms. The specific balance between these factors will be determined based on the project's intended use case.

3.4 SAFETY REQUIREMENTS

While lightweight cryptographic algorithms offer potential benefits like speed and efficiency, they introduce certain safety considerations compared to well-established and rigorously tested algorithms. Here's a breakdown of the key safety requirements for your project:

1. Security Focus:

- **Acknowledge Limitations:** Emphasize that for real-world applications involving highly sensitive data, established and rigorously tested cryptographic algorithms like AES are the recommended choice due to their proven security. Lightweight algorithms may have vulnerabilities that haven't been identified through extensive cryptanalysis.

2. Secure Key Management:

- **Key Strength:** Utilize key lengths appropriate for the chosen algorithm (e.g., 128-bits for AES). Longer keys provide stronger security.
- **Key Storage:** Implement secure mechanisms for storing the generated key. This might involve:
    - User password management with strong password hashing techniques (e.g., bcrypt).
    - Key derivation from a user passphrase using a secure key derivation function (e.g., PBKDF2).
- **Key Exchange (if applicable):** If the system involves key exchange between parties, ensure a secure key exchange protocol is used (e.g., Diffie-Hellman key exchange).

3. Cryptographically Secure Randomness:

- **Salt Generation:** Utilize a cryptographically secure random number generator (CSPRNG) for generating random salts during key derivation. This helps prevent predictable key generation based on passwords.

4. Input Validation:

- **Data Integrity:** Implement mechanisms to validate the integrity of input data (e.g., message authentication codes) to prevent potential manipulation during encryption/decryption.
- **Password Strength:** If using user passwords for key generation, enforce password complexity requirements (e.g., minimum length, character variety) to improve key security.

5. Error Handling:

- **Graceful Error Handling:** Implement mechanisms to handle errors gracefully, such as:
    - Invalid passwords or corrupted data.
    - Encryption/decryption failures.
    - Providing informative error messages without revealing sensitive information.

6. Code Review and Testing:

- **Peer Review:** Encourage code review by a security-conscious developer to identify potential vulnerabilities in the algorithm implementation.
- **Fuzz Testing:** Consider employing fuzz testing techniques to identify unexpected behavior or crashes in the algorithm when handling malformed or invalid inputs.

3.5 SECURITY REQUIREMENTS

The security of the lightweight cryptographic algorithm is paramount. While it offers potential benefits in terms of performance and efficiency, it's crucial to acknowledge the inherent trade-offs compared to well-established and rigorously tested algorithms. This section outlines the security requirements to mitigate risks and ensure responsible development.

1. Algorithm Security:

- **Selection and Justification:** Clearly state the chosen lightweight algorithm and provide a rationale for its selection. Consider factors like security analysis history, suitability for the target use case, and potential vulnerabilities identified in the research community.
- **Custom Algorithm Risks:** If developing a custom algorithm, acknowledge the significant complexity of achieving cryptographically secure designs. Emphasize the importance of rigorous cryptanalysis by security experts to identify and address potential vulnerabilities before real-world deployment. For sensitive data applications, established algorithms are generally the preferred choice due to their proven security track record.

2. Key Management:

- **Key Strength:** Utilize key lengths appropriate for the chosen algorithm and industry standards (e.g., 128-bits for AES). Longer keys provide stronger security against brute-force attacks.
- **Secure Key Generation:** Employ a well-established key derivation function (KDF) like PBKDF2 to generate keys from user passwords. This process should combine the password with a cryptographically secure random salt to prevent predictable key generation.
- **Key Storage:** Implement secure mechanisms for storing the generated key. This might involve:
    - User password management with strong password hashing techniques (e.g., bcrypt) to protect passwords from unauthorized access.
    - Secure key storage mechanisms provided by the operating system or platform.

    - Avoiding storing keys in plain text or easily reversible formats.

3. Cryptographic Primitives:

- **Random Number Generation:** Utilize a cryptographically secure random number generator (CSPRNG) for generating random salts and any other randomness required by the algorithm. Weak randomness can compromise security.
- **Modes of Operation:** If the algorithm uses block ciphers, ensure a secure mode of operation is employed (e.g., CBC with padding, GCM) to prevent attacks like ciphertext malleability.

4. Input Validation:

- **Data Integrity:** Implement mechanisms to validate the integrity of input data (e.g., message authentication codes or digital signatures) to prevent potential manipulation during encryption/decryption. This ensures the authenticity and unaltered state of the data.
- **Input Sanitization:** Sanitize user input to prevent injection attacks (e.g., SQL injection) that might exploit vulnerabilities in the system.

5. Error Handling:

- **Graceful Error Handling:** Implement mechanisms to handle errors gracefully, such as:
    - Invalid passwords or corrupted data.
    - Encryption/decryption failures.
    - Providing informative error messages without revealing sensitive information (e.g., avoiding error messages that disclose specific details about the encryption process or key lengths).

6. Code Review and Testing:

- **Peer Review:** Encourage code review by a security-conscious developer to identify potential vulnerabilities in the algorithm implementation. This review should involve individuals familiar with cryptographic principles and best practices.
- **Static Code Analysis:** Utilize static code analysis tools to detect common coding errors and vulnerabilities in the algorithm implementation.
- **Fuzz Testing (Optional):** Consider employing fuzz testing techniques to identify unexpected behavior or crashes in the algorithm when handling malformed or invalid inputs. This can help uncover potential edge-case vulnerabilities.

7. Documentation and Transparency:

- **Clear Documentation:** Document the design and implementation of the algorithm in detail. This documentation should be clear, concise, and accessible to security professionals for review. However, avoid publicly disclosing critical details that could be exploited by attackers (e.g., specific key schedules or internal algorithm operations).
- **Transparency:** Be transparent about the limitations of the lightweight algorithm, especially regarding its security posture compared to established algorithms.

3.6 SOFTWARE QUALITY ATTRIBUTES

Security:

- This is paramount. While lightweight algorithms offer performance benefits, they might have lower security compared to established ones.
- Key considerations include:
    - Algorithm security (established vs. custom).
    - Secure key management (strength, generation, storage).
    - Cryptographic primitives (random number generation, modes of operation).
    - Input validation (data integrity, sanitization).
    - Error handling (graceful, informative).

Performance:

- A key advantage of lightweight algorithms.
- Focus on:
    - Encryption/decryption speed (throughput, benchmarks).
    - Resource usage (memory footprint, processing power) - especially for constrained devices.
- Remember the trade-off between performance and security.

3.7 DATABASE REQUIREMENTS

The project does not require a traditional database.

4. OTHER REQUIREMENTS

Besides core functionalities, security, performance, and software quality, there might be additional considerations for lightweight cryptographic algorithm. These depend on your specific goals:

- **Integration:** If your algorithm needs to work with existing systems, define how they'll interact (APIs, data formats, communication protocols).
- **Interoperability:** For working with different encryption algorithms, outline requirements like supporting standard formats (OpenPGP) or format conversion capabilities.
- **Logging/Auditing (Optional):** Depending on security needs, consider logging encryption/decryption events or implementing audit trails for user actions and potential security incidents.
- **Regulations (Optional):** Research and outline any relevant data security and encryption regulations your project might need to comply with.
- **Documentation:** Consider user manuals or tutorials if you have a user interface, to guide users on how to effectively use the system.

- **Future Enhancements:** Briefly mention any potential future improvements you envision, such as supporting additional data types or integrating with specific platforms.

4.1 LEGAL AND REGULATORY REQUIREMENTS

Legal and Regulatory Requirements for Lightweight Cryptographic Algorithm:
The legal and regulatory landscape surrounding cryptography can be complex and vary depending on your location and the intended use of your lightweight cryptographic algorithm. Here's a brief overview of some key considerations:

Export Controls:
- International regulations like the Wassenaar Arrangement control the export of cryptography with varying levels of restriction. Research the specific regulations in your country to determine any limitations on exporting or distributing your algorithm.

Data Privacy Laws:
- Depending on your target use case, data privacy laws like GDPR (General Data Protection Regulation) or CCPA (California Consumer Privacy Act) might have implications for your project. These laws often mandate specific security measures for protecting personal data, which could influence your choice of encryption algorithm and key management practices.

Encryption Regulations:
- In some countries, specific regulations might govern the use of encryption for certain purposes. For example, some countries restrict the use of strong encryption for financial transactions. Research any relevant regulations to ensure your algorithm complies with local laws.

Intellectual Property:
- If your algorithm is based on existing patented cryptographic techniques, you might need to consider licensing agreements or potential intellectual property infringement.

Disclaimer:
- It's important to include a disclaimer in your documentation stating that your lightweight algorithm is for educational purposes or a proof-of-concept, and established algorithms are generally preferred for real-world sensitive data applications due to their proven security track record.

4.2 ENVIRONMENTAL REQUIREMENTS

The project doesn't have specific environmental requirements.

4.3 DISASTER RECOVERY AND BACKUP

- The cryptographic algorithm itself doesn't require specific backup or recovery procedures.
- Emphasize the importance of backing up your source code, documentation, and any user management databases (if applicable) using appropriate methods like version control and regular backups.

4.4 MAINTAINENACE AND SUPPORT REQUIREMENTS

Maintenance and Support Requirements for Lightweight Cryptographic Algorithm

While the core functionality of a lightweight cryptographic algorithm revolves around logic and algorithms, there are still aspects that require maintenance and support considerations. Here's a breakdown of the key areas to address:

1. Code Maintainability:

Modular Design: Structure your code in a modular and well-organized manner. This allows for easier understanding, modification, and debugging in the future.

Clear Documentation: Write clear and concise comments within the code to explain the purpose of different functions and algorithms. This will aid future developers who need to understand and maintain the codebase.

Coding Standards: Adhere to established coding standards and conventions for the chosen programming language. This improves code readability and maintainability for developers familiar with those standards.

2. Error Handling and Logging:

Implement robust error handling mechanisms to gracefully handle unexpected situations like invalid inputs or encryption failures.

Consider incorporating logging to track system activity and identify potential issues. Logs can be helpful for troubleshooting errors and monitoring system performance.

3. Version Control:

Utilize a version control system (VCS) like Git to track changes made to the codebase over time. This allows for reverting to previous versions if necessary and facilitates collaboration among developers.

4. Testing:

Develop a comprehensive test suite that covers various functionalities of the algorithm, including encryption, decryption, error handling, and edge cases. This helps ensure the algorithm behaves as expected and reduces the risk of regressions when making future changes.