ADVANCED ENCRYPTION ALGORITHM

alexandrubrabete@yahoo.com

TEHNICAL UNIVERSITY OF CLUJ- NAPOCA, SOFTWARE ENGINEERING DEPARTMENT

Advanced Encryption Algorithm Documentation

Overview

This program implements a complex encryption algorithm that utilizes multiple encryption steps and techniques. It incorporates key expansion, value rotation, frequency-based weighting, and a checksum generation mechanism, all bundled into a single program. The core functionality involves encrypting a given string using an encryption key and applying additional layers of complexity to generate a final encrypted result.

Key Features

- **Key Expansion**: The encryption key is expanded to match the length of the input text, ensuring that every character in the text has a corresponding character in the key.
- **Value Assignment**: Each character in the expanded key is assigned a pseudo-random value based on its position and character code.
- **Value Rotation**: The values associated with each character in the key are rotated, adding an additional layer of complexity to the encryption process.
- **Frequency-Based Weighting**: The frequency of characters in the input string influences their corresponding value, altering the final sum calculation.
- **Encryption Modes**: The program supports multiple encryption modes:
    - **Basic**: Adds the base values of the characters.
    - **Weighted**: Adds values along with their positions and frequencies.
    - **Chaotic**: Introduces a more complex summing formula that includes the salt and position.
- **Checksum & Validation**: A SHA-256 checksum and validation code are generated to ensure the integrity of the encryption process.

How It Works

1. User Input

The program begins by prompting the user for the following inputs:

- **Input Text (A)**: The string that will be encrypted.
- **Encryption Key (B)**: The key used to perform the encryption.

- **Case Sensitivity**: Determines whether the case of the characters should be considered during encryption.
- **Encryption Mode**: Defines the encryption method to use:
    - **basic**: Basic encryption.
    - **weighted**: Encryption that incorporates character frequency.
    - **chaotic**: Encryption using a more complex calculation with additional variables.

2. Key Expansion

The encryption key is expanded to match the length of the input text. If the key is shorter than the text, the key is repeated. If it's longer, only the relevant portion of the key is used. This ensures that each character in the input text has a corresponding character in the key.

3. Value Assignment

Each character in the expanded key is assigned a pseudo-random value. The value is calculated using:

- A random number between 1 and 100.
- The ASCII value of the character.
- The position of the character in the key.

These values are stored in a Map<Character, List<Integer>>, where each key (character) is mapped to a list of values.

4. Value Rotation

The values assigned to the characters in the expanded key are rotated by a number of steps, determined by the salt. This introduces additional complexity by altering the order of the values.

5. Weighted Sum Calculation

The algorithm calculates the final encrypted result by summing the values for each character in the input text. The sum is weighted according to the chosen mode:

- **Basic**: Only the value associated with the character is added.
- **Weighted**: The value is added along with the position of the character in the text and its frequency in the text.
- **Chaotic**: A more complex calculation is used that includes the salt value and position of the character.

6. Final Adjustments

The program adds the salt to the final sum to increase randomness.

7. Checksum & Validation Code

- **Checksum**: A SHA-256 checksum is generated from the sum, key, and mode.
- **Validation Code**: A separate validation code is generated using a SHA-256 hash of the key and sum.

8. Output

The program outputs:

- The final encrypted sum.
- A checksum (SHA-256 hash).
- A validation code (SHA-256 hash).
- A step-by-step cryptographic report showing each phase of the encryption process.

Classes

1. EncryptionAlgorithm

This is the main class containing the main method. It handles user input and initiates the encryption process using the EncryptionEngine class.

2. EncryptionConfig

The EncryptionConfig class stores configuration settings for the encryption process:

- **caseSensitive**: A boolean that indicates if case sensitivity is enabled.
- **mode**: The encryption mode selected by the user.
- **salt**: A random integer added to the calculation for additional randomness.
- **debug**: A boolean that determines whether debugging information is logged.

3. EncryptionEngine

The EncryptionEngine class is the core of the encryption logic. It performs the following tasks:

- Expands the key to match the input length.
- Assigns values to characters in the expanded key.
- Rotates values in the Map to increase complexity.
- Computes the weighted sum based on the selected encryption mode.

- Generates the final checksum and validation code.

4. EncryptionResult

The EncryptionResult class stores the result of the encryption process. It includes:

- **finalSum**: The final encrypted sum.
- **checksum**: The SHA-256 checksum.
- **validationCode**: The SHA-256 validation code.
- **logSteps**: A list of log entries detailing the steps performed during the encryption process.

5. KeyExpander

The KeyExpander class is responsible for expanding the encryption key to match the length of the input text. It repeats the key if necessary, ensuring that each character in the input has a corresponding key character.

6. ValueRotator

The ValueRotator class handles the rotation of values assigned to characters. It rotates the lists of values in the Map based on a given number of steps, which is determined by the salt.

7. Utility Methods

There are several utility methods used to:

- Compute SHA-256 hashes (computeSHA256).
- Generate pseudo-random values (generateValue).
- Calculate character frequencies (computeFrequencies).

Usage

1. Input

The program will prompt you for the following inputs:

- **Input Text (A)**: The text that will be encrypted.
- **Encryption Key (B)**: The key used for encryption.
- **Case Sensitivity**: Choose whether the encryption should be case-sensitive.
- **Encryption Mode**: Select the encryption mode (basic, weighted, chaotic).

2. Output

After processing the inputs, the program will display a detailed cryptographic report with the following information:

- Final encrypted sum.
- SHA-256 checksum.
- SHA-256 validation code.
- A log of each encryption step.

Skills Gained

Working through this advanced encryption algorithm offers a range of valuable technical and conceptual skills that are applicable in various fields of computer science and software development. Below are the key skills gained:

1. Understanding of Cryptography Basics

- **Key Concepts**: Gaining a foundational understanding of cryptographic principles, including the importance of key management, randomization (salt), and the role of checksum/validation in ensuring data integrity.
- **SHA-256 Hashing**: Learning how to compute cryptographic hashes using the SHA-256 algorithm, a widely used cryptographic hash function.
- **Encryption & Decryption**: Although this implementation focuses on encryption, the fundamental principles are transferrable to decryption algorithms as well.

2. Randomization Techniques

- **Secure Random Number Generation**: Using the `SecureRandom` class in Java to generate cryptographically secure random numbers. This adds complexity and unpredictability to the encryption process.
- **Salt**: Understanding how salts (random values) are used to add unpredictability to the encryption process, making it more resistant to attacks like rainbow tables.

3. Advanced Java Programming

- **Collections Framework**: Mastering the use of `Map` and `List` collections in Java. Specifically, the use of `Map<Character, List<Integer>>` to store complex mappings and the manipulation of collections using methods like `computeIfAbsent` and `Collections.rotate`.
- **Random Value Assignment**: Implementing the logic to assign pseudo-random values to characters based on multiple factors such as their position, ASCII value, and random values.
- **Data Structure Manipulation**: Understanding the importance of manipulating and rotating lists within a `Map` for adding cryptographic complexity, providing a deeper understanding of how data structures can be used creatively in algorithms.

4. Understanding Encryption Modes

- **Custom Encryption Algorithms**: Designing custom encryption algorithms, including basic, weighted, and chaotic modes, and understanding how different modes can affect the final encrypted result.
- **Mode Selection**: Implementing logic to select and apply different modes of encryption based on user input, and understanding how each mode alters the encryption process.

5. Key Expansion and Management

- **Key Expansion**: Learning how to expand a shorter key to match the length of the input text. This process helps to understand how encryption keys interact with the data they encrypt and why keys need to be handled carefully.
- **Handling of Input and Key Length Mismatch**: Grasping the concept of ensuring the key length aligns with the text to be encrypted, which is a common issue in real-world encryption implementations.

6. Complex Sum Calculation & Weighting

- **Weighted Sum Calculation**: Gaining hands-on experience in calculating weighted sums where the weights depend on factors like frequency of characters, positions in the text, and random values. This teaches how to apply multiple variables to achieve more complex cryptographic transformations.
- **Chaotic Encryption**: Delving into more advanced concepts like chaotic systems in encryption, where non-linear and unpredictable patterns are introduced through salt, positions, and rotations.

7. Debugging and Logging

- **Logging**: Learning the importance of logging and reporting in algorithms. The `logSteps` feature allows step-by-step tracking of the encryption process, which is essential for debugging and understanding complex algorithms.
- **Debugging**: Through step-by-step logging, one can identify issues and understand how each phase of encryption is executed, which improves problem-solving and debugging skills.

8. Cryptographic Integrity and Validation

- **Checksum & Integrity**: Gaining practical experience in ensuring the integrity of data through checksums and validation codes. This teaches why cryptographic integrity is essential in secure communications and data storage.
- **Validation Codes**: Implementing a validation code (SHA-256) that ensures the encryption was performed correctly and that the result can be verified independently.

9. Understanding Real-World Security Practices

- **Security Awareness**: By working with encryption techniques, you learn the challenges of secure data storage and transmission, which are critical in real-world applications like banking, secure messaging, and data protection.
- **Improvement of Security Designs**: Even though this custom algorithm is more for educational purposes, you gain insights into designing more secure encryption mechanisms and understanding the limitations of custom solutions compared to industry-standard algorithms (like AES).

Conclusion

By working through the encryption algorithm, you gain a strong understanding of cryptography, advanced Java programming concepts, and security techniques. You will have learned not only how to design and implement encryption algorithms but also how to enhance their security with custom randomization techniques, modes of encryption, and integrity validation. These skills are highly valuable for anyone interested in cybersecurity, secure software development, or cryptographic research.