Project Report: Security Toolkit Application

I. Introduction / Background

Overview of the Project Goal

The objective of this project is to design and implement a **console-based Security Toolkit** using the Go programming language. The application integrates two fundamental security functionalities: a **classical cryptographic cipher module** and a **file integrity checking module**. The toolkit aims to demonstrate core security principles in a practical and educational manner.

Problem Statement

In basic information security education and small-scale systems, users often lack simple tools to: - Protect message confidentiality through encryption. - Verify whether a file has been altered or tampered with during storage or transmission.

Without these mechanisms, sensitive data may be exposed, and file integrity violations may go undetected.

Proposed Solution

This project provides a unified command-line application that offers: 1. **Message encryption and decryption** using the Rail Fence Cipher. 2. **File integrity verification** using the SHA-256 cryptographic hash function.

The solution emphasizes clarity, modularity, and correctness rather than enterprise-grade security.

Motivation

The motivation behind this project is to: - Apply cryptographic concepts in a real programming environment. - Understand the difference between classical ciphers and modern cryptographic primitives. - Gain hands-on experience with Go's standard libraries for security-related tasks.

Related Cryptographic Concepts

- **Confidentiality**: Ensuring messages are unreadable without the correct key.
- **Classical Ciphers**: Substitution and transposition-based encryption techniques.
- **Cryptographic Hash Functions**: One-way functions used to ensure data integrity.
- **SHA-256**: A secure hash algorithm widely used in modern systems.

II. System Design / Architecture

High-Level Architecture

The system follows a **menu-driven console architecture** with modular components:

- User Interface Layer (CLI Menu)
- Cipher Module (Rail Fence Encryption/Decryption)
- File Integrity Checker (SHA-256 Hashing & Verification)

Each module is implemented in a separate Go source file to improve maintainability and readability.

Data Flow Description

Cipher Module Flow: 1. User inputs plaintext or ciphertext. 2. User provides a numeric key (number of rails). 3. The system applies Rail Fence encryption or decryption. 4. Result is displayed on the console.

File Integrity Checker Flow: 1. User selects hash generation or verification. 2. File is read from disk. 3. SHA-256 hash is computed. 4. Hash is displayed or compared with a known hash. 5. Integrity result is shown to the user.

Design Characteristics

- Modular design using multiple Go files.
- Separation of concerns between UI logic and security logic.
- Reusable cryptographic and utility functions.

III. Implementation Details

Programming Language and Libraries

- **Language**: Go (Golang)
- **Standard Libraries Used**:
- `bufio`, `fmt`, `os`, `strings`, `strconv` for input/output handling
- `crypto/sha256` for hashing
- `encoding/hex` for hash representation
- `io` for efficient file reading

Key Components

1. Main Application (`main.go`)

- Implements the interactive menu system.
- Handles user input and routing to selected modules.
- Ensures continuous execution until the user exits.

2. Rail Fence Cipher

Encryption (`encryptRailFence`): - Implements a transposition cipher using a zigzag rail pattern. - Characters are distributed across rails and read row by row.

Decryption (`decryptRailFence`): - Reconstructs the zigzag pattern using a matrix. - Fills the cipher text into marked positions. - Reads characters following the zigzag path to recover plaintext.

Algorithm Type: Classical transposition cipher (not secure for modern use).

3. File Integrity Checker

Hash Generation (`computeSHA256`): - Reads file content as a stream. - Computes SHA-256 digest. - Returns a hexadecimal string representation.

Verification (`verifyFile`): - Recomputes file hash. - Compares with a known hash value. - Returns a boolean integrity result.

Data Structures

- Slices of `rune` for character manipulation.
- Two-dimensional rune matrices for Rail Fence decryption.
- Strings for hash comparison.

IV. Usage Guide

Prerequisites

- Go compiler installed (Go 1.20 or later recommended).
- Supported operating system: Windows, Linux, or macOS.

Build and Run Instructions

1. Place all `.go` files in the same directory.
2. Open a terminal in that directory.
3. Run the application using:

```
go run *.go
```

Using the Cipher Module

1. Select **Option 1** from the main menu.
2. Choose encryption or decryption.
3. Enter the message.
4. Enter the numeric key (number of rails).

5. View the encrypted or decrypted result.

Example: - Input: `HELLO WORLD`, Key: `3` - Output (Encrypted): `HOREL OLLWD`

Using the File Integrity Checker

Generate Hash: 1. Select **Option 2** → Generate SHA-256 Hash. 2. Enter file path. 3. View the computed hash.

Verify File: 1. Select **Option 2** → Verify File Integrity. 2. Enter file path. 3. Enter known hash. 4. Receive integrity status.

Result Interpretation

- ✅File is intact: No modification detected.
- ❌File has been modified: Integrity compromised.

V. Conclusion and Future Work

Conclusion

This project successfully demonstrates the implementation of fundamental security concepts using Go. The Security Toolkit integrates classical encryption and modern cryptographic hashing into a single, easy-to-use application. It serves as an effective educational tool for understanding confidentiality and integrity mechanisms.

Limitations

- Rail Fence Cipher is not secure against modern attacks.
- No authentication or access control mechanisms.
- Console-based interface only.

Future Work

- Replace Rail Fence Cipher with modern algorithms (AES, RSA).
- Add digital signature support.
- Implement password-based key management.
- Extend to a graphical user interface (GUI).
- Add logging and error handling enhancements.

VI. References

1. William Stallings, *Cryptography and Network Security*, Pearson Education.
2. NIST FIPS 180-4, Secure Hash Standard (SHS).
3. Go Documentation – https://golang.org/pkg/crypto/sha256/