File Encryption and Decryption Tool

Software Requirements Specification (SRS)

Table of Contents

1. INTRODUCTION

1.1 PURPOSE

The purpose of this document is to specify the requirements for the **File Encryption and Decryption Tool**. This tool enables users to securely encrypt and decrypt files using symmetric key cryptography, ensuring data confidentiality and integrity for sensitive files stored on a local system.

1.2 SCOPE

This project delivers a command-line utility for encrypting and decrypting files using the Fernet symmetric encryption algorithm from the cryptography Python library. The tool is intended for personal use to protect sensitive files from unauthorized access.

1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

- **Encryption:** The process of converting data into a coded form to prevent unauthorized access.
- **Decryption:** The process of converting encrypted data back to its original form.
- **Fernet:** A symmetric encryption method from the cryptography library.
- **Key:** A secret value used for encryption and decryption.
- **CLI:** Command Line Interface.

1.4 REFERENCES

- Cryptography Library Documentation
- Fernet Symmetric Encryption
- Python 3 Official Documentation

1.5 OVERVIEW

This document describes the overall requirements, architecture, and design constraints for the File Encryption and Decryption Tool. It details the tool's intended features, user characteristics, interfaces, and specific requirements for implementation.

2. GENERAL DESCRIPTION
2.1 PRODUCT PERSPECTIVE

The tool is a standalone Python application that provides file encryption and decryption capabilities via a command-line interface. It uses the Fernet algorithm for secure symmetric encryption and manages the encryption key locally.

2.2 PRODUCT FUNCTIONS

- Generate and store a symmetric encryption key.
- Encrypt user-specified files using the key.
- Decrypt files previously encrypted by the tool.
- Provide user prompts and feedback.
- Handle errors such as missing files or invalid keys.

2.3 USER CHARACTERISTICS

- Users are expected to have basic familiarity with running Python scripts and using the command line.
- No prior cryptography knowledge is required.
- Users must have read/write permissions for the files they wish to encrypt or decrypt.

2.4 GENERAL CONSTRAINTS

- The tool operates only on files accessible from the local file system.
- The encryption key must be preserved; loss of the key results in permanent loss of access to encrypted files.
- The tool overwrites original files during encryption/decryption.
- Requires Python 3.x and the cryptography library.

2.5 ASSUMPTIONS AND DEPENDENCIES

- Python 3.x and the cryptography package are installed.
- The user will not delete or tamper with the key file (Secret.key).
- The user has appropriate permissions to access and modify files.

3.1 EXTERNAL INTERFACE REQUIREMENTS

3.1.1 User Interfaces

- Command-line interface for user interaction.
- Prompts for operation selection (encrypt/decrypt) and file names.
- Displays success and error messages.

3.1.2 Hardware Interfaces

- No special hardware required; runs on any standard computer with Python installed.

3.1.3 Software Interfaces

- Python 3.x interpreter.
- cryptography Python package.

3.1.4 Communications Interfaces

- No network or external communication required; all operations are local.

3.2 FUNCTIONAL REQUIREMENTS

3.2.1 File Encryption

- The system shall prompt the user for the file to encrypt.
- The system shall generate a new encryption key if none exists.
- The system shall encrypt the specified file and overwrite the original content.
- The system shall save the encryption key in a file named Secret.key.

3.2.2 File Decryption

- The system shall prompt the user for the file to decrypt.
- The system shall load the encryption key from Secret.key.
- The system shall decrypt the specified file and overwrite the original content.
- The system shall display an error if the key is invalid or the file is corrupted.

3.2.3 Error Handling

- The system shall display an error if the specified file does not exist.
- The system shall display an error if the key file is missing when decrypting.
- The system shall display an error for invalid user choices.

3.3 NON-FUNCTIONAL REQUIREMENTS

3.3.1 Performance

- Encryption and decryption operations should complete within a reasonable time for files up to several megabytes.

3.3.2 Reliability

- The tool should reliably encrypt and decrypt files as long as the key is preserved.

3.3.3 Availability

- The tool should be available for use on any system with Python 3.x and the required library installed.

3.3.4 Security

The encryption key must be securely stored and never exposed in plaintext.

- The tool uses Fernet, which provides confidentiality and integrity.

3.3.5 Maintainability

- The code should be modular and documented for ease of maintenance and future enhancements.

3.3.6 Portability

- The tool should run on Windows, macOS, and Linux platforms.

3.4 DESIGN CONSTRAINTS

- The tool must be implemented in Python 3.x.
- The tool must use the Fernet symmetric encryption algorithm from the cryptography library.
- The tool must operate via a command-line interface.