Cryptographic Library & App

Program Overview and User Guide

Caroline El Jazmi, Andy Comfort, Brandon Morgan

Last revised 12/09/2023

Table of Contents

1. Project Report

1.1 Project Implementation Overview

Part 1:
The initial phase successfully implements the SHA-3 derived function KMACXOF256, along with all necessary supporting functions, in adherence to the NIST Special Publication 800-185 guidelines. The core of this implementation, including the SHA-3 function and the SHAKE256 extension, draws inspiration from Markku-Juhani Saarinen's C implementation. This foundational work establishes a robust groundwork for the cryptographic processes central to our project.

Part 2:
Building upon the established foundation, the second phase of development focuses on enhancing the Java-based cryptographic library and application. This phase centers on asymmetric encryption and digital signatures, targeting a 256-bit security level. Central to this phase are the implementations of DHIES encryption and Schnorr signatures, utilizing the Ed448-Goldilocks elliptic curve, all in accordance with NIST FIPS 186-5 guidelines. The execution of this phase demanded meticulous attention to elliptic curve arithmetic and modular operations, aligning with the latest standards in Digital Signature Standard compliance.

1.2 Solution Development

Part 1:
In the initial phase of solution development, we focused on implementing the KMACXOF256 function with a 512-bit output, accommodating various customization parameters. This function computes a plain hash from either a file or direct user input, utilizing an empty key string and a customization string "D". For generating a Message Authentication Code (MAC), the function incorporates a passphrase as the key string and the customization string "T".

Encryption involves multiple invocations of KMACXOF256, synthesizing the key from a passphrase and a random value. This yields two 512-bit keys, used for generating an authentication tag (with customization string “SKA”) and a mask (with customization

string “SKE”). Encryption is achieved by XORing the mask with the input data, with the final output comprising the random value, XORed data, and the authentication tag. Decryption reverses this process.

Part 2:
The second phase advances the project by implementing cryptographic algorithms using the Ed448-Goldilocks elliptic curve. Initially, we developed an elliptic curve arithmetic library, crucial for digital signatures and key generation. This library features implementations of Edwards curve point addition, doubling, and optimized scalar multiplication, including the double-and-add algorithm and pre-computation tables. The subsequent step integrated these elliptic curve operations into cryptographic protocols.

We established a DHIES setup for secure message exchange and a Schnorr signature scheme for digital signatures. The DHIES implementation utilized our elliptic curve library for key exchange, coupled with KMACXOF256 for encryption and authentication. The Schnorr signature scheme leveraged the Ed448-Goldilocks curve's strengths in tandem with KMACXOF256, ensuring secure and efficient signature generation and verification.

1.3 App Services

1.3.1 The application offers the following services:

1. **Generating an elliptic key pair** from a given passphrase and writing the public key to a file.
2. **Encrypting a data file under** a given elliptic public key file and writing the ciphertext to a file.
3. **Decrypt a given elliptic-encrypted file** from a given passphrase and write the decrypted data to a file.
4. **Sign a given file** from a given passphrase and write the signature to a file.
5. **Verify a given data file** and its signature file under a given public key file.

1.3.2 The application contains the following text files:

Located under *src/text_files/*

- ***my-passphrase.txt*** : File for user to manually input and store the password needed to access or decrypt certain files within the program.
- ***my-message.txt*** : File for user to manually input and store Unencrypted message used for encryption and signature generation.
- **private-key.txt** : Contains the encrypted private key.
- ***public-key.txt*** : Contains the public key for encryption and signature verification.
- ***encrypted-message.txt*** : Message content generated by encryption
- ***decrypted-message.txt*** : Original message content after decryption.
- ***signature.txt*** : Contains generated signature

2. User Interface and Instructions

2.1 Getting Started

Before running the application, ensure the following files are correctly populated in the **src/text_files/** directory:

- my-passphrase.txt : Your selected passphrase.
- my-message.txt: The message you intend to encrypt or sign into this file.

2.1.1 Command Line Arguments Set-Up

Execute the application with these command line arguments in their specific order:

1. **First Argument - Passphrase File Path**: ‘src/text_files/my-passphrase.txt’
2. **Second Argument - Message File Path**: ‘src/text_files/my-message.txt’
3. **Third Argument - Public Key File Path**: ‘src/text_files/public-key.txt’
4. **Fourth Argument - Private File Path:** ‘src/text_files/private-key.txt’
5. **Fifth Argument - Encrypted Message File Path**: ‘src/text_files/encrypted-message.txt’
6. **Sixth Argument - Decrypted Message File Path:** ‘src/text_files/decrypted-message.txt’
7. **Seventh Argument - Signature File Path:** ‘src/text_files/signature.txt’

2.2 Main Menu Options

Example of ‘MAIN MENU’ interface.

```
             MAIN MENU
 Please generate an elliptic key pair before
 using encryption, decryption, or signature
 services to ensure secure and valid operations.

 1 - Generate elliptic key pair
 2 - Encrypt data file using public key
 3 - Decrypt elliptic-encrypted file
 4 - Sign a file
 5 - Verify data and signature
 6 - Exit

 Enter digit for mode of operation:
```

The main menu offers six options, covering all available services, including an option to exit the program for convenience.

2.3 Submenu Options

The submenus covers all available submenu services, including an option to go back to the main menu for convenience.

2.3.1 Generate Elliptic Key Pair Submenu

Example of ‘Generate elliptic key pair’ submenu interface

```
Generate Elliptic Key Pair Menu

1 - Generate an elliptic key pair from passphrase
2 - Go to Main Menu

Enter Choice:
```

In this submodule, the user is prompted to input a passphrase via a file path provided in the first command line argument named ***'src/text_files/passphrase.txt'***

Successful execution results in the generation of a public key, which is subsequently stored in ***'src/text_files/public-key.txt'*** and private key is stored in '***src/text_files/private-key.txt'*.** In case of a failure, an error message alerts the user of the unsuccessful operation.

2.3.2 Encrypt Data File Using Public Key Submenu

Example of 'Encrypt data File using public key Submenu' submenu interface.

```
Choose Your Encryption Method

1 - File Encryption (Encrypt a specified file)
2 - Text Encryption (Input and encrypt text manually)
3 - Go to Main Menu

Enter Choice:
```

In this submodule, the user is prompted to encrypt a message by either two modes of input:

- 'File Encryption', allowing the selection of a message file specified in the second command line argument upon initiation of the application.
- ''Text Encryption', enabling direct text entry via the keyboard.

Post encryption, the encrypted message is stored in '***src/text_files/encrypted-message.txt'.*** In case of a failure, an error message alerts the user of the unsuccessful operation.

2.2.3 Decrypt Elliptic-Encrypted File Submenu

Example of 'Decrypted Elliptic-Encrypted FileSubmenu' submenu interface.

```
Decrypt Elliptic-Encrypted File

1 - Decrypt Elliptic-Encrypted File
2 - Go to Main Menu

Enter Choice:
```

In this submodule, the user is prompted to decrypt encrypted data stored '***src/text_files/encrypted-message.txt'.***

Post decryption, the decrypted data is stored in '***src/text_files/decrypted-message.txt***'. In case of a failure, an error message alerts the user of the unsuccessful operation.

2.2.4 Sign a File Submenu

Example of 'Sign a File' submenu interface.

```
Choose Your Signature Generation Method

1 - File Signature (Generate signature for a specified file)
2 - Text Signature (Input and generate signature for text manually)
3 - Go to Main Menu

Enter Choice:
```

In this submodule, the program deals with the generation of cryptographic signatures, effectively associating a user-defined passphrase with a message.

It permits message input from user by one of the following:

1. 'File Signature', allowing the selection of a message file specified in the second command line argument upon initiation of the application
2. 'Text Signature', enabling direct text entry via the keyboard

Successful signature creation leads to its storage in '***src/text_files/signature.txt*** '. In case of a failure, an error message alerts the user of the unsuccessful operation.

2.2.5 Verify Data and Signature Submenu

Example of ‘Verify data and signature’’ submenu interface.

```
Choose Method of Signature Verification

1 - File Verification (Verify signature of a specified file)
2 - Text Verification (Input and verify signature for text manually)
3 - Go to Main Menu

Enter Choice:
```

In this submodule, the program verifies the authenticity of a signed message using the stored public key and the corresponding signature file.
It permits message input from user by one of the following::

1. ‘File Verification', allowing the selection of a message file specified in the second command line argument upon initiation of the application
2. ‘Text Verification', enabling direct text entry via the keyboard

Whether the operation is successful or unsuccessful, is clearly communicated to the user via console messages.