Practical Project Report

Andrew Lim

August 2020

Introduction

This app offers 10 different services:

1. Compute a plain cryptographic hash of a given file.
2. Compute a plain cryptographic hash of text input.
3. Encrypt a given data file symmetrically under a given passphrase.
4. Decrypt a given symmetric cryptogram under a given passphrase.
5. Compute an authentication tag (MAC) of a given file under a given passphrase.
6. Generate an elliptic key pair from a given passphrase and write the public key to a file.
7. Encrypt a data file under a given elliptic public key file.
8. Decrypt a given elliptic-encrypted file from a given password.
9. Sign a given file from a given password and write the signature to a file.
10. Verify a given data file and its signature file under a given public key file.

SHAKE.java is the implementation of cSHAKE256 and KMACXOF256. SymmetricCryptography.java and EllipticCurveCryptography.java contain the solution to each part of the practical project. To use the app, run CryptographicApp.java. The app will then display the above services in the console. Select a service by entering the number associated with it into the console.

1 Compute a plain cryptographic hash of a given file

1. Select a file when prompted by the app.
2. The plain cryptographic hash will be printed onto the console.

2 Compute a plain cryptographic hash of text input

1. Enter text into the console when prompted by the app.
2. The plain cryptographic hash will be printed onto the console.

3 Encrypt a given data file symmetrically under a given passphrase

1. Select data file when prompted by the app.
2. Enter a passphrase into the console when prompted by the app.
3. The same file will be encrypted in the same location it was selected at.

4 Decrypt a given symmetric cryptogram under a given passphrase

1. Select symmetric cryptogram when prompted by the app. This is the file that was encrypted in the previous service.
2. Enter the passphrase into the console when prompted by the app. This is the same passphrase used to encrypt the file in the previous service.
3. The cryptogram will be decrypted in the same location it was selected at.

5 Compute an authentication tag (MAC) of a given file under a given passphrase

1. Select a file when prompted by the app.
2. Enter the passphrase into the console when prompted by the app.
3. The authentication tag will be printed onto the console.

6 Generate an elliptic key pair from a given passphrase and write the public key to a file

1. Enter a passphrase when prompted by the app.

2. Enter a file name into the console to be saved as when prompted by the console. No need to include the file type in the name. The file will be automatically saved as a .public_key file. The elliptic key pair will be saved into this file. The file will be saved in the same directory where CryptographicApp.java is located in.

7 Encrypt a data file under a given elliptic public key file

1. Select a data file when prompted by the app.
2. Select an elliptic public key file when prompted by the app. This is the .public_key file generated in the previous service.
3. The same data file will be encrypted in the same location it was selected at.

8 Decrypt a given elliptic-encrypted file from a given password

1. Select an elliptic-encrypted file when prompted by the app. This is the encrypted file from the previous service.
2. Enter the password when prompted by the app. This is the same password used to encrypt the file in the previous service.
3. The cryptogram will be decrypted in the same location it was selected at.

9 Sign a given file from a given password and write the signature to a file

1. Select a file when prompted by the app.
2. Enter a password when prompted by the app.
3. Enter a file name into the console to be saved as when prompted by the console. No need to include the file type in the name. The file will be automatically saved as a .signature file. The signature will be saved into this file. The file will be saved in the same directory where CryptographicApp.java is located in.

10 Verify a given data file and its signature file under a given public key file

1. Select a data file when prompted by the app.
2. Select the data file's signature file when prompted by the app. This is the .signature file generated in the previous service.
3. Select the public key file when prompted by the app.
4. The console will output true if the signature is valid and false otherwise.

Works Cited

SHAKE.java by Markku-Juhani Saarinen (original Keccak and Model.SHAKE implementation in C) and Paulo S. L. M. Barreto (Java version, cSHAKE, KMACXOF).