TCSS 487 - Cryptographic Project Report

Author: Joseph Yun

This project implements (in Java) a library and an app for asymmetric encryption and digital signatures at the 256-bit security level.

```
1) Compute a cryptographic hash of a given file or input text.
2) Encrypt or decrypt a given file symmetrically under a passphrase.
3) Generate an elliptic key pair file from a passphrase.
4) Encrypt or decrypt a given file under an elliptic public key file.
5) Sign or verify a given file.
6) Quit.
```

Usage:

Simply run 'CryptographicApp.main()'.

Services offered by the app & how to use:

- [10 points] Compute a plain cryptographic hash of a given file (this requires implementing and testing cSHAKE256 and KMACXOF256 first).

```
Choose one of the sub-options below:
    1) Compute a cryptographic hash of a file.
    2) Compute a cryptographic hash of an input text.
    3) Go back.

Enter a numbered option: 1
Waiting for file selection...
The file "test1.txt" has been hashed: 3b478ac3ecd7d5cfaa20db279b79bd89744d5f2210874f806ba5936874c13395af66498961eca999da82e5764db47d240c1fff15b1fdf239e2b115ae77e9397d
```

1. Choose option 1-1
2. Select desired file to hash (a pop-up file dialog will appear).

- BONUS: [4 points] Compute a plain cryptographic hash of text input by the user directly to the app instead of having to be read from a file.

```
Choose one of the sub-options below:
    1) Compute a cryptographic hash of a file.
    2) Compute a cryptographic hash of an input text.
    3) Go back.

Enter a numbered option: 2
Enter the string to hash: hello
The text "hello" has been hashed: 5d1326841e932dc7a27508b059d371ec711971e7916f46eb93450257692eff23fc458f00dd28c5bc44404f4c1cbb22cf7ab562a098ad831773c7b31c80be78b8
```

1. Choose option 1-2
2. Enter desired text to hash.

- [10 points] Encrypt a given data file symmetrically under a given passphrase.

```
Choose one of the sub-options below:
    1) Encrypt a file symmetrically under a given passphrase.
    2) Decrypt a symmetric cryptogram under a given passphrase.
    3) Go Back.

Enter a numbered option: 1
Enter a passphrase: password
Waiting for file selection...
Encryption successful. test1.txt has been encrypted to: 72616e646f6d2074657874
```

1. Choose option 2-1
2. Enter desired passphrase.
3. Select desired file to encrypt (a pop-up file dialog will appear).

- [10 points] Decrypt a given symmetric cryptogram under a given passphrase.

```
Choose one of the sub-options below:
    1) Encrypt a file symmetrically under a given passphrase.
    2) Decrypt a symmetric cryptogram under a given passphrase.
    3) Go Back.

Enter a numbered option: 2
Enter a passphrase: password
Waiting for file selection...
Decryption successful. test1.txt.cryptogram has been decrypted to: 72616e646f6d2074657874
```

1. Choose option 2-2
2. Enter required passphrase.
3. Select .cryptogram file to decrypt (a pop-up file dialog will appear).

- [10 points] Generate an elliptic key pair from a given passphrase and write the public key to a file.

```
Choose one of the options below:
    1) Compute a cryptographic hash of a given file or input text.
    2) Encrypt or decrypt a given file symmetrically under a passphrase.
    3) Generate an elliptic key pair file from a passphrase.
    4) Encrypt or decrypt a given file under an elliptic public key file.
    5) Sign or verify a given file.
    6) Quit.

Enter a numbered option: 3
Enter a passphrase: password
Waiting for user to specify output directory...
Private Key: -26085180719205745096643981018084016499844591356984133682862309994922931894576582109636666429465456573624616417632781258662265736210427971923379776930882036
Public Key X: 136587025557703422181246305415664122694067442069578855981708572147937180638098890062375728872260427931112112540975560956651515327162299288693082916646708650
Public Key Y: 668357828089355414378739872592686345661682424835093543315337820439525818206031986592741504709413956750846281461807238904805920174666429621522484910162460628 7

The public key file has been saved to: C:\Users\Joseph\IdeaProjects\tcss487\test\pk_passphrase=password
```

1. Choose option 3

2. Enter desired passphrase.
3. Select desired output directory (a pop-up file dialog will appear).

- **[10 points] Encrypt a data file under a given elliptic public key file.**

```
Choose one of the sub-options below:
    1) Encrypt a data file with a public key file.
    2) Decrypt a .ECC file with a passphrase.
    3) Go Back.

Enter a numbered option: 1
Waiting for public key file selection...
Waiting for data file selection...
Encryption successful: 6576656E206D6F72652072616E646F6D2074657874
```

1. Choose option 4-1
2. Select public key file.
3. Select data file to encrypt.

- **[10 points] Decrypt a given elliptic-encrypted file from a given password.**

```
Choose one of the sub-options below:
    1) Encrypt a data file with a public key file.
    2) Decrypt a .ECC file with a passphrase.
    3) Go Back.

Enter a numbered option: 2
Enter a passphrase: password
Waiting for .ECC file selection...
Decryption successful: 6576656E206D6F72652072616E646F6D2074657874
```

1. Choose option 4-2
2. Enter required passphrase.
3. Select .ECC file to decrypt (a pop-up file dialog will appear).

- **[10 points] Sign a given file from a given password and write the signature to a file.**

```
Choose one of the sub-options below:
    1) Sign a given file under a passphrase and write the signature to a file.
    2) Verify a given file and its signature file under a public key file.
    3) Go Back.

Enter a numbered option: 1
Enter a passphrase: password
Waiting for file selection...
Signature file successfully generated.
```

1. Choose option 5-1
2. Enter required passphrase.
3. Select file to sign (a pop-up file dialog will appear).

- **[10 points] Verify a given data file and its signature file under a given public key file.**

```
Choose one of the sub-options below:
    1) Sign a given file under a passphrase and write the signature to a file.
    2) Verify a given file and its signature file under a public key file.
    3) Go Back.

Enter a numbered option: 2
Waiting for public key file selection...
Waiting for file selection...
Waiting for signature file selection...
Signature is illegitimate.
```

1. Choose option 5-2
2. Select public key file.
3. Select signed file.
4. Select .Signature file.

Works Cited

- Java implementation of SHA3 and SHAKE256 (SHA3.java) based on Markku-Juhani Saarinen's C implementation.
- EllipticCurvePoint.sqrt() taken from '**Appendix: computing square roots modulo p**' in the assignment specification.