Project Report

Instructions:

1. Open java project and change all output file names to the absolute path names that you would like to store on your computer.

Output paths are located in the following classes:

EnDeEllipticCurve.java

EnDeCrypt.java

There are a few different output paths per class. If unchanged they will likely print to the src file, but there is a possibility that they will not print if a file with the same name is on your computer.

The names for input files will be prompted in the dialog screen, the absolute path name starting with C:// must be used in order for the program to read test files.

2. Run the Main.java file.

3. All UI is handled via the console. This app is a console-based interaction app. The user can simply run it as a Java Application. The app was written on eclipse, there may be unknown encoding errors when run on other IDE's.

4. What will be displayed is the Main Menu where the user will have the choice to either do a symmetric cryptography or an elliptic curve arithmetic. If the user chooses symmetric cryptography, they have a total of 5 options: compute a plain cryptographic hash by either a given file or a user's input, encrypt or decrypt a given file, and compute an authentication tag of a given file that is under a given passphrase. If the user chooses elliptic curve arithmetic, they have a total of 7 option: generate an elliptic key pair from a given passphrase or encrypt the private key under a given password and write either of these data to a file, encrypt or decrypt a user's input, sign a given file from a given password and write the signature to a file and verify a given file and signature under a given public key file.

Known Bugs:

Signature verification implementation is unfinished. All other methods are finished. Extra credit is completed for all of part 1.

Other bugs may exist, parts of the program remain untested particularly in signature generation.

Class Implementation:

- KECCAK.java
    - This class was provided in the assignment specification as a C file. The translation from C to Java was referenced from the office hours meetings.
- cSHAKE256.java
    - This class was based off the NIST Specification given to us in the project description
- KMACXOF256.java
    - This class was based off the NIST Specification given to us in the project description
- Supporting Function (left_encode, right_encode)
    - These functions were based off the NIST Specification page
- Supporting Function (bytepad, encode_string)
    - These functions were based off the NIST Specification page
- EllipticCurve.java
    - Most information was given under the Elliptic Curve Arithmetic section in our project description.
    - The square root calculation was given in Appendix: computing square roots modulo p
    - The scalar multiplication was given in Appendix: multiplication by scalar
- EnDeCrypt.java
    - Will accept a file and or passphrase and either compute, encrypt, or decrypt and write to an output file

- EnDeEllipticCurve.java
    - Generate an elliptic key pair from a given passphrase and write the public key to a file

For our elliptic curve class, we based upon the given appendix compute square and appendix multiplication code. This class allows us to Encrypt and decrypt a given elliptic-encrypted file from a given password

- Part 1 BONUS problems

Our cSHAKE25s and KMACXOF256 can:

1. Compute a plain cryptographic hash of text input by the user directly to the app instead of having to be read from a file.

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

Our Elliptic key can:

1. Encrypt the private key under the given password and

write it to a file as well.

2. Encrypt/decrypt text input by the user directly to the app instead of having to be read from a file.

We came about this using the high-end specs of the document given and writing our code using it as inspiration.

Cryptogram and Signature are both objects that contain the elliptic curve cryptogram and signature object respectively. They are used to store and recover the encryptions.

Other Notes

Our project is also not dynamic, we need to implement the actual path to file which

want to Encrypt or Decrypt

Sources:

NIST inspiration

https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

SHA-3 inspiration

https://dx.doi.org/10.6028/NIST.SP.800-185

Markku-Juhani Saarinen's C implementation:

https://github.com/mjosaarinen/tiny_sha3/blob/master/sha3.c