TCSS 487 Cryptography

Practical Project – Part 2

Justin Goding, Yeseong Jeon, Andrew Lau

Solution:

Controller code gathers the appropriate arguments for the requested functionality and then executes the appropriate method. There are 5 core methods to implement the Schnorr/DHIES functionality, keyPair() which generates the elliptic key pair, encrypt(), decrypt(), sign(), and verify(). These methods follow the high level specifications given in the assignment almost exactly. With variances for the processing of byte arrays and Java BigIntegers. The Ed448GPoint class was implemented to handle the creation of Edwards Elliptic Curve points and all arithmetic operations on them using Java BigIntegers. For writing a public key to a file which is an Edwards point, the class contains a method call getBytes() which turns the BigInteger value for x into a byte array, left pads it with zeroes to be of length 57, then adds a 0 or 1 to the end on the right for if the y value is even or odd. The class also contains a method called pointFromBytes() which takes in a byte array and reverses the process to get an Edwards point.

User Instructions:

Running the program will bring up a CLI prompting the user to input which part of the app they want to access, '1. Symmetric' for the features of the first part of the project, or '2. Asymmetric' for the features of this part of the project.

Once '2' is selected, the options listing part 2 functionality will be printed to the screen:

1. Generate a (Schnorr/DHIES) key pair

2. Encrypt a given data file or message

3. Decrypt a given data file

4. Generate a signature for a given data file or message

5. Verify a signature for a given data file

6. Exit

Selecting options 2 or 4 will ask the user if they want to pass a file name containing the data/message (option 'a') or input the message to the console directly (option 'b').

Then the required data will be prompted from the user, the CLI text will specify what is being requested at each step.

For example, if we pick option 5, the following message will be displayed:

```
>> Option 5:"Verify a signature" selected


- Requirement 1: Message File -


Please enter the name of the file below:
```

Once the user enters the name of the message file, such as 'message.txt' it will ask for the next required information: the file containing the signature to be verified, and so on until it has all necessary information.

NOTE: ANY ENTERED FILES MUST BE IN THE JAVA WORKING DIRECTORY

Outputs:

Option 1 will write the public key and private key on separate lines in hexadecimal to the console in that order and also write the bytes to separate text files: 'publicKey.txt' and 'privateKey.txt' respectively.

Option 2 will write the ciphertext in hexadecimal to the console and the bytes to the file: 'encryption.txt'

Option 3 will write the decrypted text in plain English to the console and to the file: 'decrypted.txt'

Option 4 will write the signature in hexadecimal to the console and the bytes to the file: 'signature.txt'. If the user inputs the message directly to the console, then it will also write the message to 'message.txt'.

Option 5 will simply write to the console whether the given signature was valid for the given data file and passphrase.

Citations:

- The Ed448GPoint.multiply() method was implemented from the pseudocode for Double-and-add, Iterative algorithm, index increasing from this page: https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication

- Some code borrowed from Professor Paulo Barreto