# Cryptography Project Part 2 Report

*TCSS 487 | Spring 2023 | University of Washington - Tacoma*
*Team members: Christopher Henderson, Lindsay Ding, Alan Thompson*

## Development process description

For the second part of our project, we have initiated the implementation of the Point class, following the specifications outlined in the project description. Our initial progress involved implementing three constructors, then the methods to verify equality between points, obtain the opposite point, and calculate the sum of two points on the curve. In one of the constructors where an x-coordinate and the least significant bit of y are provided as inputs, we used the sqrt method from Appendix A to generate the corresponding y-coordinate.

Once the implementation of the Point class was completed, we then started to work on the scalar multiplication method. To implement multiply by scalar, we referred to the pseudo code provided in Appendix B as well as the lecture slides for guidance and reference. For testing and debugging, we generated the public generator G using x = 8 and the least significant bit of y = 0. Subsequently, we proceeded to evaluate G against all arithmetic properties listed under Appendix C to ensure the correctness. After confirming the correctness and functionality of scalar multiplication with all the properties, we then worked to implement a basic command-line interface for performing all of the different functions required by the project.

## Usage description/instructions

*NOTE: all files the user names are assumed to be in the 'files' sub folder*

**The user interaction has 13 numbered options which each perform a main operation independent of other operations/options.**

**The first 6 of those are just the options for part 1 of this project.**
**For Options 1-6 see Project Part 1 Report**

### Main User Options:

- **7 |** ***Generate an elliptic key pair from a given passphrase***
  - User is prompted to type in a passphrase
  - User input the filename for the public key file to be saved
  - User is asked if they will save a encrypted private key to file as well

        - valid responses are {yes, no, y, n 1, 0} for yes or no
        - if yes user is asked to provide the filename for that key
        - The private key is symmetrically encrypted using the encryption Part 1

- **8 |** ***Encrypt a data file under a given elliptic public key file***
    - User is asked to give a filename of the file they wish to encrypt
    - Then asked to enter filename of the elliptic public key they wish to encrypt their file with
    - Lastly they are asked to input the filename of the saved encrypted file.
        - **FunTip**: If you enter the same filename for both inputs, the original file will be overwritten by its encrypted version!
- **9 |** ***Encrypt your console input under a given elliptic public key file***
    - User inputs the text they wish to be encrypted
    - Then enter the the name of the public key file
    - Enter the file name to save the encrypted file
    - Ciphertext will be saved under files' folder with the file name given by the user
- **10 |** ***Decrypt a given elliptic-encrypted file from a given password***
    - User inputs the name of encrypted file
    - Then enter the given password (the private key)
    - Enter the file name to save the decrypted file with file extension
    - Decrypted message will be saved under files' folder with the file name given by the user
        - **FunTip**: If you enter the same filename for both inputs, the original file will be overwritten by its encrypted version!
- **11 |** ***Sign a given file from a given password***
    - User inputs the data file name they wish to sign (with file extension)
    - Then enter the given password (the private key)
    - Enter the file name to save the signature with file extension
    - Signature will be saved under files' folder with the file name given by the user
- **12 |** ***Sign your console input text from a given password***
    - User inputs the text they wish to sign
    - Then enter the given password (the private key)

- Enter the file name to save the signature with file extension
- Signature will be saved under files' folder with the file name given by the user

- **13 |** ***Verify a given data file and its signature file***
  - User inputs the data file name they wish to verify the signature (with file extension)
  - User inputs the signature file name with file extension
  - User inputs the public key file name with file extension
  - Successful verification will output "The file signature matches with the given public key." to console. Otherwise, the output will indicate the signature does not match.
    ***Note*: The public key used to verify signature should be in the same key pair as the private key used to sign the document***

**User entered filename handling:**

If the user enters an invalid filename they will be asked to enter a valid name. Additionally if the user inputs the name of a file that doesn't exist they will be prompted to try again.

**Non-Text files can be encrypted too, but only if that are not very large