Vision:

The final vision for the project is a secure, email-esque program that allows users to send encrypted messages to each other without fear of hackers or men-in-the-middle accessing plaintext messages. Our goal was to create a program to facilitate the implementation of the Diffie-Hellman Key Exchange algorithm and a cipher or encryption system that used a shared key, and we decided that a message sending program would best fit that goal. Each user's account is stored in a JSON file and will have a public key and a private key, where the public key is accessible by all other users and the private key is not. When User A wants to send a message to User B, A uses B's public key and A's private key to encrypt the message using the Diffie-Hellman algorithm and a cipher using the key from the DH key exchange, and sends the message. The message is stored in another JSON file in the encrypted form, so that even if the JSON file is compromised, no plaintext messages will be recoverable. When B opens the message, the message will automatically be decrypted through the Diffie-Hellman algorithm using B's private key and A's public key.

Summary:

We have completed all aspects of the project. Since MS2, we completed the Diffie-Hellman Key Exchange algorithm, which included many hours of researching and understanding mathematical concepts like prime factors and primitive roots, and completed the user interface and JSON file storage system. We also completed the implementation of both the Caesar Cipher and the Affine Cipher. The user interface now includes the option to open the user's inbox and outbox and view any message in both, sending a message, and the ability to go "back" a page, so for example, if a user opens their inbox, then a message from their inbox, they can go back to the inbox, then back to the home page. The user interface has remained terminal based and still uses the read_line function to process user input. New usernames and passwords are now required to be at least eight characters to prevent interference with navigational inputs, eg: "I" for inbox and "B" for back, and to increase account security. We also spent some time implementing the time component of the messages, so that a user will know at what time a message was sent. We used the Time_Now module, which gives the current time in an Epoch timestamp as an Int-63 value, and we converted it into an integer, then converted Epoch time into the American standard convention date and time format. We also made sure to catch all bad inputs with an error message and prompt to enter another input so as to not crash the system with bad input. We also decided on JSON files to store user accounts and messages. We made two separate JSON files: one for users and one for messages. When a user or message is created, it gets put into a hashtable of its respective kind, and is then read into a JSON file. When a user logs in or wants to view a message, the JSON file is read and converted back into a hashtable so that the relevant information can be accessed. We first convert the data into JSON acceptable strings, then use Yojson functions to convert the data to and from the JSON files. Finally, we also created our test suite and ensured that all of our tests passed.

Most of our initial testing was in Utop and in using the user interface, so creating the test suite allowed us to black and glass box test all of our functions to be sure there were no bugs hiding in our code.

Productivity:

As a team, we were more or less as productive as we originally planned to be. All of the functionality we were hoping to submit has been accomplished, and we have achieved the required lines of code for the project. We even implemented two separate encryption algorithms, and we were initially only planning on doing one. Group sprints were very productive, because we were able to do the most collaboration during those times and work together to fix bugs and figure out the optimal ways to implement the various parts of the program. The bulk of the work for MS3 was done in group sprints. Individual or smaller group sprints were also productive, and were often used to finish the work started in a group sprint. Overall, we stayed on schedule well and got everything done on time.

Activity Breakdown:

Activity participation for all members:

- Group planning sessions
- Group coding sprints
- Individual coding sprints
- PM and Demo meetings

Matt Lim:

Responsibilities and features delivered:

- Created and implemented Caesar encryption algorithm
- Helped debug JSON file conversion and user interface

Hours working: 35

Chris Johnson:

Responsibilities and features delivered:

- Implemented time feature for messages
- Created test suite, tested the bulk of the functions in the D-H key exchange algorithm
- Helped debug helper functions in the D-H key exchange algorithm

Hours working: 32

Andre Foster:

Responsibilities and features delivered:

- Implemented the remainder of the user interface
- Created JSON file/Hashtable conversion system

Hours working: 37

Sarah Feng:

Responsibilities and features delivered:

- Created and implemented D-H key exchange algorithm
- Wrote Affine Cipher algorithm
- Integrated encryption and decryption into message sending and receiving
- Wrote report and YAML file

Hours working: 34