For our project, we made an application for communication between client and server. We wanted to learn more about the security aspects of computer networks, so in addition to having the same functionalities as a Java sockets API, we added in encryption for messages over the network. The encryption method we used is the Advanced Encryption Standard (AES), which has a history of getting adopted first by the U.S. government for sensitive information and is used world-wide today. AES uses block cipher, meaning that compared to stream cipher in which one bit is processed at a time, it looks at a chunk of data at a time. This is useful when the data size is known. Also, in order to avoid encrypting the same block of data the same way, it uses the ciphertext produced in the previous block for the next block for additional security. AES also uses symmetric cipher, which means it uses the same key to encrypt as well as decrypt and although the most secure uses 256-bit keys, for our demonstration we used 128-bit keys. 128-bit keys are still secure in the sense that in order to guess the key correctly using brute force, it would take an enormous amount of computing power like quantum computing. The logic for this encryption and decryption is written in AES.py. Setting up the keys requires a process between a server and a client called Transport Layer Security (TLS) handshake, which is a cryptographic protocol with applications in the web, emails and so on. During this process, the server and client communicate to establish a private connection. There are a few phases in TLS handshake. The first is the negotiation phase in which the client first sends a message to the server with information such as which protocol version it can support and a random number, to which the server responds with similar information. If this is done successfully, the server sends a certificate to the client, and the client can use the public key in the certificate to encrypt and send a secret back to the server. Then, the client and server use the random numbers generated and this secret to create a common secret for the connection. Then, the client sends a “finished” message which the server uses the secret to decrypt and verify the client’s identity and the server sends a similar message to the client for the client to do the verification. Once this is complete, a secure connection is successfully established an the server and client can send messages back and forth while encrypting and decrypting using the same secret key. It is also important to note that this negotiating process is secure even against attackers who manage to place themselves in the middle of the server and client and the attackers cannot modify any exchanged information without getting detected. Furthermore, each session creates a unique key. The logic for this handshake is contained in TLSHandshake.py. During the setup, it will print out messages for the steps I defined above like “sending client_random:” and “Generated session_key:”. The application was written in Python and Daniel wrote the Advanced Encryption Standard (AES) code as well as making a video demo. Kevin wrote the code for the server and client to be able to connect and send messages to each other and wrote the report. Micah then wrote the logic for setting up the 128-bit session keys for AES by implementing TLS handshake as well as cleaning up the code in server and client.