Michelle Tran, mitr7803@colorado.edu
Nathan Reed, nare6880@colorado.edu
Neel Karsanbhai, neka4836@colorado.edu

1. The salt for each of our passwords is 32 bytes long. Each salt is randomly generated using os.urandom(), which is a function in the os library. Each time a user tries to add a login, the user-entered password is concatenated with a randomly generated salt. The concatenated string is then hashed with SHA-265. The library, hashlib has a function to hash strings using SHA-256. In an offline attack, our passwords would be secure, because the attacker would need to combine each possible password with every salt, and then hash the value (password and salt). This would make a rainbow table attack infeasible due to the large number of passwords an attacker would need to test.

2. To generate our public/private key pair, we used ssh-keygen. In our python files, we hardcoded the private, and public keys. This is yet another vulnerability, as the private key should not be known to anyone. Normally keys would be managed with PKI, but we obviously do not have that.

3. To encrypt our messages, we used AES.MODE_ECB. This encrypts the message using the Electronic Codebook. Using ECB creates a vulnerability in our system, because an attacker can use a replay attack, since ECB doesn't cover up patterns. We still feel safe in using ECB because a lot of the data that is being sent will not have detectable patterns.

4. Since our encryption and decryption uses ECB, this could allow an attacker to eavesdrop and guess the content sent. Because ECB mode produces the same result for identical encrypted blocks, the attacker could guess the encrypted message (identical plaintext blocks will result in identical ciphertext).

5. Our program could be vulnerable to replay attacks because we used ECB. But we believe that it is somewhat secure from replay attacks because the session key is random everytime, so replay attacks might not be possible in this case.

6. We learned how to generate key pairs and how to communicate between server and client. We also learned how to generate RSA keys by using ssh-keygen. We had to do a bit of research between ECB and CBC to see which mode to use when encrypting and decrypting. Although we started with CBC, we couldn't get it to work so we had to switch to ECB, which has many vulnerabilities compared to CBC.