Jacob Christiansen
Nick Price
Devin Murray

1) How did you hash the passwords in the file? Include information such as the hashing algorithm, the library you used, how you appended the password with the salt before hashing, and other information we've discussed related to password storage. Please provide brief explanations as to why you chose to use what you did. Convince the reader that your password storage algorithm is secure.

Hashlib, os, and random are the 3 external libraries used. A salt is created using os.urandom. This creates a string of, in our case, 16 bytes. This method has been shared on the internet to be reasonably random and if the internet does not have a known solution, vetted or otherwise, it should be deemed reasonable for the time being. Python allows for string concatenation so the salt is added to the end of the password because the end is better than the beginning. A hashing step is taken using the hashing.sha1 method. The parameter used is an encoded version of the recently salted password. The sha1 method was used because it was developed by the NSA and is widely regarded as a robust hashing algorithm. Hexdigest is used because it is another highly regarded source in transmuting an object into a string of hexadecimal more easily used with other functions, methods, and other non-binary environments such as ours.

2) Describe your approach to public/private key use. How did you generate your public/private key pair? What library did you use to encrypt messages? In which folder did you store your keys? Please provide any details you think would be relevant.

We generated our keys using a python program we found on https://www.dlitz.net/software/pycrypto/api/current/Crypto.PublicKey.RSA-module.html that utilizes RSA from the Crypto.PublicKey library. The file we used to run the code is in our project as "keygen_rsa.py". It output .pem files which we named "pub_key.pem" and "priv_key.pem". The public key was stored in our "client" folder and the private in the "server" folder.

3) How did you manage symmetric encryption? What encryption mode did you use, and why did you think that was best? What tradeoffs did you make for that encryption mode?

The well-known tradeoff to either RSA or AES encryption is in memory and key access. Memory wise, a symmetric system is superior. It does not have the issue of data expansion or any significant toll on thread usage (we didn't use a massive key so memory is not a real issue).

We chose a hybrid method. It was spoken about by peers and upon very little research here (https://crypto.stackexchange.com/questions/10685/hybrid-encryption-with-rsa-and-aes-versus-spliting-into-multiple-rsa-messages) we can see almost the best of both worlds. RSA and AES were used from the crypto public and cipher libraries respectively. The initial handshake is done with RSA while this session key is then asymmetrically encrypted to send a message. This is done so that a message is not constrained by a massive data bomb while the handshake is primarily just the key so size, albeit not static, is less of a concern over a message that can change from use to use.

4) Provide a brief explanation of why your program would be secure from eavesdroppers if you were to run it on a publicly visible network, from start to finish. Write your program like you're protecting yourself from Comcast or the NSA!

If someone were sniffing some ports in Norlin Library while we shared messages and added user accounts our information would be safe given a few assumptions. The first would be that the user already exists. It should not be expected for any individual to be able to overcome our salting and hashing of passwords with reasonable resources. This leaves only the NSA as a potential threat as any student or employee of Comcast should not have the hardware required to attack a user with a properly saved password (as described in Q1). The NSA should have difficulty compromising the integrity of our system. Our program clearly implements a handshake method that allows a public key to be encrypted and returned. Without knowing how our session key is formed, how that key is encrypted, and what else may have been done to this key, there is very little a malicious figure can do to break the integrity of our program.

5) Is your program secure from replay attacks? If not, why not? Use protocols that we discussed in class to show that it isn't vulnerable. If it is vulnerable, what could you do to prevent it? Could you perform a replay attack against yourself?

Replay attacks are possible with our program. It most likely won't go well because of secure encryption methods, a handshake, keys, and it barely working even when proper info is input as is. The question of attempting attack against ourselves leads to a form of a replay attack that may work. We have no checks over duplicate messages besides our port connection. If an attacker were to compromise a user they would not need to know what a message was or how to encrypt it but wait until a message with valuable info was sent and ask for a second response to the same user which they've compromised.

6) What else did you learn from the project? Did you have to do some research? Collectively, or individually, give some of the key takeaways you had when doing this.

https://docs.python.org/3/library/hashlib.html
https://www.pythoncentral.io/hashing-strings-with-python/
https://albertx.mx/https-handshake/
https://docs.python.org/3/library/os.html
https://stackoverflow.com/questions/9594125/salt-and-hash-a-password-in-python

These are most of the sources used to research encryption methods that are not only neat and proper in the eyes of a developer but considered adequate in the eyes of the internet and industry when dealing with password encryption and decryption. The work was divided which made individual research easy while merge conflicts were a little more difficult. We learned that it's not too difficult for a regular individual to apply protocols and methods commonly found in those protocols such as hashing and salting that result in what seems to be a production-ready solution. At the same time having a server accept or actively reject a connection is truly the first component of most programs. Once a connection is established we have little knowledge or teachings in what and how things can be done with this connection but any program utilizing and requiring a secure connection (such as a login system) has its foundation here.