CS355 Project Spec

Github: https://github.com/spencerdouglas7/cs355_project

Instructions:

1. **Make sure you are using python3**
2. **Install libraries (mentioned below)**
3. **Run filecreate.py**
4. **Run Server.py**
5. **Run Client.py**

Languages/Libraries:

- Python 3.9
- Python ‘secrets’ library
- PyCryptodome library (RSA encryption and digital signatures, SHA-256 hashing)

Protocol:

Remark: In our implementation, we exchange keys on the wire. This is obviously bad practice, but it fulfills the purposes of key exchange to set up the scenario as it has been outlined in the assignment. ***For the purposes of your attacks, you may not leverage this; consider our protocol as beginning when Alice and Bob begin exchanging their random nonces.***

Alice and Bob each hold a passwords file, and they each want to reliably verify that the other party holds the same file, without ever exchanging the file. To achieve this, we propose the following protocol:

1. Alice and Bob each generate a random nonce using a secure PRG. We shall refer to these random values as r_a, r_b for their respective parties and for simplicity assume that $r_a \neq r_b$.
2. Alice and Bob each encrypt and sign their nonces according to RSA protocol and send it to the other party.
3. Having received the other’s nonce, they each append the nonce to their password files P_a, P_b and hash the result. WLOG for P_b, r_a, we denote the result as $H(P_a||r_b)$.
4. Alice and Bob exchange their resulting hashes after encrypting and signing them according to RSA protocol.
5. Each party can then verify that $H(L_a||r_a) = H(L_b||r_a), WLOG\ for\ r_b$. If the hashes are a match, they know they have the same lists.

Security Analysis of Goals:

1. **No exchange of the passwords themselves, to protect their security.**

   There is *no* exchange of the file contents. The file is first hashed and then the hash is sent. Because we use a secure CRHF for use in our hash, we assume it is infeasible to determine the file contents.

2. **Message integrity: Alice should be able to trust that the communication she receives is from Bob, and vice versa.**

   We use RSA digital signatures and verification for message integrity. WLOG for Bob, Alice signs her messages with her private key, and Bob can verify this with her public key.

3. **Message security: only Alice should be able to decrypt/read Bob's messages, and vice versa.**

   We use RSA encryption/decryption for message security. WLOG for Bob, Alice encrypts her messages to Bob with Bob's public key, and Bob can decrypt the message with his private key.

4. **Alice and Bob should be convinced that the other has the same password file and has not simply replayed the other's verification of their own file.**

   WLOG for Bob, we ensure secure and party-verifiable transmission of a random nonce from Alice to Bob, which Bob appends to his password file and hashes, sending the resulting hash to Alice under the same security and integrity conditions. Seeing the hash, Alice can verify by using her own list and the nonce she produced for Bob to similarly hash her list and the nonce. If the result is the same, without ever exchanging their lists, Alice can verify that Bob has the same one.

Potential Issues/Shortcomings:

Man-in-the-middle attack: A man in the middle could manipulate communications to trick Alice and/or Bob into thinking that the other is acting nefariously or that they do not have the same passwords when in fact they do. Because no passwords are ever sent, however, the most that the man in the middle could corrupt even in the worst case is the trust between the two parties.