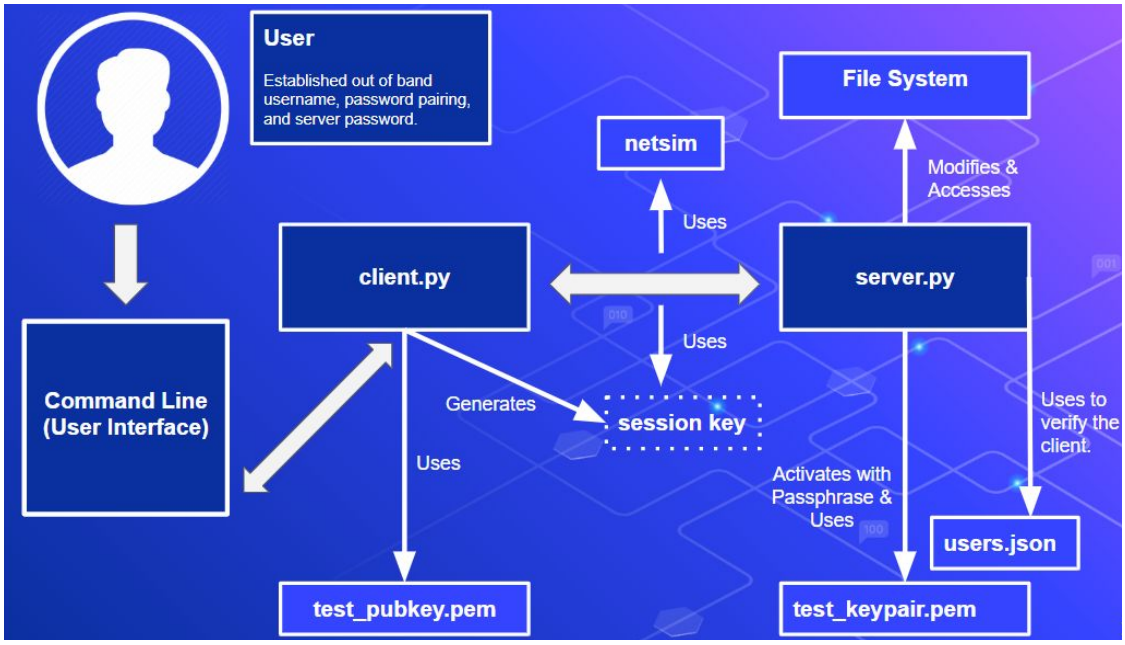
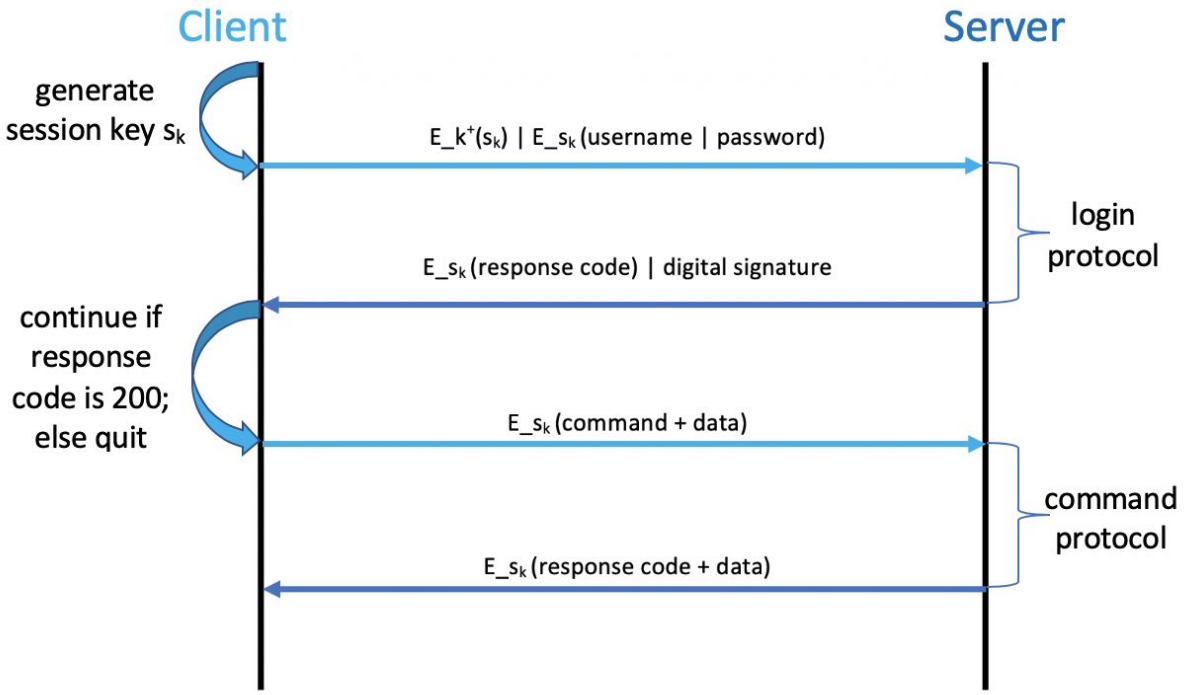
System Design



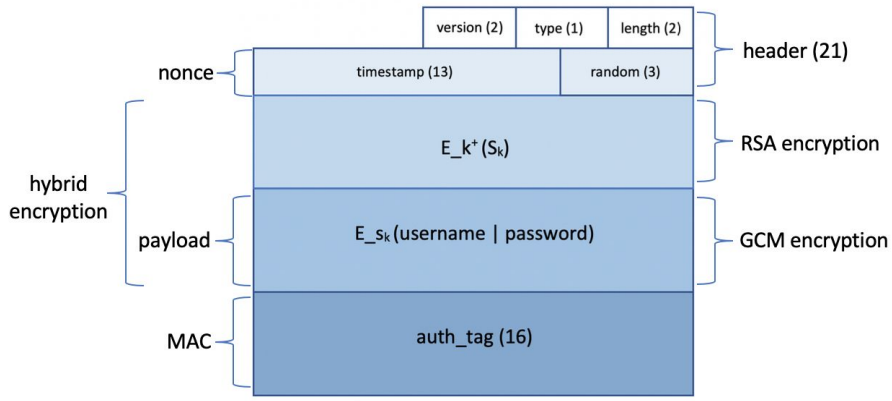
Login Protocol



Generation of Client Login Message (client.py)

- Login messages have the same header format as all other messages, containing the protocol version (1.0), the header type (login), the header length, and the nonce.
- In order to generate the nonce, a timestamp is generated using Python's current date-time which is then concatenated with 3 random bytes.
- Hybrid encryption is used:
    - A session symmetric key is generated as 32 random bytes and is encrypted using the RSA public key stored in test_pubkey.pem.
    - The payload includes the user's username and password. The payload is encrypted using AES in GCM mode with the nonce at the beginning of the message.
- AES in GCM mode is also used to generate an authentication tag based on the header, hybrid encryption, and the payload. This is concatenated after the encrypted payload.

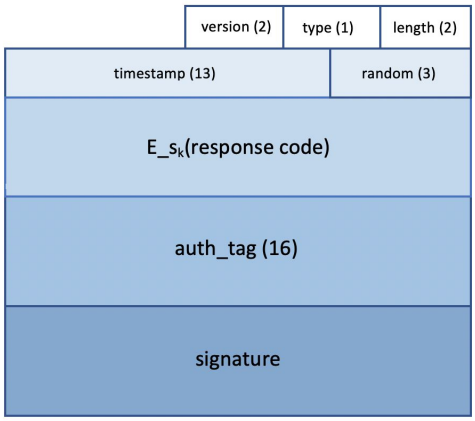
The login protocol format is depicted below:



Server Processing of Client Login Message (server.py)

- Upon identifying that the message is a login message (based on the header), the server is authenticated as it must use the current passphrase to access the private key—in real life, tamper-resistant hardware would be used to store the private key, but to simplify our implementation we have the server enter a passphrase. The server then validates the authentication tag.
- Next, the server checks that the timestamp is within the accepted time frame.
- Then, the server identifies the session key using its private key (accessed using the passphrase).
- Finally, the server then decrypts the username and password, identifying the appropriate directory and username-password validity using users.json.
- If successful, the server will send the client a login success message encrypted with the session key from the client's login initiation message. If unsuccessful, the server sends the appropriate failure code, which is also encrypted. Both messages will include a signature of the header, the payload, and the authentication tag using the private key.

The server's response to login messages is shown below:



Storage of Sensitive User Information

Users each have a unique username and password combination that is determined out of band. For each user, the pair of the hash of the username and the hash of the password is stored in user.json. The hashes are independently generated using SHA256. There is a second file 'addr_mapping.json' which contains the maps of usernames to network addresses, both in plaintext. For our demonstration three users are established, and their information is summarized in the following table:

Username	**Password**	**Network Address**
levente12	Ey3L0v3m@@thH	A
istvanist	tEs$or1t2	B
gabor@ait	aitA!TaitA1T	C

Storage of Private-Public Key Pair

An out of band public-private RSA key pair is pre-established and stored in test_keypair.pem and test_pubkey.pem. In order to access the test_keypair.pem, an out of band password is used by the server to initiate private key usage. The out of band password is cryptography.

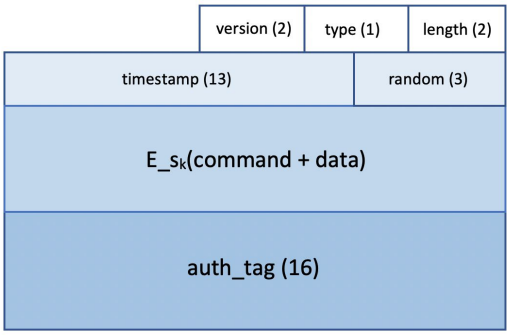
Transient Information

Session records and session keys are not stored in a file format; they are generated in login.py, sent and verified between server and client, and used for communication. This reduces system vulnerability as it removes the possibility of accessing past session information.

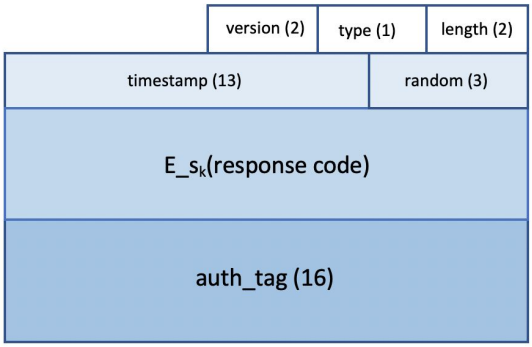
Command Protocol

Session commands are prompted by the user on the command line interface. After each command, the user is prompted if they want to continue or not using (Y/N). The client.py then converters the user command input into a message that is sent to the server. This message includes the command type, which is converted to the corresponding command code found in the client.py code body, the relevant data (file or folder name), and an authentication tag. Server.py parses the message and establishes its validity (similar to as in the login protocol). If valid, server.py will implement the action onto the file system. Finally, server.py will send a response code to the user based on the validity of the command and the successful execution of the command. Different response codes correspond to different types of failure. Client.py will interpret this verification or failure and notify the user if their execution was successful on the command line interface appropriately.

Client Message



Server Response



Commands

MKD –

Usage: MKD -n <name_of_folder>
Success message: “The folder ${folder_name} was created”
Error message: “There was an error in creating the folder.”

RMD –

Usage: RMD -n <name_of_folder>
Success message: “The folder ${folder_name} was removed”
Error message: “There was an error in removing a folder.”

GWD –

Usage: GWD
Success message: “The current folder is ${current_folder_name}”
Error message: “There was an error in getting the name of the current folder.”

CWD –

Usage: CWD -p <path_to_folder>
Success message: “The current folder is now ${current_folder_name}”
Error message: “That path is invalid or that folder could not be found.”

LST –

Usage: LST
Success message: a list of all the items in the current folder, one on each line
Error message: “There was an error in listing out the contents of the folder ${current_folder_name}”

UPL –

Usage: UPL -f <path_of_file>
Success message: “The file ${filename} was uploaded to the folder ${current_folder}”
Error message: “There was an error in uploading the file.”

DNL –

Usage: DNL -f <filename> -d <destination_path>
Success message: “The file ${filename} was downloaded from the folder ${current_folder}”
Error message: “There was an error in downloading the file.”

RMF –

Usage: RMF -f <filename>
Success message: “the file ${filename} was removed from the folder ${current_folder}”
Error message: “There was an error in removing the file.”

Usage & Implementation
We need to simultaneously simulate three programs:

1. **Network**
2. **Server**
3. **Client**

Recommended Commands
Assuming user begins in the LevenTransfer directory.

python3 netsim/network.py -p ‘./network’ --clean
python3 server.py
python3 client.py -u levente12 -p Ey3L0v3m@@thH

Note: password username pairings can be changed, based on which account the user wants to implement using the command line interface.