Intro:

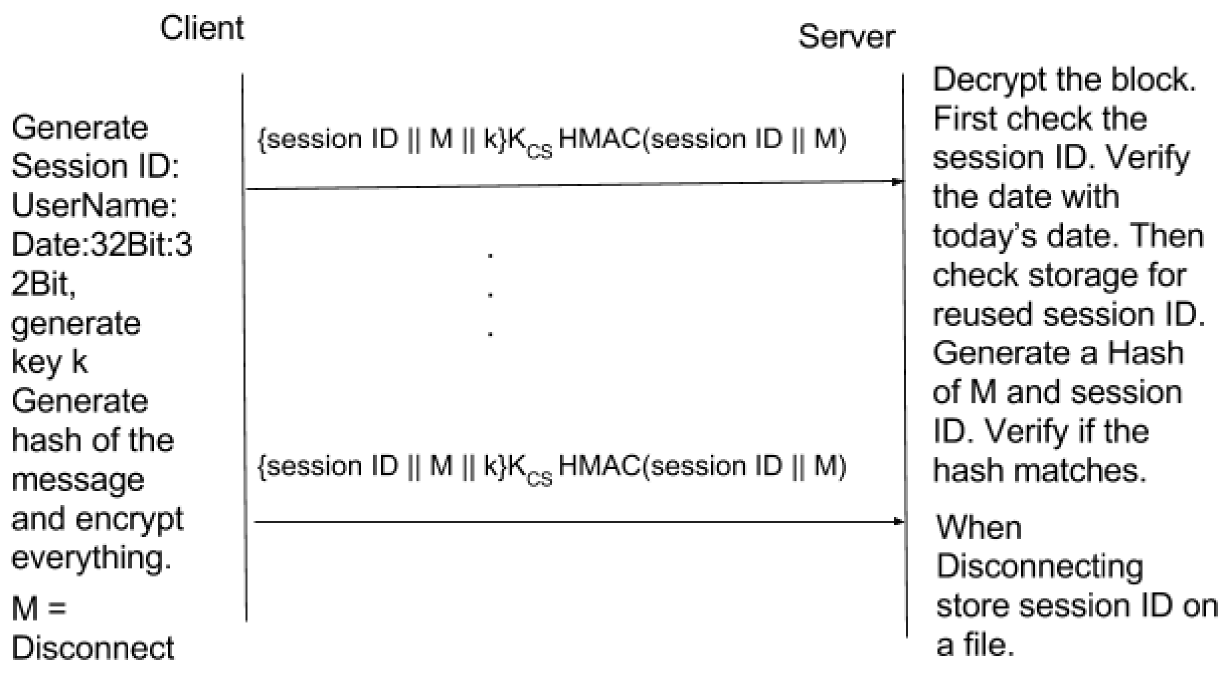
To protect against the added threats presented to us, our group either added new mechanisms or made use of old mechanisms from threats 1-4 to ensure that the file system is secure. We will continue to use AES encryption with 256 bit keys and will use SHA 256 for any hashing. Some new mechanisms include sequence numbering, timestamps and nonces in a session ID to protect against threat 5. To ensure protection against file leakage, we encrypted all the file data with AES using a key that the file server has no means of accessing. Finally to protect against token theft, we made sure that the file server could operate after authenticating that the token should work for the server.

T5 Message Reorder, Replay, or Modification
Summary:

This threat model describes an active attacker attempting to modify, reorder, or replay an old message to gain unwanted access to the server. To prevent replay attacks--attacks that take an old message and replay it to make it seem as if an authorized user is requesting something--we will include a session ID to every message. The session ID can also prevent reorder attacks--attacks that change the order at which the server sees a message or request. This session ID will contain the clients username followed by a timestamp which contains current Date plus a random 32 bit number and a 32 bit message number. After the connection ends the session ID will be stored on the server as an invalid session ID. Before performing each action the server will look at the session ID and verify that the date match the current date. If the session ID date matches, the server will then look at the stored session ID and verify that the current session ID doesn’t match any ID found in the file. Finally the server will check the message number and make sure that the number is one after the last message. After verifying the session ID, the server will add it to an ArrayList which it can check to make sure the messages are in order. At the end of the connection--when the client sends a disconnect message--the session ID will be written to a file named Unaccepted Session ID. Each server will have their own file containing the Unaccepted Session IDs. To handle the modification of messages, we will send a hash of the message along with the message. Should the message be modified the generated hash of this modified message will greatly differ from the hash that's being sent. The hash we will use is HMAC because we need a message authentication code between the servers and the client. First the client will generate a session ID. Then the client will encrypt the session ID and the message. The client will then generate a random 128 bit key for HMAC, and the client will encrypt this key with the symmetric key he has with either servers. Finally, the client will generate a hash of the session ID and message combination using the key and will send both the hash and the encrypted block over to the server. The server will first decrypt the block and then generate a hash of the block and compare that to the hash that it received. After verifying the hashes match it will check the session ID.

Justification:

This works because if the attacker were to modify the data by changing a single bit, the hash generated from the modified message will be much different. To generate a hash an adversary will need to decrypt the encrypted block, make the modification he wanted, and then encrypt the data. Since the adversary can’t decrypt the data, he can’t generate an accurate hash of the data. Replay attacks will fail because the session ID will keep track of session ID by username and date so any replay attack containing data from a previous date will fail. The 32 bit number after the date will help prevent same day replay attacks. After a connection ends we will store the session ID in a file. Each server will have their own file to contain these unauthorized session IDs. So if an attacker attempts to replay on the same day, the first test of the date will pass. However, the server will check the session ID in the file and find that the session ID was already used and will kill connection. To Optimize, the file can be deleted at midnight because the date changes so the server won't need to save the session ID. The username is needed because each user should have his own set of session ID per day. Reorder attacks will fail because the final 32 bits of the message is a number that increases by 1 after each message passed by the client. The server will save the previous message number and compare it to the next message number. If the number it receives is not one more than the last message the server will know that message has been reordered so it will ignore it.


Client
Server
Generate Session ID: UserName: Date:32Bit:32Bit, generate key k Generate hash of the message and encrypt everything.
M = Disconnect
{session ID || M || k}K_CS HMAC(session ID || M)
{session ID || M || k}K_CS HMAC(session ID || M)
Decrypt the block. First check the session ID. Verify the date with today's date. Then check storage for reused session ID. Generate a Hash of M and session ID. Verify if the hash matches.
When Disconnecting store session ID on a file.

http://www.practicalnetworking.net/series/cryptography/message-integrity/
https://security.stackexchange.com/questions/37734/man-in-the-middle-attack-over-an-encrypted-channel

T6 File Leakage
Summary:

This threat is saying that any data in the file server can be assumed to be stolen or given away. This means all the data that the client sends to the file server should be hidden so that only the authorized people can see the data and if the file server gives it away, the data is hidden from any outside views. To ensure that the file server can't see the data we will encrypt all the data using a private key that only the client knows. The file server will only see the encrypted data and won't be able to decrypt that data because it doesn't have that key.

Group server will store all the AES keys via a hashtable that has the group name as the key and an arraylist of keys for each group as the value (hashtable doesn't have to be encrypted because it is on a trusted server). The AES keys will use the CTR mode of operation and will be 128 bits. Every time a client wants to connect to the file server, the group server will gather and give a set of group AES keys to the client based on their group memberships. The client will then use the set of keys to encrypt/decrypt files sent/retrieved from the file server. When the client wants to upload a file, they use the latest key from their corresponding group's list of keys to encrypt the file as well as send a key number that tells which key was used from that group's list of keys. When the client wants to download a file, they use their list of keys for the corresponding group to decrypt, then the file server sends the key number to the client to let them know which key from the group's list of keys to use in order to decrypt. Whenever a user is removed from the group, the group server will generate a new AES key for the groups that user was previously involved in and add those new keys to their corresponding list of keys for each group.

Justification:

This will work because the data that the file server will see is the file names and key numbers. Other than these file names, the server will see encrypted mess that it can't decrypt. So if the file server can't decrypt it then any person it gives the file to can't decrypt it therefore, the file will be safe. The key number will simply be integers that act as a numbering system to keep track of which key to use. Since the file server has no access to a user's keys, the key number is essentially useless information to the file server. Whenever a user is deleted, we simply only need to generate new group AES keys for the groups that user was involved in and increment the key numbers for each group because we only need to worry about the user not being able to see files created in the future. We don't care if the can still see old files because chances are some or all of those files were saved before the user was deleted. With the new group AES key, they will essentially be locked out from future files created within their groups.

T7 Token Theft

Summary:

This threat is understood to mean that after a usertoken is sent to the file server, the server will attempt to pass that token to another user. To fix this the client will send the group server the FileServer's IP address and port to the group server. After group server has authenticated the client, it will add the IP address and the port to the user's information to stringify, hash all the stringified information with SHA256, store that hash inside the UserToken, and finally sign the hash with its private key. Then the group server will put the FileServer's IP address and port inside the UserToken. Essentially T7 is an added extension to what is already being done in T2 with Token Modification/Forgery. When the client is ready to connect to the file server, he will send the token over where the server will be forced to generate its own hash of the UserToken and compare it with the hash stored in the UserToken. Then the server will check that its IP and port match those found in the UserToken. The server will only proceed after confirming that the hash and the server data match the Token.

Justification:

This mechanism works because any server that the token is not supposed to work on will not accept the token because the IP address and the Port number will not match as denoted by the unmatched hash values. The servers or users could attempt to modify the token but since the trusted source of the group server has signed it, the modification will be found as a result of the different hash values and the server can ignore all tokens that do not pass the signature.

Threat 1: Unauthorized Token Issuance

Our mechanisms for Threats 5-7 don't take away the effectiveness of our Threat 1 mechanism, but instead build upon it. Our threat 1 mechanism makes sure that each client gets the appropriate UserToken, while our threat 7 mechanism builds upon that by making sure that even if the security of our threat 1 gets bypassed, the other possible threat is token theft, which is handled by our threat 7 protocol.

Threat 2: Token Modification/Forgery

Our mechanisms for threats 5-7 don't take away the effectiveness of our threat 2 mechanism, but instead build upon it. Our threat 2 mechanism was a signature from the group server to make sure the token isn't modified. Threat 7 makes use of that signature to ensure that the IP address and the Port number remain unmodified in the token.

Threat 3: Unauthorized File Servers

Our mechanisms for threats 5-7 don't take away the effectiveness of our threat 3 mechanism, but instead build upon it. Threat 3 provides the possibility of a malicious file server, while threat 7 assumes the file server is malicious. Our protocol protects against malicious file servers that will give away tokens to other users upon verifying that the requested file server is what the user wants, while also checking the IP address and port number of the file server requested by the client.

Threat 4: Information Leakage via Passive Monitoring

Our mechanisms for Threats 5-7 don't take away from the effectiveness of out threat 4 mechanism because all the data is still encrypted so any passive monitor will not be able see anything of value.

Conclusion:

In conclusion, we used various algorithms to implement a strong file sharing system. The flow between all 7 of the mechanisms we used was that a lot of the keys generated in the early mechanisms helped protect the data being passed in the later mechanism. For example the session ID should be hidden from any passive attacker and our threat 4 makes use of threat 1 and 3's key generation to encrypt the session ID to ensure that threat 5 is protected against. Something we had an issue with was finding an easy way to handle threat 6. We knew that the files will need to be encrypted but we couldn’t think of how to use the information that each group had to create a unique key. In the end, we realized that as long as that each member of the group had the same key while the members outside the group didn’t have access to that key, threat 6 will be handled. With this realization we came up with having the group server create a unique set of keys for each group uses these keys to encrypt the file data so that only the group members can see the files should they ever get leaked.