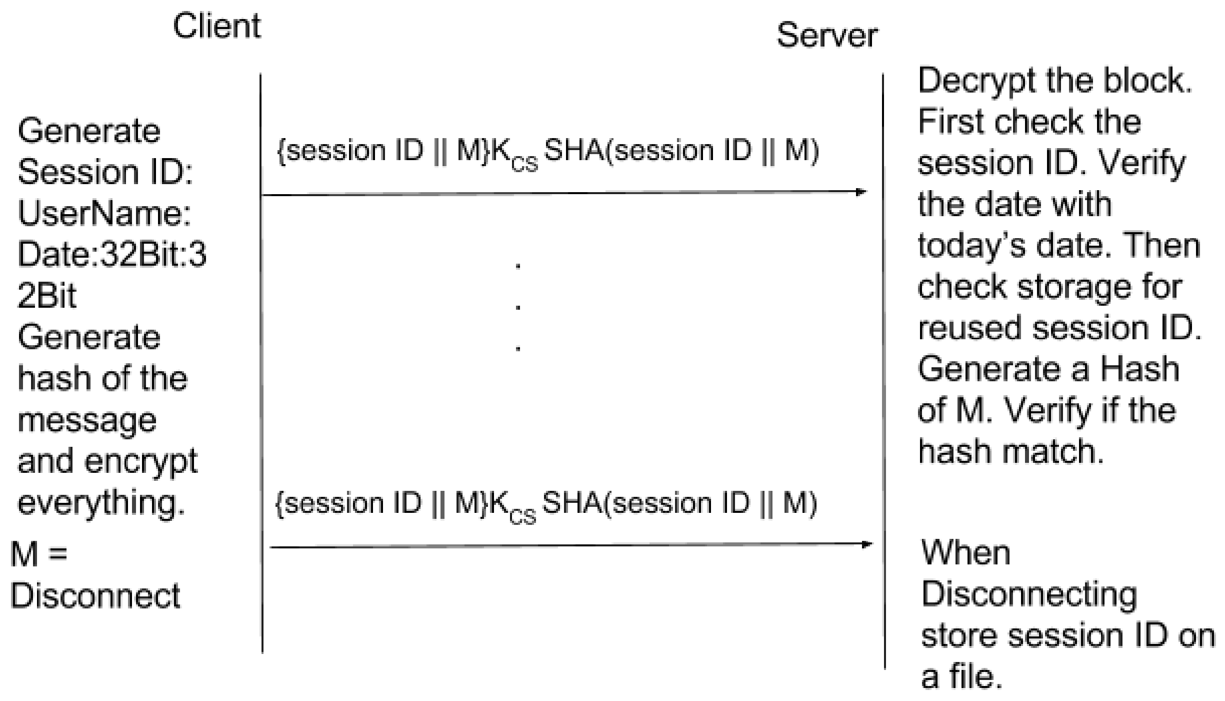
Intro:
To protect against the added threats presented to us, our group either added new mechanisms or made use of old mechanisms from threats 1-4 to ensure that the file system is secure. We will continue to use AES encryption with 256 bit keys and will use SHA 256 for any hashing. Some new mechanism will 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 only move one after authenticating that the token should work for it's 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--take the session ID and write it to a file named Unaccepted Session ID. 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 SHA 256 because we just need a simple hash function that has the properties of hashing like preimage and second preimage resistance along with collision resistance. We don't need a cryptographic hash so HMAC isn't needed. First the client will generate a session ID. Then the client encrypt the session ID and the message. The client will generate a hash of the session ID and message combination and will send both the hash and the encrypted block over to the server. The server will first generate a hash of the encrypted block and compare that to the hash that it received. After verifying the hashes match it will then decrypt the message and 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 to 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 of the final of 32 bits is message 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 hash of the message and encrypt everything.
{session ID || M}K_CS SHA(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. Verify if the hash match.
M = Disconnect
{session ID || M}K_CS SHA(session ID || M)
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 can't decrypt that data because it doesn't have that key. We will use the shared key between the file server and the client to encrypt the message like “UPLOADF” “DOWNLOADF”, “LFILES” since these are messages that the file server needs to function. The file name will also be encrypted using the shared key because the file server will need the name to look up the file or store the file. The file will be encrypted using some private key that the client generates. To generate the key, the group server will generate a normal 256 bit aes key when the first member of a certain group connects. We will check this by checking our object Group and will check a boolean variable that signifies if the key has been generated or not. If the key has not been generated the group server will generate one, set the boolean to true, and store the key in the group object. This will happen the first time a group is created. Then the group server will send all the keys that a client has access to--meaning the keys for the groups he is in. The client will then encrypt the file information using the shared key between the file and the client and will encrypt the file data using the key the group server generated. Should a person be deleted from a group or deleted overall, the group server will generate a new key and send the new key and the old key to the client. This client will be the admin because only the admin could delete someone. The admin will then be forced to connect to the file server decrypt all the files that were encrypted with the old key and encrypt the files with the new key. This happens to all the files that the deleted user or the affected group has access to. The encryption that will be used is AES because we have a asymmetric key architecture and AES is a fast encryption.

Justification:

This will work because the data that the file server will see is the file names. 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. This may seem inefficient because deleting a person could resulting decrypting and encrypting of a lot of files but the assumption made by the group is that deleting isn’t an operation that occurs often. We are assuming that because clients are not trustworthy, they have the ability to save the key given to them by the group server on their local machines. Even though they can save the key and possibly give the key away to another user, that user can’t log in to the file server without the correct corresponding usertoken that has permissions to use that key since each key is tied to a certain group.

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 put the IP address and the port in the token and sign it. When the client is ready to connect to the file server he will send the token over where the server will be forced to check the IP address and Port number inside the token and compare it to its own IP address and port number. The server will only proceed after confirming that the IP address and Port number matches.

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. 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 and the server can ignore all tokens that do not pass the signature. Also in T2 there is a mechanism in place to protect against token modification which can be used to ensure that the token isn't modified here.

Threat 1: Unauthorized Token Issuance
Our mechanisms for Threats 5-7 don't take away the effectiveness of our Threat 1 mechanism, but instead builds 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 out threat 2 mechanism, but instead builds 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 out threat 3 mechanism, but instead builds 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.