fPart 4

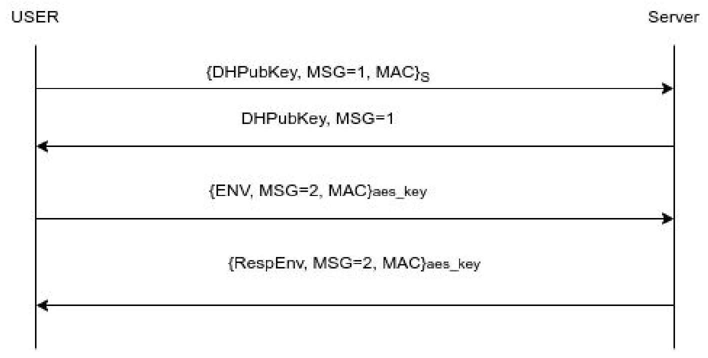
Intro:

For this phase, we will be using all of the same techniques that we used in the last phase. Namely, our signed DH exchange will use symmetric and asymmetric cryptography. The DH keys will be generated from the same group as the last phase, which is known to be safe. They will be encrypted via 8192 bit RSA and will be used to derive 256 bit AES keys. We will still be using signing of tokens which will include verification of the file server that the token will be for. This signing will be done using RSA as well, with a hash over the public key for the file server. The hash will be done with SHA256.

T5: Message Reorder, Replay, or Modification After connecting to a properly authenticated group server or file server, the messages sent between the user and the server might be reordered, saved for later replay, or otherwise modified by an active attacker. You must provide users and servers with a means of detecting message tampering, reordering, or replay. Upon detecting one of these exceptional conditions, it is permissible to terminate the client/server connection.

In this threat, we are worried about an adversary being able to use the messages being sent in the network to subvert the normal workflow, even if they do not know the contents of the messages because they are encrypted. Since we do not send more than one message at a time without an ack, message reorder is not an issue. Upon detection of garbage, we close the connection. What we need to worry about is a very specific replay. General replay is not an issue as the encrypted channel uses a different key each time, and later sessions cannot use encrypted messages from older sessions. However, during the same session, while transferring a file, a duplicated message could insert duplicate data into a file that is being uploaded or downloaded. It is also possible that large file chunks could be modified.

To mitigate this, we will use a sequence number for each message in the socket connection. This sequence will be increased by one for each message. The sequence number will be stored by each connection object and included in the envelope sent. This will allow for the detection of a duplicate message. An adversary will not be able to tamper with this number as it will be encrypted in a message for which they do not have the key. The second message in the system will be the only one that could be changed, but this is easy to detect. We will also use a MAC for each message to detect tampering and disconnect upon detection.


USER
Server
{DHPubKey, MSG=1, MAC}$_S$
DHPubKey, MSG=1
{ENV, MSG=2, MAC}$_{aes\_key}$
{RespEnv, MSG=2, MAC}$_{aes\_key}$

T6: File Leakage Since file servers are untrusted, files may be leaked from the server to unauthorized principals. You must develop a mechanism for ensuring that files leaked from the server are only readable by members of the appropriate group. As in previous phases of the project, we stress that the group server cannot be expected to know about all file servers to which its users may wish to connect. Further, your proposed mechanism must ensure that some level of security is maintained as group memberships change.

In this scenario, we are concerned with the file servers leaking information to parties that should not have access to the information. Since the information is stored unencrypted, if a File Server were to leak a file that is meant for Group A, and it ends up being seen by Group B, the confidentiality of that information has been compromised. We are also concerned with the access to be had by members of a group that leave the group. If a user leaves a group they should not be able to go back in and view files added since their departure.

In order to mitigate this threat we are implementing a mechanism in which the files will be stored on file servers encrypted. Since the Group Server is the only party that the users can trust, encryption keys will be stored on the group server. The group server remembers two values: a 256 bit AES key *k* and a number *n*. *n* is the number of times that *k* must be hashed (SHA-256) for the current file encryption key. When a user gets their token from the Group Server, they will also receive *k'* and *n*, where *k'* is the n^{th} hash of k. The user can use *k'* to encrypt their files and they store that encrypted version on any File Server, along with *n*. When a user wants to decrypt a file, they can download the file and check the File Server's *n* for that file (n_{FS}). If n_{FS} is greater than the user's *n*, they can simply hash the extra times necessary for that version of the key and decrypt the file. Whenever a user is removed from the group, the Group Server decrements their *n*, which causes newly issued tokens to have a new and unique version of the encryption key which can be used to derive previous keys but not future keys.

This approach means that any user added to a group even briefly may gain access to *all* of that group's passed files, but upon removal will be completely unable to decrypt files that were encrypted after their departure. Note, the responsibility for the encryption is on the shoulders of the client. If the client fails to properly encrypt, they may create their own vulnerability or make their encrypted file irrecoverable.

T7: Token Theft A file server may "steal" the token used by one of its clients and attempt to pass it off to another user. You must develop a mechanism for ensuring that any stolen tokens are usable only on the server at which the theft took place (and are thus effectively useless, as this rogue file server could simply allow access without checking the token)

In order to prevent the theft of tokens, we have decided that a usable token must specify a recipient File Server on which that token can be used. This "recipient" field is populated with a hash (SHA-256) of that File Server's public key which the Client provides the group server when they ask for a token for the purposes of speaking with a

File Server. The Client can receive a token with no recipient field, but a valid File Server should immediately reject that connection. When the Client wants to speak to a File Server, they must ask the trusted Group Server to provide them with a recipient field which is the hash, as described before, signed by the Group Server. The signature prevents the Client from being able to alter this value after receipt.

To continue upholding T2 from the previous phase, the signed hash of the recipient File Server's public key needs to be incorporated into the token's signature. In order to handle this, a "recipient" field will be added to the `toSignatureString` output that is populated with the hexadecimal representation of the signature (to avoid any character complexity). The signature of this further prevents tampering with the token after receipt.

With this alteration to the Token, a genuine File Server, upon receiving a request from a Client, will check that the Client's token is valid by first checking the that recipient field matches their public key and is signed by the Group Server, then checking that the Client has the necessary permissions for the action that they are requesting to perform and an untampered token (which is done by comparing the signature of the Token to the output of the `toSignatureString` function as described in T2). This means that the only time that stealing a token by having it leaked from a File Server would benefit an attacker was if they were going to use the stolen token on the same File Server that leaked it or another disingenuous File Server that wasn't checking permissions anyway.

Conclusion:

Still satisfies T1-4:

We will address each of the previous 4 threats one by one.

T1: Unauthorized Token Issuance

Nothing about token issuance is being changed, except for the fact that a message number is being added to the envelope.

T2: Token Modification/Forgery

The approach to T7 strengthens defense against this. The change made is that a token is made specific for a single file server. This is added to the token and signature validation of the group server's signature is done on the token in the same exact way.

T3: Unauthorized File Servers

This was solved via the signed DH exchange. Nothing about that has changed other than a message number being added to the envelope.

T4: See reasoning for T3.