CS 1653: Project Phase 4

Group Members: Solomon Astley, John Fahnestock, Alex Mesko

Name	Definition	Notes
k_{gs}	The public RSA key of the group server.	Assumed to be known by any client that wishes to access the group server.
k_{gs}^{-1}	The private RSA key of the group server.	Assumed to be known only by the group server.
k_{fs}	The public RSA key of a file server.	
k_{fs}^{-1}	The private RSA key of a file server.	Assumed to be known only by the relevant file server.
K_{cs}	The AES key used to encrypt messages between a client and server for a given session.	
K_A	The AES key used to generate HMACs for messages exchanged between clients and servers.	

Notes

AES keys shall be 128 bits and the chosen mode of operation shall be CBC with PKCS5 padding. RSA keys shall be 4096 bits. The SHA-256 algorithm shall be used to generate HMACs.

Mechanisms

This section of the report will consist of a description of each of the mechanisms that will be deployed in order to protect against the threats outlined in the project description. Each threat will be described, followed by a description of the mechanism being used to protect against the threat and a short argument explaining the correctness of the preceding mechanism to protect against the given threat.

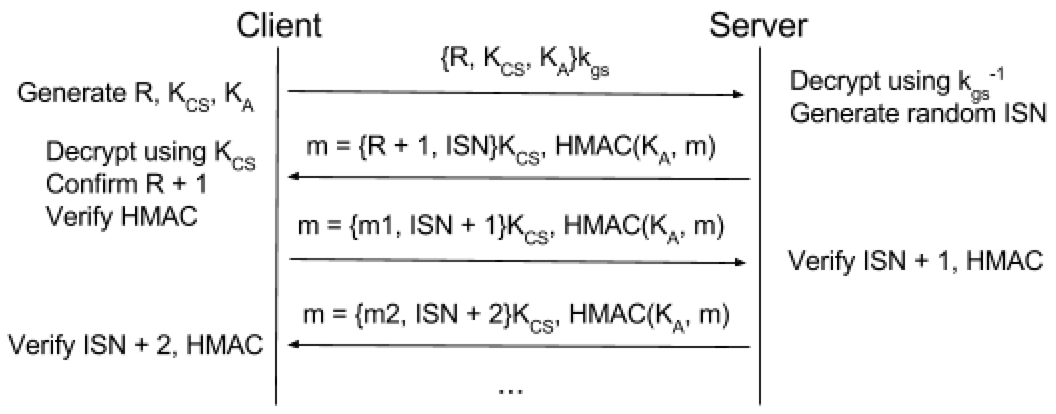
Threat 5: Message Reorder, Replay, or Modification

In this phase of the project, it is assumed that all communications between clients and servers will be intercepted by an *active attacker* that can modify, reorder, or replay messages. These types of attacks may result in various undesirable consequences for the users of the client application. Most notably, an active attacker has the ability to replay login messages or hijack login protocols while they are being carried out in order to impersonate a given user. Additionally, an active attacker may reorder messages in an attempt to undermine authentication protocols which require a specific order of messages in order to work properly. In this phase of the project, the server should have some method of detecting any of these attacks and terminating the connection when they occur.

To deal with this threat, a two-fold protection scheme will be employed. Firstly, all messages exchanged between a client and a server for a given session will contain sequence numbers. This idea is similar to the sequence numbers used in TCP packets which ensure that packets are delivered to a network application in the proper order. The sequence numbers will begin with a random initial sequence number (ISN) which the server will create for each new session (more detail on why the ISN must be random as opposed to a constant can be found later on in this section). In order to achieve this, Protocol 1 (see the phase 3 writeup for more details on Protocol 1) only needs to be slightly modified. Recall that Protocol 1 is carried out at the beginning of each new session between a client and a server in order to establish a shared secret key. In this modified version of Protocol 1, which will be referred to as Protocol 2, the server's response to a client's challenge will also contain a random ISN for that session.

After the ISN is created and established, each additional message sent between the client and server shall contain the previous message's sequence number plus one. For example, after the client connects to a server and receives the ISN for this session, its next message will contain the ISN + 1, and the server's response to that message will contain the ISN + 2. In the case that either the client or server sends more than one message in a row without a response from the other party, the sequence number shall be incremented for each message. This mechanism assumes that messages exchanged between the client and server shall be synchronous and involve only the singular client and server. These assumptions align with the current implementation of the system.

While sequence numbers allow a server to detect the replay and reordering of messages (explanation of why this is true below), they do not provide a mechanism to detect if a message has been modified. To achieve this, the second part of this mechanism will utilize an HMAC over each message sent between a client and a server. In addition to the aforementioned changes to Protocol 1, Protocol 2 will also involve the generation of a second secret key by the client, K_A. After K_A has been established, each subsequent exchange between the client and server will require that an HMAC be generated over the message using K_A, and that the HMAC be checked upon the message's receipt in order to verify the integrity of the message. See below for a diagram illustrating the finalized Protocol 2:



There are several points to consider when confirming that this protocol sufficiently protects against Threat 5. Firstly, the mechanism must enable the detection of message reordering and replay. To achieve this, the mechanism uses sequence numbers to track the order of messages as they are sent and received. If an attacker made an attempt to reorder and replay any of the messages for a given session, the server would detect that the replayed messages were out of order because they did not contain the appropriate sequence numbers (the sequence number of the last correctly received message plus one). An attacker might also attempt to replay the entire session in this protocol, starting with the first message which does not contain a sequence number. In this case, the attacker cannot accomplish anything due to the fact that a replay of the first message would cause the server to generate a new, random ISN which would be incompatible with the rest of the messages in the replay. This is why the ISN generated by the server must be random and unique.

Next, the mechanism must also enable the detection of message modification. This is accomplished with the use of an HMAC over all messages. It is well-known that using the same secret key for multiple purposes is not secure, so an additional secret key K_A is generated by the client for this purpose. Given the HMAC of a message, it is infeasible for an attacker to discover either the contents of that message or the secret key used to generate the HMAC. Additionally, it is infeasible for an attacker to modify a message and then generate a new HMAC of that message without knowing the secret key used to generate the original HMAC. As such, if a server regenerates the HMAC of a received message, it may be confident that the message was not modified if the newly generated HMAC matches that which was sent with the message.

Threat 6: File Leakage

In this phase of the project, file servers are untrustworthy and may potentially leak files to the outside world. In the last phase of the project, files stored on the file server were completely unencrypted and thus could be read by anyone if they were leaked from the file server.

To assure security for this threat, first a few assumptions are necessary to be outlined. First, any files stored on a file server for a group are accessible to any member of that group. A new member just added to the group after multiple files were already on it will gain access to those files. However, a user that was a part of that group and was removed, may still have access to any files on that server that were present when they were a member, and also unchanged from the time of the removal of this member. This assumption is made because it is reasonable to assume a user can download all files on a server and have them stored before the user is removed from the group. Any subsequent updates to files, or any new files added, that removed user will not have access to.

To provide security from file servers leaking files they will necessarily be encrypted. The files will be encrypted with a secret that only the group server, and members of a specific group will have access to. The mechanism for ensuring this will work as follows:

1. Upon creation of a group, an AES key is generated for that group and stored in a KeyList file on the group server.
    a. The KeyList file will store all keys for every group on the server. This will work by having the KeyList store a Hashmap where the keys are the group names and the values are an ArrayList of the AES keys
2. Upon revocation of a user's membership in a group, a new AES key for that group will automatically be generated and appended to the end of the list of keys for that group. The key at the end of the list will be the "current" or active key.
3. In order to encrypt a file, a user will first obtain a key from the GroupServer for a group. The user will tell the GroupServer which group they want the key for and also an index or integer identifier of which key in the list they want. Next the user will use that AES key to encrypt a file sitting on their machine. At this point the user can upload that file to the file server securely. Since a group may have multiple keys due to users being removed, the user along with uploading the file, will provide the integer identifier they used to upload that key. This identifier will be stored on the file server along with the file.
4. In order to Decrypt a file, the user will connect to the file server and download an encrypted file along with the integer identifier for the key, which will be saved. Next, the user will ask the group server for a key for their group, and provide it with the index they got from the file, which corresponds to the key that was used to encrypt it. The Group Server will give the user the key and they can then use it to decrypt the file.

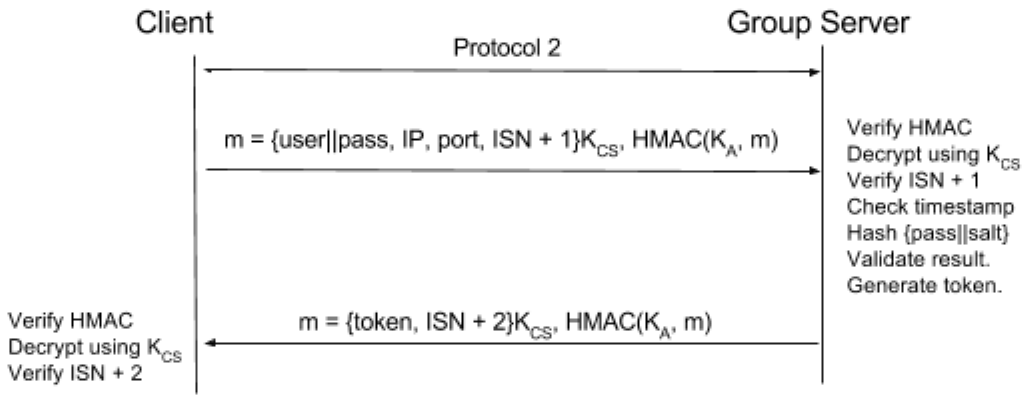
This mechanism works to defend against file servers leaking files to unauthorized client's because every file on the file server will be encrypted. The files will be encrypted with keys that only the current members and the GroupServer have access to. If a file server wanted to leak a file to an unauthorized user, they would not have access to the key to decrypt (unless this user was a previous member and was given a file they already had access to as outlined in the above assumptions). The case of a user being removed from a group is covered due to the fact that a new key is automatically generated each time a user is removed, and will be the new active key for that group that any subsequent file uploads or changes will use.

Threat 7: Token Theft

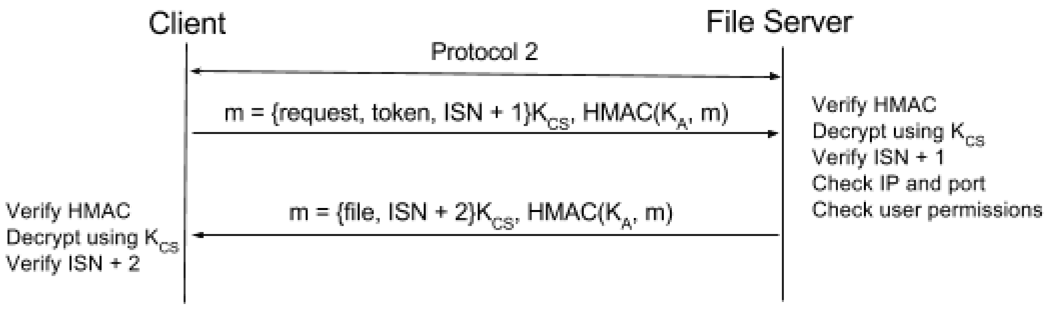
In this phase of the project, file servers may attempt to “steal” the tokens they receive from clients and use them to access the client’s files on other file servers. In the last phase of the project, there was no mechanism in place preventing an untrustworthy file server from reusing a token because there was no information in the token that designated where it should be used. This phase of the project should provide a mechanism to ensure that tokens issued to clients are only usable at the file server which the client wishes to access.

In order to protect against this threat, additional information which identifies a specific file server will be added to user tokens that are issued by the group server. When a client wishes to obtain a token to access a file server, they will issue a request to the group server for a token which contains the IP address and port number of a file server which they wish to access. The group server will then generate, sign, and return a token which additionally contains the IP address and port number of the file server requested by the user. In order to sign the token, the group server will first compute a hash of a string which uniquely identifies the token for a specific user and a specific file server. The identifying string will be of the form (issuer || subject || groups || IP address || port number). This string will then be used to generate a signature which will be added to the token itself and returned to the user.

If the user wishes to receive a token only for the purpose of performing group server actions (such as adding/removing users), they may request a token with the special IP address and port number of 0.0.0.0 and 0, respectively. A modified version of the login and token retrieval mechanism from phase 3 is illustrated below:



Additionally, the updated protocol used to access files on a file server is illustrated below:



This mechanism works due to the uniqueness of file server IP addresses and port numbers and the correctness of the mechanism developed in the last phase which prevents attackers from modifying tokens issued by the group server. When the group server generates a token, it adds the requested IP address and port number to that token and then signs it. Due to the uniqueness of usernames and file server IP addresses and port numbers, each generated token will be unique to a specific user and a specific file server. If a malicious file server attempted to use this token elsewhere, it would need to modify the token to reflect the new file server's IP address and port number. In doing so, however, it would invalidate the token due to the fact that the group server's signature on the token was generated using the old IP address and port number.

Concluding statements:

Our group started coming up with ideas first for threat 5 and 7, which we deemed to be more straightforward and we decided on the altering of our protocol 1 from the first phase to use sequence numbers and an ISN. We then started coming up with ideas for threat 6 and came up with 3 or 4 different ideas before setting on the final one. We had the idea of using the group server to perform encryption, and then moved to having a third party encryption server. We finally settled on having the groupserver store keys and the clients perform the encryption/decryption because it was easier to implement and a better design than the previous two. All of our mechanisms complement eachother in that the mechanism or threat 5 protects against an active attacker through both traffic with the Group and File servers. Our protocol for threat 7 preventing tokens from being stolen helps to stop clients from getting files from other servers that they shouldn't have access to which goes along with keeping files secure from unauthorized users which out threat 6 mechanism protects against.

Threats 1-4 from last phase:

Threat 1: Unauthorized token issuance

Our mechanisms from this phase still protect against this threat because they still require a user and password from the user to login, only a user knowing this information get get a token.

Threat 2: Token Modification/Forgery

Our mechanisms for this phase still cover this threat because tokens are still signed by the Group Server, so that a client cannot forge this signature or modify their token without know the group server's private key.

Threat 3: Unauthorized File Servers

This threat is still covered because the only thing changing here is the user's token is setup to talk to only one file server. When this user connects to a file server it still gives it its public key that the user manually verifies to be correct. So any subsequent messages encrypted with this public key, any file server that doesn't have access to the private key will not be able to decrypt protecting against any file server pretending to be another one.

Threat 4: Information Leakage Via Passive monitoring

This threat is still protected because all traffic is still encrypted via an AES key generated and agreed upon by the client and server at the beginning of the handshake. This protocol only got stronger with the addition of the ISN to prevent not only passive monitoring but an active attacker.