Introduction: CS 1653 Phase 3

The explosion of the digital era has led to the entanglement of cryptography in securing information. Code has always been used as a form of communication when channels can be potentially compromised. This has not changed with computers and information that can be sent at the blink of an eye. In fact, now more than ever there are demands for higher security. In this phase of our project we identify potential threats that could wreck havoc to file/group servers. We leverage RSA to build symmetric key encryption and decryption between the user and the respective file and server client. We built a user list with it's private and public keys to utilize its functionalities. At the end, RSA and any other encryption will still run the risk of being broken by a savvy hacker. The goal is to make it difficult and string a combination of different crypto techniques.

T1: Unauthorized Token Issuance

Threat:

Based on the description of T1, we need to ensure that the token issued by the group server is given to the specified requesters. In the realistic situation, Alice, a potential attacker, might try to request a security token from the group server. Without comprehensive and strict regulation, the group server might render Bob's security token to Alice. By obtaining Bob's security token, Alice will be able to do anything that Bob's security token allows Bob to do. Such a situation will not only exert a negative influence on Bob but also the users who share the same group with Bob. Alice will even cause more significant harm if Bob is in the ADMIN group because Alice can modify the user and group data of the entire system through the identity of the manager of Bob's security token. ADMIN's token gained by the potential attacker will have a severe impact on the whole file-sharing system.

Mechanism:

The use of the RSA algorithm can reduce the occurrence of the above risks as much as possible. Through the mechanism that the user signs on the Client Application and then verifies on the group server, the group server can determine the requester's true identity. Group server end can perform the random key pair generation, where encryption of generated token also happens, and then sends the private key to the verified requester. Finally, it renders the encrypted token to the corresponding requester. After receiving the encrypted token, the requester uses the private key obtained in advance to decrypt it. This method reduces the possibility that a third party will decode the token.

Defense:

Through sign and verify, the group server effectively determines the requester's real information and prevents transmitting token to error object. By securely assigning the public and private keys through key exchange protocol, the possibility of third-party stealing encrypted token is effectively avoided.

T2: Token Modification/Forgery

Threat:

Based on the description of T2, we need to ensure that the UserToken interface can be extended to allow file servers or a third-party to determine whether the token is valid. In the realistic situation, Alice, a potential attacker, might try to forge a security token, which should only be issued by a group server. Assuming that Alice's forged token is consistent with a user's token in the system, Alice will then get the priority that does not belong to her, and the authority given to her through the forged token destroys the system. This situation dramatically affects the system's overall security and the satisfaction of all users in the network.

Mechanism:

The aforementioned threats can be eliminated through the use of RSA. As mentioned in T1, we can perform random key pair generation at the group server end. When the users request their personal token, the group server can package the encrypted token and return it with the private key through the UserToken interface. The user then decrypts the ciphertext version token at the Client Application end. When the user needs to access the file server, the user needs to sign first, then uses the private key to encrypt his token and then send it to the file server. The file server then sends the information to the group server, and the group server verifies the user's signature. After that, the group server chooses the corresponding key to decrypt the user token. Finally, the group server returns the decrypted token information to the file server, and the file server provides the permission that matches the token. This method reduces the possibility that a forged token can trick the file server.

Defense:

Through sign and verify, the group server effectively determines the requester's real information and utilizes the corresponding key stored in to decrypt token encryption sent by that requester. Even though an attacker might forge the key which belongs to another user in the system, because of the different key pairs between the attacker and the system, the system can’t assign the same authority as the real owner of the key to the attacker.

T3: Unauthorized File Server

Threat:

In the instance the user connects to the file server, if the connection is not authenticated then they could be subject to a malicious file server. In essence if you are uploading a file to an unsecure server, sensitive/private information could be compromised.

Mechanism:

In order for a file server to be trusted by a user, it must provide some authentication to the user that only the actual server can know. Though certificates would be an ideal solution, they are outside the scope of our project. Instead, we can rely on public and private key pairs as a way of validating a File Server. Upon the first connection to a File Server, the user will be asked if they wish to accept the public key K. If the user agrees, K is stored by the user through their client application along with other identifying details of the File Server (e.g. server name, IP address, etc.). When the user wishes to authenticate the server, he or she can use K to encrypt a challenge to f. If f is in possession of the private key K^{-1} then f is able to return the challenge to the user and authentication is complete. An unauthorized server f` would be unable to correctly guess K^{-1} and therefore could not correctly decrypt the challenge and complete authentication.

Defense:

This mechanism's integrity lies within the held private key of the file server being secretive. In addition, the strength of the communication between the two servers must be upheld.

T4: Third Party Monitoring / Information Leakage

Threat:

A third party could monitor the connection or exchange between two users using the same file/group servers. Thus it would compromise the information/data being stored in the server and expose the server or users to malicious attacks. This could lead to directly cracking the authentication - private keys of the server or impersonating certain verified users.

Mechanism:

In order to avoid the compromised connection. One method would be to have a session key or use the same RSA pair that monitors the connection made with user and server. This would be an added layer of protection as both ends of the connection would have access to this shared session key that would enable the use of a symmetric key or private key access. The session key can be created from the public key authentication protocols that will give privy to the shared connection key that will last for a unique session or each time the user connects to the server.

Defense:

If the key exchange holds and is not disrupted then each session should be properly encrypted. Correctness can be maintained through the symmetric key protocol and then the proper order of public and private key encryption. The security is held in the strength of the passwords and the methods used for the encryption.

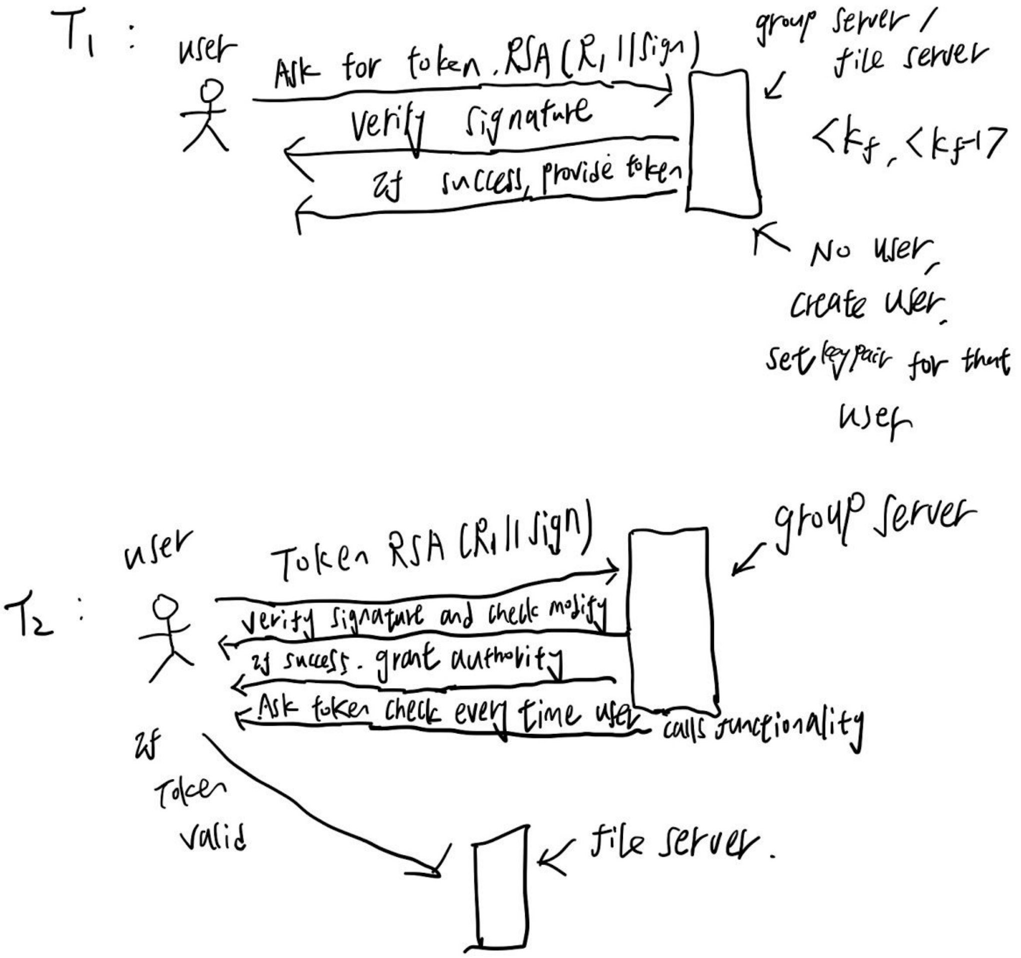
T1 :
user
Ask for token, RSA(R1||Sign)
group server / file server
verify signature
if success, provide token
<Kf, <Kf-1>
No user, create user, set keypair for that user
T2 :
user
Token RSA(R1||Sign)
group server
verify signature and check modify
if success, grant authority
Ask token check every time user calls functionality
if Token valid
file server.

Figure 1. Diagrams for Threat Models

Conclusion

RSA indeed contributes to solving this threat. By generating key pairs at the group server side when there is no UserList.bin file and when ADMIN user creates other user, and set the keypairs at UserList.java toward the specific user under above situations, the system is able to check the request information and signature provided by user with the signature and information stored in group server; if user indeed request his or her own token, then the group server provide the token to him or her; otherwise, return error information.

While the user tries to utilize any feature on both file server and group server, the token they provided will be checked under two steps. First, whether the signature information stored in their token matches with the one stored on the group server. Second, whether the detailed information in their token, like issuer, subject, and each group, are matched with the one stored in the group server. If a user's provided token passes the two steps mentioned above, then the authentication to utilize the specific feature will be given. User's token will be checked every time when a user requests for certain functionality corresponding to complete mediation.

Finally, the threats of unauthorized file servers and data leakage can be remedied by simply using the models described above. We would have the group server authenticate the connections between the client and file servers. Making sure that it is verified through the user list we can still use RSA to encrypt and decrypt the keys to authenticate.