Mark Shanoudy – mhs38@pitt.edu
Rich Mau – rim20@pitt.edu
CS 1653 Spring 2014
February 28, 2014

Project Phase 3

Introduction

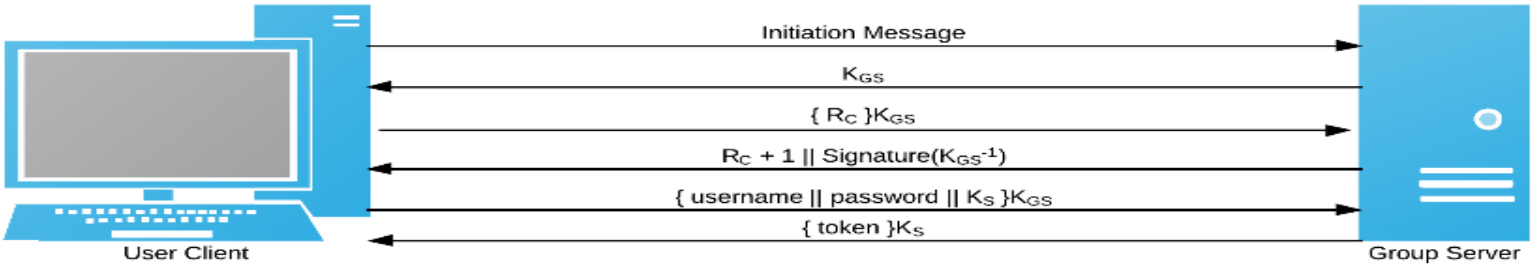
There are many different cryptographic techniques that could be applied in order to address the security threats described in this phase of the project. In order to address said threats, we have chosen to use a combination of both public key cryptography and secret key cryptography. We have also opted to make use of one-way transformations to help protect stored data where deemed necessary.

Threat 1: Unauthorized Token Issuance

The trust model assumes that clients are untrusted. Therefore, the system must protect against the threat of illegitimate clients requesting tokens from the group server. To do so, it must be ensured that all clients are authenticated in a secure manner prior to the issuance of tokens.

In phase two, the trust model assumed that all clients are trustworthy. This design assumption is easily exploitable when carried over to phase three because any client could easily request the token of any user at any time. This is problematic since it could lead to users gaining additional access to parts of the system through the use of another user's token. Furthermore, an illegitimate client could gain access to the system by using a legitimate user's token.

In order to address this threat, public key cryptography will be used to carry out a handshake protocol through which mutual authentication of the client and group server will take place. The handshake protocol will include challenge-response authentication for the group server and password-based authentication for the client.



The handshake protocol begins when the user client sends an initiation message to the group server. The group server then sends its public key, K_{GS}, of size 1024bits to the user client. Once the user client has received K_{GS}, it generates a large random integer, R_C, of size 1024bits. R_C is then encrypted using RSA with K_{GS} as the encryption key. The user client then sends the cipher text containing R_C to the group server. Upon reception of the cipher text from the user client, the group server proceeds to decrypt it using RSA with its private key, K_{GS}^{-1}, of size 1024bits as the decryption key. The group client then sends back R_C + 1 and a signed hash of R_C + 1 (RSA-FDH). The user client then verifies R_C + 1 is the correct answer and uses K_{GS} to verify the signature. The group server has been authenticated by the user client. The user client now encrypts a message containing the username, password, and a randomly generated session key, K_S, of size 256bits using RSA with K_{GS} as the encryption key. The cipher text is sent to the group server. The group server then decrypts the cipher text using RSA with K_{GS}^{-1} as the decryption key. The group will attempt to generate a token for the requesting user client. If token generation is unsuccessful, the group server will reply with *null*. If token generation is successful, the group server will encrypt the token using AES with K_S as the secret key. The group server will then send the resulting cipher text to the user client. The user client has been authenticated by the group server. The handshake is now complete.

The handshake protocol makes use of two main cryptosystems. The first is RSA. RSA plays multiple roles in the aforementioned protocol. It is the means by which the challenge-response authentication takes place. It is the means by which the session key is shared with the group server. It also allows for convenient digital signature creation. Furthermore, RSA was not only chosen for the initial parts of handshake because of its convenience. The key sizes of 1024bits help ensure security and confidentiality between the user client and the group server during the

handshake.

The second cryptosystem is AES. AES was chosen as the last part of the protocol because a symmetric cipher is much faster than RSA. Once the session key has been securely shared, both parties now have access to the secret key and therefore the problem of key sharing has been solved. This key expires once a connection is terminated; so it's only valid during the current session between the user client and the group server. Furthermore, cipher-block chaining with a key size of 256bits will be the mode of operation used for the symmetric cipher because ECB can leave vulnerabilities in the cipher text.

The correctness of this mechanism is based on the assumption that the protocol will be correctly followed by both the user client and the group server in order to achieve mutual authentication. Security of this mechanism is based on the strength of the cryptographic techniques employed by the handshake protocol.

Threat 2: Token Modification/Forgery

The trust model assumes that users are expected to attempt to modify their tokens to increase their access righsts, and to attempt to created forged tokens. It must, therefore, be possible for a third party to verify that a token was in fact issued by a trusted group server and was *not* modified after issuance.

In phase two, the trust model assumed that all tokens were valid, issued by a trusted group server, and were not modified after issuance. This design assumtion is easily exploited when carried over to phase three because there is no mechanism in place that allows for verification of token integrity.

In order to address this threat, a token must be verified every time an operation is to take place on one of the file servers. The trusted group server will sign a SignedObject field within the token itself that is a copy of the current token. This field can then be verified by the file server using the group server's public key.

The correctness of this mechanism is based on the group server's ability to sign the token upon its creation. The security of this mechanism is based on the secrecy of the group server's private key as well as the security of the communication channel by which the token is sent.

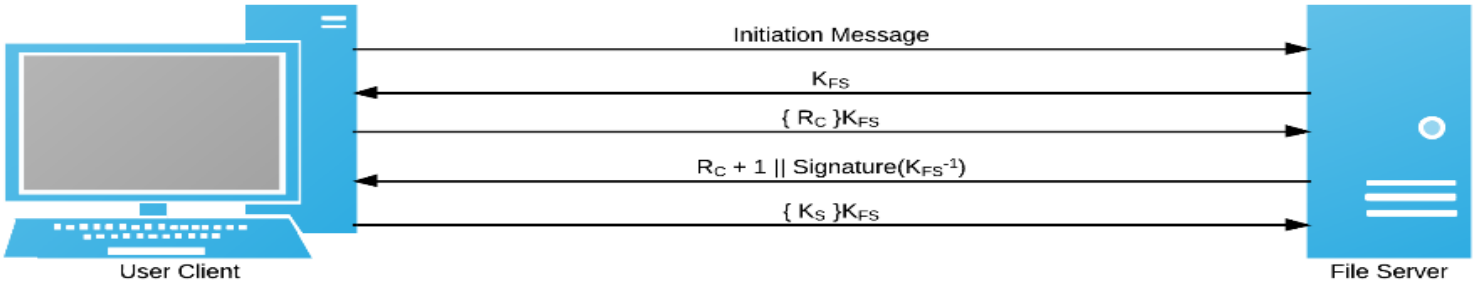
Threat 3: Unauthorized File Servers

The trust model assumes that properly authenticated file servers are guaranteed to behave as expected. In order for this guarantee to mean anything, the system must ensure that if a user attempts to contact

some server, *s*, then they actually connect to *s* and not some other server *s'*. It is important to note that any user may run a file server. As such, the group server is *not* required to know about all file servers. The mechanism for enabling users to authenticate file servers should require communication between only the user and the file server.

In phase two, the trust model assumed that all file servers were trusted. This design assumption is easily exploitable since anyone may run a file server. This may lead to legitimate users unknowingly uploading files to illegitimate file servers. Furthermore, the illegitimate file servers could be used to steal a legitimate user's token.

In order to address this threat, public key cryptology will be used to carry out a handshake protocol similar to that outlined in Threat 1. This handshake will include the same challenge-response authentication for the file server found in the mechanism for Threat 1. However, in this version of the protocol, there will be no password-based authentication for the user client. The user client's authentication will be instead based on their token as outlined in the phase two of this project.



The handshake protocol begins when the user client sends an initiation message to the file server. The group server then sends its public key, K_{FS}, of size 1024bits to the user client. Once the user client has received K_{FS}, it generates a large random integer, R_C, of size 1024bits. R_C is then encrypted using RSA with K_{FS} as the encryption key. The user client then sends the cipher text containing R_C to the group server. Upon reception of the cipher text from the user client, the group server proceeds to decrypt it using RSA with its private key, K_{FS}^{-1}, of size 1024bits as the decryption key. The group client then sends back $R_C + 1$ and a signed hash

of R_C + 1 (RSA-FDH). The user client then verifies R_C + 1 is the correct answer and uses K_{FS} to verify the signature. The file server has been authenticated by the user client. The user client now encrypts a message containing a randomly generated session key, K_S, of size 256bits using RSA with K_{FS} as the encryption key. The cipher text is sent to the group server. The group server then decrypts the cipher text using RSA with K_{FS}^{-1} as the decryption key. The handshake is now complete.

This version of the handshake protocol makes use of RSA only. It is the means by which the challenge-response authentication takes place. It is the means by which the session key is shared with the group server (although this key is not used in the actual handhsake in this version). It also allows for convenient digital signature creation. Furthermore, RSA was not only chosen for the handshake because of its convenience. The key sizes of 1024bits help ensure security and confidentiality between the user client and the file server during the handshake.

The correctness of this mechanism is based on the assumption that the protocol will be correctly followed by both the user client and the file server in order to achieve mutual authentication. Security of this mechanism is based on the strength of the cryptographic techniques employed by the handshake protocol.

Threat 4: Information Leakage via Passive Monitoring

The trust model assumes the existence of passibe attackers. The system must ensure that all communications between the user client and server applications are hidden from outside observers. This will ensure that file contents remain private, and that tokens cannot be stolen in transit.

In order to address this threat, symmetric key encryption will be used to encrypt all communications between the user client and the server applications after the initial handshake (outlined in Threats 1 and 3). The encryption will use AES-256. The session keys will be exchanged during the respective handshake protocols for each servers. Each server is treated as a separate session, so the keys will be different for communication with the group server and file server. The mode of operation will be CBC because EBC is not secure enough. Upon disconnect, the session will end and the session keys will be disgarded.

The correctness of this mechanism is based on the assumption that the handshake protocols are followed correctly during connection to each respective server application. The security is based on the strength of the encryption algorithm.

Summary and Notes

All four mechanisms interact together in some way to achieve greater security for the system. The handshake protocols outlines in Threats 1 and 3 result in a session key being shared which is used in for the mechanism in Threat 4. The signing of the token in Threat 2 is done by the private key generated in Threat 1 by the group server.

Here is a quick summary of the two handshake protcols:
Group Server Handshake

- User initiates connection
- Server sends back its public key
- User encrypts its challenge using the server public key
- User sends the challenge to the server
- Server decrypts the message with its private key
- Server sends back challenge + 1 appended with its signature
- Server is now authenticated
- User encrypts username, password, and session key with public key
- User sends the information to the server
- Server decrypts message using its private key
- Server verifies the user
- If user was verified, server encrypts token using session key
- Server sends back token
- Handshake complete

File Server Handshake

- User initiates connection
- Server sends back its public key
- User encrypts its challenge using the server public key
- User sends the challenge to the server
- Server decrypts the challenge using its private key
- Server sends back the challenge + 1 appended with its signature
- Server is now authenticated
- User encrypts the session key using server public key
- User sends the session key to the server
- Server decrypts session key using its private key
- Handshake now complete

User password storage: User passwords will be stored in the group server. Passwords are set when a user is created by the admin. Passwords are not stored as plaintext. Instead, they are stored as a SHA-256 hash of the password pre-appended with salt. The salt will be a random byte array of size 256bits. The salt will be different for each password. The salt will be stored along with the username and hash. Whenever a user is being verified during the handshake protocol described in Threat 1, the

password will be pre-appended with the appropriate salt and then hashed. This will be compared with the hash already saved in order to see if there is a match.