

# CS 1653: Applied Cryptography and Network Security

## Term Project, Phase 3

Lindsey “Hellman” Bieda   `leb35@pitt.edu`   Tucker “Diffie” Trainor   `tmt33@pitt.edu`

March 2, 2012

### Introduction: Cryptographic Techniques

Protecting communication over networks is a byzantine affair involving prime numbers, modulo arithmetic, and inscrutable cipher text. No one protocol is enough to cover all vulnerabilities of a secure file server implementation, and often hybridization of protocols is the best solution to thorny security issues. In our implementation of a secure file server, we will include proven methods from both private key cryptography and public key cryptography, and sometimes even combine the two. From the realm of private key cryptography, we will use AES for block cipher encryption. From public key cryptography, we use RSA public/private key encryption to leverage security and efficiency where possible. We will also implement hash functions such as SHA-1 for authenticating data where necessary.

### Threat 1: Unauthorized Token Issuance

#### Threat Description

For a user to begin using the secure file server, they must be issued a token with which they can perform all the functions of the Group and File Servers. Thus, we need to protect each user’s token from anyone other than the user. In Phase 2 of the project, we considered the users trustworthy and therefore didn’t require any credentials other than knowing the username of the token’s owner. This is almost completely insecure against any malicious agent who knows a username on the system, as they can simply enter any valid username to be issued that user’s token.

#### Mechanism Description

We can implement a password mechanism to verify that a user is who they claim to be. By requiring a login sequence consisting of a username and password tuple, we can do a basic authentication of the user. Before defining a sequence of operations to authorize issuance of a token, we must list assumptions of existing conditions:

- The user has been created by an administrator and is in possession of their correct password.
- The user has connected through the client application to the Group Server and has already saved the Group Server's public key in the file that their client uses to store public keys.
- The Group Server's private key has not been compromised.
- The user has an active connection to the Group Server.

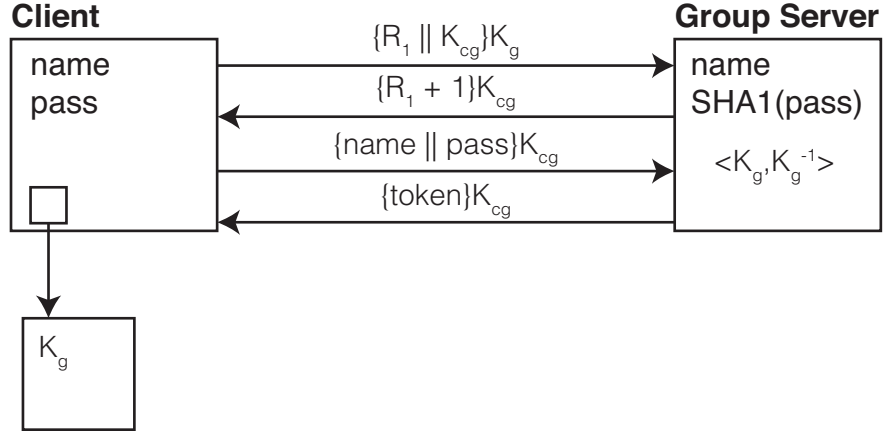


Figure 1: Threat 1 Mechanism

With these assumptions in place, we can outline the steps to maintain the confidentiality and integrity of each user's token:

1. Authenticate the Group Server – through the client, the users generates a random number  $R_1$  and concatenate it with a randomly chosen AES key  $K_{cg}$ . This data is then encrypted with the Group Server's public key  $K_g$  via RSA, all represented as  $\{R_1 \parallel K_{cg}\}_{K_g}$ . The Group Server. The Group Server decrypts this cipher using its private key  $K_g^{-1}$ . The Group Server now uses the shared symmetric key  $K_{cg}$  to encrypt the value of  $R_1 + 1$ , represented as  $\{R_1 + 1\}_{K_{cg}}$ , and sends this new cipher to the client. The client decrypts the cipher, and by noting that the value  $R_1$  and the key  $K_{cg}$  was successfully decrypted by the Group Server, the client has authenticated the Group Server.
2. Authenticate the user – Now that the shared symmetric key  $K_{cg}$  is agreed upon, the user encrypts their name and password with it, represented as  $\{name \parallel pass\}_{K_{cg}}$ , as sends the cipher text to the Group Server. The Group Server decrypts the cipher

text to recover the user's name and password, and then performs an SHA-1 message digest on the password. As the Group Server stores the hashes of passwords instead of the passwords themselves, it checks the SHA-1 hash of the user-submitted password against what it has stored. If the two hash values agree, the user is authenticated to the Group Server.

3. Issue token – The Group Server can now send the user their token over the encrypted channel, ensuring that only the user, who is authorized to receive the token, is able to access it.

We store the username with a hash of the password instead of the password in cleartext to avoid revealing passwords in the event of a compromise to the Group Server, where a hash table of usernames and hashes would be stored. The user would enter their password, which would be sent to the Group Server over an encrypted channel. If the hashes match, the user is authenticated as who they claim to be and issued their token, otherwise access is denied.

## **Correctness and Security of Mechanism**

To maintain the security of the password authentication process, there are two layers that need to be maintained. The password must be hashed so that it remains obfuscated after the user enters it into a login prompt. As a measure of correctness, the hash used by the client should match that which was used by the authenticating server. Additionally, the exchange between the client and server must be encrypted so that the hash value is obscured from an eavesdropper. The inclusion of a random IV in the encryption process is also necessary as a measure to prevent replay attacks.

## **Threat 2: Token Modification/Forgery**

### **Threat Description**

By modifying or forging tokens issued by the Group Server, a user may be able to gain access to files that would otherwise be forbidden. By changing the group information embedded in a token, a malicious user can access groups that they are not members of, which would grant unauthorized access to files belonging to those groups. Furthermore, if a malicious user assigns him or herself as the owner of a specific group, he or she will have the ability to add or delete group members, as well as freely modify files belonging to that group. This threat is a direct attack on the integrity of a secure group server.

### **Mechanism Description**

In order to maintain the integrity of token issuance, we should verify that a token is valid every time an operation involving a token is invoked. The best way to verify would be

to match the token in question with the one on the Group Server. To efficiently perform this mechanism, we take a hash of both tokens, create an authenticated and encrypted channel between the user's application and the Group Server, and compare the hashes. If the hashes match, then the user is using the same token that is on the Group Server and is thus valid.

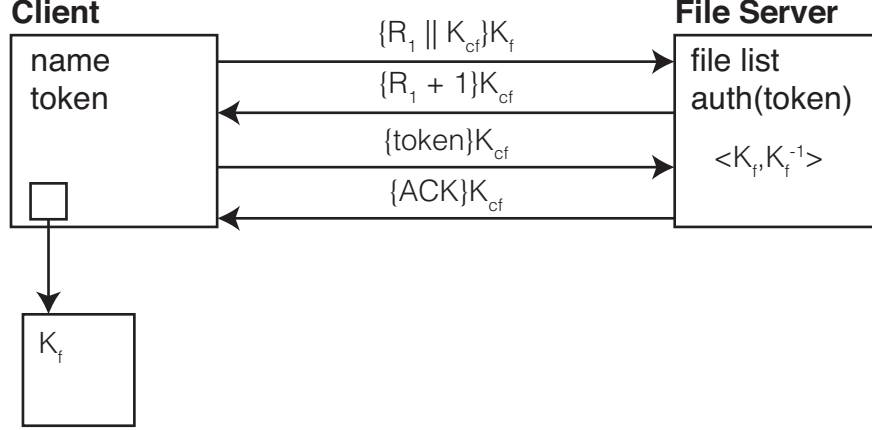


Figure 2: Threat 2 Mechanism

## Correctness and Security of Mechanism

As the hashing process does not necessarily require explanation, the security of the mechanism lies with the verification channel between the Group Server and the application requesting validation. If we can maintain the security of the channel and introduce a degree of randomness to foil replay attacks, then we should be able to eliminate token fraud.

## Threat 3: Unauthorized File Servers

### Threat Description

The purpose of a secure file server is that, put simply, your files are secure. If a user can unknowingly be directed to an unauthorized file server, any other security safeguards are rendered moot, thus threatening the confidentiality of the user's files and the perceived integrity of the entire service. If a user can be convinced that they are connected to file server  $s$  while actually being connected to a malicious file server  $s'$ , they are at risk of uploading confidential data to an untrusted source or downloading malicious content from an untrusted source.

## Mechanism Description

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 keys as a way of validating a file server. Upon the first connection to a server  $s$ , the user will be asked if they wish to accept the public key  $s_k$ . If the user agrees,  $s_k$  is stored by the user in their client application along with other identifying details of the server (e.g. server name, IP address, etc.). When the user wishes to authenticate the server, he or she can use  $s_k$  to encrypt a challenge to  $s$ . If  $s$  is in possession of the private key  $s_k^{-1}$ , then  $s$  is able to return the challenge to the user and authentication is complete. An unauthorized server  $s'$  would be unable to correctly guess  $s_k^{-1}$  and therefore could not decrypt of the challenge and complete authentication.

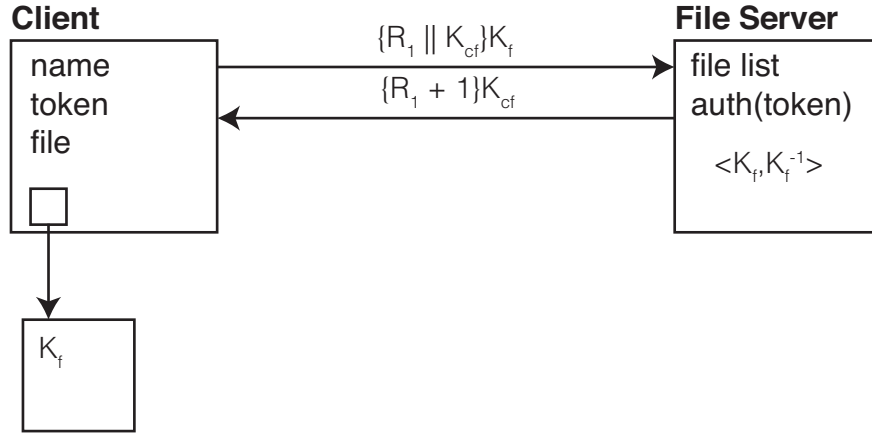


Figure 3: Threat 3 Mechanism

## Correctness and Security of Mechanism

The main threat to this mechanism is replay attacks, where a passive monitor might see a repeat of a challenge and be able to perform a man-in-the-middle attack. Therefore, a level of randomness in the challenge as well as encrypting the exchange between the user and  $s$  may be necessary to ensure correctness and security.

## Threat 4: Information Leakage via Passive Monitoring

### Threat Description

Suppose Eve can listen to an information exchange between Alice and Bob. Even without being able to interrupt or modify the exchange, Eve can still glean enough information to perform malicious acts. If insufficient security is in place, Eve may be able to gather enough data to

- know the contents of the exchanges;
- to impersonate Alice or Bob;
- use offline password guessing to discover passwords or other secret information.

Eve does not need to be an active participant in a conversation to illicitly benefit from it, and thus exchanges between Alice and Bob must be kept secure. Or, to put into context of a secure file server, any exchanges between a user and any server (or even between servers) must be kept confidential if proper security is to be maintained.

### Mechanism Description

To maintain a secure channel between Alice and Bob during a continuous series of messages, the most efficient solution may be to create a session key between them. A session key avoids having to recreate a secure channel after each exchange, resulting in increased efficiency over alternative methods that would require more frequent overhead or computation. To create a session key, we can implement public-key authentication protocols to provide authentication between two parties and provide them with a shared secret key. The secret key can then be used as a password for symmetric key encryption of all communication during the lifetime of the session. The steps to follow are similar to Threat 2, in which both parties have public/private key pairs.

### Correctness and Security of Mechanism

If the key exchange is performed securely, then each session is properly encrypted.

### Summary and Errata

There are common elements to the protocols that address the above threats. Since Threat 4 involves eavesdropping, most if not all communication over the network will need to be encrypted. Thus, handshake protocols will be present in many of the implementations we have discussed.

Before further elaboration can be done here, the implementation will need to be performed. As all good intentions lead to something else, the process of implementing the protocols will undoubtedly lead to changes in design to accommodate shifting requirements or practicalities.