Tim OShea and Matt Rubel

Term Project: Phase 3 Proposal

Overview

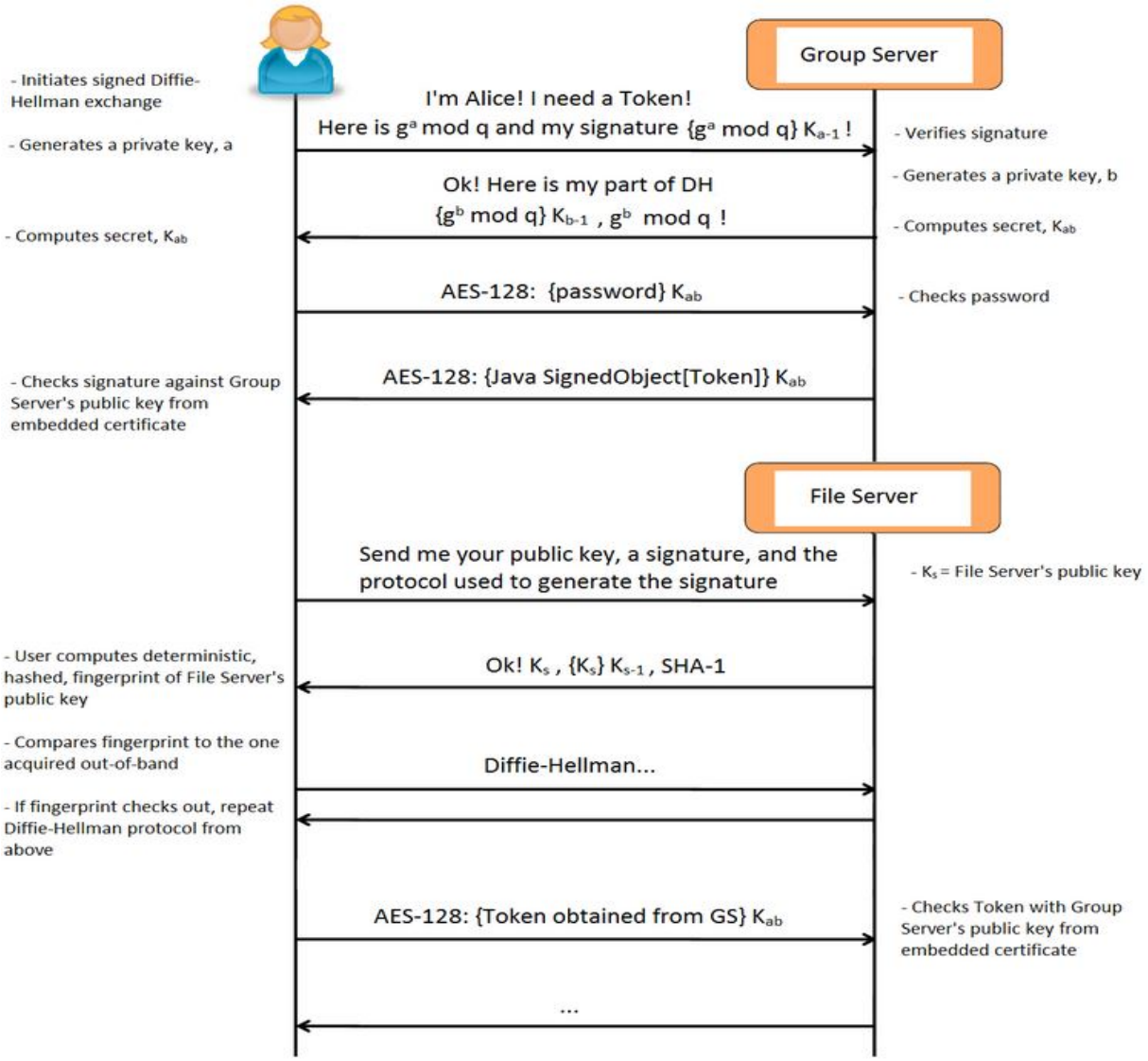
Intro

Based on our first draft, one major change we would like to implement is to embed all principals with a copy of the Group Server's digital certificate. Because in this phase the Group Server is entirely trusted the Group Server will be the Root Certificate Authority. All principals will come embedded with the Group Server's certificate because the Group Server will have to self sign its certificate. This approach will allow File Server's to validate that user tokens are legitimate. All references made to "encrypting with 'x's' public key" will be regarding using RSA-1024. All hashing will be assumed to be SHA-1. All symmetric keys will be implemented using AES-128. For the Diffie-Hellman algorithm we will use 2048 bits. Our g and q values will be discussed below.

To authenticate users to the Group Server we will have all users login with a password. Before asking for a password though the Group Server and user will establish a shared key using Diffie-Hellman which will be used for all subsequent communication. Only after the user passes authentication will a token be issued. To ensure that tokens are not tampered with for malicious purposes every token will be signed using the Group Server's private key. This makes it impossible for the user or any malicious party to modify the token that the Group Server has issued. When a user sends its token to a File Server in order to gain access the File Server can verify the token using the Group Server's public key from the pre installed certificate, guaranteeing that the token hasn't been tampered with.

Lastly, for users to authenticate the File Servers we operate under the assumption that users have out-of-band correspondence with File Server administrators. This correspondence will involve the exchange of RSA public key fingerprints. We make this assumption because for this phase of the project anybody can operate a File Server and there is no File Server "overseer". Therefore, because users are aware that there is no "overseer" they put the responsibility on themselves to seek out the File Server that they want access to and acquire the proper credentials.


Group Server
-Generates challenge, c and sends to server
Please encrypt this value, c, with your private key, so I know it comes from you
-Encrypts c with its private key
{c}Ka-1
-Decrypts returned challenge with GS's public key and verifies that the result is equivalent to the c it initally generated


Group Server
- Initiates signed Diffie-Hellman exchange
I'm Alice! I need a Token!
Here is g^a mod q and my signature {g^a mod q} Ka-1 !
- Verifies signature
- Generates a private key, a
- Generates a private key, b
Ok! Here is my part of DH
{g^b mod q} Kb-1 , g^b mod q !
- Computes secret, Kab
- Computes secret, Kab
AES-128: {password} Kab
- Checks password
AES-128: {Java SignedObject[Token]} Kab
- Checks signature against Group Server's public key from embedded certificate
File Server
Send me your public key, a signature, and the protocol used to generate the signature
- Ks = File Server's public key
- User computes deterministic, hashed, fingerprint of File Server's public key
Ok! Ks , {Ks} Ks-1 , SHA-1
- Compares fingerprint to the one acquired out-of-band
Diffie-Hellman...
- If fingerprint checks out, repeat Diffie-Hellman protocol from above
AES-128: {Token obtained from GS} Kab
- Checks Token with Group Server's public key from embedded certificate
...

Threat 1: *Unauthorized Token Issuance*

The lack of an authentication process is a very serious flaw in the current version our file sharing system. Suppose an adversary wants access to all of the files Bob has access to. All he needs is the account name of Bob and the adversary has access, modification and deletion power to all of the files in the File Servers that Bob has access to.

Our proposed mechanism to alleviate this issue is to implement a password system that would require all users to enter a password at login. Upon account creation a new password will have to be entered for the account. This new password will be salted, hashed, and entered into the userlist on the Group Server, along with the associated salt. The hashing of the password ensures that the plaintext version of the password would be very hard to deduce in case an adversary is able to gain access to the userlist. We will have the salt be randomly generated at password creation and be 16-characters (128-bit).

Every time a user tries to login, the client will first authenticate the group server. The client will do this by sending the group server a challenge value, c. The c should be a very large integer, one that would be nearly impossible for an adversary to "guess". We would have the value be a 32-bit integer, giving an adversary (spoof group server) an infeasible chance at guessing it (approximately 1/4.3million). The group server would have to use its private key to encrypt this integer, then the user can decrypt this with the group server's public key and should get the same value, c, after the decryption. Then, the client and group server will go through a Diffie-Hellman key exchange. The associated q value will be 2^2048 - 2^1984 - 1 + 2^64 * { [2^1918 pi] + 124476 } and the g will be 2[1]. After the Diffie-Hellman exchange the user must enter their password which the client will encrypt with the shared key and send to the group server. The group server will then decrypt the password string with the key and add the specific user's salt to the end and run it through the hash function. The hashed password would then be checked against the hashed password stored in the userlist. As long as the hashed passwords match correctly, a token will be issued.

Threat 2: *Token Modification/Forgery*

The main goal for users that would attempt to forge or modify tokens is access to files that don't belong to them. A secondary goal might be to just cause general mayhem and delete users or groups from the Group Server. Being able to detect token forgery/modification will fulfill the 'confidentiality' principle for every group and its files.

To protect against this threat we will have the Group Server digitally sign every token and send the signature along with token to the user. User's will then need to present the Token and signature together to

File Servers to authenticate themselves. By going the digital signature route File Servers that ask for the user's token can just verify the signature with the Token and the Group Server's public key instead of explicitly trusting that the users haven't tampered with the token since it was issued. Because all File Servers will come packaged with the Group Server's public key they will only be able to verify signatures signed by the Group Server's private key. If users create a phony signature with a different private key the File Servers will not be able to verify the signature. Digital signatures by the Group Server addresses token modification because the Group Server is the only one that has access to the private key that it uses to sign. If a malicious user tried to make their own token with a different private key, the token would be unreadable.

Threat 3: *Unauthorized File Servers*

Malicious File Servers pose a direct risk to everyone using the file sharing system. In contrast to threat two where someone that modifies their token might not have their presence felt at all, an unauthorized File Server has the opportunity to severely impact anyone that comes into contact with it. Someone in control of a malicious File Server can steal your files, and potentially your password if they prompt you for it.

To address this issue we will have all File Servers send users their public key and the public key's fingerprint. User's will then hash their own fingerprint of said public key. Because the hashing protocol is deterministic the user can now compare the fingerprint generated with the one received out-of-band while communicating with the File Server's administrator. Finally, if the public key checks out the user can then initiate the Diffie-Hellman protocol to generate a symmetric key just like it did with the Group Server. The symmetric key will be more useful than asymmetric because users and File Servers could potentially exchange large, data intensive documents.

Users will then store the File Server's IP, public key, and public key fingerprint offline for future use. For every connection to a File Server the client will look up the IP to see if the fingerprint still matches, and also to see if the fingerprint is unique. If a File Server shows a fingerprint that the user already has stored with another File Server, the user will be warned that the File Server they are connecting to is potentially trying to spoof them.

Threat 4: *Information Leakage via Passive Monitoring*

Passive monitoring by a malicious party is a constant threat that can not be taken lightly. There is no way in which we can prevent passive monitoring without ceasing all communication. While we cannot stop

adversaries from monitoring this communication, we can encrypt it so that the adversaries cannot make use of the information passage that they are monitoring.

Communication between users and File Servers will be data intensive so we will employ symmetric-key cryptography. We will start every session between users and File Server with a signed Diffie-Hellman-Merkle exchange. After the keys are established all communication will be encrypted using the appropriate key. We will set the q to be equal to 2^2048 - 2^1984 - 1 + 2^64 * { [2^1918 pi] + 124476 } as this will allow for a secure exchange with a g value of 2[1]. The symmetric key algorithm we will use is AES-128, implementing chain blocking cipher. Our implementation will randomly generate a 128-bit IV for this implementation of CBC.

The importance of generating new keys every time a user connects to a File Server is that adversaries can not crack an old key and use it in a future session. With the correctly-implemented Diffie-Hellman exchange, both the user and server parties should be able to produce the same key while adversaries will not be able to figure out this key without knowing either of the secret integers that the user or the server decide upon when generating their key. Using this key, the user and servers can encrypt and decrypt information safely.

We chose symmetric key cryptography for communication with the Group Server also for two reasons: The Group Server will need to send the user a (potentially large) Token and because the Group Server could end up potentially servicing a lot of users. It would become unwieldy to store the public keys of every user on the Group Server. Ephemeral, Diffie-Hellman session keys would be a better approach in our opinion.

Conclusion

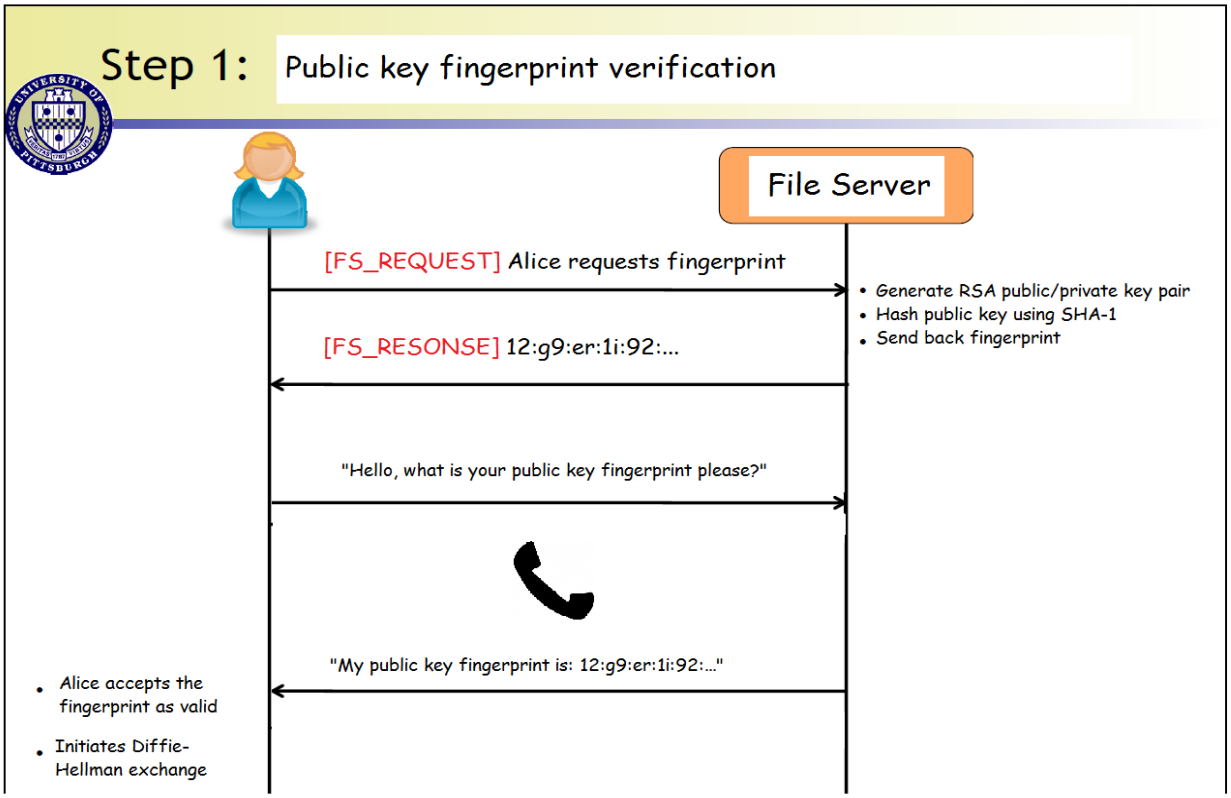
The one big assumption that drove our design was that the Group Server has no knowledge of running File Servers. If this wasn't the case a mediated authentication protocol would have have been the best option. Because the Group Server is assumed to be trusted in this phase of the project, all communication between users and File Servers could have gone through the Group Server, but because File Servers can pop up anytime we needed to take a different approach. We chose to have our Group Server sign all tokens with their private key so that any file server or third party could verify the token was issued by the group server. The file server tracks authorized users via these tokens, so it is important that they cannot be tampered with.

Because there is no mediator our system becomes very "paranoid", the users don't trust the File Servers because they believe they're trying to steal their files, and the File Servers don't trust the users because they expect users to modify/forge their tokens in order to gain access to files they're not supposed to have access to. At the very highest level though, even if a token forger and a phony File Server want to

communicate, they still need a way to do this away from the eyes of a third malicious user running Wireshark. To accomplish this all parties must go through a Diffie-Hellman exchange and talk using AES.

A digitally signed token from the Group Server gives File Servers peace of mind that users trying to connect aren't malicious, but on the other hand, having users authenticate the File Servers they are connecting to was the most difficult problem to propose a solution for. Because anyone can run a File Server, the responsibility must be put on the user to know who they are connecting to. Our implementation will involve the user having to ask the File Server administrator for the RSA fingerprint that is produced when they first connect.

[1] http://datatracker.ietf.org/doc/rfc3526/?include_text=1



Step 2: Signed Diffie-Hellman exchange and AES file encryption

Lecture 11...