Group PPKJJ: Phase 3 Proposal

Introduction

Parv, Purva

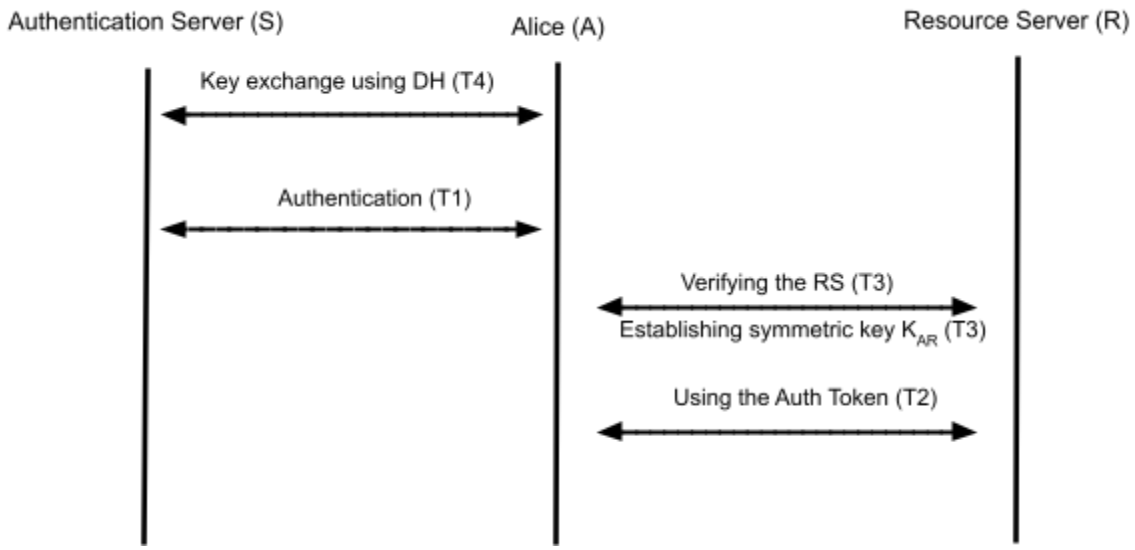
5chan is a content sharing platform with a system design that involves communication between the user and a variety of servers, and storing of a large amount of sensitive data in these servers. The design introduces a number of threats and requires mechanisms to ensure that the system remains secure. To implement this, we have used numerous techniques for proper mutual authentication between the user and the servers and to secure communication from eavesdroppers.

To elaborate, the algorithms implemented in the system are:

- RSA using 4096-bit key pair for public key cryptography. OAEP is used for padding.
- AES using a 256-bit key in CBC mode for symmetric key cryptography. PKCS7 is used for padding.
- Diffie-Hellman for key exchange. The prime number (p) used is 4096-bits. a, b and g are cryptographically secure random values.
- SHA-256 for hashing

For both symmetric and public key cryptographic algorithms, we use the *pycryptodome* library. For SHA-256, we use the hashlib library. We use the *sympy* library to generate a 4096-bit prime number for the Diffie-Hellman algorithm.

The Unified Diagram for the Authentication Server is:



Please refer to the diagrams for all the threats for elaboration.

T1 Unauthorized Token Issuance

Parv

Threat:

Under our current build, users are authenticated using only their username. This makes it possible for an attacker to impersonate another user. With this, it can gain access to the user's authentication token and hence post, comment or edit posts/comments on behalf of this user. This might lead to sensitive or offensive content coming out of an innocent victim user. It can also lead to the user's private information being disclosed.

Mechanism:

A password authentication system ensures only the user with the correct user-password pair receives a proper token from the authentication server.

*All communication between the user and the authentication server will be encrypted with the implementation described in T4.

During sign up:

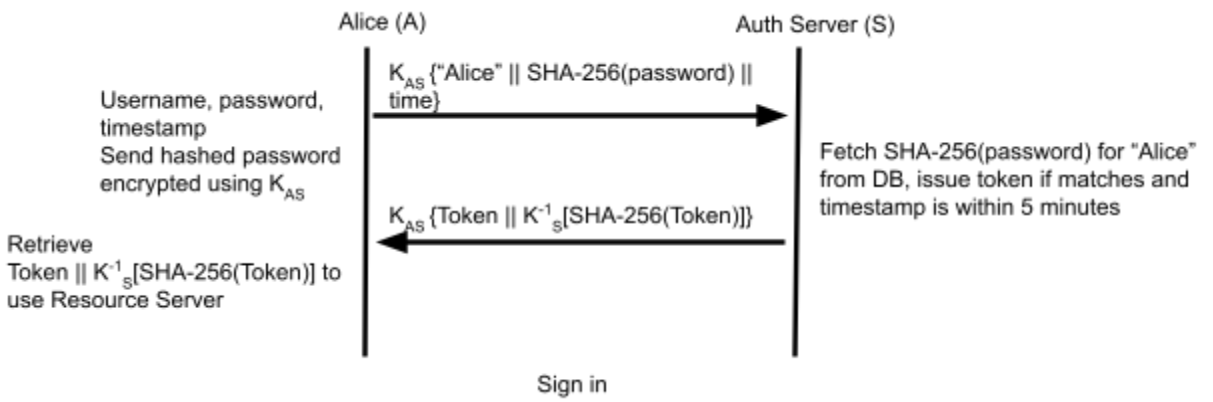
1. User should enter the username and the password.
2. The new password should be more than 8 characters, should have at least 1 special character, should have at least 1 number and should have both upper and lower case alphabets.
3. The client hashes the password using SHA-256.
4. The client sends the unique username, this hashed value, and the current timestamp to the authentication server to store. K_{AS} (“Alice” || H(password) || time)

During sign in:

1. Once the account is created, the user should be able to login using its password.
2. On entering the username and the password, the client will hash the password using SHA-256 and send the username and hash value to the authentication server.
3. The authentication server will check the timestamp of the message. If the message was received 5 minutes or more after the timestamp was created, assume the message is a part of a replay attack and deny access.
4. The authentication server will check the hash value against its stored value for that user. If it matches, it will go on to generate an authorization token for this user.

The details about the token generation and what the user does with it are mentioned in the subsequent sections.

The following diagram is after the symmetric key exchange (K_{AS}) using implementation in T4 has already occurred between Alice and the authentication server. H(Token) in the diagram means SHA-256(Token).



Assumptions:

- Each user only knows its username and the password associated with it.
- The authentication server is entirely trustworthy. This implies that the passwords stored with the server are secure, and the values can not be read or modified by an adversary.
- The authentication server does not require to be authenticated. The assumption is that there is no adversary trying to pretend to be the authentication server.

Justification:

- Given that the authentication server is entirely trustworthy and keeps the data secure, we can trust that the stored hash passwords would not get leaked.
- Our implementation for T4 ensures that the communication between the user and the authentication server is encrypted. Essentially, the hashed value will also be encrypted so that even if an adversary is snooping this communication, it would not be able to retrieve the hash value. This is necessary because if an adversary gets its hands on the hash value, it can basically impersonate the user and send this hash value itself to gain the authorization token from the authentication server.
- With the property of hash functions to be preimage resistant, the password can't be derived from looking at the hash value.

T2 Token Modification/Forgery

Jiang, Kadin

Threat:

In our current build, the token is a simple set of plaintext corresponding to each privilege the user concatenated together by the authentication server. There is no mechanism in place that protects the integrity of the token.

After receiving the token from auth server, a malicious user might modify the token to gain unauthorized access to resources. There are multiple attacks that this can lead to, such as privilege escalation, where an attacker gains administrator privilege and misconfigures our system, and information disclosure, where an attacker gains access to a user's private information.

This would be problematic because resource servers grant access to users based on the token they receive from the user.

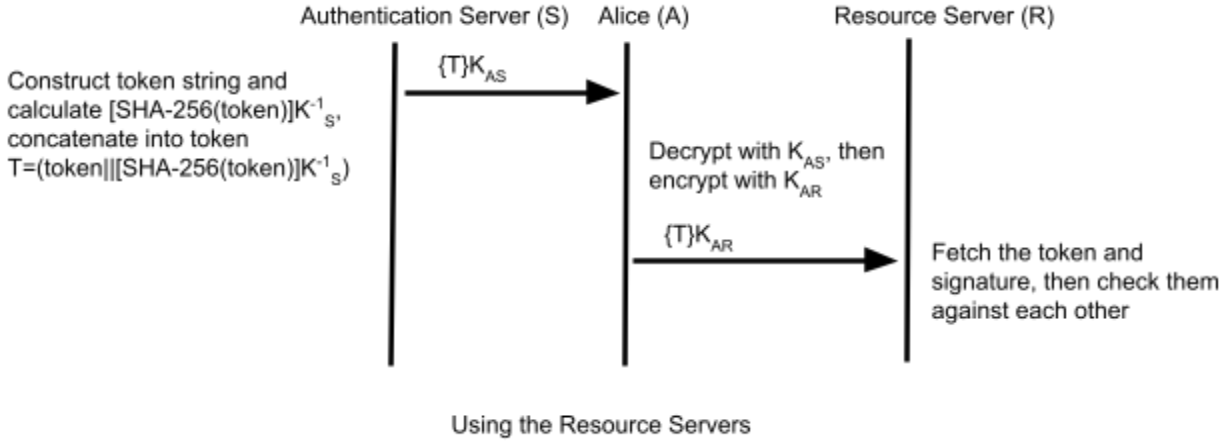
Mechanism:

On resource server initialization, resource server downloads public key from auth server. Each token issued is signed by auth server before distribution, and verified by resource server upon receiving.

- When a resource server is created, it must be configured with the IP address or domain name of the authentication server. During initialization, the resource server will download and store the authentication server's public key.
- The authentication server will first generate the following string for an authenticated user, where User_id is a 4 byte hexadecimal string with 4 bits of hyphens and privileges are a string specifying the privilege (26 character word): <user_id>|<privilege 1>|...|<privilege n>|<session_id>
- Then the auth server will hash the string using SHA-256 and sign the hash value with its 4096-bit RSA private key, and concatenate the signed hash at the end of string.

- The resulting token will be transmitted to the authenticated client, which will be transmitted to the resource servers each time the client makes a request. The resource servers will verify the plaintext portion of the token string against the signature. The resource server owner can download the authentication server's public key to check the signature.

The following diagram is after the Resource server is authenticated using implementation in T3 and the symmetric key exchange between Alice and the resource server has been carried out using implementation in T4.



Using the Resource Servers

Assumption:

The auth server is a centralized server that is trusted and protects its private key. The resource server owner obtained the auth server's public key.

Justification:

Given that the auth server is entirely trustworthy, the content of the token received from auth server is also trustworthy. Therefore we need only care about the integrity of the token. Assuming the auth server's private key is protected, by signing the hashed token with its private key, any modification to the token can be detected by the resource servers. This also makes it imperative that the authentication server's public key is downloaded to the resource server at the time of server initialization.

T3 Unauthorized Resource Servers

Jiang, Jack

Threat:

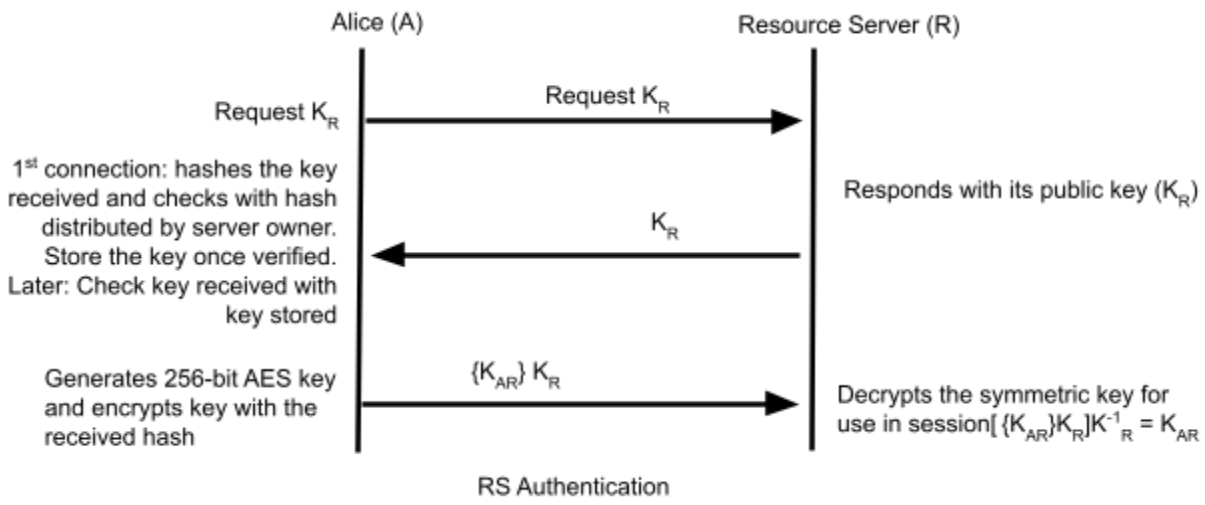
In our current build, resource servers are all assumed to be legitimate. This makes it very easy for adversaries to fake resource servers and collect data about a given user. It will be catastrophic if an adversary can get hold of a moderator's account. This will lead to user's posts/comments being deleted and users being banned at the adversary's will.

Mechanism:

Before each session, a challenge-response procedure is in place to verify the identity of the resource server using the public key.

1. When the resource server gets created, it generates a 4096-bit RSA key-pair and the owner distributes its public-key hashed using SHA-256 out of band.
2. When a client connects to a resource server, it will ask for its public key. Upon receiving, it will hash this public key and verify against the stored hashed value. On first time connection to the resource server, no stored key will exist, so it will be the client's burden to check the public key returned with the previously distributed key.
3. Because the client's identity can be confirmed by the authorization token it obtained from the authentication server, the resource server does not need to authenticate it again. The client will then generate a 256-bit AES key (K_{AR}), encrypt the key with the RS server's public key, and send the encrypted key to the RS server to establish a secure channel.
4. If any of the above steps fail, the process will be terminated and no connection will be established.

Once the first-time connection is done, Alice keeps a record of R's public key associated with the IP. This diagram is to carry out any of the subsequent connections.



Assumption:

The hashed RSA key distributed by the server owner correctly identifies the server. The client is assumed to check the hashed RSA key distributed by the server owner with the key returned by the server on first connection.

Justification:

After the very first connection, the user will store the resource server's public key associated with its IP. With this information, the user will lookup the public key stored in its record to verify its authenticity (As SSH does with the known_hosts file). The mutual authentication shown in the implementation above can then be carried out securely.

T4 Information Leakage via Passive Monitoring

Jiang

Threat:

A passive attacker can listen to the communication between client and auth/resource servers and gain confidential information such as password and token. Once the attacker gains access to such information, our assumption that each user only has access to its own token is broken, and the attacker may create a new user and access unauthorized resources using another user's token.

Mechanism:

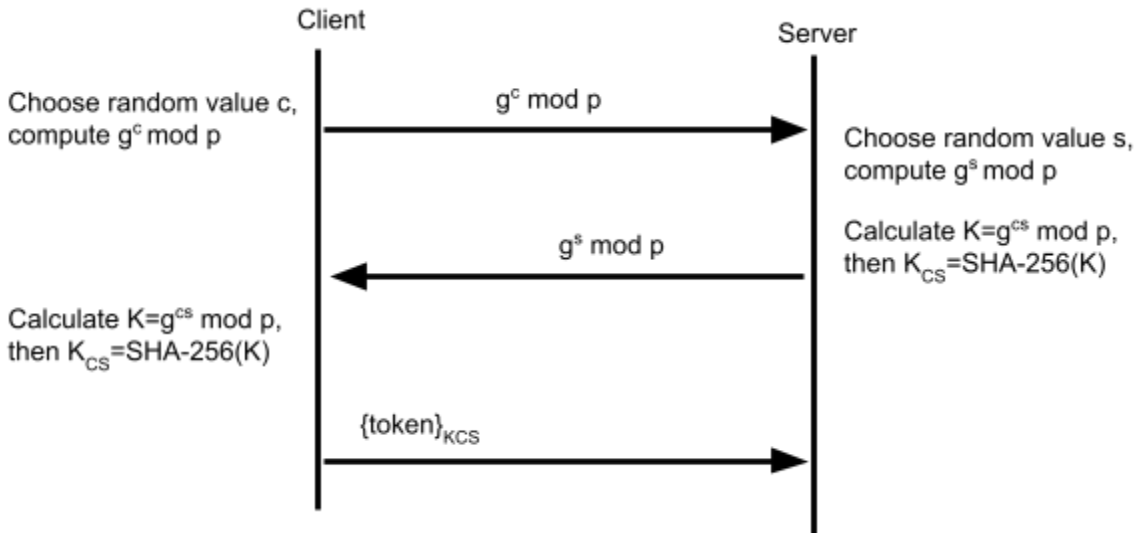
For each session with each server, a symmetric key will be generated and exchanged between the client and the server. The key will be used to encrypt all further traffic in that session.

Authentication Server

- Upon connection, the authentication server generates a session ID and starts a diffie-hellman exchange to establish a shared secret associated with the session ID.
- Further communication will all be encrypted with the shared secret.
- g will be established as 5, p as a 4096-bit prime number.

Resource Server

- Symmetric key distribution is done in T3 after establishing a secure connection via the server's public key, so no further steps are required as all the communication between the user and the resource server will be done encrypted using the key.



Assumption:

The authentication server is trusted and only issues the correct token to authorized users.

Justification:

Our trust model states that there is no active attacker, therefore we run no risk of replay attacks, and an attacker cannot simply gain access to resource servers by repeating the encrypted token. Therefore, as long as it is infeasible for the attacker to decrypt the transmitted data, the system is secure.