Phase 4 Writeup

Trevor Lingle trl46, Di Wu diw26, Aaron Wu faw21

Overview

The following describes how our file system protects against the following threats: message reorder, message replay, message modification, file leakage, and token theft. These threats challenge the user's privacy and functional use of the file system. Messages will require a means to know if there has been modification, reorder, or replay during communication. This is solved by verifiable HMACs and message sequence numbers. To protect privacy of a user's files, the files will be encrypted using a changing key stored by the Group server. The key will prevent any users from accessing even leaked files which they never had access to. Finally, token theft by a compromised or malicious file server is made pointless by including the intended File server's public key in the Token. The File Server will be responsible for verifying the public key on the token matches their own. That public key is provided by the client to the Group server so that the client can verify the token is going to the same file server they are expecting.

Threats

T5 - Message Reorder, Replay, or Modification

Threat Description

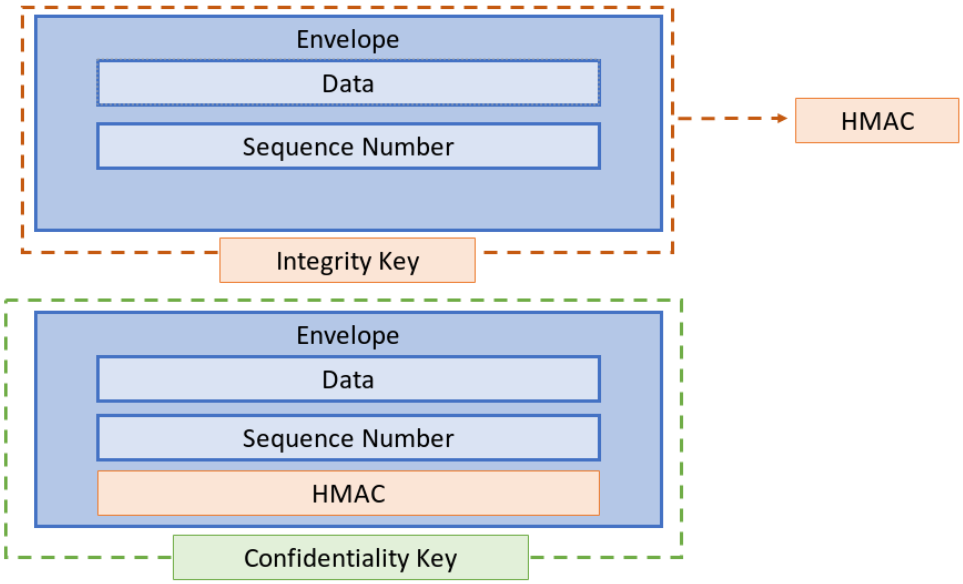
Messages sent between the client and the server could be tampered with, modified, or replayed. A malicious client may be removed from a group but replay a stored token to gain unintended access to a file. Alternatively, in an unsecured channel an attacker could replay a request with a token to up/download files they never had access to.

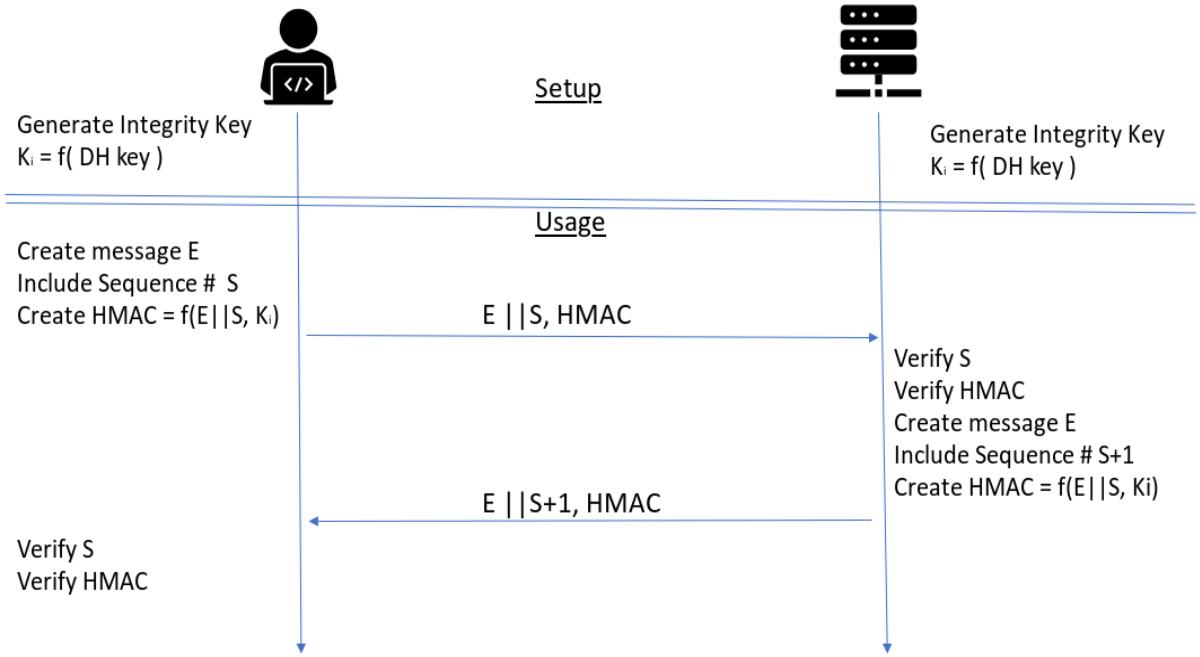
Mechanism

Message envelopes will have a Sequence Number of the envelope representing the number of messages sent from the client and server. The Envelope and sequence number will have their integrity assured by an included HMAC which will be included inside of the encrypted message

The HMAC consists of the entire envelope including the Sequence Number with an integrity key derived from a to the DH secret with concatenating a 1 bit (where as the confidentiality key uses a 0).

The server and client will verify that each message received matches the HMAC, and then verifies the Sequence Number is in order.





Mechanism Defense

Any message sent out of order will be compared to the the expected sequence number preventing replay or reordering. Due to the changing session key (T4) provided by a Diffie-Hellman exchange, messages can not be replayed in order for new sessions either.

The HMAC prevents any modifications being valid unless the attacker will be able to modify both the HMAC and the message in a meaningful way. This would require the confidentiality DH key that is assumed secret.

Notes

Integrity Key is primarily computed running SHA-2 on the Diffie-Hellman Key.

The message and sequence number will need to be converted to a formatted hashable string.

HMAC will be a SHA-2 hash using a 256 bit AES key.

T6 - File Leakage

Threat Description

File Leakage is the concept when the File Server exposes a group's files to unauthenticated users or users who are not in the group to view that file. If this were to happen, the files thought to be private, would be exposed to users who could use the files maliciously. Ensuring files are not leaked can protect personal information, trade secrets, and/or financial documentation.

Mechanism

Group Servers maintain a base key for each group as well as a decrementing value. The value represents the number of times the base key is hashed before giving it to the client with their token.

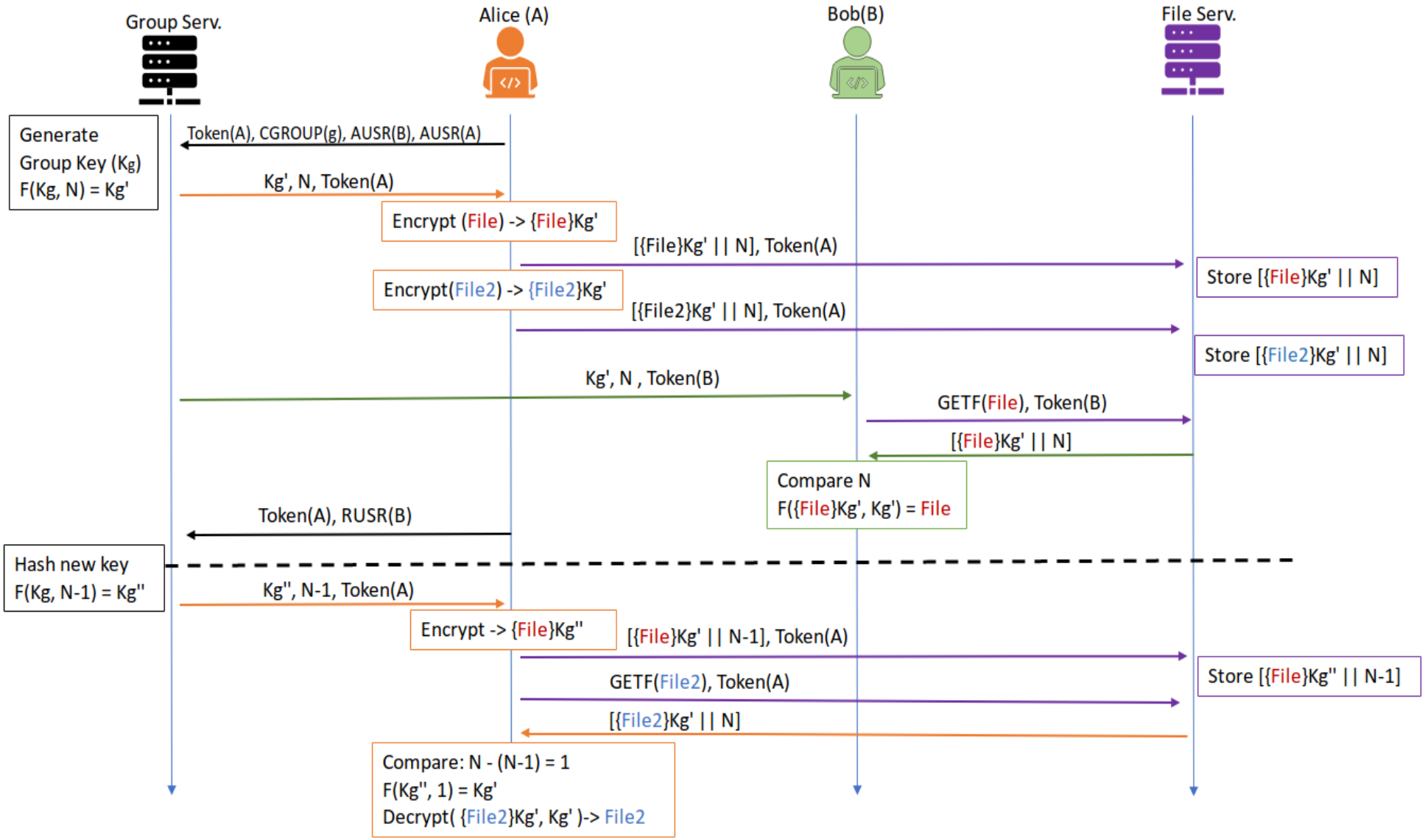
The value is decremented each time a user is removed from the group, and the key supplied with the token is hashed 1 less time. This results in a user being able to re-create older versions of the key but not newer ones.

When a client encrypts and uploads/updates a file, they use the most recent (lowest value) hashed key and uploads a file header detailing what hash of which key to use (N).

When a client needs to decrypt a file, they strip the header and hash the key. The key is hashed until it matches the key used to encrypt the file. The number of times to hash the key is stored in the file header.

When value reaches 0, a new key is generated by the group server and the value is reset. The old key is stored in a list and is passed with the new key. When the Client needs to decrypt a file that used the old key, the file header will include information to denote which key to use from the list of keys.

Tokens have a timeout value set for 24 hours after they are issued. Those timeout values are checked by the file server to ensure that they are recent. If the timeout is passed then the token is refused.


Group Serv.
Alice (A)
Bob(B)
File Serv.
Generate
Group Key (Kg)
F(Kg, N) = Kg'
Token(A), CGROUP(g), AUSR(B), AUSR(A)
Kg', N, Token(A)
Encrypt (File) -> {File}Kg'
[{File}Kg' || N], Token(A)
Store [{File}Kg' || N]
Encrypt(File2) -> {File2}Kg'
[{File2}Kg' || N], Token(A)
Store [{File2}Kg' || N]
Kg', N , Token(B)
GETF(File), Token(B)
[{File}Kg' || N]
Compare N
F({File}Kg', Kg') = File
Token(A), RUSR(B)
Hash new key
F(Kg, N-1) = Kg''
Kg'', N-1, Token(A)
Encrypt -> {File}Kg''
[{File}Kg' || N-1], Token(A)
Store [{File}Kg'' || N-1]
GETF(File2), Token(A)
[{File2}Kg' || N]
Compare: N - (N-1) = 1
F(Kg'', 1) = Kg'
Decrypt({File2}Kg', Kg')-> File2

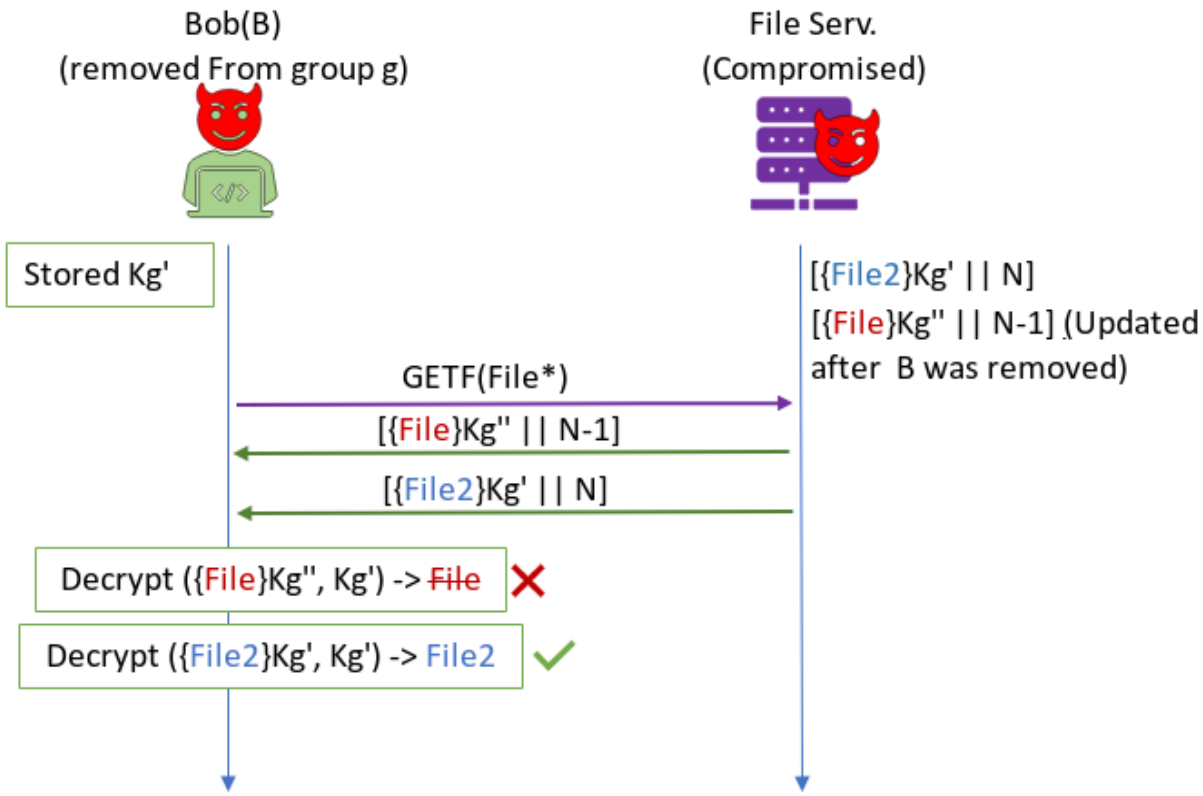
Mechanism Defense

Users removed from a group will have no way to decrypt a file updated after they have been removed. Even from a file server who does not authenticate, receiving a new file will not be able to be decrypted with the old key.

By sending a hashed version of the key, users will only be able to create older keys, not newer ones since the base keys never leave the server.

If the appended value is less than the newest key the user posses, the user will need to request a new token from the Group server, and if the key value is still more than the file has been tampered with.

Token time outs will prevent a user from keeping their token forever and accessing a group which they are no longer part of beyond the timeout.



Notes

The function applied to the base key is a SHA-2 hash to create a new key.

If a File, which has not been updated since a user was removed, is leaked, that file will still be able to be decrypted by the user possessing the key.

The base group key will be a AES256 key.

T7 - Token Theft

Threat Description

Token Theft is when a user's Token is taken out of traffic or from a malicious/compromised server and distributed to other users.

Token Theft needs to be handled or else anyone could impersonate another user and have access to all their files if they get that user's token.

The Token could be taken from a compromised file server that a malicious user has control of. The malicious user could have the server forward them another user's token to impersonate them on a different file server.

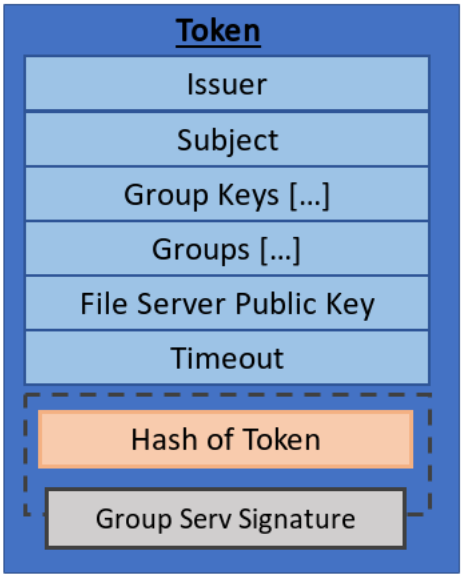
A malicious user may also be able to pull the token out of traffic and forward that token on to a file server to impersonate that user.

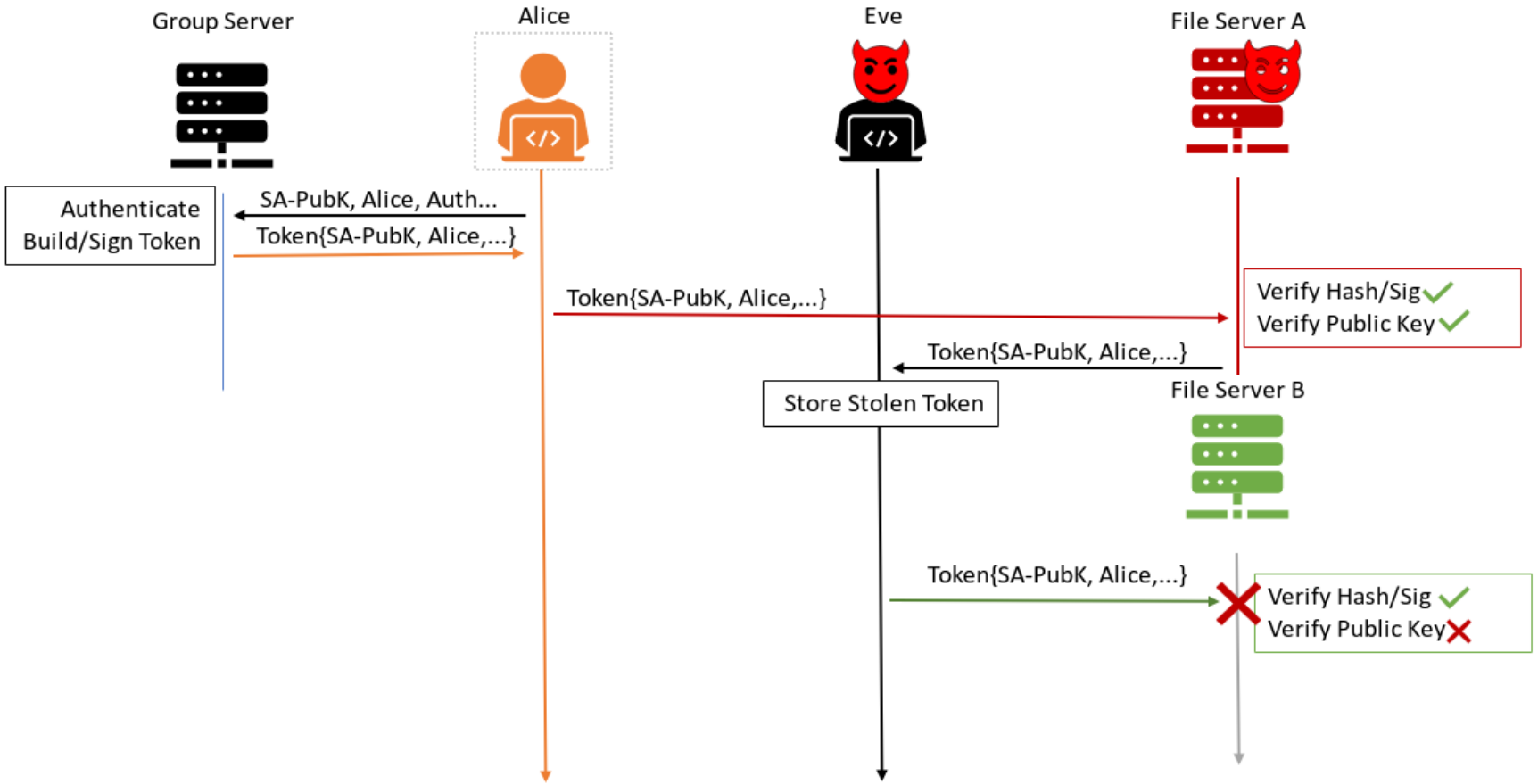
Mechanism

Tokens will include the File Server public key that the client wants to connect to. The Client sends the Public Key in addition to its request for a token.

The public key is also signed when the entire key is signed by the Group Server.

The File Server will verify the signed hash on the token and that the public key in the token matches the server's own public key. If any part is not verified, the connection is terminated.




Group Server
Alice
Eve
File Server A
Authenticate
Build/Sign Token
SA-PubK, Alice, Auth...
Token{SA-PubK, Alice,...}
Token{SA-PubK, Alice,...}
Verify Hash/Sig
Verify Public Key
Token{SA-PubK, Alice,...}
Store Stolen Token
File Server B
Token{SA-PubK, Alice,...}
Verify Hash/Sig
Verify Public Key

Mechanism Defense

Using the public key as the identifier prevents a malicious server from impersonating a valid file server, and valid servers are unlikely to share the same identifier.

If a token is stolen they will not be able to modify the token thanks to the signed hash(T2), and therefore unable to change the identifier. Without updating the identifier, compliant file servers will not accept the token meant for a different file server modified or not.

Notes

The client uses the public key gained out of band during setup with that file server. This ensures that the same key is in the key is used to create a DH key when creating the secure channel. If a secure channel can not be created, they user will not continue to use the file server.

T1 - T4

Defense Summaries

T1 unauthorized token issuance is prevented by passwords following EKE protocol. The user must send the hash of their password to the server. Using the stored username and hashed password, nonces are created and shared with a known g and p to create a mutual key K. K is then used to send a challenge/response. Once the challenge response is verified, the Group Server sends the user the token.

T2 Token Modification is still prevented by having a signed hash of the token passed to the file server with the actual token. File server will verify the signature using the group server's public key, and then verify the hash matches the token it received.

T3 unauthorized file servers are prevented by the client only connecting to file servers after completing a Diffie-Hellman key exchange. The clients get the file server's public key out of band and thus will only be able to complete the exchange with the server possessing the corresponding private key.

T4 information leakage via passive monitoring is prevented by a signed Diffie-Hellman key exchange at the start of every communication between a group/file server and a client. The Diffie-Hellman exchange can not be easily broken by a passive attacker and provides forward secrecy.

Impact of T5 - T7 on T1 - T4

All the the threat defenses for T1 - T4 are not impacted by defenses against T5 - T7. Password verification still happens the same way. T2 is still a signed hash but the hash consists of more data now that the token includes a timestamp and the group keys. But token creation and verification still works the same way. The Diffie-Hellman exchanges done for data confidentiality are the same and occur at the beginning at each connection.