CS4032 Protocol

killian

November 27, 2016

Contents

1 General Protocol

1.1 Propagation of certs

##ile:certificate$_{\text{propogation.png}}$]]

All connecting servers must report their public keys to the auth server. The auth server can only trust the identity of the server providing the cert

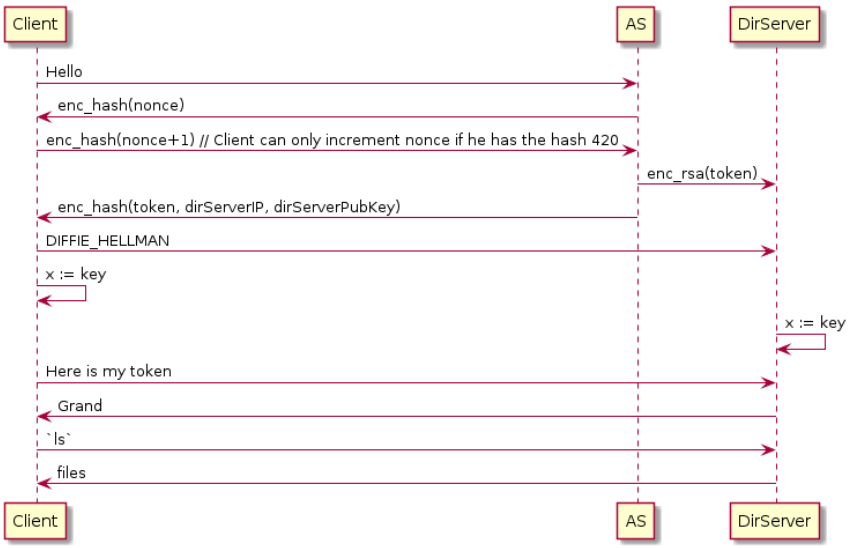
by encrpyting this channel with a pre shared key. The server admin manages the sharing of this key.

1.2 TLS and server trust model

####iletls$_{servertrust.png}$]]

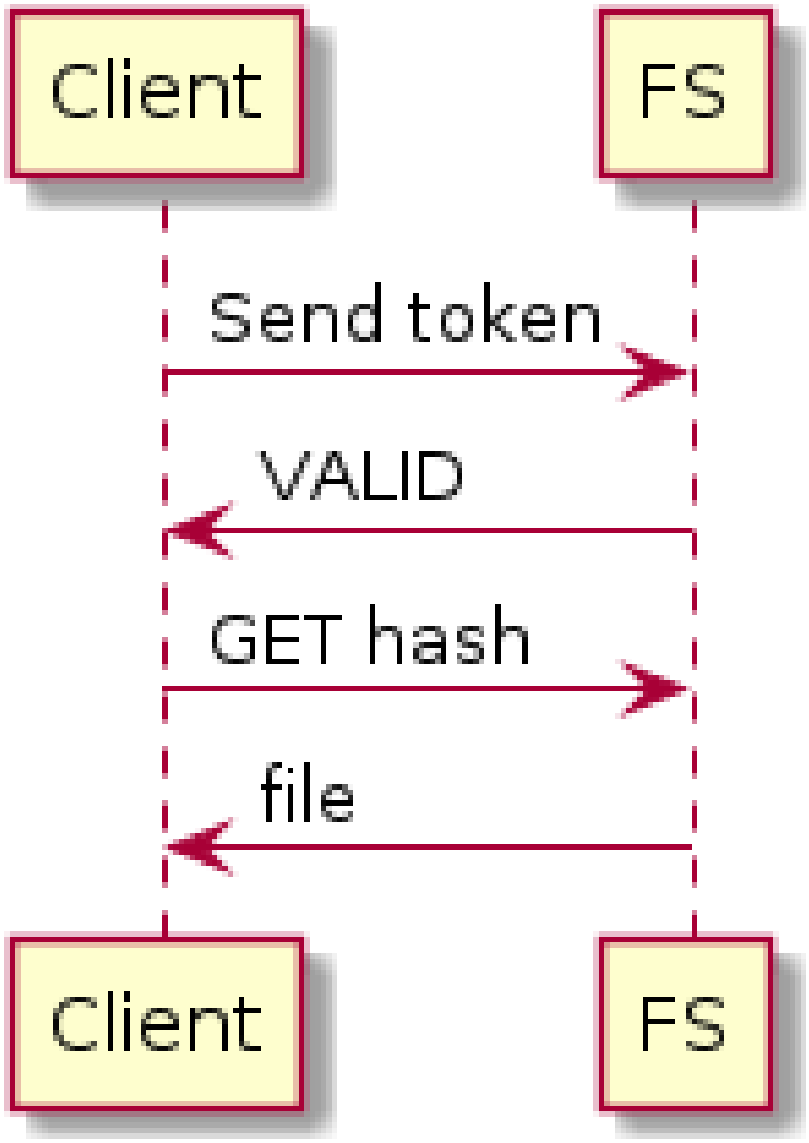
All comms are over TLS. If the admin sets everything up correctly, all servers will have the self signed certs of all the other servers, or at least the ones which they will communicate with. If a user connects to the auth server, they must decide whether to trust the auth servers cert. If they do, the auth server can vouch for the validity of the public keys for all the other servers. Therefore, the user can now communicate with encrypted and authenticated comms with integrity.

1.3 User auth

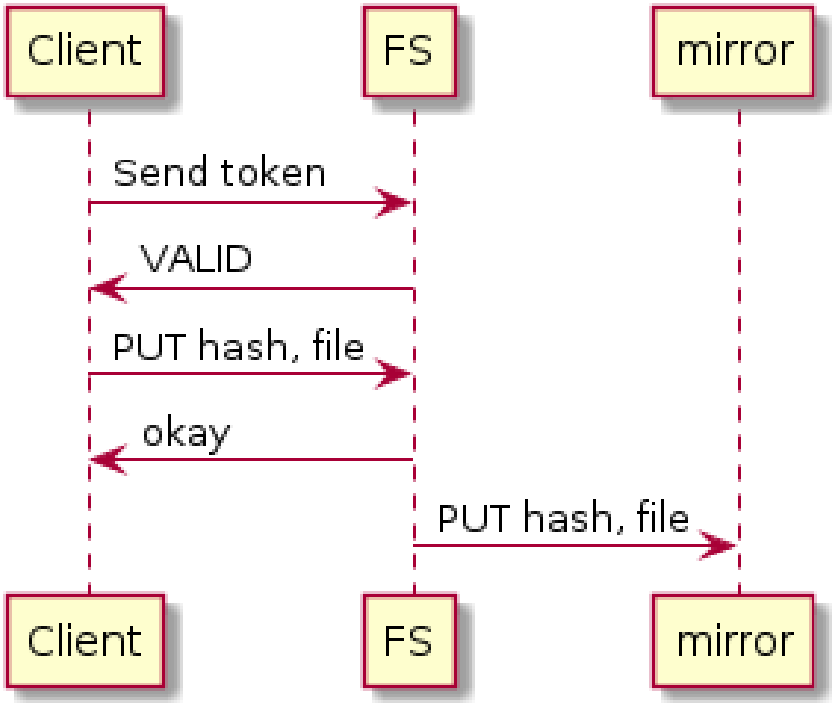


After TLS is setup to the auth server, all we need worry about is authing the user. User hashes their password, send that hash and user$_{id}$ to the auth server. Auth server then looks up the data and can verify the user. The auth server generates a random token, and distributes this to all the other servers with a specified expiry time. The user also gets this token. Very simply, if a user wants to connect to a server, they provide the token thereby proving that they have authed with the auth server.

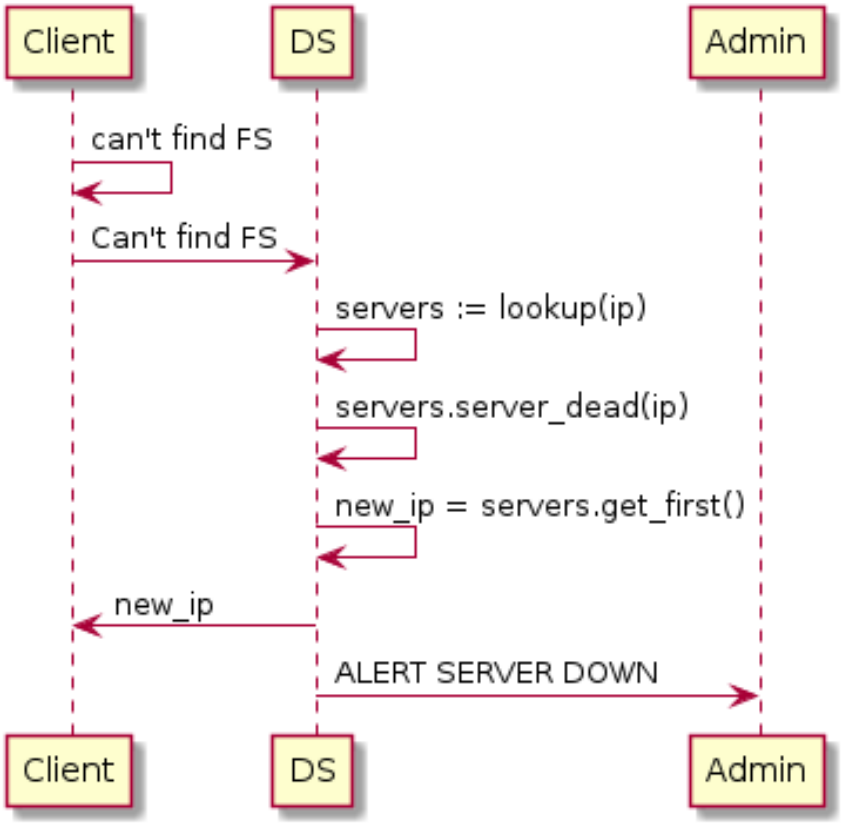
1.4 Get file from server



1.5 Put file from server



1.6 Case file server is dead



2 Security

Through TLS the main pillars of security are achieved

- Encryption
- Authentication
- Integrity

A careful admin can setup self sign certs in such a way as to guarantee this, in addition, it is very possible to add additional servers into this security model.

TLS allows us to use our simple but effective token protocol to auth users.

2.1 The token

A 3-tuple

$$token := (t, c, exp)$$

Where:

- $t := SHA256(urandom)$
- c := datetime.now()
- exp := datetime.now + 1 week

The resulting token is encrypted under AS's private key1

t needs to be a quality random value, if anyone could predict it, they could forge a token and gain access.

related to this, it's also important that we encrypt under AS's priv key so that a fileserver can prove that the token was created by the AS.

c, and exp are self explanatory. A server knows that if the expiry date has been reached, not to accept the token. In addition a server might could be configured to not accept tokens of non realistic expirys. e.g if a user got a token with expiry 5 years from now, a fileserver might want to reject. The extra value in the token just allows a bit of flexibility in the protocol in this regard.

3 Directory Service

4 Replication

5 Caching

6 Transacctions

7 Locking