Certificate Transparency

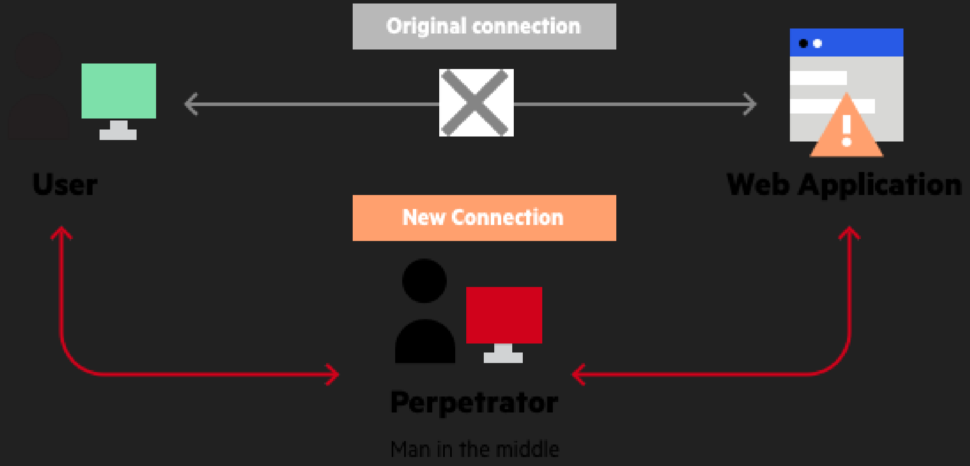
Background

In the previous video, I mentioned some network protocols that we can use to send byte streams over the internet in a reliable manner. However, how do we ensure that the byte streams are going to the right place? This is where certificate transparency comes in.

Certificate transparency is used in HTTPS, which takes traditional HTTP requests and adds an additional security layer (known as SSL/TLS).

Man In the Middle Attacks

Third party reads and intercepts HTTP packets, can see any of the information being sent.

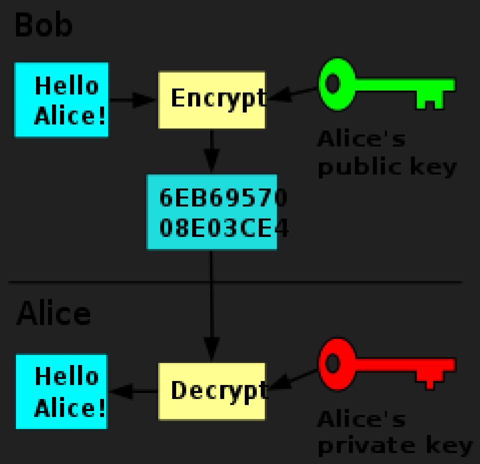


Only way to prevent data from being stolen is by encrypting it such that only the server can understand it!

RSA (Public Key) Encryption

- Server generates a public key and a private key
- It broadcasts the public key over the network to the client
- Client encodes any messages it wants to send over the network using the public key
- The server keeps the private key to itself, as the private key is the only way to decrypt the message

Client has to make sure they are using the public key from the server, and not from a man in the middle!!



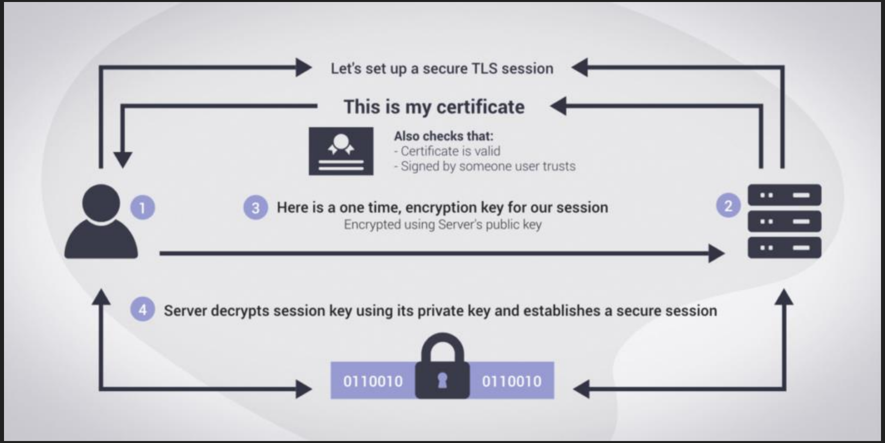
Certificate

Used to verify that a server is actually who it says it is.

Each server must register with a Certificate Authority, and the certificate provides information about the server such as its owner, its IP address, its public key, and the authority that it registered with (verifiable via a digital signature).

TLS Handshake

When a browser connects to a server, the server sends its certificate which the browser can verify via the digital signature. Upon completion, it will send some randomly generated encrypted data (using the server public key) to the server, which proves that it has the private key by sending it back properly decrypted.


Let's set up a secure TLS session
This is my certificate
Also checks that:
- Certificate is valid
- Signed by someone user trusts
1
2
3 Here is a one time, encryption key for our session
Encrypted using Server's public key
4 Server decrypts session key using its private key and establishes a secure session
0110010
0110010

Certificate Authority

CA also has a public and private key pair, and uses the private key to encrypt the certificate, which it uses as the digital signature (anyone with the public key can then decrypt the signature).

Browsers contain public keys of trusted certificate authorities, and use them to decrypt the message and check if it matches the certificate. Hence we are certain that one of the “trusted” CAs issued it.

Can we trust Certificate Authorities?

What if a CA got hacked? Then a malicious actor could start giving themselves certificates for domain names that they don’t own.

What we need is certificate transparency.

Certificate Transparency

A way of publicly broadcasting all issued certificates so that bogus certificates can be quickly noticed and thrown out.

- Certificate Logs
- Monitors
- Auditors

Certificate Logs

Certificate authorities add all certificates to append only logs, which can be queried by a web browser auditor to ensure that a certificate is in it.

Each website can then run its own set of programs, known as monitors, which poll the logs for bogus certificates and will report them to be revoked if they find any.

However, even this is not perfect:

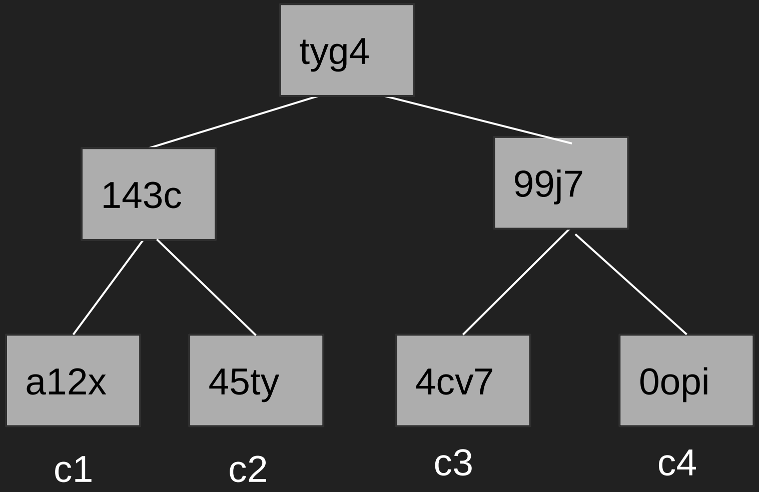
- Log shows auditor an entry and then entry is deleted before monitor catches it
- Log server shows browser and auditor different logs (forks of same log, holding two versions)

Certificate Logs Continued

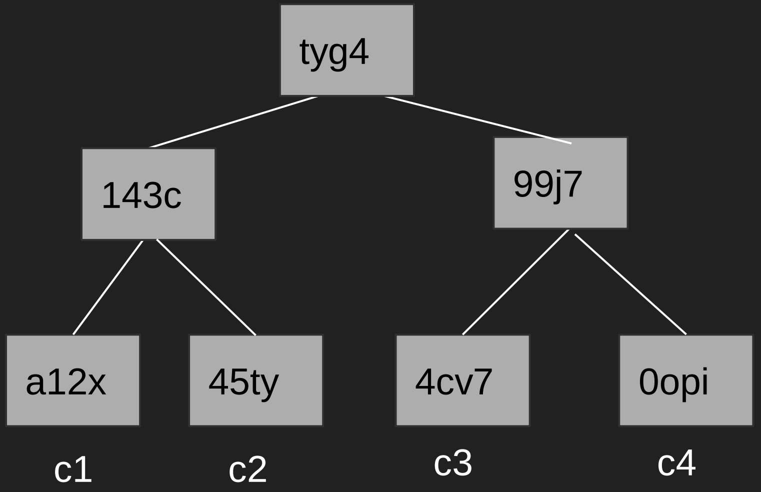
How can we detect whether an element is actually in the log and that the log server isn’t lying? Merkle Trees!!

- Generate a Merkle Tree from the log, and get the root hash
- To ensure a given sequence of elements is in the Merkle Tree, basically recreate it on the browser using said sequence of elements, and see if the root hashes match
  - Can be optimized such that you don’t need to have every element in the log, only log(n) elements of the log (basically you just need the hashes from every one of the opposite branches)
- This is really hard to be faked because it would require some other combination of elements that perfectly hashes to the root node

Merkle Tree Proof Visualized

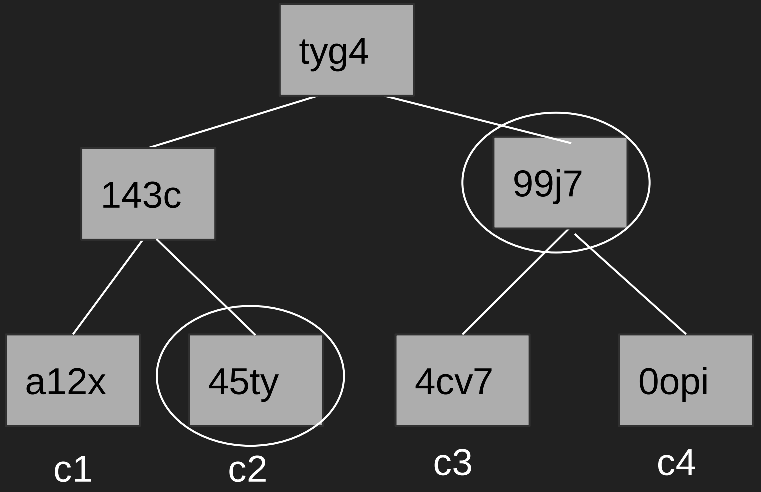


Merkle Tree Proof Visualized



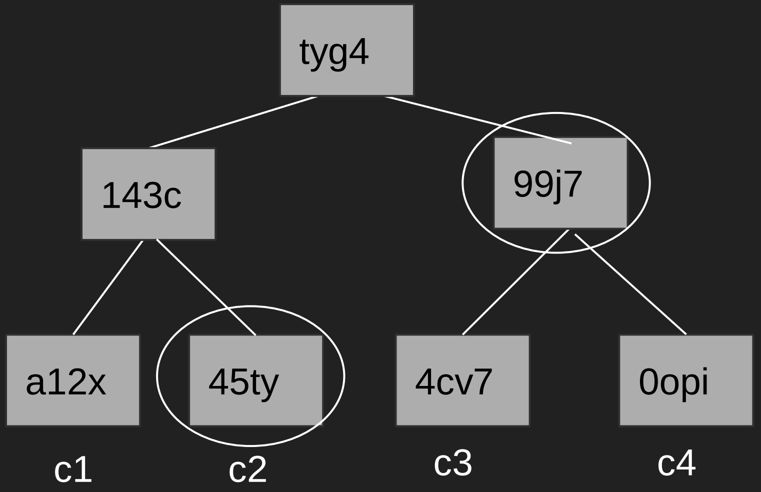
Is c1 in the log?

Merkle Tree Proof Visualized



Is c1 in the log?
Gather all the hashes from opposite branches.

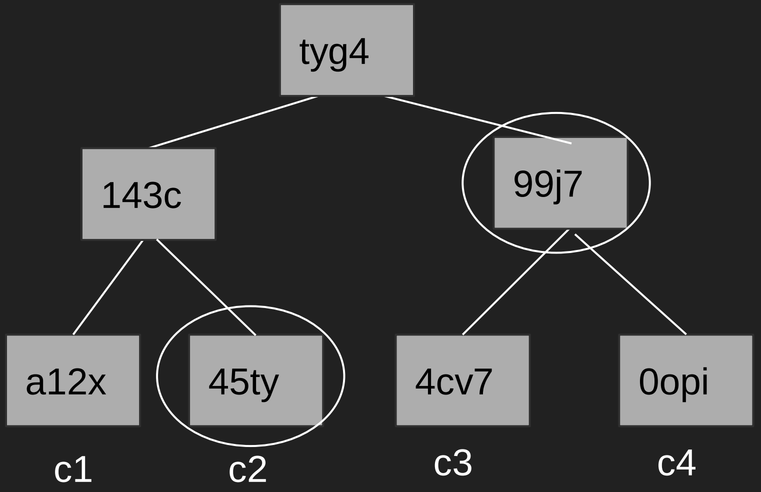
Merkle Tree Proof Visualized



Is c1 in the log?
Gather all the hashes from opposite branches.

If H(H(H(c1) + 45ty) + 99j7) = tyg4, we know that it is!

Merkle Tree Proof Visualized



Is c1 in the log?
Gather all the hashes from opposite branches.

If H(H(H(c1) + 45ty) + 99j7) = tyg4, we know that it is!

But how can we be sure that we even received a correct root hash in the first place? Fork attack! Log shows monitors and browser different versions of the log.

Fork Consistency

In order to make sure that it is not being shown a bogus fork of the log, the browser will exchange merkle tree roots with monitors to see if they differ. However, if one log has a few more entries than another, and hasn’t been forked, they will have different Merkle roots.

You can use Merkle Trees (similarly to how we did before) to prove that one Merkle Tree is a prefix of the other, by showing how the root of the earlier one can be combined with the hashes of new elements in the second tree to output the root of the second tree.

Hence a browser can be sure it is always reading from the same fork, and if it is bad, it will eventually be caught by the gossip pool.

Certificate Transparency is not Perfect

While all of the above cryptographic proofs are useful for pointing out a bad actor, the truth of the matter is that they can’t do so instantly, and as a result it is still possible for a malicious certificate authority to get away with stealing passwords for at least some small period of time until a monitor catches it.

Conclusion

Certificate Transparency is our first introduction to dealing with decentralized distributed systems where we cannot trust the actors. We will see these types of cryptographic proofs come up over and over again, especially when we cover Bitcoin in the near future!