# Chapter 1: TLS

Stands for Transport Layer Security, this works on top of TCP/UDP Formally called SSL, HTTP inside of TLS is called HTTPS.

Security is about

- Identity - Who are you talking to?
- Privacy - Who can hear what you're saying?
- Authenticity - Is what you're hearing really coming from the person you think you're talking to?

## Identity

Without TLS, I don't know who I'm talking to, things can be intercepted by IP, TCP, and DNS attacks. Without TLS it's relatively easy to intercept connections meant for a web server and serve them yourself.

### Certificates

To check for identity TLS will use certificate, this works like an ID card. The server sends a certificate with its domain name (called a Common Name) and a big number called a public key. The certificate also includes other details, such as who made it, for example, "Bank Inc."

The certificate's signature is unique to that certificates. If you change the public key or the domain name, the signature won't match anymore.

The certificate's signature also comes from a specific entity that made that signature. It tells who signed it, and only they could have produced that signature. No one else can pretend to have signed the certificate, or the signature won't match.

It uses public-private keys to sign the certificates. You use the private key to sign the certificate and the public key to verify the certificate.

Specifically you use the CA's public key to verify the certificate.

#### Issues

- You don't have the CA's public key - There's no way to know if the bank's domain name and public key are correct
- The CA is evil - There's no way to know if the bank's domain name and public key are correct
- The CA's private key got leaked - There's no way to know if the bank's domain name and public key are correct
- The bank didn't send a signature from a CA - There's no way to know if the bank's domain name and public key are correct

Usually these are combatted by the browser makers:

- Only allow trusted CAs
- Remove bad or leaked CAs
- Not allowing you to view sites without a valid signature from a good CA
- Checking lists of known certificates to make sure someone didn't get a second certificate to pretend to be the bank (Certificate Transparency)
- Checking lists of known bad certificates and certificates that have been "revoked" (CRLs)
- Ensuring the public key is the same as the last time you connected to the site (HPKP)

## Privacy

Once the identity of the server is established, and you know the server's public key is really for the server you want to be talking to. TLS can provide privacy and authentication.

**Privacy** is hiding the content being transited back and forth from eavesdropping, snooping, man-in-the-middle. We want to prevent anyone but us and the server we want to send our information to from being able to read anything exchanged between us and the server.

## Authenticity

Separate from privacy, every message is hashed to ensure it really came from the server (client). Prevent replay attack,s injection attacks, MITM, everything encrypted is also authenticated.

## TLS Handshake

1. TCP Handshake
2. Client sends TLS "hello" as well as
   - TLS version
   - Encryption algorithms
   - ALPN: Application-Layer Protocol Negotiation e.g., select HTTP/2
   - SNI: Server Name Identification Equivalent to Host: header
3. Server sends TLS "hello"
   - Chosen encryption algorithm
   - Certificate
4. Client verifies server certificate using CA
5. Client and server agree on an encryption key either:
   - Client sends encryption key encrypted with server's public key
   - Client and server agree on encryption key by sending random bytes both directions (Diffie-Hellman Key Exchange) (more common)
6. Client sends encrypted "ready"
7. Server sends encrypted "ready"

# Perfect Forward Secrecy

Newer versions of TLS. Data remains private even if the server or client later get hacked. Uses *Ephemeral Diffie-Hellman Key Exchange*

# Reducing Latency

Traditional TLS requires at least 2 round trip latency on top of TCP handshake. Newer TLS supports 1 round trip handshake protocol.

# Cipher Suites

Client and server agree on crypto algorithm to use during handshake. Many different crypto algorithms are available.

## Example

`TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256` with 256-bit keys, TLS 1.2

- Using TLS 1.2 protocol
- Using ECDSA (Elliptic Curve Digital Signature Algorithm)
- Using ECDHE (Ephemeral Elliptic Curve Diffie-Hellman Exchange) to agree on an encryption key
- Using CHACHA20 encryption algorithm with 256-bit key
- Using Poly1305 to check authenticity of most messages
- Using SHA-256 to check authenticity of some message

# TLS Problems

TLS has had lots of different security problems over the years, typically with fun names!

- Renegotiation
- FREAK
- Logjam
- DROWN
- BEAST
- CRIME
- BREACH
- padding oracle
- Lucky Thirteen
- POODLE
- Old, insecure crypto algorithms
- Truncation
- Unholy PAC
- Sweet32
- Heartbleed
- BERserk
- Cloudbleed

# Avoiding Security Problems

TLS everything

- Make sure all traffic to/from your web app is running over TLS
- Example: it may not help if only login is over TLS because an attacker can replace your TLS-encrypted login page with an unencrypted one by replacing the link on the unencrypted main site

# Avoiding TLS security Problems

- Keep software up to date - No Windows XP
- Don't communicate with out-of-date software
- Example: if something on your site is meant to be secret, even one bad client can leak the secret!