Sketch of TLS:
1. Alice gets server’s public key: $pk_s$; related: PKIs and CAs;
2. Alice and server perform a key-exchange: $k = KeyExchange(pk_s)$; they now have a shared secret that no one else knows; e.g., $KeyExchange$ could be Diffie-Hellman, RSA, etc.;
3. $\{k_{Enc}, k_{MAC}\} = PRG(Ext(k))$; fixed function based on hashing;
4. $c = Enc_{k_e}(m||MAC_{k_m}(m)||pad)$; $MAC$: HMAC, hash-based (MD5, SHA1, SHA256); $Enc$: 3DES, AES (CBC, GCM), RC4 (no padding necessary)

Background of SSL/TLS

HTTPS (HTTP over SSL/TLS)

SSL/TLS: The crypto part; provides a secure channel in which to communicate with a web server.

SSL: 1.0 (Not released publically), 2.0, 3.0 (Developed by Netscape) which turned into: TLS: 1.0, 1.1, 1.2 (managed by IETF)

Two participants:
- **Browser**: Most recent browsers support the latest version (TLS 1.2);
- **Server**: Not many servers support latest; most servers on TLS 1.0, some back on SSL 3.0. New servers should always conform to newer standards.

Cipher Suites

- Specify which algorithms you use;
- Browsers will support many;
- Server will have prioritized list of cipher suites;
- Negotiate to use the highest priority suite from the server’s list that the client supports
Syntax

An example of a cipher suite syntax would be:
TLS_XXX_WITH_YYY

Variables:
- XXX: which key exchange you will use (Step 2);
- YYY: which encryption and MAC you will use (Step 4)

*Remember:* Step 3 in the *Sketch of TLS* is fixed.

Examples for key exchange:
- XXX = DHE_DSS (DSS = DSA \approx Schnorr)
- XXX = DHE_RSA (Signature used in DHE)
- XXX = RSA (RSA Encryption!)

*Note:* RSA can be both an encryption and signature algorithm.

*Note:* Might see ECDHE, which stands for Elliptic Curve DHE, just kind of think of it as DHE with different underlying mechanisms.

Examples for encryption and MAC:
- YYY = AES_256_CBC_SHA256
- YYY = RC4_128_SHA256

*Note:* Format will usually follow: Encryption, bit length, mode of operation, MAC signature function

Certificates

- Really just a signature on a key (how do we know if a key given to us is actually Google’s, or Facebook’s?)
- Browsers come with a list of “certificates” (“root certificates”) for companies called “certificate authorities” (CAs)
  
  CAs are trusted by the browser to correctly bind public keys to domain names

Certificate Chain

- Server \((S)\): \text{Sig}_{CA_1}(server.com, pk_s, CA: true)
  
  *CA_1* is not a root, so browser doesn’t trust it! In order to complete the chain back to the root certificate, the CA intermediate needs their own certificate:

- CA-Intermediate \((CA_1)\): \text{Sig}_{CA_0}(CA_1, pk_{CA_1}, CA: true)
- CA-Root \((CA_0)\): \text{Sig}_{sk_{CA_0}}(CA_0, pk_{CA_0}, CA: false)
**Other Certificate Properties**

- Expiry date
- Revocation information
  - Link to a server
  - Two protocols:
    - CRL
    - OCSP

**Extra**

**In Your Browser**

Specifically, for Chrome:

When visiting a website over HTTPS, you’re able to view the connection properties by clicking on the little ‘lock’ icon to the left of the URL.

In here, you can see who the identity is verified by (if it is), how it is encrypted, what protocol it uses, and other various details related to the connection. You can also view the certificate information to view the certificate chain and see it’s properties (listed earlier).

**Qualys SSL Labs**

Available at: [https://www.ssllabs.com/ssltest/](https://www.ssllabs.com/ssltest/)

You’re able to enter in a domain name and generate an SSL report based on four properties:

- Certificate;
- Protocol Support;
- Key Exchange;
- Cipher Strength

Details about protocols, cipher suites, and certificate paths will be generated.