SSL and TLS

5 June 2025
Lecture 10

Some Slides Credit: Steve Zdancewic (UPenn)

Topics for Today

- SSL/TLS
  - SSL attacks

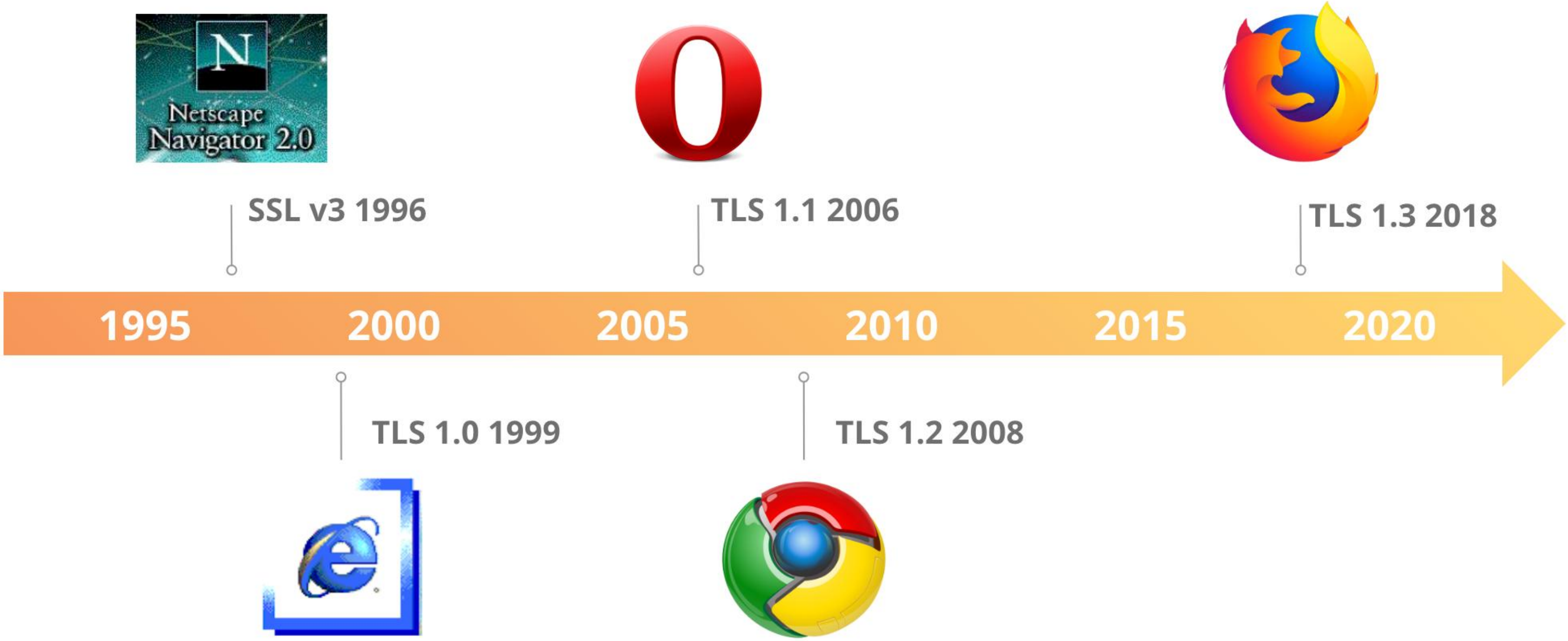
Overview: SSL

- One real world application for the techniques we have discussed so far: Secure Sockets Layer (SSL)
  - Or Transport Layer Security (TLS) Protocol
  - Versions: SSLv2.0, SSLv3.0, TLSv1.0, TLSv1.1, TLSv.1.2, TLSv1.3

- Designed by Netscape in 1996
  - Adapted by IETF to TLS
  - Now in RFC 8846 – TLS 1.3 in Aug 2018
  - Many extensions and outside applications

- Most important use is on the web (HTTP)
  - Commonly called HTTP**S**
  - SSL has **no relation** to HTTP, however
  - Security: https://www.trustworthyinternet.org/ssl-pulse/

SSL/TLS Versions



SSL v3 1996

TLS 1.0 1999

TLS 1.1 2006

TLS 1.2 2008

TLS 1.3 2018

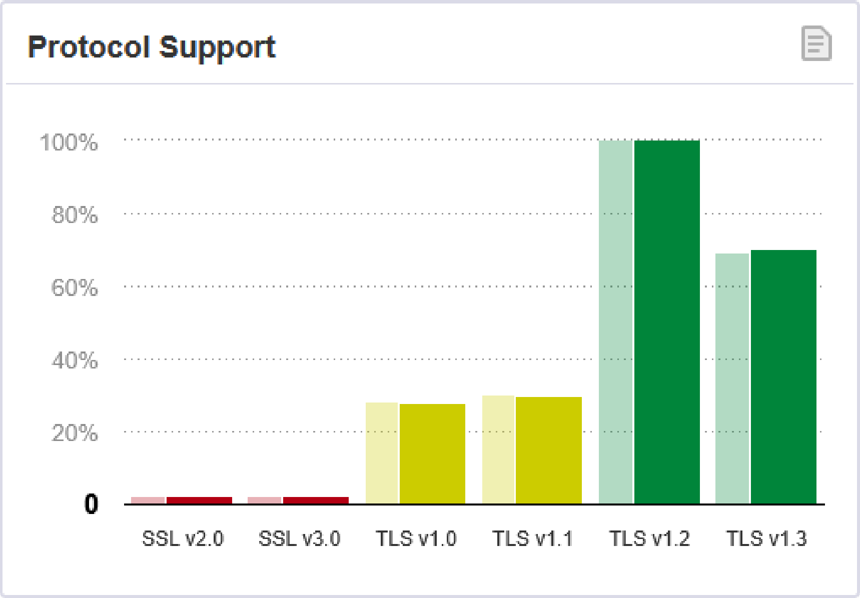
“*everything* less than TLS 1.2 with an AEAD cipher suite is cryptographically broken”

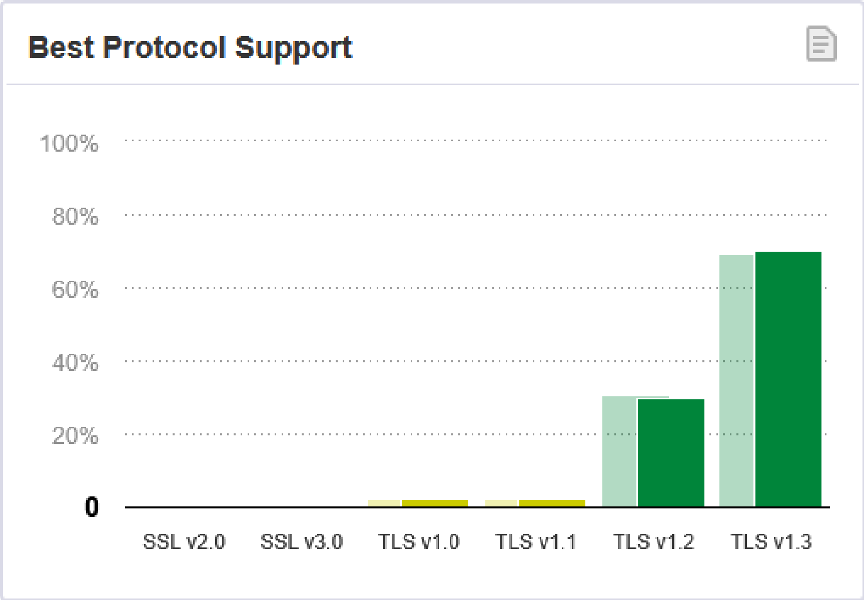
- *Adam Langely*

Senior Staff Software Engineer, Google

December 2014

State as of May 2024





https://www.ssllabs.com/ssl-pulse/

Secure Sockets Layer

Goal: Establish a secure communication channel between two computers

We've been talking about this the whole semester, so what's so hard?

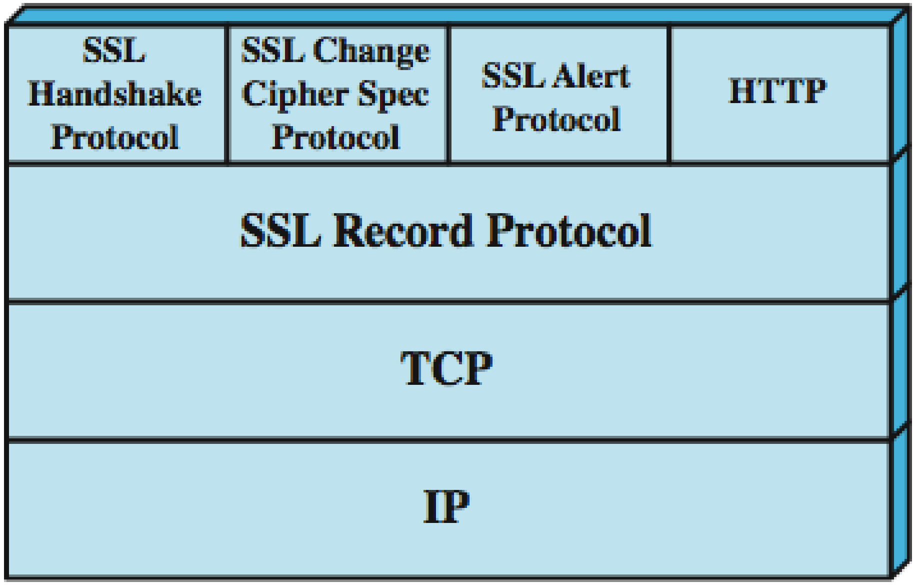
- Different operating systems (easy)
- Different cryptographic services (harder)
- Different versions (harder)
- No Trusted Third Party (?)
- One side may not have any authentication tokens (harder)

Also:

- It must be efficient
- Must be flexible
- It must be exportable
- Online negotiation (!)

Secure Sockets Layer

- Solution: Add another layer in the protocol stack on top of TCP
  - Well, two layers really
  - Several sub-protocols too



SSL Handshake Protocol

SSL Change Cipher Spec Protocol

SSL Alert Protocol

HTTP

SSL Record Protocol

TCP

IP

Sessions and Connections

Setting up a secure conversation involves online negotiation

- Expensive! 2 RTTs minimum…

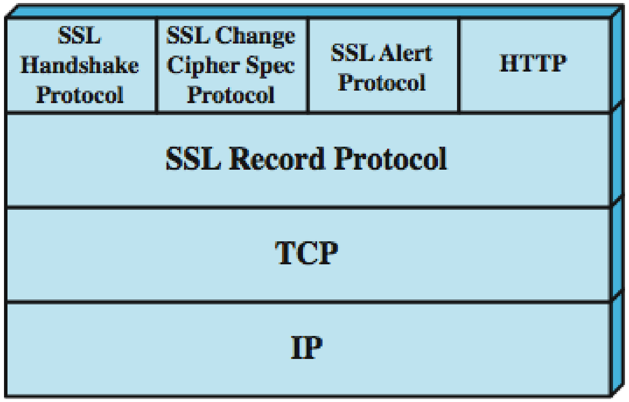
Web content is sent in a series of Requests

- Each request (connection) gets 1 item
- HTTP 1.1 changes this a bit
- That shouldn't mean we negotiate for each request!
- Solution: Long running **Sessions** and short-lived **Connections**

Do the negotiation once for the session

- Make many connections on the same session
- Technique for 0 RTT setup (session resume)

The SSL Protocols



Record Protocol

- Move data

Handshake Protocol

- Negotiate security decisions

Change Cipher Spec

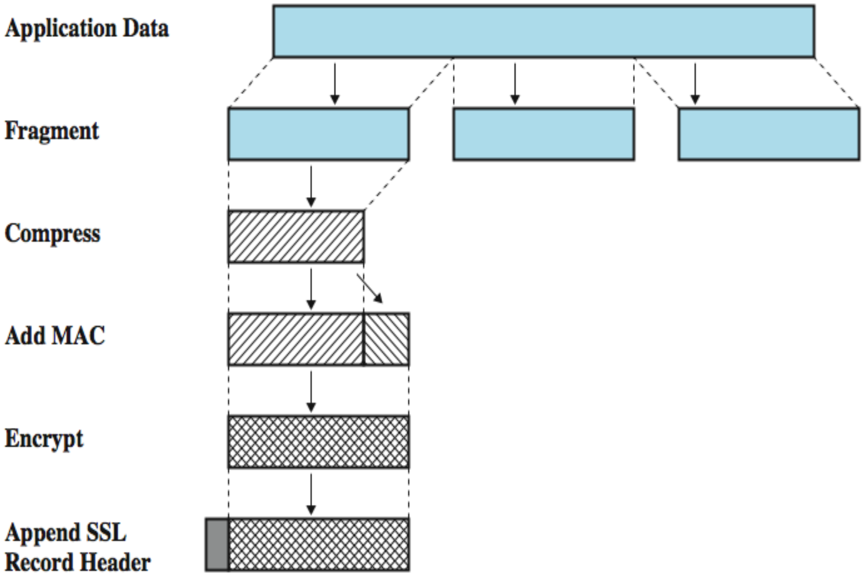
- Activate the negotiated security decisions

Alert Protocol

- Warnings and Errors

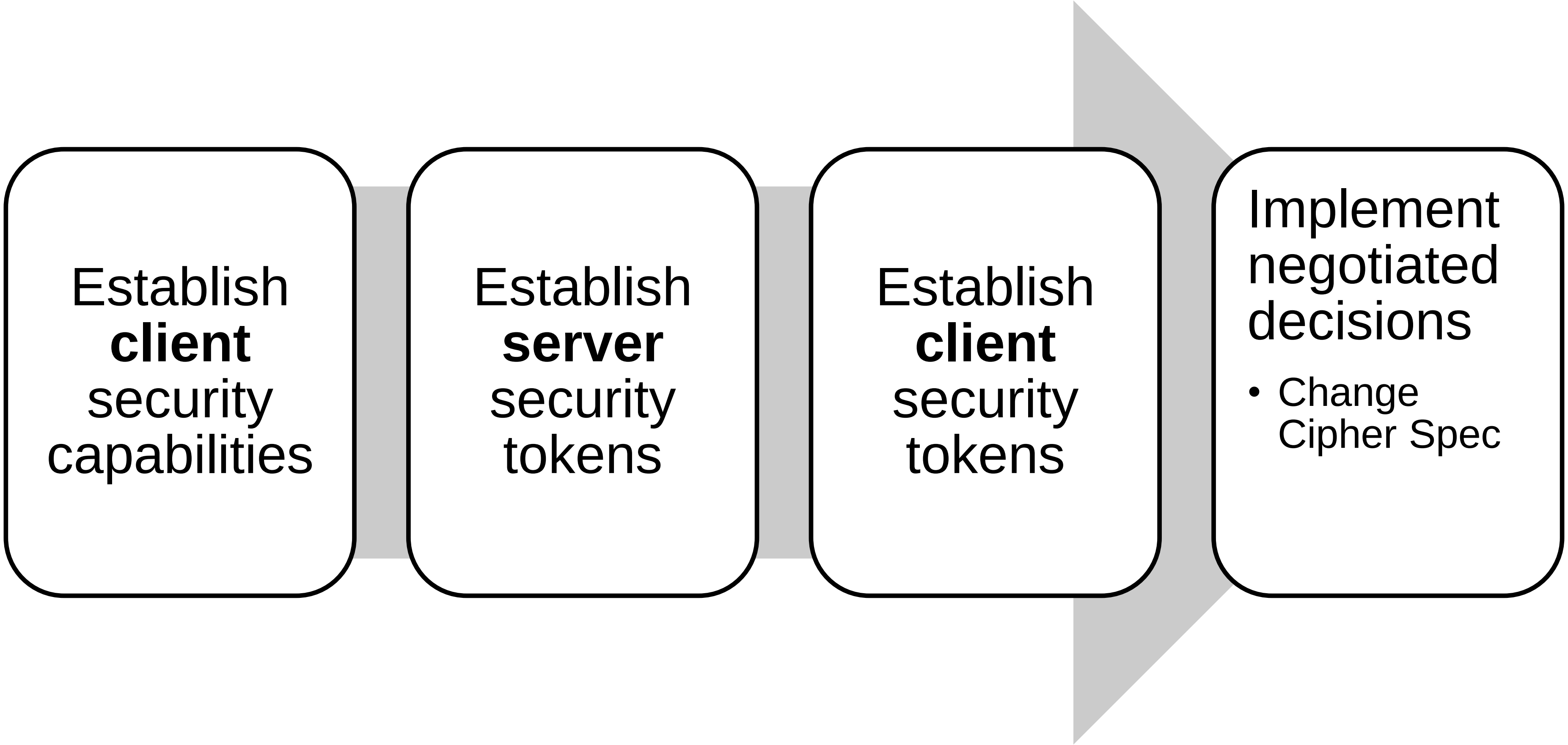
SSL Record Protocol

1. Fragment packets into 2^{14} bytes or less (16,384)
2. Compress (if you want)
3. Message Authentication Code
4. Encrypt
5. Append Header
   - Content Type (Protocol)
     - Change Cipher Spec
     - Alert
     - Handshake
     - Application_Data
   - Major Version
   - Minor Version
   - Compressed length



SSL Handshake Protocol

- Does the negotiation
- Four phases:



Establish **client** security capabilities

Establish **server** security tokens

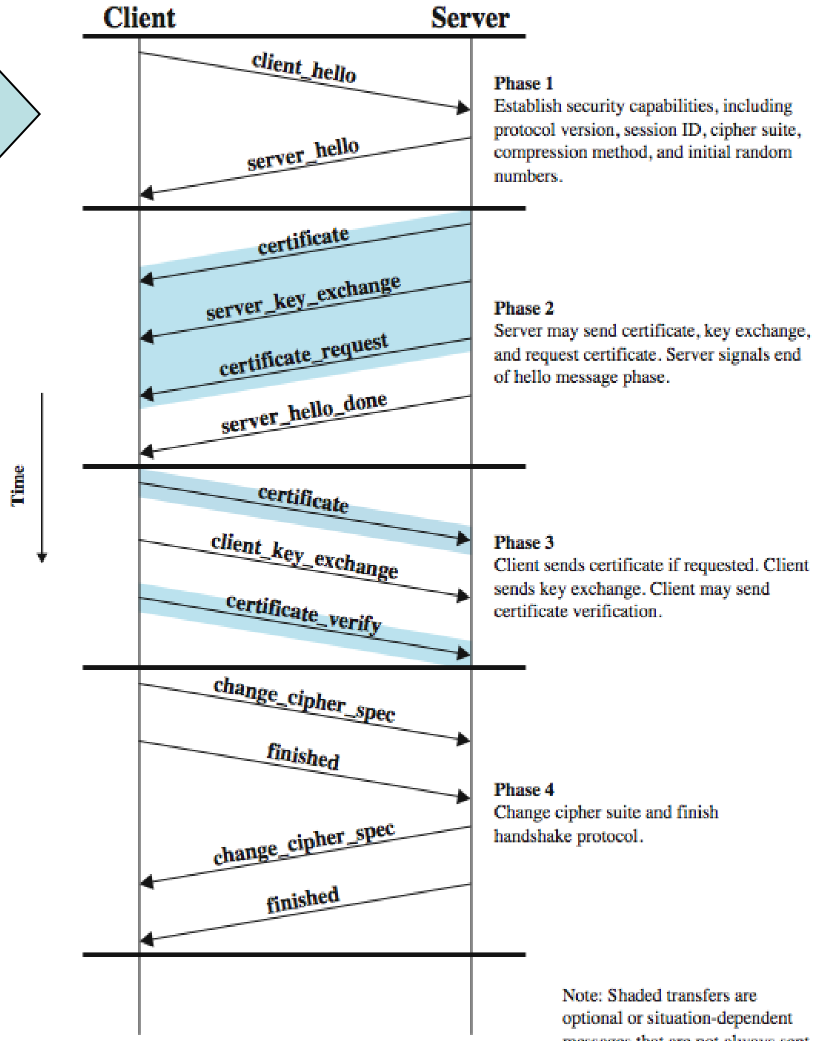
Establish **client** security tokens

Implement negotiated decisions

- Change Cipher Spec

SSL Handshake Protocol

- Phase 1: Client Starts
  - (Highest) SSL Version
  - Client Nonce: n_c
  - Session Id
    - If it’s 0 – a new session
    - If it’s not – continue a session
  - Cipher Suite
    - List of crypto algorithms supported
    - In order of preference
  - Compression Method
    - List of supported methods
- Client waits…



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

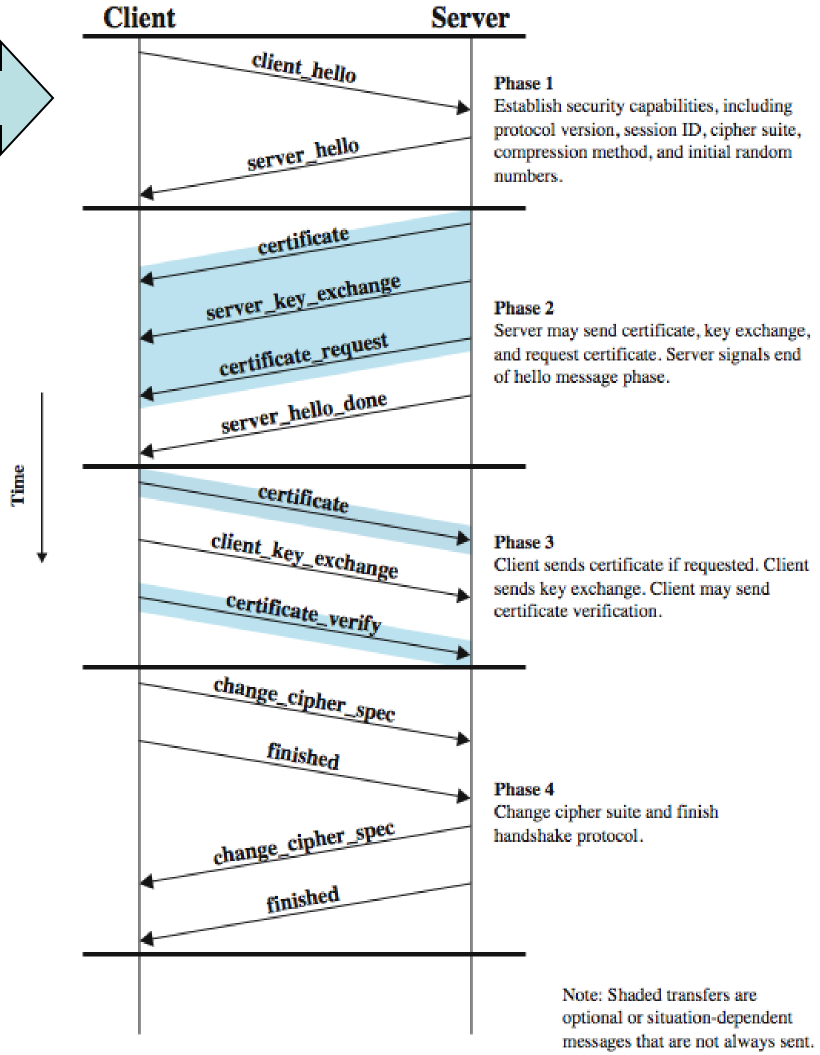
Phase 4
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

SSL Handshake Protocol

- Phase 1: Server Responds
  - *Chosen* SSL Version
  - Server nonce: n_s
  - Session Id
    - Old one if continuing
  - *Chosen* Cipher Suite
  - *Chosen* Compression Method

- Phase 2: Server tokens
  - Server Certificate
  - (Optional) Request Client Certificate
  - $Server\_Hello\_Done$



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

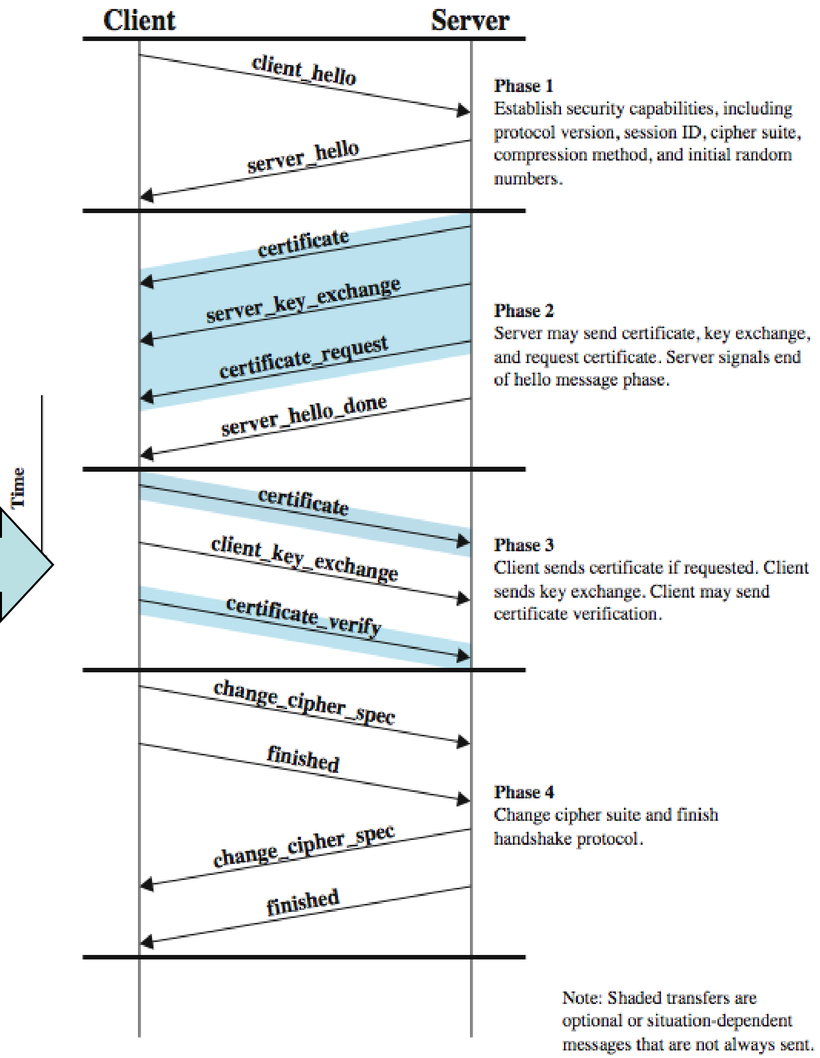
Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

Phase 4
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

SSL Handshake Protocol

- Phase 3: Client tokens
  - Client verifies certificate
  - Client sends security tokens
- Certificate (Optional)
  - Signs previous messages with Certificate private key (Client Verify)
- If no certificate: Pre-master secret (48 bits)
  - Encrypted with Server Key



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

Phase 4
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

Pre-master Secret

- Using Pre-master Secret (PMS)
  - 48 bit random number
  - Combined with n_c and n_s to make a full secret

- Old Algorithm: $master\_secret\ =$
  - $MD5(PMS + SHA('A'\ +\ PMS\ + n_c\ +\ n_s))\ +$
  - $MD5(PMS + SHA('BB'\ +\ PMS\ + n_c\ +\ n_s))\ +$
  - $MD5(PMS + SHA('CCC'\ +\ PMS\ + n_c\ +\ n_s));$

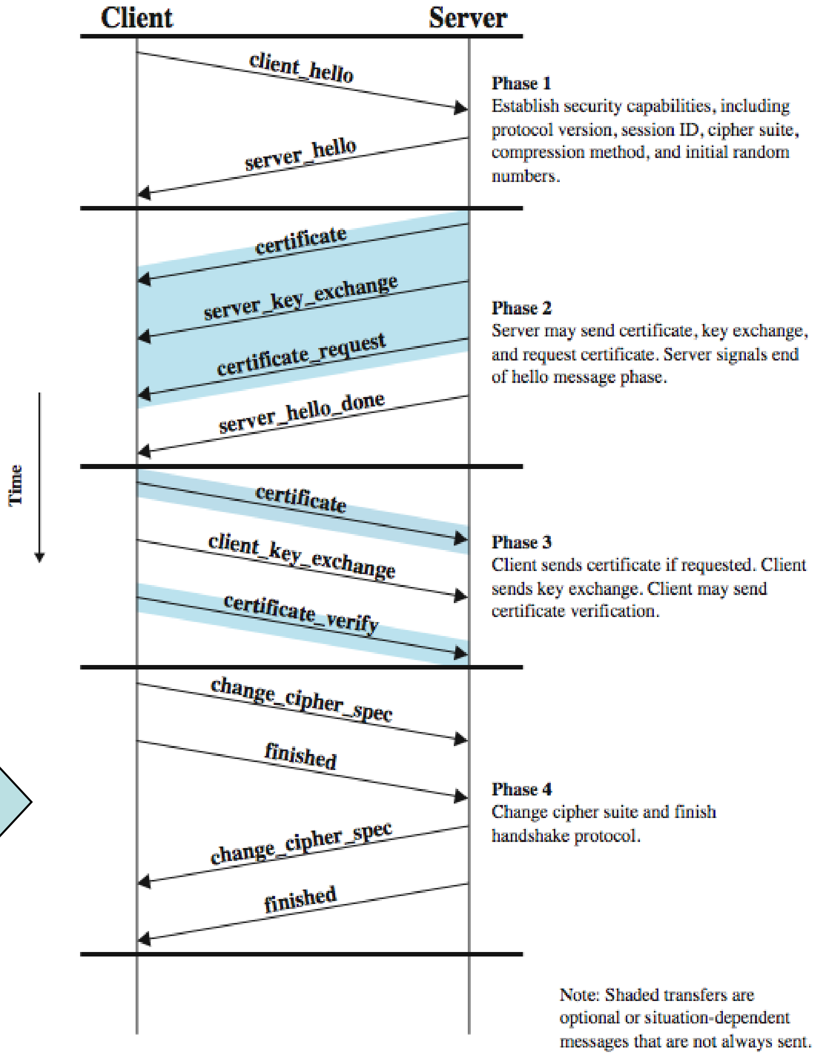
- New Algorithm: master secret is defined per cipher suite
  - Varying length supported by iterated (and concatenated) hashes
  - Based on $SHA256$

- Master secret processed using Key Derivation Function (KDF) which produces encryption and MAC keys
  - See NIST 800-108 for details (see *counter mode* section)

SSL Handshake Protocol

- Phase 4: Implement
  - Client sends: Change Cipher Spec
  - Server sends: Change Cipher Spec

- Both indicate they are ready to use what has been negotiated
  - Both send a keyed hash digest of all messages sent in the handshake process



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

Phase 4
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

SSL Change Cipher Spec

- Simple protocol: 1 message with 1 byte of data
  - Byte set to 1
- Tells the other side to implement the agreed upon cipher suite

SSL Alert Protocol

- Two bytes of data
- Byte 1: Severity of alert
  - $= 1$: Warning
  - $= 2$: Fatal (terminates connection)

- Byte 2: Alert Codes
  - Examples:
    - Close notify
    - Decompression failure
    - Bad certificate
    - Certificate revoked
    - Illegal parameter
    - Decode error
    - Insufficient security

Reflection: SSL

Enables secure communication over the internet

Works even if only one side has a certificate

- Client authentication must be done some other way

Main application for certificates and PKI

- Has helped sell many certificates
- Market of $187M in 2023

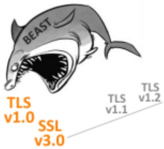
Secures the communication channel

- But not the data stored on the other side
- A thief can still steal your credit card information from the server
- Has made it harder for governments to spy on web traffic

SSL Attacks: Protocol Level

BEAST (2011):

- Browser Exploit Against SSL/TL
- Breaks encryption using CBC based on padding.



CRIME (2012):

- Compression Ratio Info-leak Made Easy
- Insert or steal data from a secured SSL connection. Works on TLS compression.

BREACH (2013):

- Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext
- Improved CRIME, works on HTTP compression

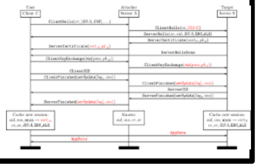
POODLE (2014):

- Padding Oracle On Downgraded Legacy Encryption
- Padding oracle attack for CBC mode in SSLv3.0 (improved BEAST)

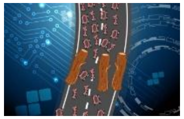
SSL Attacks: Protocol Level

Triple-Handshake attack (2014)

- A malicious server can impersonate a client that uses a client certificate



Logjam (2015):



- Can precompute Diffie-Hellman prime/primitive root combinations to break DH key establishment

ROBOT Attack (2018):

- **Return Of Bleichenbacher's Oracle Threat**
- Padding vulnerability that leads to private key compromised

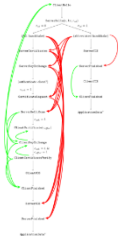
SSL Attacks: Implementation

Heartbleed (2014):

- OpenSSL bug, forgot bounds checking on message

Skip-TLS (2015):

- Can force Java implementations of SSL to skip encryption steps



FREAK (2015):

- Factoring RSA Export Keys
- Force browser or server to use a weak (Export approved) encryption key

TLS 1.3 Changes

Forward Secrecy

- Removed RSA key agreement
- Now only ECDHE

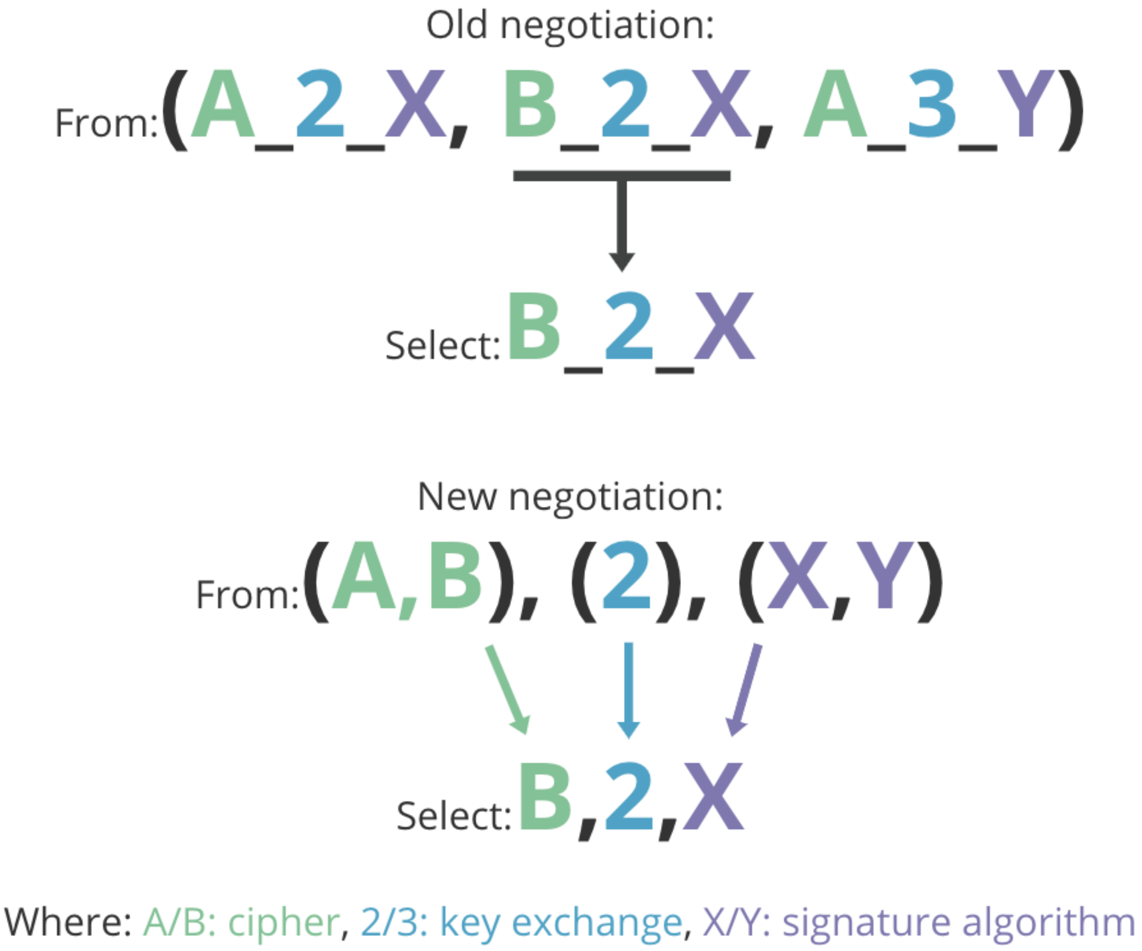
Message integrity

- More of the handshake is encrypted
- From server hello and on
- Everything in the handshake is signed at the end

Improved negotiation

- Removed complex cipher suite names
  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- Now just negotiate three elements:
  - Cipher + Hash
  - Key Exchange
  - Signature Algorithm
  - TLS_AES_256_GCM_SHA384
- No more change cipher spec

Negotiation Changes



Old negotiation:

From: (A_2_X, B_2_X, A_3_Y)

Select: B_2_X

New negotiation:

From: (A,B), (2), (X,Y)

Select: B,2,X

Where: A/B: cipher, 2/3: key exchange, X/Y: signature algorithm

https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/

1-RTT Mode – New Sessions

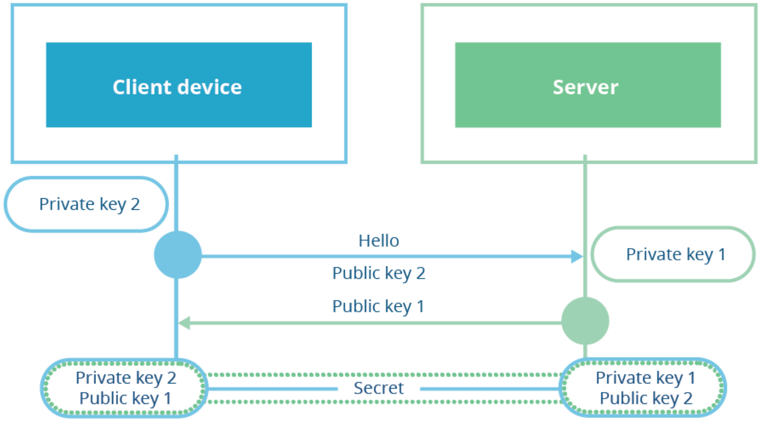
Reduce time for new sessions

- 2 common ECDHE curves
- Client sends key shares in first message
- Guesses server supports them

If server supports any of the suites, responds with key share and approval

- If guessed wrong, needs to try again
- Server sends other options

DH 1.3 handshake



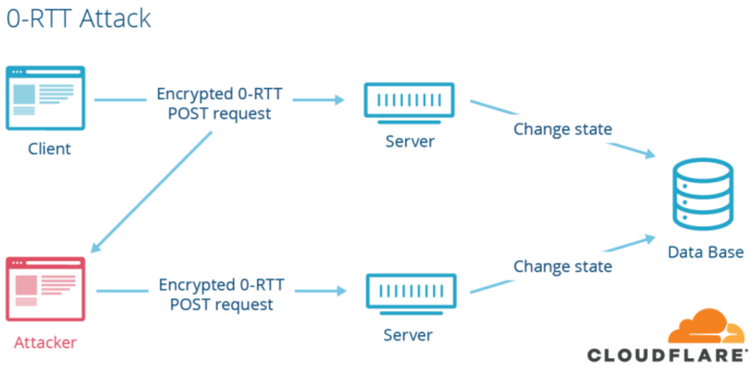
CLOUDFLARE

https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/

0-RTT Mode - Resumption

Resume faster

- When resuming session, use info from previous session
- Once a conversation is setup, client and server can set up Resumption Main Secret to use later in a "session ticket"



Opens replay attack problems

- Data sent is already encrypted in first message
- Attacker can replay 0-RTT messages and server can't tell
- Don't state changing actions based on them

https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/

Conclusion

- SSL/TLS