Web Security

PART II: TLS/SSL

Dr. Bheemarjuna Reddy Tamma

IIT HYDERABAD

Note: This is a revised version of slide deck of Prof. Dan Boneh (Stanford) with material from IETF, Cloudflare and various other Internet sources

Outline

- How SSL/TLS protocols work
- Various attacks on SSL/TLS variants
- TLS 1.3

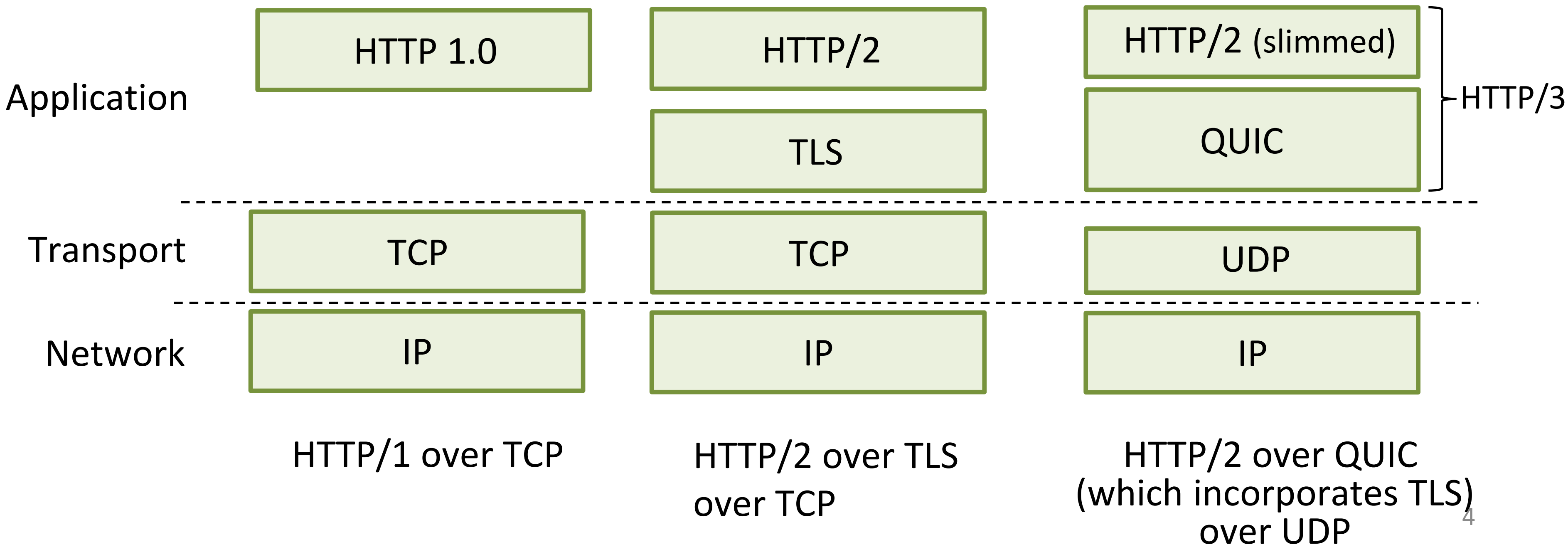
Transport Layer Security (TLS)

- The widely deployed security protocol above the transport layer
  - Supported by almost all browsers, web servers: https (port 443)
  - Primarily used with TCP (reliability and in-sequence delivery)
  - Datagram TLS (DTLS) variant for use with UDP/SCTP/SRTP/CAPWAP
- Provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing*
  - authentication: via *public key cryptography*

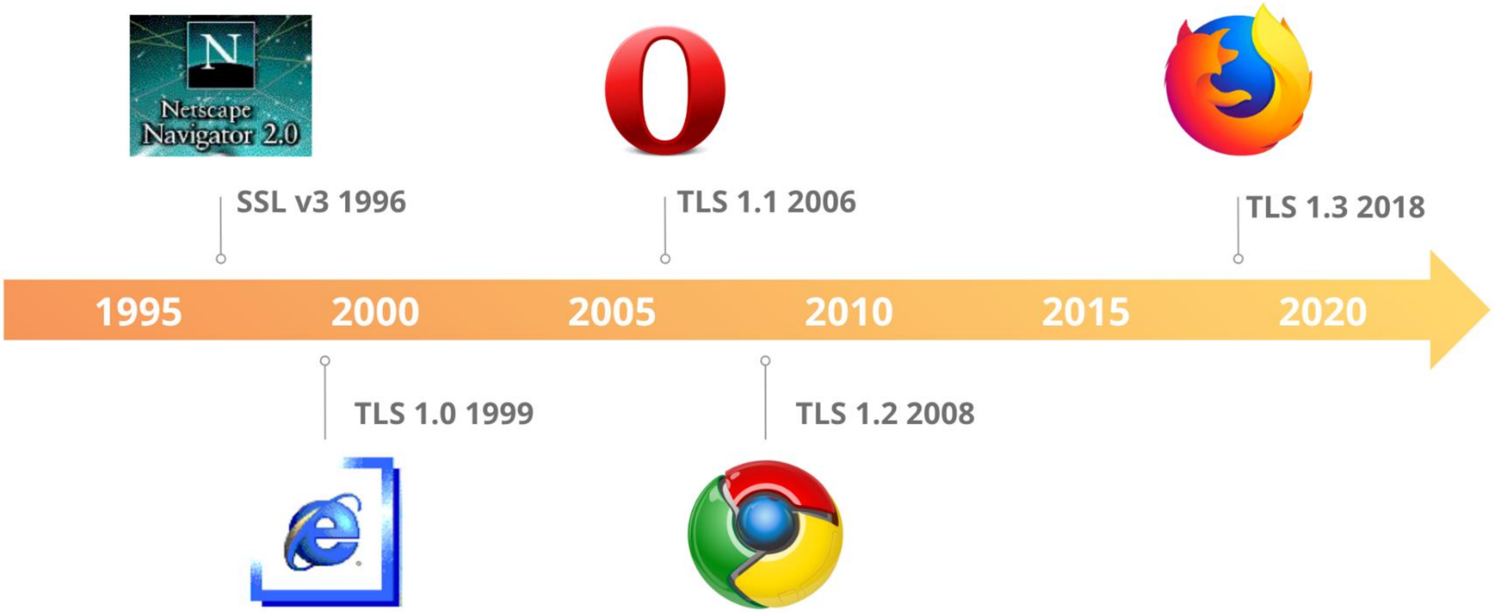
all techniques we have studied!

Transport Layer Security (TLS)

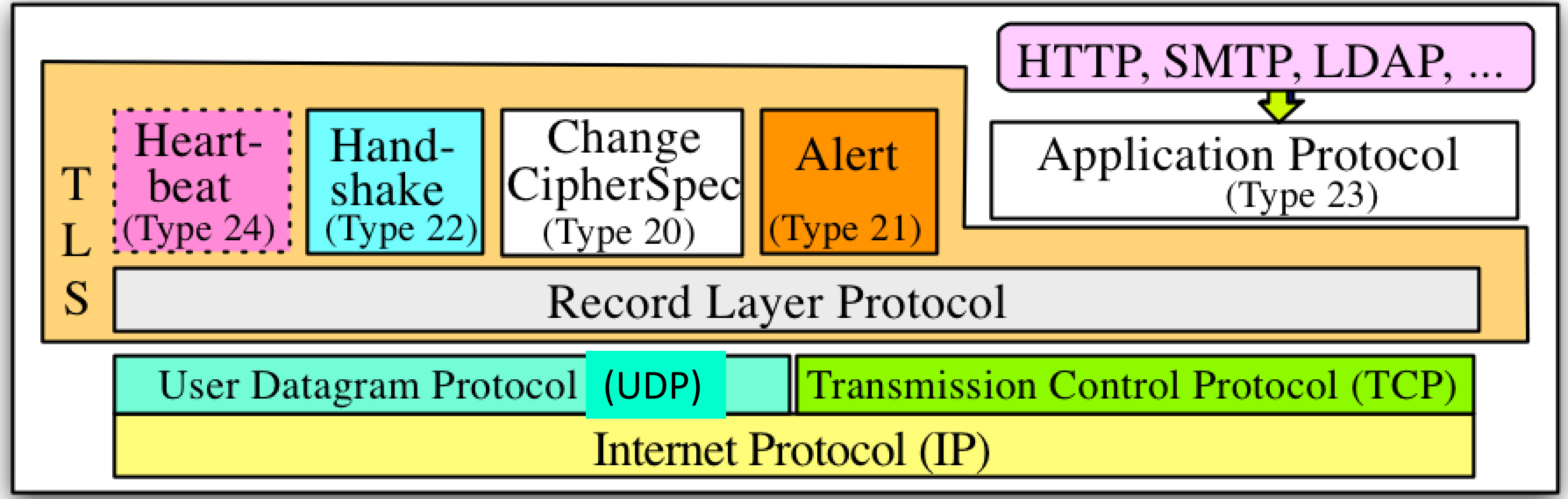
- TLS provides an API that *any* application can use
- HTTP view of TLS:



SSL/TLS Variants



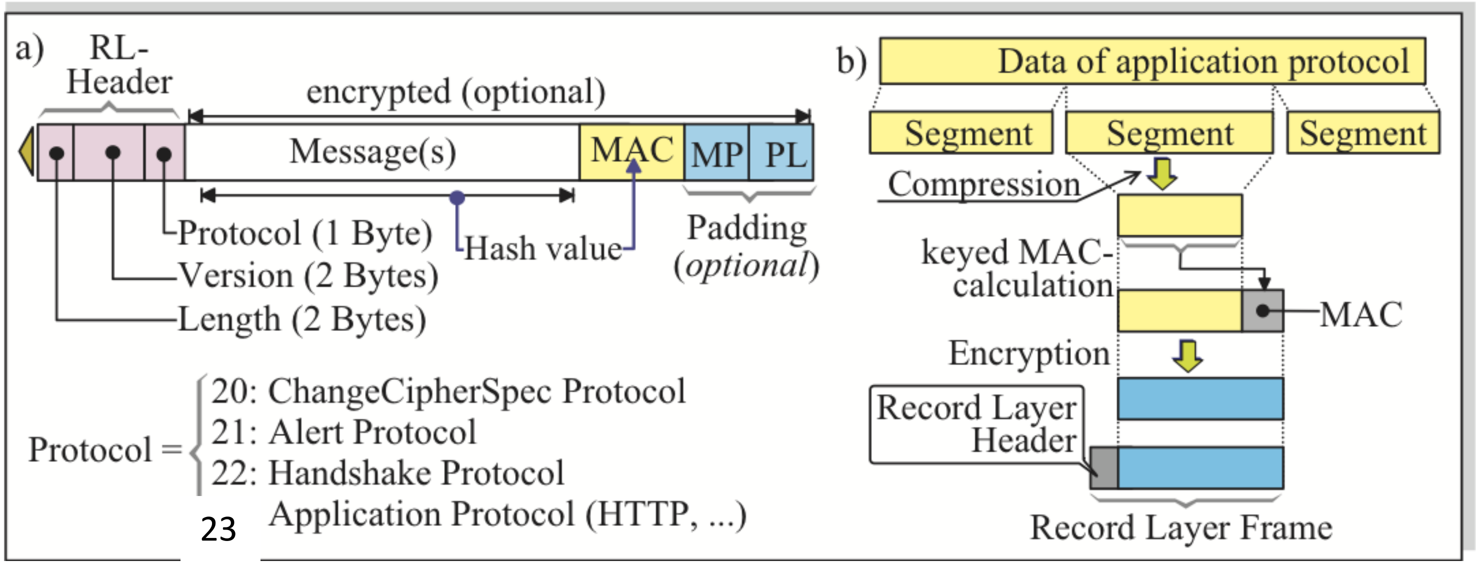
Layered Architecture of TLS



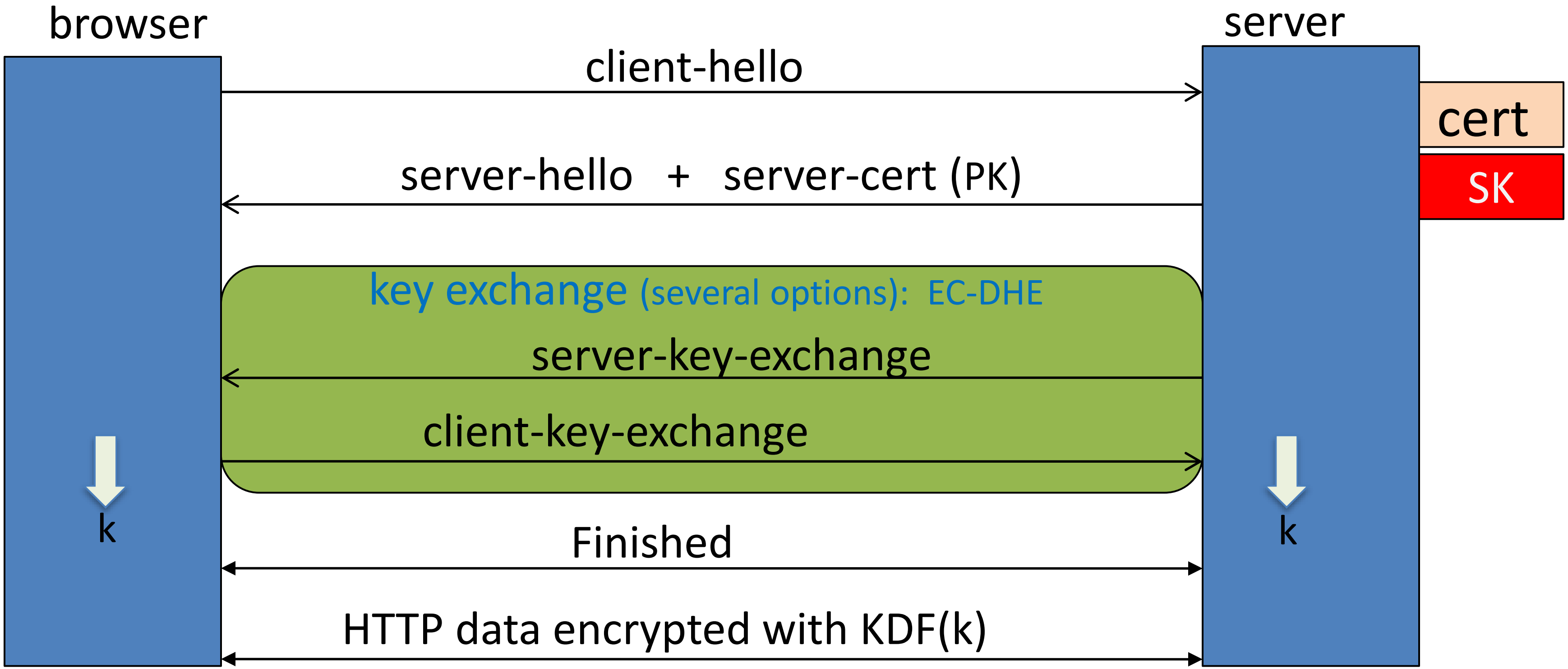
https://www.fehcom.de/qmail/smtptls.html

TLS: Record Layer

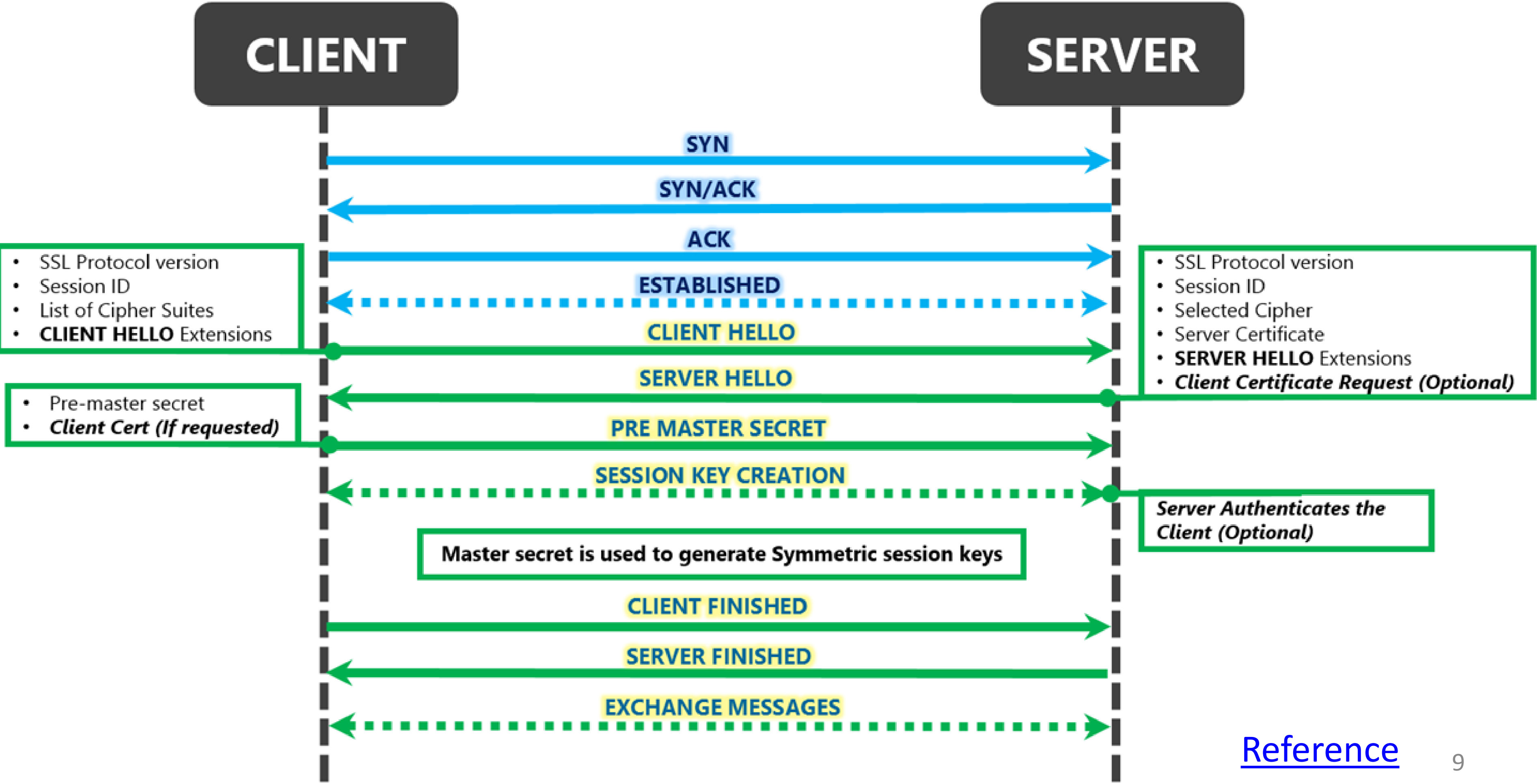
- RL is the workhorse of TLS
  - *fragment* the application data into segments
  - Compression of segments
  - Integrity by adding MAC, padding (if needed), and then Encryption
  - Finally, adding required RL Header



Brief summary of SSL/TLS



Most common: server authentication only


CLIENT
SERVER
SYN
SYN/ACK
ACK
ESTABLISHED
CLIENT HELLO
SERVER HELLO
PRE MASTER SECRET
SESSION KEY CREATION
CLIENT FINISHED
SERVER FINISHED
EXCHANGE MESSAGES
• SSL Protocol version
• Session ID
• List of Cipher Suites
• CLIENT HELLO Extensions
• SSL Protocol version
• Session ID
• Selected Cipher
• Server Certificate
• SERVER HELLO Extensions
• Client Certificate Request (Optional)
• Pre-master secret
• Client Cert (If requested)
Server Authenticates the Client (Optional)
Master secret is used to generate Symmetric session keys

[Reference]

Four Phases of TLS Handshake Protocol

- ❖ **Phase-1**

  Both ends agree upon Cipher Suite

  - TLS_**RSA**_WITH_**AES**_256_CBC_**SHA**256
  - TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
  - AEAD_**AES**_256_GCM_**SHA**384 (TLS 1.3)

- ❖ **Phase-2**

  Server sends its digital Cert signed by a CA

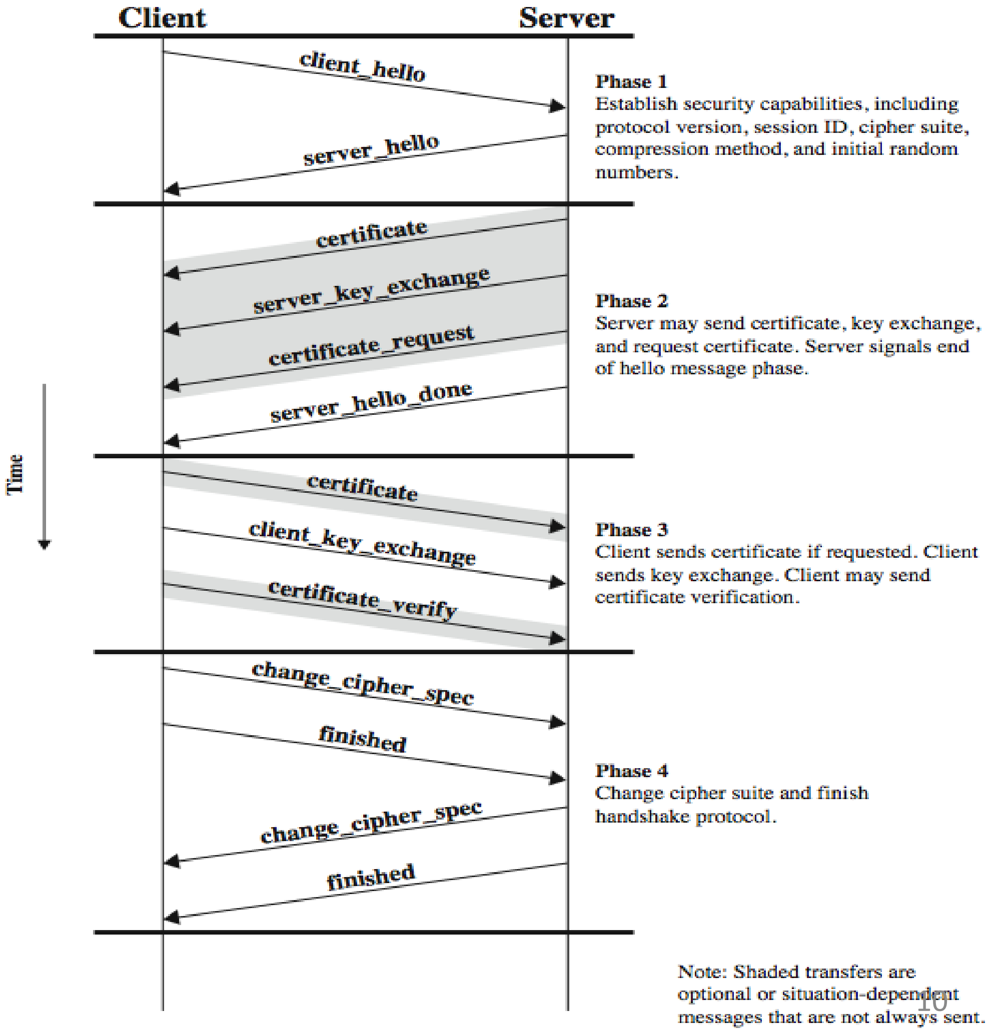
- ❖ **Phase-3**

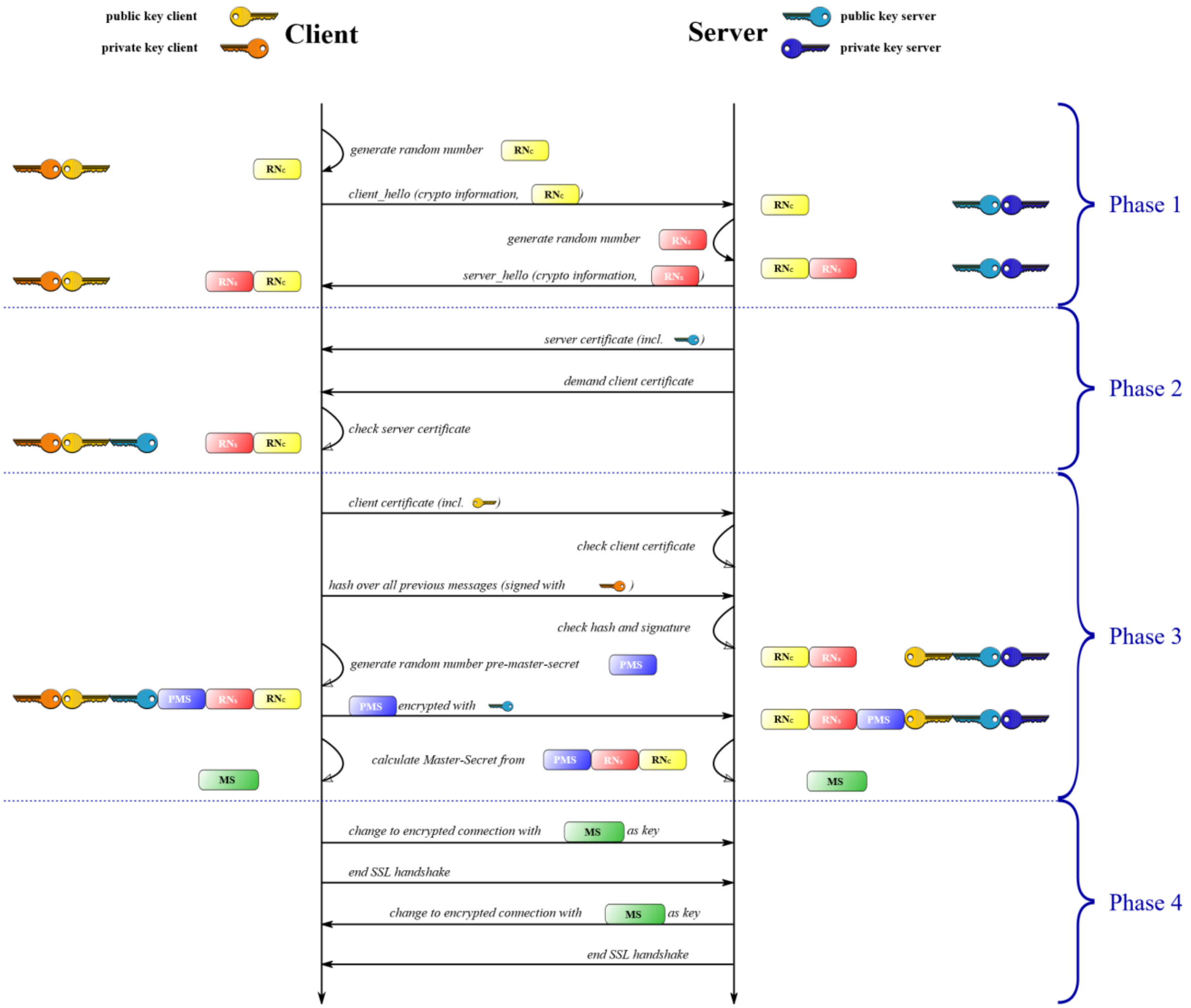
  Client sends a secret master key (PMS) encrypted with Server’s public key

  Client may also send a signed hash of all of its previous messages in Cert_Verify msg

- ❖ **Phase-4**

  Handshake is completed and a secure connection is established

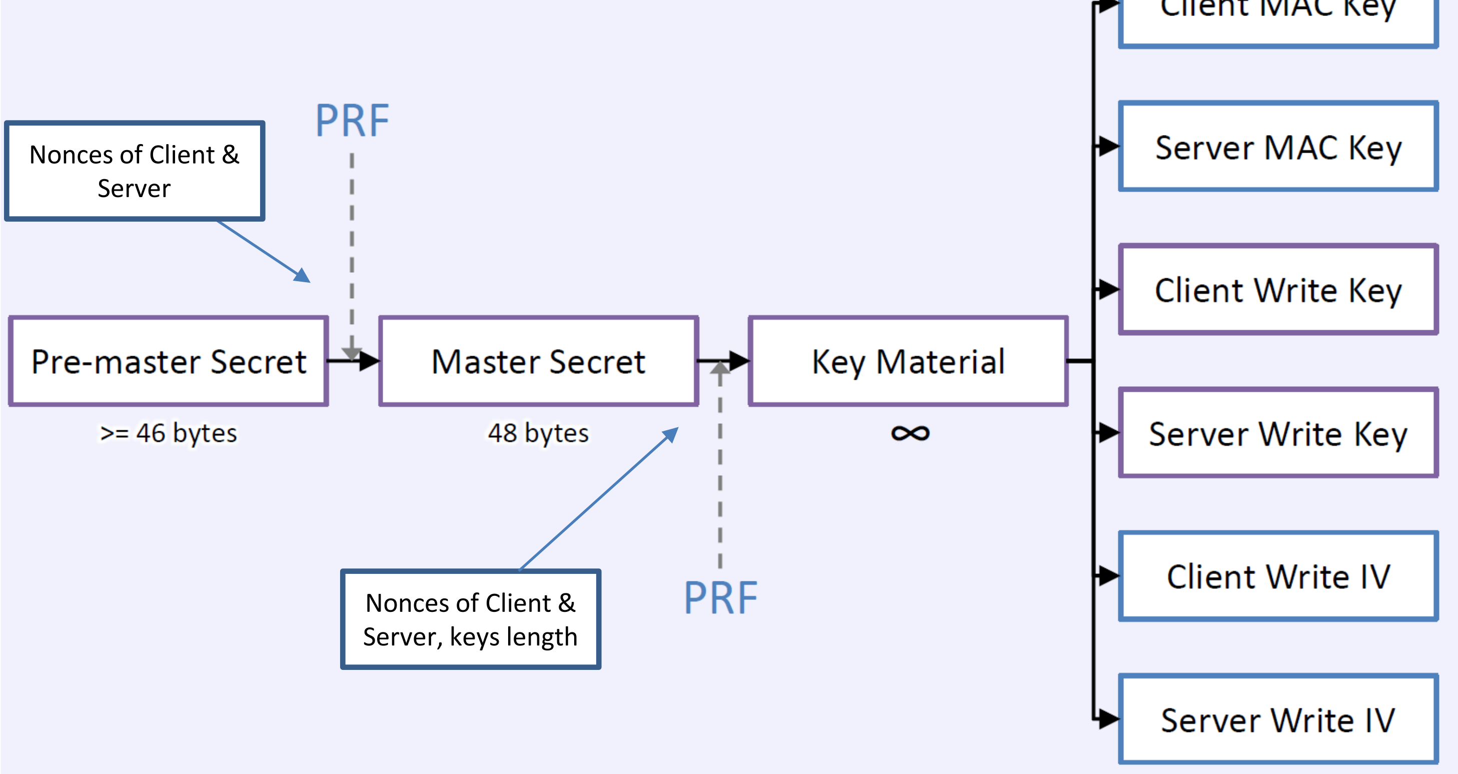




public key client
private key client
Client
Server
public key server
private key server
generate random number
RNc
client_hello (crypto information, RNc)
RNs
generate random number
server_hello (crypto information, RNs)
server certificate (incl.)
demand client certificate
check server certificate
client certificate (incl.)
check client certificate
hash over all previous messages (signed with)
check hash and signature
generate random number pre-master-secret
PMS
encrypted with
calculate Master-Secret from
MS
change to encrypted connection with MS as key
end SSL handshake
change to encrypted connection with MS as key
end SSL handshake
Phase 1
Phase 2
Phase 3
Phase 4

Key Generation in TLS 1.2

https://www.ietf.org/rfc/rfc5246.txt



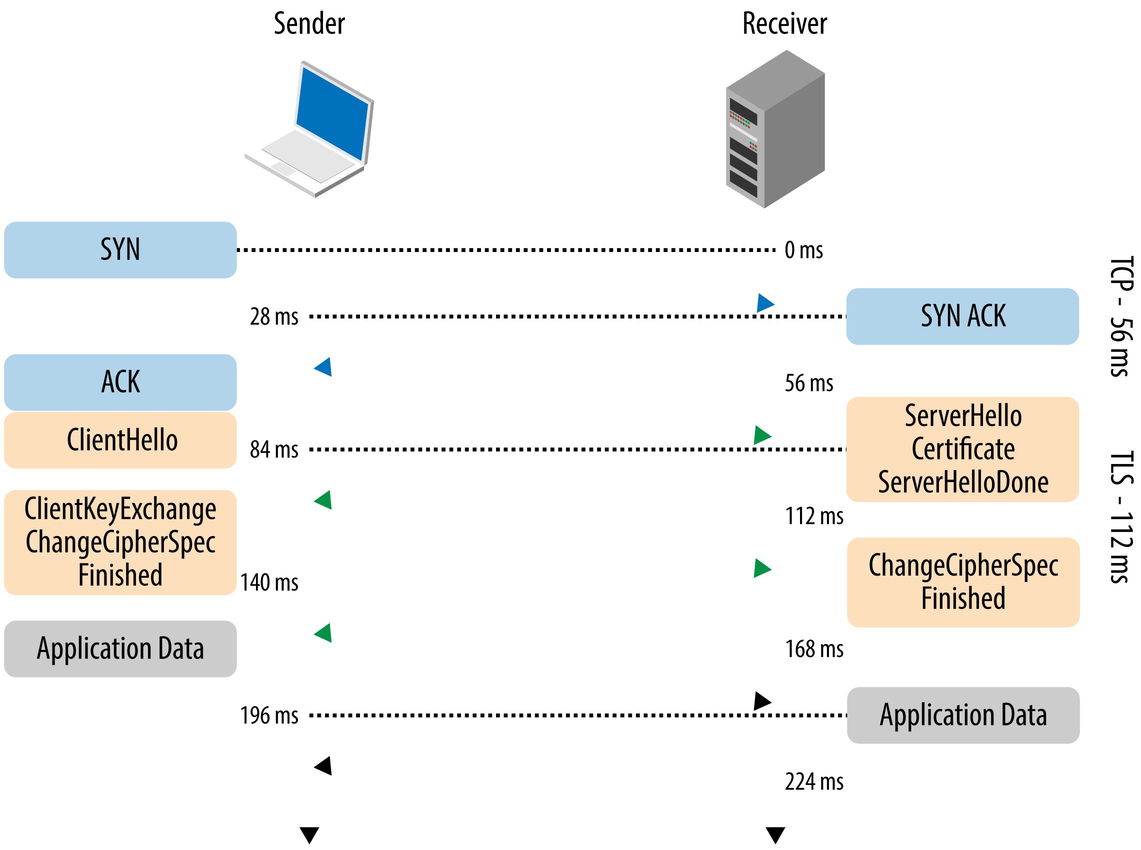
Query

- Could Trudy launch any MITM attacks and replay attacks by snooping into the HTTPS-TLS-TCP session established between browser (Alice) and web server (Bob)?

TLS: Guarding against simple attacks

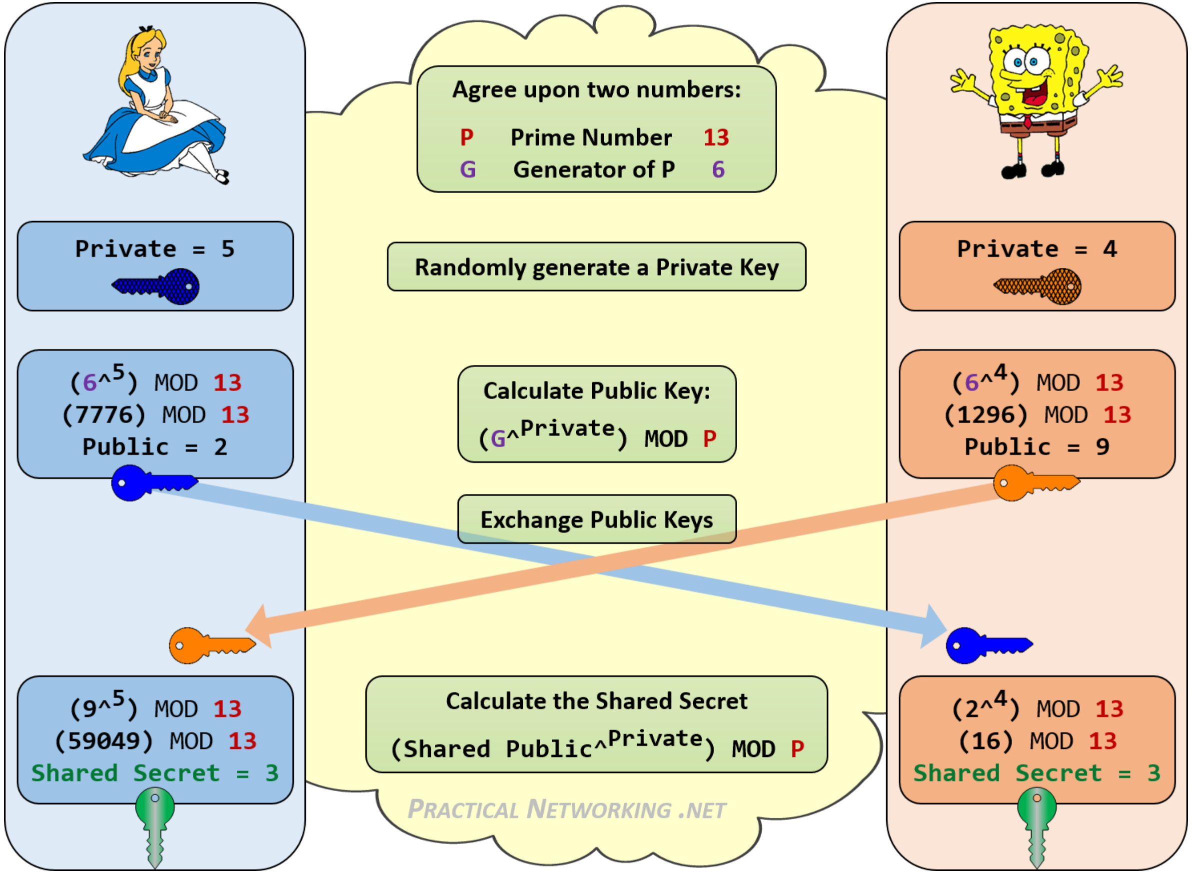
- Role of random numbers (nonces) in TLS handshake
  - Protect against connection/session replay attacks
- Role of sequence numbers in TLS session
  - Different from TCP Sequence Numbers, not added explicitly into Record Protocol Header
  - Protect against segment replay attacks
  - Protect against segment reordering or deletion by modifying TCP Sequence Numbers

Full TLS 1.2 handshake with timing information



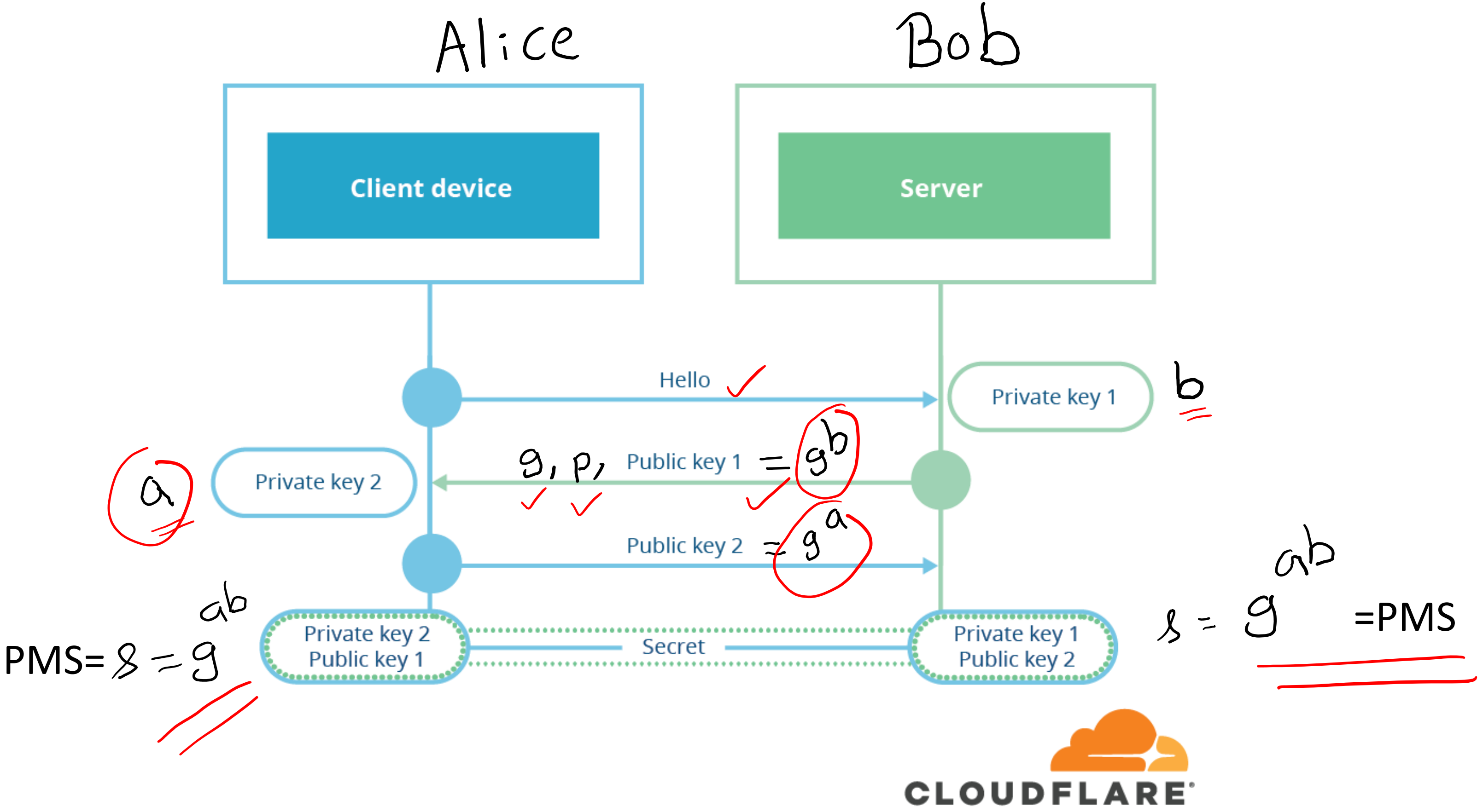
Reference

Diffie-Hellman Key Exchange



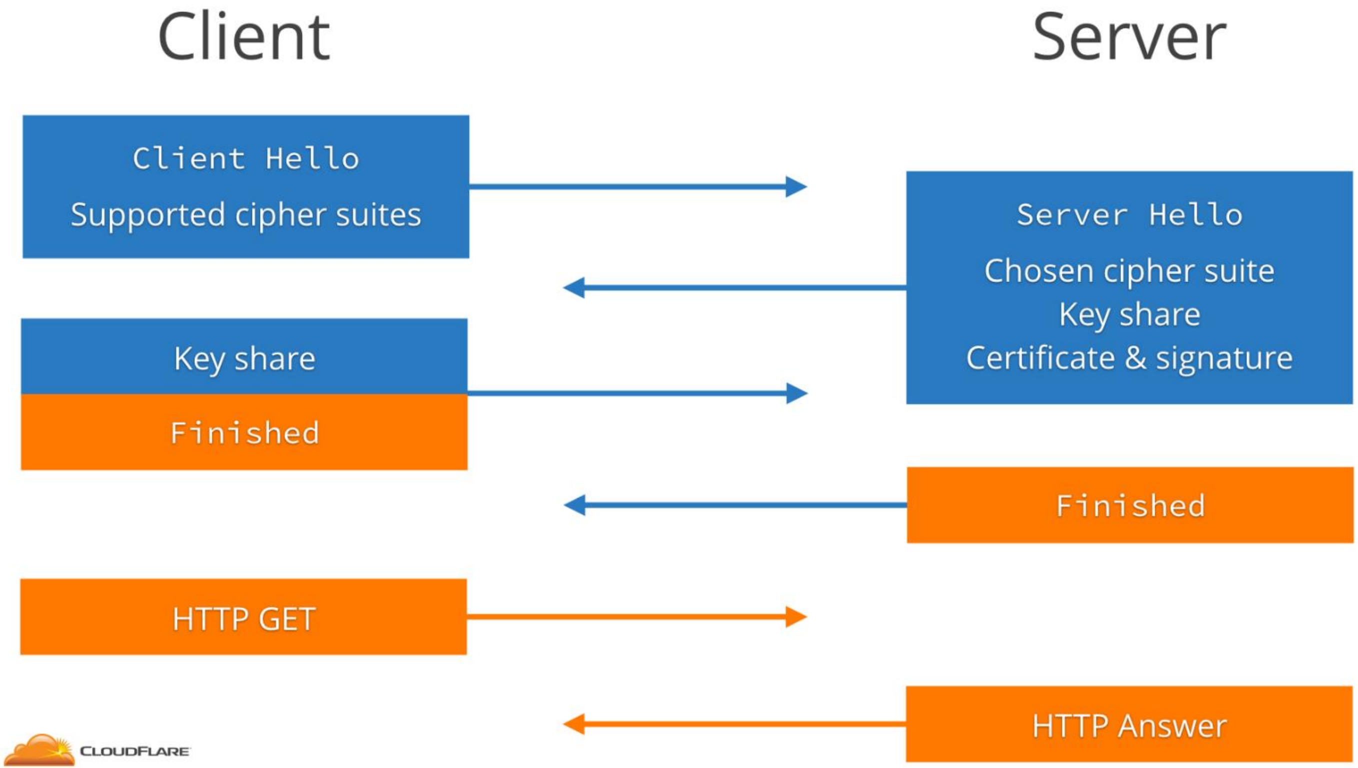
Note: Only a key exchange/agreement algo; Can't be useful for Authentication!

DH 1.2 handshake



Note: No exchange of PMS unlike when RSA is used for key exchange

TLS 1.2 (ECDHE)



[Diffie–Hellman key exchange - Wikipedia]

Diffie-Hellman in SSL/TLS

- Fixed or Static Diffie-Hellman
  - Server's public DH paras like g, p and public key (g^b) are kept in Digital Cert and signed by CA
  - CipherSuite: TLS_**DH_RSA**_WITH_AES_128_CBC_SHA256
  - No Perfect Forward Secrecy (PFS)
- Ephemeral Diffie-Hellman
  - Server and client generate fresh DH keypairs for each session
  - Public DH parameters for ephemeral keypairs are signed by the private key (RSA/DSS) of Server
  - CipherSuite: TLS_**DHE_RSA**_WITH_AES_256_CBC_SHA256
  - Offers PFS
- Anonymous Diffie-Hellman
  - No authentication, possible MITM attacks
  - CipherSuite: TLS_**DH_anon**_WITH_AES_256_CBC_SHA256

Comparison of Cipher Suites

- TLS_**RSA**_WITH_AES_256_CBC_SHA256
  - Static RSA keys for authentication and session key exchange
  - PMS is encrypted with Server’s Public RSA key
  - No PFS (Perfect Forward Secrecy)
  - No Server Key Exchange Msg in TLS handshake

- TLS_**DHE_RSA**_WITH_AES_256_CBC_SHA256
  - Static RSA keys for authentication
  - DH with ephemeral key pairs for session key exchange
    - Server Key Exchange Msg in TLS handshake carries public DH parameters
    - PMS (g^{ab}) is never exchanged, but locally derived by both!
  - Offers PFS

Note: Static RSA keys for authentication and ephemeral RSA keys for key exchange offer PFS but never used as DHE/ECDHE are more efficient

TLS 1.2 Cipher Suites (RFC 5246)

Cipher Suite	Key Exchange	Cipher	Mac
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA
TLS_RSA_WITH_NULL_SHA256	RSA	NULL	SHA256
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	AES_128_CBC	SHA
TLS_RSA_WITH_AES_256_CBC_SHA	RSA	AES_256_CBC	SHA
TLS_RSA_WITH_AES_128_CBC_SHA256	RSA	AES_128_CBC	SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256	RSA	AES_256_CBC	SHA256
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE_CBC	SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE_CBC	SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE_CBC	SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES_EDE_CBC	SHA
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DH_DSS	AES_128_CBC	SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA	DH_RSA	AES_128_CBC	SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DHE_DSS	AES_128_CBC	SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DHE_RSA	AES_128_CBC	SHA
TLS_DH_anon_WITH_AES_128_CBC_SHA	DH_anon	AES_128_CBC	SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA	DH_DSS	AES_256_CBC	SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA	DH_RSA	AES_256_CBC	SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DHE_DSS	AES_256_CBC	SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DHE_RSA	AES_256_CBC	SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA	DH_anon	AES_256_CBC	SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA256	DH_DSS	AES_128_CBC	SHA256
TLS_DH_RSA_WITH_AES_128_CBC_SHA256	DH_RSA	AES_128_CBC	SHA256
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	DHE_DSS	AES_128_CBC	SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	DHE_RSA	AES_128_CBC	SHA256
TLS_DH_anon_WITH_AES_128_CBC_SHA256	DH_anon	AES_128_CBC	SHA256
TLS_DH_DSS_WITH_AES_256_CBC_SHA256	DH_DSS	AES_256_CBC	SHA256
TLS_DH_RSA_WITH_AES_256_CBC_SHA256	DH_RSA	AES_256_CBC	SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	DHE_DSS	AES_256_CBC	SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	DHE_RSA	AES_256_CBC	SHA256
TLS_DH_anon_WITH_AES_256_CBC_SHA256	DH_anon	AES_256_CBC	SHA256

Cipher	Type	Key Material	IV Size	Block Size
NULL	Stream	0	0	N/A
RC4_128	Stream	16	0	N/A
3DES_EDE_CBC	Block	24	8	8
AES_128_CBC	Block	16	16	16
AES_256_CBC	Block	32	16	16

MAC	Algorithm	mac_length	mac_key_length
NULL	N/A	0	0
MD5	HMAC-MD5	16	16
SHA	HMAC-SHA1	20	20
SHA256	HMAC-SHA256	32	32

Supported Cipher Suites in Openssl

$ **openssl ciphers -v**

TLS_AES_256_GCM_SHA384 TLSv1.3 Kx=any   Au=any Enc=AESGCM(256) Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any   Au=any Enc=CHACHA20/POLY1305(256) Mac=AEAD
TLS_AES_128_GCM_SHA256 TLSv1.3 Kx=any   Au=any Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH   Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH   Au=RSA Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH   Au=RSA Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH   Au=ECDSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH   Au=RSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
DHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=DH   Au=RSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH   Au=ECDSA Enc=AESGCM(128) Mac=AEAD
ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH   Au=RSA Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=DH   Au=RSA Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH   Au=ECDSA Enc=AES(256) Mac=SHA384
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH   Au=RSA Enc=AES(256) Mac=SHA384
DHE-RSA-AES256-SHA256   TLSv1.2 Kx=DH   Au=RSA Enc=AES(256) Mac=SHA256
ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH   Au=ECDSA Enc=AES(128) Mac=SHA256
ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH   Au=RSA Enc=AES(128) Mac=SHA256
DHE-RSA-AES128-SHA256   TLSv1.2 Kx=DH   Au=RSA Enc=AES(128) Mac=SHA256
ECDHE-ECDSA-AES256-SHA TLSv1 Kx=ECDH   Au=ECDSA Enc=AES(256) Mac=SHA1
ECDHE-RSA-AES256-SHA   TLSv1 Kx=ECDH   Au=RSA Enc=AES(256) Mac=SHA1
DHE-RSA-AES256-SHA   SSLv3 Kx=DH   Au=RSA Enc=AES(256) Mac=SHA1
ECDHE-ECDSA-AES128-SHA TLSv1 Kx=ECDH   Au=ECDSA Enc=AES(128) Mac=SHA1
ECDHE-RSA-AES128-SHA   TLSv1 Kx=ECDH   Au=RSA Enc=AES(128) Mac=SHA1
DHE-RSA-AES128-SHA   SSLv3 Kx=DH   Au=RSA Enc=AES(128) Mac=SHA1
……

Try Out…

- Wireshark: www.amazon.in

- openssl s_client -connect www.amazon.in:443
  - Shows only the leaf cert

- openssl s_client -showcerts www.amazon.in:443
  - Shows all certs in the path given by amazon.in

https://www.feistyduck.com/library/openssl-cookbook/online/ch-testing-with-openssl.html

Classification of TLS Vulnerabilities

I. Conceptual flaws in TLS and the resulting exploits
   - Protocol downgrades, connection renegotiation, session resumption, incomplete/vague specs
     - 3SHAKE, TLS Renego MITM attacks, POODLE, LOGJAM, FREAK

II. Vulnerabilities due to using weak crypto primitives
   - Block ciphers that operate in CBC mode
     - Sweet32, ROBOT, Lucky13

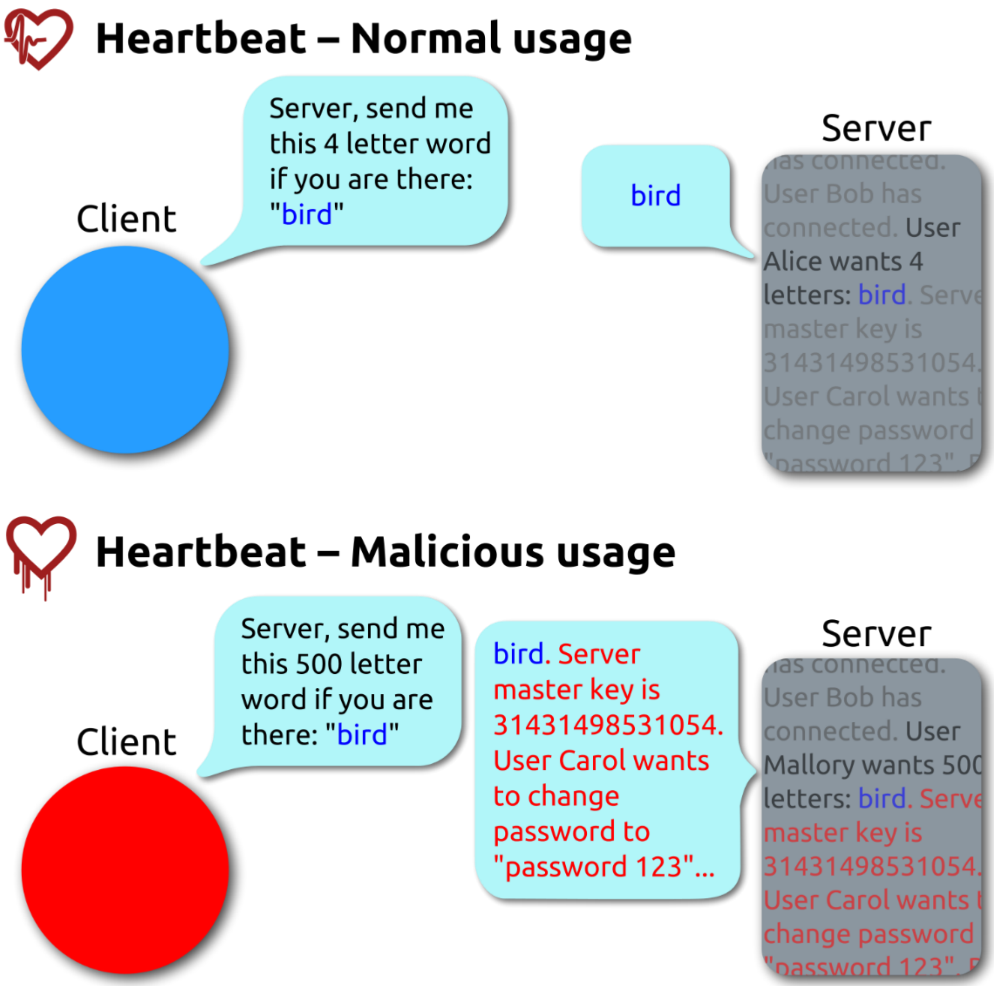
III. Implementation vulnerabilities
   - Faulty implementations gave rise to cross-layer protocol attacks and/or side channel attacks
     - BEAST, CRIME, TIME, BREACH, HEIST, SLOTH, DROWN
     - SMACK, ROCA, HeartBleed

SSL/TLS Attacks (in detail)

- Heartbleed attack on Heatbeat extension
- TLS DoS/DDoS attacks
- POODLE (Padding Oracle On Downgraded Legacy Encryption)
- *FREAK: A Downgrade attack*
- TLS Renegotiation MITM attacks
- Replay attacks

Heatbeat Extension

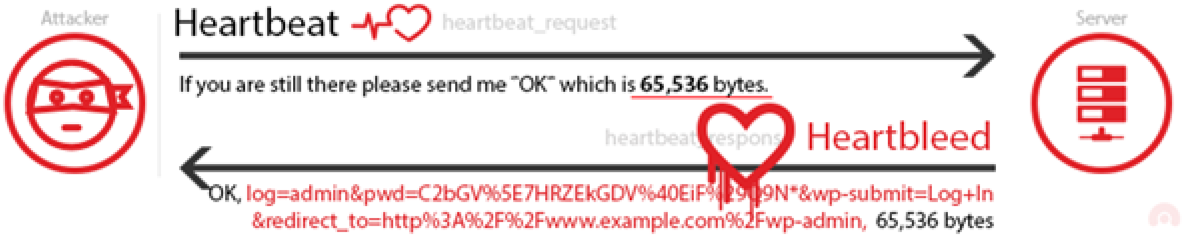
- DTLS/TLS: Heatbeats to keep the connection alive without continuous data transfer and renegotiations (liveness check)
- Liveness check at a much higher rate than TCP keep-alive feature
  - Multiples of RTT (1 sec to several minutes)
- DTLS uses it also for path MTU (PMTU) discovery
- Modes: peer_allowed_to_send / peer_not_allowed_to_send in Client_Hello message
- Alice: heartbeat_request and its echo (heartbeat_response) by Bob
- heartbeat_request: msgType, payloadLen, payload, padding



Credit: Fenix Feather

- Purpose: Proving to other party that connection is still alive by sending keep-alive messages
  - heartbeat extension ([RFC 6520](RFC 6520))
- It includes msg length
- Password/key leaking security bug in OpenSSL
- In 2014, affected 17% of SSL servers
- Is it design flaw in TLS?

SSL Heartbleed Attack



The coding mistake in OpenSSL that caused Heartbleed!

memcpy(bp, pl, payload);

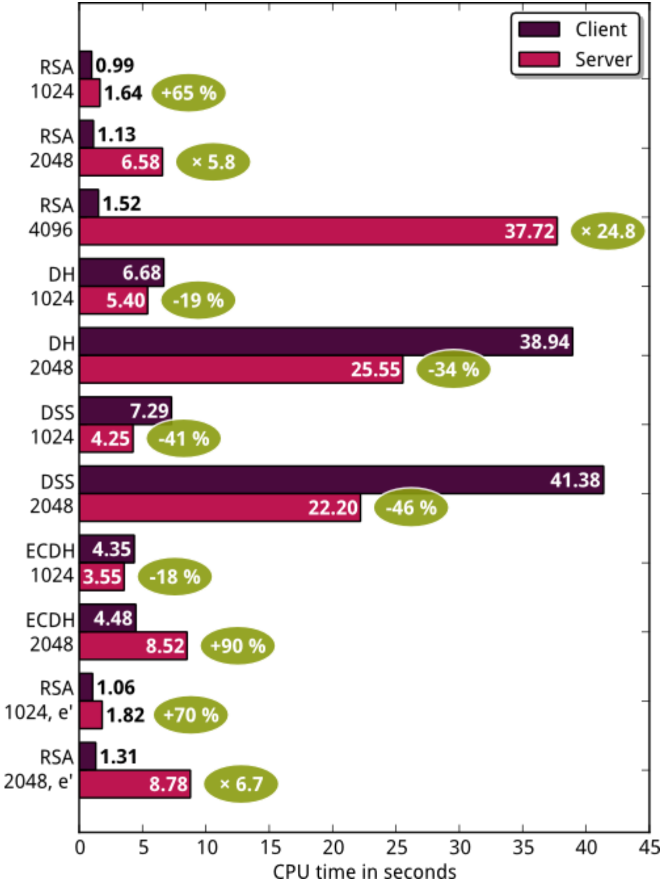
CVE-2014-0160

Prevention: Update to the latest version of OpenSSL and if that is not possible, recompile the already installed version with -DOPENSSL_NO_HEARTBEATS

https://heartbleed.com/

SSL/TLS DoS/DDoS Attacks

- HTTPS Floods
- Launching many SSL sessions per second
  - (Bogus) SSL handshake messages consume more resources (15x) at Server than at client (attacker)
    - Client encrypts pre-master-secret which server has to decrypt in RSA based key exchange
  - Solution: Rate limit TLS handshakes per source IP address at server



Src: [Link]

Openssl speed test

$openssl speed rsa ecdsa (SK) (PK)

	sign	verify	sign/s	verify/s
rsa 512 bits	0.000046s	0.000003s	21619.9	342048.0
rsa 1024 bits	0.000128s	0.000009s	7814.4	111940.4
rsa 2048 bits	**0.000885s**	**0.000028s**	**1130.5**	**36281.7**
rsa 3072 bits	0.003011s	0.000056s	332.1	17799.6
rsa 4096 bits	0.006268s	0.000099s	159.5	10088.4
rsa 7680 bits	0.054402s	0.000341s	18.4	2934.4
rsa 15360 bits	0.313750s	0.001314s	3.2	760.8

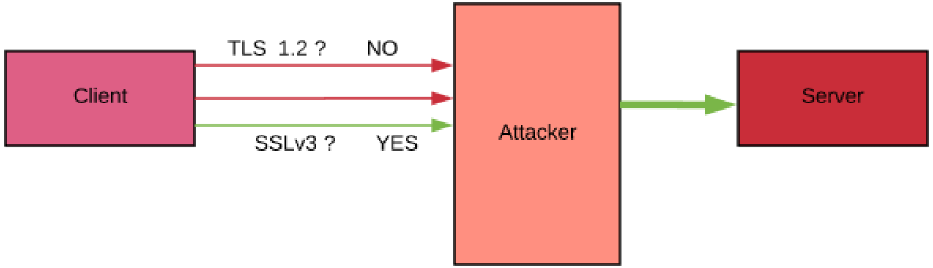
1 core

	sign	verify	sign/s	verify/s
160 bits ecdsa (secp160r1)	0.0002s	0.0002s	4482.6	5297.3
192 bits ecdsa (nistp192)	0.0003s	0.0002s	3699.6	4338.3
224 bits ecdsa (nistp224)	0.0001s	0.0001s	15738.8	7248.7
256 bits ecdsa (nistp256)	**0.0000s**	**0.0001s**	**30536.6**	**12621.9**
384 bits ecdsa (nistp384)	0.0010s	0.0007s	993.7	1351.0
521 bits ecdsa (nistp521)	0.0003s	0.0007s	2920.2	1522.8

- Also try $openssl speed to test for all algos
- $openssl speed -multi 2 rsa

POODLE Attack

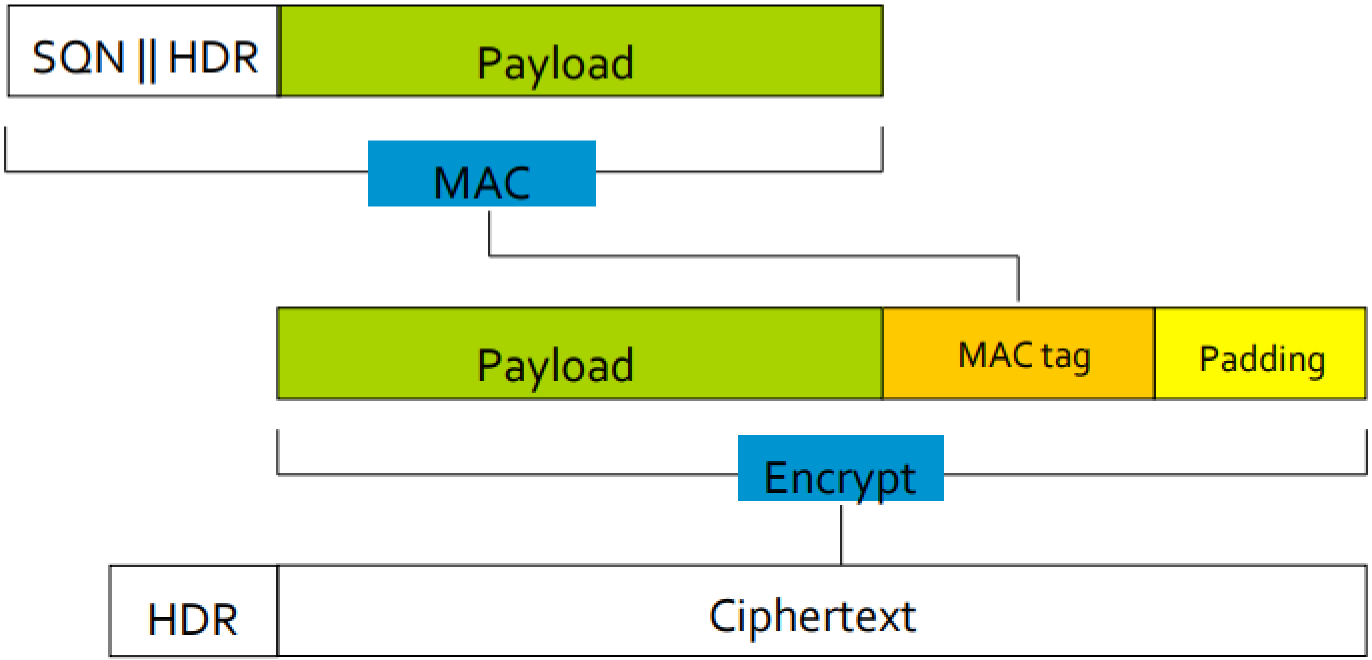
- Padding Oracle On Downgraded Legacy Encryption (POODLE)
- Exploited the fact that many servers still supported SSLv3 and suffer from a CBC vulnerability



https://github.com/thomaspatzke/POODLEAttack

[What Is the POODLE Attack? | Acunetix]()

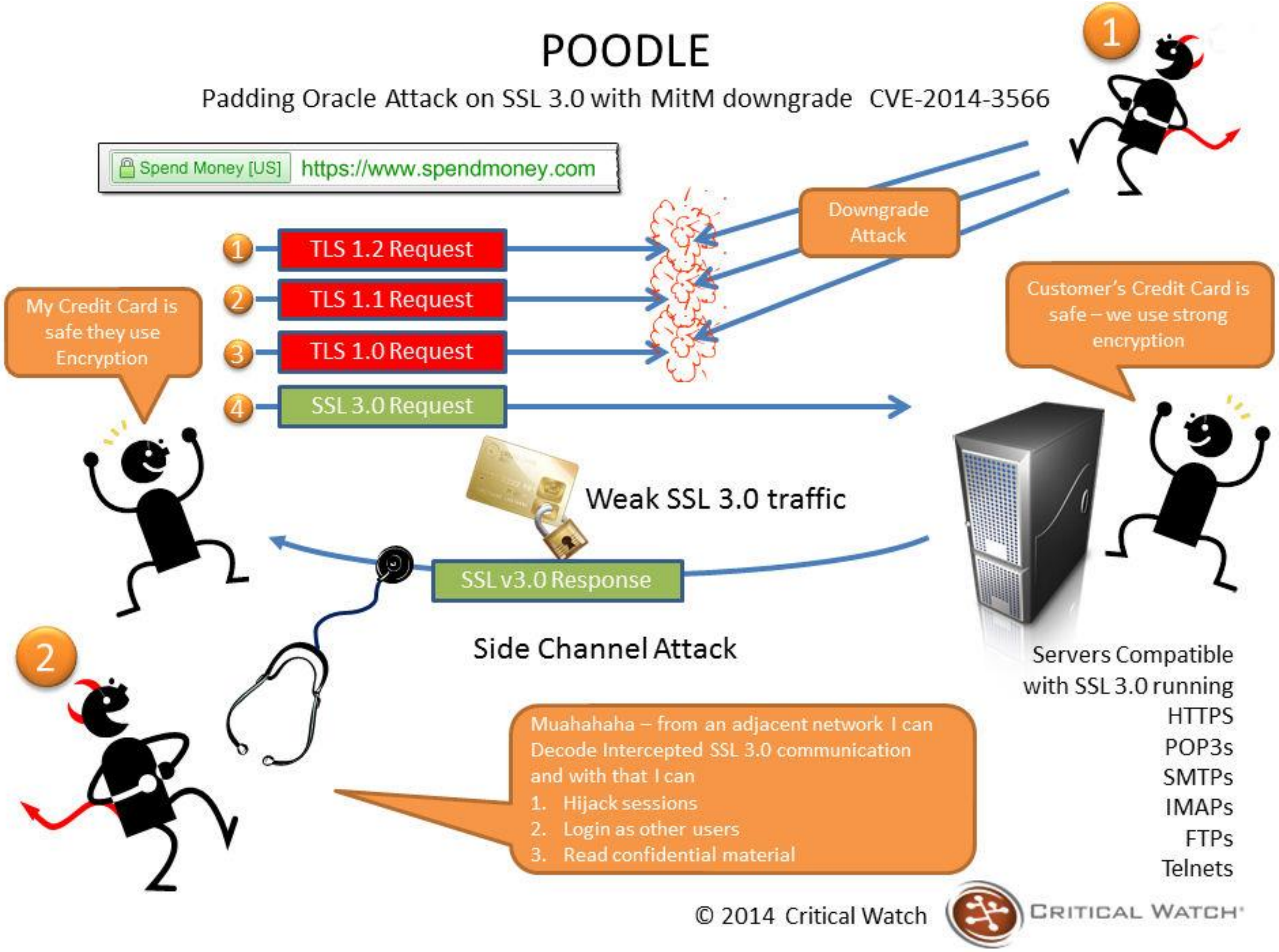
MAC and Padding in TLS



https://medium.com/@c0D3M/lucky-13-attack-explained-dd9a9fd42fa6

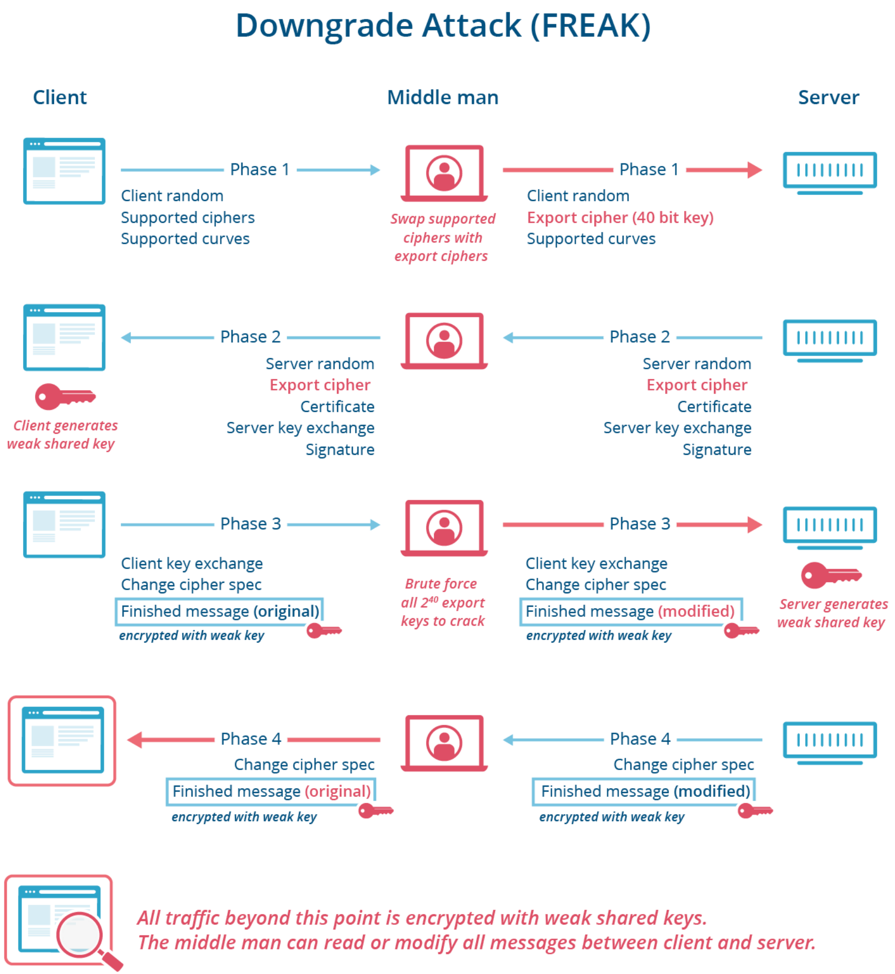
POODLE

Padding Oracle Attack on SSL 3.0 with MitM downgrade CVE-2014-3566



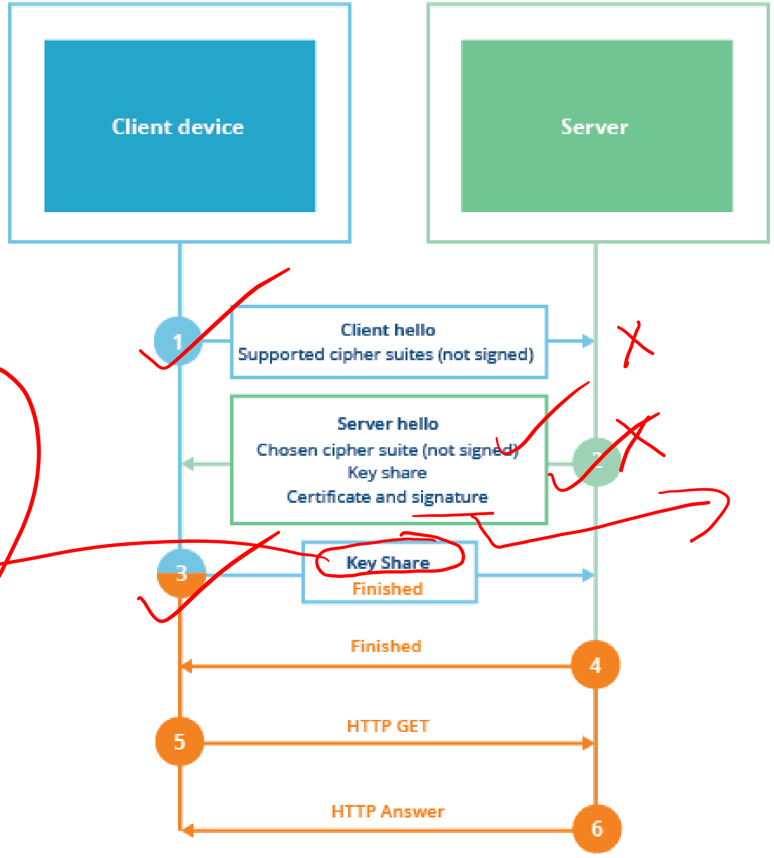
https://www.openssl.org/~bodo/ssl-poodle.pdf

- FREAK, LogJam & CurveSwap attacks took advantage of two things:
  - Support for weak ciphers in early TLS versions
  - Part of handshake which is used to negotiate which cipher to use is not signed



Phase-4: Attacker has to modify FINISHED msg before sending it to Client as it includes hash of all handshake msgs exchanged in both the directions. [FREAK Attack Explained | Medium]

TLS 1.2 ECDHE



g^a

Key Material $= g, P, g^b$

$\Downarrow$

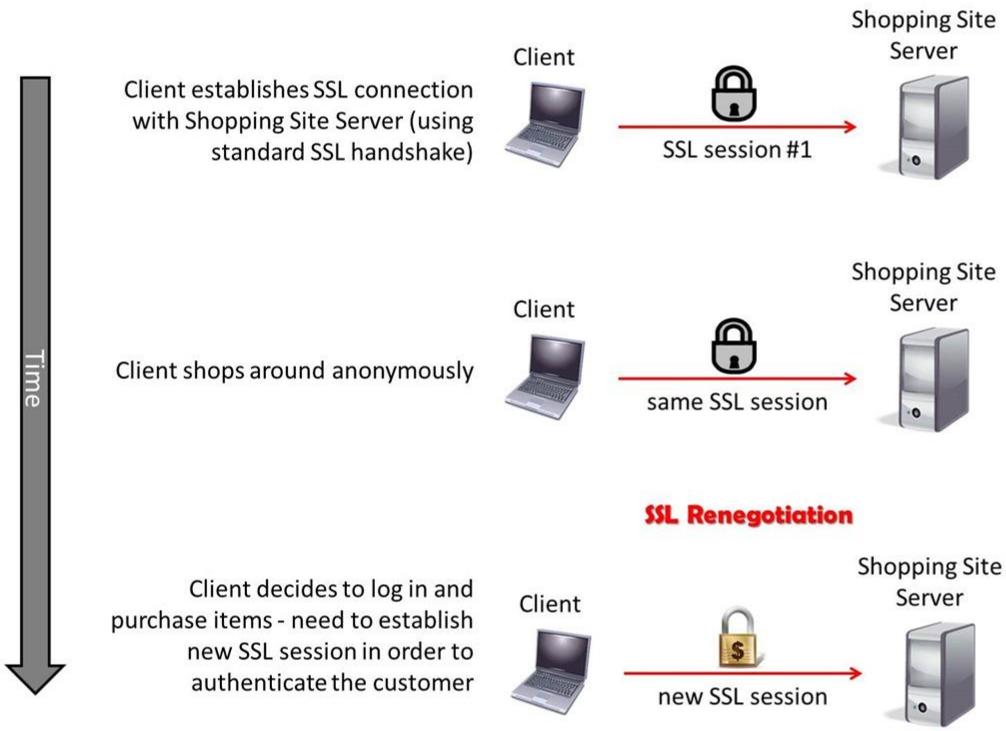
Sign(Digest, RSA Private key of server)

CLOUDFLARE

[The Illustrated TLS Connection: Every Byte Explained (ulfheim.net)](https://tls12.ulfheim.net/)

TLS renegotiation process

- TLS 1.2 allows either the client or the server to initiate renegotiation -- a new full handshake on encrypted channel to establish new cryptographic parameters
- Eg: HTTP servers support renegotiation to request client certs for a protected resource
- TLS renegotiation messages (including types of ciphers and key material) are encrypted and then sent over the existing SSL connection (encrypted)
- Unfortunately, although the new handshake is carried out using the cryptographic parameters established by the original handshake, there is no cryptographic binding between the two.

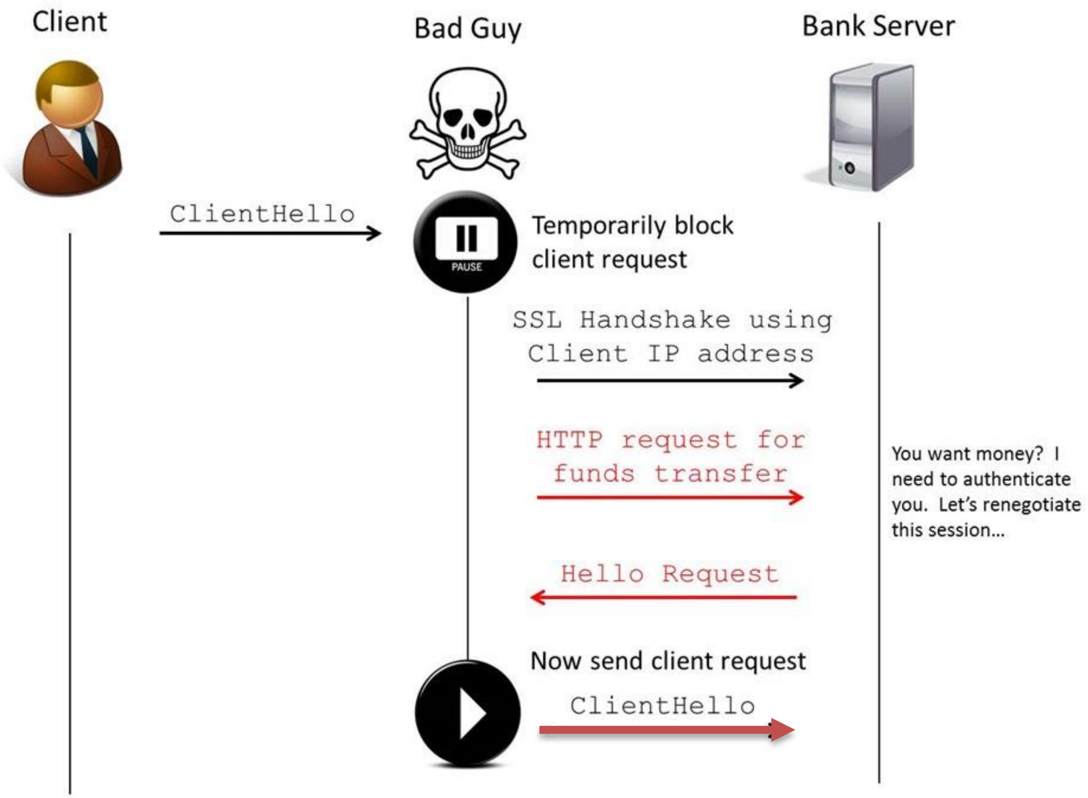


TLS Renegotiation Attacks

- Discovered by Marsh Ray and Steve of PhoneFactor (2009)
- Assumption: client-initiated renegotiation supported at Server
- DoS attack: One TCP session which is performing several expensive crypto operations
- A single server can perform between 150-300 handshakes per second while a single client can request around 1000 handshakes per second!
- Detection is hard bcz fewer TCP sessions unlike TCP SYN Flood attack and fewer visible TLS handshakes unlike TLS DoS attacks
- Webservers can limit no. of renegotiations per client or do not allow client-initiated renegotiations

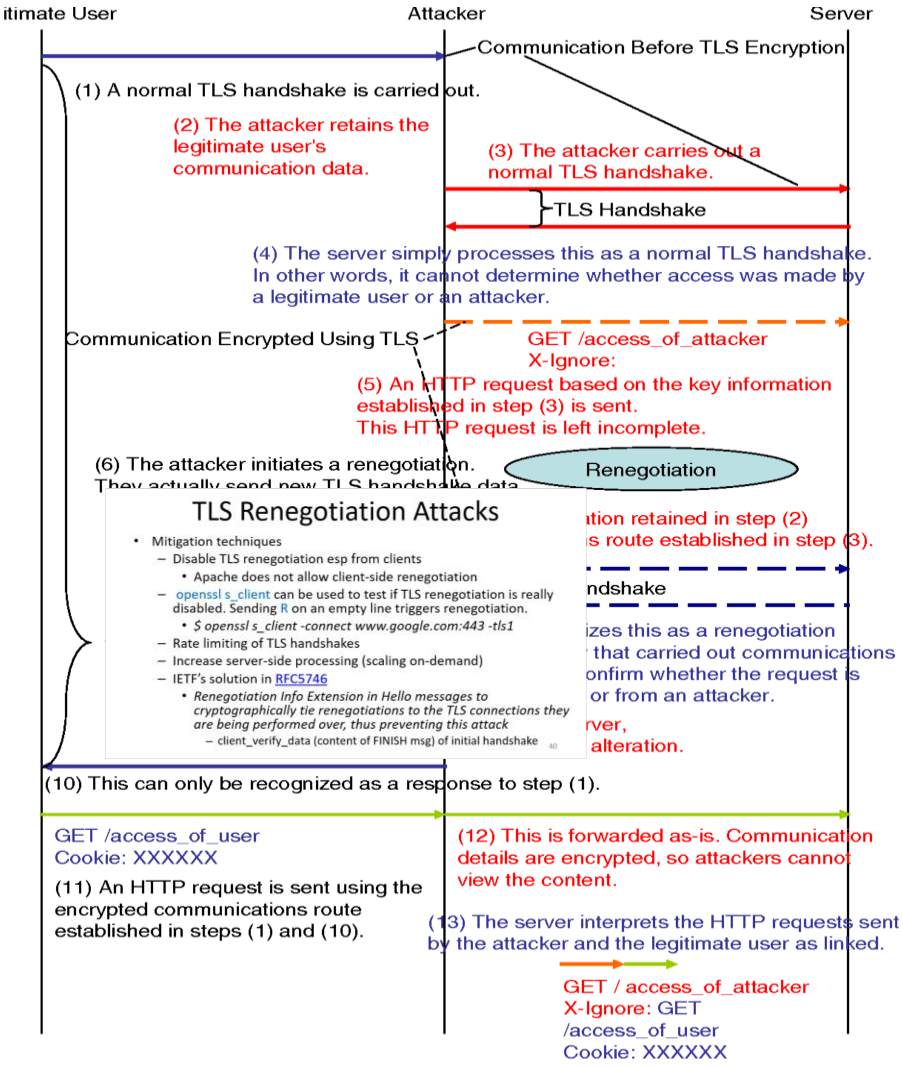
TLS renegotiation based MITM Attacks

(Server-Initiated Case)



Client completes handshake in clear text. Attacker intercepts handshake info and encrypts and sends to Server using Renegotiated session

TLS renegotiation based MITM Attacks (Client-Initiated Case)



- *Attacker sends incomplete HTTP request (Invalid header "X-Ignore", w/o a carriage return) that will be added as a prefix to client's own request*
- *HTTP is a stateless protocol, so cookies are used to link requests*
- The attacker cannot read this traffic from client, but the server believes that the initial traffic to and from the attacker is the same as that to and from the client.
- The attacker may be able to generate a request of his/her choice validated by the client's cookie☹

TLS Renegotiation Attacks

- Mitigation techniques
  - Disable TLS renegotiation esp from clients
    - Apache does not allow client-side renegotiation
  - openssl s_client can be used to test if TLS renegotiation is really disabled. Sending R on an empty line triggers renegotiation.
    - *$ openssl s_client -connect www.google.com:443 -tls1*
  - Rate limiting of TLS handshakes
  - Increase server-side processing (scaling on-demand)
  - IETF's solution in RFC5746
    - *Renegotiation Info Extension in Hello messages to cryptographically tie renegotiations to the TLS connections they are being performed over, thus preventing this attack*
      - client_verify_data (content of FINISH msg) of initial handshake

Outline

- TLS 1.3
  - Handshake Protocol
  - Key Schedule Generation
  - Session Resumption
  - Replay Attacks
- Backward compatibility and extensibility of TLS protocols

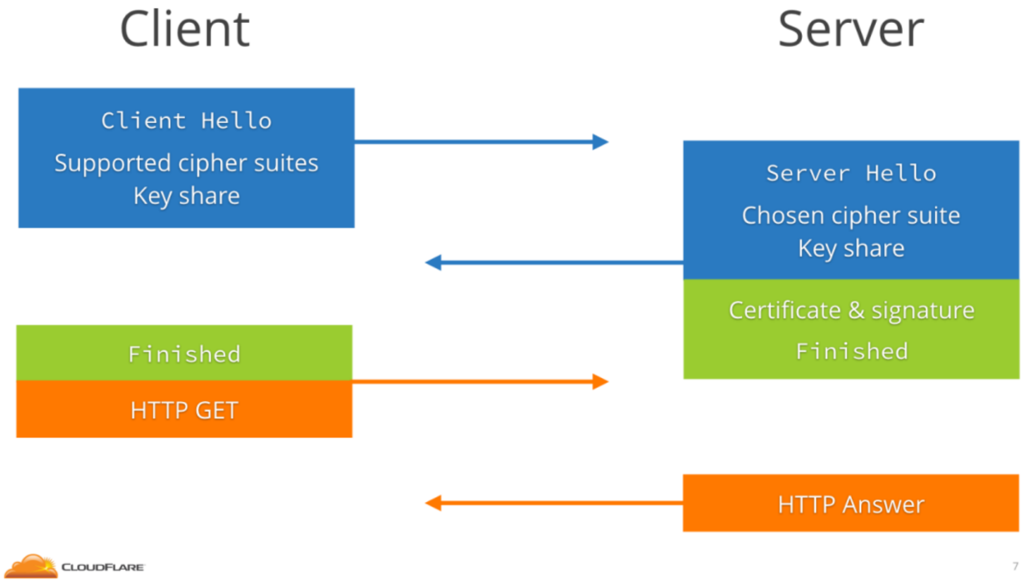
TLS 1.3 (RFC 8446)

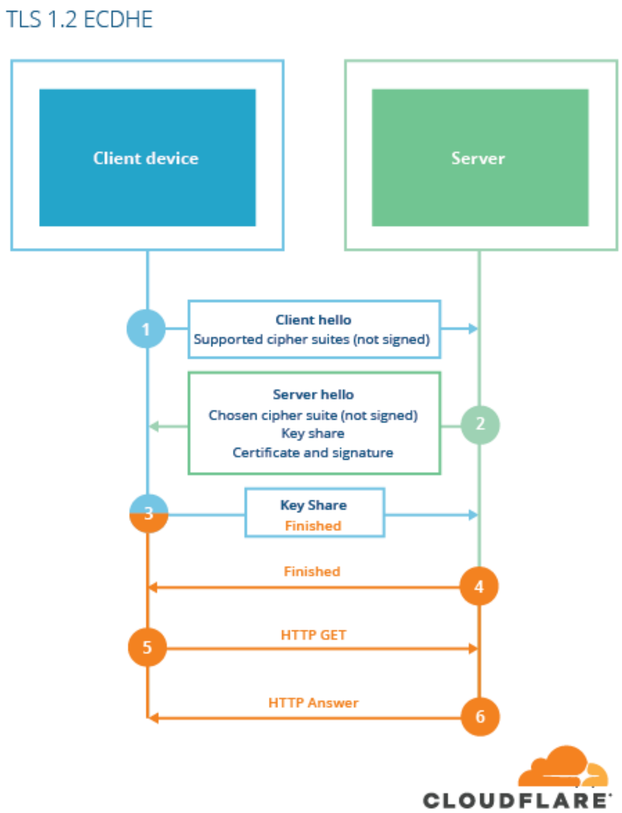
- Much faster and more secure compared to TLS 1.2
  - 1-RTT for connection setup (vs 2-RTT in TLS 1.2)
  - 0-RTT for resumption (vs 1-RTT in TLS 1.2) for some type of application data
- More of the handshake is encrypted and full handshake is signed
- Downgrade protection
- AEAD cipher suites (which compute MAC & Encrypt at one shot) instead of MAC-then-Encrypt in TLS 1.2
- Phased out insecure protocols, ciphers & algorithms present in TLS 1.2 while bringing in secure replacements
- Some of deprecated features in TLS 1.3
  - RC4 stream cipher, CBC mode ciphers (POODLE, Lucky 13)
  - Export-strength ciphers (FREAK)
  - RSA key transport (Only (EC)DHE, PSK-only and hybrid key agreement modes for ensuring forward secrecy) (DROWN)
  - Various Diffie-Hellman groups (Note: Restricted DH parameters to ones that are known to be secure)
  - SHA-1, MD5 (SLOTH)
  - DES, 3DES
  - Compression (CRIME)
  - Renegotiation

TLS 1.3 Cipher Suites

- Cipher suites are simplified a lot!
- Do not specify the certificate type (e.g., RSA/DSA/ECDSA) or the key exchange mechanism (e.g., DHE or ECHDE)
  - DHE and ECDHE
  - Pre-shared key (PSK)
  - PSK /w (EC)DHE
- Digital Signature (Authentication) algorithms
  - RSA (PKCS#1 variants)
  - ECDSA
- Only 5 Cipher Suites:
  - TLS_AES_256_GCM_SHA384
  - TLS_CHACHA20_POLY1305_SHA256
  - TLS_AES_128_GCM_SHA256
  - TLS_AES_128_CCM_8_SHA256
  - TLS_AES_128_CCM_SHA256

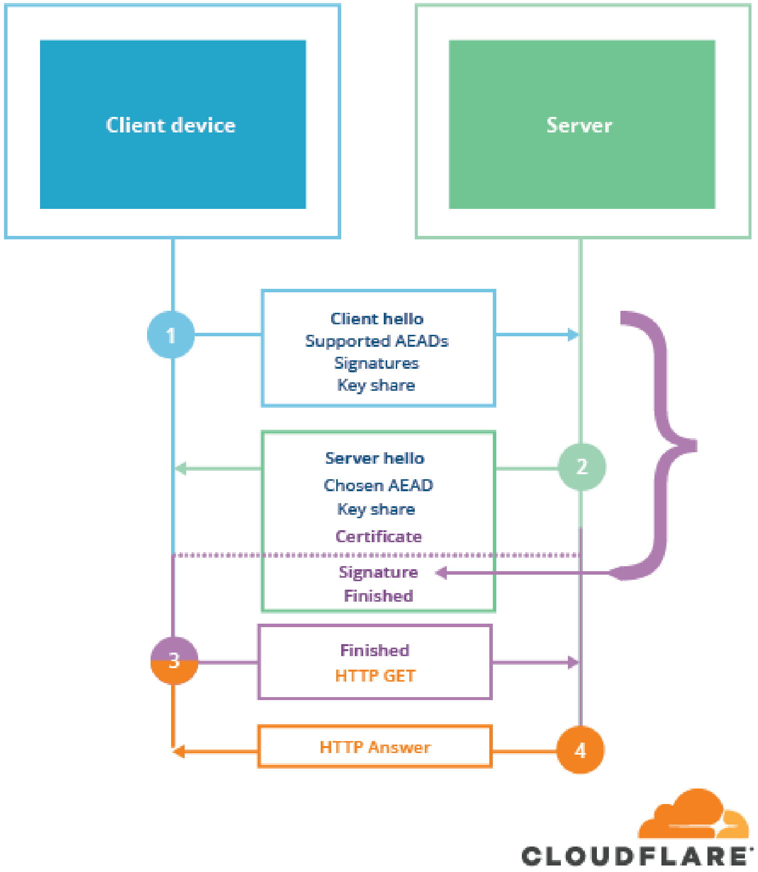
TLS 1.3 (ECDHE)



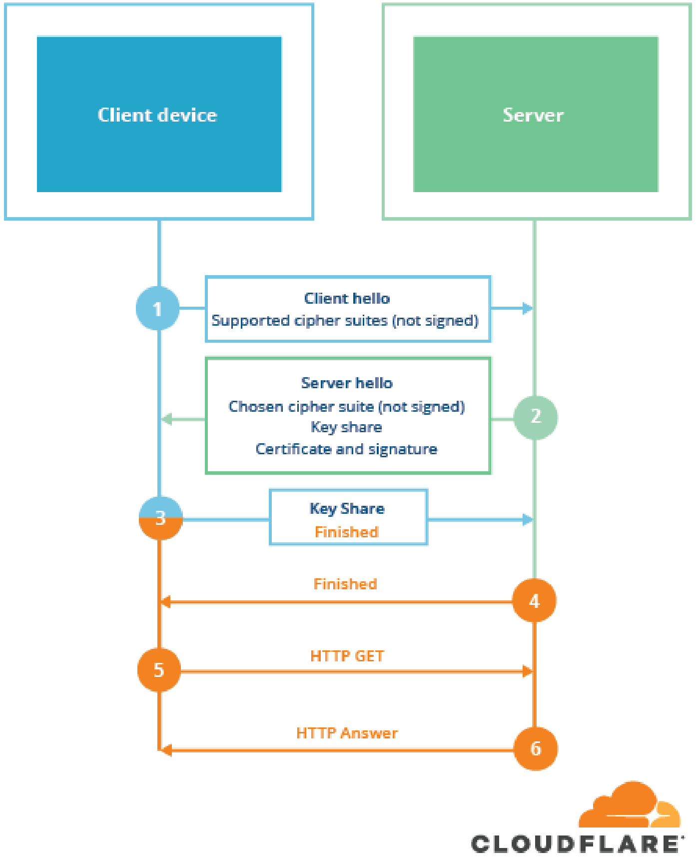


HelloRetryRequest instead of Server Hello,
if Client sends unacceptable (EC)DHE groups

TLS 1.3



TLS 1.2 ECDHE



Signature extension in Server to Client: All previous msgs are signed with RSA private key of Server to ensure no tampering in TLS 1.3

TLS 1.3

Client		**Server**
ClientHello {+KeyShare}	⟶	
		ServerHello {+KeyShare} EncryptedExtensions CertificateRequest Certificate CertificateVerify Sign Finished
	⟵	[Application Data]
Certificate CertificateVerify Finished [Application Data]	⟶	
Application Data	⟷	Application Data

GET

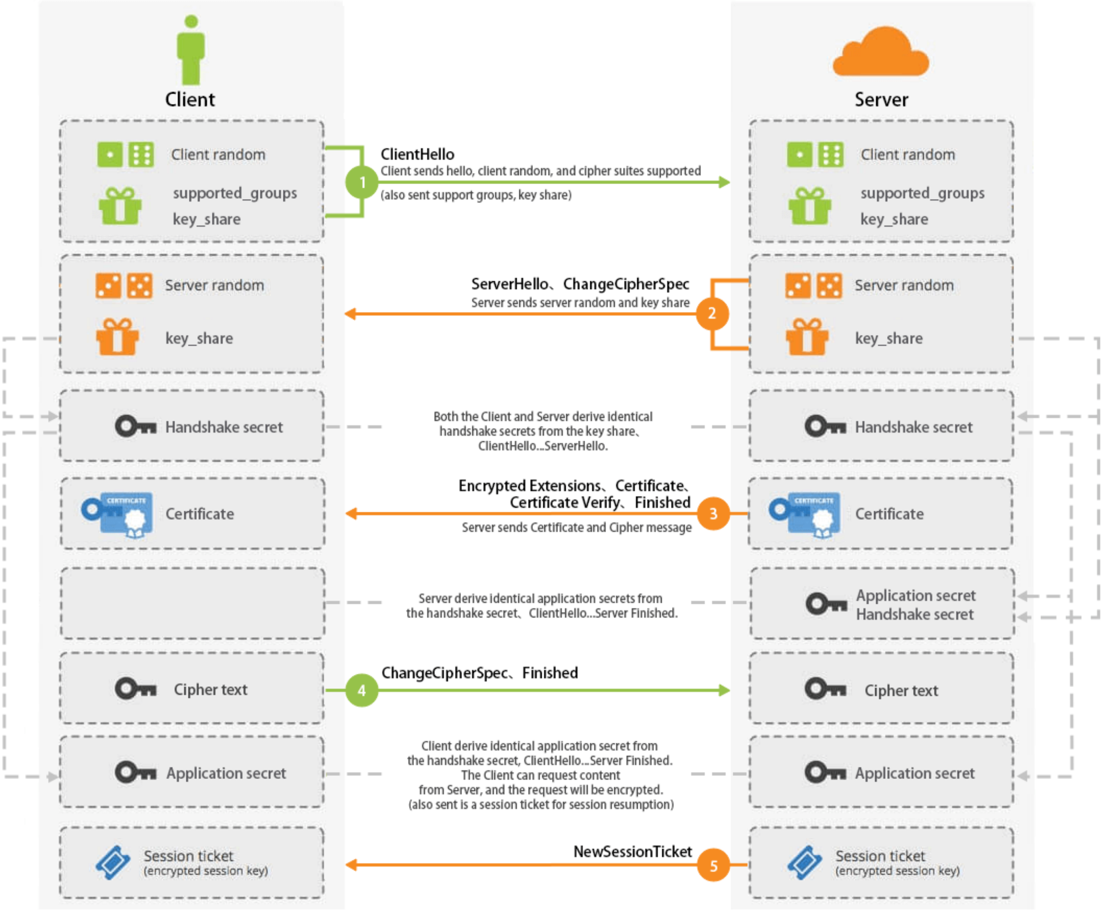
Respond

Pre-fetch

https://tools.ietf.org/html/rfc8446
https://www.wolfssl.com/tls-1-3-performance-part-5-client-server-authentication/

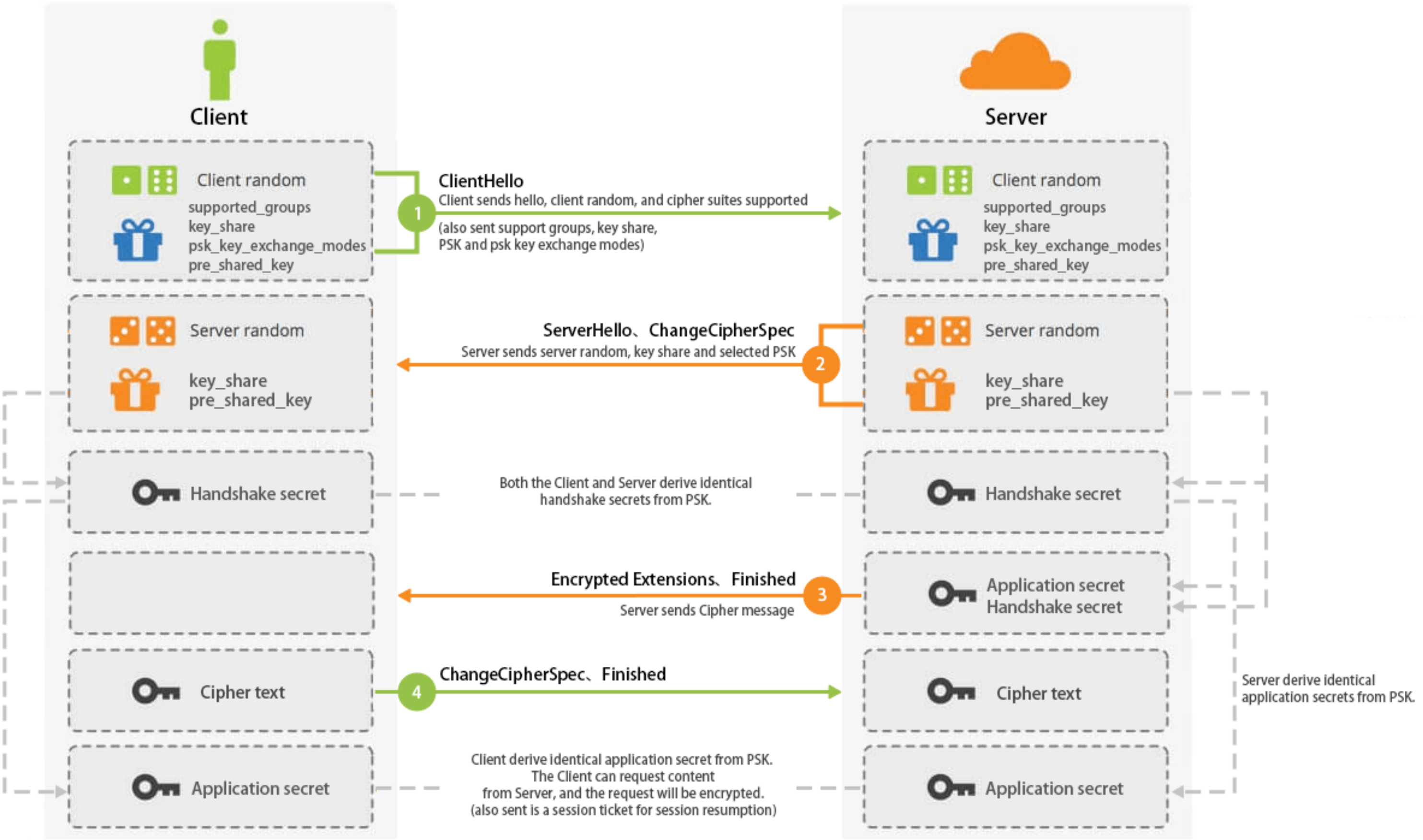
TLS Handshake (Diffie-Hellman)

Handshake

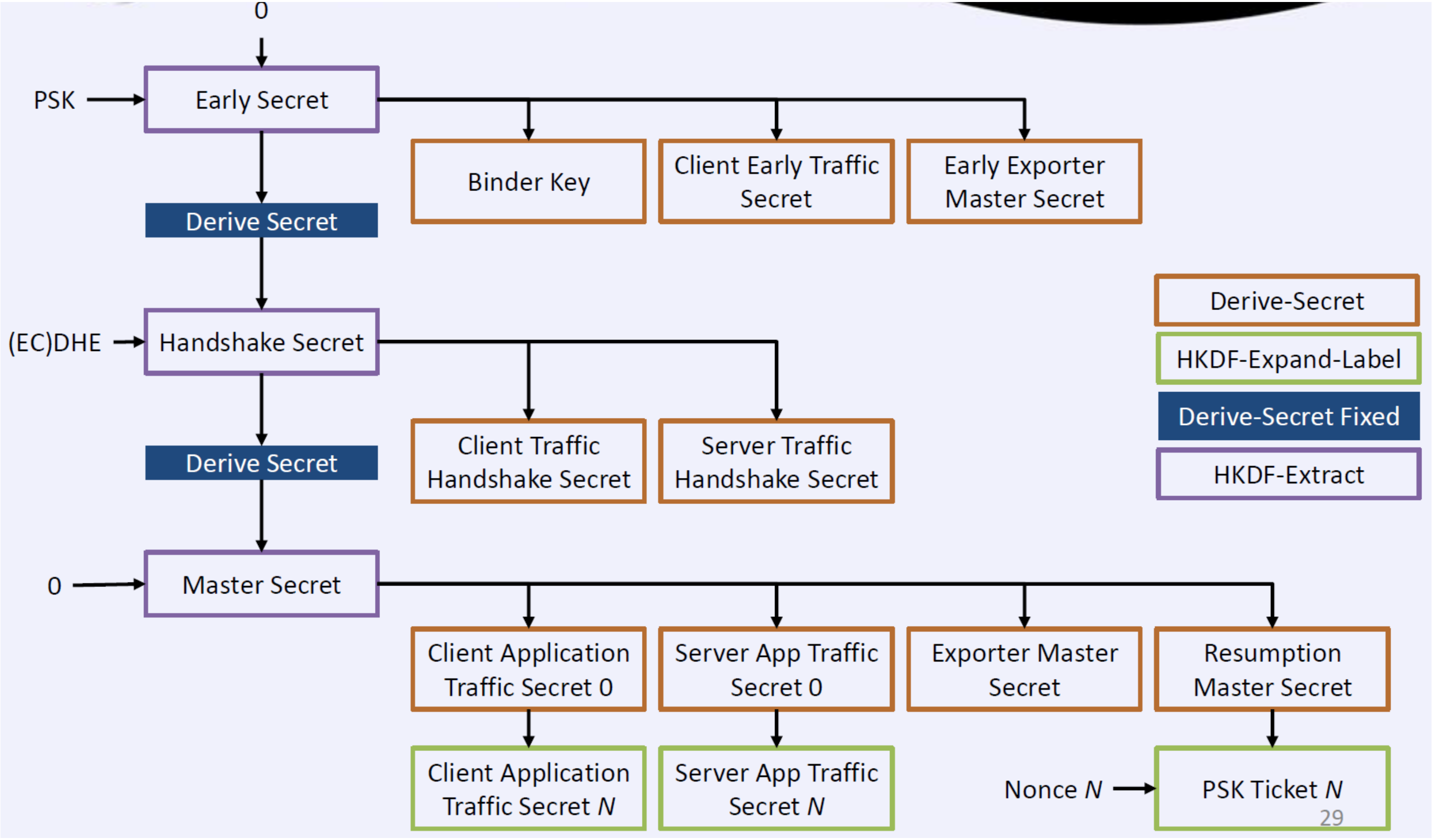


TLS Handshake (Diffie-Hellman) With PSK

Handshake

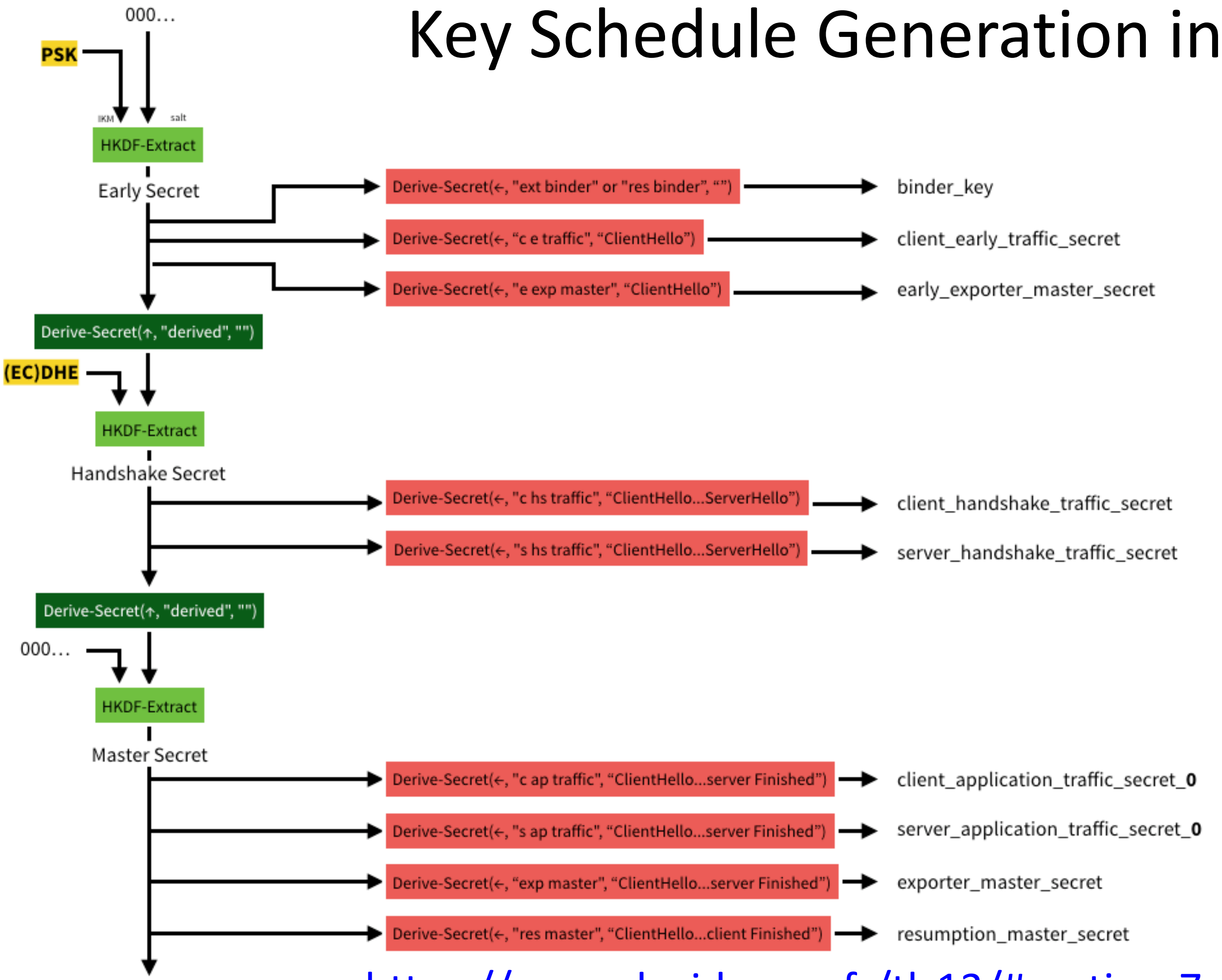


Key Schedule Generation in TLS 1.3



[RFC 5869 - HMAC-based Extract-and-Expand Key Derivation Function (HKDF) (ietf.org)]

Key Schedule Generation in TLS 1.3

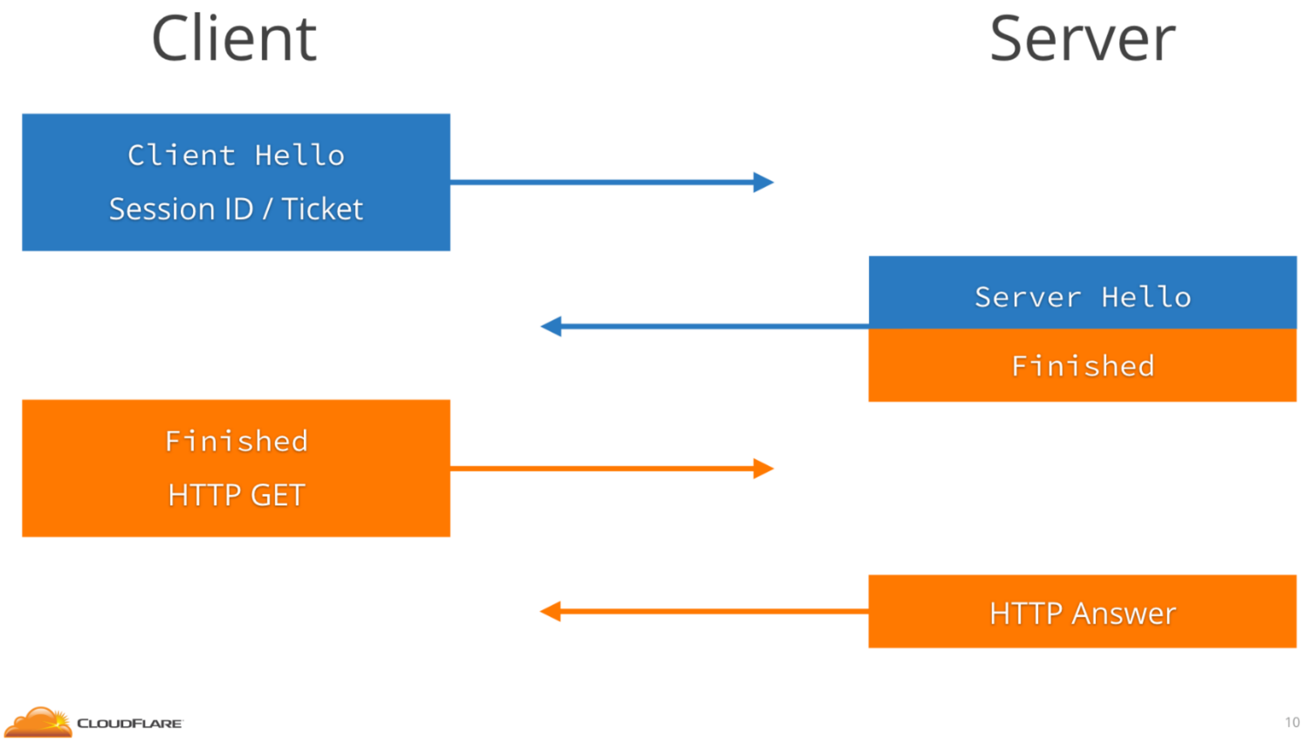


https://www.davidwong.fr/tls13/#section-7

Outline

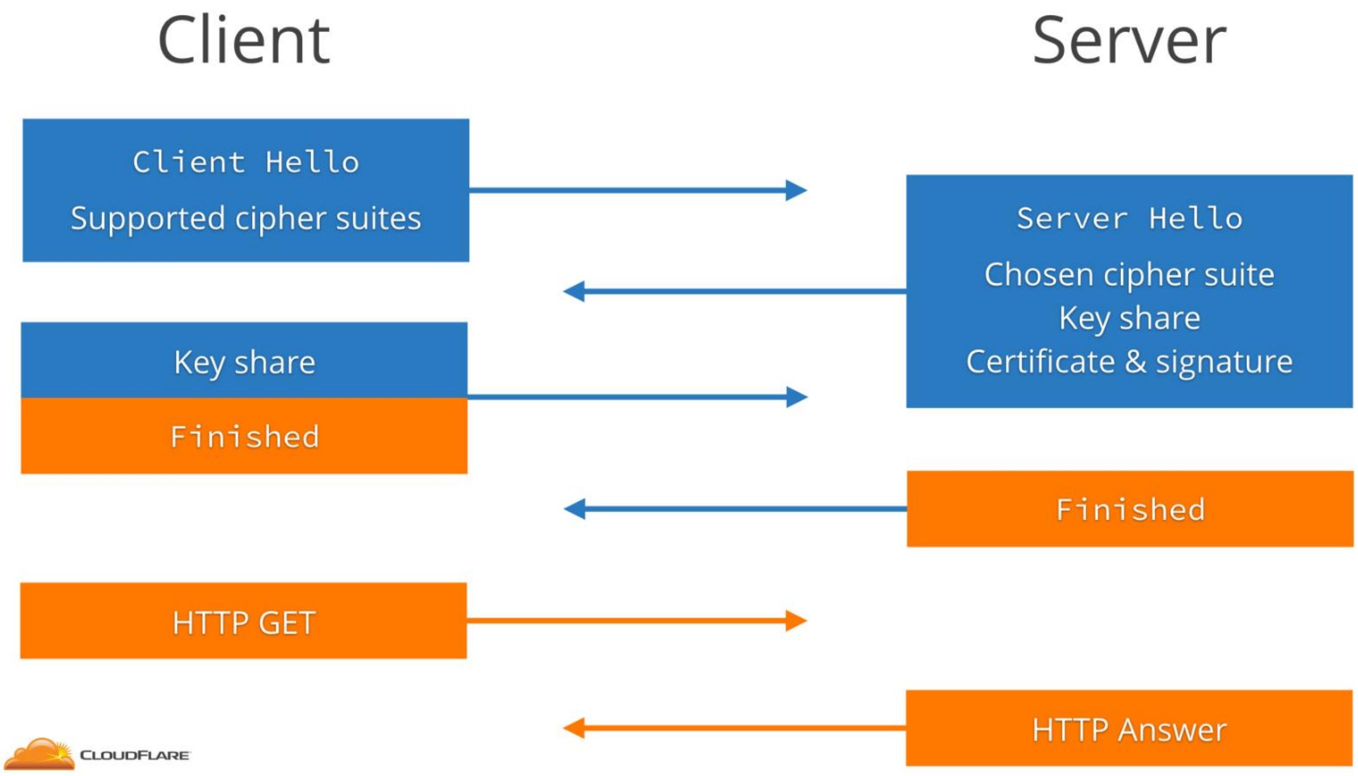
- TLS 1.3
  - Handshake Protocol
  - Key Schedule Generation
  - Session Resumption
  - Replay Attacks
- Backward compatibility and extensibility of TLS protocols

TLS 1.2 Session Resumption (1-RTT)



[RFC 5077 - Transport Layer Security (TLS) Session Resumption without Server-Side State (ietf.org)]
[TLS Stateful vs Stateless Session Resumption - DevCentral (f5.com)]

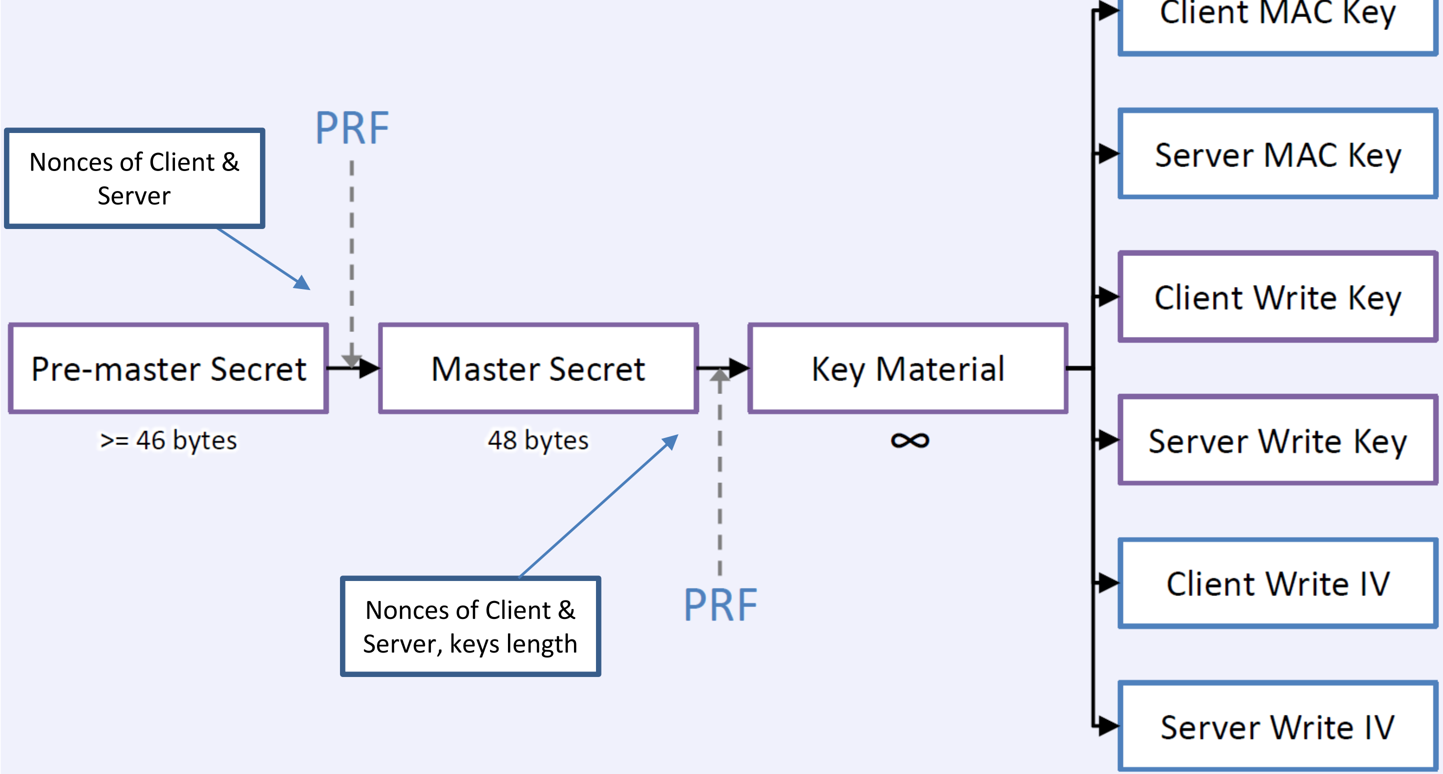
TLS 1.2 (ECDHE)



[Diffie–Hellman key exchange - Wikipedia]

Key Generation in TLS 1.2

https://www.ietf.org/rfc/rfc5246.txt

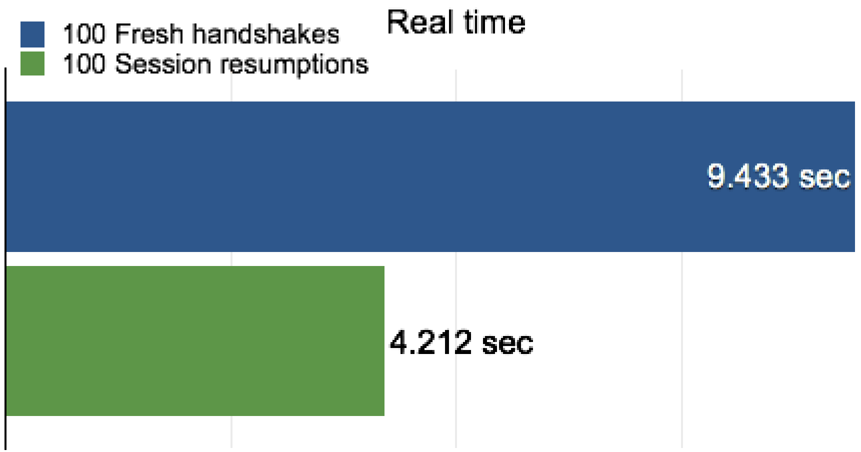


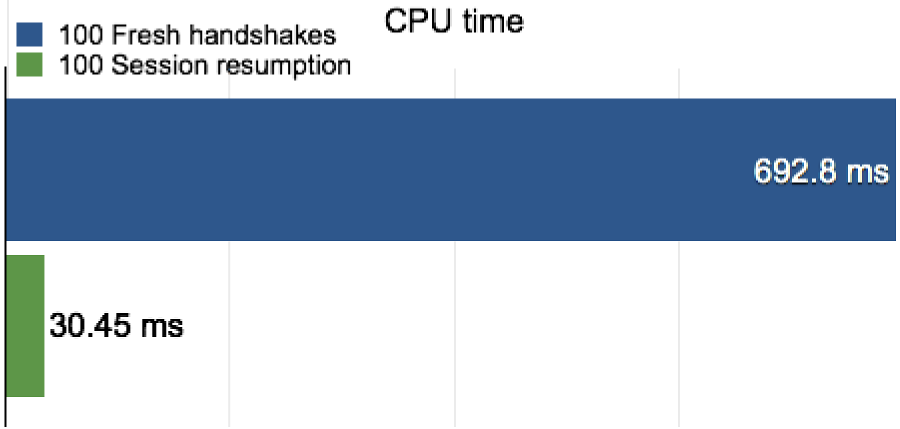
Session Resumption in TLS 1.2

- Helps to speed up the handshake process
- Do not confuse Session Resumption with Session Renegotiation
  - Renegotiation uses the same TLS/TCP connection to renegotiate security parameters which does not involve Session ID or Session Tickets

1) Stateful Resumption (Server side, Session ID)
   - Server saves state of the previous session, in particular the KeyMaterial derived, in a db/cache indexed with Session ID
   - Stored *MasterSecret* is used to generate session keys together with new *random variables* for the new *session*

Performance of Session ID Resumption



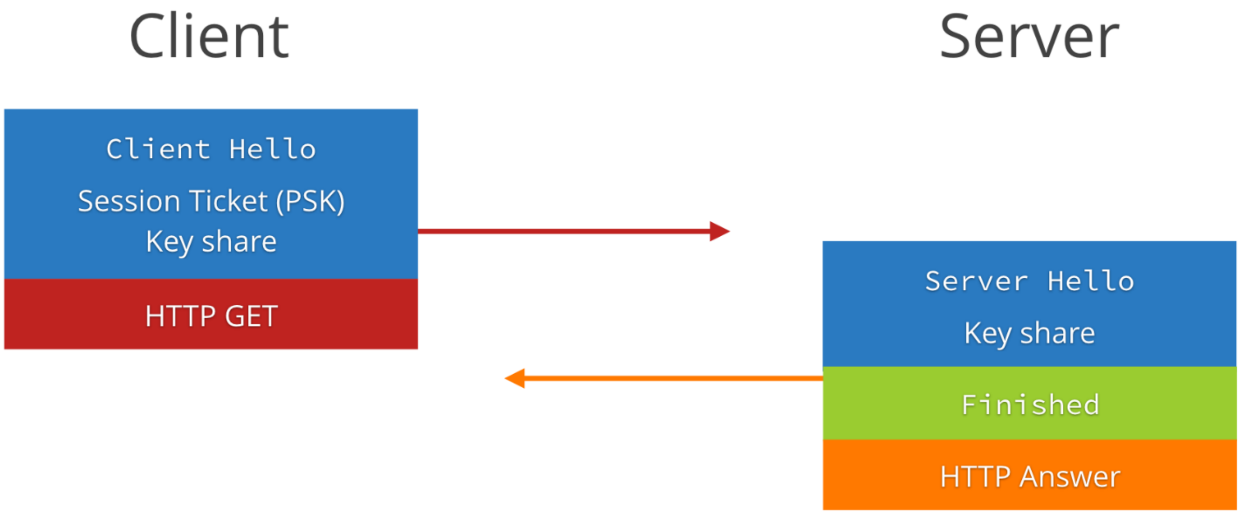


https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/

Session Resumption in TLS 1.2

2) Stateless Resumption (Client side, Session Ticket) RFC5077
   - Server sends Session Ticket to client in the previous session
     - Improves scalability of servers
     - Helps to load balance requests (TLS sessions) across servers
       - Possible in case of Session ID based resumption?
     - Ticket includes encrypted state <ciphersuite, *MasterSecret, and Timestamp (for checking validity of ticket), et> and MAC*
       - Encrypted using a unique session ticket enc key (STEK) known only to the server and MAC protected (HMAC-SHA-256)
     - *NewSessionTicket* handshake msg from the server contains *session ticket and* ticket_lifetime_hint (not encrypted)
       - A server may treat a ticket as valid for a shorter or longer period of time than what is given in the ticket_lifetime_hint ☺
     - Protection against Stolen tickets, Forged tickets and DoS attacks by choosing strong encryption/MAC algos

TLS 1.3 Resumption (0-RTT Data)

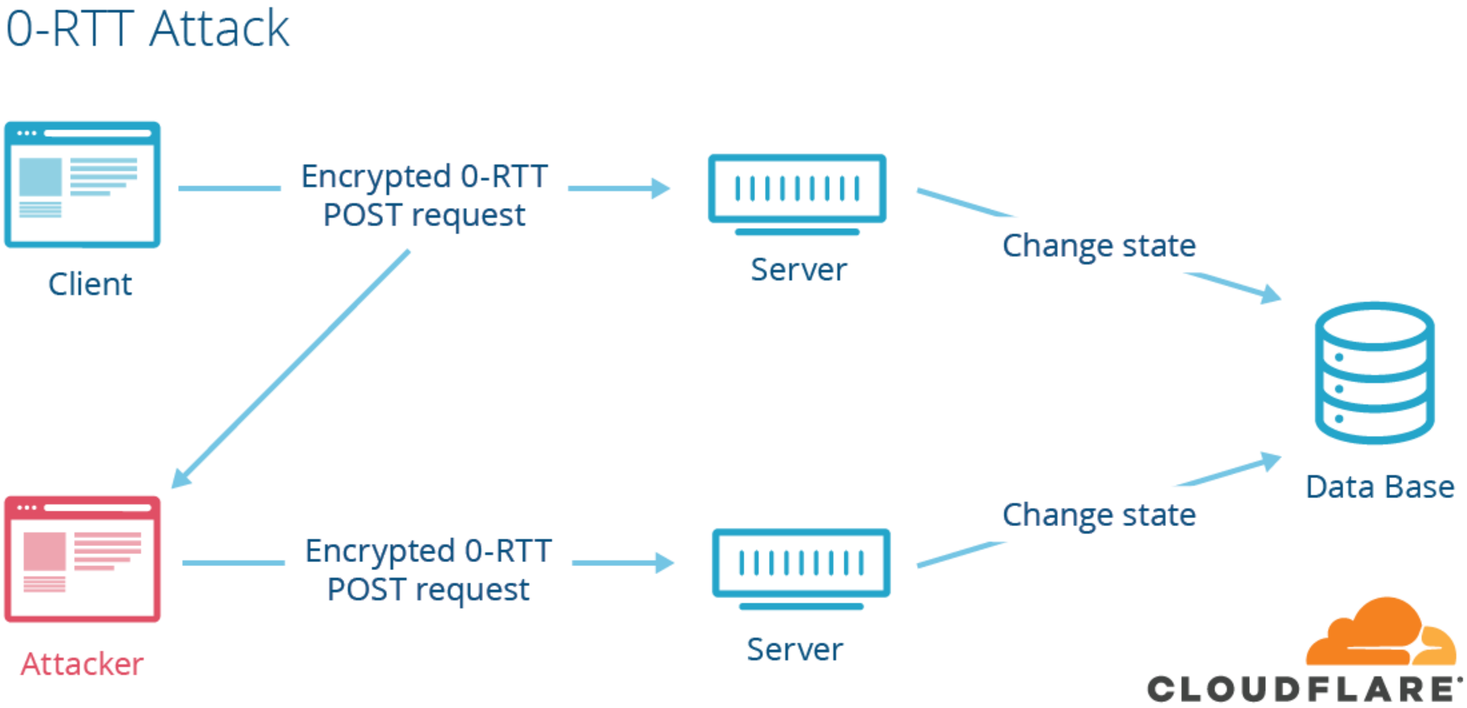


Note: Session Ticket=PSK encrypted by Server /w[12] session ticket enc key (STEK) known only to the server

Issues /w 0-RTT Resumption

- Lack of full forward secrecy
  - If session ticket enc keys (STEK) are compromised, an attacker can decrypt 0-RTT data (encrypted using PSK as salt to HKDF-Extract function) sent by the client on the first flight (but not the rest of the data in session)
  - Solution: Rotating STEKs regularly (e.g., weekly)
- Replay attacks
  - Attacker replays 0-RTT data (e.g., HTTP GET/POST)
  - POST is not replay-safe
  - Solutions: 1) Don’t allow 0-RTT resumption on POST and allow resumptions only for some period on “safe” requests (GET) and 2) Each session tkt is valid only once

Replay attacks on TLS 1.3!



[Introducing Zero Round Trip Time Resumption (0-RTT) (cloudflare.com)]

Backward Compatibility & Extensibility

- Version in Record Header is set as "3,1" (TLS 1.0) instead of "3,4" (TLS 1.3 for interoperability with earlier implementations
- Even for TLS 1.3, client protocol version is hardcoded to "3,3" (TLS 1.2) in Handshake messages for backward compatibility with middleboxes and legacy servers!
  - Version negotiation is performed using the "Supported Versions" extension in client & server hello msgs
- A number of TLS protocol values, referred to as GREASE (Generate Random Extensions And Sustain Extensibility) values, are reserved
  - A client **MAY** select one or more GREASE cipher suite values at random and advertise them in the "cipher_suites" field
  - Correctly implemented server will ignore these values like any unknown values so that new capabilities may be introduced while maintaining interoperability

[RFC 8701: Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility]

Research Articles

- Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users across the Web via TLS Session Resumption. In Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18). Association for Computing Machinery, New York, NY, USA, 289–299. DOI: https://doi.org/10.1145/3274694.3274708

- Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2020. Enhanced performance for the encrypted web through TLS resumption across hostnames. In Proceedings of the 15th International Conference on Availability, Reliability and Security (ARES '20). Association for Computing Machinery, New York, NY, USA, Article 16, 1–10. DOI: https://doi.org/10.1145/3407023.3407067

References

- SSL Server Test (Powered by Qualys SSL Labs)
- Computer and Network Security by Avi Kak (purdue.edu)
- A Cryptographic Analysis of the TLS 1.3 Handshake Protocol:1044.pdf (iacr.org)
- RFC 8701: Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility (rfc-editor.org)
- The Illustrated TLS Connection: Every Byte Explained (ulfheim.net)
- The Illustrated TLS 1.3 Connection: Every Byte Explained (ulfheim.net)
- https://en.wikipedia.org/wiki/Transport_Layer_Security
- RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2 (ietf.org)
- Networking 101: Transport Layer Security (TLS) - High Performance Browser Networking (O'Reilly) (hpbn.co)
- SSL/TLS beginner’s tutorial. This is a beginner’s overview of how… | by German Eduardo Jaber De Lima | Talpor | Medium
- Tutorial: SMTP Transport Layer Security (fehcom.de)
- Diffie–Hellman key exchange – Wikipedia
- What Is the POODLE Attack? | Acunetix
- Examples of TLS/SSL Vulnerabilities TLS Security 6: | Acunetix

References

- https://vincent.bernat.ch/en/blog/2011-ssl-dos-mitigation
- https://www.gnutls.org/documentation.html
- https://www.us-cert.gov/ncas/alerts/TA14-290A
- https://www.thesslstore.com/blog/tls-1-3-approved/
- https://vimeo.com/177333631
- https://www.cloudflare.com/learning-resources/tls-1-3/
- https://en.wikipedia.org/wiki/Transport_Layer_Security
- https://www.davidwong.fr/tls13/
- https://caniuse.com/#feat=tls1-3
- https://www.wolfssl.com/docs/tls13/
- https://www.fehcom.de/qmail/smtptls.html
- https://www.cloudflare.com/ssl/encrypted-sni/
- TLS Session Resumption: Full-speed and Secure (cloudflare.com)
- https://www.cloudinsidr.com/content/known-attack-vectors-against-tls-implementation-vulnerabilities/
- OpenSSL Cookbook: Chapter 1. OpenSSL Command Line (feistyduck.com)
- https://www.cryptologie.net/article/340/tls-pre-master-secrets-and-master-secrets/
- TLS Stateful vs Stateless Session Resumption - DevCentral (f5.com)
- Session Resumption Protocols and Efficient Forward Security for TLS 1.3 0-RTT (springer.com)
- Introducing Zero Round Trip Time Resumption (0-RTT) (cloudflare.com)

Announcements

- Quiz-1
  - Feb 28th at 2PM
  - Syllabus: L1 to L4
- Term Projects
- Secure Chat Programming Assignment