Web Security

PART II: TLS/SSL

Dr. Bheemarjuna Reddy Tamma

IIT HYDERABAD

Note: This is revised version of slide deck of Prof. Dan Boneh (Stanford) with material from various Internet sources

Outline

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

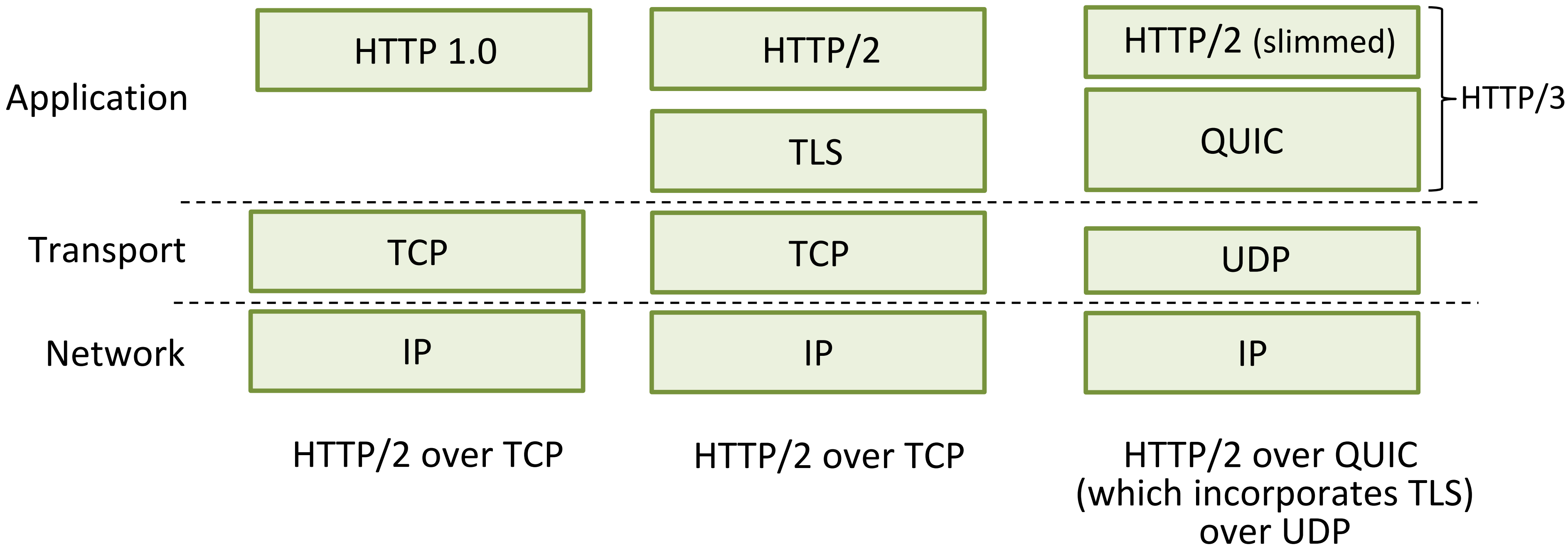
Transport Layer Security (TLS)

- 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*

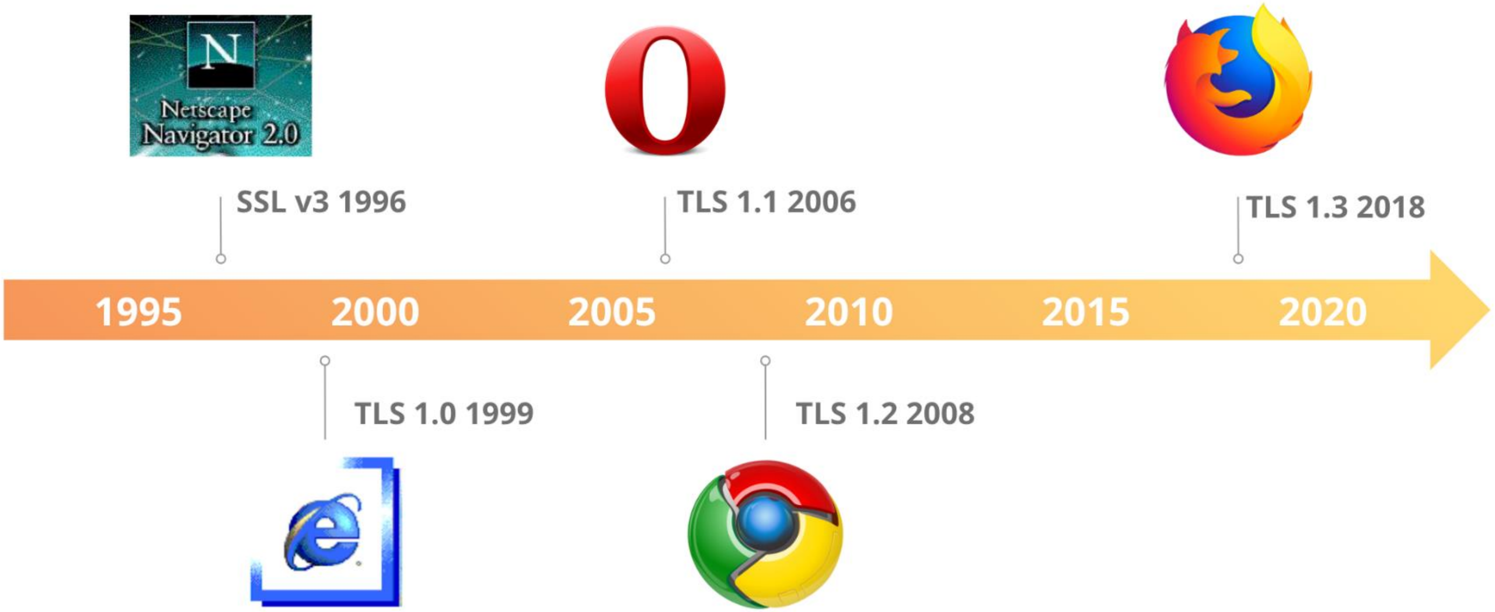
all techniques we have studied!

Transport Layer Security (TLS)

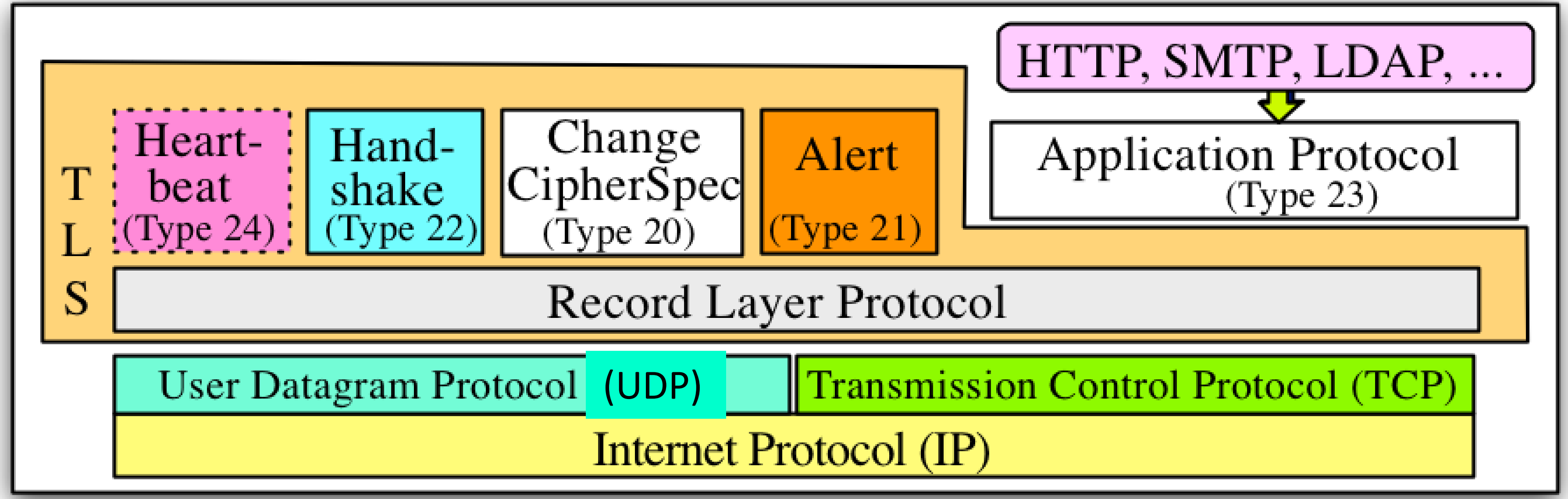
- TLS provides an API that *any* application can use
- HTTP view of TLS:



SSL/TLS Variants



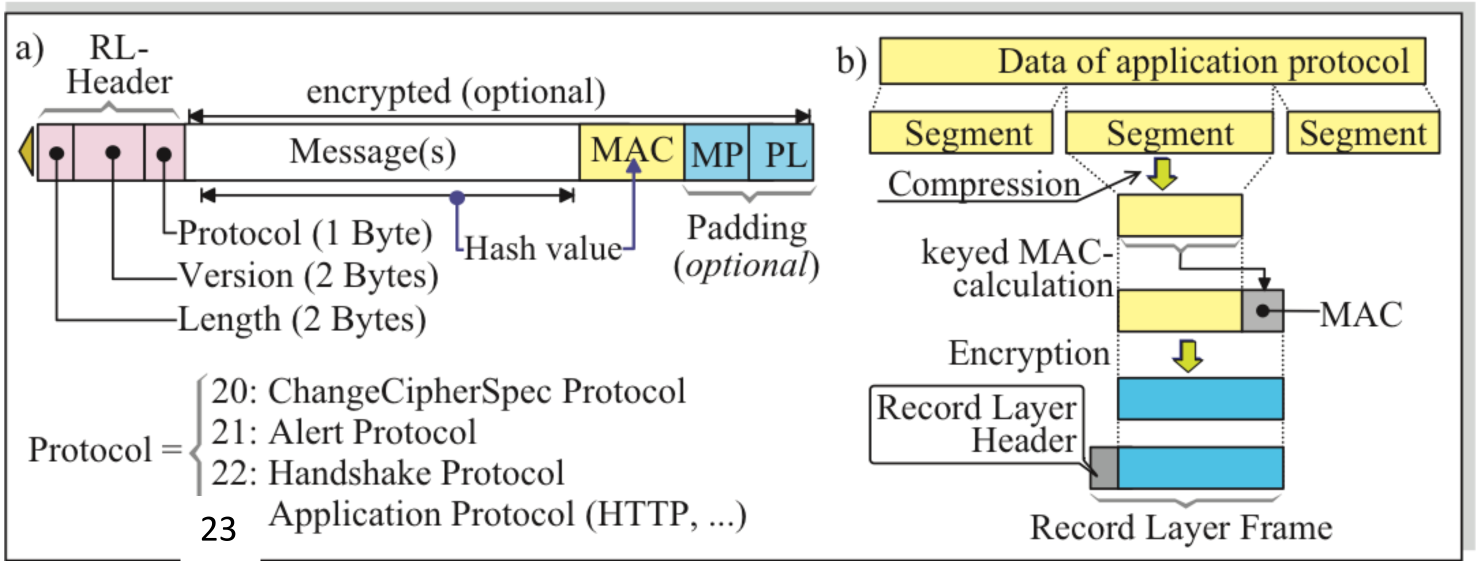
Layered Architecture of TLS



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), Encryption
  - Finally, adding required RL Header



Four Phases of TLS Handshake Protocol

- **Phase-1**

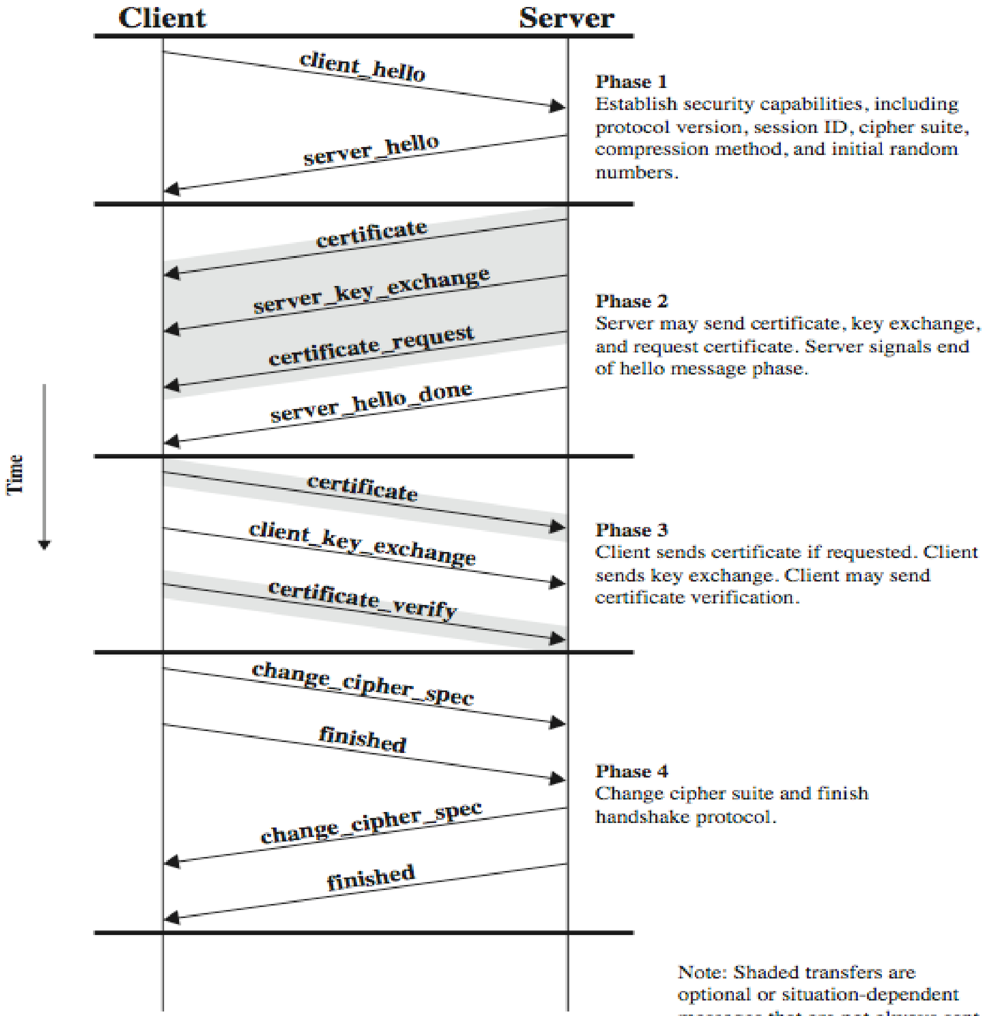
  Both ends agree upon Cipher Suite
  - TLS_**RSA**_WITH_**AES**_256_CBC_**SHA**256
  - TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
  - AEAD_**AES**_256_GCM_**SHA**384 (TLS 1.3)
- **Phase-2**

  Server sends its digital Cert signed by a CA
- **Phase-3**

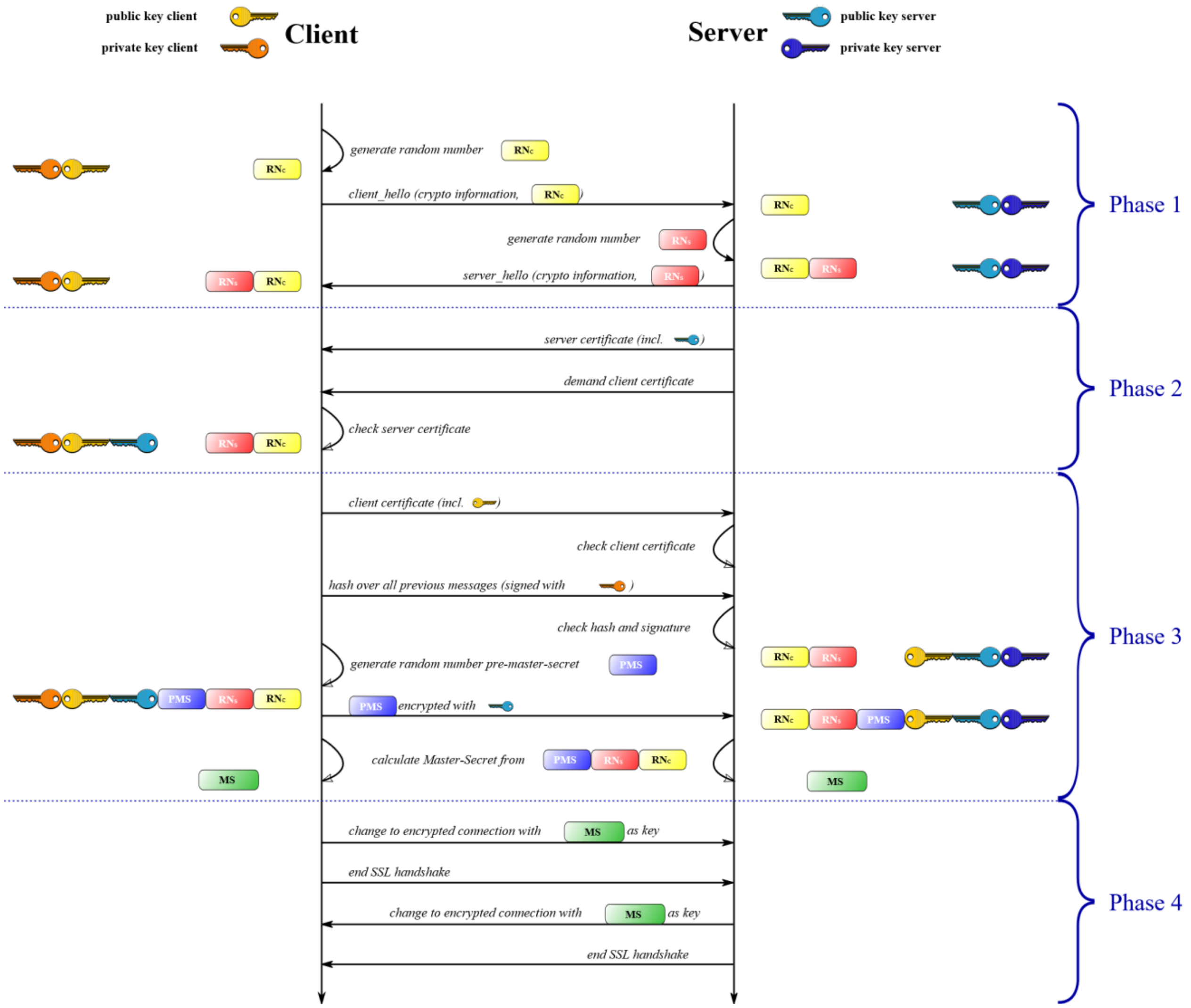
  Client sends a secret master key encrypted with Server's public key

  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

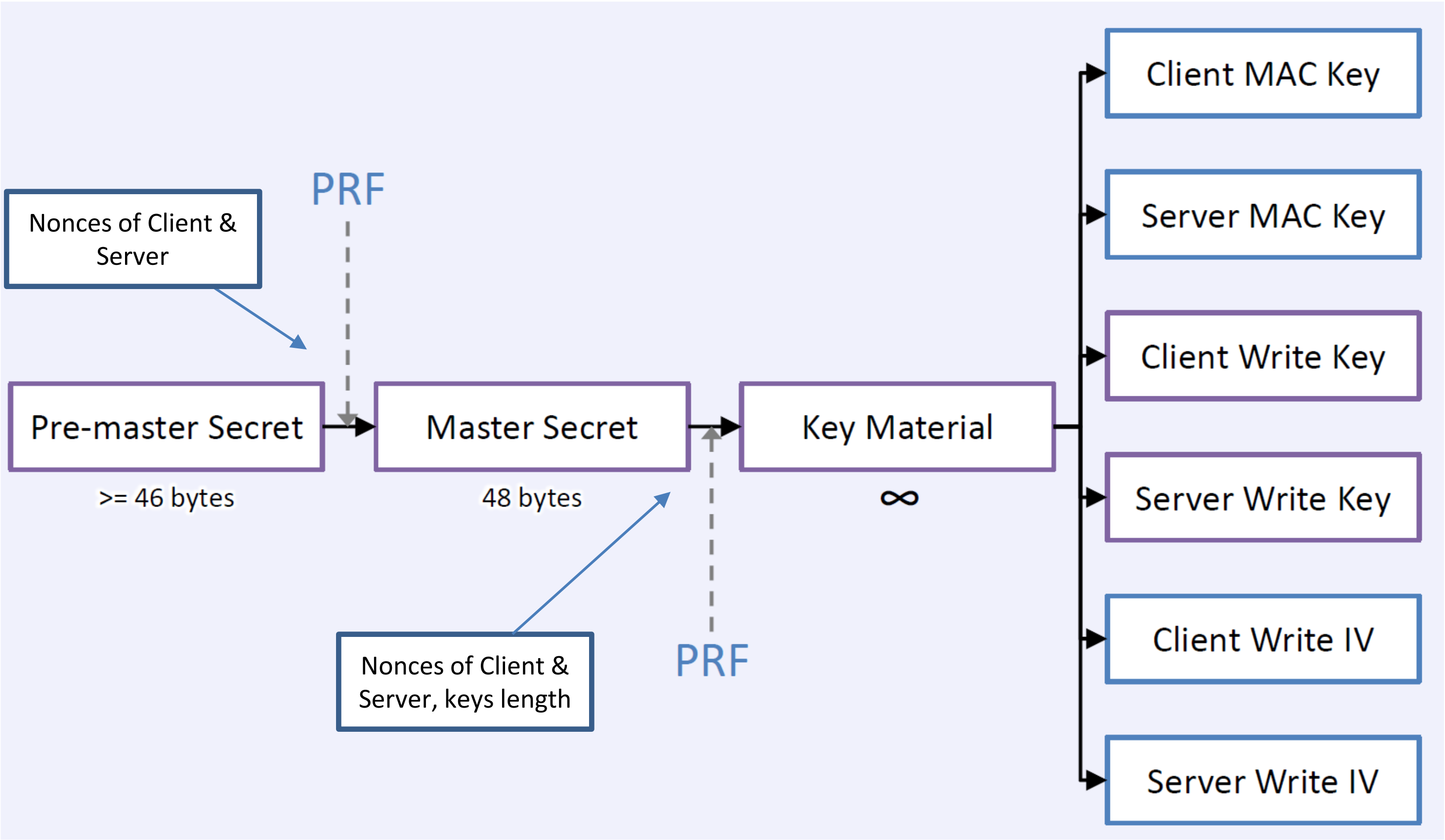


Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

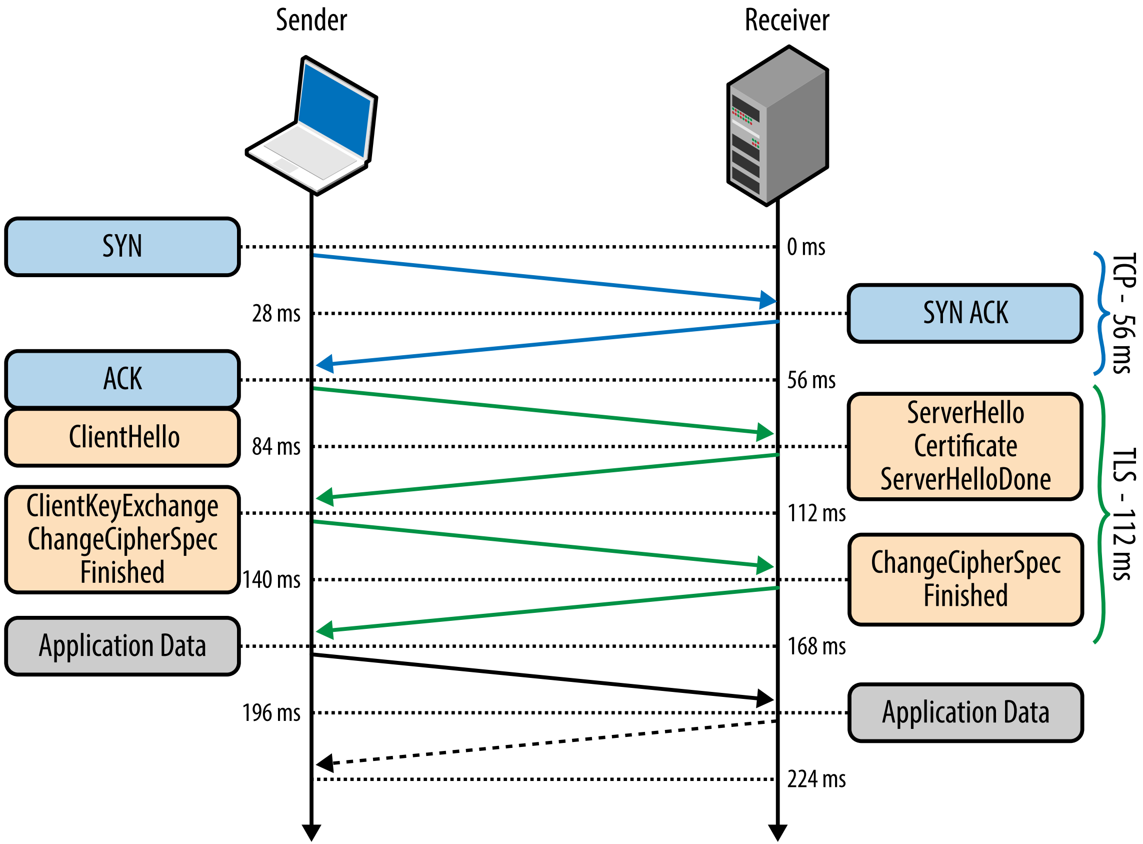


public key client
private key client
Client
Server
public key server
private key server
generate random number
RNc
client_hello (crypto information, RNc)
RNc
generate random number
RNs
server_hello (crypto information, RNs)
RNs
RNc
Phase 1
server certificate (incl.)
demand client certificate
check server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

Key Generation in TLS 1.2



Full TLS 1.2 handshake with timing information

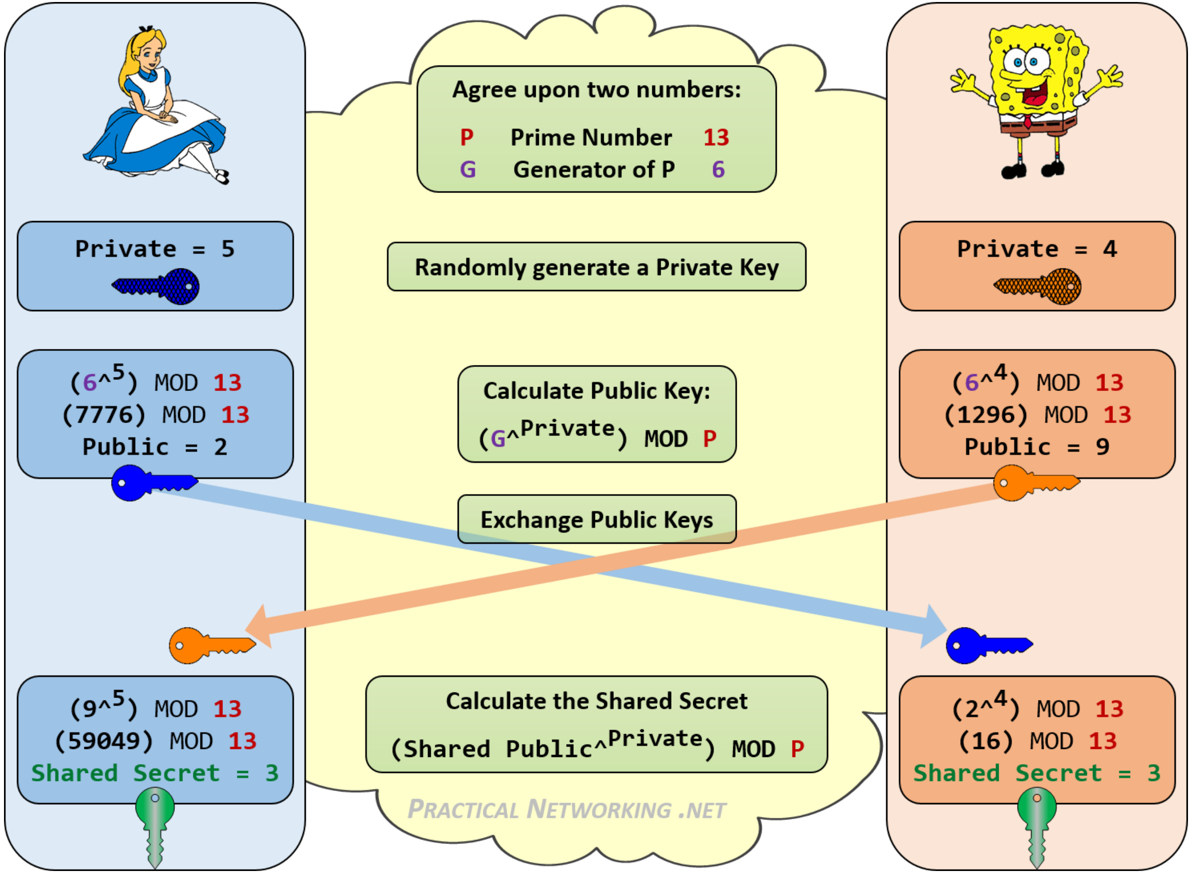


Reference

TLS: Guarding against simple attacks

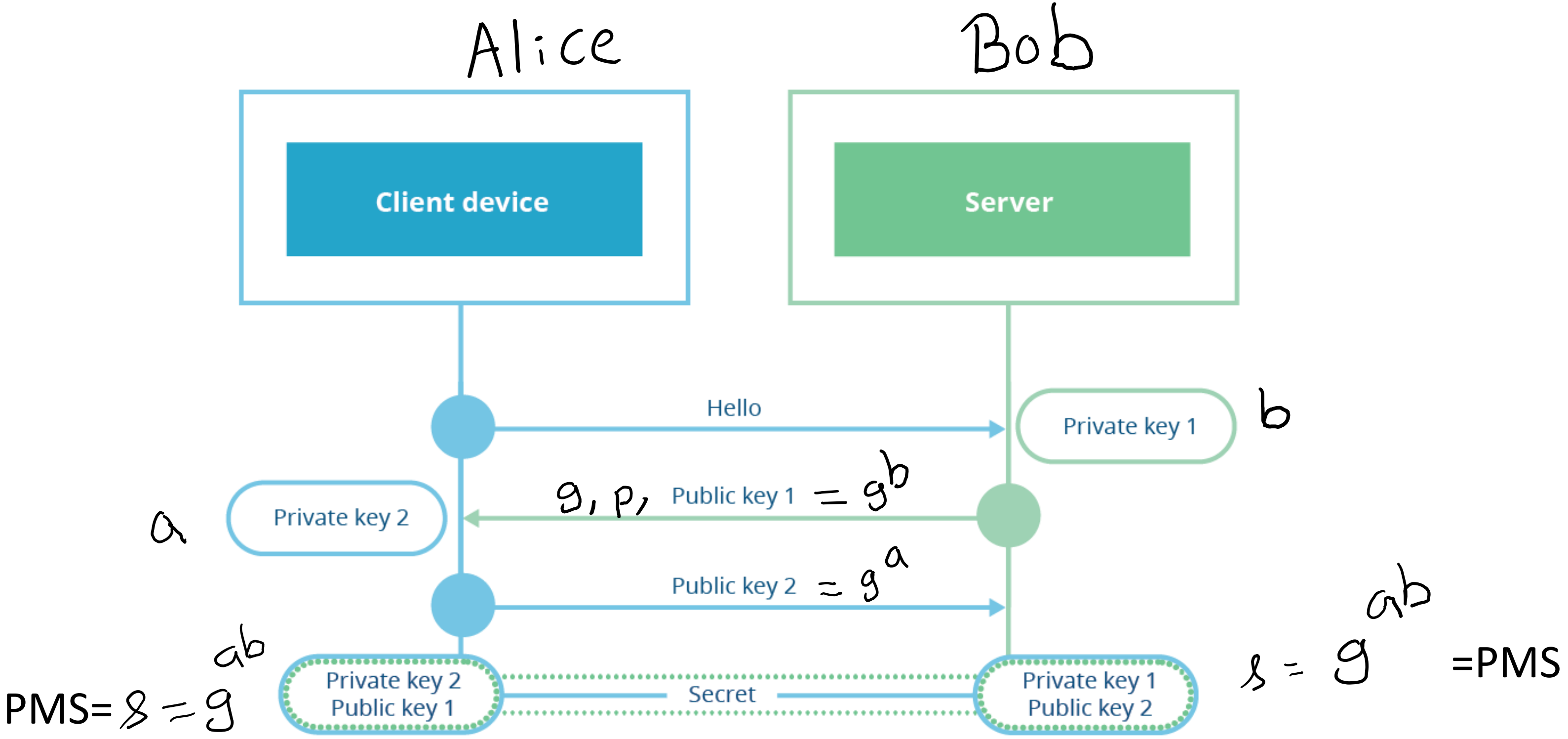
- Role of random numbers (nonces) in TLS handshake
  - Protect against connection/session replay attacks
- Role of sequence numbers in TLS session
  - Different from TCP Sequence Numbers, not added explicitly into Record Protocol Header
  - Protect against segment replay attacks
  - Protect against segment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

Diffie-Hellman Key Exchange



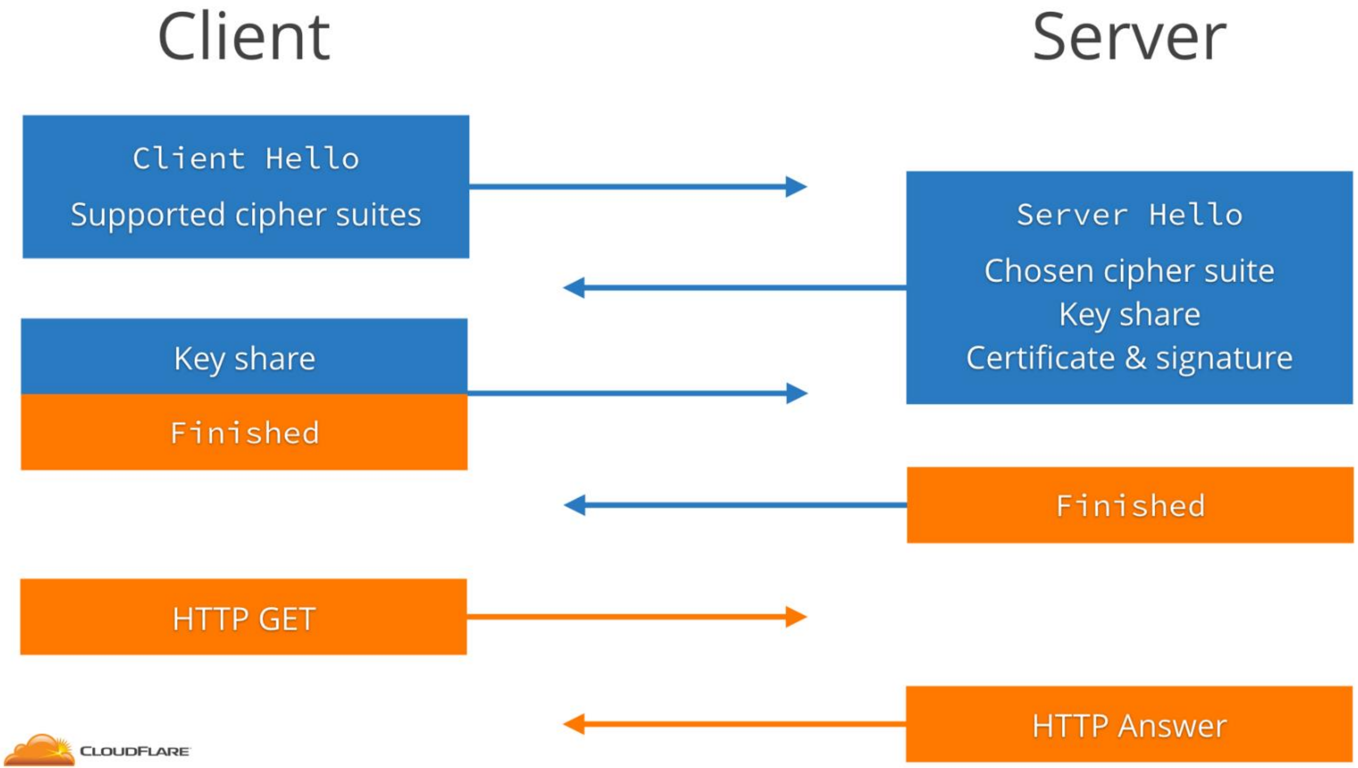
Note: Only a key exchange algo; Can't be useful for Authentication

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
  - 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
……

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

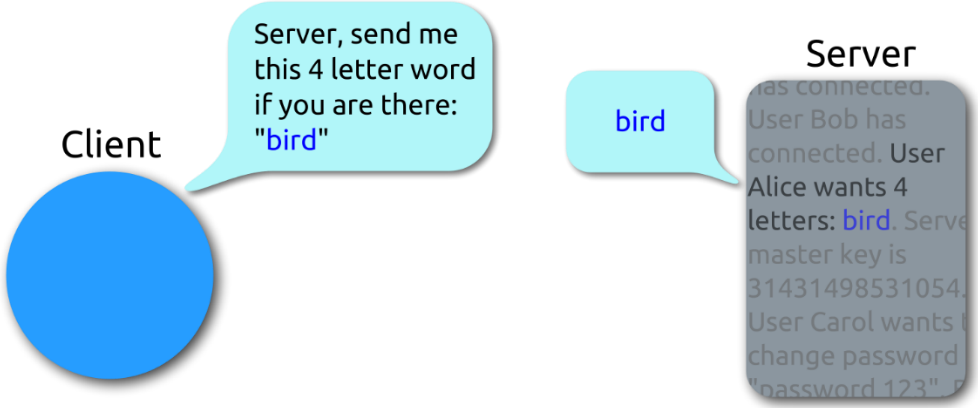
II. Vulnerabilities due to using weak crypto primitives
   - Block ciphers that operate in CBC mode
     - Sweet32, ROBOT, Lucky13

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

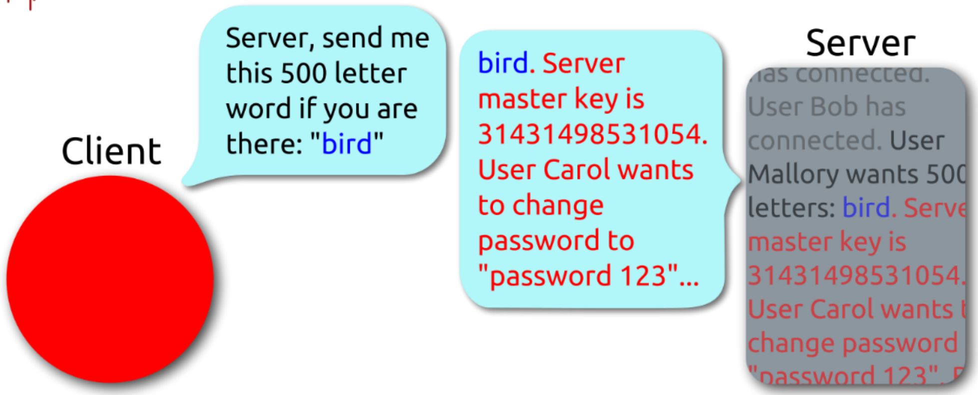
SSL/TLS Attacks (in detail)

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

Heartbeat – Normal usage



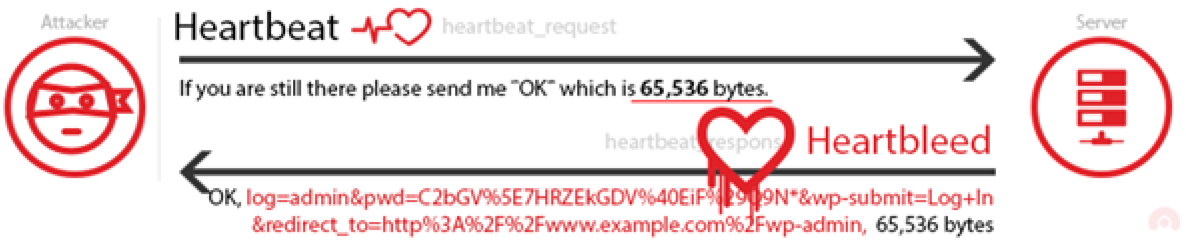
Heartbeat – Malicious usage



Credit: Fenix Feather

- Purpose: Proving to other party that connection is still alive by sending keep-alive messages
- 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



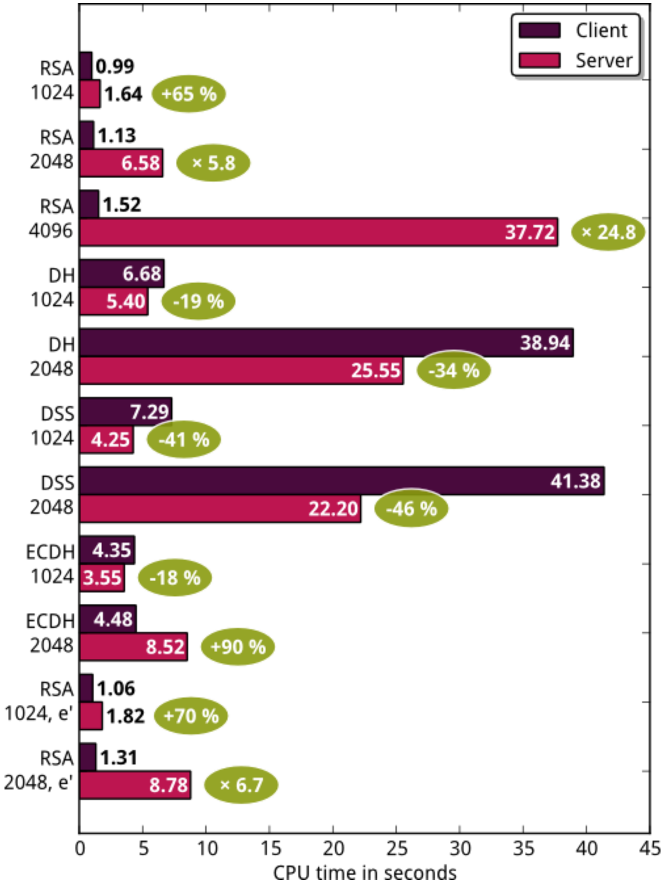
The coding mistake in OpenSSL that caused Heartbleed!

memcpy(bp, pl, payload);

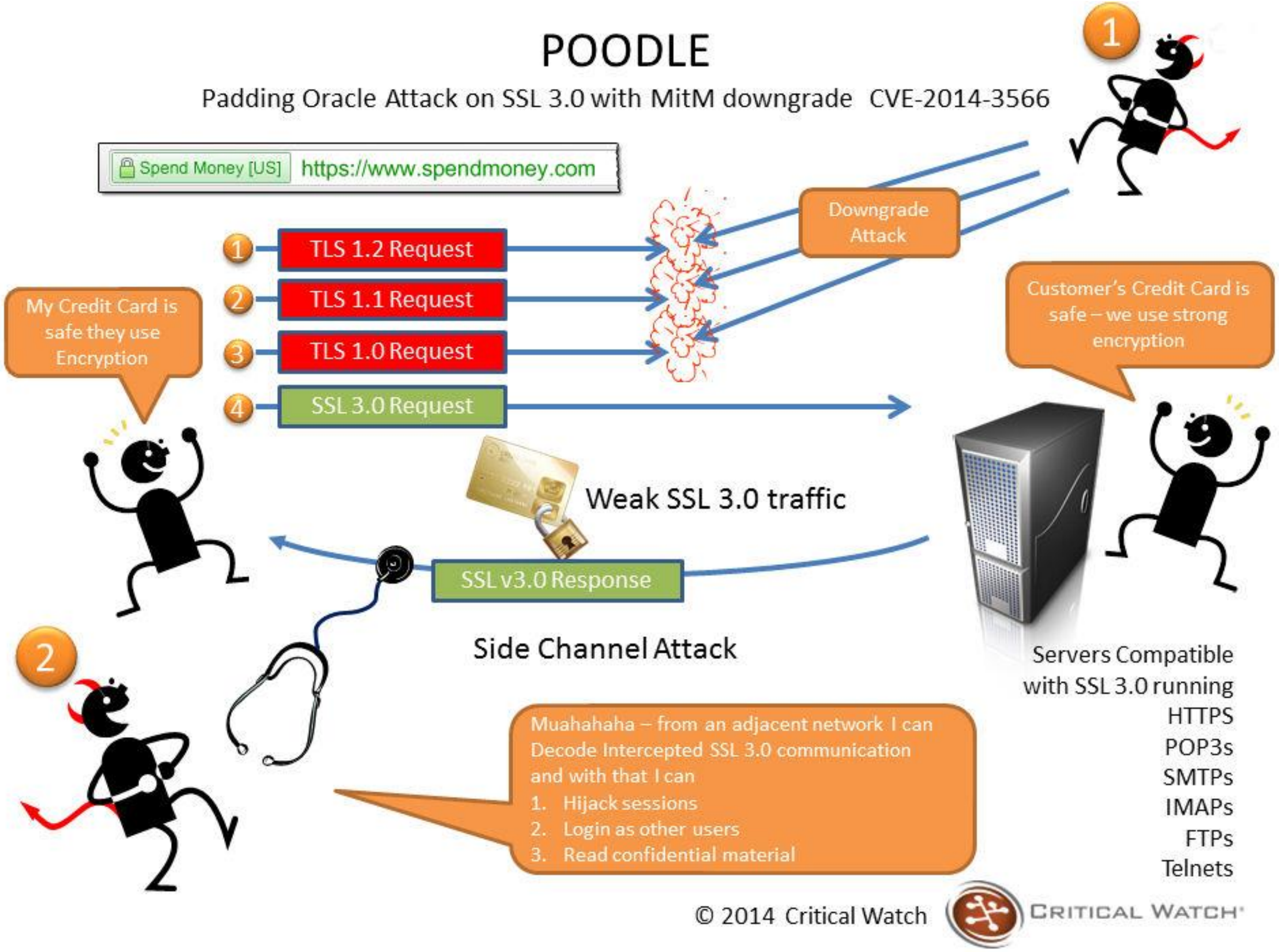
Prevention: Update to the latest version of OpenSSL and if that is not possible, recompile the already installed version with -DOPENSSL_NO_HEARTBEATS

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



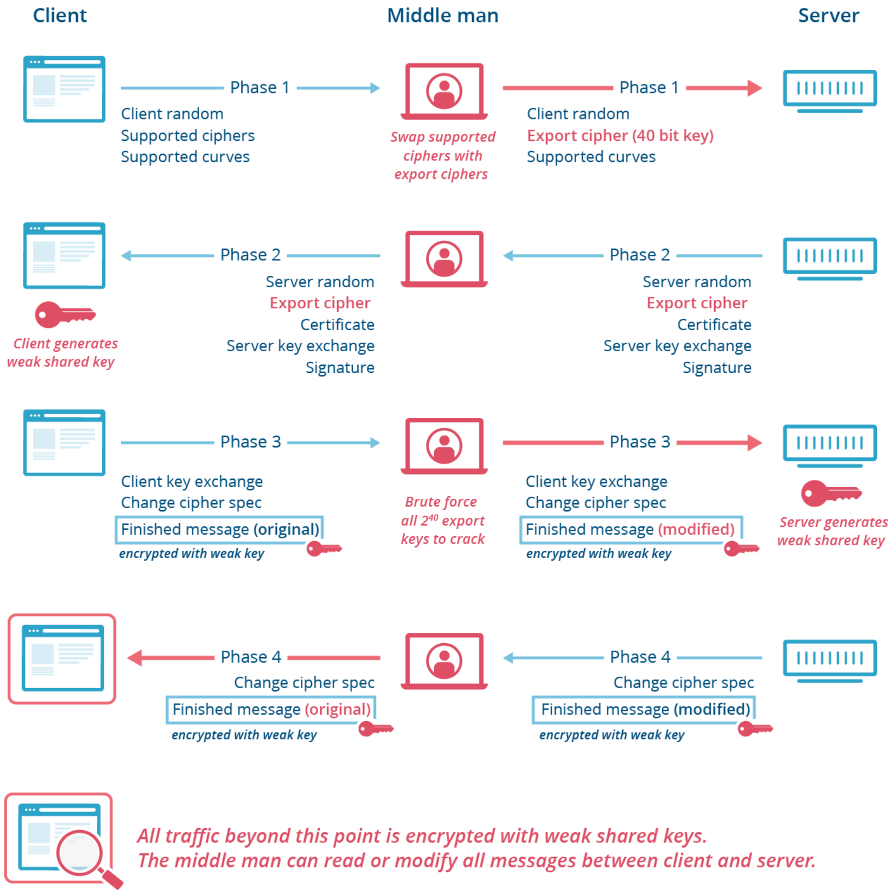
Src: Link



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

- FREAK, LogJam & CurveSwap attacks took advantage of two things:
  - Support for weak ciphers in TLS 1.2
  - Part of handshake which is used to negotiate which cipher to use is not signed

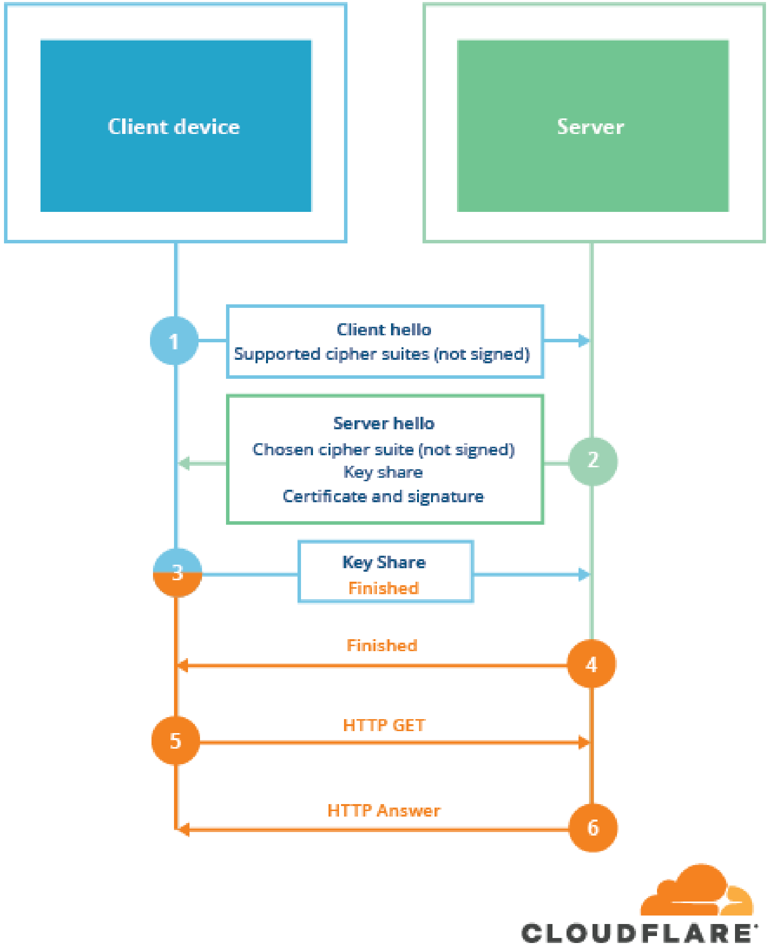
Downgrade Attack (FREAK)



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]

CLOUDFLARE

TLS 1.2 ECDHE



Client device
Server
1
Client hello
Supported cipher suites (not signed)
Server hello
Chosen cipher suite (not signed)
Key share
Certificate and signature
2
3
Key Share
Finished
Finished
4
5
HTTP GET
HTTP Answer
6
CLOUDFLARE

References

- 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/
- 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/