State of TLS usage current and future

Dave Thompson

TLS Client/Server surveys

Balancing backward compatibility with security.

As new vulnerabilities are discovered, when can we shutdown less secure TLS server options without losing customer?

Turning off SSLv3

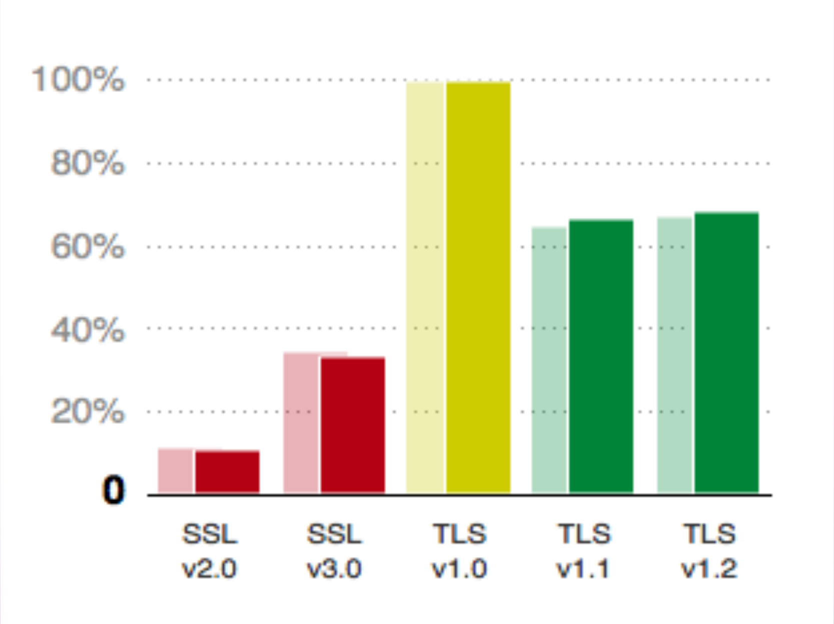
- Released in 1996, obsoleted in 1999 by TLSv1.0

Why should you care?

- Handshake is not protected from MITM
  - Precursor to POODLE attacks
  - Precursor to protocol downgrade attacks
- MAC tied to deprecated MD5 and SHA1
- No TLS-extensions (e.g. No TLS Session-Tickets; No ALPN/NPN/HTTP2/SPDY, No EC specs, No GCM ciphers, No SNI (Server Name Indicator)

SSLv3 Usage

- Server survey 31.2% servers supported (November 2015) of top 200k Alexa list. *



- Yahoo client usage survey in October 2015 showed <.01% clients connect with SSLv3.

* https://www.trustworthyinternet.org/ssl-pulse/

RC4 Usage

- Most common 128-bit stream cipher used throughout 90's and 2000's.
- Software performance is fast.
- Not vulnerable to block padding, CBC and timing attacks e.g. POODLE (2014), BEAST (2011), Lucky13 (2013)

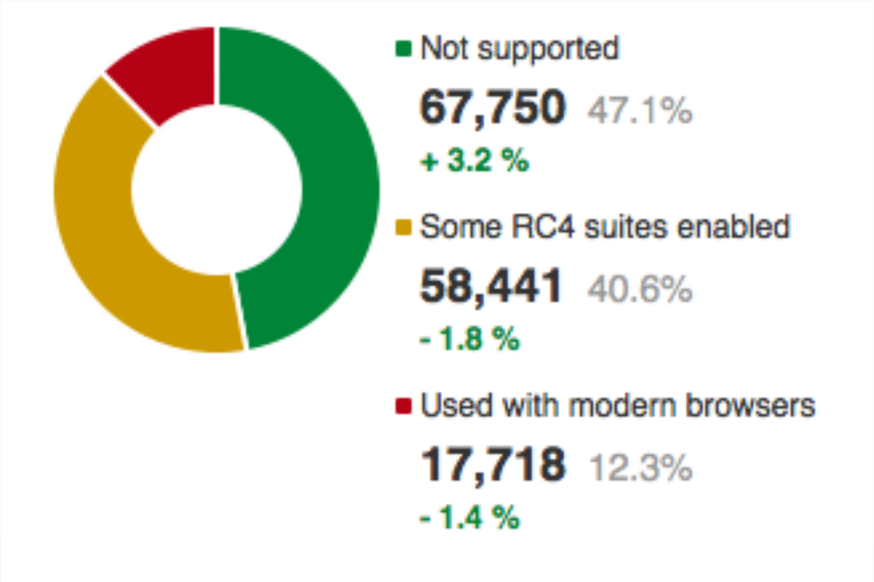
RC4 Usage

Why should you care?

- Numerous key stream bias attacks
  - Rolland Holloway attack (3/2013) – reduces effectiveness to 2^24.
  - Bar Mitzvah attack (2015)

RC4 Usage

Server acceptance survey (October 2015) of top Alexa 200k.



https://www.ssllabs.com/ssltest/clients.html

H Kario top 500k July survey similar results

RC4 Usage

Global Yahoo client cipher suite usage survey (November 4, 2015)   **< .01%** required RC4

(End of list ECDHE-RSA-RC4-SHA, RC4-SHA)

```
Preferred order list:
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-SHA384
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES256-SHA
AES128-GCM-SHA256
AES256-GCM-SHA384
AES128-SHA256
AES256-SHA256
AES128-SHA
AES256-SHA
DES-CBC3-SHA
ECDHE-RSA-RC4-SHA
RC4-SHA
```

PFS Key Exchange

Perfect Forward Secrecy – After temporal session keys are destroyed by peers, the ability to decrypt cipher stream is lost.

- DHE and ECDHE are examples of possible PFS key exchanges.
- RSA key exchange is not, as recovery of private key unravels all data (past, current, and future) that rely on it.

PFS Key Exchange

Why should you care?

In RSA KE, recorded cipher streams are decrypted should private key be discovered.

- Most servers have private key in file system. **Compromise of one server can mean compromise of all past, current and future traffic from pool that shares same certificate**.
- Heartbleed exploit (April 2014) – attacker can send a malformed DTLS packet to server, and receive up to 64kB chunks of server memory. Full private key extraction demonstrated in under 8 hours.

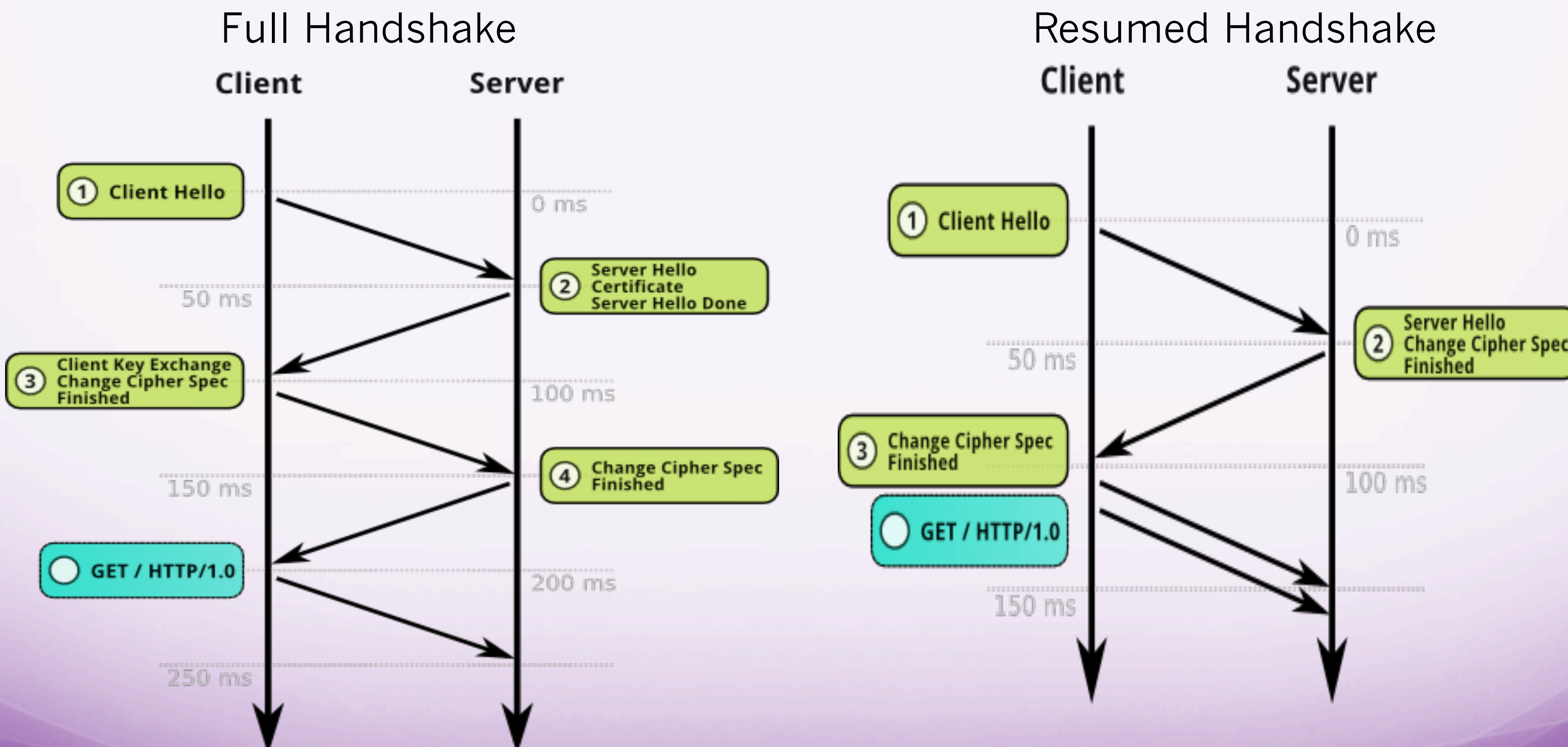
PFS Key Exchange

- ECDHE – introduced in 2008 with TLS 1.2

Presented clients with priority ordered cipher list with ECDHE first. Yahoo global client survey (November 2015), shows 91-97% of clients (depending on region) are ECDHE cipher capable.

```
Preferred order list:
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-SHA384
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES256-SHA
AES128-GCM-SHA256
AES256-GCM-SHA384
AES128-SHA256
AES256-SHA256
AES128-SHA
AES256-SHA
DES-CBC3-SHA
ECDHE-RSA-RC4-SHA
RC4-SHA
```

TLS Session Resumption



Diff: Full network round trip time savings + authentication and key exchange

SSL Session-ID

- Initial method dating back to SSLv2 (1995)
- Session-ID's require caching of negotiated handshake parameters by both client and server.
- Can be a problem for load balanced server deployments with no source hash routing. In between connects, server must share negotiated credentials with other servers of cluster before client reconnect.

New: TLS Session-Tickets

- Introduced in 2008
- Negotiated handshake parameters stored in client presented session-ticket.
- No caching required for server.
- No sharing required amongst server pool, of client's session parameters.
- Ideal for multi-node server installs.

TLS Session-Tickets

- Session tickets have priority in protocol.
  - If both session ticket and session-id presented, session-ticket is used.

- All common current browsers support TLS session tickets except Safari (iOS and OSX)
  - Chrome, Firefox, Android, Baidu, OpenSSL, IE (since IE11/Win 8.1)

https://www.ssllabs.com/ssltest/clients.html

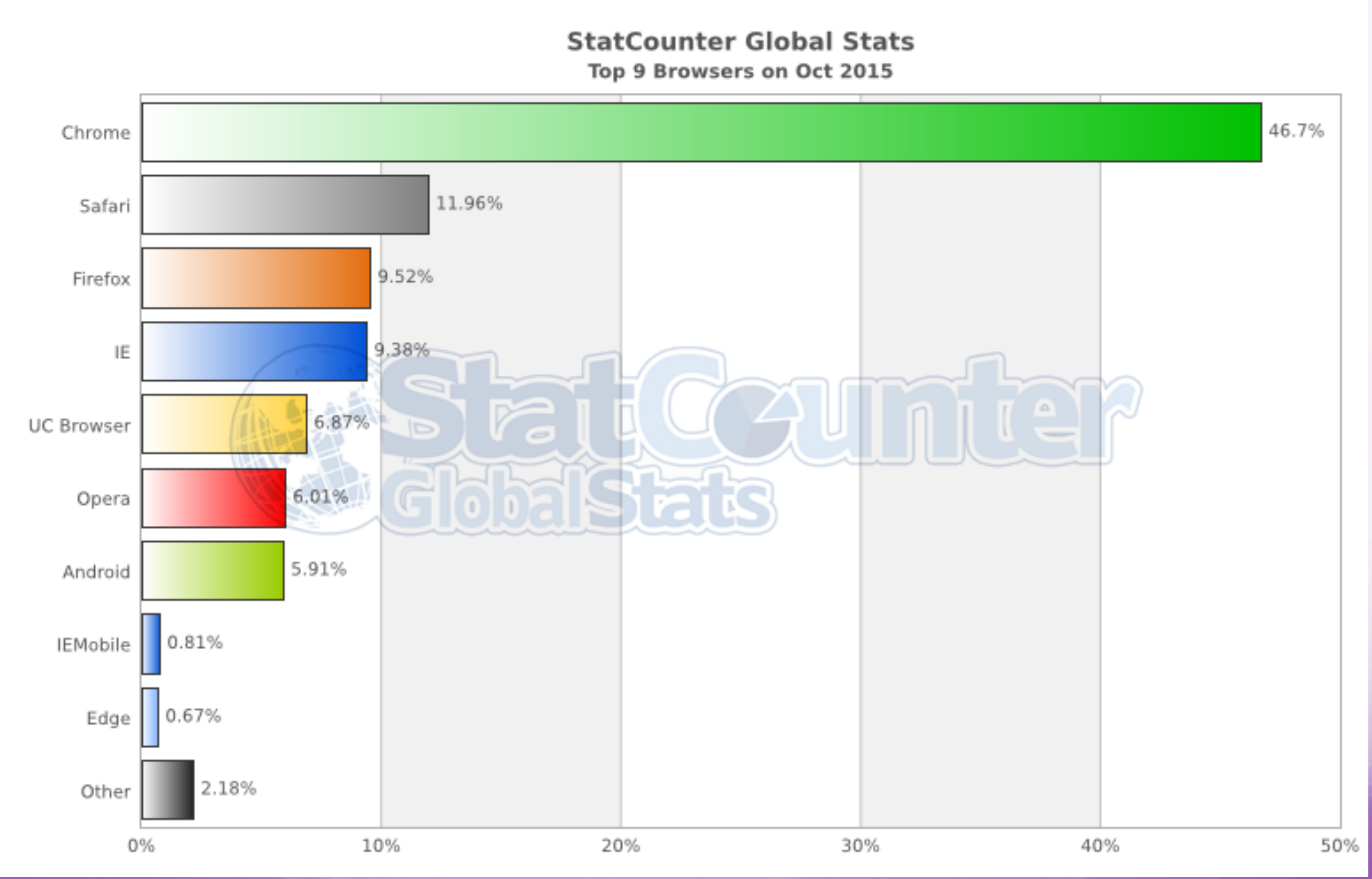
TLS Session-Tickets

- Client indicates session ticket capability in client hello.
- ATS's traffic_line metrics

Approximate number TLS-session-ticket capable clients = total_tickets_created/total_success_handshake_count_in

Yahoo survey, 51% clients TLS-session-ticket capable, *though this survey likely skewed negatively by disproportionate safari clients (vs Chrome) to yahoo*.

Browser Usage Distribution



http://gs.statcounter.com/#all-browser-ww-monthly-201510-201510-bar

Certificates

Primary role in TLS is to authenticate peer. Though public key may additionally be used for session key exchange. Certificates are signed by a peer trusted third party.

rsaSHA-1 signed cert – standard in 90's and 200X.

1024-bit standard in 90's, 2048-bit 200X.

Certificates Issue Categories:

1. SHA-1 vs SHA-256 (security issue) *
2. 1024-bit modulus vs 2048 vs 4096 (security vs performance). In TLS, impacts security of session key exchange. *
3. SHA-256 vs ECDSA (performance vs acceptability)
4. PKCS1 vs PSS (performance vs improved security)

Certificates
SHA-1 vs SHA256

Why should you care?

1) As of Oct 2015, 57-bits demonstrated security strength against collisions with SHA-1.*

Approximately $2k rent time on EC2 to find collision ($75k-120K for full collision map) *

2) Deprecation of SHA-1 encouraged by Google search ranking, Chrome browser shaming, Apple App Transport Security blocking, and others.

* Stevens, Karpman, Peyrin

Certificates
SHA-1 vs SHA256: server

Certificate signature, server deployment Alexa top 500k:

- sha1WithRSAEncryption 29.4%
- sha256WithRSAEncryption 63.9%
- ecdsa-with-SHA256 6.7%

H. Kario -https://securitypitfalls.wordpress.com/2015/07/29/july-2015-scan-results/

Certificates
SHA-1 vs SHA256: client

Client acceptance of SHA256 signed certificate

- August 2015, measured < 0.189% clients did **not** accept sha256WithRSAEncryption certificates.

Source: Yahoo YCPI survey

Certificates
Modulus Strength

Server deployment (Alexa top 200k):

<0.1% Below 2048-bit modulus
94.4% 2048-bit (or equiv e.g. ECDSA 256-bit)
1.4% 3072-bit
4.1% 4096-bit.

PQC may start to shift this to 3072 and 4096.

November 2015– Alexa top 200k- https://www.trustworthyinternet.org/ssl-pulse/

Future TLS

IETF94 TLS working group

- TLSv1.3
- Discussed change PKCS1 cert signing to PSS
- Cipher suite specification including new curves 25519 (fast) and 442 (strong)
- Re-keying (applicable to large data using AES-GCM, ChaCha20)
- HKDF – defining HMAC Key Derivation Functions

TLS 1.3

- Currently most significant change ever to SSL protocol. TLS 1.2 is far more similar to SSLv2, than TLS 1.3

- Key portions are currently being worked out (state flow, security structures, even TLS record layer re-arrangement)

A flavor of TLS 1.3

- 0-RTT, 1-RTT Handshake, leveraging QUIC-crypto and will ultimately replace.
- Use of short life PSK for resumption (0-RTT)
- Cipher suite changes (prohibit RC4, deprecate camellia, others)
- Record layer changes (drop version, possibly reorder)
- Move to HMAC Key Derivation Functions (HKDF)
- Remove ChangeCipherSpec
- Removed renegotiation, though may be back in another form (HelloRetryRequest) for re-key cipher exhaustion)
- Remove GMT time from peer random

A flavor of TLS 1.3

- Remove support for compression
- Remove static RSA key exchange
- Remove support for non-AEAD ciphers (Authentication Encryption Associated Data)
- Introduce new curve 22519 (fast), curve 448 (strong)
- Considering move of RSA certificate signatures from PKCS1 to PSS
- Specification of certificate acceptance criteria, rather than peer guessing (e.g. rsaSHA1, rsaSHA256, PKCS1, PSS, ECDSA).
- Possible interest in encrypted SNI, though very preliminary.

TLS 1.3 handshake 11/3/15

```
            ClientHello
              + ClientKeyShare
        ^     + EarlyDataIndication
 0-RTT  |   (Certificate*)
 mode   |   (CertificateVerify*
        v   (Finished)  // Note: new message.
            (Application Data*)        -------->
                                                        ServerHello
                                                    ServerKeyShare*
                                              {EncryptedExtensions}
                                              {CertificateRequest*}
                                             {ServerConfiguration*}
                                                     {Certificate*}  ^
                                               {CertificateVerify*}  | Server Auth.
                                       <--------         {Finished}  v
1-RTT   ^   {Certificate*}
Client  |   {CertificateVerify*}
Auth    |   {Finished}                 -------->
        v   [Application Data]         <------->   [Application Data]


                                       <--------  [CertificateRequest]   ^
            [Certificate]                                                | Post-HS
            [CertificateVerify]                                          | Auth.
            [Finished]                 -------->                         v
```

Note: As of 11/4/15, this is out of date.

TLS 1.3

- First connect is 1-RTT (since ietf92)
- Resumption is 0-RTT, using temporal PSK for resumption (ietf93)
- Client side-authentication for 1.3 fleshed out at (ietf94), HelloRetryRequest, Re-Keying (to support large data sets over AES-GCM, or ChaCha20

TLS 1.3 API impact

Possible TLS API support for version 1.3 support:

- With 0-RTT up to 8k data is carried on first (TLS connect) flight.
- 1-RTT vs 0-RTT will likely be abstracted as 1-RTT is fallback for failed 0-RTT, in which case ~8k data buffered during 1-RTT handshake with async operations.

ChaCha20+Poly1305

As of May 2015 adopted as RFC7539

- ChaCha20 is a 256-bit stream cipher
- Poly1305 is a message authenticator
- Considered replacement stream cipher for now deprecated 128-bit RC4.

ChaCha20+Poly1305

- Currently deployed and supported by Google Servers and in Chromium
- Patch available for NSS (Firefox) and OpenSSL

TLS cipher suites:

TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256

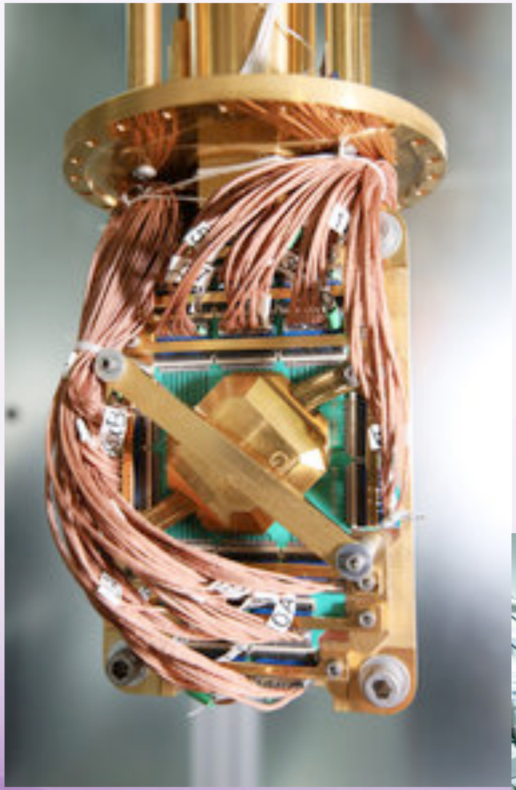
ChaCha20+Poly1305

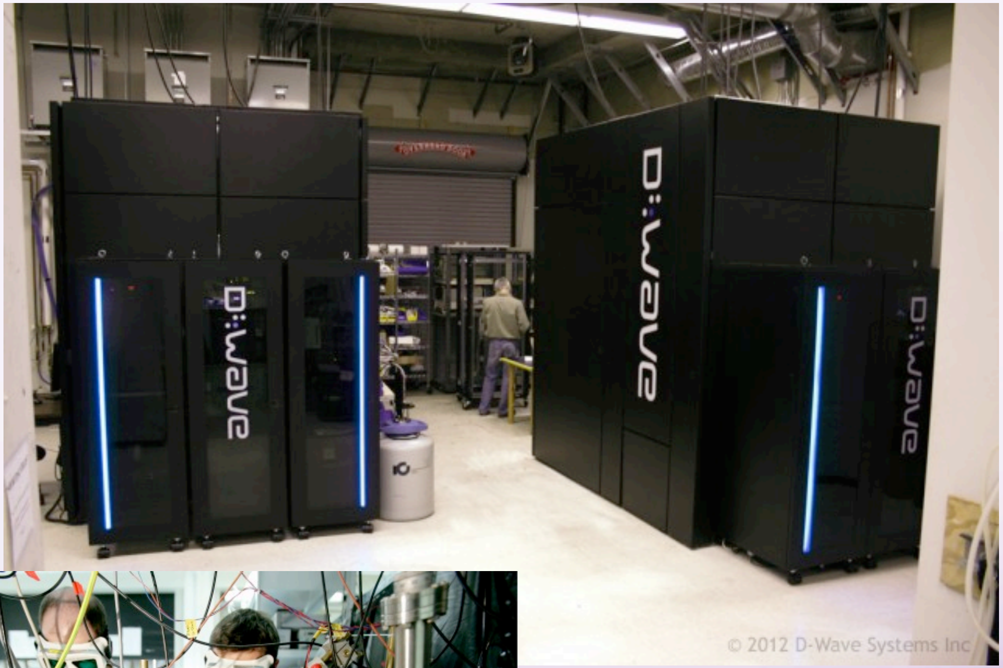
Performance:

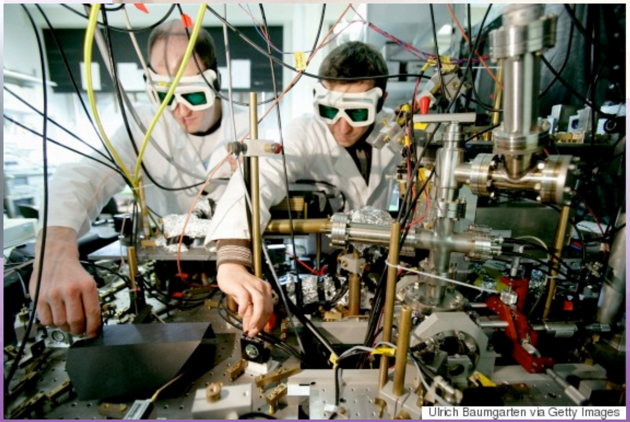
Chip	AES-128-GCM	ChaCha20-Poly1305
OMAP 4460	24.1 MB/s	75.3 MB/s
Snapdragon S4 Pro	41.5 MB/s	130.9 MB/s
Sandy Bridge Xeon (AES-NI)	900 MB/s	500 MB/s

Measurements by Adam Langley, published in RFC7539, Appendix B

Impending Quantum CRYPTOPOCALYPSE







Impending Quantum CRYPTOPOCALYPSE

- Problem: Quantum computer make it trivial to break RSA, ECC, DH.
  - Current TLS traffic is susceptible to a harvest-then-decrypt attack from passive attacker

- Best quantum algorithms (conjecture) put risk as follows:
  - AES – brute force n-bit key search effectively reduces to (n/2) key-bit strength (Grover's algorithm).
  - RSA – Time required to break is same time as RSA encrypting (Shor's algorithm)

Impending Quantum CRYPTOPOCALYPSE

Post Quantum Cryptography

- August 2015, NSA announced a deprecation of transition to Suite B cryptography and instead begin focusing on quantum resistant attacks.
- Suite B cryptography includes: AES, ECDH, ECDSA, SHA2 (SHA-256, SHA-384)
- Quantum resistant cryptography suite not yet announced. NSA says It's coming.

https://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

Quantum Attack Resistance

If fastest quantum attack known for symmetric-key encryption recovering a k-bit secret key takes 2^k/2…

What to do?

Quantum Attack Resistance

If fastest quantum attack known for symmetric-key encryption recovering a k-bit secret key takes 2^k/2…

What to do?

Double the key strength.

To have AES128 bit level security post quantum, switch to AES256 now.

Quantum Resistance Interim Prep:

NSA Guidelines (8/2015):

- Block cipher: Use 256-bit AES
- ECDH: use curve P-384
- ECDSA: use curve P-384
- SHA: SHA-384
- Diffie-Hellman key exchange: min 3072-bit modulus
- RSA Key exchange: min 3072-bit modulus
- RSA signature: min 3072-bit modulus

https://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

Quantum Resistance Interim Prep for TLS:

- TLS cipher suites:
  - TLS_RSA_WITH_AES_256_GCM_SHA384
  - TLS_DH(E)_RSA_WITH_AES_256_GCM_SHA384
  - TLS_ECDH(E)_RSA_WITH_AES_256_GCM_SHA384

Questions