

Introducing TLS

Kenny Paterson
@kennyog

Information Security Group



ROYAL
HOLLOWAY
UNIVERSITY
OF LONDON

CWTLS1.3@CRYPTO2018

Agenda

- An overview of SSL/TLS
- Results from an SSL/TLS measurement study
- Concluding remarks

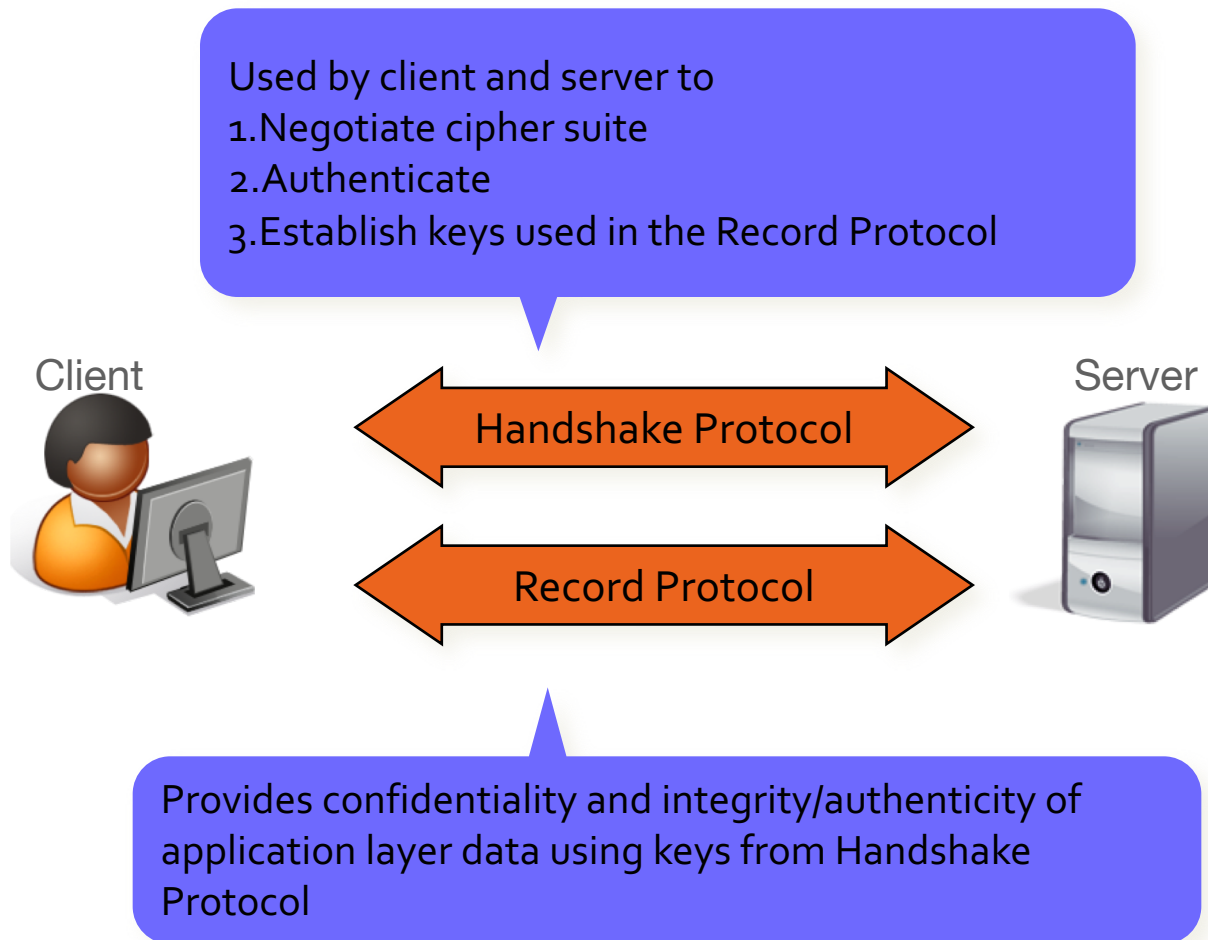
SSL/TLS Overview

- SSL = Secure Sockets Layer.
 - Developed by Netscape in mid 1990s.
 - SSLv1 broken at birth.
 - SSLv2 seriously flawed in various ways, now IETF-deprecated (RFC 6176).
 - SSLv3 now considered broken (POODLE + RC4 attacks), but still widely supported.
- TLS = Transport Layer Security.
 - IETF-standardised version of SSL.
 - TLS 1.0 in RFC 2246 (1999).
 - TLS 1.1 in RFC 4346 (2006).
 - TLS 1.2 in RFC 5246 (2008).
 - TLS 1.3 in RFC 8446 (2018).

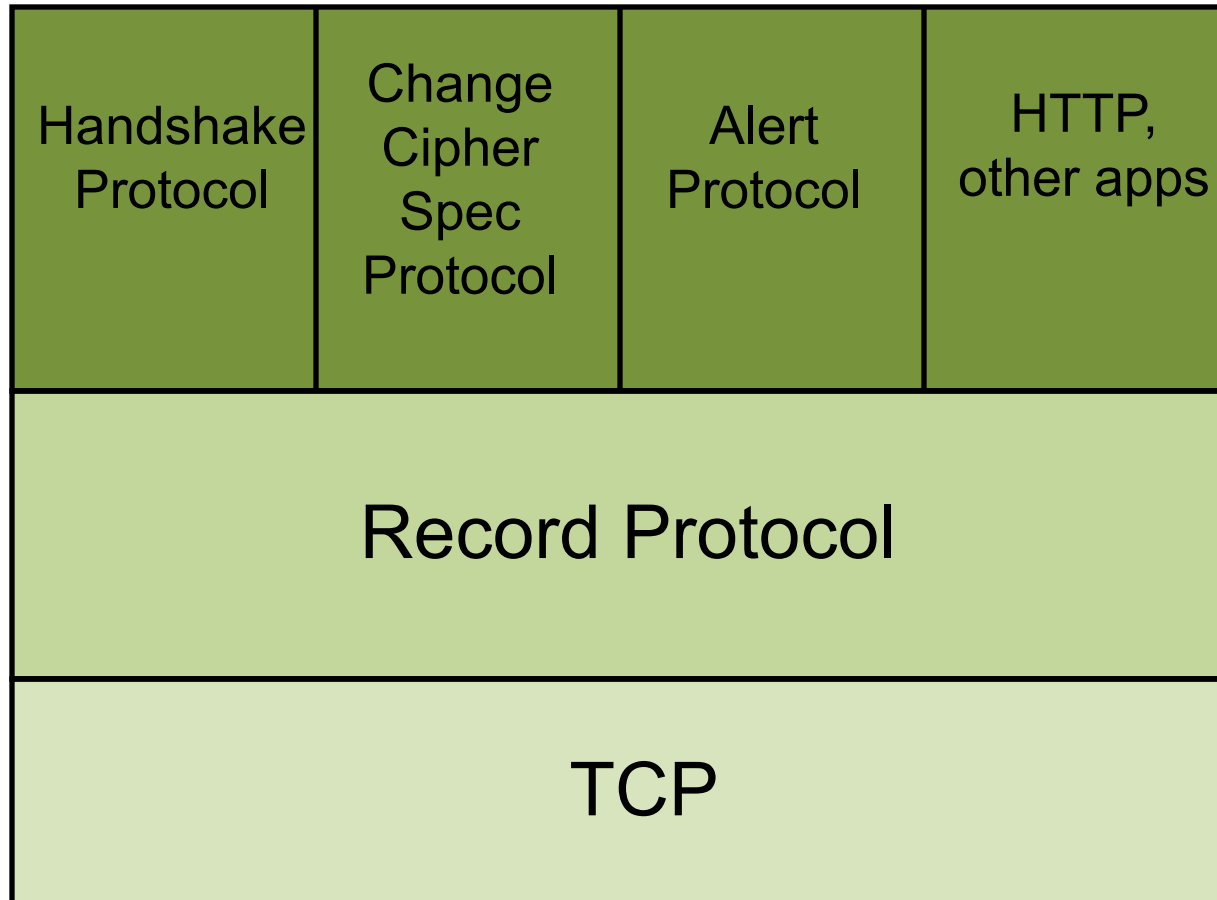
Importance of TLS

- Originally designed for secure e-commerce over http, now used much more widely.
 - Retail customer access to online banking facilities.
 - Access to gmail, facebook, Yahoo, etc.
 - **>70% of web traffic is now encrypted using TLS.**
 - Mobile applications, including but not only banking apps.
 - Payment infrastructures.
 - Back-end operations of large organisations, e.g. AWS.
- TLS has become the *de facto* secure protocol of choice.
 - Used by hundreds of millions/billions of people and devices every day.

Highly Simplified View of TLS



TLS Protocol Architecture



The TLS Ecosystem (1/3)

- Servers
 - Including managed service providers (Cloudflare, Akamai,...).
- Clients
 - Of all shapes and sizes.
 - Web browsers to embedded devices.
- Certification Authorities (CAs)
 - Of all shapes , sizes and levels of security.
 - Typically 300 root CA keys in browser.
- Implementations
 - From Google (BoringSSL), Facebook, AWS down to one-person open-source operations.
 - OpenSSL somewhere in-between, once used by 80- 90% of web servers.
- Hardware vendors

The TLS Ecosystem (2/3)

- TLS versions:
 - SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2, TLS 1.3.
 - Some servers even still support SSL 2.0 (!)
- 200+ cipher suites
 - Some very common, e.g.
 TLS_RSA_WITH_AES_128_CBC_SHA256.
 - Some highly esoteric, e.g.
 TLS_KRB5_WITH_3DES_EDE_CBC_MD5.
 - Some offering no security:
 TLS_NULL_WITH_NULL_NULL !
- TLS extensions
- DTLS

The TLS Ecosystem (3/3)

- IETF TLS Working Group
 - And CFRG (Crypto Forum Research Group)
- TLS research community
 - Finding attacks and building security proofs
 - Analysis of TLS 1.3 during its development
- The TLS ecosystem has become very complex and vibrant.
 - With great industry-academia-IETF interaction during the development of TLS 1.3.

The TLS Ecosystem (4/3) (bonus slide)

OpenSSL Fact @OpenSSLFact · 24 Jul 2013

/*The aim of right-shifting md_size is so that the compiler doesn't figure out that it can remove div_spoiler...which I hope is beyond it.*/

14 7

OpenSSL Fact retweeted



JP Aumasson @veorq · 3 Feb 2013

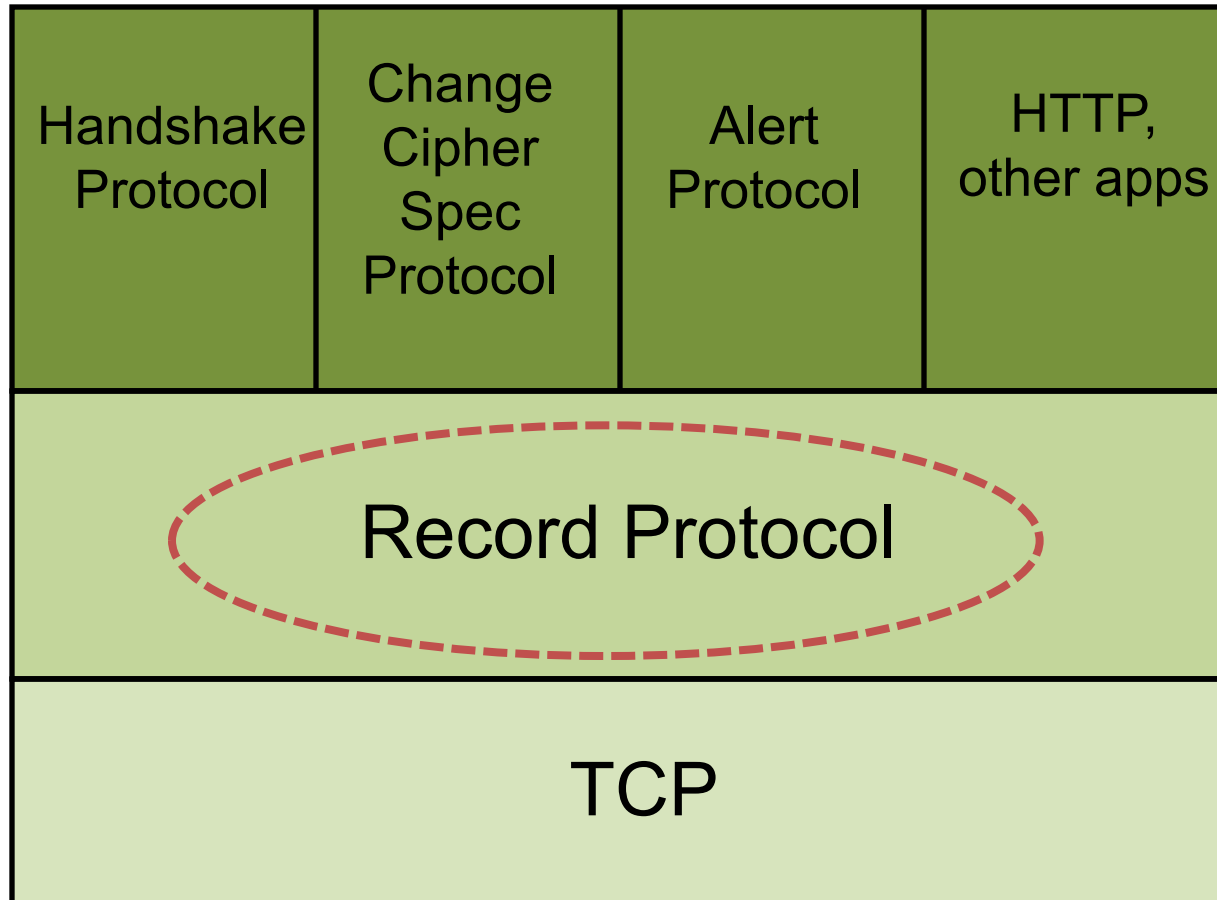
OpenSSL wikibook, nice initiative en.m.wikibooks.org/wiki/OpenSSL

12 7

OpenSSL Fact @OpenSSLFact · 31 Jan 2013

```
#else
    if (0)
        ;
#endif
```

TLS Record Protocol

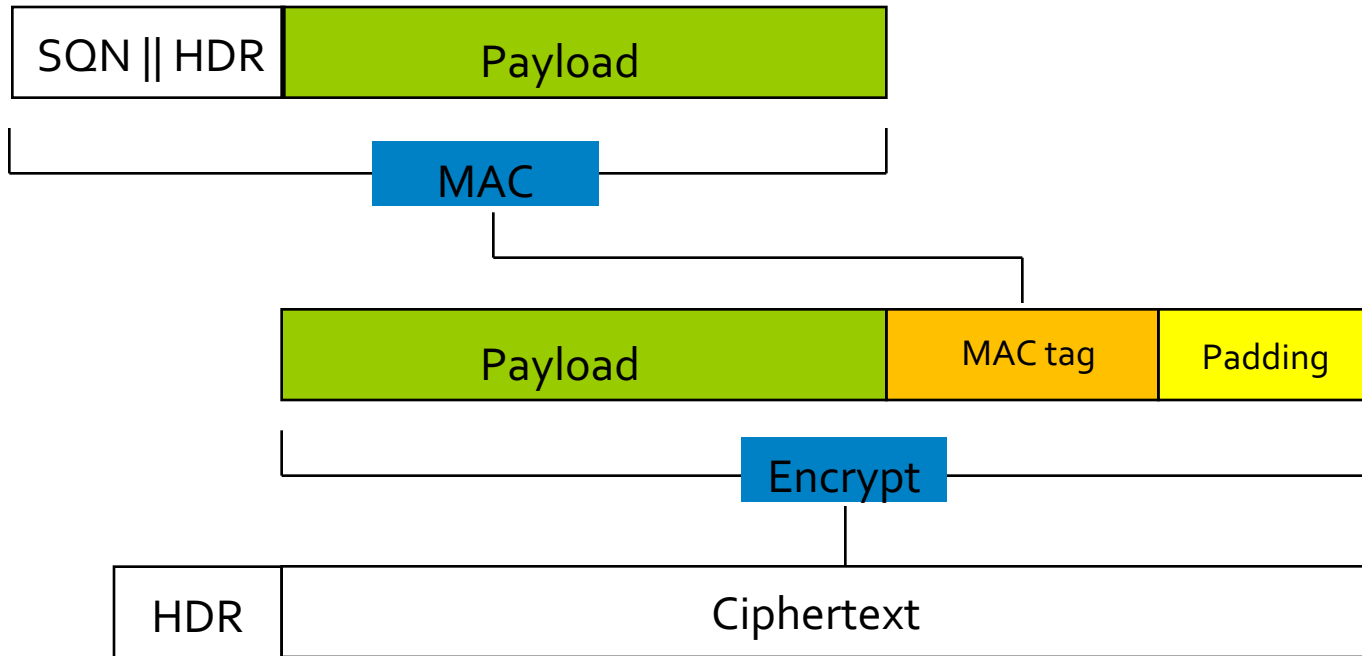


TLS Record Protocol

TLS Record Protocol provides:

- Data origin authentication, integrity using a MAC.
- Confidentiality using a symmetric encryption algorithm.
- Anti-replay using sequence numbers protected by the MAC.
- Optional compression.
- A TCP-like streaming interface for the delivery of higher layer protocol messages, e.g. application data.

TLS Record Protocol Prior to TLS 1.2: MAC-Encode-Encrypt



MAC

HMAC-MD5, HMAC-SHA1, HMAC-SHA256,...

Encrypt

CBC-AES128, CBC-AES256, CBC-3DES, RC4-128,...

Padding

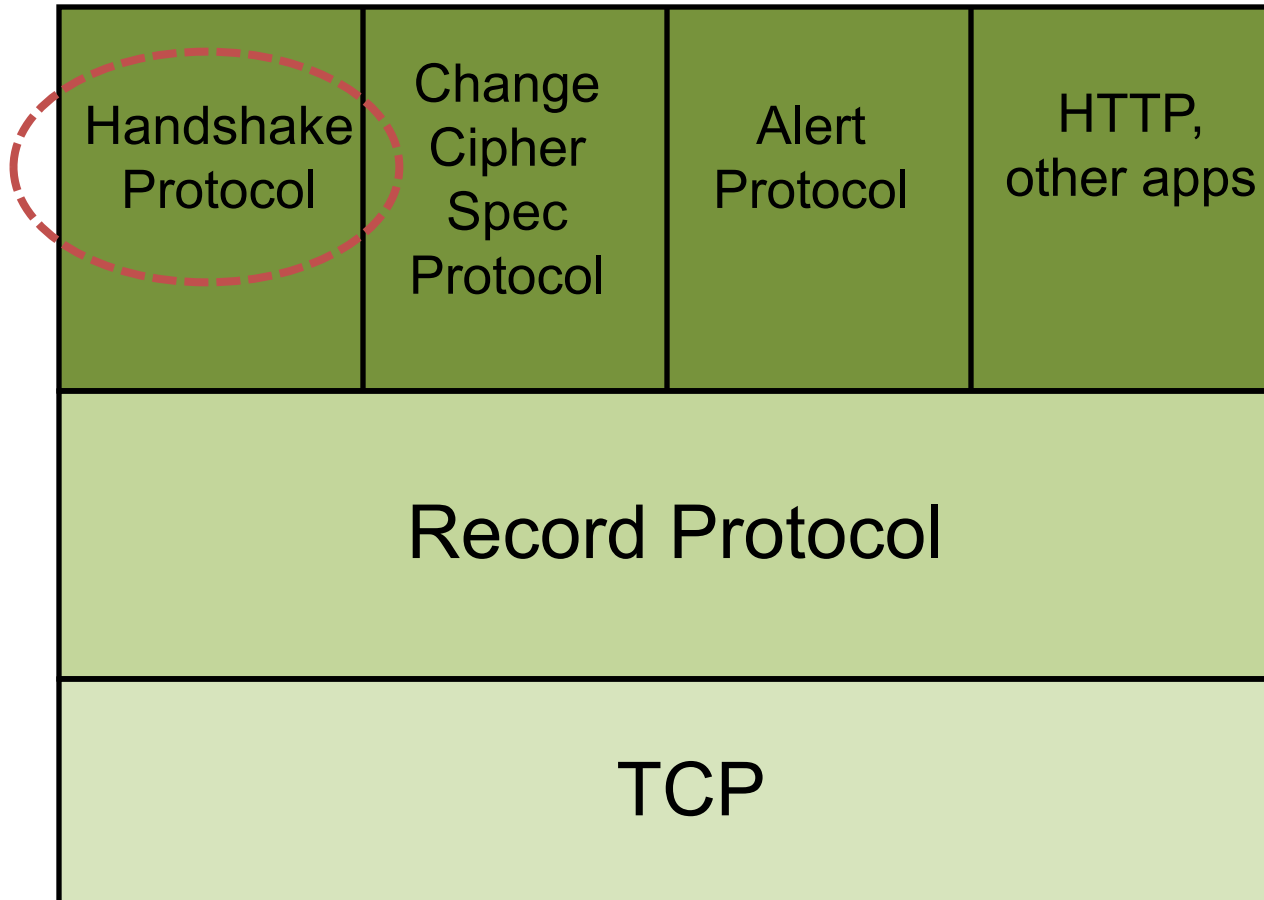
“00” or “01 01” or “02 02 02” or or “FF FF....FF”

AEAD and TLS Record Protocol

Dedicated *Authenticated Encryption with Associated Data* (AEAD) algorithms were added in TLS 1.2, in addition to MEE.

- Single algorithm providing both confidentiality and integrity/data origin (authentication)
- Need not conform to MEE template.
- General AEAD interface specified in RFC 5116.
- AES-GCM specified in RFC 5288.
- AES-CCM specified in RFC 6655.
- ChaCha20-Poly1305 specified in RFC 7539.

TLS Handshake Protocol



TLS Handshake Protocol – Goals

- Establishes keys needed by the Record Protocol.
 - Via establishment of the TLS master secret and subsequent key derivation.
- Provides authentication of server (usually) and client (rarely)
 - Using public key cryptography supported by digital certificates.
 - Or pre-shared key (less commonly?).
- Protects negotiation of all cryptographic parameters.
 - SSL/TLS version number, encryption and hash algorithms, authentication and key establishment methods.
 - To prevent version rollback and cipher suite downgrade attacks.

TLS Handshake Protocol – Negotiation of Cipher Suites

- TLS supports dozens of key establishment mechanisms, authentication methods, and symmetric encryption/integrity options.
- The method used in a given session is negotiated during the Handshake Protocol itself.
 - Client sends list of **cipher suites** it supports in `ClientHello`; server selects one and tells client in `ServerHello`.
 - Each cipher suite is encoded as a 2-byte value
 - e.g. `0x00003D`: `TLS_RSA_WITH_AES_256_CBC_SHA256`
 - e.g. `0x00C011`: `TLS_ECDHE_RSA_WITH_RC4_128_SHA`
 - <https://www.thesprawl.org/research/tls-and-ssl-cipher-suites/>

TLS Handshake Protocol – RSA-based Key Establishment (simplified)

Client

Server

ClientHello (TLS_RSA_WITH_AES_256_CBC_SHA256)



ServerHello, Cert, ServerHelloDone



1. Check Cert
2. Extract PubK from Cert
3. Select random PreMasterSecret
4. Compute $\text{Enc}_{\text{PubK}}(\text{PreMasterSecret})$

RSA encryption using
PKCS#1 v 1.5
padding

ClientKeyExchange: $\text{Enc}_{\text{PubK}}(\text{PreMasterSecret})$



Decrypt to find
PreMasterSecret

TLS Handshake Protocol – Ephemeral DH-based Key Establishment (simplified)

Client

Server

ClientHello (TLS_DHE_RSA_WITH_RC4_128_SHA)

ServerHello, Cert, ServerKeyExchange, ServerHelloDone

1. Check Cert
2. Extract PubK from Cert
3. Use PubK to check server signature
4. Choose y , compute g^y , g^{xy}

params = p, g, g^x ,
RSAsig(nonces, params)

ClientKeyExchange: g^y

PreMasterSecret:
 g^{xy}

TLS Handshake Protocol – Other Key Establishment Options

Static Diffie-Hellman

- Server certificate contains DH parameters (group, generator g) and static DH value g^x .
- Client chooses y , computes g^y and sends to server.

$$\text{PreMasterSecret} = g^{xy}.$$

Anonymous Diffie-Hellman

- Each side sends Diffie-Hellman values in group chosen by server, but no authentication of these values.
- Vulnerable to man-in-middle attacks.

Pre-shared key (PSK)

- `PreMasterSecret` is derived from PSK and nonces.

ECC-based Cipher Suites for TLS

- ECC-based cipher suites for TLS were first defined in RFC 4492 (Blake-Wilson *et al.*, 2006).
- Negotiated via **TLS extensions** sent in `ClientHello/ServerHello` messages.
- 25 different curves + 3 point formats defined in RFC 4492, along with the ability to negotiate bespoke curve.
- Many curves taken from NIST and ANSI standards, e.g. NISTp256.
- Dozens of new ciphersuites (56 with “ECDH(E)”, 24 with “ECDSA”), e.g.:
 - TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
 - TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
 - TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
 - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

TLS Handshake Protocol – RSA-based Authentication

Client

Server

ClientHello (TLS_RSA_WITH_AES_256_CBC_SHA256)



ServerHello, Cert, ServerHelloDone



ClientKeyExchange: $\text{Enc}_{\text{PubK}}(\text{pms})$



1. Decrypt to find pms
2. Derive ms
3. Compute
ServerFinished =
 $\text{PRF}(\text{ms}, \text{transcript})$

ServerFinished



1. Derive ms
2. Compute
ServerFinished' =
 $\text{PRF}(\text{ms}, \text{transcript})$
3. Compare to received version

Server authenticated to client
by proving its ability to
decrypt using its private key.

TLS Handshake Protocol – Authentication for Ephemeral DH-based Key Establishment

Client

Server

ClientHello (TLS_DHE_RSA_WITH_RC4_128_SHA)

ServerHello, Cert, ServerKeyExchange, ServerHelloDone

1. Check Cert
2. Extract PubK from Cert
3. Use PubK to check server signature
4. Choose y , compute g^y, g^{xy}

params = $p, g, g^x,$
 $RSAsig(nonces, params)$

ClientKey

Server authenticated by
proving its ability to sign client
nonce using its private key

ServerFinished

secret_secret:
 g^{xy}

TLS Handshake Protocol – Adding the ClientFinished Message

Client

Server

ClientHello

ServerHello, Cert, [ServerKeyExchange,] ServerHelloDone

1. Derive ms
2. Compute
ClientFinished =
PRF(ms, transcript)

ClientKeyExchange, ClientFinished

ServerFinished

1. Derive ms
2. Compute
ClientFinished' =
PRF(ms, transcript)
3. Compare to received version

TLS Handshake Protocol – Finished Messages

- TLS Finished messages enable each side to check that both views of the Handshake Protocol are the same.
- Computed as:

$\text{PRF}(\text{ms}, \text{transcript})$

where `transcript` = sender's view of all protocol messages sent and received up to *this* point.

- Compared by recipient to expected value; protocol aborts if mismatch is observed.
- Designed to prevent version rollback and cipher suite downgrade attacks.
- Ineffective if attacker can somehow compute `ms` during protocol run...

The Full TLS Handshake Protocol

Client

Server

ClientHello

ServerHello, Cert, [ServerKeyExchange,] ServerHelloDone

ClientKeyExchange, CCS, ClientFinished

CCS, ServerFinished

First flow containing
application data: TLS
is a 2-RTT protocol!

ClientData

ServerData

Indicator from
client to server: I
am switching to
recently agreed
keys and cipher
suite



TLS Handshake Protocol – Adding Client Authentication

Client

Server

ClientHello



ServerHello, Cert, [ServerKeyExchange, CertificateRequest,]
ServerHelloDone



[Cert,] ClientKeyExchange, [CertificateVerify,] CCS,
ClientFinished



CCS, ServerFinished



TLS Handshake Protocol – Renegotiation

- **Renegotiation** allows re-keying and change of cipher suite during a session.
 - For example, to force strong client-side authentication before access to a particular resource on the server is allowed.
 - Or to upgrade security from an EXPORT cipher suite to a strong one.
- Renegotiation is initiated by client sending `ClientHello` or server sending `ServerHelloRequest`.
 - Followed by a **full** run of the Handshake Protocol.
 - Takes place under the protection of the existing Record Protocol.

TLS Handshake Protocol – Session Resumption

- **Session resumption** allows authentication and shared secrets to be efficiently reused across multiple, parallel *connections* in a single session.
- Allows, for example, fetching multiple resources from same website in parallel, without re-doing full, expensive Handshake Protocol and overcoming TCP slowstart feature.

TLS Handshake Protocol – Session Resumption

Client

Server

ClientHello (SessionID)

N_C, N_S

ms

PRF

key_block

ServerHello (SessionID), CCS, ServerFinished

N_C, N_S

ms

PRF

key_block

CCS, ClientFinished

TLS Extensions

- Many **extensions** to TLS exist.
- Allows extended capabilities and security features.
- Examples:
 - Elliptic curve crypto extensions, RFC 4492.
 - Renegotiation Indicator Extension (RIE), RFC 5746.
 - Application layer protocol negotiation (ALPN), RFC 7301.
 - Authorization Extension, RFC 5878.
 - Server Name Indication, Maximum Fragment Length Negotiation, Truncated HMAC, etc, RFC 6066.
- Extensions supported are signalled in `ClientHello` and `ServerHello` messages.

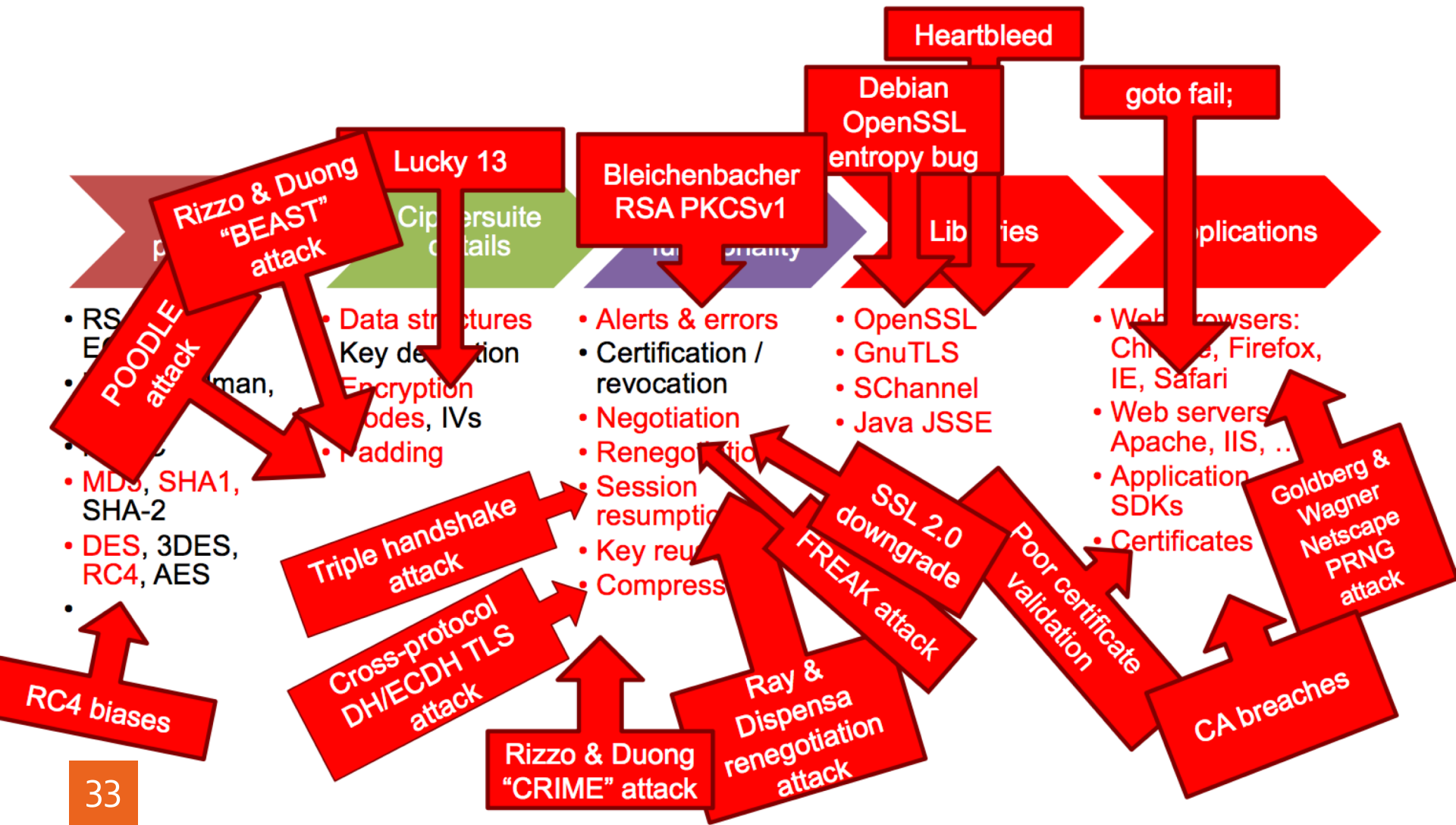
TLS Protocol Complexity

- The TLS Protocol is much more complex than a “textbook” key exchange protocol + AEAD scheme:
 - The protocol is “self-negotiating”.
 - There are many options and extensions.
 - There are interactions between sub-protocols (CCS, Record Protocol, Alert Protocol, Handshake Protocol).
 - There are interactions between versions (downgrade issues).
 - Keys are re-used across authentication modes (RSA for both signatures and key transport) and versions.
 - There is no clear state machine or API in the specification.

What could possibly go wrong?

Attacks on TLS

(Slide from Douglas Stebila.)



TLS 1.3

- A major redesign of TLS commenced in the TLS WG of IETF in 2014.
- High-level aims: **improve security; improve performance** (by reducing latency to first secure flow).
- Remove old and broken crypto (e.g. AEAD only, no RSA encryption in handshake, limited set of DH groups,...).
- Reduce complexity by removing features like renegotiation.
- Introduce a 1-RTT, forward-secure handshake protocol, and a 0-RTT option (sacrificing forward security).
- Technical work on TLS 1.3 is finally complete, RFC 8446 is published, and deployment is underway.

SSL/TLS Measurement Study

Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G. Paterson, Narseo Vallina-Rodriguez, Juan Caballero.

Coming of Age: A Longitudinal Study of TLS Deployment.

ACM Internet Measurement Conference 2018, *to appear.*

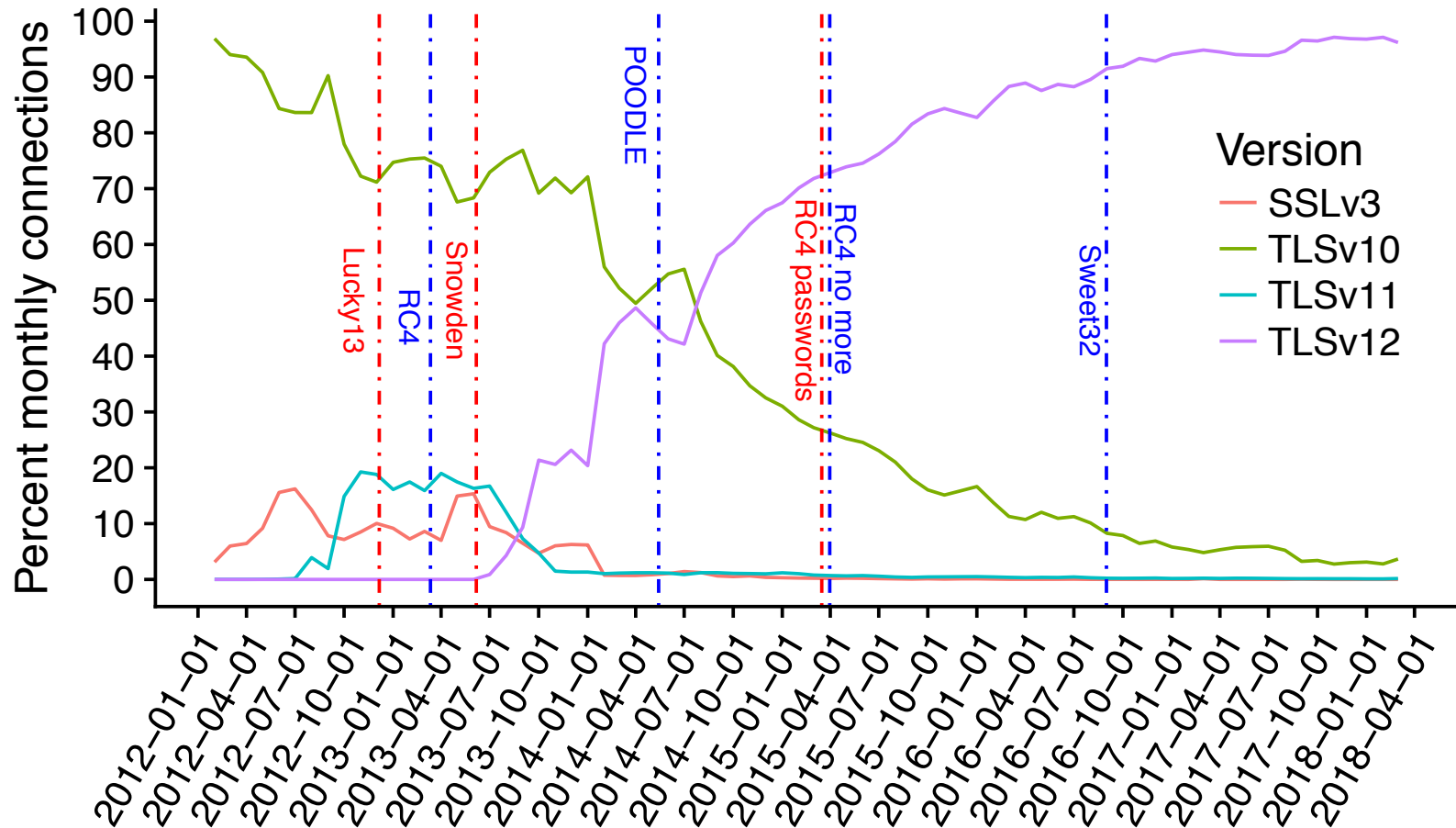
SSL/TLS Measurement Study

- Data from ICSI Certificate Notary @ University of Berkeley.
 - Tracking SSL/TLS connections on all ports from 2/2012 to 3/2018, at roughly 200k hosts from a variety of sites.
 - Metadata captured from [319.3 billion SSL/TLS connections](#).
 - Recording main characteristics of connections: versions, cipher suites offered and selected, extensions offered and accepted.
 - Diversity in handshake allows limited client fingerprinting (1684 fingerprints, 69% of connections).

SSL/TLS Measurement Study

- This data can be “tortured” in various ways.
- We focussed on the question: **What impact have attacks had on the TLS ecosystem?**
- **Examples:**
 - What versions of TLS are in use, and how is this changing?
 - Can we see a change in cipher suites being used in response to attacks?
 - Can we see any impact from the Snowden revelations?
 - How quickly do clients and servers get updated in response to attacks?

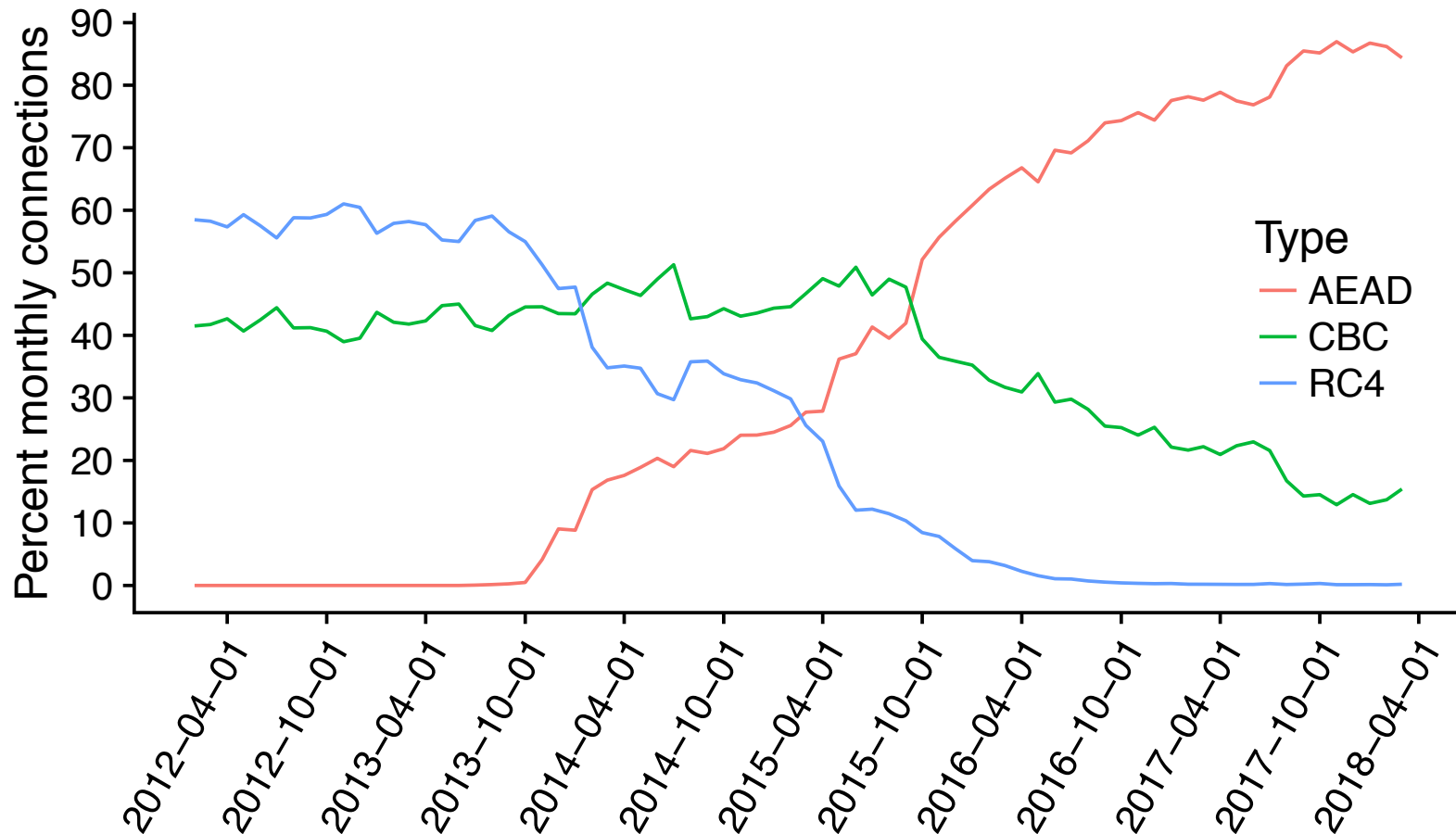
SSL/TLS Negotiated Versions



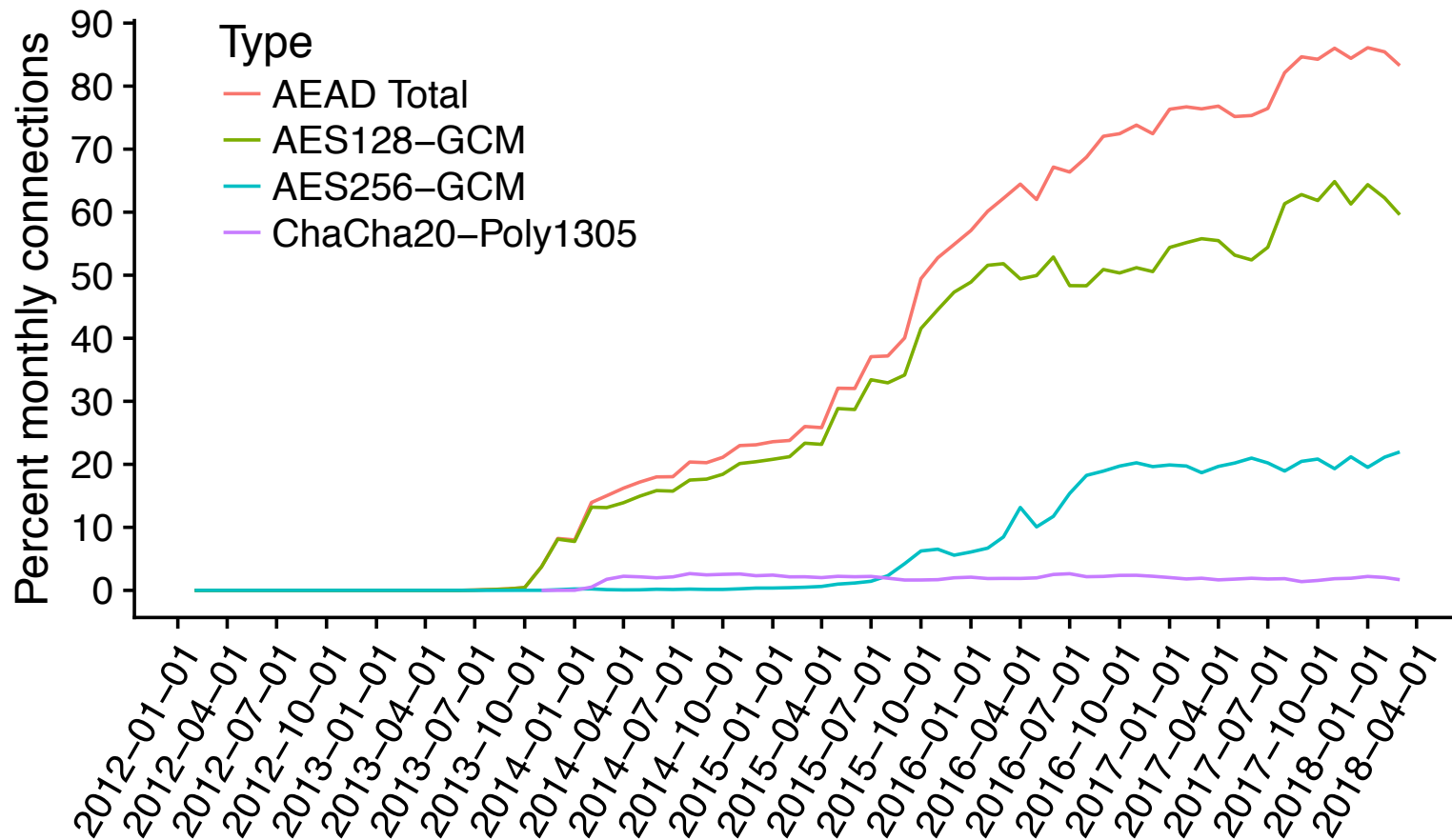
SSL/TLS Negotiated Versions

- Big uptake in TLS 1.2 starting in late 2013.
- Corresponds to increase in use of AEAD algorithms, avoiding issues arising in CBC-mode (BEAST, Lucky 13, POODLE) and first attacks on RC4.
- Almost no SSLv2 (1.2k connections in Feb. 2018).
- 360.1K SSLv3 connections in Feb. 2018 to 1789 different servers.
- 4 servers received more than 50,000 SSLv3 connections; all belong to Symantec and Wayport.
- Less than 25% of servers support SSLv3 (May 2018).

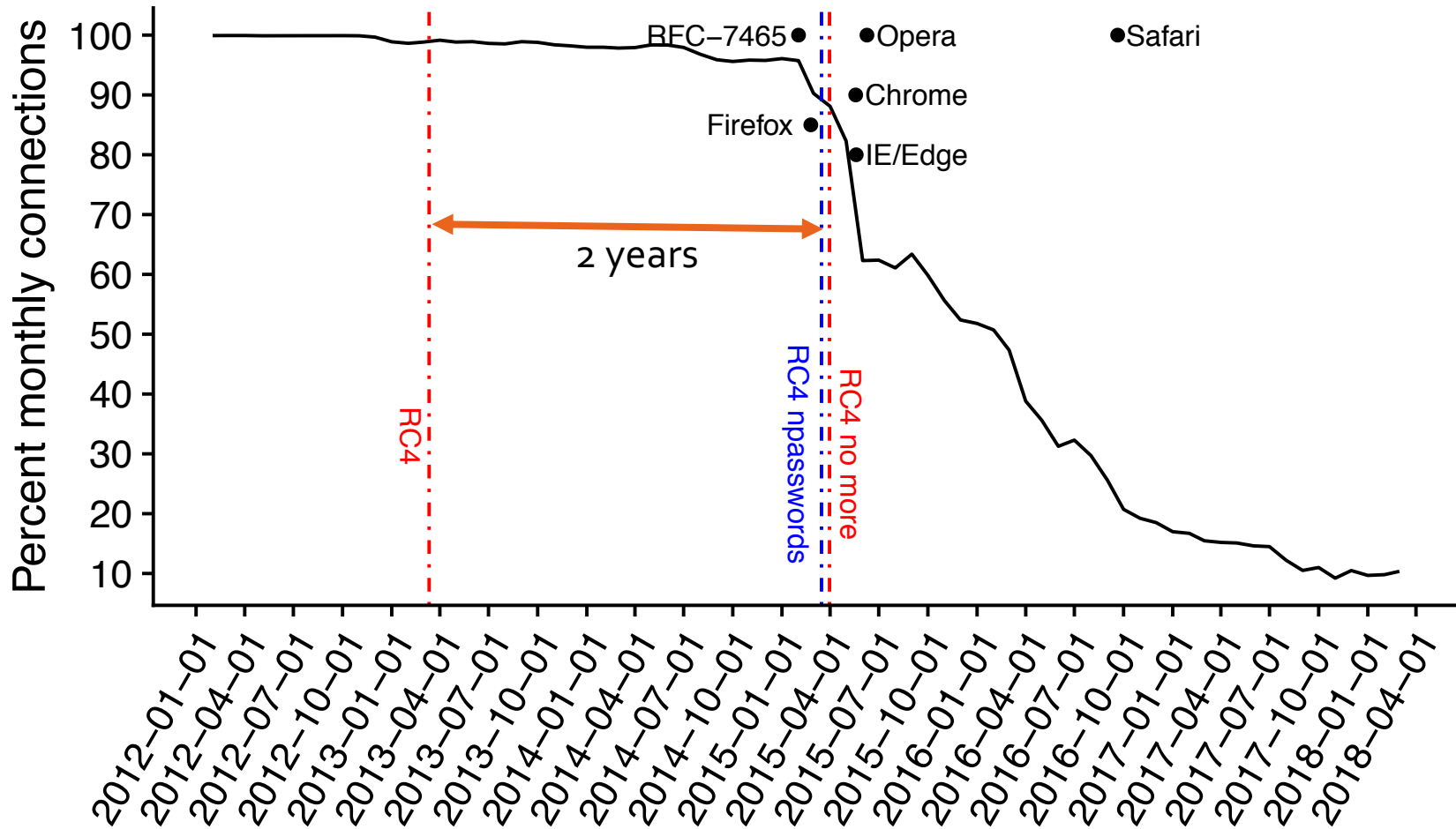
SSL/TLS Record Protocol Algorithms in Use



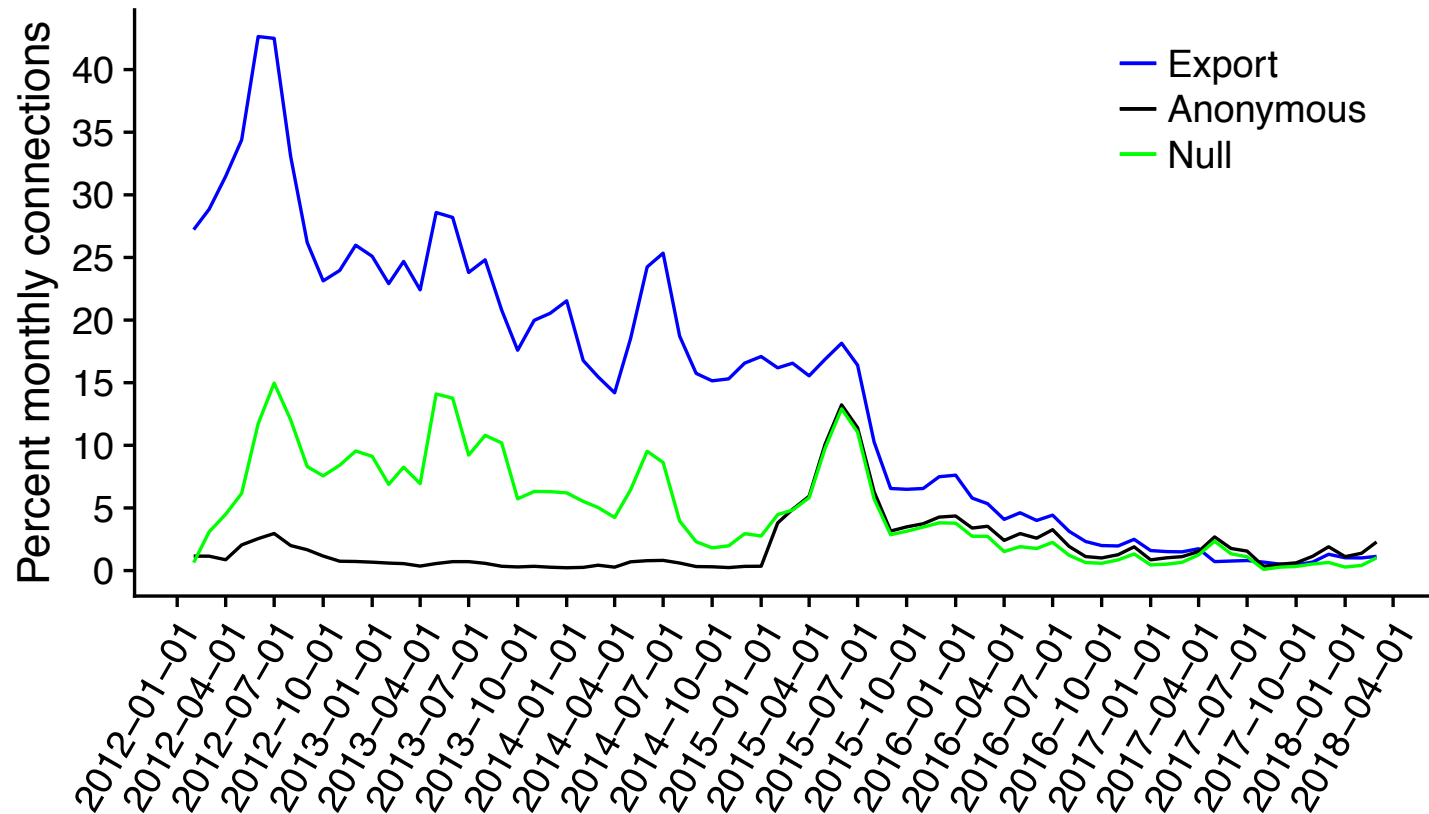
SSL/TLS AEAD Algorithms in Use



SSL/TLS Connections in Which Clients Offer RC4



SSL/TLS Connections with Client Offering Null, Anonymous or Export Cipher Suites

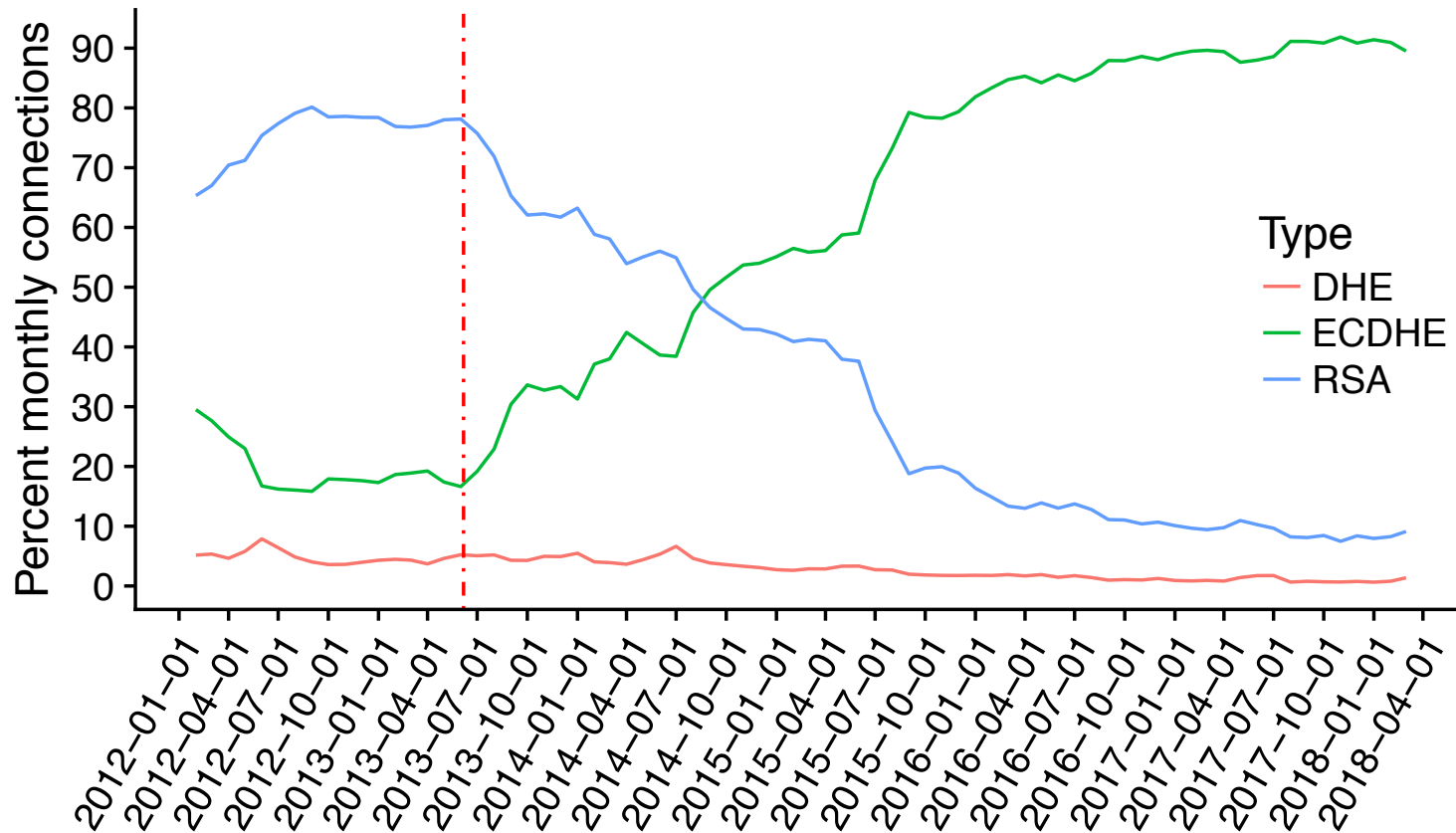


Export: typically 40-bit security level, legacy of 1990s crypto restrictions.

Anonymous: client/server not authenticated.

Null: mostly grid traffic, integrity only (atypical)

SSL/TLS: Key Exchange Methods



Dotted line: beginning of Snowden revelations
(and implementation of ECC in OpenSSL).

Concluding Remarks

- **Complexity:** the TLS protocol is complicated with many options and interacting components.
- **Long-tail effects:** almost anything you can imagine happening does happen in some SSL/TLS connections somewhere, and will continue to do so for some time!
- **Evolution:** the TLS ecosystem has been evolving rapidly in response to increased attention from the research community: not only attacks, also security proofs.
- **TLS 1.3 is coming – and quickly!**

Extra slide: TLS 1.3

- TLS was just starting to see adoption at the end of our study.
 - 0.5% of clients advertised TLS 1.3 in February 2018.
 - 9.8% in March 2018.
 - 23.6% in April 2018.
 - But only 1.3% of connections actually negotiated TLS 1.3 in April 2018: server-side deployment lagging client-side.
- Complex picture because of use of “private version numbers” on experimental basis by Google, Mozilla and others.
- We plan to track adoption of TLS 1.3 over months and years ahead.