A messy state of the union:

Taming the Composite State Machines of TLS

http://smacktls.com

Benjamin Beurdouche, *Karthikeyan Bhargavan*,
Antoine Delignat-Lavaud, Cédric Fournet,
Markulf Kohlweiss, Alfredo Pironti,
Pierre-Yves Strub, Jean Karim Zinzindohoue

Agile Cryptographic Protocols

Modern protocols negotiate crypto parameters

- Many key exchanges (RSA, DHE, PSK)
- Many authentication mechanisms (Cert, Password)
- Many encryption schemes (AEAD, RC4-HMAC)
- *Much of the complexity of TLS, IKEv2, SSH is in the composition of these mechanisms*

How do we implement such protocols correctly?

- What can go wrong? Can we prove them correct?

Transport Layer Security (1994—)

The default secure channel protocol?

HTTPS, 802.1x, VPNs, files, mail, VoIP, …

20 years of attacks, and fixes

1994 Netscape's Secure Sockets Layer
1996 SSL3
1999 TLS1.0 (RFC2246)
2006 TLS1.1 (RFC4346)
2008 TLS1.2 (RFC5246)

2015 TLS1.3?

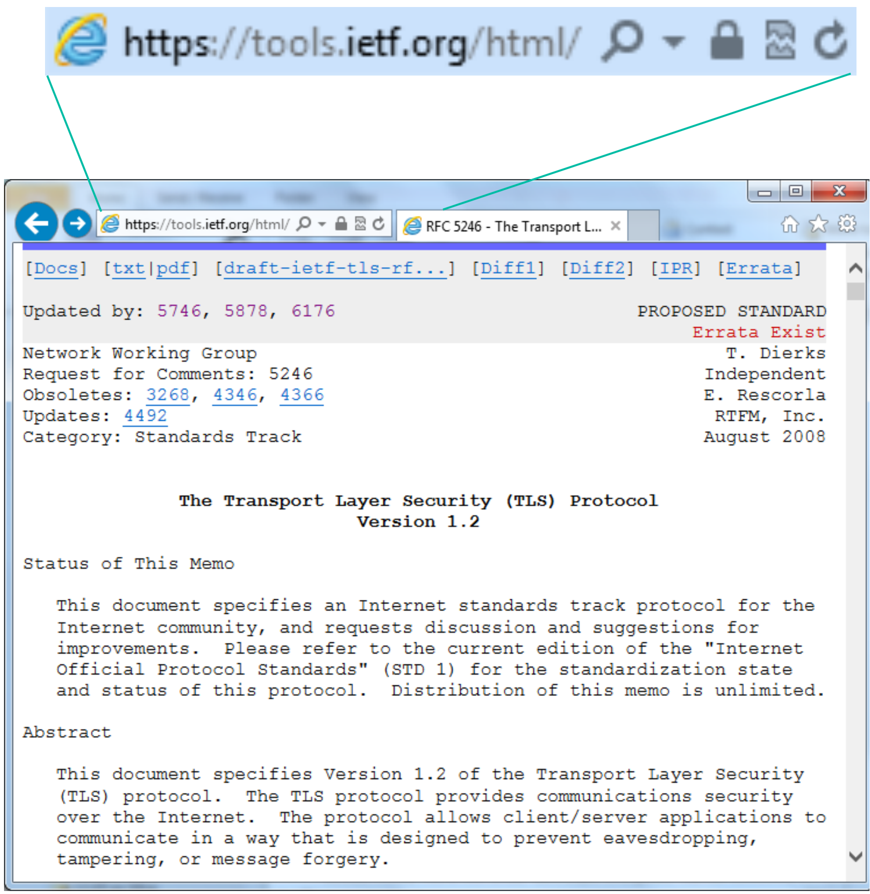
Many implementations

OpenSSL, SecureTransport, NSS, SChannel, GnuTLS, JSSE, PolarSSL, …

many bugs, attacks, patches every year

Many security theorems

mostly for small simplified models of TLS



[Docs] [txt|pdf] [draft-ietf-tls-rf...] [Diff1] [Diff2] [IPR] [Errata]

Updated by: 5746, 5878, 6176 PROPOSED STANDARD
Errata Exist

Network Working Group T. Dierks
Request for Comments: 5246 Independent
Obsoletes: 3268, 4346, 4366 E. Rescorla
Updates: 4492 RTFM, Inc.
Category: Standards Track August 2008

The Transport Layer Security (TLS) Protocol
Version 1.2

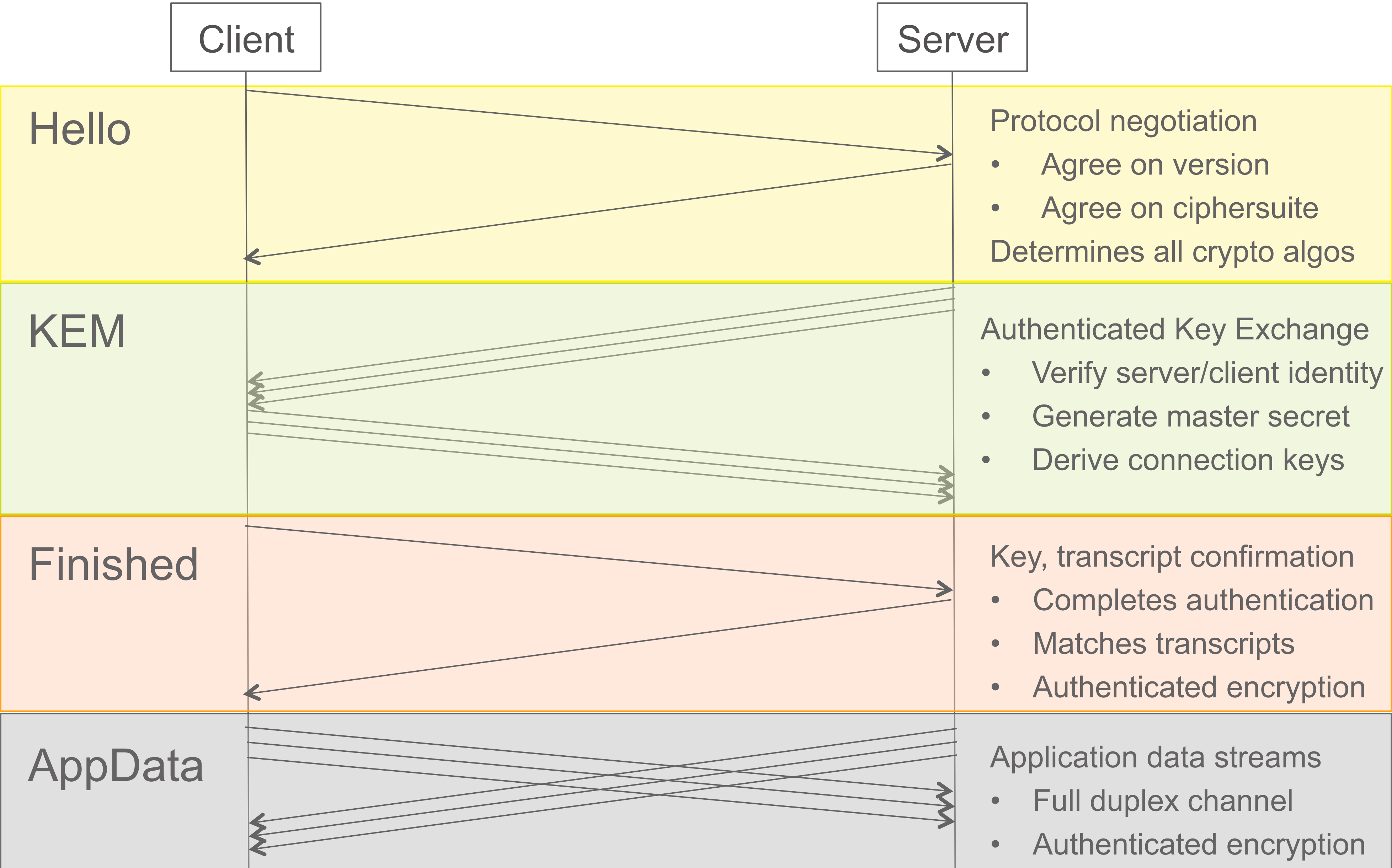
Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

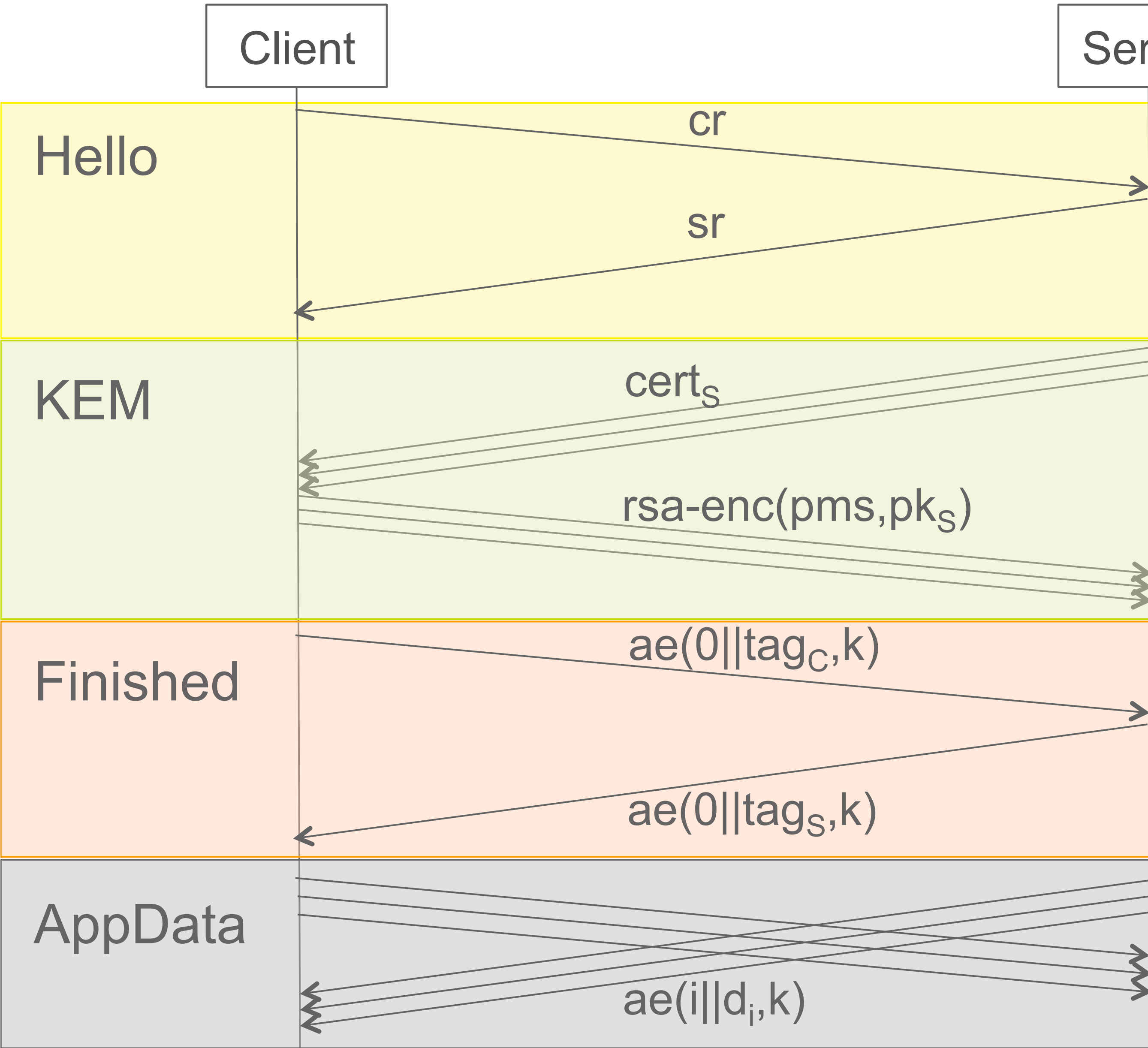
Abstract

This document specifies Version 1.2 of the Transport Layer Security (TLS) protocol. The TLS protocol provides communications security over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

TLS protocol overview



RSA Key Transport



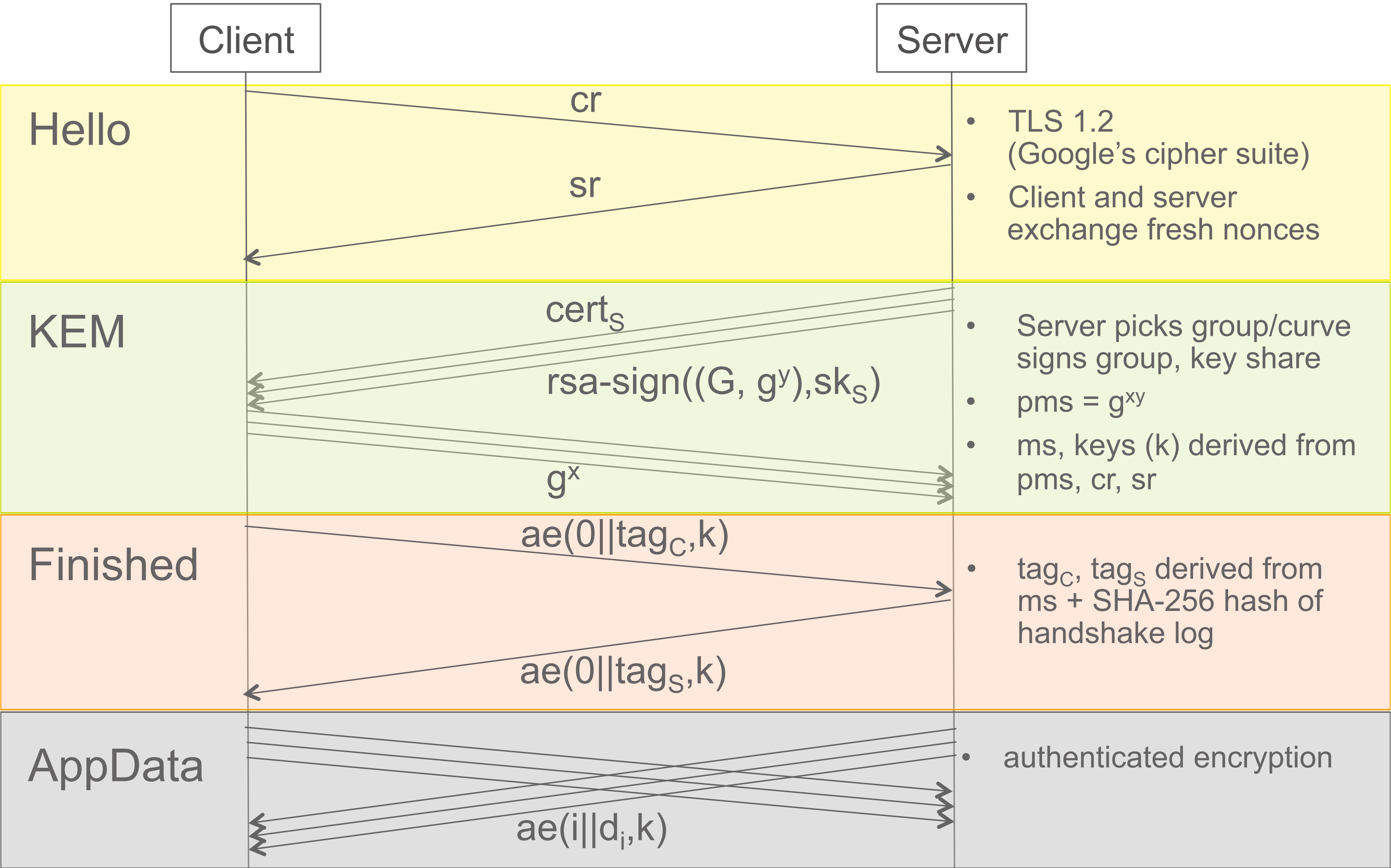
Hello
- TLS 1.2 (mandatory cipher suite)
- Client and server exchange fresh nonces

KEM
- Server certificate cert$_S$ supports RSA encryption
- Client generates pms
- ms, keys (k) derived from pms, cr, sr

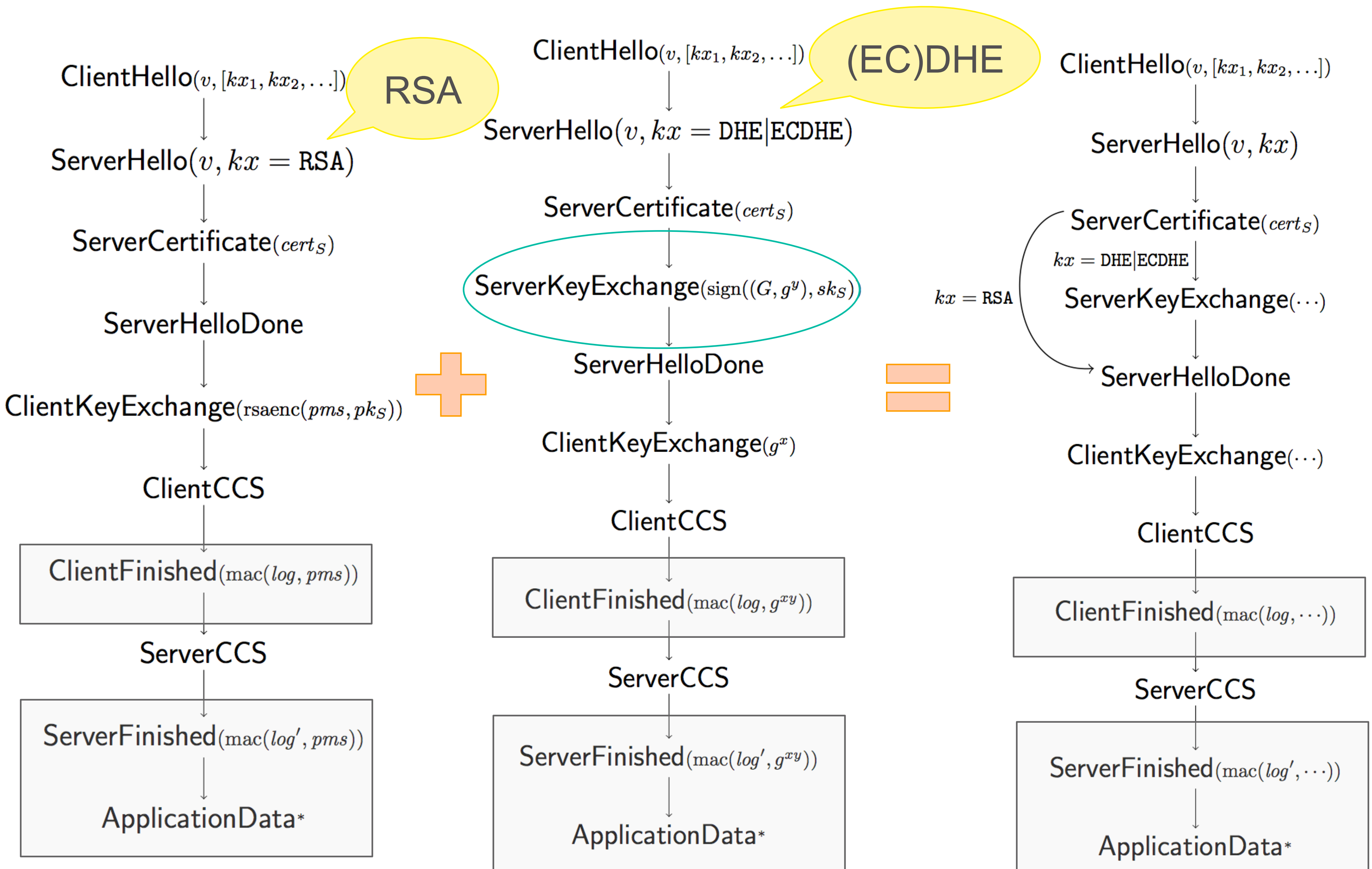
Finished
- tag$_C$, tag$_S$ derived from ms + SHA-256 hash of handshake log

AppData
- authenticated encryption

(EC)DHE Key Exchange



Composing Key Exchanges



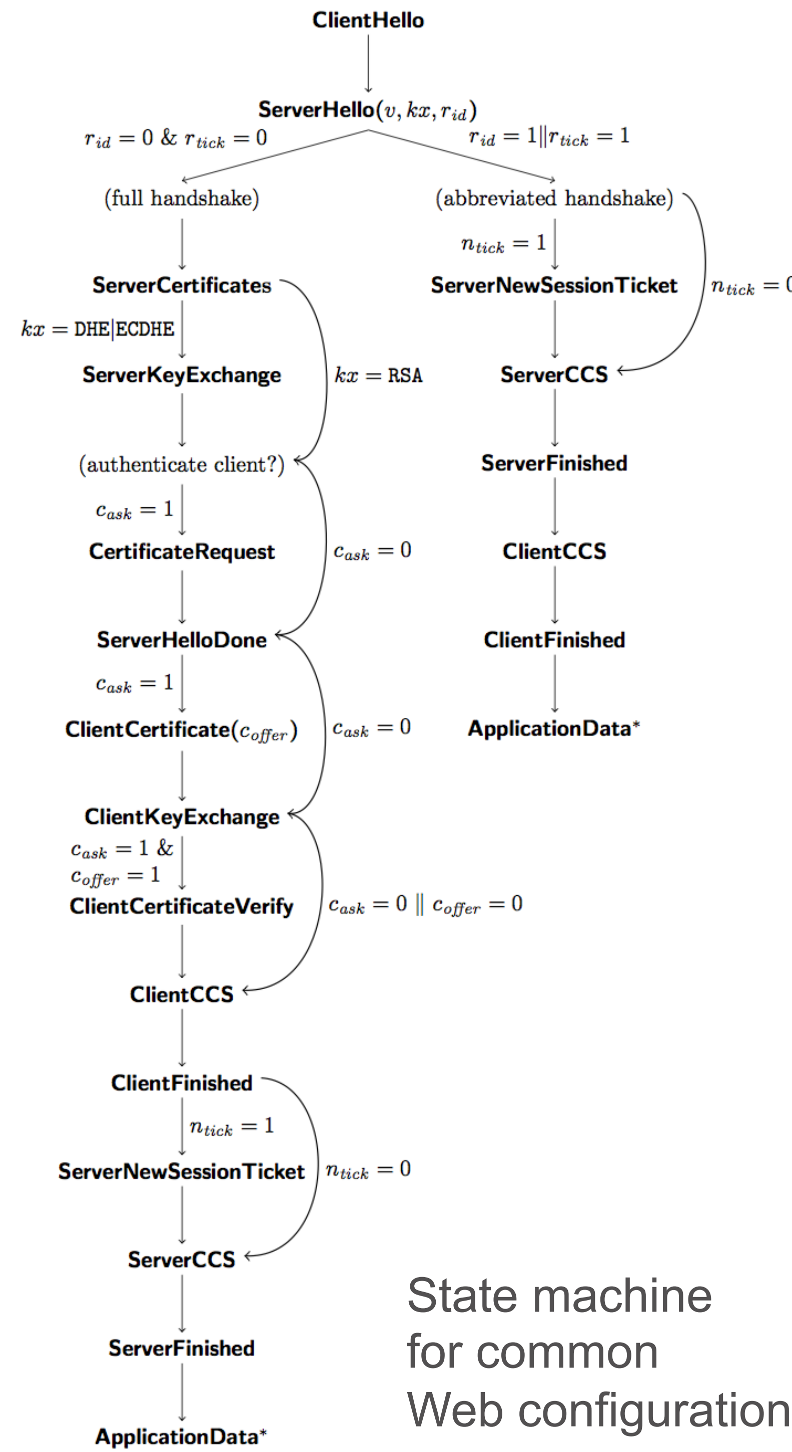
TLS State Machine

RSA + DHE + ECDHE
+ Session Resumption
+ Client Authentication

- Covers most features used on the Web
- Composition proved secure for miTLS implementation [IEEE S&P'13, CRYPTO'14]

  http://mitls.org
- Reference code written for verification, in F#

Can this proof technique be applied to OpenSSL?

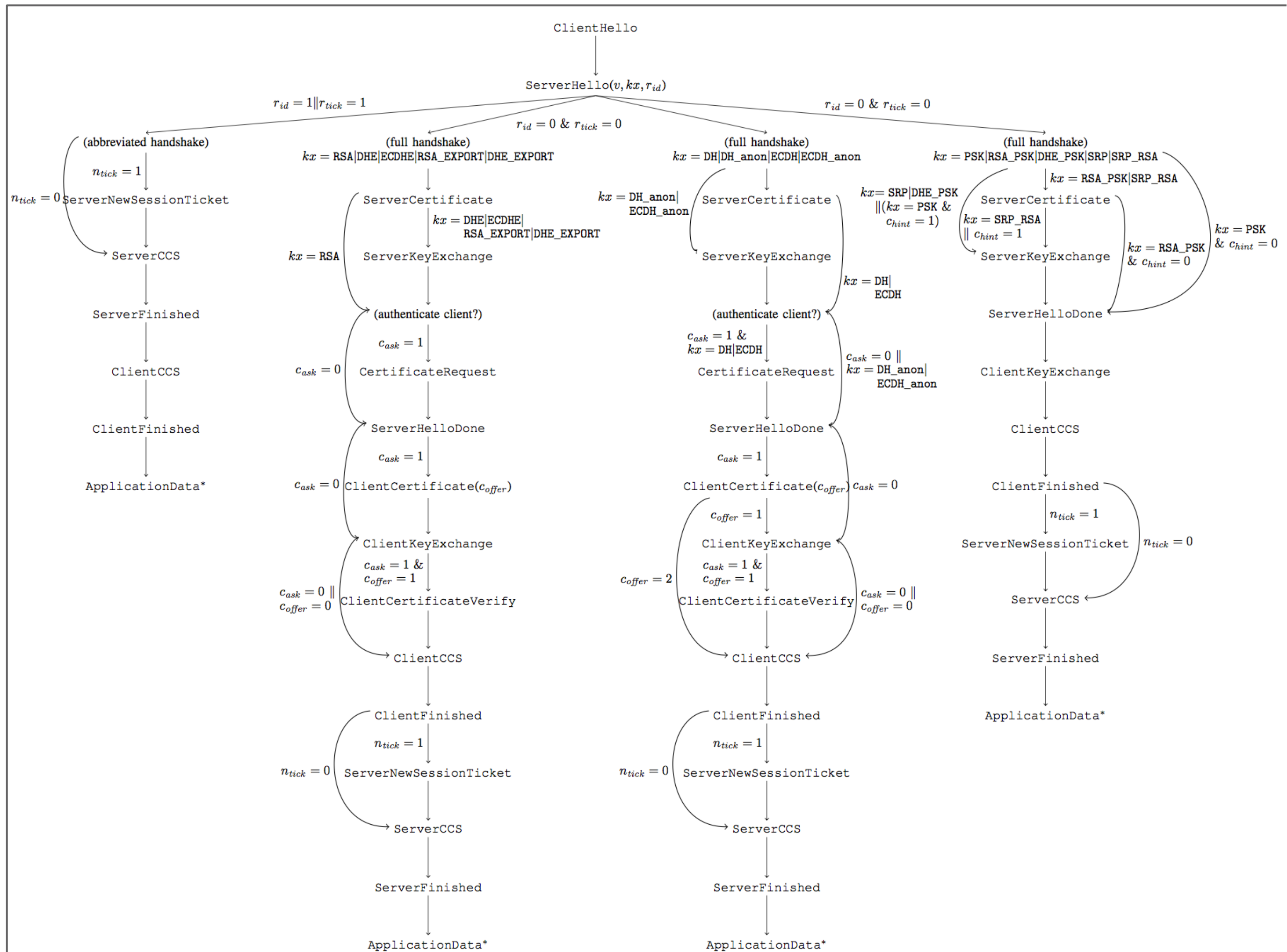


State machine for common Web configurations

OpenSSL State Machine

+ Fixed_DH
+ DH_anon
+ PSK
+ SRP
+ Kerberos
+ *_EXPORT
+ …

We cannot ignore all these because they share code/keys with RSA/DHE



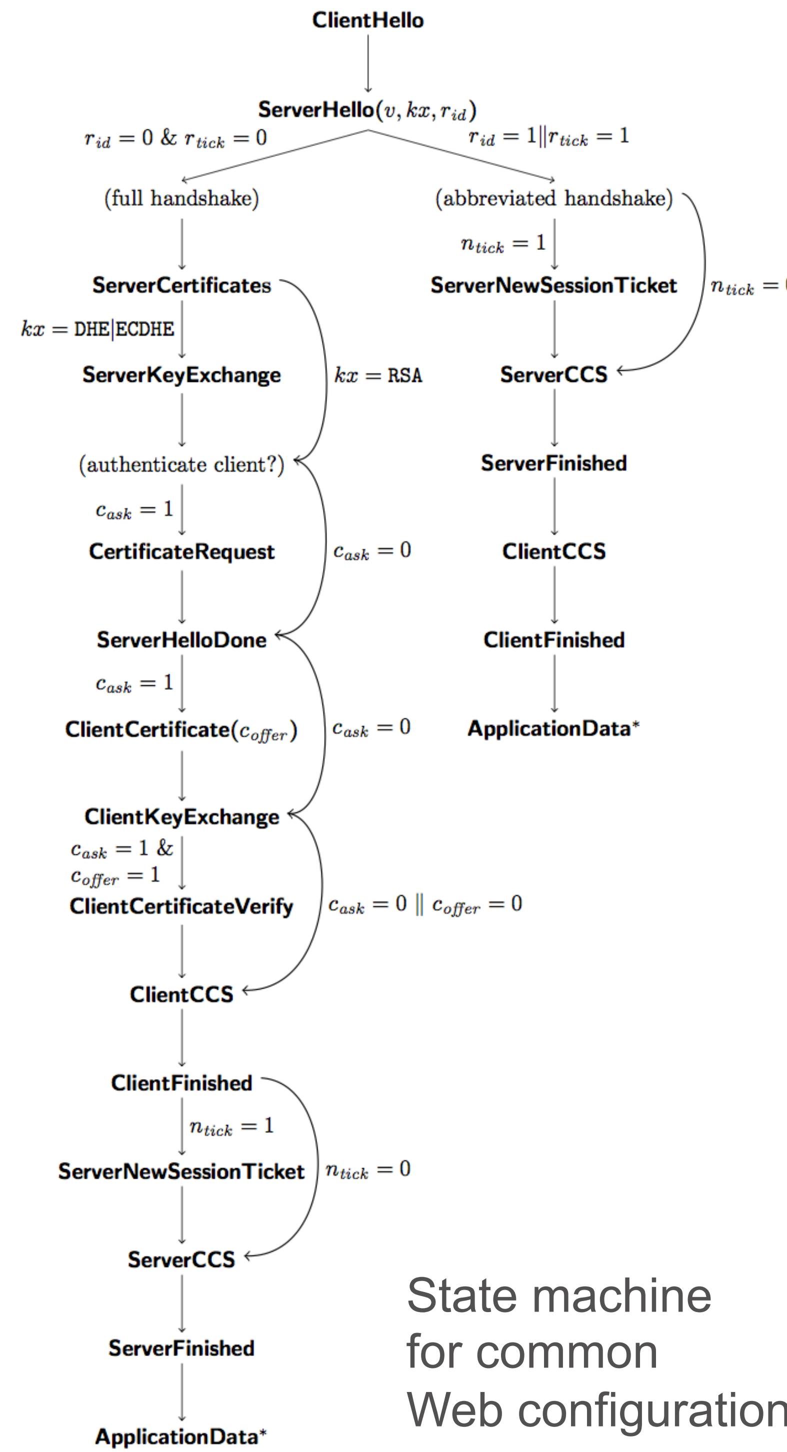
Fuzzing TLS

Does OpenSSL conform to the miTLS state machine?

- There are known attacks if it doesn't [EarlyCCS 2014]

We built a test framework

- FlexTLS, based on miTLS
- Generates 100s of non-conforming traces from a *state machine specification*
- We tested many TLS libraries



State machine for common Web configurations

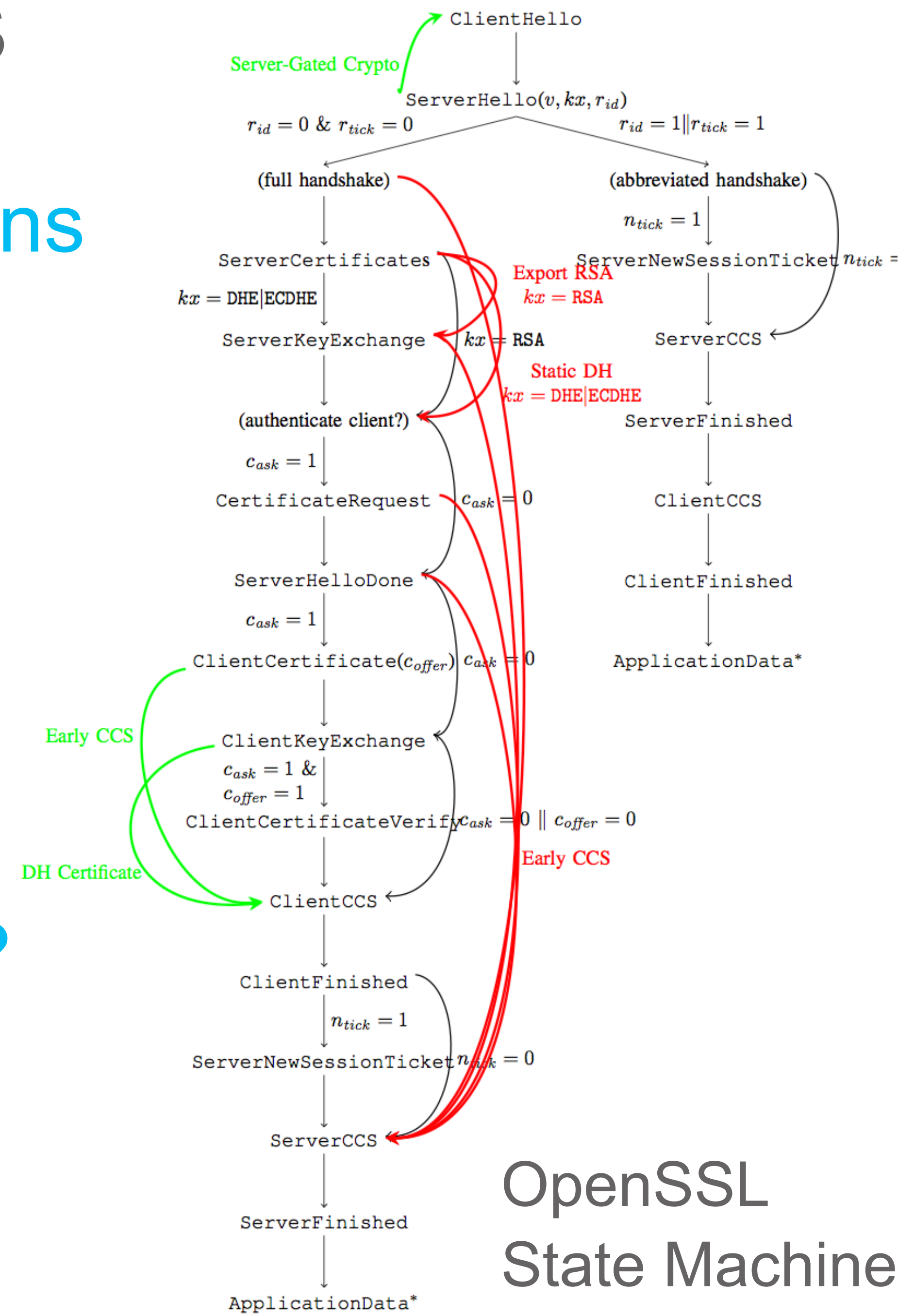
Many, Many Bugs

Unexpected state transitions in OpenSSL, NSS, Java, SecureTransport, …

- Required messages are allowed to be skipped
- Unexpected messages are allowed to be received
- CVEs for many libraries

How come all these bugs?

- In independent code bases, sitting in there for years
- Are they exploitable?



OpenSSL State Machine

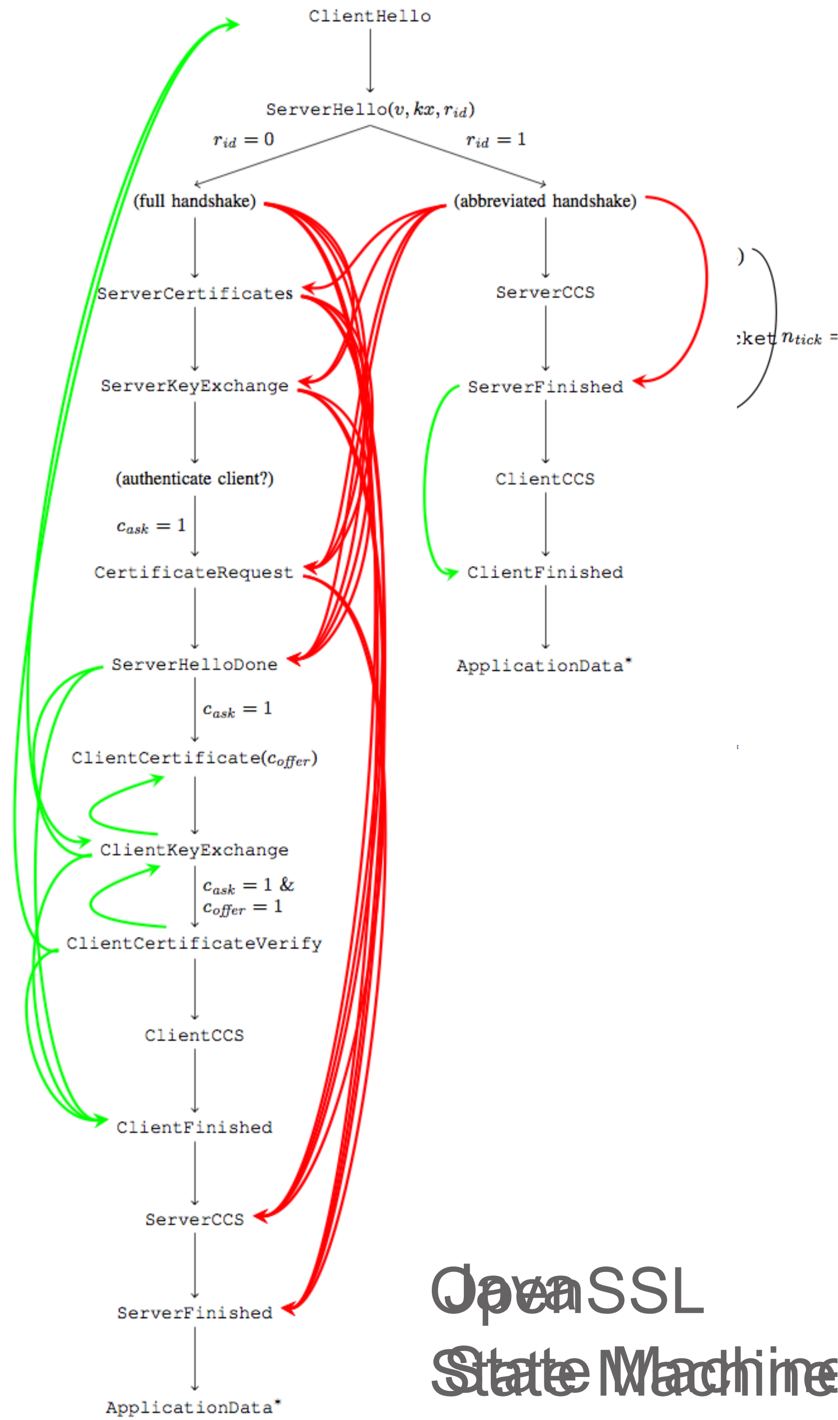
Many, Many Bugs

Unexpected state transitions in OpenSSL, NSS, Java, SecureTransport, …

- Required messages are allowed to be skipped
- Unexpected messages are allowed to be received
- CVEs for many libraries

How come all these bugs?

- In independent code bases, sitting in there for years
- Are they exploitable?



OpenSSL
State Machine

Culprit: Underspecified State Machine

TLS specifies a ladder diagram with optional messages

- Handshake ends with agreement on transcript

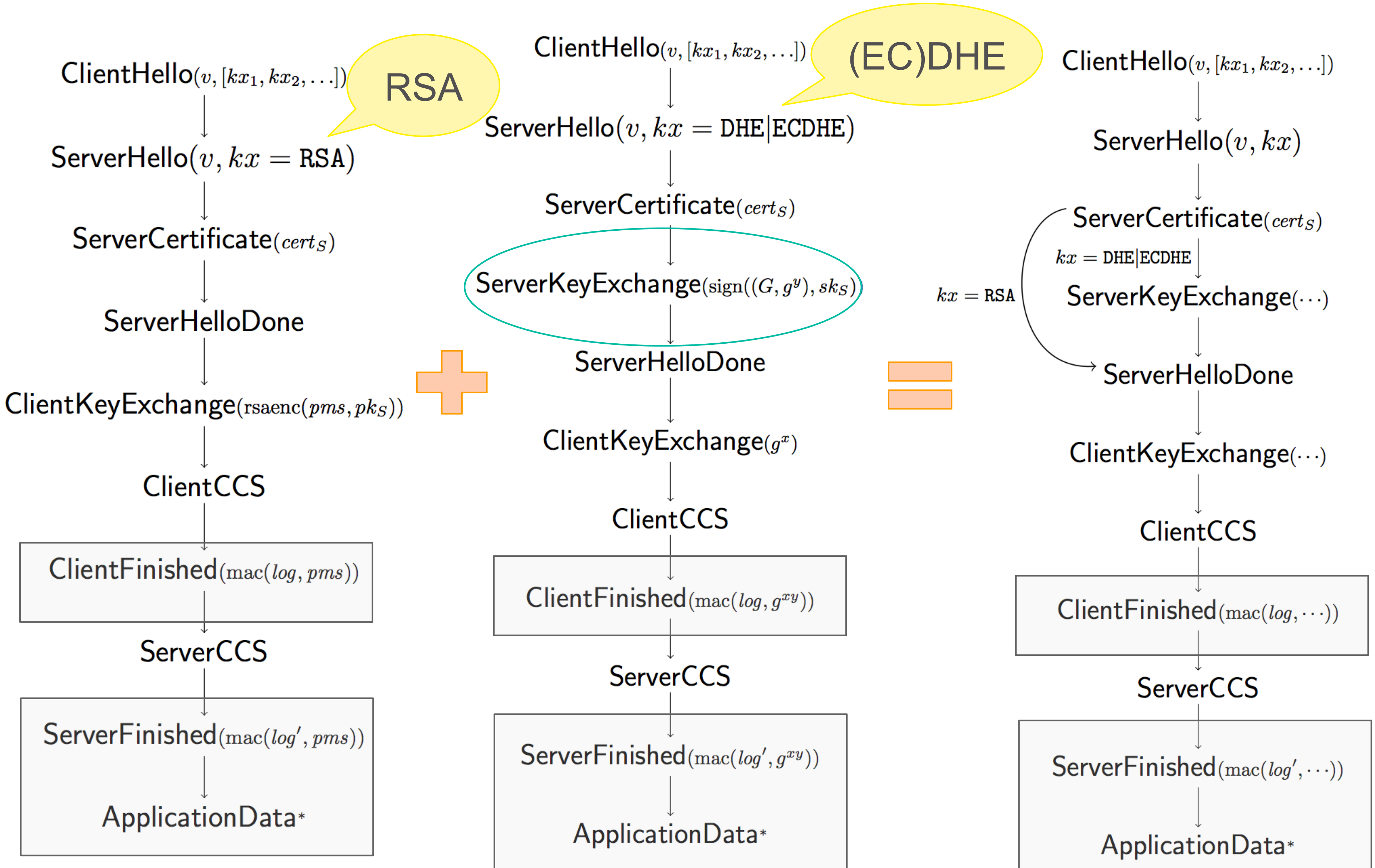
RFC 5246 TLS August 2008

```
      Client                                               Server

      ClientHello                  -------->
                                                      ServerHello
                                                     Certificate*
                                               ServerKeyExchange*
                                              CertificateRequest*
                                   <--------      ServerHelloDone
      Certificate*
      ClientKeyExchange
      CertificateVerify*
      [ChangeCipherSpec]
      Finished                     -------->
                                               [ChangeCipherSpec]
                                   <--------             Finished
      Application Data             <------->     Application Data
```

Figure 1. Message flow for a full handshake

Composing Key Exchanges



Composing with Optional Messages

Treat ServerKeyExchange as optional

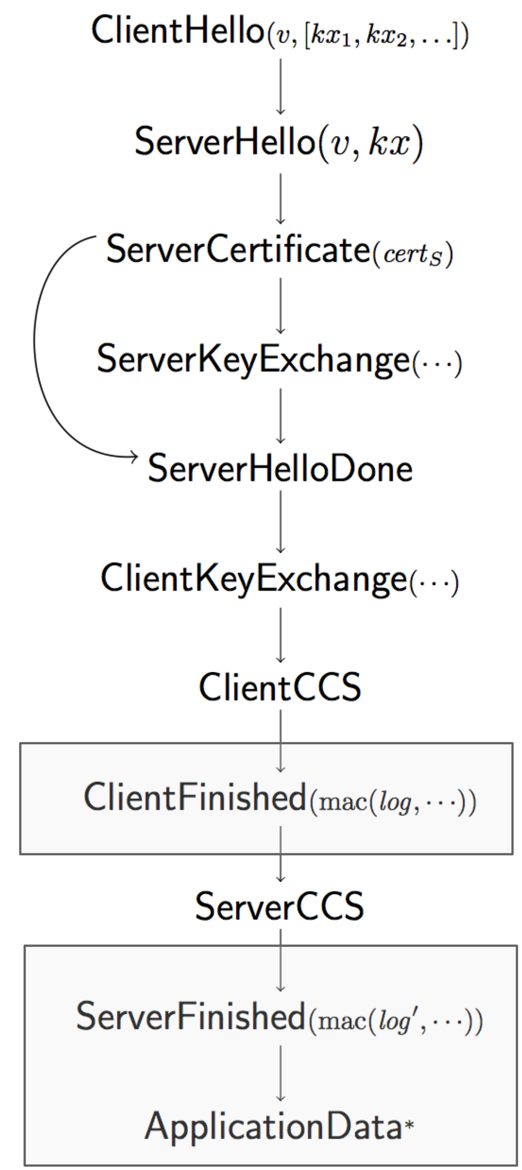
- Server decides to send it or not
- Client tries to handle both cases
- Consistent with Postel's principle: "*be liberal in what you accept*"

Unexpected cases at the client

- Server skips ServerKeyExchange in DHE
- Server sends ServerKeyExchange in RSA

Clients should reject these cases

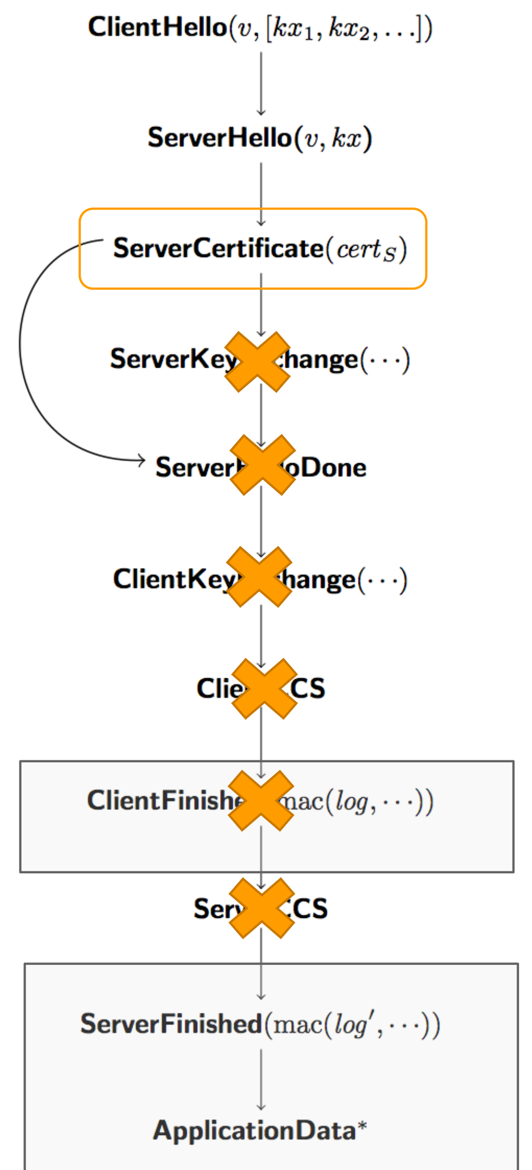
- In practice: clients accept and perform unexpected cryptographic computations, breaking the security of TLS



SKIP: Server Impersonation with DHE

Network attacker impersonates S.com to a Java TLS client

1. Send S's cert
2. SKIP ServerKeyExchange (bypass server signature)
3. SKIP ServerHelloDone
4. SKIP ServerCCS (bypass encryption)
5. Send ServerFinished using uninitialized MAC key (bypass handshake integrity)
6. Send ApplicationData (unencrypted) as S.com



Export-Grade RSA in TLS

TLS 1.0 supported weakened ciphers to comply with export regulations in 1990s

- RSA keys limited to 512 bits
- Export keys are sent in a signed **ServerKeyExchange**
- Client uses the 512-bit key instead of S's public key

EXPORT deprecated in 2000

- (Dead) code still exists in OpenSSL and many other libraries
- Can be triggered by sending an unexpected ServerKeyExchange

ClientHello$(v, [kx_1, kx_2, \ldots])$

↓

ServerHello$(v, kx = \texttt{RSA_EXPORT})$

↓

ServerCertificate$(certs_S)$

↓

ServerKeyExchange$(\mathrm{sign}(pk_{512}, sk_S))$

↓

ServerHelloDone

↓

ClientKeyExchange$(\mathrm{rsaenc}(pms, pk_{512}))$

↓

ClientCCS

↓

ClientFinished$(\mathrm{mac}(log, pms))$

↓

ServerCCS

↓

ServerFinished$(\mathrm{mac}(log', pms))$

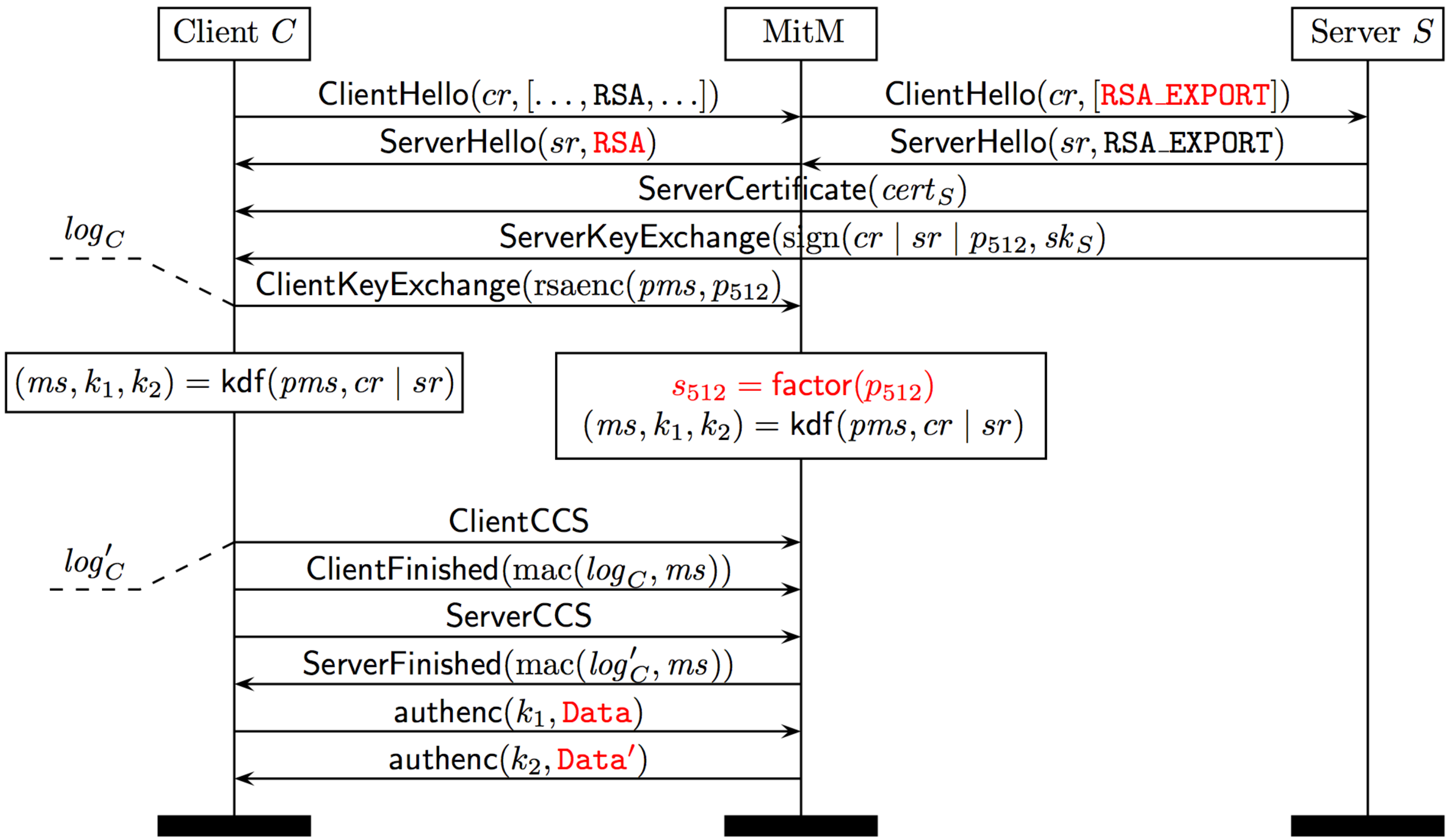
↓

ApplicationData*

FREAK: Downgrade to RSA_EXPORT

A man-in-the-middle attacker can:

- impersonate servers that support RSA_EXPORT,
- at buggy clients that allow ServerKeyExchange in RSA



FREAK: Exploit and Impact

Many servers in 2015 offer RSA_EXPORT

- 37% of browser-trusted servers in March 2015
- After FREAK: came down to 6.5% [Zmap team, 2015]
- See: www.smacktls.com/#freak
- Vulnerable sites included nsa.gov, hsbc.com, …

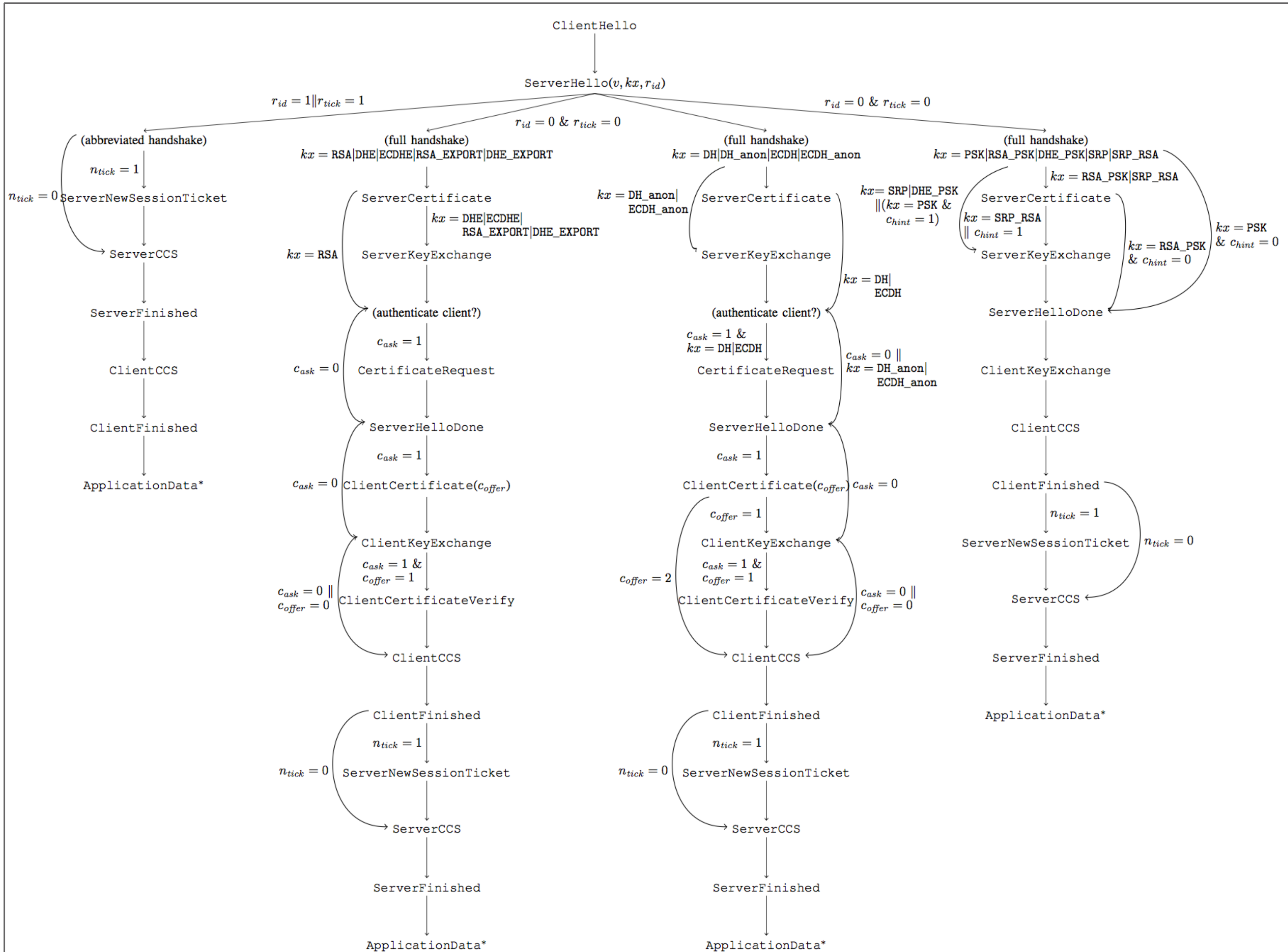
Factoring 512-bit RSA keys is easy

- First broken with CADO-NFS in 2000 [EuroCrypt'00]
- Now: 12 hours and $100 on Amazon EC2 [N. Heninger]

Client-side state machine bugs are widespread

- Same bug in SChannel, SecureTransport, IBM JSSE, …
- CVEs for all major libraries and web browsers

A Verified State Machine for OpenSSL



A Verified State Machine for OpenSSL

OpenSSL has two state machines (client/server)

- A bit of a mess: many protocol versions, extensions, optional, and experimental features

We rewrote this code and verified it with Frama-C

- 750 lines of code, 460 lines of specification
- 1 month of a PhD student’s time
- Reused logical specification from miTLS
- Eliminates all state machine bugs in OpenSSL
- No impact on performance.

Conclusions

Cryptographic protocol testing needs work

- We used a specification-driven fuzzing tool to find critical state machine bugs in a number of libraries
- This should be done systematically by developers

Open source code is not immune from attack

- Security bugs can hide in plain sight for years

Verification of production code is feasible

- We focused on the core state machine, one small step towards verifying OpenSSL

Beware of deliberately weakened cryptography

- Backdoors come back to bite you even decades later