Imperfect Forward Secrecy:

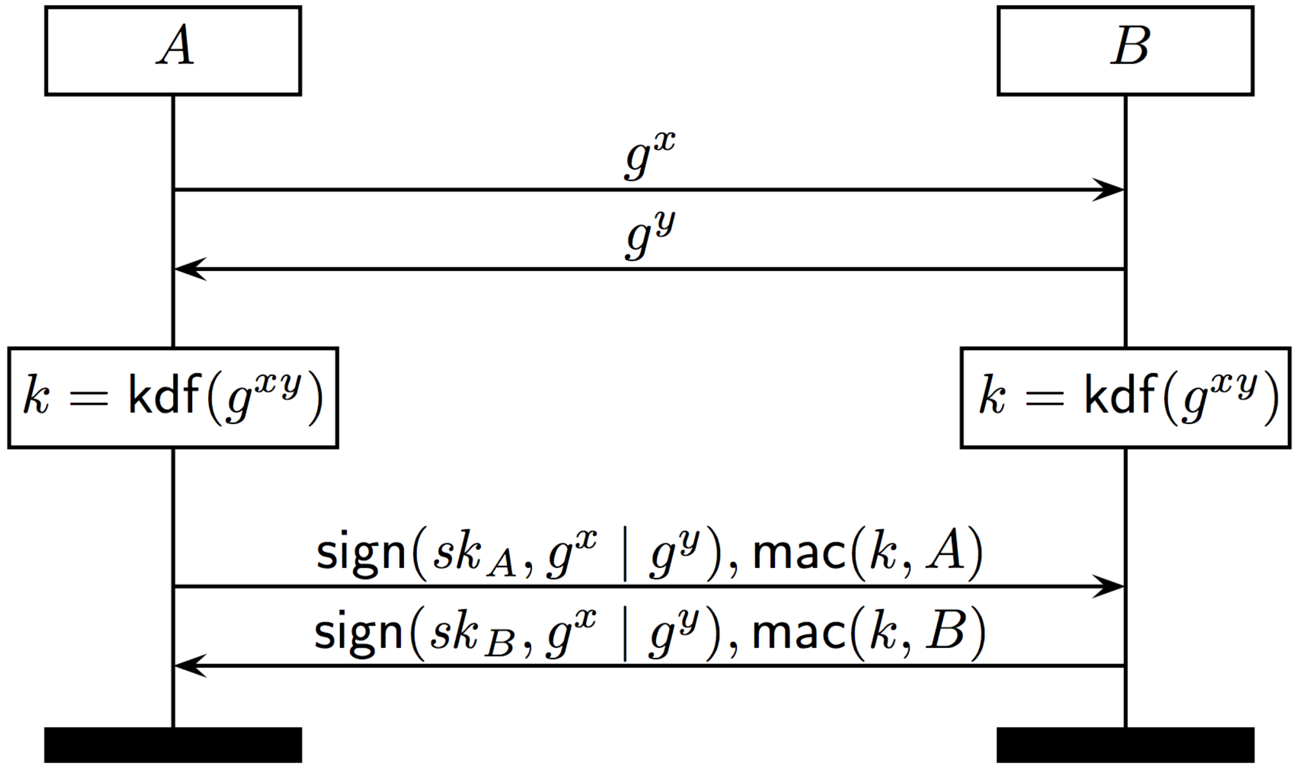
How Diffie-Hellman Fails in Practice

http://weakdh.org
http://mitls.org

Karthikeyan Bhargavan
+ many, many others.
(CNRS, INRIA, Microsoft Research, IMDEA,
Univ of Pennsylvania, Univ of Michigan, Johns Hopkins)

INVENTORS FOR THE DIGITAL WORLD

Authenticated Diffie-Hellman



If deployed correctly, *k* enjoys many properties:

- authenticity, confidentiality, forward secrecy
- + resistance to UKS, KCI? future secrecy?

DH in real-world protocols

Who chooses the group (p,g)?

- client? server? standard writers?

What other protocols are running?

- do they use the same long-term keys (sk_A, sk_B)?

Can the DH key shares be reused?

- do we need to validate public values (g^x, g^y)?

How does the application use k?

- does k need to be unique for each session?

This Talk

DH key exchange is well-understood, but real-world protocols based on DH often broken

- buggy ADH implementations (SKIP)
- weak DH groups (Logjam)
- unexpected security requirements (3Shake)

Understanding protocol details can make DLP-based attacks more practical

Case study: Transport Layer Security (TLS)

- Only modp groups, not elliptic curves

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)

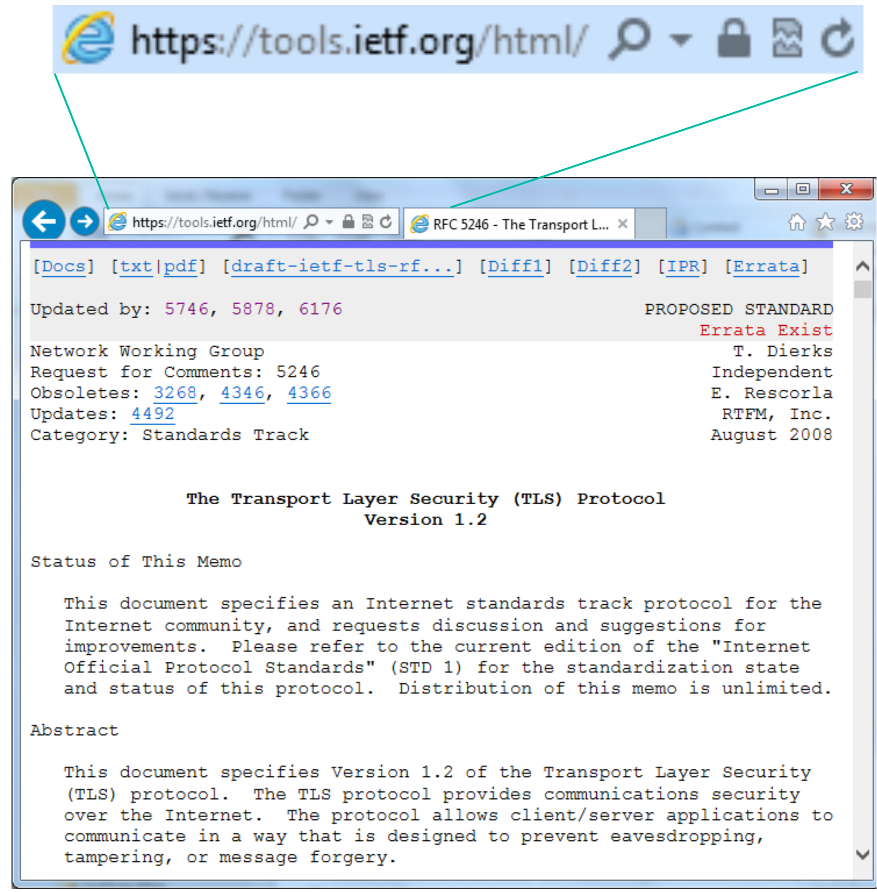
2015 TLS1.3?

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

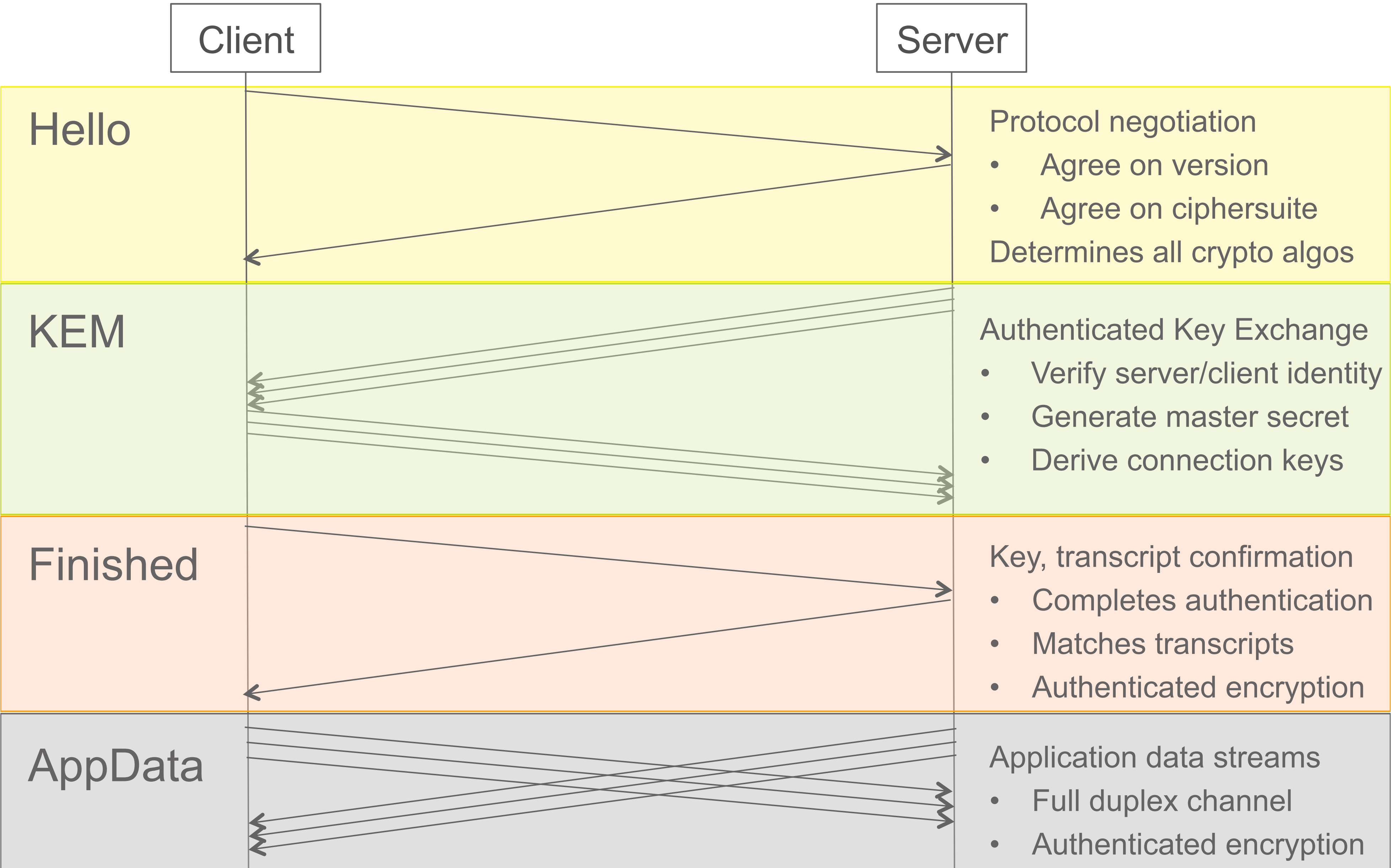
Status of This Memo

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.

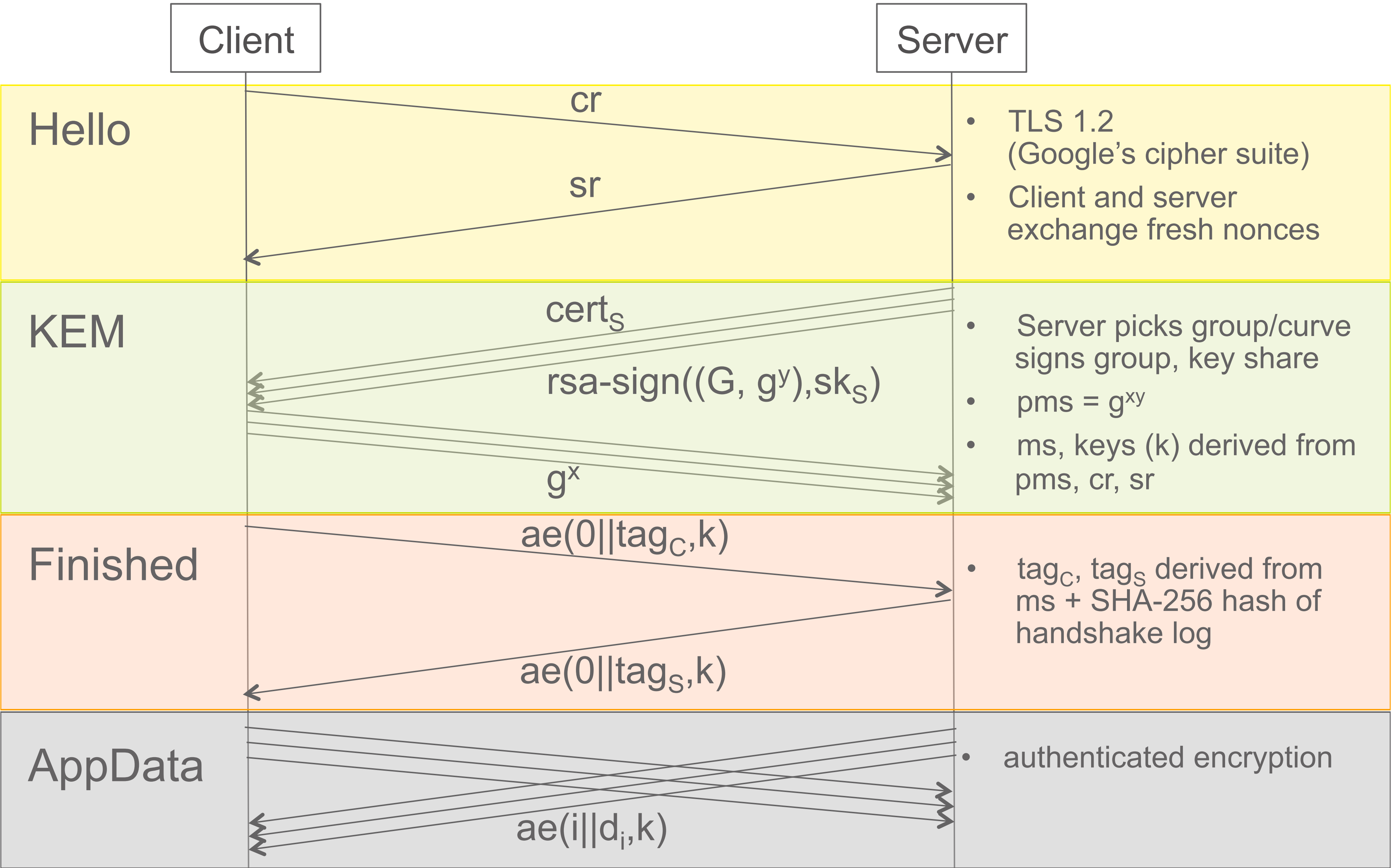
Abstract

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



(EC)DHE Handshake in TLS



DHE in TLS

Who chooses the group (p,g)?

- server sends: $sign(sk_S, cr \mid sr \mid p \mid g \mid g^y)$

What other protocols are running?

- RSA key transport using same (sk_A, sk_B)?

Can the DH key shares be reused?

- yes, and public values are not usually validated

How does the application use k?

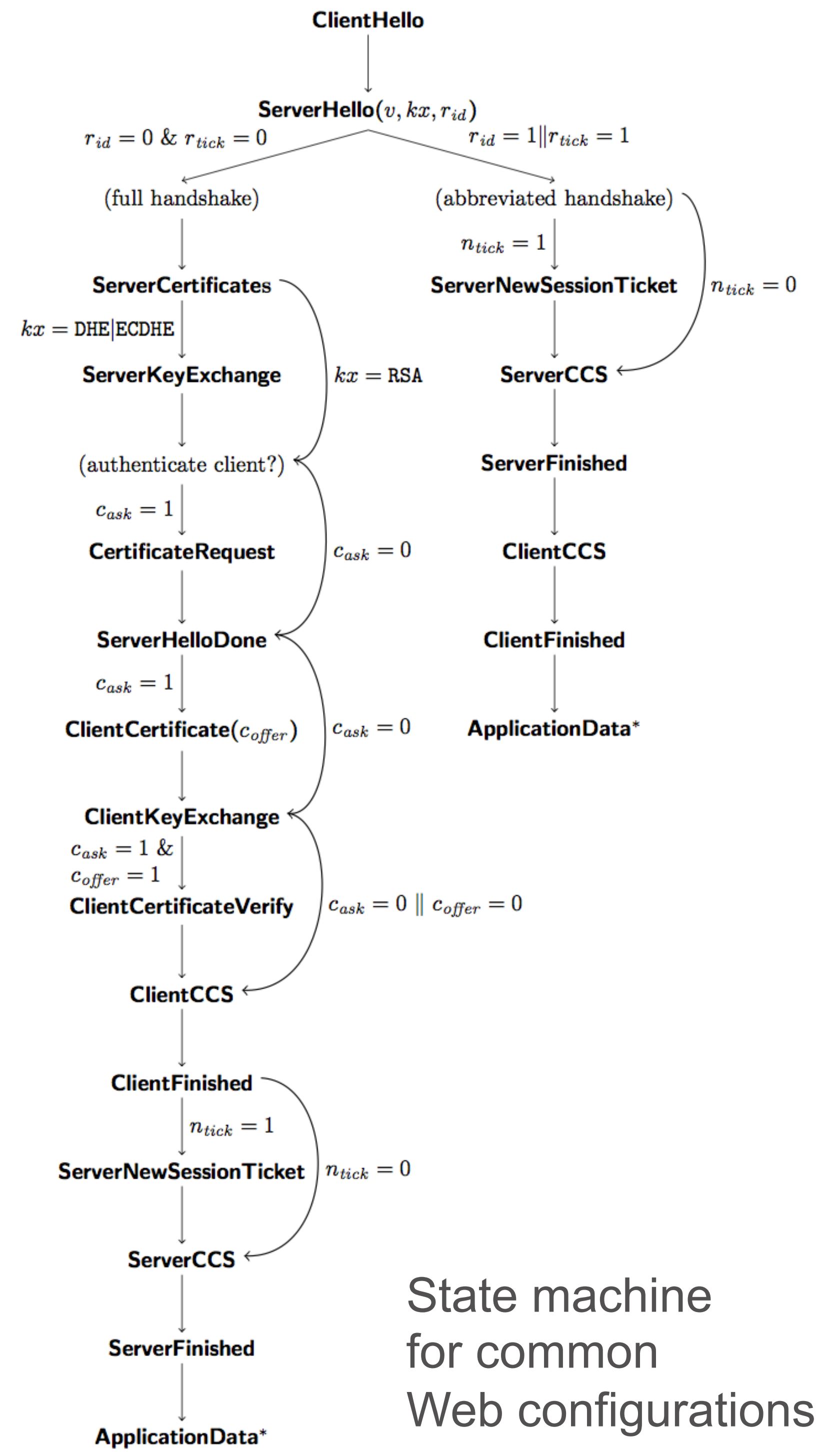
- fast session resumption, unique channel ids, …

TLS State Machine

RSA + DHE + ECDHE
+ Session Resumption
+ Client Authentication

- Covers most features used on the Web
- Already quite a complex combination of protocols!

- Composition proved secure for reference implementation [S&P'13, Crypto'14] [see http://mitls.org]

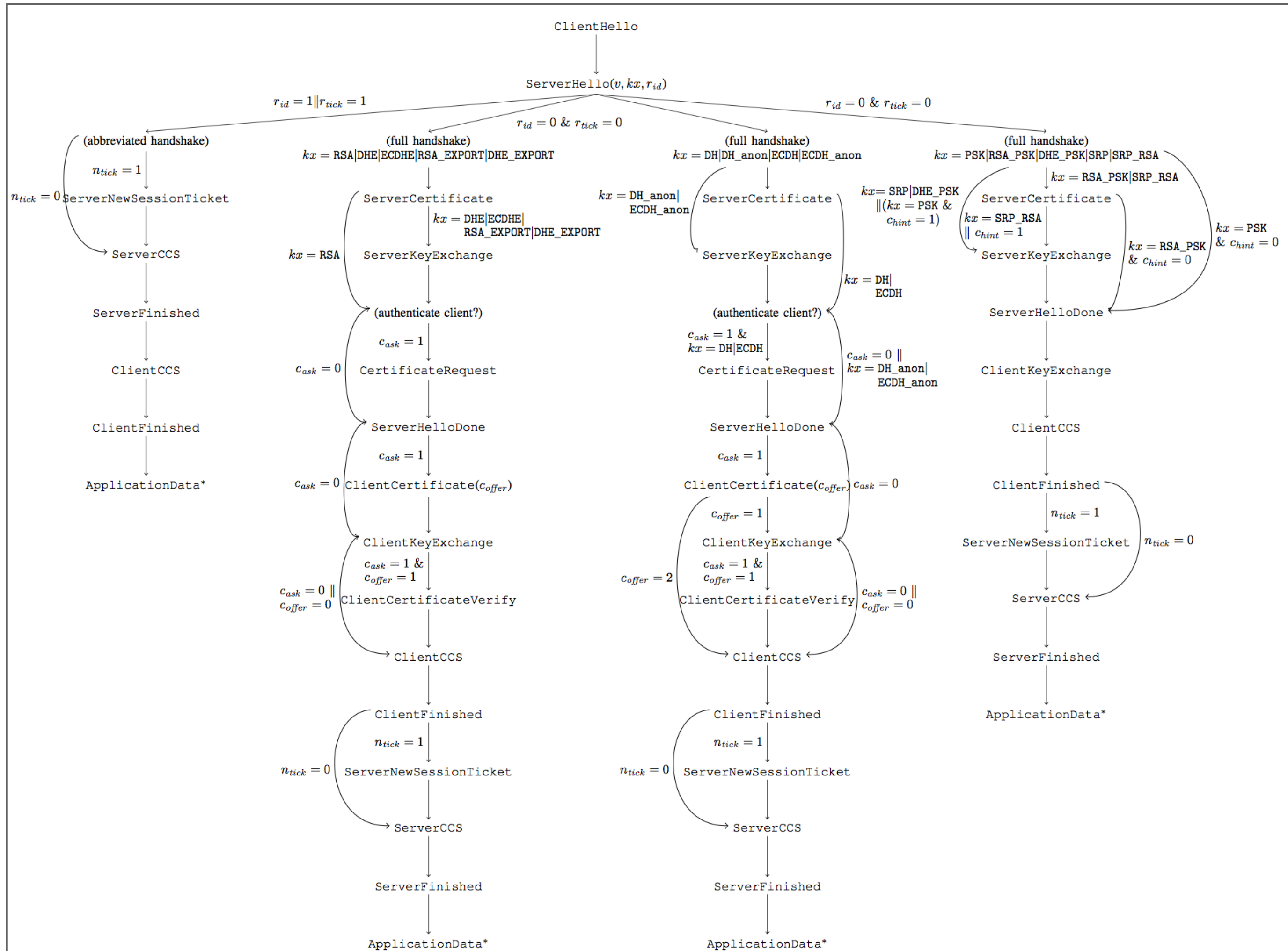


State machine for common Web configurations

Full SSL/TLS State Machine?

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

All implemented in OpenSSL



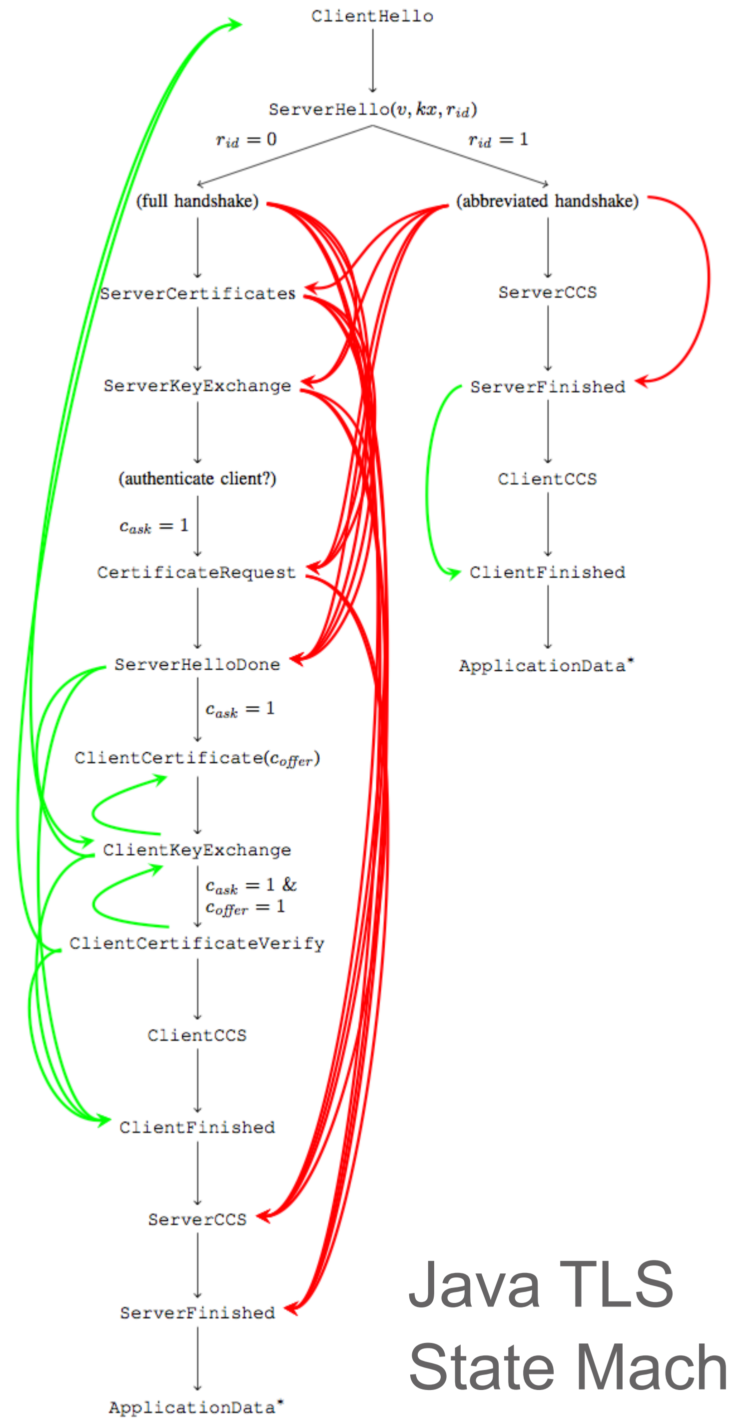
Implementation Bugs

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

- Required messages can be skipped
- Unexpected messages may be received
- CVEs for many libraries

How come all these bugs?

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

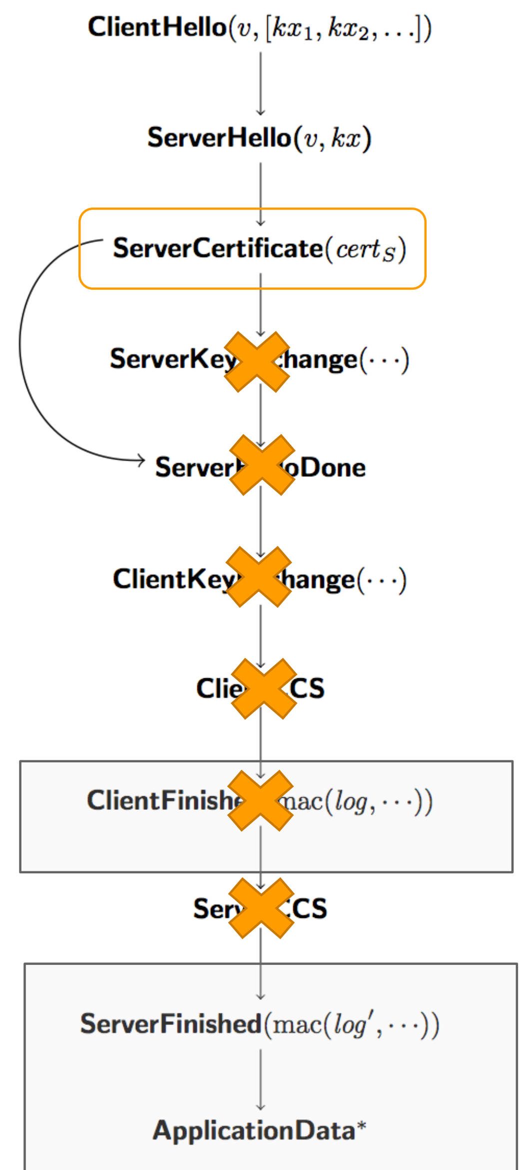


Java TLS State Machine

SKIP Inconvenient Messages

Network attacker impersonates api.paypal.com to a JSSE client

1. Send PayPal's cert
2. SKIP ServerKeyExchange
   bypass server signature:
   $\textit{rsa-sign}(sk_S, cr \mid sr \mid p \mid g \mid g^y)$
3. SKIP ServerCCS
   bypass encryption
4. Send ServerFinished
   using uninitialized MAC key
   bypass handshake integrity
5. Send ApplicationData
   unencrypted as S.com



SKIP Impact

- A network attacker can impersonate *any* server (Paypal, Amazon, Google) to *any* Java TLS client (built with JSSE)
- Affects all versions of Java until Jan 2015 (CVE-2014-6593)
- Similar bugs also found in:
  OpenSSL, wolfSSL, mono TLS, GNU classpath
- Reality check: our efforts in securing ADH can be made irrelevant by ridiculous implementation bugs

Choosing Good DH Groups

TLS-DHE in practice

Internet-wide scan of HTTPS servers (Zmap)

- 14.3M hosts, 24% support DHE
- 70,000 distinct groups *(p,g)*

Composite-order groups with short exponents

- 4,800 groups where *(p-1)/2* was not prime
- Applied ECM to opportunistically factor *(p-1)/2*
- Got prime factors for 750 groups (40K connections)
- Some servers used short exponents: 128/160 bits
- Used Pohlig-Hellman to compute:
  full secret exponent for 159 servers
  (partial exponent for 460 servers)

TLS-DHE in practice

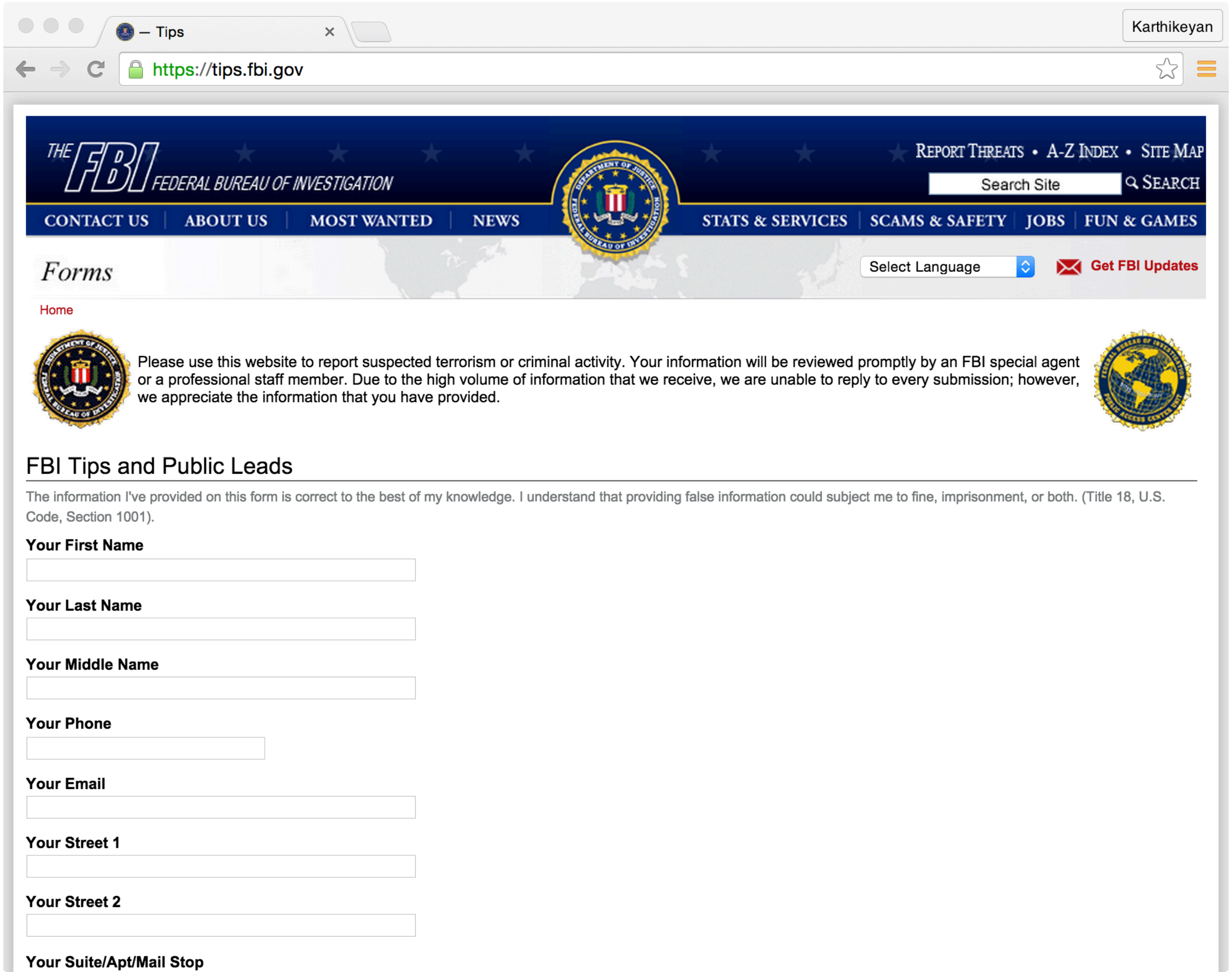
Internet-wide scan of HTTPS servers (using Zmap)

- 14.3M hosts, 24% support DHE
- 70,000 distinct groups *(p,g)*

Small-sized safe primes

- 84% (2.9M) servers use 1024-bit primes
- 2.6% (90K) servers use 768-bit primes
- 0.0008% (2.6K) servers use 512-bit primes
- But 512-bit DLP is solved since 2014,
  so can we break these connections?

Who uses 512-bit DHE?



Export-grade DHE in TLS

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

- DHE_EXPORT: groups limited to 512 bits

DHE_EXPORT deprecated in 2000

- 8.4% (489K) HTTPS servers still support it
- … but only when client asks for it
- Web browsers never negotiate DHE_EXPORT, we should be safe, yes?

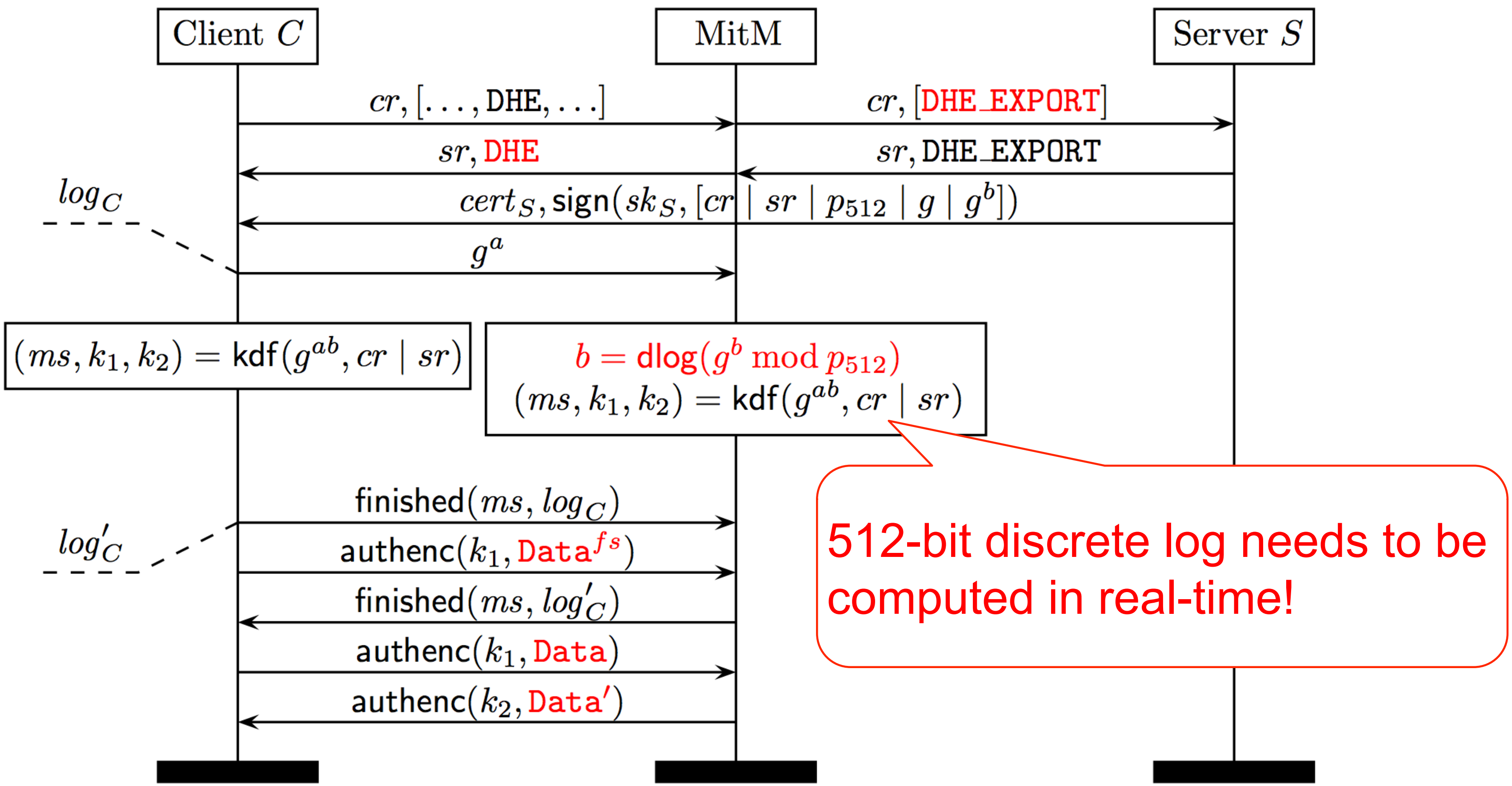
DHE_EXPORT handshake looks just like DHE

- Server uses same long-term signing key for both
- Difference is prime-size, which clients don't check
- Opens the way to a <u>downgrade attack!</u>

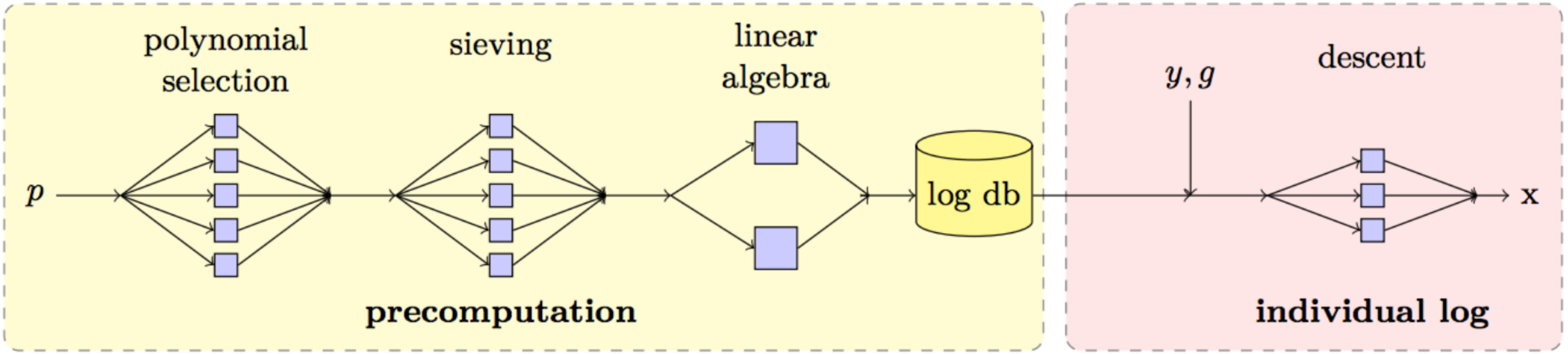
Logjam: Downgrade to DHE_EXPORT

A man-in-the-middle attacker can:

- impersonate ANY server that supports DHE_EXPORT,
- at ANY client that accept 512-bit DHE groups



512-bit Discrete Logs with CADO-NFS



	Sieving			Linear Algebra		Descent
	I	log B	core-years	rows	core-years	core-time
RSA-512	14	29	0.5	4.3M	0.33	
DH-512	15	27	2.5	2.1M	7.7	10 mins

Times for cluster computation:

	polysel	sieving	linalg	descent
	2000-3000 cores		288 cores	36 cores
DH-512	3 hours	15 hours	120 hours	70 seconds

Logjam: Exploiting pre-computation

Most DHE_EXPORT servers use the same groups

- 92% of these use one of two 512-bit primes

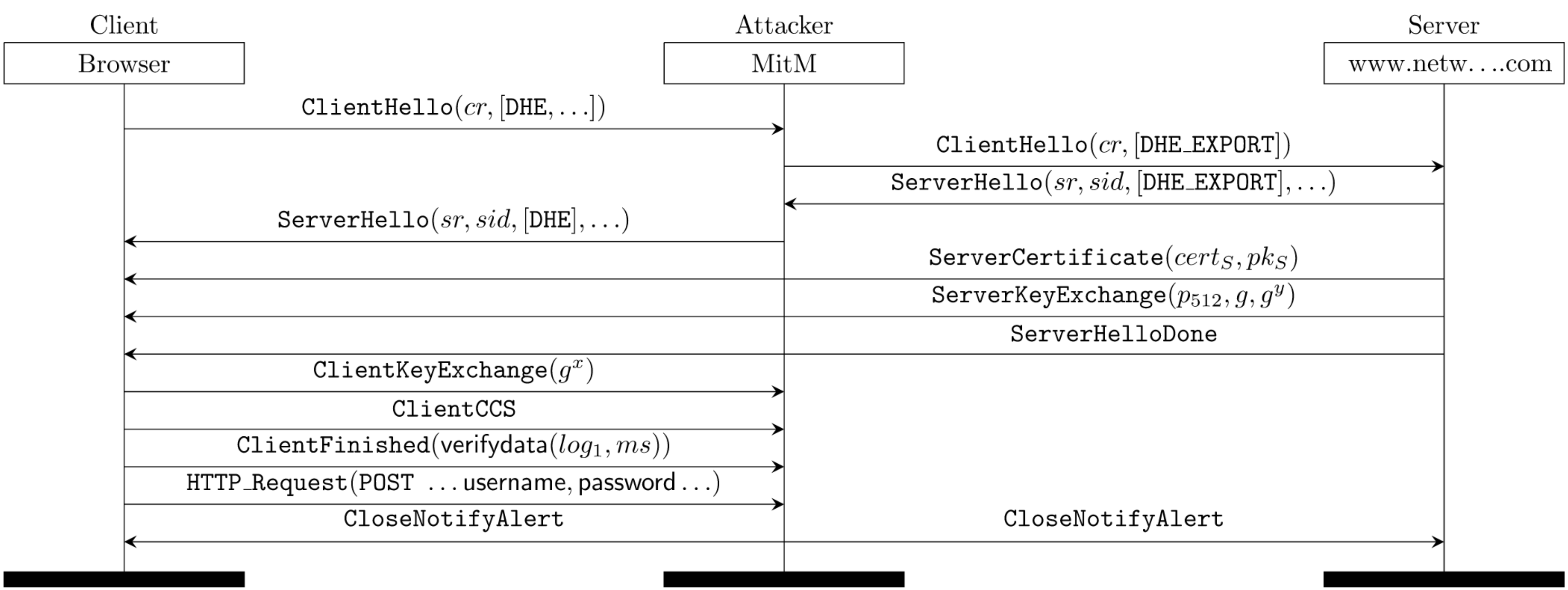
Source	Popularity	Prime
Apache	82 %	9fdb8b8a004544f0045f1737d0ba2e0b 274cdf1a9f588218fb435316a16e3741 71fd19d8d8f37c39bf863fd60e3e3006 80a3030c6e4c3757d08f70e6aa871033
mod_ssl	10%	d4bcd52406f69b35994b88de5db89682 c8157f62d8f33633ee5772f11f05ab22 d6b5145b9f241e5acc31ff090a4bc711 48976f76795094e71e7903529f5a824b
(*other*)	8%	(463 distinct primes)

- We performed pre-computation for these primes
- About 1 week each one 2000-3000 cores
- Per-connection descent computation: 30-150 seconds

Logjam: Exploiting False Start

Some web browsers start sending data too early

- Reason: optimize TLS performance for PFS ciphersuites
- But now no need to wait 150 seconds for DLP
- We can capture this early application data and compute DLP at leisure to read password/cookies



Cost estimates for bigger groups

For DHE_EXPORT connections

- Connections between Chrome/Firefox/IE and 8.4% of websites can be broken offline (no forward secrecy)

For regular DHE, we need to break bigger groups

- For academics, probably needs algorithmic improvements
- For governments, 768 bits is definitely reachable.

	Sieving			Linear Algebra		Descent
	I	log B	core-years	rows	core-years	core-time
RSA-512	14	29	0.5	4.3M	0.33	
DH-512	15	27	2.5	2.1M	7.7	10 mins
RSA-768	16	37	800	250M	100	
DH-768	17	35	8,000	150M	28,500	2 days
RSA-1024	18	42	1,000,000	8.7B	120,000	
DH-1024	19	40	10,000,000	5.2B	35,000,000	30 days

Impact of breaking bigger groups

IKEv1, IKEv2, SSH all use 768-bit/1024-bit groups

- 6% of IKEv2 servers use Oakley 1 (768-bits)
- 64% of IKEv2 servers use Oakley 2 (1024-bits)
- 26% of SSH servers use Oakley 2 (1024-bits)
- 13% of HTTPS servers use 1024-bit Apache group

	If the attacker can precompute for ...			
	all 512-bit groups	all 768-bit groups	one 1024-bit group	ten 1024-bit groups
HTTPS Top 1M w/ active downgrade	45,100 (8.4%)	45,100 (8.4%)	205,000 (37.1%)	309,000 (56.1%)
HTTPS Top 1M	118 (0.0%)	407 (0.1%)	98,500 (17.9%)	132,000 (24.0%)
HTTPS Trusted w/ active downgrade	489,000 (3.4%)	556,000 (3.9%)	1,840,000 (12.8%)	3,410,000 (23.8%)
HTTPS Trusted	1,000 (0.0%)	46,700 (0.3%)	939,000 (6.56%)	1,430,000 (10.0%)
IKEv1 IPv4	–	64,700 (2.6%)	1,690,000 (66.1%)	1,690,000 (66.1%)
IKEv2 IPv4	–	66,000 (5.8%)	726,000 (63.9%)	726,000 (63.9%)
SSH IPv4	–	–	3,600,000 (25.7%)	3,600,000 (25.7%)

Solutions?

Short-term fixes

Security updates to major TLS libraries, web browsers, websites, mail servers, …

- Disabling 512-bit, then 768-bit, then 1024 bit
- We recommend 2048-bit safe primes

Fixes are surprisingly hard to deploy

- Many libraries hard-code DH parameters
- Hardware devices difficult to update
- May be easier to move to ECDHE

A new protocol: TLS 1.3

Stronger key exchanges, fewer options

- ECDHE and DHE by default, no RSA key transport
- Fixed DH groups (> 2047 bits) and EC curves (> 255 bits)
- Only AEAD ciphers (AES-GCM), no CBC, no RC4

Signatures, session keys bound to handshake params

- Server signature covers ciphersuite (preventing Logjam)

Faster: lower latency with 1 round-trip

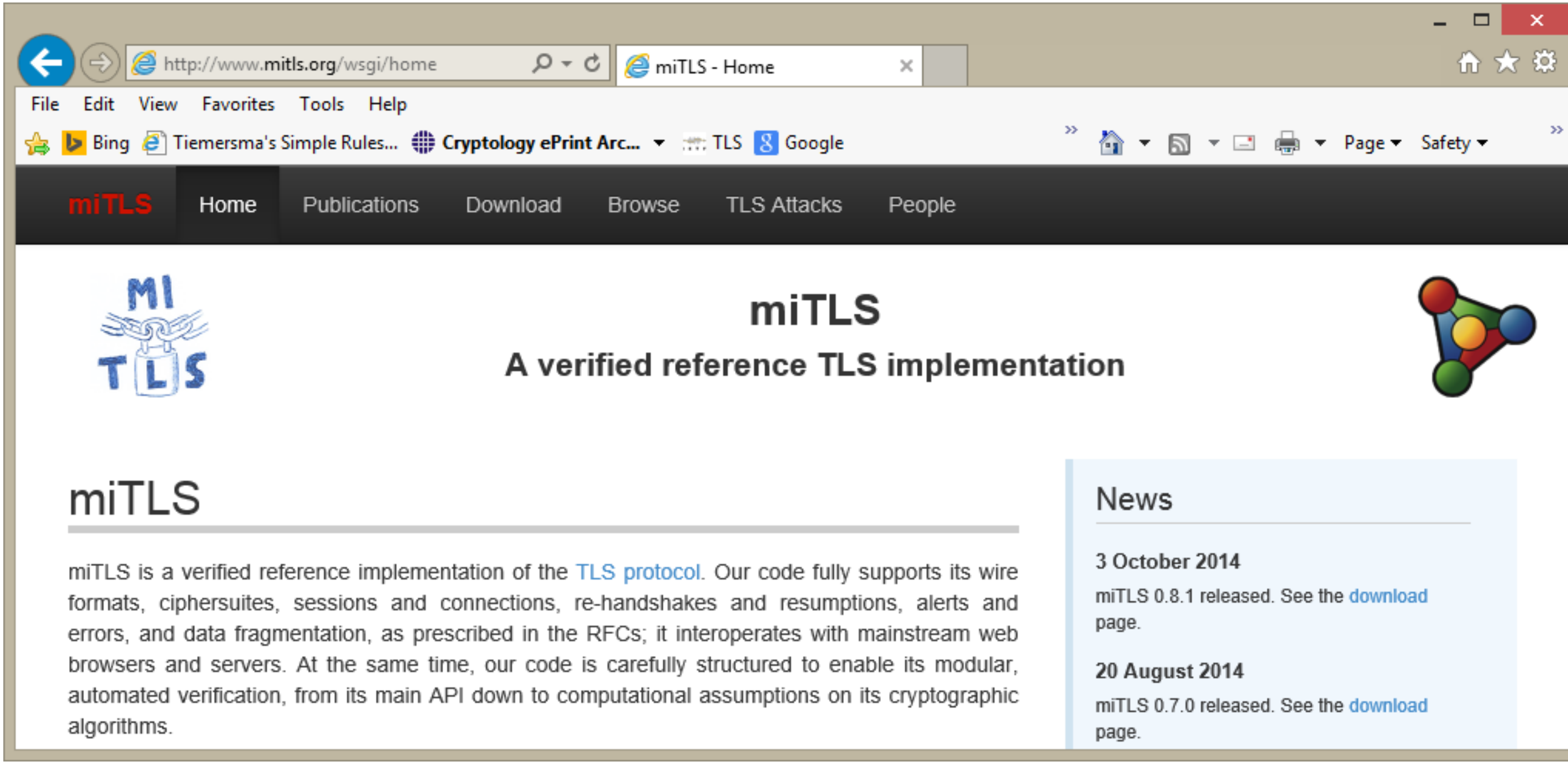
- 0-round trip mode also available
- Many security analyses ongoing (!)

Implementing TLS correctly

Use formal methods!

- Use a type-safe programming language
  - **F#**, OCaml, Java, C#,…
  - (No buffer overruns, no Heartbleed)
- Verify the logical correctness of your code
  - Use a software verifier: **F7/F***, Why3, Boogie, Frama-C,…
- Link software invariants to cryptographic guarantees
  - Use a crypto verifier: **EasyCrypt**, CryptoVerif, ProVerif
  - Hire a cryptographer!

miTLS: a verified implementation



- A strong security theorem links software invariants to standard cryptographic assumptions

Conclusions

Protocols use and compose Diffie-Hellman key exchanges in various (weird) ways

- Complex compositions lead to implementation bugs, downgrade attacks, …

Don’t assume that servers know how to generate good DH groups or keys

- Most don’t validate groups or keys
- Off-curve and small-subgroup attacks are feasible

Beware of cryptographic front-doors (EXPORT)

- Obsolete crypto can bite you decades later

Questions?

weakdh.org

mitls.org

smacktls.com

Papers:

- *Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice.* ACM CCS, 2015
- *A Messy State of the Union: Taming the Composite State Machines of TLS*. IEEE S&P, 2015