The Road to TLS 1.3

Eric Rescorla

Mozilla

ekr@rtfm.com

Overview

- How we got to the point of doing TLS 1.3
- Overview of TLS 1.3
- Interaction with the real world
- What can we learn?

The State of the World in January 2013*

- Universal support of SSLv3, TLS 1.0, 11% support for TLS 1.2
- Nearly all certificates are SHA-1
  - MD5 disabled in clients in 2012
- Plenty of AES-CBC
  - But still lots of RC4
  - Chrome and Firefox don't even support AES-GCM
- Worries about the BEAST attack [DR11]
  - People are recommending switching to RC4
- Renegotiation attack is in the rear view mirror
  - Though almost no deployment of the fixes

*`https://www.ssllabs.com/ssl-pulse/`

TLS WG Charter (ca. 2013)

The primary goals of the WG are to maintain:
- The TLS protocol, RFC 5246;
- The DTLS protocol, draft-ietf-tls-rfc4347-bis.

Significant changes to the protocol, such as a new version 1.3, are not within scope of the working group unless they are explicitly added to the charter.

So TLS 1.2 looks pretty solid

- No big changes on the horizon
- Big challenge is updating algorithms
  - AES-GCM (RC4 attacks still to come)
  - SHA-256 for certs
- ... and I'm mostly talking about how hard it is to change anything

2013!

Rebuilding the airplane in Flight

Eric Rescorla
ekr@rtfm.com

Real World Cryptography 2013

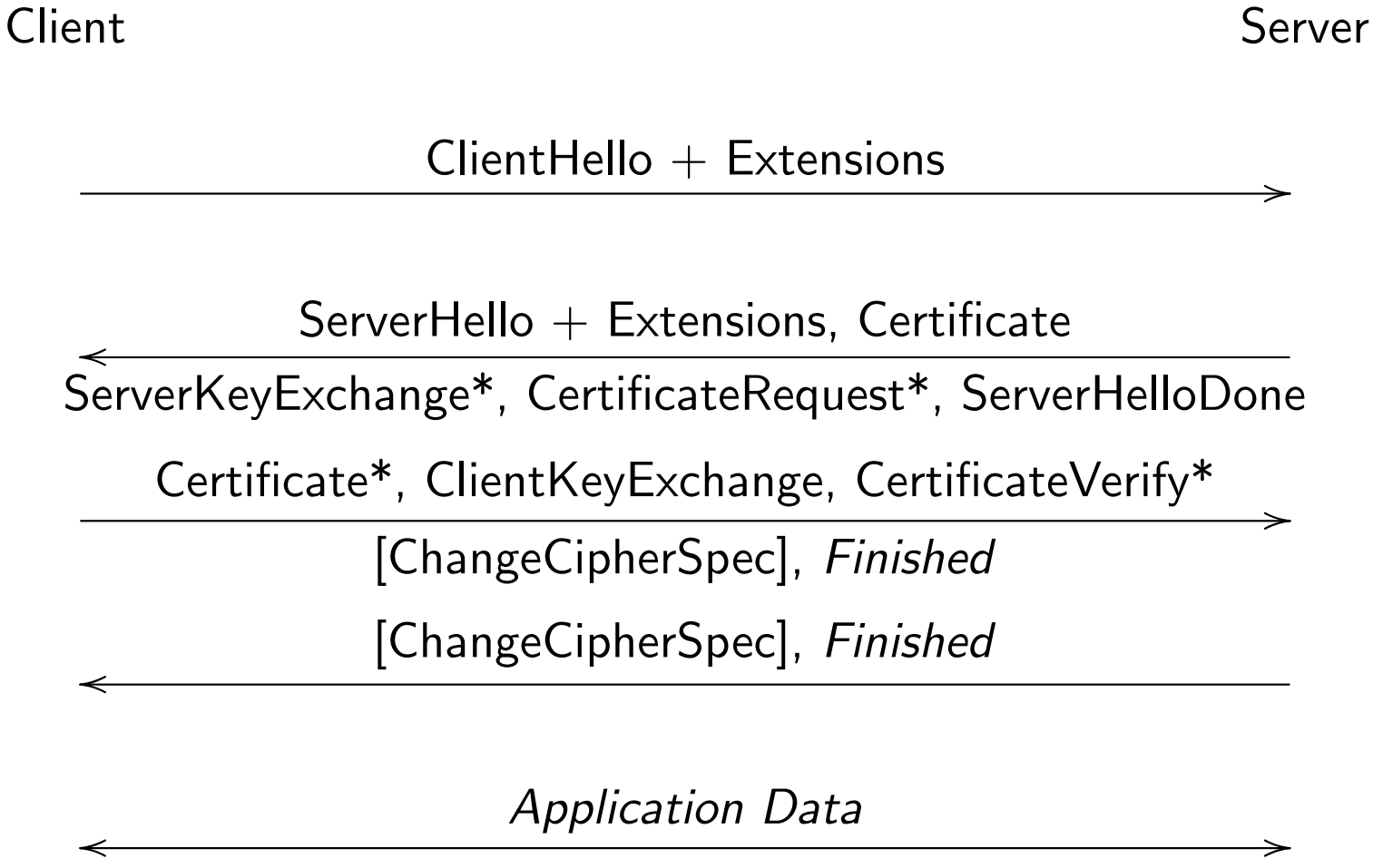
Summary

2013!

- Many of the extension points aren't
  - Code (or standards) which hasn't been tested doesn't work
  - ... any new primitive needs to look exactly like an existing primitive
- Changes in only one side are easier
  - But this generally precludes protocol/algorithm changes
  - And needed anyway to support older peers
- Hard to evaluate the security impact of cryptographic issues
  - Cryptographers tend to work in "abstract" environments
  - The real protocol is more complicated
  - COMSEC engineers don't understand the crypto well enough
- Incentives favor interoperability over security

So what happened?

Reminder: TLS 1.2 Handshake



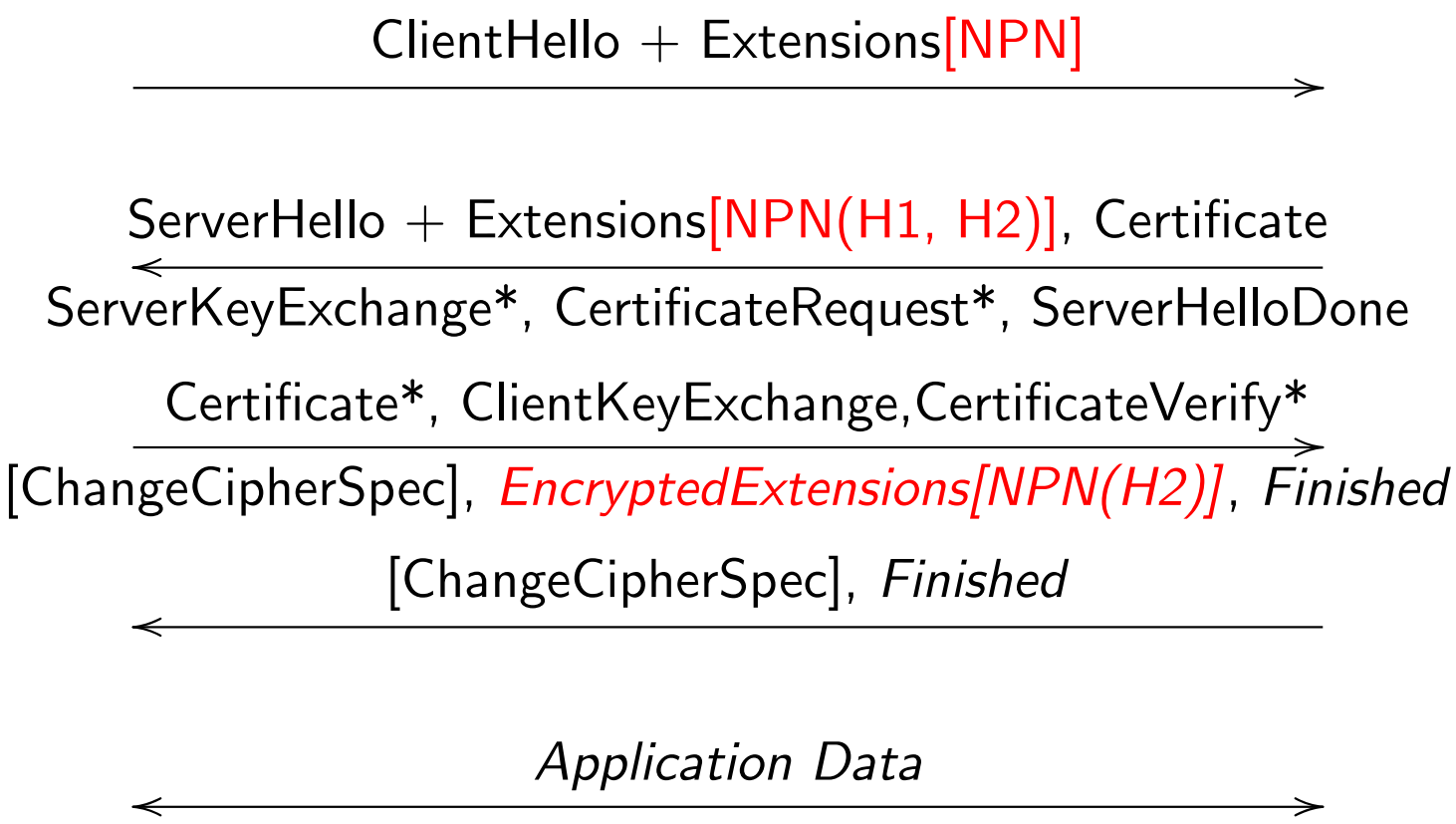
Factor 1: Unencrypted Handshake

- There sure is a lot of stuff in the clear
  - Server identity (Server Name Indication and Certificate)
  - Client identity (if any)
  - Any other extensions
- Repeated proposals to encrypt more of the handshake
  - With various amounts of improvement
  - ... and various degrees of violence to the TLS state machine
  - None really got WG acceptance

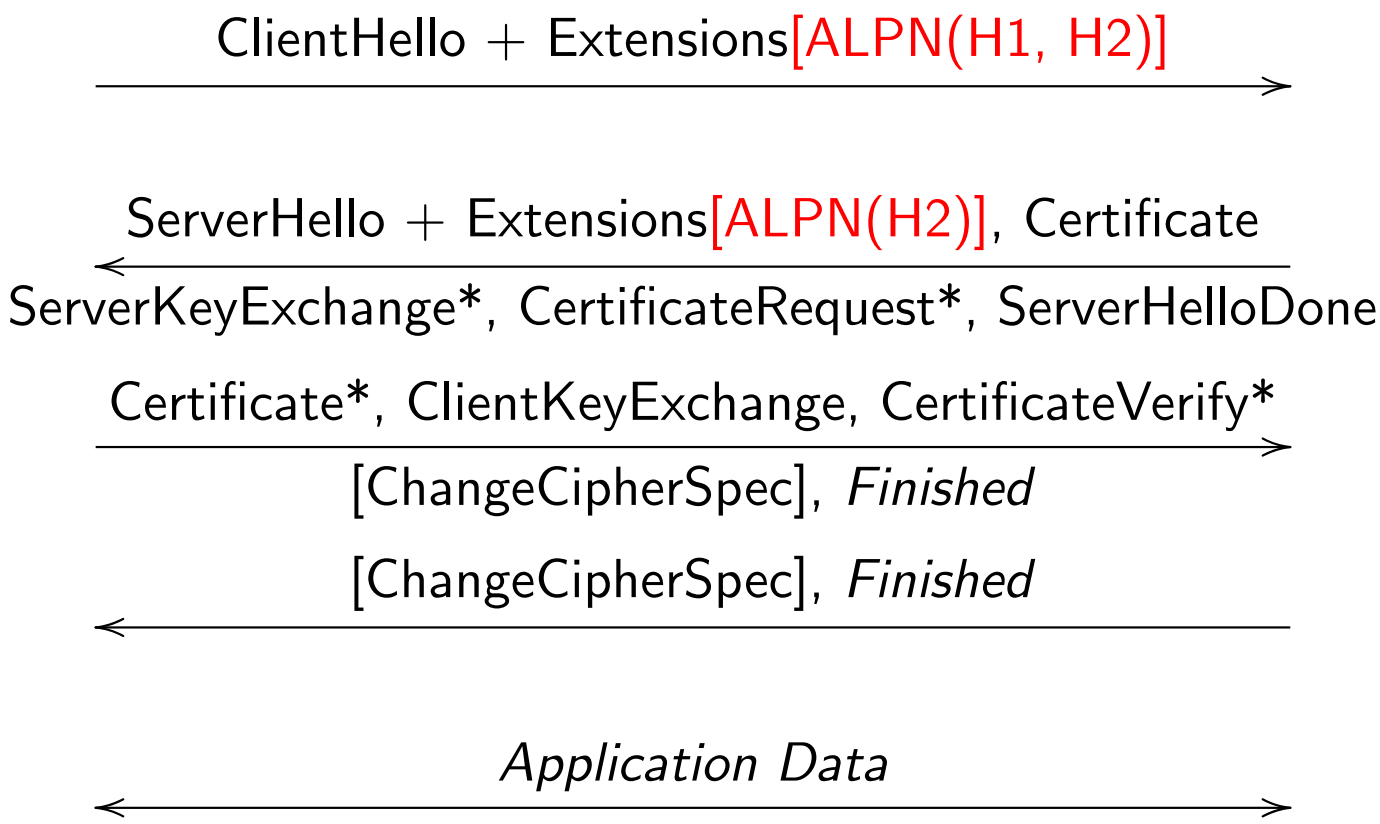
Enter ALPN and NPN

- Background: HTTP/2 negotiation
  - Client supports HTTP/2
  - Knows that the server supports HTTPS but doesn't know if it supports HTTP/2
  - Idea: use TLS handshake to discover this
    - ... without additional round trips
- SPDY initially rolled out with "next protocol negotiation" (NPN)

NPN Overview



ALPN


ClientHello + Extensions[ALPN(H1, H2)]
ServerHello + Extensions[ALPN(H2)], Certificate
ServerKeyExchange*, CertificateRequest*, ServerHelloDone
Certificate*, ClientKeyExchange, CertificateVerify*
[ChangeCipherSpec], Finished
[ChangeCipherSpec], Finished
Application Data

We ended up with ALPN

- It's more TLS-like
  - Client offers/server chooses
  - No extra messages
- But privacy is worse
  - It doesn't protect the selected protocol
- It *was* starting to look like we wanted to encrypt more stuff
  - But we needed a more generic solution

Factor 2: Latency

- Latency is a key performance metric
  - Especially as everything else gets faster
  - It's dominated by round-trip time
- TLS 1.2's best case scenario is 1-RTT
  - With resumption or false start/cut-through
  - Officially 2-RTT for full handshake
- Existing experiments with 0-RTT data [Lan10, HIS+16]
  - Establish context on initial connection
  - In later connections, send data in first flight
- Clear demand for a handshake with less latency

Factor 3: Problems with existing algorithms

- CBC: BEAST, Lucky 13 [AP13]
- RC4: (No cute name) [ABP+13]
- Compression: CRIME [DR12]
- Plus a pile of old/unused algorithms: 3DES, Camellia, SEED, secP256k1, ...

- Strong desire to trim things down

Factor 4: Triple Handshake [BLF+14]*

- First real indication that there were structural problems with the handshake
  - People had mostly filed renegotiation away...
- Very complicated to reason about
- How could we not understand TLS 1.2 after 20+ years?

*Logjam, FREAK, etc. still in the future at this point

The reasons build up...

- We want to make a lot of changes
- We want to remove a lot of stuff
- This is all disruptive
- Time for a new version

Original Goals for TLS 1.3

Clean up: Remove unused or unsafe features

Improve privacy: Encrypt more of the handshake

Improve latency: Target: 1-RTT handshake for naïve clients;
0-RTT handshake for repeat connections

Continuity: Maintain existing important use cases

Revised Goals for TLS 1.3

Clean up: Remove unused or unsafe features

Improve privacy: Encrypt more of the handshake

Improve latency: Target: 1-RTT handshake for naïve clients;
0-RTT handshake for repeat connections

Continuity: Maintain existing important use cases

Security Assurance: Have analysis to support our work

Look, just don't break anything...

1. It *must* be safe to
   - Be a TLS 1.3 server with any client
   - Offer TLS 1.3 to any server
   - Use TLS 1.3 on almost any network*
2. Drop-in for both servers and clients
   - *Must* work with the same certificates
   - Should be able to just update your library
3. Some use cases may require reconfiguration
   - But this needs to be detectable

*Only learned this one later

Removed Features

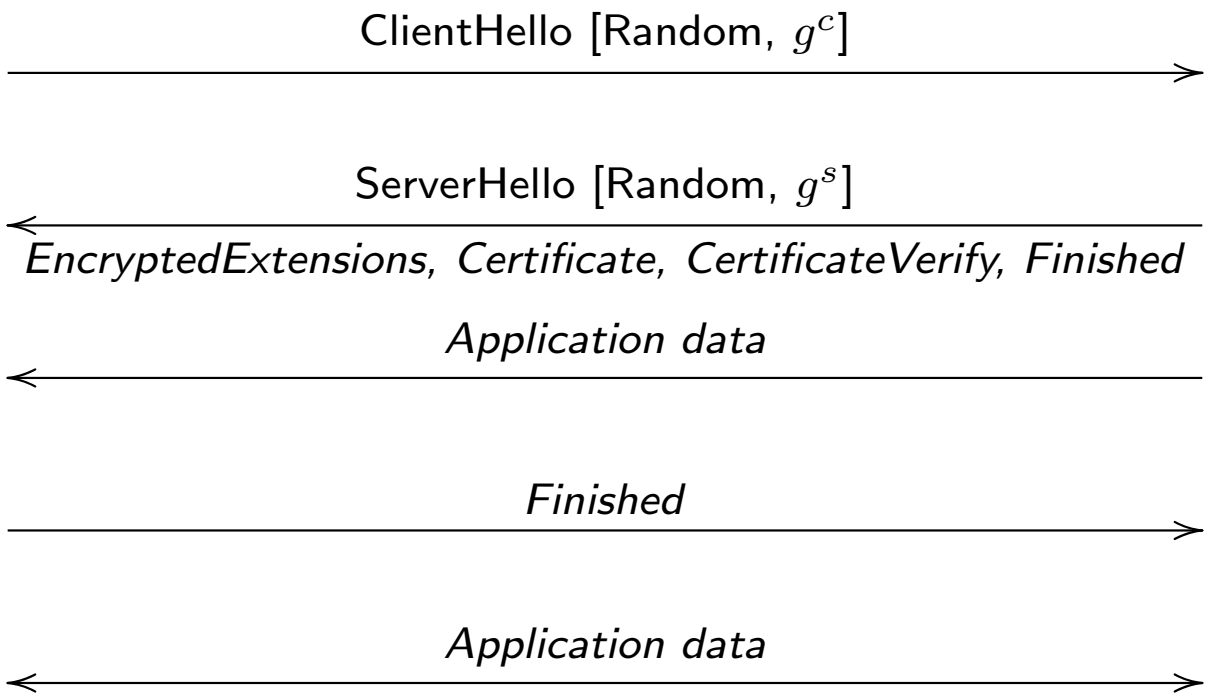
- Static RSA
- Custom (EC)DHE groups
- Compression
- Renegotiation*
- Non-AEAD ciphers
- Simplified resumption

*Special accommodation for inline client authentication

Optimizing Through Optimism

- TLS 1.2 assumed that the client knew nothing
  - First round trip mostly consumed by learning server capabilities
- TLS 1.3 narrows the range of options
  - Only (EC)DHE
  - Limited number of groups
- Client can make a good guess at server's capabilities
  - Pick its favorite groups and send DH share(s)

TLS 1.3 1-RTT Handshake Skeleton

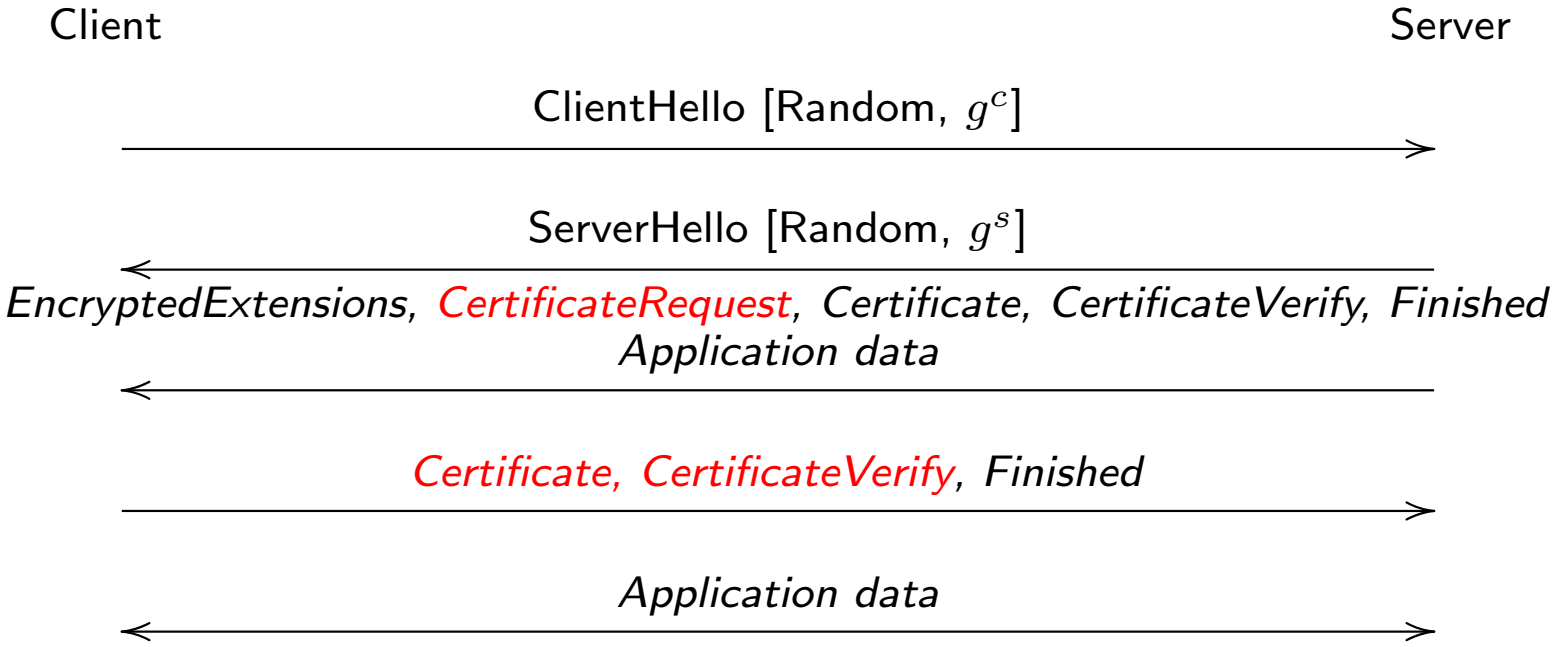


- Server can write on its first flight
- Client can write on second flight
- Keys derived from handshake transcript through server Finished
- Server certificate is encrypted
  - Only secure against passive attackers

Why are we using signatures here?

- Constraint #2: This needs to work with existing certificates
  - Biggest issue for RSA (though ECDSA certificates $\neq$ ECDHE certificates)
- Why not statically sign an (EC)DHE share (cf. QUIC, OPTLSv1 [KW16])?
  - Concerns about bogus signatures
    - Temporary compromise becomes permanent compromise (big deal if the *signing* key is in an HSM)
    - Remote cryptographic attacks as in [JSS15]
  - Concerns about analyzing delegation

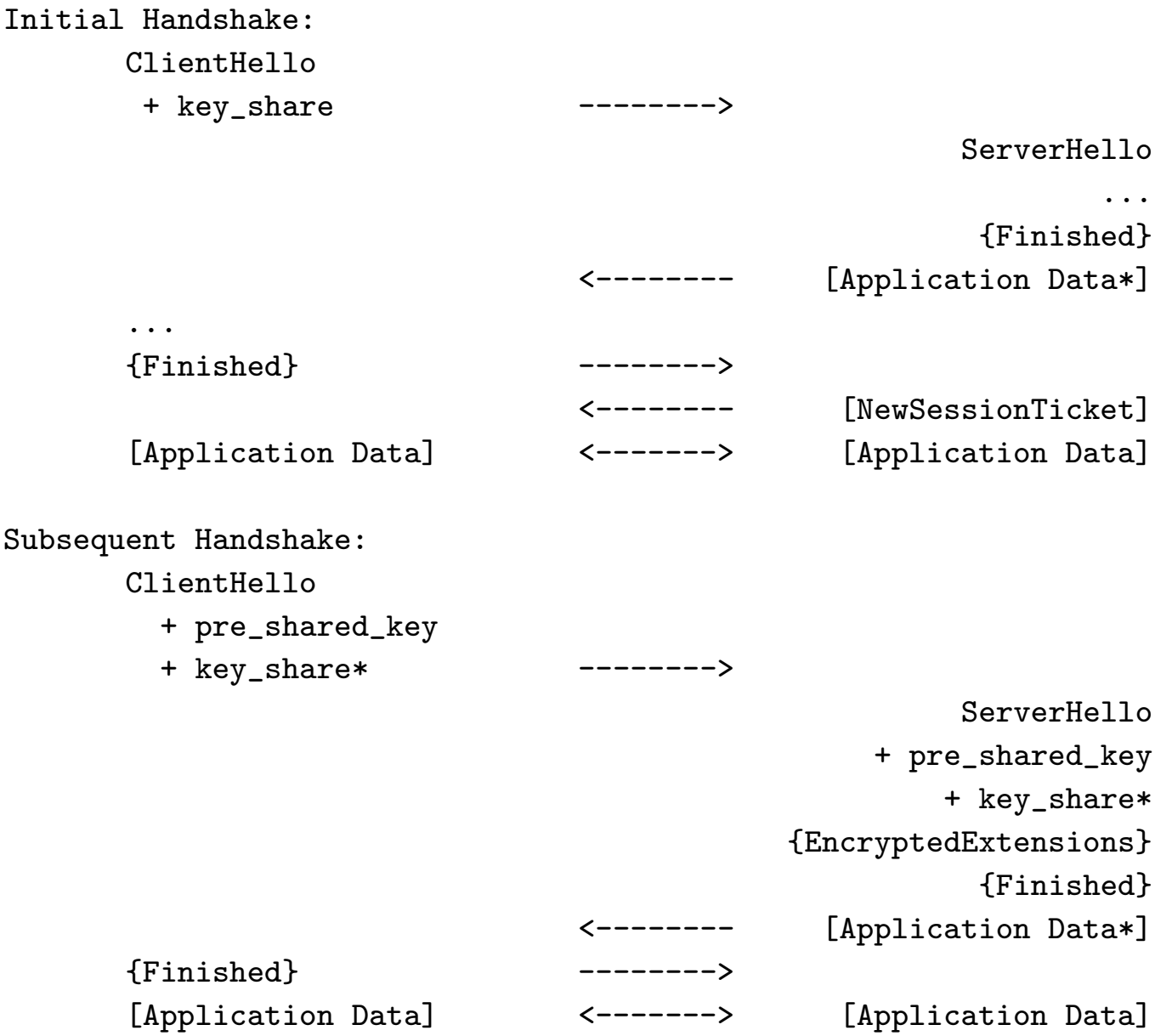
TLS 1.3 1-RTT Handshake w/ Client Authentication Skeleton



- Client certificate is encrypted
- Secure against an active attacker
- Effectively SIGMA [Kra03]

Pre-Shared Keys and Resumption

- TLS 1.2 already supported a Pre-Shared Key (PSK) mode
  - Used for IoT-type applications
- TLS 1.3 merges PSK and resumption
  - Server provides a key label
  - ... bound to a key derived from the handshake
  - Label can be a "ticket" (encryption of the key)
- Two major modes
  - Pure PSK
  - PSK + (EC)DHE

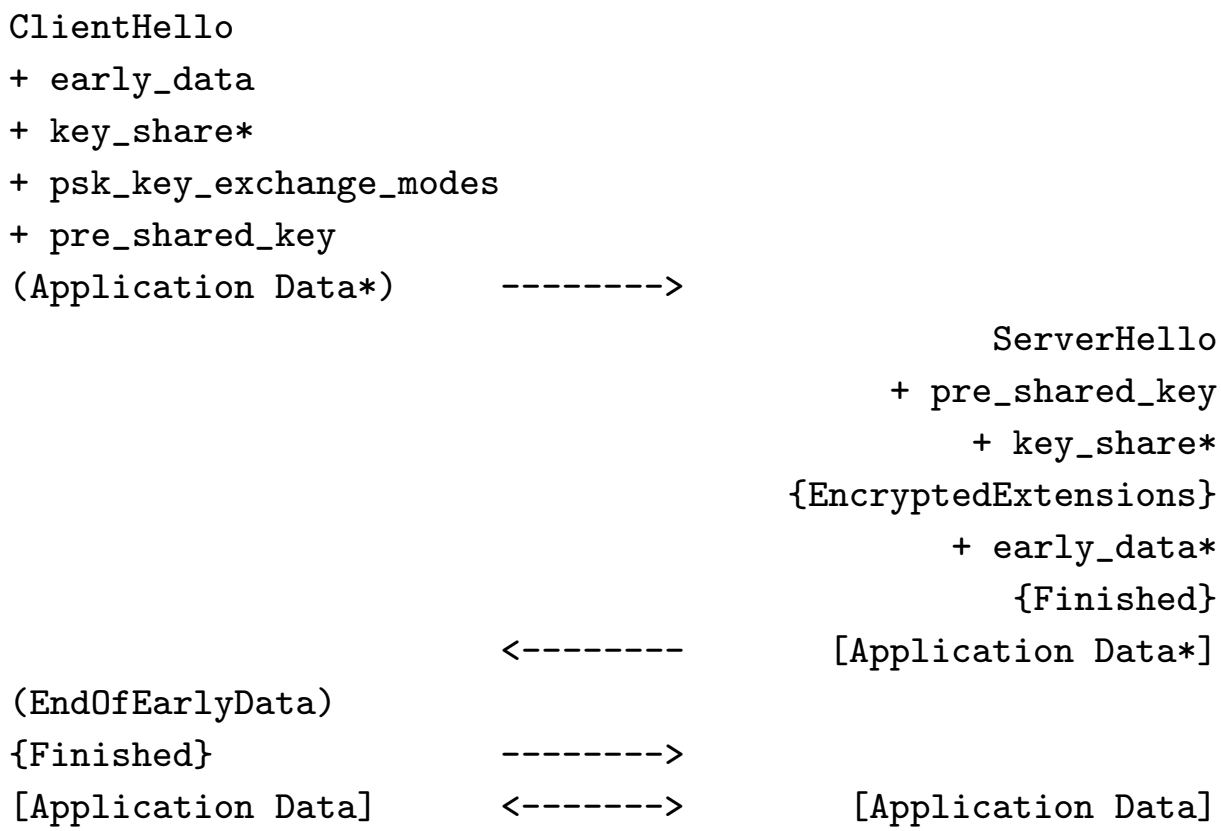

Initial Handshake:
ClientHello
+ key_share
-------->
ServerHello
...
{Finished}
<-------- [Application Data*]
...
{Finished}
-------->
<-------- [NewSessionTicket]
[Application Data] <-------> [Application Data]
Subsequent Handshake:
ClientHello
+ pre_shared_key
+ key_share*
-------->
ServerHello
+ pre_shared_key
+ key_share*
{EncryptedExtensions}
{Finished}
<-------- [Application Data*]
{Finished}
-------->
[Application Data] <-------> [Application Data]

0-RTT Handshake

- Basic observation: once we have established a ticket we have a shared key
  - With someone we have authenticated
- We can send *application data* on the first flight

- TLS 1.3 used to have a DH-based 0-RTT mode
  - Got stripped out due to academic and implementor feedback

TLS 1.3 0-RTT Handshake Skeleton



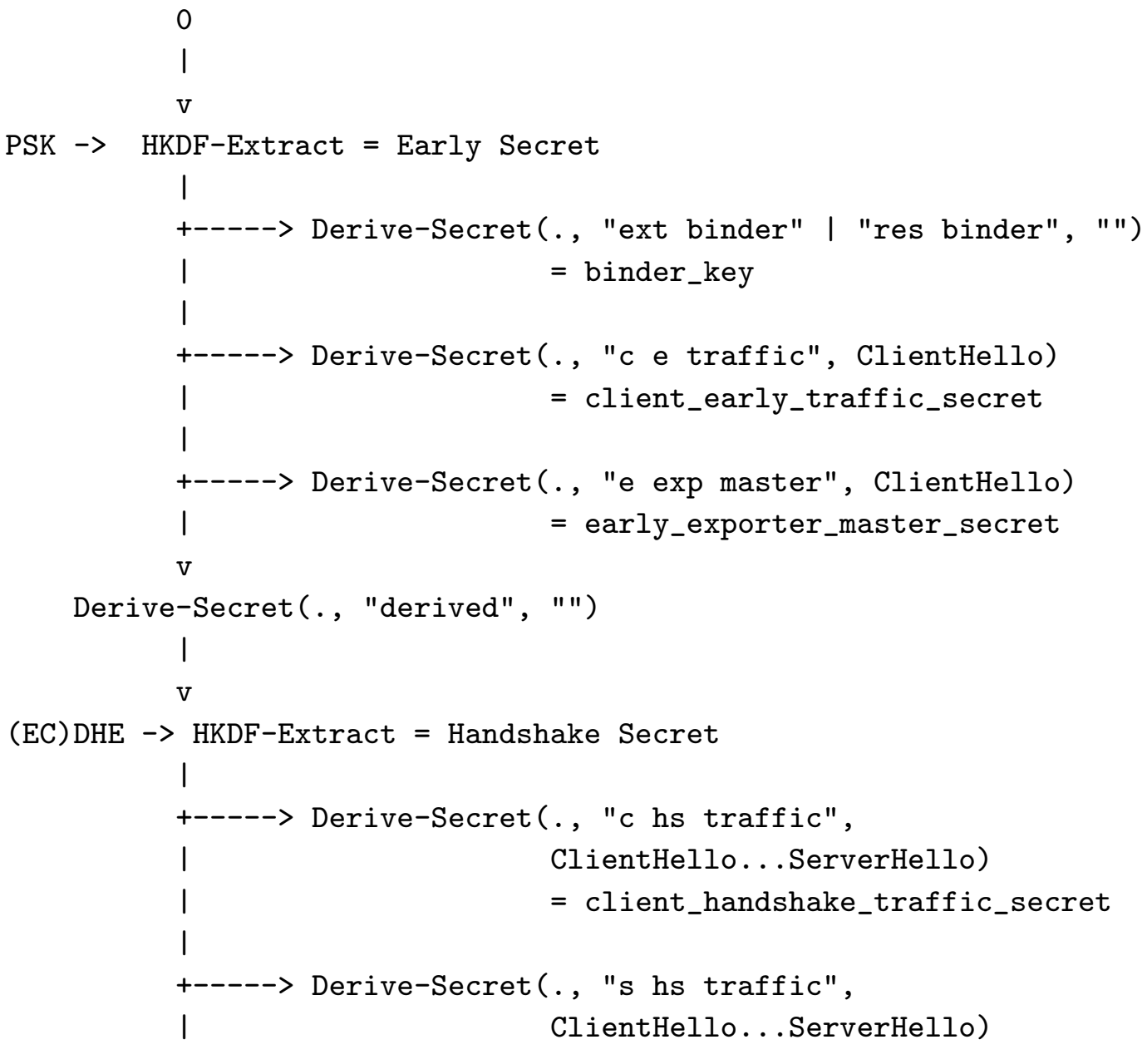
Original Anti-Replay Plan (borrowed from Snap Start)

- Server needs to keep a list of client nonces
- Indexed by a server-provided context token
- Client provides a timestamp so server can maintain an anti-replay window

- Unfortunately, this doesn't work...

Oops...

- The real problem is multiple data centers
- This is a distributed state problem
  - It's broken in QUIC (both versions) and Snap Start too
- Resolution: mostly don't try
  - Only use 0-RTT client data for "safe" requests (GETs)
  - Encourage people to use anti-replay techniques
  - But too big a win not to do
  - Discourage libraries from enabling by default
  - Difficult application integration issue


0
|
v
PSK -> HKDF-Extract = Early Secret
|
+-----> Derive-Secret(., "ext binder" | "res binder", "")
| = binder_key
|
+-----> Derive-Secret(., "c e traffic", ClientHello)
| = client_early_traffic_secret
|
+-----> Derive-Secret(., "e exp master", ClientHello)
| = early_exporter_master_secret
v
Derive-Secret(., "derived", "")
|
v
(EC)DHE -> HKDF-Extract = Handshake Secret
|
+-----> Derive-Secret(., "c hs traffic",
| ClientHello...ServerHello)
| = client_handshake_traffic_secret
|
+-----> Derive-Secret(., "s hs traffic",
| ClientHello...ServerHello)

```
          |                     = server_handshake_traffic_secret
          v
    Derive-Secret(., "derived", "")
          |
          v
0 -> HKDF-Extract = Master Secret
          |
          +-----> Derive-Secret(., "c ap traffic",
          |                     ClientHello...server Finished)
          |                     = client_application_traffic_secret_0
          |
          +-----> Derive-Secret(., "s ap traffic",
          |                     ClientHello...server Finished)
          |                     = server_application_traffic_secret_0
          |
          +-----> Derive-Secret(., "exp master",
          |                     ClientHello...server Finished)
          |                     = exporter_master_secret
          |
          +-----> Derive-Secret(., "res master",
                                ClientHello...client Finished)
                                = resumption_master_secret
```

Packet Format

TLS 1.2 Packet Layout

TLS 1.3 Packet Layout

Traffic Analysis Defenses

- TLS 1.2 is very susceptible to traffic analysis
  - Content "type" in the clear
  - Packet length has minimal padding
    - 0-255 bytes in block cipher modes
    - No padding in stream and AEAD modes
- TLS 1.3 changes
  - Content type is encrypted
  - Arbitrary amounts of padding allowed
  - ... but it's the application's job to set padding policy

The role of analysis

- Find problems early
- Get confidence in the security of the various designs
- Shape the protocol so that analysis is easier

Example: PSK and Client Authentication

- What happens when you combine PSK and client auth?
- This is something you want to work but we hadn't put in the spec
  - Idea is to add client authentication to "resumed" sessions
  - In TLS 1.2, this is done with renegotiation
- Naïve design: just send Certificate, CertificateVerify, Finished
  - In draft-10, client didn't sign over server Finished
  - ... no binding to previous handshakes $\rightarrow$ Attack![CHvdMS]
- Resolution: sign over server Finished
- Supported by analysis [Kra16, CHSvdM16]

- **Lesson: Get analysis for everything**

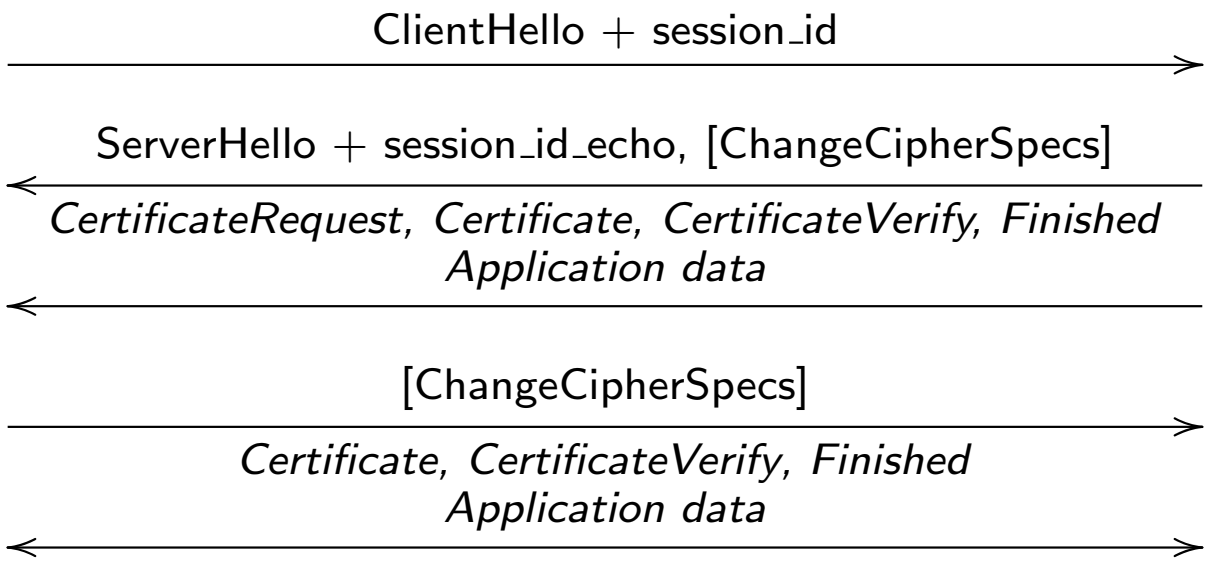
Example: Key Separation

- TLS 1.2 uses the same key for Finished and for application data
  - This causes huge problems for a compositional analysis of the handshake and the record layer [KPW13]
  - Number one request from cryptographers to fix...
- TLS 1.3 uses separate keys for handshake and application layer traffic
  - Also allows us to derive the application keys from more of the handshake
- But not complete separation
  - NewSessionTicket is encrypted with traffic keys
  - ... key separation here was too hard to make work
- **Lesson: Protocol engineering involves compromise**

The Great Middlebox Mess

- Some middleboxes break when you negotiate TLS 1.3
- Error rates (Firefox Beta versus Cloudflare)
  - 2.2% for TLS 1.2
  - 3.9% for TLS 1.3
- What's happening?
  - They're trying to look at handshake details
  - Even when they don't know the version
- This means you need fallback to deploy TLS 1.3
- ... which also breaks anti-downgrade
- Only found this out right when everything else was done
  - Only see it when you try to deploy

The fix: TLS 1.3 looks like TLS 1.2 Resumption



- CCS is just a dummy and doesn't affect the state machine
  - Recipient ignores it
- Middlebox expects everything after CCS to be encrypted
  - And doesn't try to look inside
- This gives comparable error rates between 1.2 and 1.3 $\rightarrow$ No fallback
- **Lesson: sometimes protocol engineering requires big compromises***

*And delays

Static RSA, Passive Inspection, and You

- A lot of enterprises do TLS passive inspection
  - Inspection box attached to a span port
  - You give the RSA private key to the inspection box
  - Decrypt the EPMS and hence the whole connection?*
- TLS 1.3 breaks this (no static RSA)
- Lot of requests from enterprises to do something
  - But we didn't.
  - (they don't really need our help)
- **Lesson: sometimes protocol engineering requires not compromising**

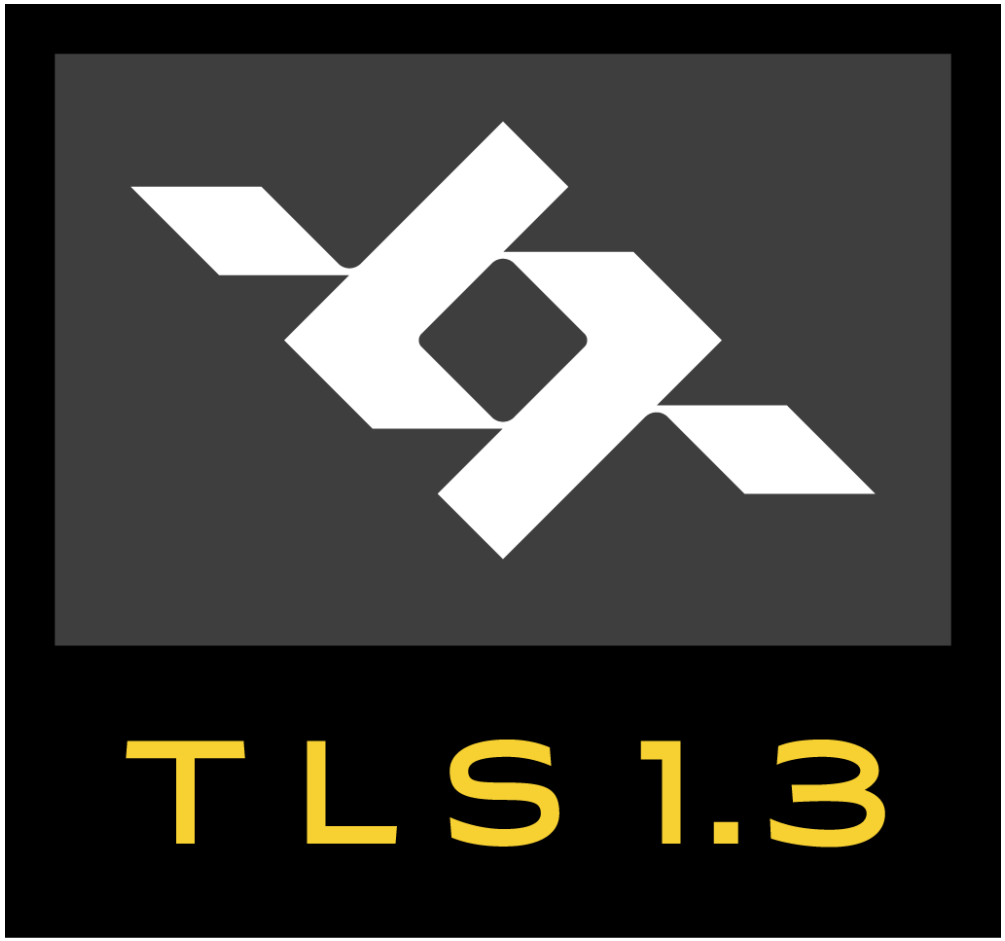
*Don't forget to disable (EC)DHE cipher suites

Where are we now

- RFC Published August 10
- Browsers: Firefox, Chrome, Safari (off by default)
- Server operators: Akamai, Cloudflare. Facebook, Google
- Libraries: OpenSSL, BoringSSL, NSS, Fizz, PicoTLS, ...
- $5+\%$ of Firefox connections
- $> 50\%$ of Facebook connections!

Lessons?

- First major security protocol to be co-designed by standards, implementation, and academic communities
- Successes
  - Got a protocol we can mostly analyze
  - Design largely informed by specific analysis
  - Lots of results already published
- Points of friction
  - Time scale of analysis versus design
  - "Semantic gap" between the communities
  - Engineering compromises
- Challenges for the future
  - How do we incrementally improve TLS (ESNI, subcerts, etc.)?
  - Applying this to a new protocol (MLS/instant messaging).



Thank you

References

[ABP+13] Nadhem J AlFardan, Daniel J Bernstein, Kenneth G Paterson, Bertram Poettering, and Jacob CN Schuldt. On the Security of RC4 in TLS. In *USENIX Security*, pages 305–320, 2013.

[AP13] N AlFardan and Kenneth G Paterson. Lucky 13: Breaking the TLS and DTLS record protocols. In *IEEE Symposium on Security and Privacy*, 2013.

[BLF+14] Karthikeyan Bhargavan, Antoine Delignat Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre Yves Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over tls. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 98–113. IEEE, 2014.

[CHSvdM16] C. Cremers, M. Horvat, S. Scott, and T. v. d. Merwe. Automated analysis and verification of tls 1.3: 0-rtt, resumption

and delayed authentication. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 470–485, May 2016.

[CHvdMS] Cas Cremers, Marko Horvat, Thyla van der Merwe, and Sam Scott. Revision 10: possible attack if client authentication is allowed during PSK. `https://www.ietf.org/mail-archive/web/tls/current/msg18215.html`.

[DR11] Thai Duong and Juliano Rizzo. Beast-here come the xor ninjas, 2011.

[DR12] Thai Duong and Juliano Rizzo. The crime attack. In *Presentation at ekoparty Security Conference*, 2012.

[HIS+16] Ryan Hamilton, Janardhan Iyengar, Ian Swett, Alyssa Wilk, et al. Quic: A udp-based secure and reliable transport for http/2. *IETF, draft-tsvwg-quic-protocol-02*, 2016.

[JSS15] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. On the security of tls 1.3 and quic against weaknesses in pkcs#1 v1.5

encryption. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1185–1196, New York, NY, USA, 2015. ACM.

[KPW13] Hugo Krawczyk, Kenneth G Paterson, and Hoeteck Wee. On the security of the tls protocol: A systematic analysis. In *Advances in Cryptology–CRYPTO 2013*, pages 429–448. Springer, 2013.

[Kra03] Hugo Krawczyk. Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike protocols. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 400–425, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Kra16] Hugo Krawczyk. A unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in tls 1.3). Cryptology ePrint Archive, Report 2016/711, 2016. `https://eprint.iacr.org/2016/711`.

[KW16] Hugo Krawczyk and Hoeteck Wee. The optls protocol and tls 1.3. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 81–96. IEEE, 2016.

[Lan10] Adam Langley. Transport Layer Security (TLS) Snap Start. Internet-Draft draft-agl-tls-snapstart-00, Internet Engineering Task Force, June 2010. Work in progress.