# TLS 1.3

Eric Rescorla

Mozilla

ekr@rtfm.com

# Overview

- Background/Review of TLS
- Some problems with TLS 1.2
- Objectives for TLS 1.3
- What does TLS 1.3 look like?
- Open issues/schedule/etc.

# What is Transport Layer Security?

- Probably the Internet's most important security protocol
- Designed over 20 years ago by Netscape for Web transactions
  - Back then, called Secure Sockets Layer
- But used for just about everything you can think of
  - HTTP
  - SSL-VPNs
  - E-mail
  - Voice/video
  - IoT
- Maintained by the Internet Engineering Task Force
  - We're now at version 1.2

# A Secure Channel

- Client connects to a known server (e.g., it has the domain name)
- Server is (almost) always authenticated by TLS
- Client may or may not be authenticated by TLS
  - Often authenticated by the application, e.g., with a password
- After setup, data is encrypted and authenticated
  - Though what “authenticated” means to the server is fuzzy

# TLS Structure

- Handshake protocol
  - Establish shared keys (typically using public key cryptography)
  - Negotiate algorithms, modes, parameters
  - Authenticate one or both sides
- Record protocol
  - Carry individual messages
  - Protected under symmetric keys
- This is a common design (SSH, IPsec, etc.)

# TLS 1.2: RSA Handshake Skeleton

Client | Server
---|---

ClientHello [Random]

$\longrightarrow$

ServerHello [Random], Certificate

$\longleftarrow$

E($K_s$, Master Secret), *Finished=MAC(MS, Handshake)*

$\longrightarrow$

*Finished=MAC(MS, Handshake)*

$\longleftarrow$

*Application data*

$\longleftrightarrow$

# More on Negotiation

- ClientHello contains more than just random values

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-2>;
    CompressionMethod compression_methods<1..2^8-1>;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16-1>;
    };
} ClientHello;
```

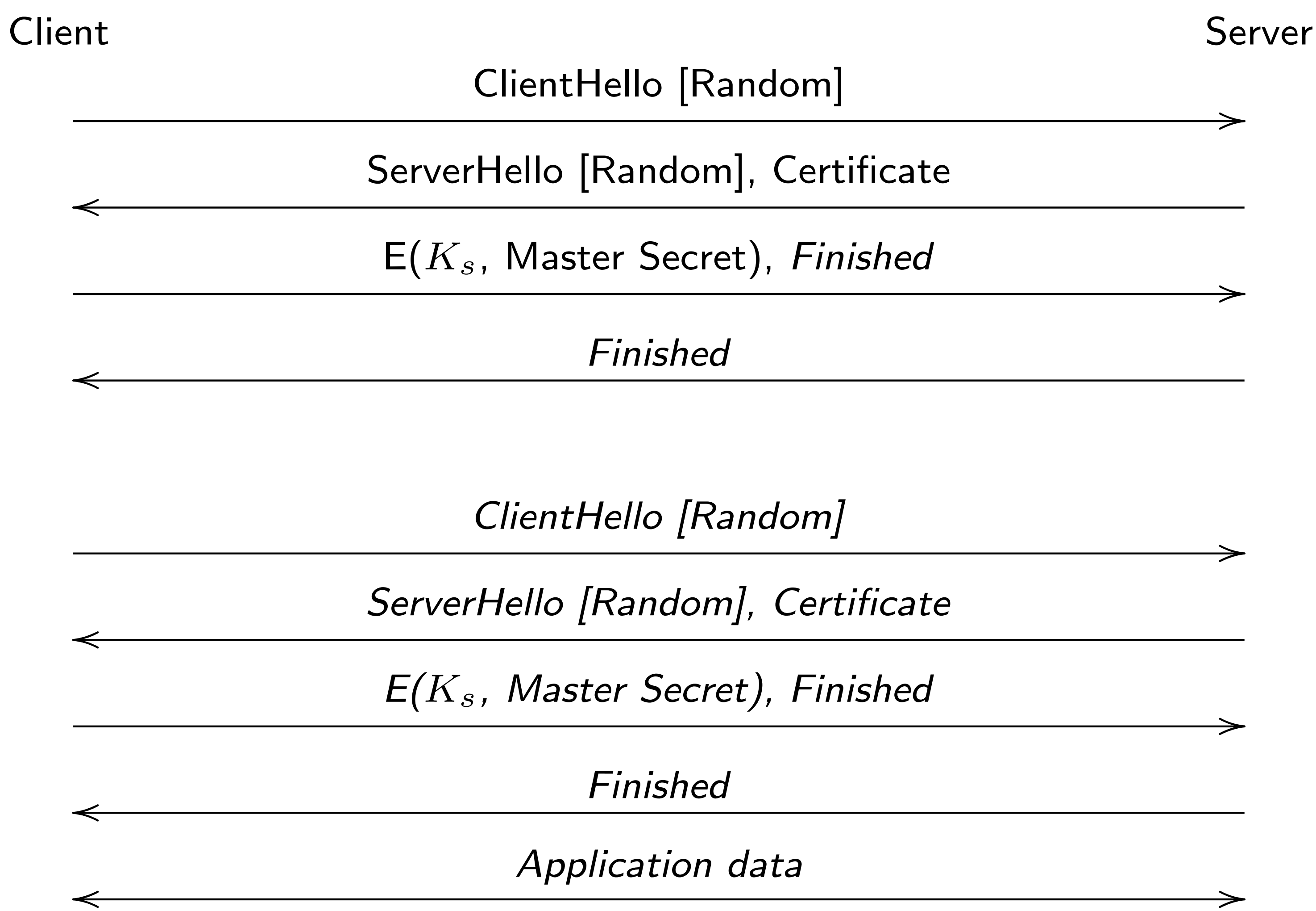
# Client Offers, Server Chooses

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16-1>;
    };
} ServerHello;
```

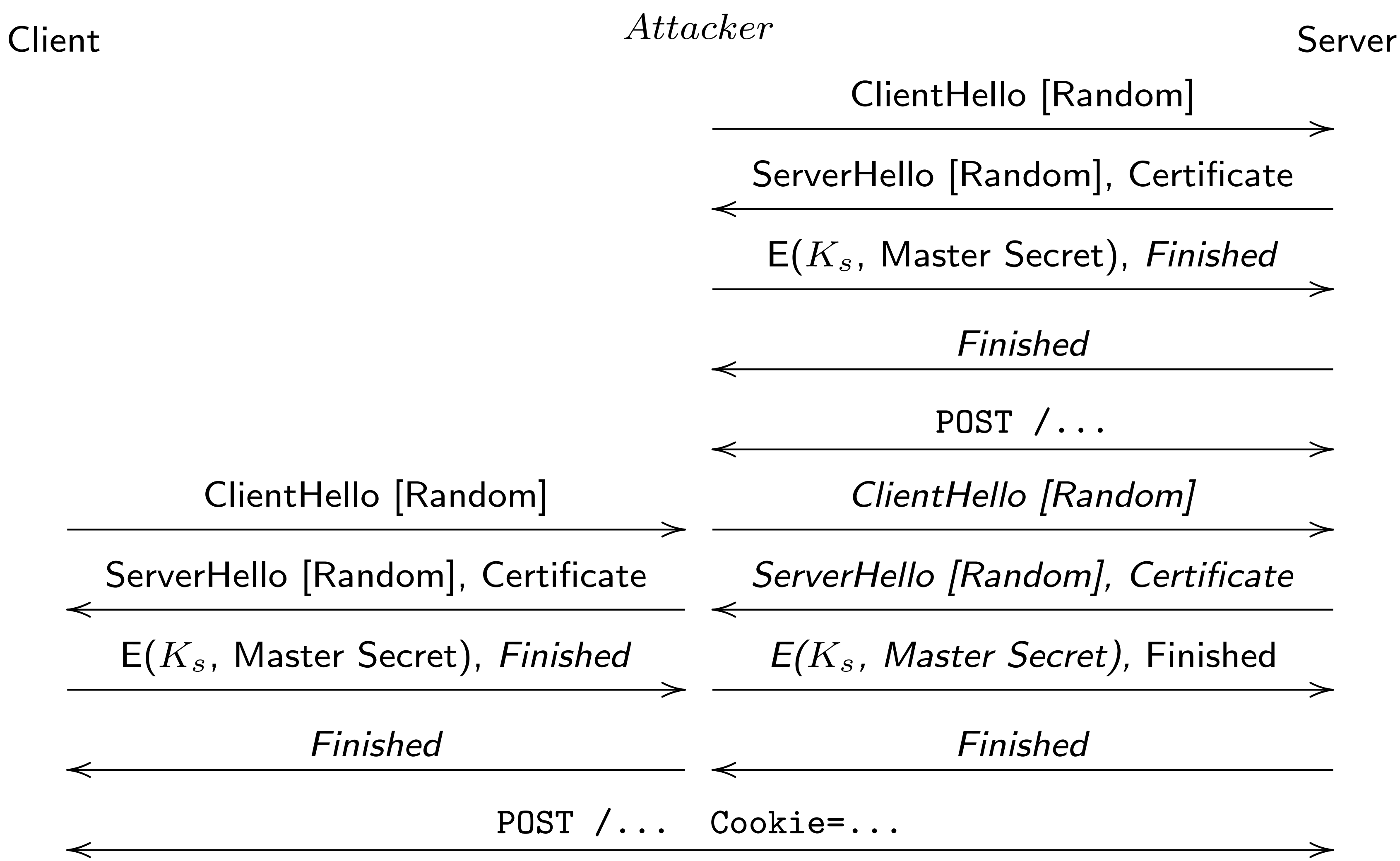
# What's in a Cipher Suite?

- Key Exchange (RSA, DHE, ECDHE, PSK, ...)
- Authentication (RSA, DSS, ECDSA, ...)
- Encryption (AES, Camellia, ...)
- MAC (MD5, SHA1, SHA256, ...)

# TLS 1.2: Renegotiation



# Renegotiation Attack [RRDO10]



# Why is this bad?

- Attacker gets to splice their data to the client's
- Example
  - Attacker-controlled request +
  - Client's credentials
- This looks like a renegotiation to server

# Renegotiation Info Extension [RFC5746]

- New extension in {Client,Server}Hello
  - Client's version contains its last Finished on this connection
  - Server's version contains last pair of Finished from this connection
- If you're not renegotiating with the same person you get a mismatch

# Uses for renegotiation (or, why can't we just get rid of it...)

- Conceal the client's certificate
- Post-handshake client authentication
- Refresh the traffic keying material

# TLS 1.2: Renegotiation for Client Authentication

ClientHello [Random] $\longrightarrow$

$\longleftarrow$ ServerHello [Random], Certificate, Sign($K_s$, $g^s$, ...)

$g^c$, *Finished* $\longrightarrow$

$\longleftarrow$ *Finished*

`GET /secure...` $\longrightarrow$

$\longleftarrow$ *HelloRequest*

*ClientHello [Random]* $\longrightarrow$

$\longleftarrow$ *ServerHello [Random], Certificate, CertificateRequest, Sign(*$K_s$, $g^s$, *...)*

$g^c$, *Certificate, Sign(*$K_c$, *...), Finished* $\longrightarrow$

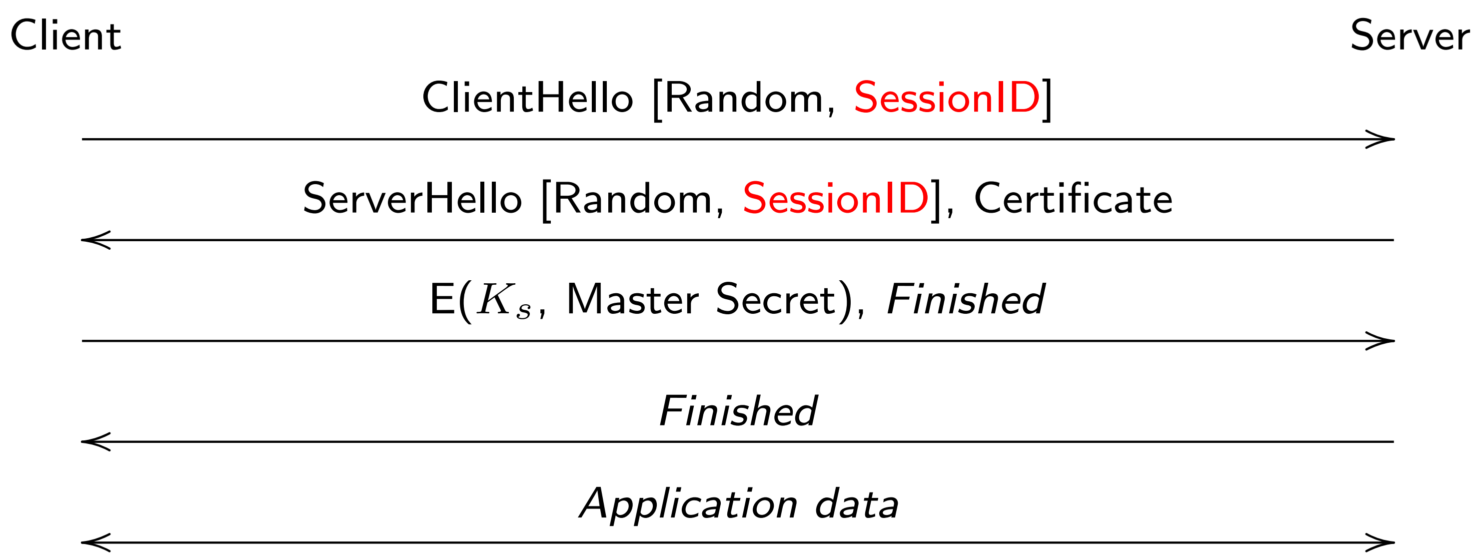
$\longleftarrow$ *Finished*

$\longleftarrow$ *Response*

# Session Resumption

- "Public key" operations are comparatively expensive
  - They used to be *really* expensive
- Solution: amortize this operation across multiple connections

# Session Establishment



# Session Resumption

Client Server

ClientHello [Random, SessionID]

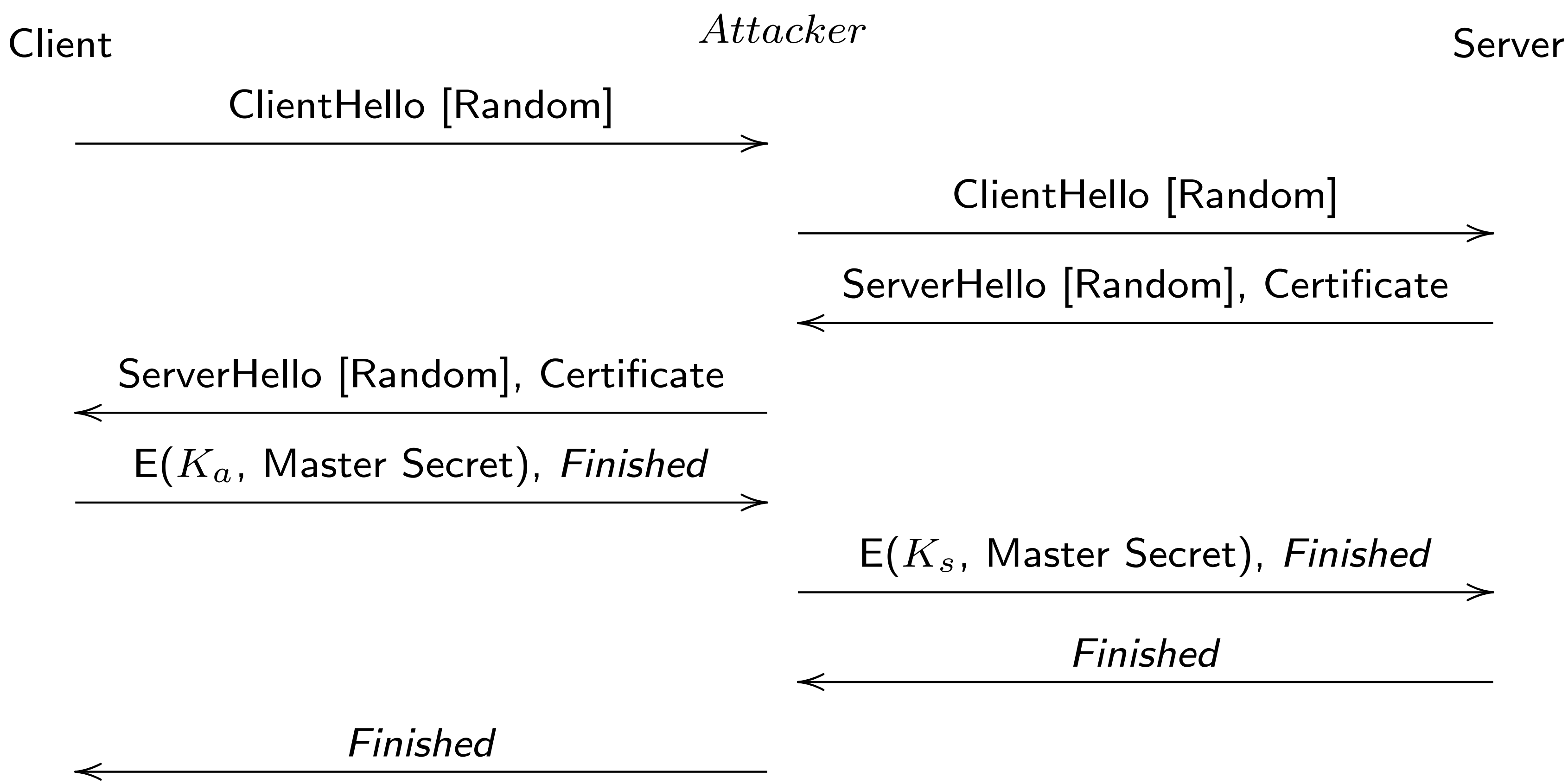
ServerHello [Random, SessionID], *Finished*

*Finished*

*Application data*

- No new public key operations
- Reuse MS from last handshake

# Triple Handshake (I)




- These connections have the same Master Secret
- “Unknown key share” attack

# Triple Handshake (II)

Client | Attacker | Server

ClientHello [Random, SessionID] →

← ServerHello [Random, SessionID], *Finished*

*Finished* →

`GET /secure-resource` →

← *HelloRequest*

*ClientHello [Random]* →

← *ServerHello [Random], CertificateRequest, Certificate*

*E*($K_s$, *Master Secret*), *Certificate*, *Sign*($K_c$, *...*), Finished →

← *MAC(MS, Handshake)*

← *Response*

# What's the impact?

- Resurrection of renegotiation attack
- Attacker controls the request
- Client authenticates it
- Thinks he's authenticating to the attacker
- ... but he's authenticating to the server

# Fixing the Triple Handshake (Session Hash)

- The problem is the unknown key share on the first handshake
- Fix is to hash the server certificate into the master secret
- Resumed handshakes inherit this context

# TLS 1.3 Objectives

- *Clean up:* Remove unused or unsafe features
- *Security:* Improve security by using modern security analysis techniques
- *Privacy:* Encrypt more of the protocol
- *Performance:* Our target is a 1-RTT handshake for naive clients; 0-RTT handshake for repeat connections
- *Continuity:* Maintain existing important use cases

# Removed Features

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

*Special accommodation for inline client authentication

# Removed Feature: Static RSA Key Exchange

- Most SSL servers prefer non-PFS cipher suites [SSL14] (specifically static RSA)
- Obviously suboptimal performance characteristics
- No PFS
- Gone in TLS 1.3
- Important: you can still use RSA certificates
  - But with ECDHE or DHE
  - Using ECDHE minimizes performance hit

# Removed Feature: Compression

- Recently published vulnerabilities [DR12]
- Nobody really knows how to use compression safely and generically
  - Sidenote: HTTP2 uses very limited context-specific compression [PR14]
- TLS 1.3 bans compression entirely
  - TLS 1.3 clients MUST NOT offer any compression
  - TLS 1.3 servers MUST fail if compression is offered

# Removed Feature: Non-AEAD Ciphers

- Symmetric ciphers have been under a lot of stress (thanks, Kenny and friends)
  - RC4 [ABP+13]
  - AES-CBC [AP13] in MAC-then-Encrypt mode
- TLS 1.3 bans all non-AEAD ciphers
  - Current AEAD ciphers for TLS: AES-GCM, AES-CCM, ARIA-GCM, Camellia-GCM, ChaCha/Poly (coming soon)

# Removed Feature: Custom (EC)DHE groups

- Previous versions of TLS allowed the server to specify their own DHE group
  - The only way things worked for finite field DHE
  - (Almost unused) option for ECDHE
- This isn't optimal
  - Servers didn't know what size FF group client would accept
  - Hard for client to validate group [BLF$^+$14]
- TLS 1.3 only uses predefined groups
  - Existing RFC 4492 [BWBG$^+$06] EC groups (+ whatever CFRG comes up with)*
  - New FF groups defined in [Gil14]

*Bonus: removed point format negotiation too

# 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 a DH share

# TLS 1.3 1-RTT Handshake Skeleton

| Client | | Server |
|---|---|---|
| | ClientHello [Random, $g^c$] $\longrightarrow$ | |
| | $\longleftarrow$ ServerHello [Random, $g^s$] | |
| | $\longleftarrow$ *Certificate, Sign($K_s$, Handshake), Finished* *Application data* | |
| | *Finished* $\longrightarrow$ | |
| | $\longleftarrow$ *Application data* $\longrightarrow$ | |

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

# TLS 1.3 1-RTT Handshake w/ Client Authentication Skeleton

Client | | Server
--- | --- | ---
| | ClientHello [Random, $g^c$] $\longrightarrow$ | |
| | $\longleftarrow$ ServerHello [Random, $g^s$] | |
| | $\longleftarrow$ *CertificateRequest, Certificate, Sign($K_s$, Handshake), Finished*<br>*Application data* | |
| | *Certificate, Sign($K_c$, Handshake), Finished* $\longrightarrow$ | |
| | $\longleftrightarrow$ *Application data* | |

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

# What happens if the client is wrong?

- Client sends some set of groups (P-256)
- Server wants another group (P-384)

Client — Server

ClientHello [Random, DH(P256)] $\longrightarrow$

$\longleftarrow$ HelloRetryRequest [P384]

ClientHello [Random, DH(P256), DH(P384)] $\longrightarrow$

...

- This shouldn't happen often because there are a small number of groups
  - Client should memorize server's preferences

# 0-RTT Handshake

- Basic observation: client can cache server's parameters [Lan10]
  - Then send *application data* on its first flight
- Server has to *prime* the client with its configuration in a previous handshake

# TLS 1.3 0-RTT Handshake Skeleton

Client Server

ClientHello [Random, $g^c$, server_configuration=XXX] →

*Application data*

ServerHello [Random, $g^s$] ←

*Certificate, Sign($K_s$, Handshake), Finished*

*Application data* ←

*Finished* →

*Application data* ↔

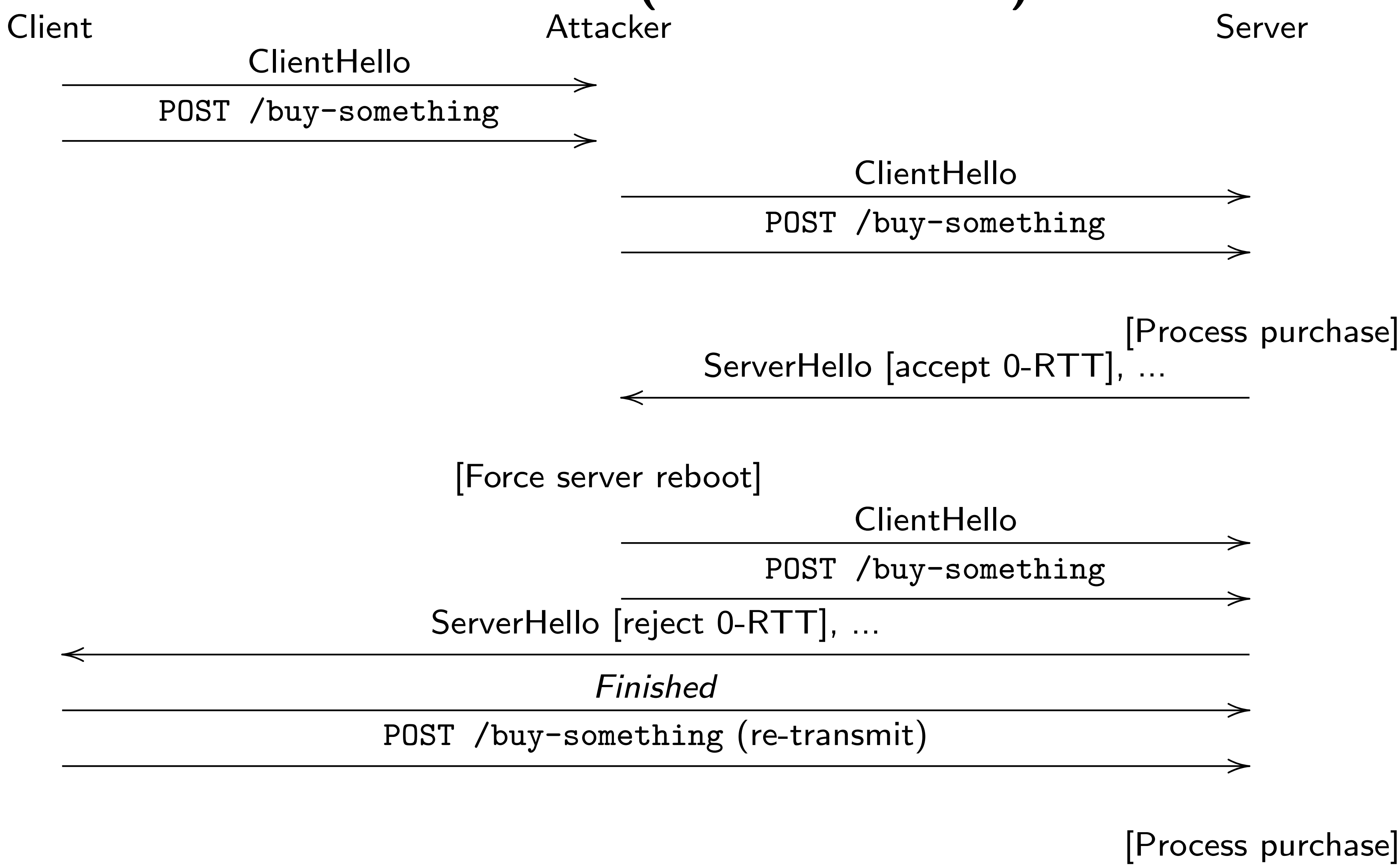
# Anti-Replay

- TLS anti-replay is based on each side providing random value
  - Mixed into the keying material
- Not compatible with 0-RTT
  - Client has anti-replay (since they speak first)
  - Server's random isn't incorporated into client's first flight

# Anti-Replay (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

# This doesn't work (thanks to DKG)



# Oops...

- The real problem is multiple data centers
- This is a distributed state problem
  - It's broken in QUIC and Snap Start too
- Resolution: dont even try
  - Only use 0-RTT client data for idempotent requests (GETs)
  - Difficult application integration issue
  - But too big a win not to do
- This can't be on by default
  - And it will need a special API

# Pre-Shared Keys and Resumption

- TLS 1.2 already supported a Pre-Shared Key (PSK) mode
  - Used for IoT-type applications
- Two major modes
  - Pure PSK
  - PSK + (EC)DHE
- 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)

```
          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
```

# Single Key Derivation and Authentication Logic

- Based on ideas from OPTLS (Krawczyk and Wee)

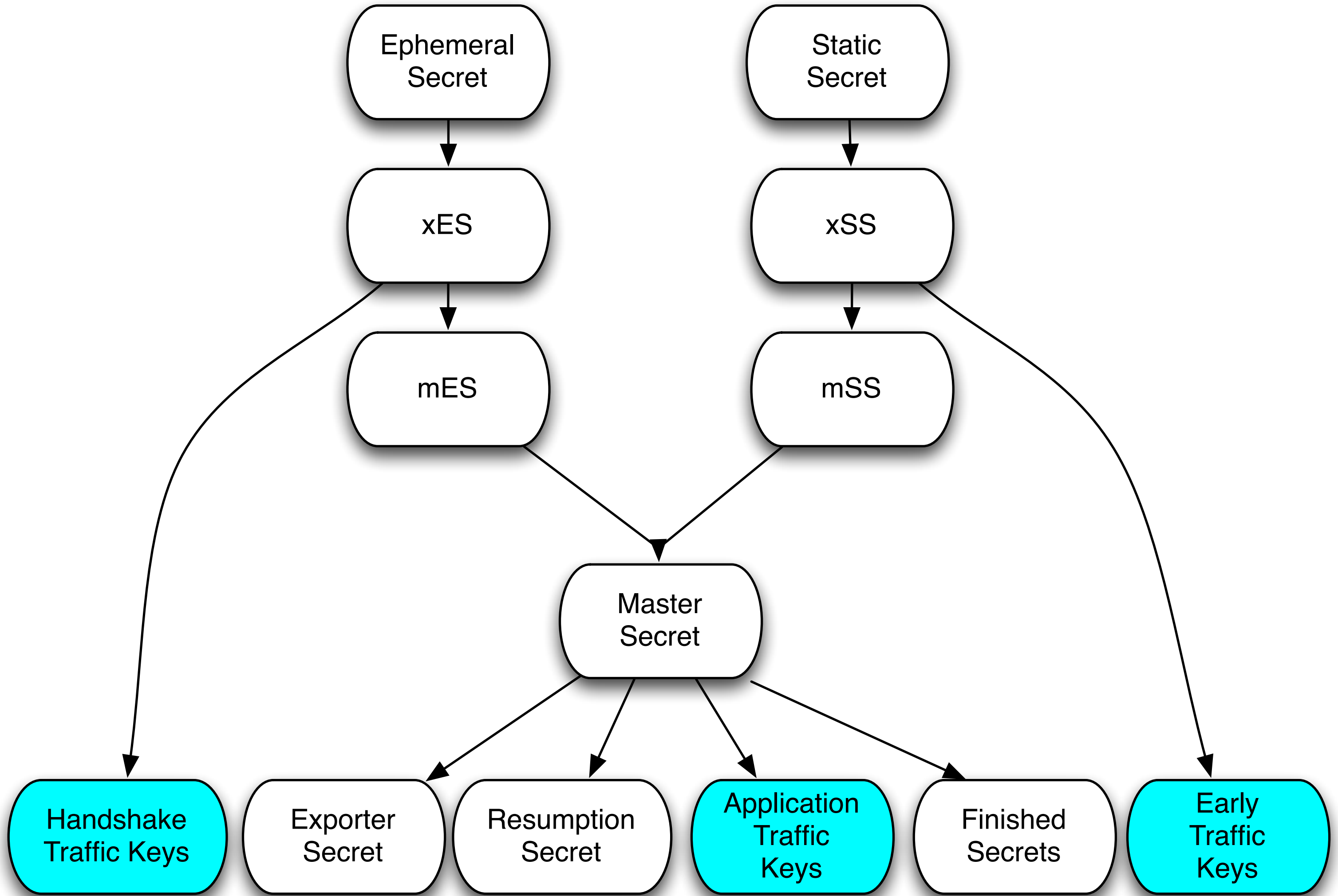
```
Key Exchange             Static Secret (SS)    Ephemeral Secret (ES)
------------             ------------------    ---------------------
(EC)DHE                    Client ephemeral         Client ephemeral
(full handshake)        w/ server ephemeral      w/ server ephemeral

(EC)DHE                    Client ephemeral         Client ephemeral
(w/ 0-RTT)                 w/ server static      w/ server ephemeral

PSK                          Pre-Shared Key           Pre-shared key

PSK + (EC)DHE                Pre-Shared Key         Client ephemeral
                                                 w/ server ephemeral
```

# Key Derivation



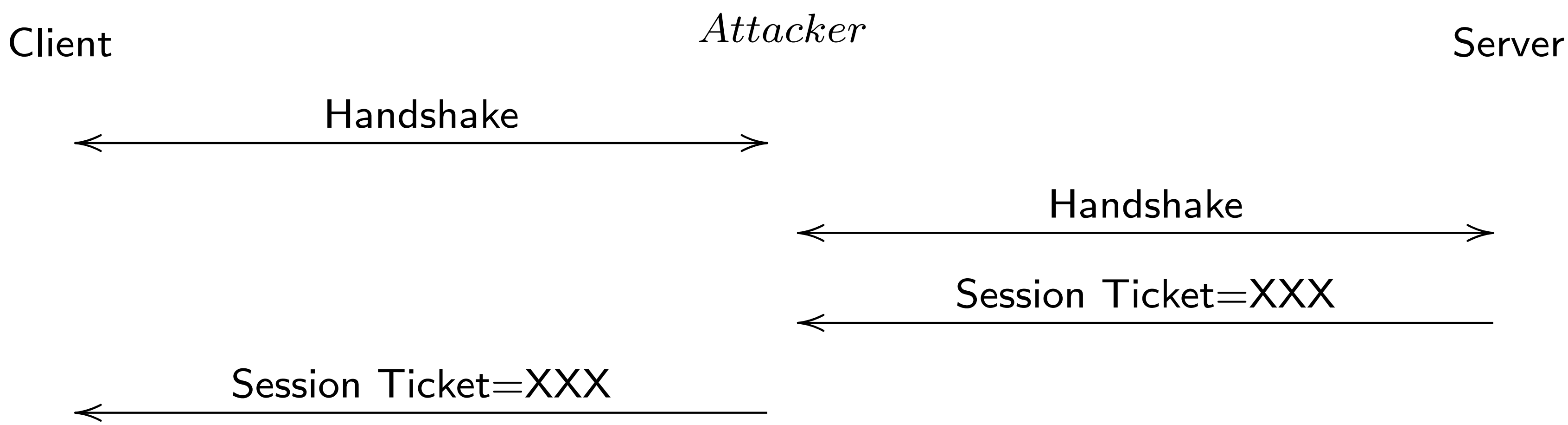
# Post-Handshake Client Auth

- We removed renegotiation
  - But that doesn't remove the *need* for post-handshake authentication
- Current plan: server can send CertificateRequest at any time
  - Client responds with "authentication block"
    * Certificate
    * Signature over the handshake through server's MAC
    * MAC over handshake + Certificate + Signature
- This piece is still under development
  - `https://github.com/tlswg/tls13-spec/pull/316`

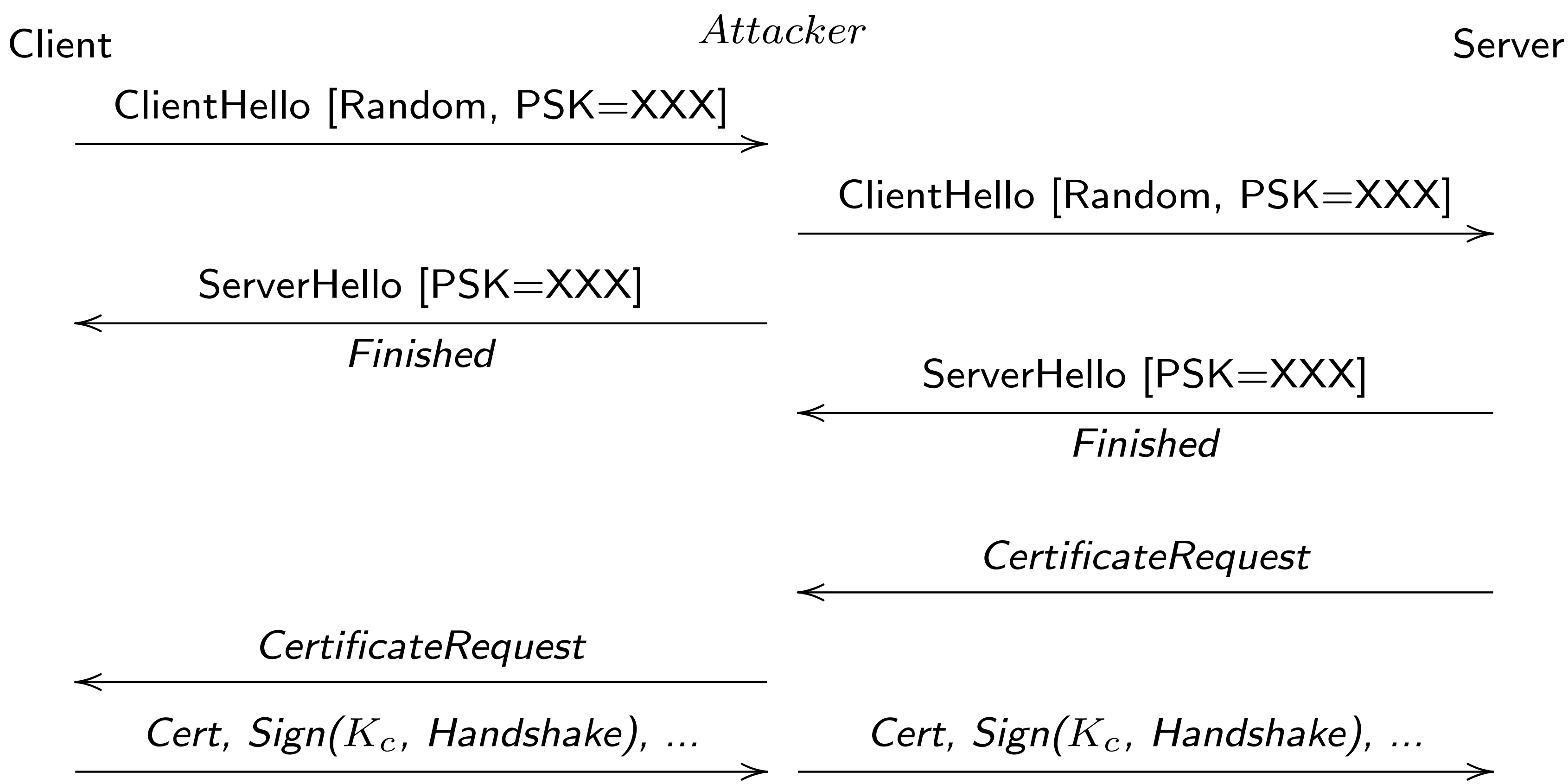
# Interactions

- What happens when you combine PSK and post-handshake client auth?
- This is something you want to work
  - Idea is to add client authentication to “resumed” sessions
  - In TLS 1.2, this is done with renegotiation

# Attack on Naive Design: Setup [CHvdMS]



# Attack on Naive Design: Reconnect



# Analysis

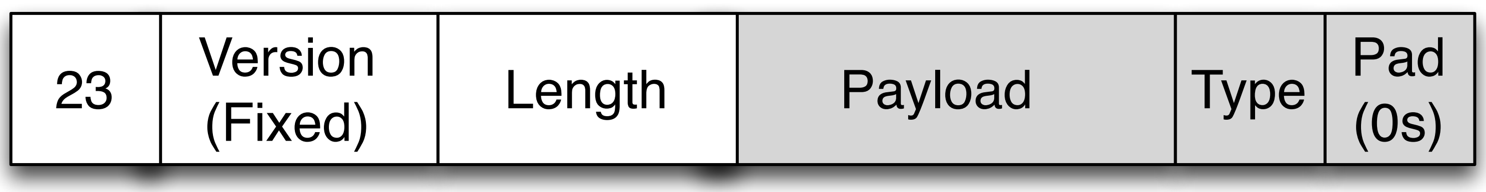
- The question is exactly what you sign
- In draft-10, client signed the server cert but not the server MAC
  - Didn't include client auth with PSK
  - ... or post-handshake
- PR#316 includes server's cert and MAC
  - Which transitively includes the server's certificate
  - This reinforces this decision
- This result comes directly from formal analysis with Tamarin
  - This is good news!
  - Big thanks to Cas Cremers, Marko Horvat, Thyla van der Merwe, Sam Scott

# 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

# Packet Format


TLS 1.2 Packet Layout




TLS 1.3 Packet Layout

# Server Name Indication

- How do you have multiple domains on the same server?
- Problem: Each domain may have its own certificate
  - How does the server know which one to present?
- Wrong way: each server gets their own IP address
  - Obvously this does not scale
  - But it's what people actually do (thanks Windows XP and Android 2.2)
- Right: ClientHello extension indicating server domain name
  - "Server Name Indication" (SNI)
- SNI is required for TLS 1.3

# Open Issue: Encrypted SNI

- SNI leaks the server's identity
  - Even if the server certificate is encrypted!
- would be nice to hide the SNI
  - So `hidden.com` and `innocuous.com` could share a server
  - Important for anti-censorship applications
- WG is still struggling with this
  - General idea is to use the 0-RTT first flight to hide SNI
  - But the details are complicated
  - Looks like we can do this without major changes (and perhaps none)

# Current Status

- Currently in draft-10
- Most major issues resolved at IETF Yokohama (two weeks ago)
- Formal models already starting to emerge
- Implementation in NSS (Firefox) by EOY
  - OpenSSL, etc. to follow
- TLS Ready or Not Workshop in February (co-located with ISOC NDSS)
- Expect Last Call in Q1

# Following the Work

- IETF TLS Mailing List:
  `https://www.ietf.org/mailman/listinfo/tls`
- Github repository: `https://github.com/tlswg/tls13-spec`
- Editor's draft: `http://tlswg.github.io/tls13-spec/`

# Questions?

# Extra Material

# Backward Compatibility Problems

1. What do you do if the other side doesn't support RI?
   - Server can refuse to renegotiate
   - Client can only refuse to connect
     - Guess what clients do...
2. Some servers are *extension intolerant*
   - Extensions were defined after SSLv3 was already published
   - Some servers choke on extensions
   - ... badly

# Special Signaling Cipher Suites (I)

- OK, so the client can't always send an extension
  - What can it safely send?

# Special Signaling Cipher Suites (II)

- OK, so the client can't always send an extension
  - What can it safely send?
  - ... *a cipher suite*

```
"IANA has added TLS cipher suite number 0x00,0xFF with name
 TLS_EMPTY_RENEGOTIATION_INFO_SCSV to the TLS Cipher Suite registry."
                                                          - RFC5746
```

- Cipher suite negotiation code gets exercised regularly
- And got a workout when we added AES
  - So it's mostly safe to send new cipher suites

# TLS 1.2: (EC)DHE Skeleton

Client | Server
---|---
ClientHello [Random] $\longrightarrow$ |
$\longleftarrow$ ServerHello [Random], Certificate, Sign($K_s$, $g^s$, ...) |
$g^c$, *Finished* $\longrightarrow$ |
$\longleftarrow$ *Finished* |
$\longleftrightarrow$ *Application data* |

# TLS 1.2: (EC)DHE + Client Authentication

Client | Server
---|---

ClientHello [Random]

$\longrightarrow$

ServerHello [Random], Certificate, CertificateRequest, Sign($K_s$, $g^s$, ...)

$\longleftarrow$

$g^c$, Certificate, Sign($K_c$, ...), *Finished*

$\longrightarrow$

*Finished*

$\longleftarrow$

*Application data*

$\longleftrightarrow$

# 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.

[BWBG+06] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492 (Informational), May 2006. Updated by RFCs 5246, 7027.

[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`.

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

[Gil14] Daniel Kahn Gillmor. Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for TLS. Internet-Draft draft-ietf-tls-negotiated-ff-dhe, Internet Engineering Task Force, August 2014. Work in progress.

[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.

[PR14] Roberto Peon and Herve Ruellan. HPACK - Header Compression for HTTP/2. Internet-Draft draft-ietf-httpbis-header-compression-09, Internet Engineering Task Force, July 2014. Work in progress.

[RRDO10] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Proposed Standard), February 2010.

[SSL14] SSL Pulse. `https://www.ssllabs.com/`, Dec 2014.