TLS 1.3: A Collision of Implementation, Standards, and Cryptography

Eric Rescorla

Mozilla

ekr@rtfm.com

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

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

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 most servers (we expect some fallback logic...)
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

Removed Features

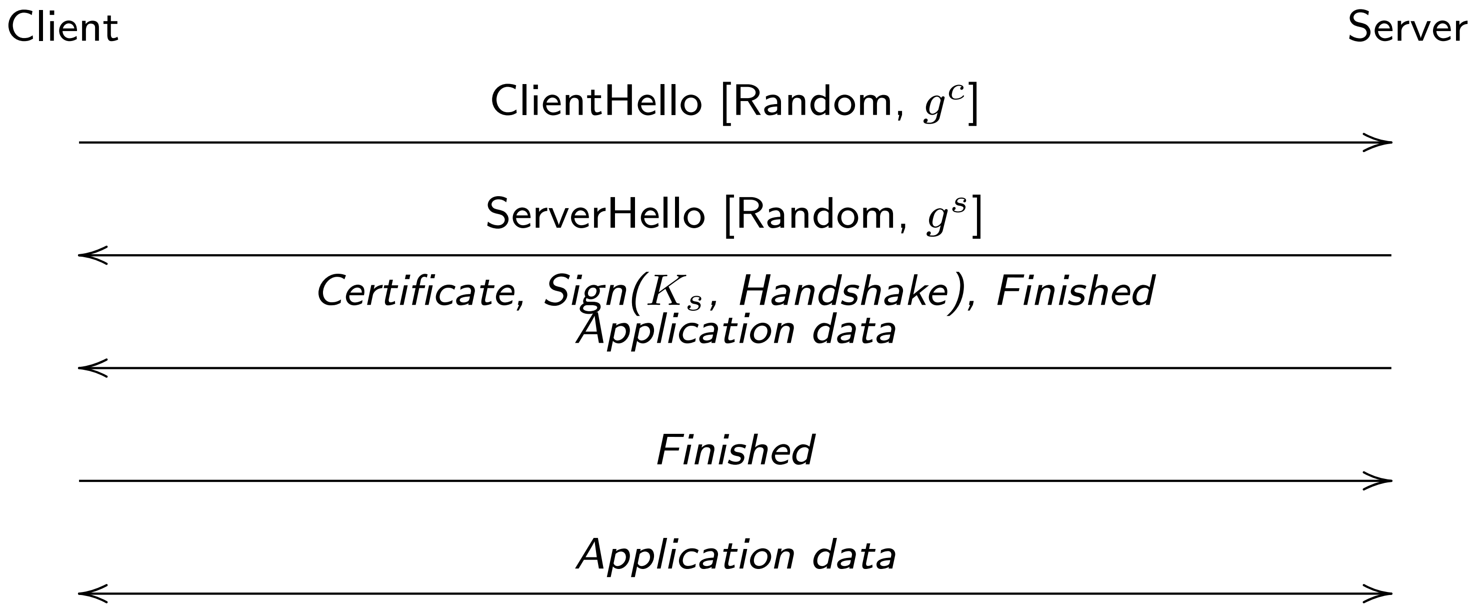
- Static RSA
- Custom (EC)DHE groups
- Compression
- Renegotiation*
- Non-AEAD ciphers
- Simplified resumption

*Special accommodation for post-handshake 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 a DH share

TLS 1.3 1-RTT Handshake Skeleton



Client

Server

ClientHello [Random, g^c]

ServerHello [Random, g^s]

Certificate, Sign(K_s, Handshake), Finished

Application data

Finished

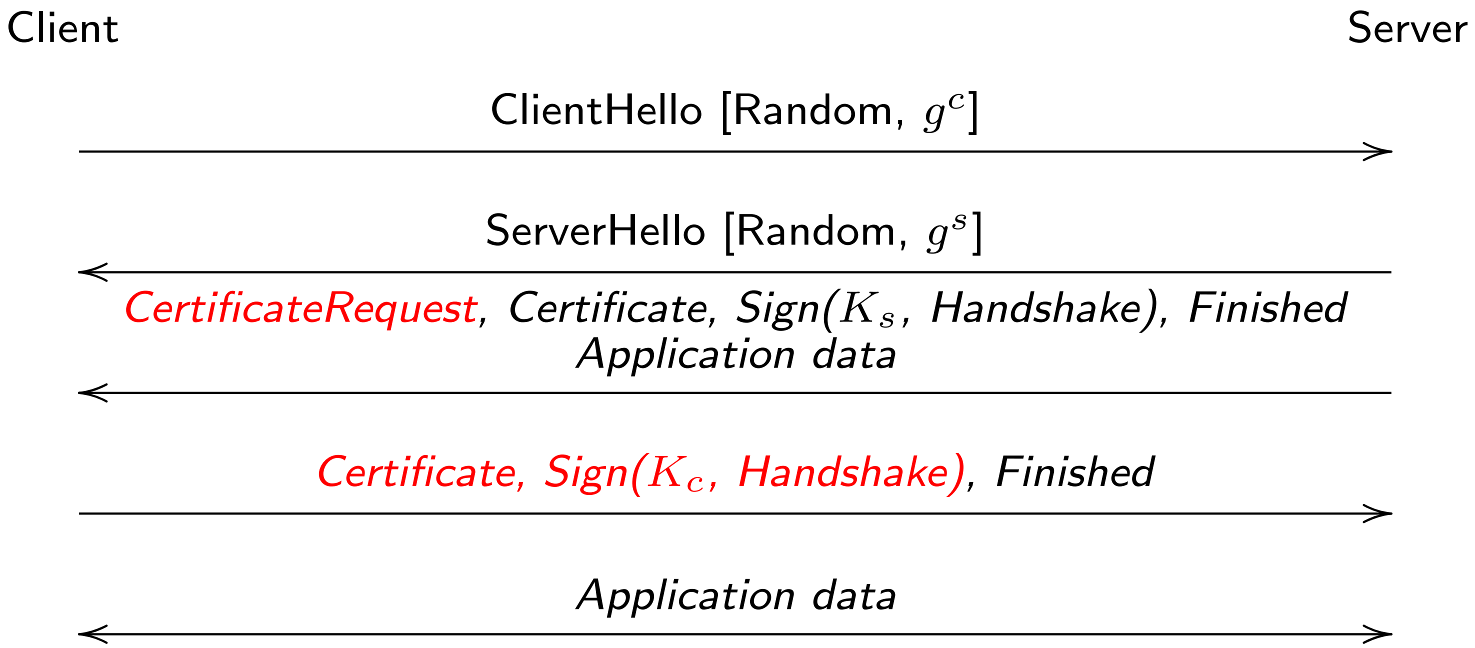
Application data

- 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 confidential 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)?
  - 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

Server

ClientHello [Random, g^c]

ServerHello [Random, g^s]

CertificateRequest, Certificate, Sign(K_s, Handshake), Finished
Application data

Certificate, Sign(K_c, Handshake), Finished

Application data

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

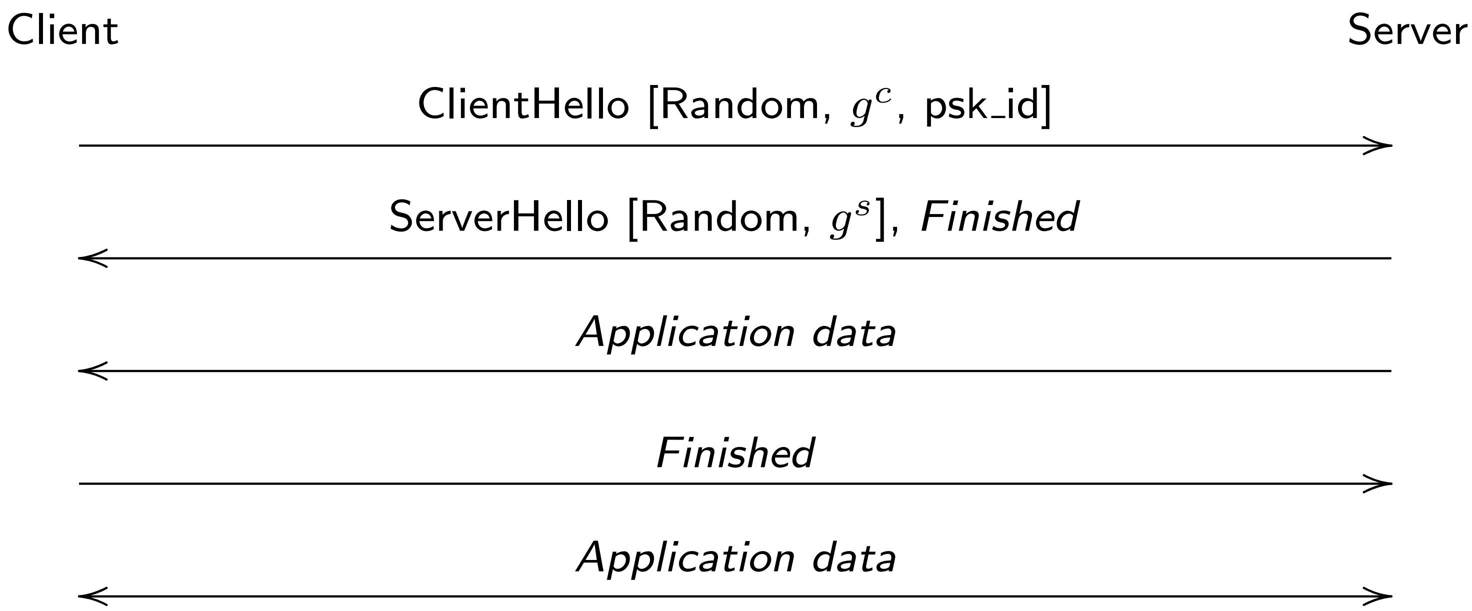
Resumption

- TLS has always supported a “resumption” mode
  - Amortize first public key exchange across multiple connections
- Historically a huge performance win
- Maybe less so now
  - Widespread use of ECC
  - Lower connection counts with HTTP/2
- But we don’t want to take a performance regression (see constraint #1)

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)
  - Improvement: this key is independent of ordinary traffic keys

Pre-Shared Key Handshake Skeleton



Client

Server

ClientHello [Random, g^c, psk_id]

ServerHello [Random, g^s], *Finished*

Application data

Finished

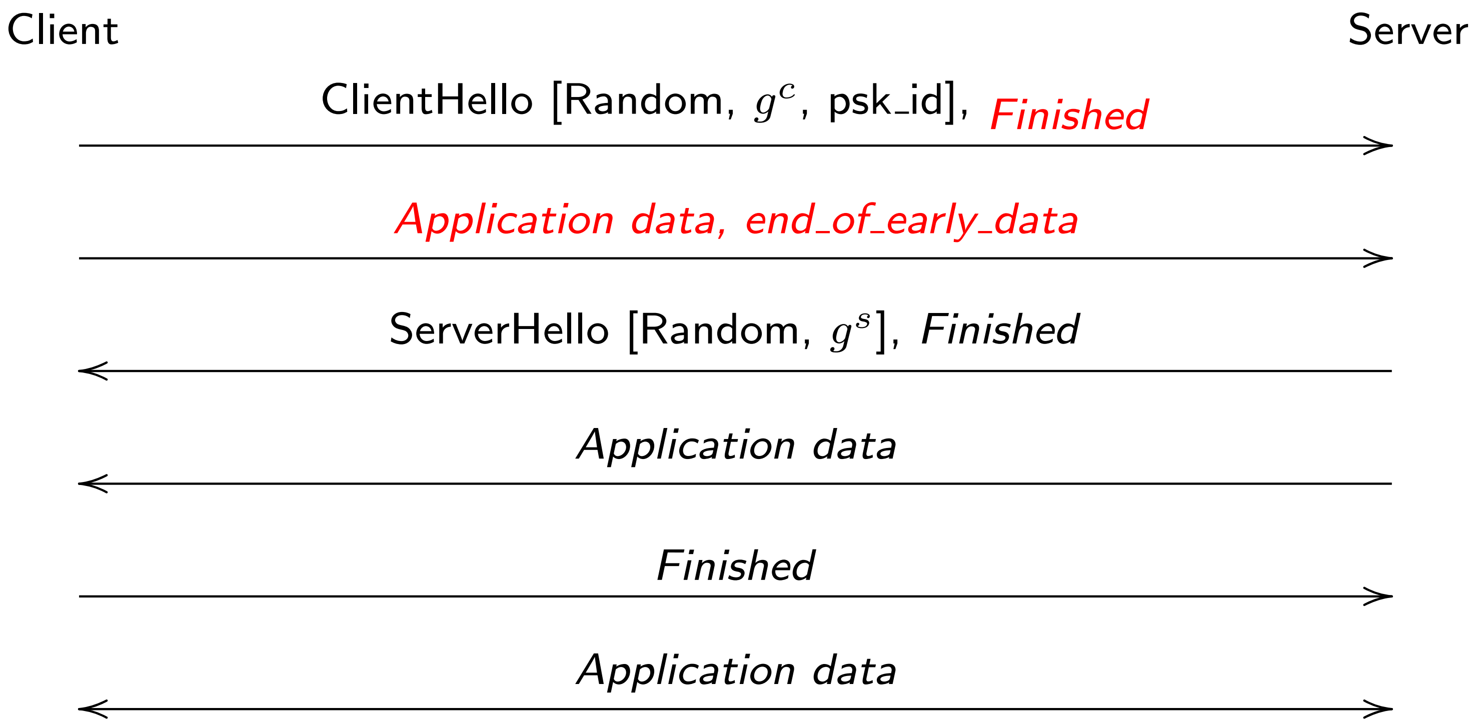
Application data

- Can use *either* PSK or (EC)DHE-PSK

0-RTT Data

- Important performance improvement for TLS 1.3 over TLS 1.2
- Initially we had two 0-RTT modes
  - Semi-static (EC)DHE
  - Pre-shared key resumption
- Strong consensus to keep only the PSK mode for now
  - Better fit with existing resumption model
  - Simpler to implement and specify
- Simultaneous suggestions by implementors (Cloudflare) and researchers (Fournet et al.)

TLS 1.3 0-RTT



Client

Server

ClientHello [Random, g^c, psk_id], *Finished*

Application data, end_of_early_data

ServerHello [Random, g^s], *Finished*

Application data

Finished

Application data

Involvement with Research Community

- Typically standards get developed and then analyzed
  - Hard to fix defects in the field
  - Takes a long time
  - *We're still finding problems in TLS 1.2*
- Trying to do something different with TLS 1.3
- Huge amount of interest from academia in TLS 1.3 development
  - 9 papers at TRON workshop
  - 3 papers at Oakland
  - At least 3 security proofs of working drafts [CHSvdM16, DFGS16, KMO$^+$16] and more on the way
- Very interactive process

Areas of collaboration (a nonexhaustive list)

- Design contributions (especially OPTLS[KW15], INRIA/Microsoft)
- Participation in the standards process
- Implementation/interop testing
- Ongoing analysis

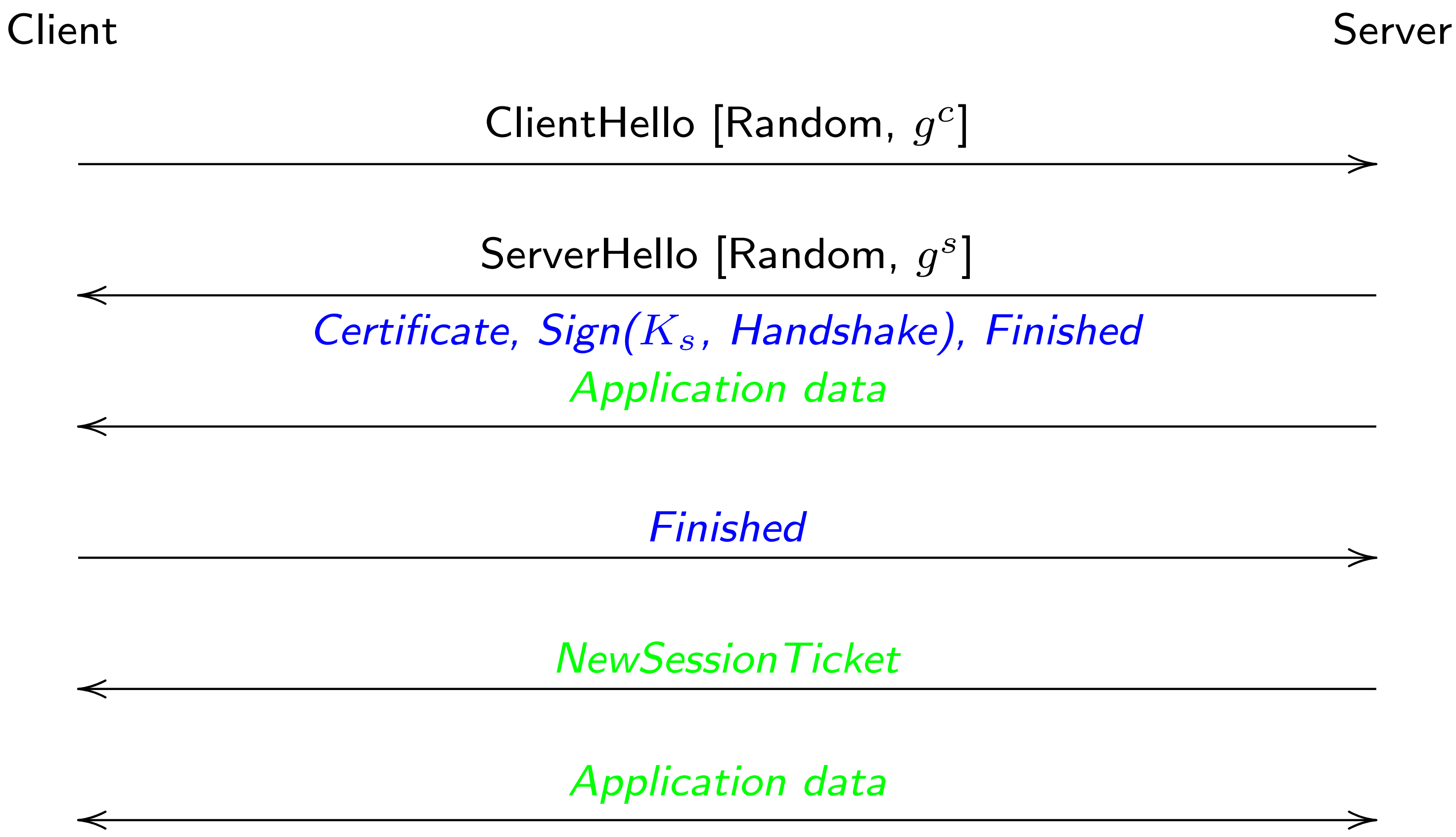
Case study 1: Key Separation

- Prior versions of TLS 1.2 use the same keys for encrypting the traffic and parts of the handshake. This made cryptographers sad:

  “Although they do not suffer from clear attacks, various key agreement protocols (for example that used within the TLS protocol) are deemed as insecure by existing security models for key exchange. The reason is that the derived keys are used within the key exchange step, violating the usual key-indistinguishability requirement.”.[BFS$^+$13]

- TLS 1.3 mostly fixes this
  - Separate keys for handshake and application data
  - Except for post-handshake traffic

TLS 1.3 Key Usage



Client

Server

ClientHello [Random, g^c]

ServerHello [Random, g^s]

Certificate, Sign(K_s, Handshake), Finished

Application data

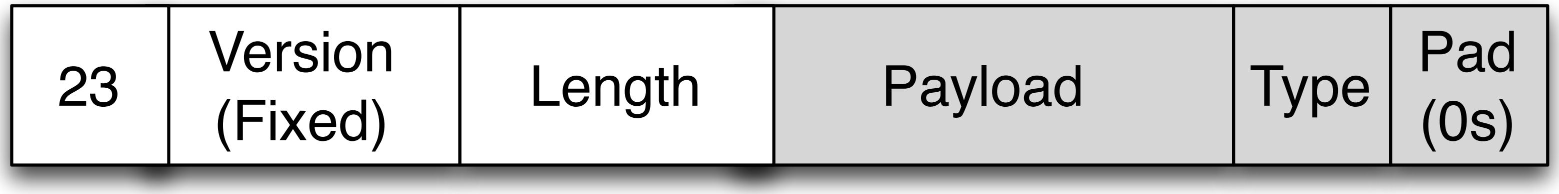
Finished

NewSessionTicket

Application data

- Same key used for application data and post-handshake traffic
  - This creates a problem for composability [DFGS16]

Just use two keys. What's the problem?



23	Version (Fixed)	Length	Payload	Type	Pad (0s)

- Two keys in use concurrently
  - Handshake (or post-handshake)
  - Application
- How do I know which key is being used?
  - Trial decryption
  - Wrap handshake-encrypted messages in application keys
  - ~~Restore the content type byte~~

Why not trial decryption?

- Seems like a good solution
  - But there are implementation problems
- Some TLS implementations decrypt in-place (or even more exotic things)
  - But AEAD decryption failure leaves you in an undefined state
  - And making a copy in advance is expensive
- Wrapping starts to look better...

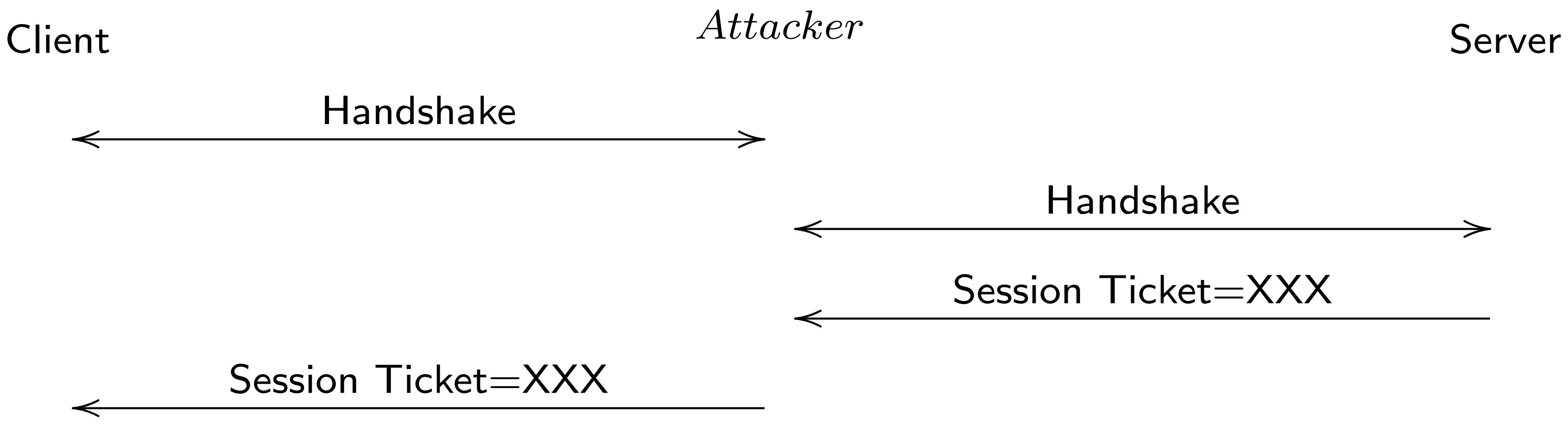
What about wrapping?

- You're still using *application* keys to encrypt *handshake* messages
  - So the application keys aren't indistinguishable with respect to those messages
- Of course you're already using the *application* keys to encrypt application traffic
  - So probably not that big a deal
- Fast work by Krawczyk, Doug Stebila, Björn Tackmann and others gave us an analysis for this case.
- IETF will probably go with wrapping

Case study 2: PSKs and Client Authentication

- 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]



Client

Attacker

Server

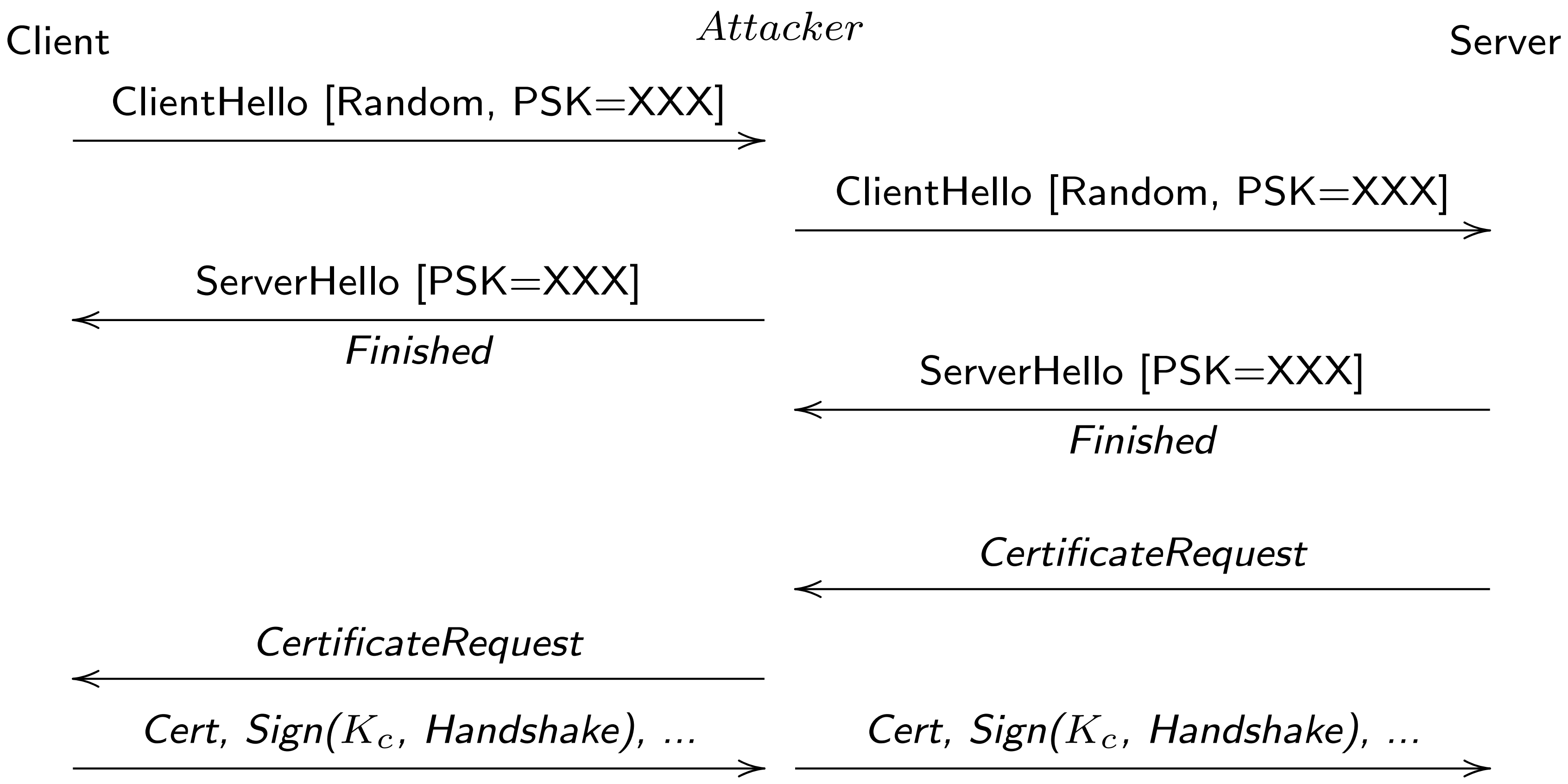
Handshake

Handshake

Session Ticket=XXX

Session Ticket=XXX

Attack on Naive Design: Reconnect



Client

$Attacker$

Server

ClientHello [Random, PSK=XXX]

ClientHello [Random, PSK=XXX]

ServerHello [PSK=XXX]

Finished

ServerHello [PSK=XXX]

Finished

CertificateRequest

CertificateRequest

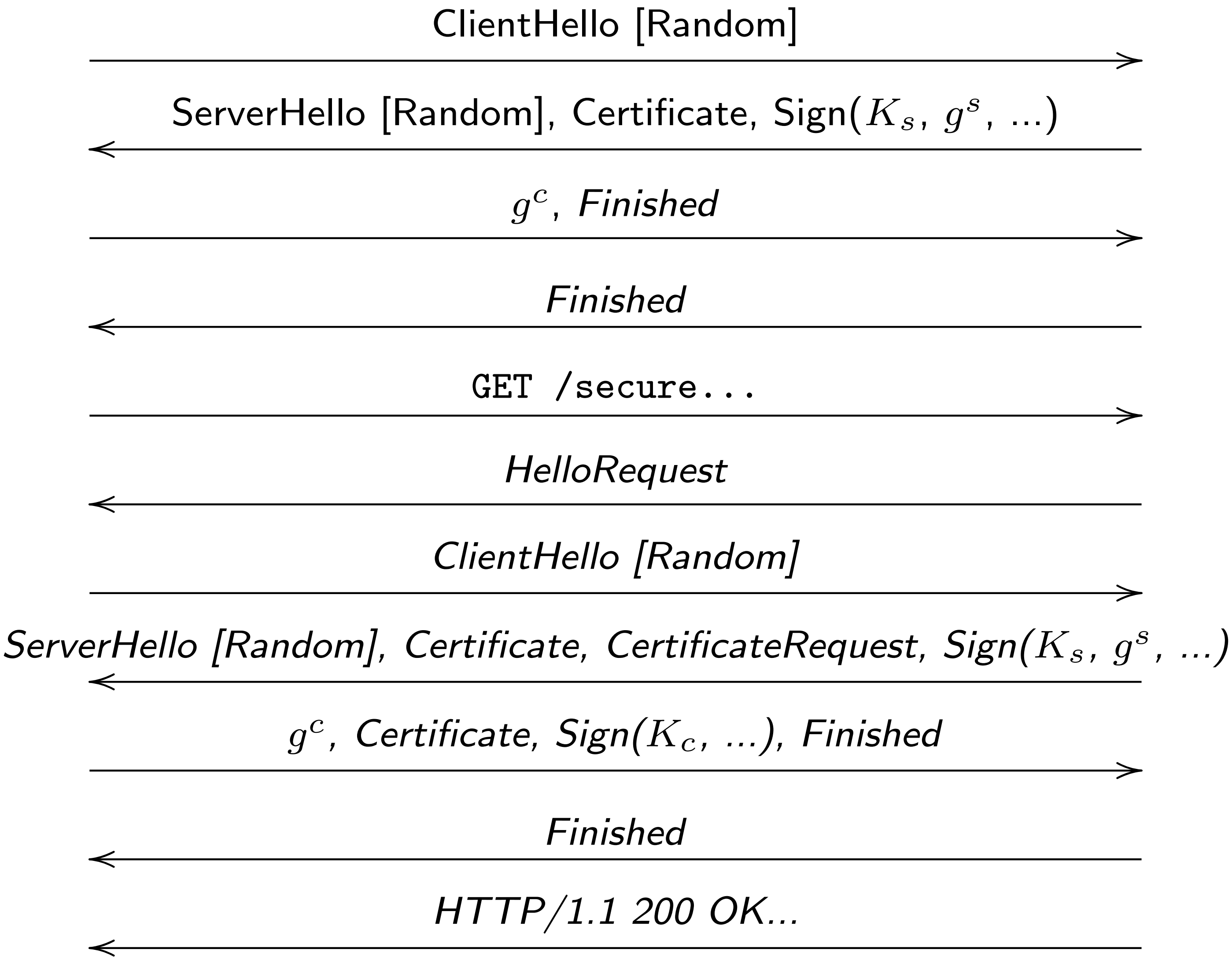
Cert, Sign(K_c, Handshake), ...

Cert, Sign(K_c, Handshake), ...

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
- TLS 1.3 draft-12 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

Case Study 3: TLS 1.2 Renegotiation for Client Auth



ClientHello [Random] →

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

g^c, *Finished* →

← *Finished*

`GET /secure...` →

← *HelloRequest*

ClientHello [Random] →

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

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

← *Finished*

← *HTTP/1.1 200 OK...*

Post-Handshake Client Auth

- We removed renegotiation
  - But that doesn't remove the *need* for post-handshake authentication
- Resolution: server can send CertificateRequest at any time
  - Client responds with "authentication block" (idea due to Bhargavan)
    - Certificate
    - Signature over the handshake through server's MAC
    - MAC over handshake + Certificate + Signature

```
           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
```

Status

- Consensus on approach for nearly all issues at IETF 95 (Buenos Aires)
  - draft-13 in preparation now (target: next week); should be ready for analysis
  - Target: last call before IETF 96 (Berlin) in July
- Multiple partially interoperating implementations
  - NSS, Mint, ProtoTLS, nqsb, miTLS
- “TRON 2” meetup and interop event at Oakland
- Follow along: `https://github.com/tlswg/tls13-spec`

Implementation Status

Name	Language	ECDHE	DHE	PSK	0-RTT
NSS	C	Yes	No	Yes	Yes*
Mint	Go	Yes	Yes	Yes	Yes
nqsb	OCaml	No	Yes	Yes	No
ProtoTLS	JavaScript	Yes	Yes	Yes	Yes
miTLS	F*	Yes	Yes	Yes	???

- NSS interops with Mint and ProtoTLS
  - NSS 0-RTT in unintegrated branch
- ProtoTLS interops with nqsb
- Other combinations untested

Questions?

References

[BFS$^+$13] C. Brzuska, M. Fischlin, N. P. Smart, B. Warinschi, and S. C. Williams. Less is more: Relaxed yet composable security notions for key exchange. *Int. J. Inf. Secur.*, 12(4):267–297, August 2013.

[CHSvdM16] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016. (to appear).

[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`.

[DFGS16] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A Cryptographic Analysis of the TLS 1.3 draft-10 Full and Pre-shared Key Handshake Protocol, 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.

[KMO+16] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. (De-)Constructing TLS 1.3, 2016.

[KW15] Hugo Krawczyk and Hoeteck Wee. The optls protocol and tls 1.3. *IACR Cryptology ePrint Archive*, 2015:978, 2015.