

TLS 1.3

Eric Rescorla

`ekr@rtfm.com`

Reminder: Objectives

- Encrypt as much of the handshake as possible
- Reduce handshake latency, with a target of 0-RTT for repeated handshakes and 1-RTT for “full” handshakes
- Reevaluate handshake contents
- Reevaluate record protection mechanisms (not discussed here)

New Handshake Flows

- Almost nothing here is new
- Ideas cribbed from
 - False Start
 - Snap Start
 - NPN
 - Marsh Ray's encrypted handshake draft
 - A bunch of other people
- Writeup in: `draft-rescorla-tls13-new-flows-01`

Basic Assumptions

- No more static RSA
 - RSA certificates still allowed (duh)
 - But only for DHE
- Encrypt extensions
 - But generally this protects only against passive attack
 - The issue here is encrypting SNI (more later)
- Only handle the “full” handshake
 - Open issue if we want to do resumption

Basic Mode: 1RTT Handshake (assumes knowledge of server key)

```

ClientHello
+ PredictedParameters
ClientKeyExchange
[ChangeCipherSpec]
{EncryptedExtensions} ----->

                                ServerHello
                                ServerKeyExchange
                                [ChangeCipherSpec]
                                {EncryptedExtensions}
                                {Certificate*}
                                {CertificateRequest*}
                                {ServerParameters*}
                                {CertificateVerify*}
                                <-----
                                {Finished}

{[ChangeCipherSpec]}
{Certificate*}
{CertificateVerify*}
{Finished} ----->
<----- {Finished ???} // Probably not needed

```

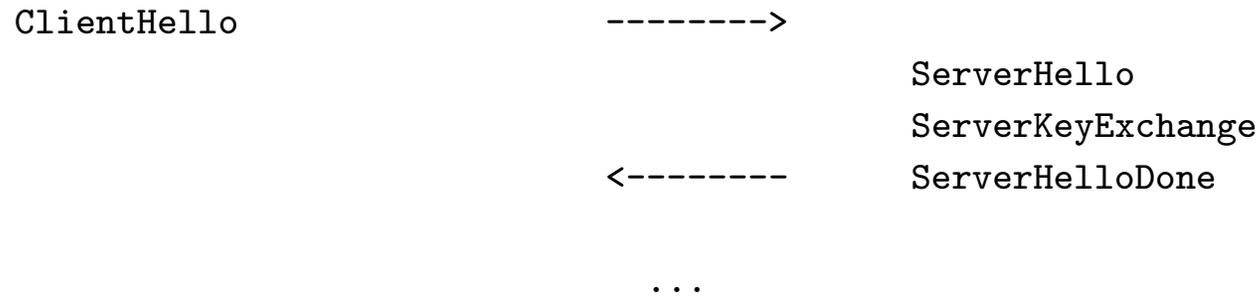
What does this provide?

- 1RTT handshake
- Privacy for the extensions (from passive attackers)
- Privacy for the client identity (from active attackers)
- Requires client-side knowledge of server parameters
- PFS for all the application data
 - ClientHello doesn't get PFS (though server can use short-lived ServerParams)

How does the client learn server parameters?

- Previous handshake (in ServerParameters message)
- Some other kind of advertisement
 - DNS (a la DANE)
 - Out-of-band signaling (e.g., SDP for DTLS-SRTP)

Variant 1: Naive client

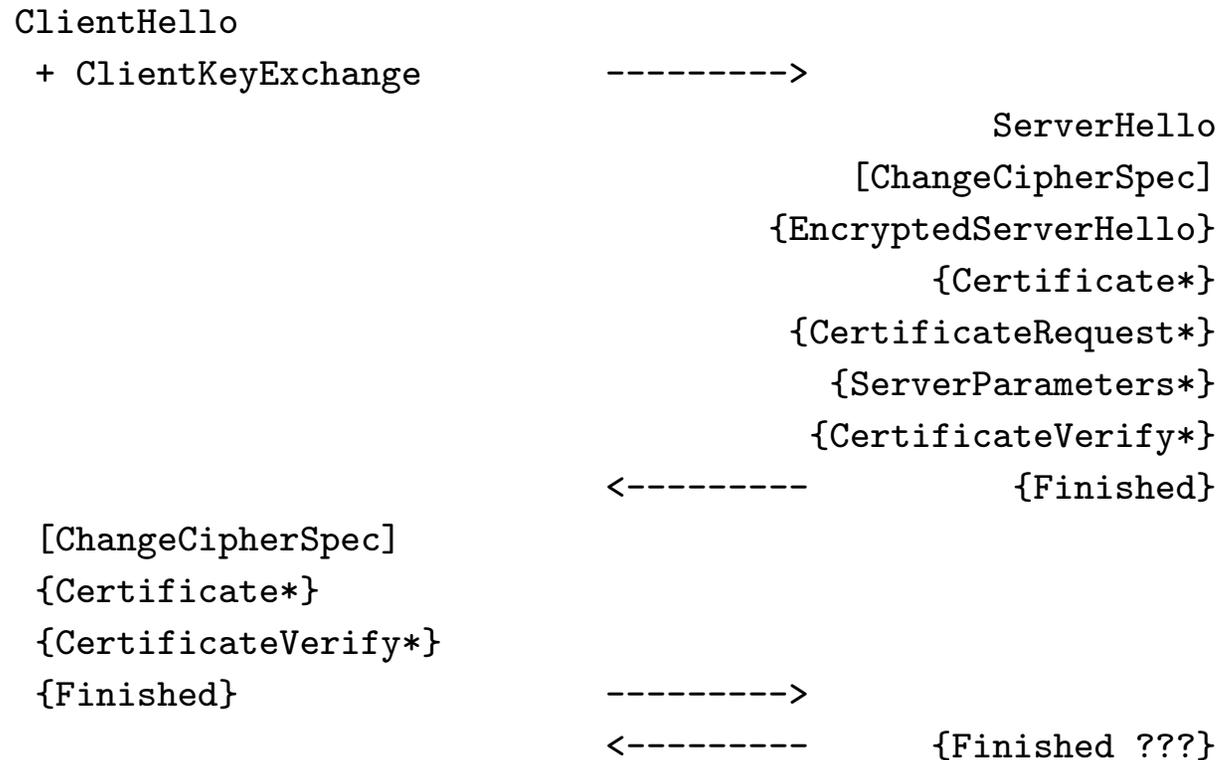


- Client solicits the server's parameters
- Server provides them
- Then you fall back to basic handshake
- This is also what happens if the client has the wrong parameters
- Client and server can use this to force PFS for the ClientHello

Intermission: Protecting server identity?

- Much of the complexity here comes from protecting SNI (and the server cert)
- If we don't protect SNI, life becomes much easier...

What we get if we decide not to protect SNI



Discussion Question: Should we protect SNI?

- Arguments for
 - Privacy
 - Pervasive monitoring
- Arguments against
 - Protocol complexity
 - Extra CPU
 - We are only getting passive protection
 - * Though maybe we could get tweak ServerParameters to get security against active attack
- Discuss

Variant 2: 0-RTT Handshakes

ClientHello

+ PredictedParameters

ClientKeyExchange

[ChangeCipherSpec]

{EncryptedExtensions

+ AntiReplayToken}

{Certificate*}

{CertificateVerify*}

{Finished}

{ApplicationData}

----->

ServerHello

ServerKeyExchange

[ChangeCipherSpec]

{EncryptedExtensions

+ AntiReplayToken}

{Certificate*}

{ServerParameters*}

{CertificateVerify*}

<-----

{Finished}

{[ChangeCipherSpec]}

{Finished???

----->

// Probably not needed

Unpacking 0-RTT

- Ordinarily, TLS uses nonces to provide anti-replay
 - The client always sends a nonce
 - In a 0-RTT handshake the server does not send one
- Alternative anti-replay approach
 - Server memorizes every client nonce and rejects replays
 - Use timestamps to minimize the amount of server storage
 - Client needs to fall back to a full handshake if the server has lost state
- This is a lot of work for the server
 - Needs to be opt-in from the server side
- Note: no PFS for the client's first flight!

Why 0-RTT?

- Milliseconds are moneyattention
- Really nice to be able to send data in first message (c.f. TFO, HTTP data in SYN)
- Issue for
 - Some big Web sites
 - DTLS-SRTP for WebRTC
 - Maybe DNS-E
- Needs to be opt-in (see above) but still means protocol complexity

Double checking that we can stop supporting RSA?

- Obviously suboptimal performance characteristics
- Complexity
 - Doesn't match the PFS pattern
 - See the handshakes above
- But everyone uses it...
 - And they have RSA certificates
 - Discuss.

Should we remove renegotiation?

- Raised by a number of people on the list
- Arguments for
 - Obvious point of complexity
 - We've had problems here before
- Arguments against
 - Change parameters
 - PFS refresh/rekey
 - To prevent cipher exhaustion (other ways to fix this)
 - Are we breaking people's actual applications
- Discuss.

Should we remove resumption?

- Servers have gotten a lot faster
 - As have our cipher suites
- Arguments for
 - Remove complexity
- Arguments against
 - People definitely use it
 - And not everyone has gone to EC
 - Some devices have gotten much slower (DICE)
- Discuss.

Remove Compression

- We don't really know how to do this securely
- The current advice is not to use it anyway
- Shal we just remove it?

Symmetric Cipher DeathMatch

- TLS specifies three types of ciphers
 - Stream ciphers (none currently defined, incompatible with DTLS)
 - Block ciphers (basically CBC)
 - AEAD ciphers (everything else)
- People are sad about CBC
- Can we just triage the cipher list to AEAD ciphers (GCM, ChaCha/Poly1305, plus maybe a few more)?

Encrypt/MAC Order

- WG moving ahead with draft-gutmann
- No need to have this be an extension for 1.3
- If we retain CBC, let's just change

Next Steps

- Dummy RFC 5246bis in submission shortly (thanks to PSA for XML translation)
- Try to make some of these decisions and edit the document accordingly