TLS 1.3

Eric Rescorla
Mozilla
ekr@rtfm.com
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
Talk Overview

• Removed features
• Current status
• Remaining work
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 \([\text{BLF}^+14]\)

- TLS 1.3 only uses predefined groups
  - Existing RFC 4492 \([\text{BWBG}^+06]\) EC groups (+ whatever CFRG comes up with)*
    - New FF groups defined in \([\text{Gil14}]\)

*Bonus: removed point format negotiation too*
Removed Feature: Renegotiation

- Previous versions of TLS allowed either side to initiate a new handshake
  - This was always kind of confusing to applications
  - And has been a source of vulnerabilities [RRDO10, BLF+14]
- TLS 1.3 simply prohibits renegotiation
Why did we want renegotiation anyway?

• Connection re-keying
  – Cryptographic exhaustion
  – PFS refresh

• Adding client authentication (or doing private client auth)

• We need to re-add at least some of this.

• For the rest, drop connection and start over
Features we need to keep

- Client authentication
- Pre-shared keys
- Session resumption (with tickets)
- Extensions (ALPN, DTLS-SRTP, etc.)
Reminder: TLS 1.2 Handshake (PFS, no client auth)

Client → Server

ClientHello

← ServerHello, Certificate

← ServerKeyExchange, ServerHelloDone

← ClientKeyExchange, [ChangeCipherSpec]

← Finished

← [ChangeCipherSpec]

← Finished

← Application Data
Basic Idea: Optimistic keying

- Client provides (EC)DHE key shares from expected groups
- Server responds with authenticated ECDHE share
- If client uses an unsupported group, server corrects
- Timing:
  - Server can send data in first flight
  - Client can send data in second flight
Basic 1-RTT TLS 1.3 Handshake

Client

ClientHello, ClientKeyShare

ServerHello, ServerKeyShare

EncryptedExtensions, Certificate, CertificateVerify, Finished\(^{K_1}\)

Finished\(^{K_1}\)

Application Data\(^{K_2}\)

Server
What if client uses an unsupported group?

Client

ClientHello, ClientKeyShare

HelloRetryRequest

ClientHello, ClientKeyShare

ServerHello, ServerKeyShare

EncryptedExtensions, Certificate, CertificateVerify, Finished

Finished

Server

Finished

Application Data

EncryptedExtensions, Certificate, CertificateVerify, Finished

Application Data

Client

ClientHello, ClientKeyShare
Backward Compatibility

- This means any new messages in first flight need to go in client extensions
  - At least for initial connections
  - Maybe always because of middleboxes
- Also questions about version number negotiation
• Measurements needed here
Client Authentication

ClientHello, ClientKeyShare

ServerHello, ServerKeyShare

EncryptedExtensions, Certificate, CertificateRequest, CertificateVerify, Finished

Certificate, CertificateVerify, Finished

Application Data
Session Resumption

• Resumption still works fine
  – ... But we just broke session tickets [SZET08]
  – And why do we have both anyway?

• Tickets are more conceptually general than resumption
  – So let’s just do tickets
Client

ClientHello, ClientKeyShare

→

ServerHello, ServerKeyShare

EncryptedExtensions, Certificate, CertificateRequest, CertificateVerify, Finished

Certificate, CertificateVerify, Finished

←

Tickets need to go here

Server
What about mid-connection client authentication?

- This was allowed in TLS 1.2 via renegotiation
  - It’s gone now
- Should be easy to put it back in technically
- But what are the semantics?
  - Retroactively bless previous messages?
  - Impact on session resumption?
- Largely application, not protocol issues
- Interaction with HTTP [BPT14, Tho14]
0-RTT

• In general we understand how to do this [Lan10]
  – Client memorizes server’s DHE parameters
  – And sends first application data
  – Server needs to keep track of every client nonce
    *
    * Typically scoped by time window and/or a context token
  – Need to fall back if server loses state

• Protocol engineering details need to be worked out
  – How does server indicate readiness to do 0-RTT?
  – How does client indicate use of 0-RTT
  – How is first-flight application data carried?

• This is next on the WG agenda
Implementations Planned/In-Progress

- NSS
- OpenSSL
- miTLS
- Pike programming language team
- Your name here

- Planning to start interop testing on -04 (1-RTT) this month
Advertisement: Interim

- Expect a call for dates on list soon
Questions?
References


