

TLSv1.3
...quite a big change

TLSv1.3

- Administrivia
- Process
- Protocol
- Issues

Administrivia

- TLSv1.3 = RFC8446
- Took 4 years to get done
- 160 pages (eek!) - do not ignore Appendices C,D and E!
- Written for implementers – you may need to read it more than once (some less clear forward references), but it's pretty readable really

List of implementations at:

<https://github.com/tlswg/tlswg-wiki/blob/master/IMPLEMENTATIONS.md>

Process

- Work started in 2014, motivations included TLS attacks seen in theory and in the wild and Snowdonia
- Represents a **major** change in the protocol - version numbering bikeshed was well painted
- Academic cryptographers worked closely with implementers to (hopefully!) ensure we don't see the same crypto/protocol failures in future
- Two academic workshops were held and the protocol design was modified numerous times to better match cryptographic theory
 - 2016 TRON: <https://www.ndss-symposium.org/ndss2016/tron-workshop-programme/>
 - 2017 TLS-DIV: <https://www.mitls.org/tls:div/>

Major Changes

- Drop less desirable algorithms and move to AEAD everywhere
- Change how new ciphersuites get defined and get RECOMMENDED
- Added “0-RTT” mode, a double-edged sword! (aka sharp implement)
- RSA key transport removed, all key exchanges provide forward secrecy
- More encryption of handshake including some extensions
- ECC is now built-in
- No more compression or custom DH groups
- Pre-shared keying, tickets and session handling all done in one way
- PKCS#1v1.5 -> RSA PSS for protocol signatures (but not certificates)
- Versioning muck – need to pretend to not be TLSv1.3 for deployment in the real world of middleboxes

TLShv1.3 Features

- These slides are **not** a replacement for reading the spec
- 1-RTT handshake
- HRR
- PSK/Resumption
- 0-RTT
- Ciphersuite re-factoring
- Key Derivation
- Versioning muck
- (Notable) extensions
- Record Protocol
- Security Properties

Full "1-RTT" Handshake

Client

Server

Key ^ ClientHello

Exch | + key_share*

| + signature_algorithms*

| + psk_key_exchange_modes*

v + pre_shared_key* ----->

ServerHello ^ Key

+ key_share* | Exch

+ pre_shared_key* v

{EncryptedExtensions} ^ Server

{CertificateRequest*} v Params

{Certificate*} ^

{CertificateVerify*} | Auth

{Finished} v

<----- [Application Data*]

^ {Certificate*}

Auth | {CertificateVerify*}

v {Finished} ----->

[Application Data] <----->

[Application Data]

Handshake with HelloRetryRequest

Client		Server
ClientHello + key_share	----->	
	<-----	HelloRetryRequest + key_share
ClientHello + key_share	----->	
		ServerHello + key_share {EncryptedExtensions} {CertificateRequest*} {Certificate*} {CertificateVerify*} {Finished} <----- [Application Data*]
{Certificate*}		
{CertificateVerify*}		
{Finished}	----->	
[Application Data]	<----->	[Application Data]

Resumption/Re-use of PSK

Client		Server
Initial Handshake:		
ClientHello		
+ key_share	----->	
		ServerHello
		+ key_share
		{EncryptedExtensions}
		{CertificateRequest*}
		{Certificate*}
		{CertificateVerify*}
		{Finished}
	<-----	[Application Data*]
{Certificate*}		
{CertificateVerify*}		
{Finished}	----->	
	<-----	[NewSessionTicket]
[Application Data]	<----->	[Application Data]
Subsequent Handshake:		
ClientHello		
+ key_share*		
+ pre_shared_key	----->	
		ServerHello
		+ pre_shared_key
		+ key_share*
		{EncryptedExtensions}
		{Finished}
	<-----	[Application Data*]
{Finished}	----->	
[Application Data]	<----->	[Application Data]

"0-RTT" Early Data

Client

Server

ClientHello

+ early_data

+ key_share*

+ psk_key_exchange_modes

+ pre_shared_key

(Application Data*) ----->

ServerHello

+ pre_shared_key

+ key_share*

{EncryptedExtensions}

+ early_data*

{Finished}

<-----

[Application Data*]

(EndOfEarlyData)

{Finished}

----->

[Application Data]

<----->

[Application Data]

“0-RTT” Issues

- “0-RTT” is a DANGEROUS IMPLEMENT

- “0-RTT” isn’t really quite accurate terminology – client needs first to have a PSK, and of course doesn’t get an answer for at least one RTT and there could be a DNS RTT first
- Browsers want to send HTTP GET requests in “first flight” - without this feature it’s likely TLSv1.3 would not have been adopted on the web
 - People need more incentives than just better security to cause them to upgrade
- Problem: **early-data can be REPLAYed**
 - Attacker records 0-RTT messages incl. early data
 - Replay that against another instance of a load-balanced server, e.g. in another data-centre where load-balanced instances can’t easily share an anti-replay cache
 - Example: DPRIVE – DNS/TLS with anycast recursives
- Bigger problem: properly handling the semantics of early-data is neither simple nor obvious, but the attraction of go-faster-stripes is simple and obvious
- Smaller problem – early-data not authenticated until server validates client’s Finished – can cause API headaches in servers, - do not act on early-data until after Finished is checked
 - Web servers might or might not (yuk) adhere to this rule, as in theory (but not in practice), HTTP GET and some other HTTP request methods are idempotent; see RFC 8470

Ciphersuite Re-factoring

- As the handshake has changed a lot, the WG decided to separate out record layer crypto from key exchange and authentication so...
- TLSv1.3 ciphersuites only reflect the record layer encryption (bulk cipher and key derivation function hash function) and not the key exchange and authentication parameters
 - TLS_AES_128_GCM_SHA256 is a TLSv1.3 ciphersuite
 - TLS_RSA_WITH_AES_128_CBC_SHA256 is a TLSv1.2 ciphersuite
- Key exchange and authentication parameters are dealt with in handshake extensions in TLSv1.3, e.g. using the key_share, supported_groups and signature_algorithms extensions in ClientHello and other handshake messages

Key Schedule/Derivation Function

```
HKDF-Expand-Label(Secret, Label, Context, Length) =  
    HKDF-Expand(Secret, HkdfLabel, Length)
```

Where HkdfLabel is specified as:

```
struct {  
    uint16 length = Length;  
    opaque label<7..255> = "tls13 " + Label;  
    opaque context<0..255> = Context;  
} HkdfLabel;
```

```
Derive-Secret(Secret, Label, Messages) =  
    HKDF-Expand-Label(Secret, Label,  
        Transcript-Hash(Messages), Hash.length)
```

HKDF is defined in RFC 5869

Key Schedule/Derivation (1/2)

```

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
```


Versioning Muck

- Middleboxes break things, so TLSv1.3 pretends to be TLSv1.2 in various ways
- supported_versions extension is where the real info is now
- ClientHello/ServerHello pretend to be TLSv1.0 or TLSv1.2
- “Dummy” change_cipher_spec messages (see Appendix D.4) make the handshake look more like TLSv1.2
- HelloRetryRequest pretends to be a TLSv1.2 ServerHello (magic values distinguish HRR)
- Record layer messages pretend to be TLSv1.2
- Absent this muck, at least 5-10% of TLSv1.3 sessions fail
- Appendix D also covers additional cases, e.g. where only some load-balanced server instances are updated at the moment (maybe due to reboots/failures or slow rollout of a new TLSv1.3 deployment)

Notable Extensions

- There are lots, some are mandatory to use for TLSv1.3, some are in-practice mandatory for the web, some not mentioned so far include:
- cookie – helps with DDoS and DTLS
- post_handshake_auth – is how TLS client auth is supported in TLSv1.3
- psk_key_exchange_modes and pre_shared_key – when using PSK
- encrypted_extensions – used from server -> client
- Some TLSv1.2 extensions remain usable in TLSv1.3 e.g. ALPN (RFC 7301)

Record Layer

- Now AEAD and differently derived keys but same max record size (2^{14} octets) and same external headers (incl. fake version)
- AEAD => “MAC-then-encrypt” issues that caused a number of problems go away

Security Properties

- See Appendix E of the spec, and the references therein, the TRON and TLS-DIV proceedings, and other publications
- Forward secrecy is not absolute – TLSv1.3 attempts to provide FS wrt long term private keys but e.g. DH public value re-use for performance reasons can result in less than perfect FS
- TLSv1.3 attempts to confidentiality protect identities, which is new. Server identity protection however cannot resist active attack.
- Separation between key purposes is much more deliberate and far less ad-hoc than earlier versions of TLS.
- Remember the security differences wrt “0-RTT”
- Traffic analysis still works – padding mechanism exists but HOWTO use it successfully is a work-in-progress

Summary

- TLSv1.3 is a real improvement in almost all respects - Other than “0-RTT,” changes are all improvements IMO, some significant
- Deployment is significant
- Careful though – it’d not be the first time we thought we’d gotten something new correct and were ultimately proven wrong