# TLSv1.3

...quite a big change

# TLSv1.3

- Administrivia
- Process
- Protocol
- Issues

## Administrivia

- TLSv1.3 = draft-ietf-tls-tls13-26
- Draft is (after 4 years) at the final stages of approval
- https://tools.ietf.org/html/draft-ietf-tls-tls13-26
- Don't worry if that -26 is incremented, it won't change much and the latest version will be fine to read
- RFC will likely pop out in a month or two
- 155 pages (eek!) do not ignore Apendices 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
- Github repo for the spec: https://github.com/tlswg/tls13-spec
- Implementations: https://github.com/tlswg/tls13-spec/wiki/Implementations

### **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
  - TRON: https://www.ndss-symposium.org/ndss2016/tron-workshop-programme/
  - 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

## TLSv1.3 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	>	
, ney_onale		ServerHello + key_share {EncryptedExtensions} {CertificateRequest*} {Certificate*} {CertificateVerify*} {Finished}
{Certificate*} {CertificateVerify*}	<	[Application Data*]
{Finished} [Application Data]	> <>	[Application Data]
[11PPIICACIOII DACA]	`	[11PPTTCacton 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	>	
1 2 3 3 3 2 2		ServerHello
		+ pre shared key
		+ key_share*
		{EncryptedExtensions}
		{Finished}
	<	[Application Data*]
{Finished}	>	[11PP+110001011 Duca ]
[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
- Motivation: browsers want to send HTTP GET requests in "first flight"
  - Without this feature it's likely TLSv1.3 would not be adopted in 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 loadbalanced 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
  - Prediction: this'll lead to headlines when it goes badly wrong
- •Smaller problem early-data is not authenticated until server has validated the client's Finished can cause API headaches in servers, but rule is to 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
  - HTTP processing of early-data: https://tools.ietf.org/html/draft-ietf-httpbis-replay-02