CTLS

draft-rescorla-tls-ctls-03

Eric Rescorla, Richard Barnes, Hannes Tschofenig

What problems we are trying to solve?

- Legacy cruft in TLS 1.3 handshake
- Ability to have reduced profiles of TLS
  - Small wire and size for constrained applications
  - "Simple" TLS for applications which don't need the entire feature set (e.g., 0-RTT)
- Clearer separation between handshake and record layer
  - Allow handshake to be used with other record layers (e.g., QUIC)

Many of these were issues we punted out of 1.3

Motivating Use Cases

- QUIC
- ATLS
- LAKE
- EAP

Two (and a half) technical pieces

- Clean up the handshake messages a bit
- A specialization mechanism for describing subsets of TLS
- More clearly delineate how to plug handshake into new record layers

Clean up handshake messages

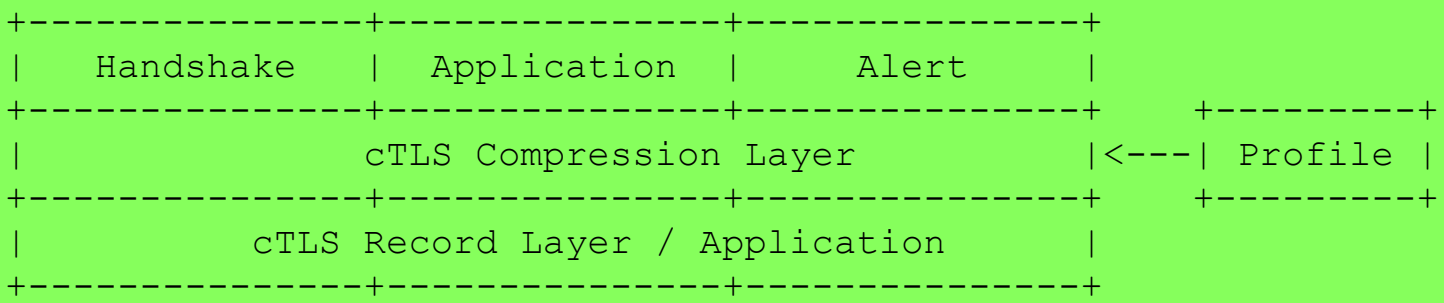
- Replace all integers with varints
- Remove some unnecessary "legacy" fields
  - E.g., session_id
- Remove handshake message length from Handshake framing
  - All messages are already self-describing

One difficulty: backward compatibility

Specialization Mechanism

- TLS is a general protocol
  - But not everyone wants all the flexibility
- General idea: *monomorphize* along individual axes (e.g., version)
  - Nail down the value of that axis
  - Remove on-the-wire representation of the negotiation point for that axis
  - Transcript is *reconstructed* to include what would have been sent
- **Specialization with forward-compatibility**
  - Remove unneeded extensions … but otherwise allow extensions
  - Compress known certificates … but also allow unknown certificates

One way of thinking about this



JSON Syntax

- Specializations are defined in a JSON syntax
- Partly just a formalism
- But also provides a machine readable form so you could automatically monomorphize
- Should we define a canonical wire encoding/defined profiles, etc.?

Example: JSON Syntax

```
{
    "version" : 772,
    "cipherSuite" : "TLS_AES_128_GCM_SHA256"
}
```

- This means “do only TLS 1.3 with AES_128_GCM_SHA256”
- Omit “supported_versions” and “cipher_suites” fields on the wire
- Decompressed transcript has single-valued fields

Predefined Extensions

- Predefined extensions don't appear on the wire
  - Generally just defined as fixed hex strings
  - But do appear in the transcript
- Otherwise extensions are encoded as usual
- All extensions have to appear in code point order
  - Except for PSK, obviously!
  - This is a change from TLS 1.3
  - … but it's compatible

Extended example


```
{
   "version": 772,
   "cipherSuite": "TLS_AES_128_CCM_8_SHA256",
   "dhGroup": "X25519",
   "signatureAlgorithm": "ECDSA_P256_SHA256",
   "randomSize": 8,
   "finishedSize": 8,
   "clientHelloExtensions": {
     "server_name": "000e00000b6578616d706c652e636f6d",
   },
   "certificateRequestExtensions": {
     "signature_algorithms": "00020403"
   },
 }
```

Known Certificates

- A map of certificates in hex and a short nickname
- Nickname just gets encoded in the CertificateEntry field
  - This means they need to be distinguishable from certs
  - Make them short, don't start with 0x30, etc.
- Expanded in the transcript like everything else

Initial Performance Numbers (short finished, random)

	ECDHE			PSK		
	TLS	CTLS	Overhead	TLS	CTLS	Overhead
ClientHello	132	50	10	147	67	15
ServerHello	90	48	8	56	18	2
ServerFlight	478	104	16	42	12	3
ClientFlight	458	100	11	36	10	1
Total	1158	302	45	280	107	21

Handshake/Record Layer Separation

- These are nominally separate but actually tied together
- QUIC separates them
  - TLS 1.3 handshake
  - Its own record layer
- Plan: firm up the interface and requirements on the "record layer"
  - Really retconning what happened in QUIC

Adopt as WG draft?