

Compact TLS (cTLS) draft-06

Eric Rescorla, Richard Barnes, Hannes Tschofenig, Ben Schwartz

TLS WG @ IETF 114

Major changes since -05

- Profile IDs are now **registrable bytestrings**
- cTLS is **no longer a compression layer!**
 - cTLS is now a *protocol generator* for protocols that are functionally equivalent to subsets of (D)TLS.
- cTLS templates are now binary objects, **not JSON**
 - A corresponding JSON format is still defined for ease of editing
- **New “handshakeFraming” option** controls handshake compaction
 - Allows template to disable fragmentation support if handshake messages will definitely be short

Profile IDs are now registrable bytestrings

opaque ProfileID<1..2⁸-1>

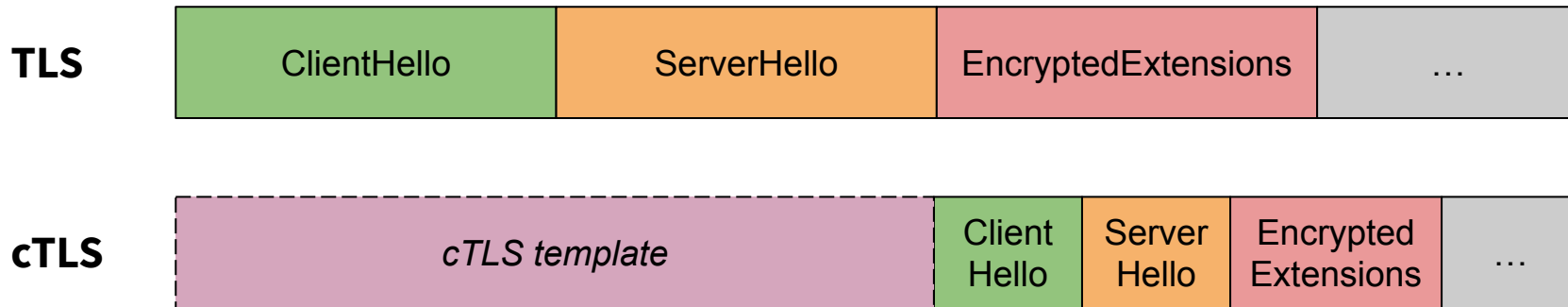
Omitted ID => use “default cTLS”

“IDs whose decoded length is 4 bytes or less are reserved.... When a reserved value is used (including the default value), other keys MUST NOT appear in the template, and a client MUST NOT accept the template unless it recognizes the ID.”

“The ID values of length 1 are subject to a "Standards Action" registry policy. Values of length 2 are subject to an "RFC Required" policy. Values of length 3 and 4 are subject to a "First Come First Served" policy. Values longer than 4 octets are not subject to registration and MUST NOT appear in this registry.”

cTLS is no longer a compression layer

- cTLS now performs validation on its own transcript
 - No need to reconstruct a standard TLS or DTLS transcript
 - Likely simpler to implement
- ...but cTLS transcripts are ambiguous!
 - **byte-identical messages are semantically different** under different cTLS templates
- **Solution:** Prepend the template to the transcript as a **synthetic message**



cTLS templates are now binary objects, not JSON

- Placing the template in the transcript requires **byte-identical agreement on the template contents**.
- Byte-identical conveyance of JSON is **extremely annoying**.
- Need a consistent binary format
 - with byte-identical reproducibility after a roundtrip through JSON!
- **Solution:** Key-value map, similar to `ClientHello.extensions` but in sorted order.

```
enum {
    profile(0),
    version(1),
    cipher_suite(2),
    ...
    optional(65535)
} CTLSTemplateElementType;

struct {
    CTLSTemplateElementType type;
    opaque data<0..2^32-1>;
} CTLSTemplateElement;

struct {
    uint16 ctls_version = 0;
    CTLSTemplateElement elements<0..2^32-1>;
} CTLSTemplate;
```

New “handshakeFraming” option

- TLS’s `Handshake` has a `uint24 length` to support long messages ($\geq 2^{16}$)
- `DTLSHandshake` adds `uint16 message_seq`, `uint24 fragment_offset`, and `uint24 fragment_length` to tolerate loss and reordering.
- cTLS is designed for compactness, so handshake messages are likely to fit in one record.
- New option: **handshake_framing = true/false**
 - **true:** Use `Handshake` or `DTLSHandshake` as usual. Long messages and fragmentation allowed.
 - **false:** Use `CTLSHandshake` or `CTLSDatagramHandshake`. The length and fragmentation fields are omitted.
- Like DTLS 1.3, the transcript always uses `Handshake` messages.

Interesting Questions

- What to do about Elliptic Curve compressed representations?
 - **Proposal:** Handle this independently as a separate draft registering new codepoints.
- Should we support compression of `CertificateEntry.extensions` and `HelloRetryRequest.extensions`?
 - **Proposal:** Support compression of extensions only on compressed certificates.
 - **Proposal:** Compress `HelloRetryRequest.extensions` independently from other messages.
- How do we version cTLS?
 - Currently, cTLS version is 0, independent of TLS version (which can be pinned or negotiated).
 - Not clear how well forward-compatibility will work for future versions of TLS/DTLS!
- Can we omit empty messages?
 - Are we sure that the recipient can always reconstruct the omitted messages? What about in future versions of TLS?
- Many other details still open!

close_notify