OpenSSL Foundation

DTLS 1.3

Ryan Hooper

October 14

Timeline

- First PR
  - October 3rd, 2023
  - Adds initial DTLS 1.3 definition
  - By Frederik Wedel-Heinen
- ACK PR merged
  - August 29th, 2025
  - Most complex part of the feature that is uniquely DTLS 1.3
- Last Commit on Feature Branch
  - August 21st, 2026
- Time Lapse
  - 2 Years, 10.5 Months
  - 1,053 Days!!!!

Why am I giving this talk?

- For over a year I have been splitting my time between Cisco Systems and working with the OpenSSL Foundation.
- DTLS 1.3 is an important feature for Cisco.
- About a year ago I started to get involved with DTLS 1.3, initially working on the remaining TODO’s in the code and enabling commented out tests.

What is DTLS 1.3?

- Based off of TLS 1.3 and DTLS 1.2
- The handshake looks like TLS 1.3 with the addition of some elements
  - Reduces the number of Round Trips
  - Removes insecure/weak ciphers
  - Supports Post Quantum algorithms: ML-KEM and ML-DSA
  - Utilizes Hello Retry Request
- Retains DTLS 1.2 processing and fragmentation support
- New Elements to DTLS 1.3
  - Unified Header
    - Sequence Number Encryption
  - ACK Record
  - Post Handshake Messages [KeyUpdate/NewSessionTicket] utilize ACK Responses
  - Connection ID (CID) – Not Supported

DTLS 1.3 Unified Header

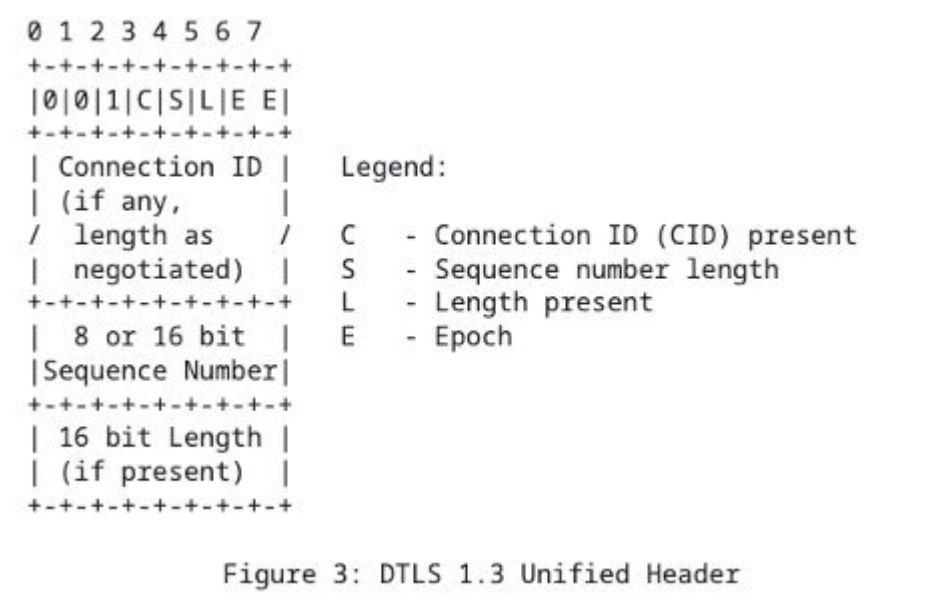


Figure 3: DTLS 1.3 Unified Header

DTLS 1.3 Unified Header

- Epoch
  - Only the last two bits are sent in the Unified Header
  - Can range from 0 to 2^{64}
    - RFC 9147 Section 4
  - Handshake Epochs
    - 0 (Client Hello and Server Hello)
    - 2 (Remaining DTLS Messages up to including the Client and Server Finished)
  - Early Data
    - 1 (Client-Only)
    - DTLS 1.3 does NOT send an end of early data message, instead the changes in the epoch Value signifies leaving early data
- Sequence Number starts at zero, but for Epoch 1+ the sequence number is encrypted
  - Encrypted for an extra layer of metadata protection

DTLS 1.3 ACK Record

```
struct {
    uint64 epoch;
    uint64 sequence_number;
} RecordNumber;
```

```
struct {
    RecordNumber record_numbers<0..2^16-1>;
} ACK;
```

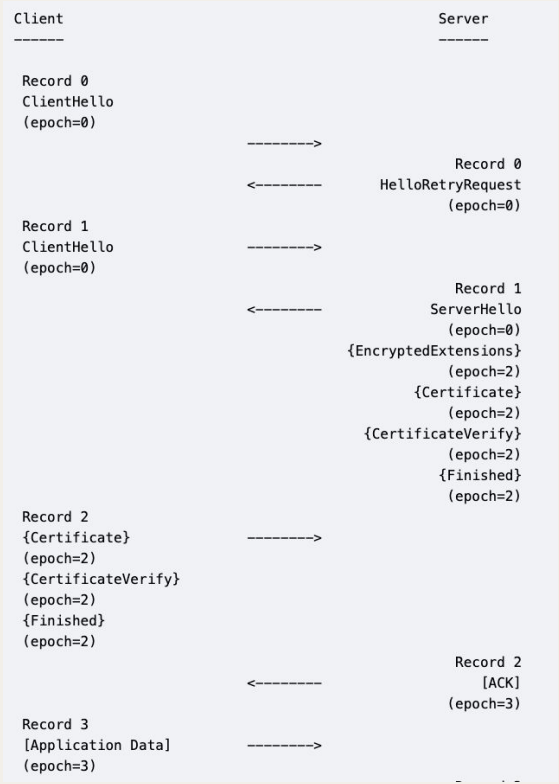
DTLS 1.3 ACK Record

- An ACK can contain a list of Record Numbers being acknowledged
- Delivered/Wrapped by a Unified Header
- Explicitly Acknowledges
  - Client Finished
  - NewSessionTicket
  - KeyUpdate
  - Post-Handshake Client Authentication

DTLS 1.3 Key Update

- Signals a key change, like TLS 1.3, but read and write directions update independently
- Read keys: update immediately on receipt (same as TLS 1.3)
- Write keys: currently updated immediately after sending the KeyUpdate
  - Matches TLS 1.3's behavior (RFC 9147 §8 permits deferring instead)
  - Known issue #32878: This is being resolved now
- Next release: write keys will defer until our own KeyUpdate is acknowledged
  - Keeps using the current (still-valid) write epoch so the KeyUpdate message itself stays retransmittable if lost
- Read epoch and write epoch are tracked and incremented independently

DTLS 1.3 Simple Handshake



DTLS 1.3 DTLSv1_listen

- Does NOT support DTLS 1.3
- Supports DTLS 1.2/1.0
- Stateless Solution
- Validates Clients with Hello Verify Requests
  - Doesn’t require fragment reassembly
- User must set the cookie generation and verification callbacks
  - SSL_CTX_set_cookie_generate_cb
  - SSL_CTX_set_cookie_verify_cb
- Caller passes in a server-role SSL object; DTLSv1_listen() updates that same object in place once a valid cookie arrives
- User must finish the handshake by calling SSL_accept

DTLS 1.3 DTLS SSL Listener

- Supports DTLS 1.3 and DTLS 1.2/1.0
- Validates DTLS 1.2/1.0 Clients with Hello Verify Requests
- Validates DTLS 1.3 Clients with Hello Retry Requests
- Has default cookie generation and validation functionality, but user can still set their own by calling
  - Hello Verify Request
    - SSL_CTX_set_cookie_generate_cb
    - SSL_CTX_set_cookie_verify_cb
  - Hello Retry Request
    - SSL_CTX_set_stateless_cookie_generate_cb
    - SSL_CTX_set_stateless_cookie_verify_cb
- Validation is on by default
  - SSL_LISTENER_FLAG_ADDRESS_VALIDATION forces validation on; if combined with NO_VALIDATE, this one wins (fail-safe)
- Validation can be turned off by setting SSL_LISTENER_FLAG_NO_VALIDATE when initializing the listener

DTLS 1.3 DTLS SSL Listener – 2

- Utilizes a packet demux to pass packets from the listener to SSL objects
- Supports Multiple Threads
  - Can use SSL_LISTENER_FLAG_SINGLE_THREAD when initializing the listener to run in a single thread
- User uses SSL_accept_connection to obtain a server-role SSL object that has been validated
- User must finish the handshake by calling SSL_accept

DTLS 1.3 DTLS SSL Listener Main APIs

- SSL_listen
  - Takes in an DTLS SSL Listener
  - Sets the listening flag
- SSL_accept_connection
  - Returns an SSL Object that is ready for the application
  - If doing HVR/HRR only connections that have been validated are returned
- SSL_get_peer_addr
  - Takes in SSL object and return address parameter of the Peer's address
  - Only way to identify which client a connection belongs to, since all connections share one socket
- SSL_poll
  - Used to wait for specific events
  - Takes in SSL objects and requested events and associated data for the poll
  - Returns 1 on success / 0 on failure; the count of ready items is written to the result_count out-parameter
- SSL_set_blocking_mode
  - Default: blocking
  - Takes in the Listener, or a connection it created, plus 1 to enable blocking / 0 to disable blocking
  - Calls to SSL_read, SSL_accept_connection and SSL_accept will block until they can complete

DTLS 1.3 DTLS SSL Listener Security APIs

- SSL_set_generic_value_uint and SSL_get_generic_value_uint
  - Takes in SSL listener object, Option ID and value
- Options:
  - SSL_VALUE_DTLS_LISTENER_MAX_PENDING_CONNS
    - Default is 256
    - Takes in the SSL Listener and max number of pending connections
    - Zero/disabling is not an acceptable value, without this set this is a resource allocation attack vector
  - SSL_VALUE_DTLS_LISTENER_PENDING_TIMEOUT
    - Default 30 seconds
    - Takes in the SSL Listener and timeout in milliseconds
    - Rejects a 0 timeout
    - Used to prevent Denial of Service by removing pending connections not making any progress
  - SSL_VALUE_DTLS_LISTENER_MAX_DGRAM_SIZE
    - Default value is 2000 bytes
    - Sets the max receive buffer of the DTLS SSL listener
    - Max excepted size is 65,535 bytes

DTLS 1.3 DTLS SSL Listener Connection Phases

- Pending Connection
  - Initial Client Hello was received
  - Awaiting HVR or HRR response
  - Default max number of pending connections is 256
  - Default 30 second timeout until connection is removed
- Incoming Connection
  - HVR or HRR has been validated
  - If no cookie validation, then initial Client Hello has been processed
  - Ready for application to take ownership via SSL_accept_connection
- Application Owned Connection
  - DTLS SSL Listener no longer owns the SSL Connection
  - Traffic is still routed through the listener, so we still need to keep track of it
  - Obtained via SSL_accept_connection

DTLS 1.3 DTLS SSL Listener Blocking verse Non-Blocking

- Blocking
  - Default Behavior for listener and all connections created by the listener
  - SSL_read, SSL_accept_connection and SSL_accept will block until they can complete
  - Blocking does not mean the socket blocks — the library waits for readiness: on the listener, until an incoming connection is ready; on a connection, until there is data to read or write.
- Non-Blocking
  - Set by SSL_set_blocking_mode passing in listener or connection plus 0
  - SSL_read, SSL_accept_connection and SSL_accept will return right away if it cannot complete the request
  - Use SSL_poll to wait for readiness instead of spin-looping your application
- SSL_poll and blocking
  - SSL_poll can block indefinitely depending on the timer value provided
  - Application calls SSL_poll and waits until data arrives; once SSL_poll reports readability, the following SSL_read completes without blocking

DTLS 1.3 DTLS Echo Demo

- Located in the source code at demos/dtlsecho
- Need to be built with enable-demos
- How it works:
  - A simple demo where a DTLS 1.3 client makes a connection to a DTLS 1.3 server.
  - The client reads data from standard input and sends it to the server.
  - The server displays the data and sends it back to the client.
  - The client then displays the echo’d data.
- How to run it
  - Terminal 1/Server: ./dtlsecho s
  - Terminal 2/Client: ./dtlsecho c localhost

DTLS 1.3 DTLS Listener Echo Demo

- Located in the source code at demos/dtlslistenerecho
- Need to be built with enable-demos
- How it works:
  - A simple demo where a DTLS client makes a connection to a DTLS Listener.
  - The client reads data from standard input and sends it to the server.
  - The server displays the data and sends it back to the client.
  - The client then displays the echo'd data.
  - The DTLS listener is multithreaded
  - The DTLS listener can accept up to 10 concurrent connections
- How to run it
  - Terminal 1/Server: ./dtlslistenerecho s
  - Terminal 2/Client: ./dtlslistenerecho c localhost
    - Optional argument [dtls12|dtls13] that can make either a DTLS 1.2 or DTLS 1.3 connection

DTLS 1.3 Debugging

- Use PCAPs
- Know what state you are in when processing a DTLS 1.3 Message
  - I added simple fprints in:
    - statem_clnt.c
      - ossl_statem_client_process_message
      - ossl_statem_client_construct_message
    - statem_srvr.c
      - ossl_statem_server_process_message
      - ossl_statem_server_construct_message
- Function dtls_get_more_records in dtls_meth.c
  - This function does a lot of processing of the DTLS 1.3 Unified Header and is a great place to start
  - Remember the Sequence number for Unified Headers are encrypted and inside this function it calls dtls_crypt_sequence_number which decrypts the sequence number
- Function dtls1_read_bytes in rec_layer_d1.c
  - Calls dtls_get_more_records
  - A return of -1 means a real error occurred; 0 can just mean "no data yet" in non-blocking use — only return 1 with readbytes set means the read actually succeeded

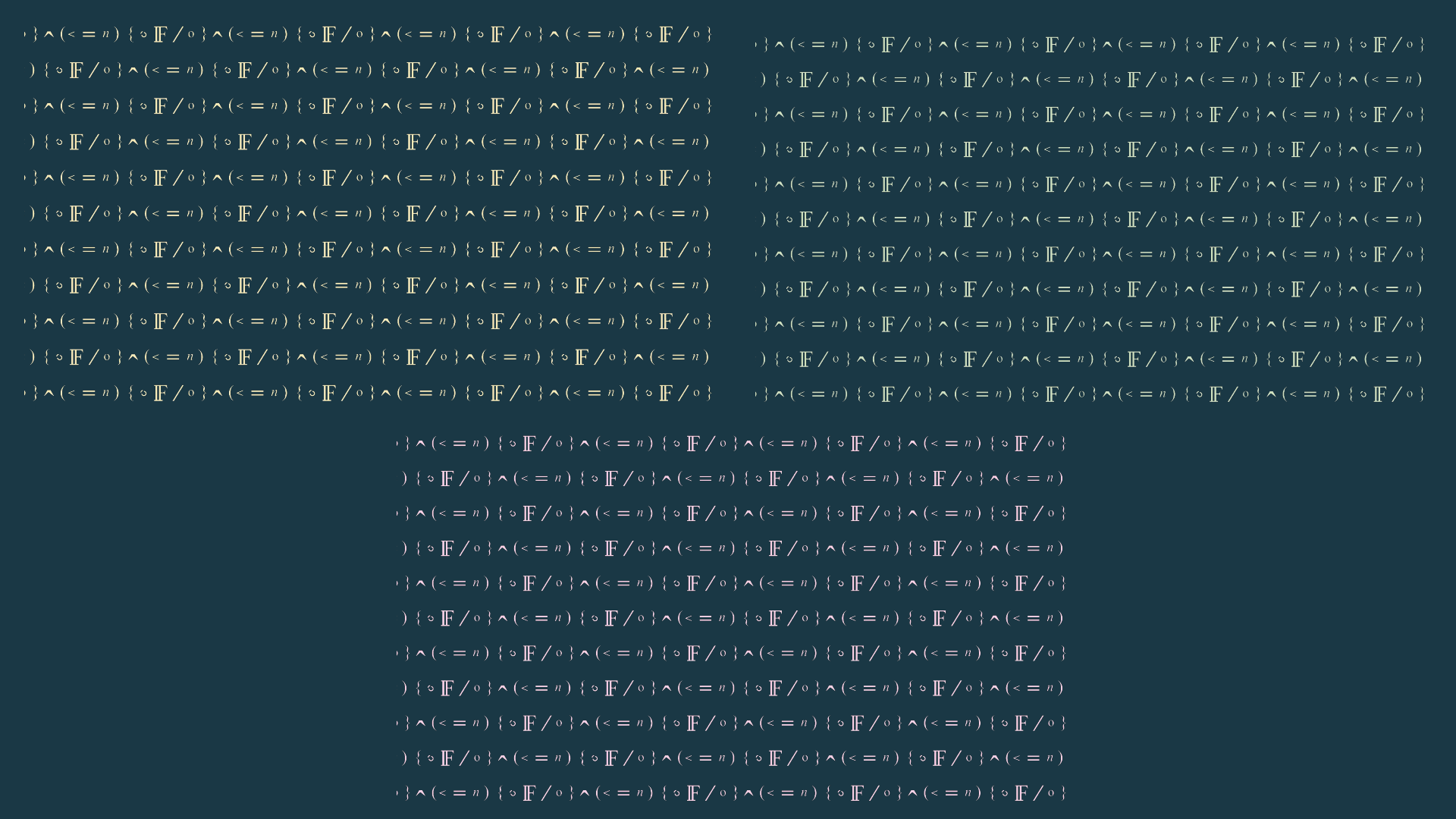
DTLS 1.3 Documentation

- DTLS 1.3 (general)
  - ossl-guide-dtlsv13(7) — DTLS 1.3 guide/overview, explicitly built around RFC 9147
  - DTLS_set_timer_cb(3), DTLSv1_get_timeout(3), DTLSv1_handle_timeout(3) — retransmission timer APIs
  - DTLS_get_data_mtu(3) — MTU handling
- DTLS SSL Listener
  - SSL_new_listener(3) — the main listener API reference (creation, SSL_accept_connection, blocking-mode flags, DTLS-specific tunables)
  - DTLSv1_listen(3) — the legacy stateless HelloVerifyRequest-only listen API
  - SSL_poll(3) — polling/readiness API

DTLS 1.3 Resources

- RFC 9147
  - https://www.rfc-editor.org/info/rfc9147/
- Helpful Website walking through the DTLS 1.3 Handshake
  - https://dtls.xargs.org/

Thank You Contributors

Many people were involved in this effort

- Frederik Wedel-Heinen
- Matt Caswell
- Mounir Idrassi
- Tomas Mraz
- Jakub Zelenka
- Timothy Copeland
- And Many Others