Overview

- DTLS version of TLS 1.3
- Still presented as a delta from TLS 1.3
- Some improvements/cleanup
- Partly informed by early implementation experience
Document Status

- Individual submission
- Currently in call for acceptance
- Here to talk about the draft...
Issue #2: ACKs

• DTLS historically used an implicit ACK
  – Receiving the start of the next flight means the flight was received
• Simple (but also simpleminded)
  – Slightly tricky to implement
  – Gives limited congestion feedback
  – Handles single-packet loss badly
• Interacts badly with some TLS 1.3 features (like NST)
• Solution: introduce an explicit ACK
Where should we ACK?

- Every flight
- Just at the end of things that aren’t explicitly acknowledged
  - Client Finished
  - NewSessionTicket
- Proposal: allow ACKs at any time
  - This allows partial retransmit of flights (if we SACK)
  - Also just means one trigger for state machine evolution
Strawman ACK format (not what’s in the draft)

```c
struct AcknowledgedMessage {
    uint16 message_seq;
    uint32 timestamp;
};
```

```c
struct {
    AcknowledgedMessage messages<2..2^16-2>;
} DtlsAck;
```

- This is a compromise between “lots of data” and “convenient”
- We could also include the DTLS records for more path feedback
What epoch should ACKs be encrypted under?

- Issue here is key transitions
- E.g., ACK of the client Finished
  - Natural to match the epoch of what you’re ACKing
  - But this may mean you have two keys
Key Update

- Key Update in TLS 1.3 is unreliable
  - This means new epoch records may appear before KeyUpdate
- Current draft just omits KeyUpdate
  - KeyUpdate from one side triggers another
  - Only one unacknowledged KeyUpdate allowed outstanding
  - Can’t unilaterally update
- Potential alternative design
  - Send KeyUpdate (using the ACK for reliability)
  - Still have to process out-of-order records
Shrinking the Packet Header

• DTLS packet header is very large

struct {
  ContentType opaque_type = 23; /* application_data */
  ProtocolVersion legacy_record_version = {254,253}; // DTLSv1.2
  uint16 epoch; // DTLS-related field
  uint48 sequence_number; // DTLS-related field
  uint16 length;
  ...
}

• Would be nice to make it smaller
  – Give us room for connection ID...
A shorter header (due to MT)

001eesss sssssssss

Where ee = epoch modulo 4 and ss..ss = sequence number modulo 2048

• Why two bits for the epoch?

• What about long header/short header as in QUIC draft?
Connection ID: Problem and Solution

- Demultiplexing based on 5 tuple
- If NAT binding expires server cannot find security context.
- Happens if IoT devices enter a sleep cycle.
- Solution: Add additional identifier to record layer header.
Connection ID: Design Decisions

- Should there be a negotiation?
- What messages should use it?
  - Everything except ClientHello (for backwards compatibility)?
  - Messages protected using keys derived from a handshake_traffic_secret
- Unlinkability property desirable to some: possible approaches
  - Static identifier similar to IPsec SPI
  - Hash chain/Counter-based approach
  - Token bucket with receiver-side refresh
- Somehow you need to demux these
  - Either outside DTLS
  - Or change the stack somehow
Hash chains

- Peers exchange $K_u$
  - Packet $i$ uses conn id $E(K_u, i)$ (or $\text{truncate}(H^i(K))$)

- Expensive to process with reordering
  - Either precompute a bunch of values (memory cost)
  - Or you have to compute packet $i$ for all $K_u$ when you get an out-of-order packet
Token Buckets

• Server has a single static key $K$

• Server gives client $n$ tokens $T_0, T_1, ... T_n$
  \[- T_i = E(K, u||i)\]

• Client uses a fresh token for each packet

• Server replenishes tokens in response to packets

• Consumes a lot of bandwidth
  \[- Each \ token \ is \ sent \ twice\]
  \[- Tokens \ have \ a \ minimum \ size\]
Handshake Message Transcript

- The TLS and DTLS transcripts are different
- Both include the message header
  - But headers are different
  - DTLS includes a (synthetic) DTLhandshake message header
- We could just do the TLS message header
  - Cross-version consistency between cross-protocol consistency
Other issues?