CCNx-KE vs (D)TLS

Marc Mosko and Christopher A. Wood
PARC, UCI
ICNRG 95 - Buenos Aires - 4/3/16
CCNx-KE is...

- A key-exchange protocol used to establish a common key between a client and producer.
- Inspired by TLS 1.3, QUIC, and DTLS.
- Modified for CCN communication.
Goals

• Near-parity with TLS 1.3.

• Session keys must be forward-secure.

• At most 2 RTTs to establish a session key, with the possibility for session resumption in 1 RTT.

• Session movement or relocation.

• Operate with an unreliable and connectionless transport mechanism.
Updates

• DTLS-like cookie for server-side DoS prevention.
  • No more 0-RTT resumption.
• New key schedule and traffic key derivation (parity with TLS 1.3)
• Explicit sequence numbers in post-exchange interest messages (for stateless encryption).
• Description of how the application data is encrypted in interests and content using TLV encapsulation.
• Client-provided prefix extension.
• Removed redirection prefix between Rounds 1 and 2.
• Miscellaneous writeup improvements.
• Addition of a pre-shared-key (PSK) mode.
Old Key Calculation

DH-1

Static secret

SS

DH-2

key exchange encrypted with SS

FSK

Traffic secret
New Traffic Key Calculation

- DH-1
  - SS
    - MS
      - TS
    - ES
      - MS
      - ES
  - DH-2
    - SS
      - MS
      - TS
    - ES
      - MS
      - ES

- Ephemeral secret
- Master secret
- Traffic secret
- Master secret
## Keying Material Sources

<table>
<thead>
<tr>
<th></th>
<th>SS</th>
<th>ES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normal exchange</strong></td>
<td>(cleartext) client and server ephemeral shares</td>
<td>(cleartext) client and server ephemeral shares</td>
</tr>
<tr>
<td><strong>Resumption (1-RTT)</strong></td>
<td>(cleartext) client ephemeral and server (CONFIG) static shares</td>
<td>(cleartext) client and server ephemeral shares</td>
</tr>
<tr>
<td><strong>PSK</strong></td>
<td>PSK</td>
<td>PSK</td>
</tr>
</tbody>
</table>
Important Differences

- Some features were designed with CCN communication in mind, e.g., session mobility

- We compare CCNx-KE to TLS 1.3 and DTLS 1.2
  - TLS 1.3 inherited many properties of QUIC
  - QUIC “will be replaced by TLS 1.3 in the future, but QUIC needed a crypto protocol before TLS 1.3 was even started” [1]

## CCN\textsuperscript{x}-KE vs TLS 1.3

<table>
<thead>
<tr>
<th>Feature</th>
<th>CCN\textsuperscript{x}-KE</th>
<th>TLS 1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record layer</td>
<td>Interests and content objects are encapsulated and assigned explicit sequence numbers</td>
<td>Streamed record layer</td>
</tr>
<tr>
<td>Session secret usage</td>
<td>Handoff to other parties</td>
<td>Pinned to the server</td>
</tr>
<tr>
<td>Transport mechanism</td>
<td>Unreliable datagrams</td>
<td>TCP</td>
</tr>
<tr>
<td>Resumption</td>
<td>Stateful resumption cookies</td>
<td>Opaque labels</td>
</tr>
<tr>
<td>DoS prevention</td>
<td>DTLS-like cookies</td>
<td>Defers to TCP</td>
</tr>
</tbody>
</table>
TLS: Encrypted Streams

TLS record layer:

- Translates plaintext into ciphertext packets
- Assumes in-order arrival of packets (no state information is passed)

```c
struct {
    ContentType opaque_type = application_data(23); /* see fragment.type */
    ProtocolVersion record_version = { 3, 1 };  /* TLS v1.x */
    uint16 length;
    aead-ciphered struct {
        opaque content[TLSPlaintext.length];
        ContentType type;
        uint8 zeros[length_of_padding];
    } fragment;
} TLSCiphertext;
```
CCNx-KE: Encrypted Datagrams

CCNx-KE encryption layer:

- CCN messages are wrapped in an “outer context” that identifies:
  - Routable prefix
  - Session ID (key ID)
  - Sequence number (salt or nonce)
- Wrapped messages are called the “inner context” and are plain, unmodified CCN messages
TLV Encapsulation

We use a new T_ENCAP TLV to do this:

\[
\begin{align*}
+------------------+ \\
| T1 | L1 | V1 | \\
+------------------+ \\
\mid \mid \\
\mid \downarrow \\
+------------------+ \\
| T2 | L2 | Enc((T1,L1,V1)) | \\
+------------------+
\end{align*}
\]

The validation information (e.g., AES-GCM tag) is contained in a separate Validation TLV.
DoS Cookies

- DoS prevention in TLS is provided by TCP, e.g., SYN-flood cookies
- CCNx-KE is connectionless and therefore introduces a unique type of cookie
Cookie Usage

Consumer

BARE-HELLO interest
+ Cookie challenge

BARE-HELLO content
+ CONFIG parameters
+ Cookie

HELLO interest
+ Cookie proof
+ Cookie

HELLO content
+ ...

Producer
Cookie Generation

- Cookie challenge:
  \[ Y = H(X) \text{ where } X \leftarrow \{0,1\}^{128} \]

- Cookie:
  \[ P = \text{timestamp} \ || \ \text{HMAC}_k(Y, \text{timestamp}) \]

- Cookie proof:
  \[ \text{timestamp} \ || \ X \]
Cookie Generation

• Cookie challenge:
  \[ Y = H(X) \text{ where } X \leftarrow \{0,1\}^{128} \]

• Cookie:
  \[ P = \text{timestamp} || \text{HMAC}_k(Y, \text{timestamp}) \]

• Cookie proof:
  \[ \text{timestamp} || X \]

Cookie check:
- Verify freshness of the timestamp
- Verify HMAC tag using X and timestamp
Session Mobility

- In TLS, sessions are pinned to endpoints
- In CCNx-KE, we allow for sessions to be migrated from one (producer) endpoint to another
  - Clients participate in generating a move proof
  - Producer endpoints provide a corresponding move token and new routable prefix (endpoint destination)
  - Both the proof and token must be presented to the server
Mobility Example

- **Consumer**
  - (round 2 interest)
  - + move challenge

- **Producer**
  - (round 2 content)
  - + move token
  - (application data interest)
  - + move token, move proof

- **Replica**
  - (application data content)
  - + new session ID

(continued communication)
Mobility Example

Consumer

 Producer

 Replica

(application data content)
+ new session ID

(round 2 interest)
+ move challenge

(round 2 content)
+ move token

(application data interest)
+ move token, move proof

(continued communication)
Resumption Cookies

- In TLS, resumption cookies are opaque identifiers.
  - The client and server negotiate a cookie and use it as a PSK (pre-shared key) when resuming a session later on.

- In CCNx-KE, resumption cookies contain state.
  - They allow a server to recover state information instead of storing it for each PSK.

**
Resumption Cookie (RC)*

**Structure:** Encryption of TS and the (MovePrefix, MoveToken) tuple (if provided), with a producer secret key that is also known to the service operating under MovePrefix (if provided)

ResumptionCookie = Enc(k2, TS || ( (MovePrefix || MoveToken) ))

**Usage:** The SessionID and ResumptionCookie are needed to resume a session (i.e., recompute SessionID and check for equality):

(TS || ( (MovePrefix || MoveToken) ))) = Dec(k2, ResumptionCookie)

SessionID = Enc(k1, H(TS || (MovePrefix)))

*This is only one way to create the RC*
### CCNx-KE vs DTLS 1.2

<table>
<thead>
<tr>
<th>Feature</th>
<th>CCNx-KE</th>
<th>DTLS 1.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoS Prevention Cookie</td>
<td>Cookie derived client-provided hash premiere and timestamp</td>
<td>Undefined</td>
</tr>
<tr>
<td>Cipher suite options</td>
<td>AES-GCM (mandatory) with options for Salsa20+Poly1305</td>
<td>Stream ciphers (RC4), non-AEAD block ciphers</td>
</tr>
<tr>
<td>Timeout and retransmission</td>
<td>TBD</td>
<td>Grouped message retransmissions</td>
</tr>
</tbody>
</table>
Outstanding Items and Open Questions

- Define the timeout and retransmission policy.
- Should we remove the resumption cookie and make it an opaque identifier as in TLS 1.3?
Implementation Status

• Round 1 exchange complete
• Round 2 in progress
• Session migration not implemented
• Client authentication not implemented