CCNx Key Exchange

IETF 94 - Yokohama - ICNRG
Christopher A. Wood
November 5, 2015
Motivation and Goals

Motivation

• We need a way to establish session keys between consumers and producers that makes use of CCN properties
  • (D)TLS, QUIC, etc. are a good start

Requirements

• Session keys must be forward secure
  • Compromising long-term secrets does not put session keys at risk
• At most 2 RTTs to establish a session key, with the possibility for session resumption in 0 RTT
• Allow extensions for client authentication in addition to server authentication
TLS and QUIC Overview

- Support 0-, 1-, and 2-RTT forward secure key derivation
  - Long-term public key shares enable faster handshakes
- Different keys are used to encrypt (and MAC) different parts of the protocol
  - A short-term ephemeral key is used for exchanging random key shares to derive a master key
- Server is authenticated to the client
- Prevents address spoofing (via SYN cookies) and replay attacks (via QUIC Source Address Tokens and TLS nonce)
CCNx Key Exchange (CCNx-KE)
Assumptions

• Consumers know the prefix of the target producer, e.g., /prefix/

• Consumers possess the appropriate trust anchors to authenticate the server

• … that’s it
Protocol Overview

• Round 1: Obtain the server config (if not available or it has expired)

• Round 2: FULL HELLO handshake and establish ephemeral keys

• Round 3: Final exchange to derive forward-secure secrets for all subsequent communication
Sketch of the Full Protocol

1. Interest: BARE HELLO
2. ContentObject: BARE HELLO reject response
3. Interest: FULL HELLO
4. ContentObject: FULL HELLO response
5. Interest: FINALIZE and data
6. ContentObject: FINALIZE response and data
Sketch of the Full Protocol

1. **Interest[/prefix/nonce1]**
   - **payload**: (HELLO)

2. **CO[/prefix/nonce1]**
   - **payload**: Config, nonce2, salt

3. **Interest[/prefix/nonce2]**
   - **payload**: ClientShare1, {AlgorithmOptions, \(y \text{ s.t. } H(y) = \text{nonce1}\), ClientShare2}_SS

4. **CO[/prefix/nonce2]**
   - **payload**: [SessionID, {RC}_FSK-P), {ACK, ServerShare2}_SS] | {REJ, Reason}_SS

5. **Interest[/prefix/SessionID/{…}_FSK-C]**
   - **payload**: {ConsumerData}_FSK-C

6. **CO[/prefix/SessionID/{…}_FSK-C]**
   - **payload**: {ProducerData}_FSK-P
SS Derivation

\[ SS = HKDF(Salt, IKM) \]
\[ Salt = CSALT1||PSALT1||"ss generation" \]
\[ IKM = 32\text{-}byte key\text{-}exchange output \]
FSK-C/P Derivation

Second key exchange uses the ServerShare2 and ClientShare2 inputs

$$\text{FSK} = \text{HKDF}(\text{Salt}, \text{IKM})$$

$$\text{Salt} = \text{CSALT2}||\text{PSALT2}||"\text{fsk generation}"$$

$$\text{IKM} = \text{Second 32-byte key-exchange output}$$

FSP-C/P and IVs are pumped from FSK in the following order:

1. FSK-C
2. FSK-P
3. FSK-CIV (client IV)
4. FSK-PIV (producer IV)
SessionID and RC Properties

SessionID

• Used to uniquely identifies a single session
• ... a random string/number suffices

RC

• Used to recover SS and FSK for a given session
Option #1: HELLO prefix redirection

1. **C**
   - Interest[/prefix/nonce1]
   - payload: (HELLO)

2. **P**
   - CO[/prefix/nonce1]
   - payload: Config, nonce2, salt, **prefix2**

3. **C**
   - Interest[/prefix2/nonce2]
   - payload: ClientShare1, {AlgorithmOptions, <y s.t. H(y) = nonce1>, ClientShare2}_SS

4. **P**
   - CO[/prefix2/nonce2]
   - payload: [SessionID, ({RC}_FSK-P), {ACK, ServerShare2}_SS | {REJ, Reason}_SS]

5. **C**
   - Interest[/prefix2/SessionID/{...}_FSK-C]
   - payload: {ConsumerData}_FSK-C

6. **P**
   - CO[/prefix2/SessionID/{...}_FSK-C]
   - payload: {ProducerData}_FSK-P
Option #2: Final prefix redirection

1. Interest[/prefix/nonce1]
   payload: (HELLO)

2. CO[/prefix/nonce1]
   payload: Config, nonce2, salt

3. Interest[/prefix/nonce2]
   payload: ClientShare1, {AlgorithmOptions, \(<y \text{ s.t. } H(y) = nonce1>, \text{ClientShare2}\}_SS

4. CO[/prefix/nonce2]
   payload: [SessionID, ({RC}_FSK-P), {ACK, ServerShare2, \(\text{(prefix3, MoveToken)}\)}_SS | {REJ, Reason}_SS]

5. Interest[/prefix3/SessionID/{…} _FSK-C]
   payload: \{MoveToken, ConsumerData\}_FSK-C

6. CO[/prefix3/SessionID/{…} _FSK-C]
   payload: \{ProducerData\}_FSK-P
Option #3: Resumption Cookie Echo

1. Interest[/prefix/nonce1] payload: (HELLO)

2. CO[/prefix/nonce1] payload: Config, nonce2, salt

3. Interest[/prefix/nonce2] payload: ClientShare1, {AlgorithmOptions, \( y \) s.t. \( H(y) = nonce1 \), ClientShare2}_SS

4. CO[/prefix/nonce2] payload: [SessionID, \{RC\}_FSK-P, \{ACK, ServerShare2\}_SS | \{REJ, Reason\}_SS]

5. Interest[/prefix/SessionID/{…}_FSK-C] payload: \{ConsumerData\}_FSK-C

6. CO[/prefix/SessionID/{…}_FSK-C] payload: \{ProducerData\}_FSK-P, \{RC\}_FSK-P
CCNx-KE Properties

• Minimal deviation from TLS and QUIC.

• Forward-secure session keys derived similar to TLS and QUIC.

• Server-to-client authentication.
  • Client-to-server authentication is future work.

• Clients are securely bound to the protocol execution (via the hash-based tokens).

• Session state can be securely migrated from the producer to a trusted party.
Backup
SessionID*

Structure: Generated as encryption of the hash digest of a server secret, FSK, and optional prefix (e.g., Prefix3). Encryption happens with a long-term, private key held by the server.

\[
\text{SessionID} = \text{Enc}(k, H(secret || FSK || (Prefix3 | \bot)))
\]

Usage: Append to service prefix (in the name) to indicate what key is used for encrypting payload data

*** This is only one way to create the SessionID
Resumption Cookie (RC)*

**Structure:** Encryption of $H$\text{(server secret)}, SS, FSK, and the (Prefix3, MoveToken) tuple (if provided), with a producer secret key that is also known to the service operating under Prefix3 (if provided)

$$ RC = \text{Enc}(k, SS||FSK|| ((\text{Prefix3}||\text{MoveToken})| \perp)) $$

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

$$(SS||FSK|| ((\text{Prefix3}||\text{MoveToken})| \perp)) = \text{Dec}(RC)$$

$$ \text{SessionID} = \text{?Enc}(k, H(secret||FSK|| (\text{Prefix3}| \perp))) $$

*** This is only one way to create the RC
Session Resumption

• Approach 0: If client has nothing, start with HELLO [2 RTT delay]

• Approach 1: If the client already has the config, start at the second step [1 RTT delay]

• Approach 2: If the client already has the SessionID and the ResumptionCookie, provide both to resume sessions after long periods of inactivity (requires producer state) [0 RTT delay]
Session Resumption (cont’d)

Note: SessionID is used to verify ownership of the SessionID, the MSK-C encryption key, and the real RC.