CCNx Key Exchange Protocol Version 1.0
draft-wood-icnrg-ccnxkeyexchange-02

Abstract

   This document specifies Version 1.0 of the CCNx Key Exchange (CCNxKE)
   protocol. The CCNxKE protocol allows two peers to establish a
   shared, forward-secure key for secure and confidential communication.
   The protocol is designed to prevent eavesdropping, tampering, and
   message forgery between two peers. It is also designed to minimize
   the number of rounds required to establish a shared key. In the
   worst case, it requires two RTTs between a consumer and producer to
   establish a shared key. In the best case, one RTT is required before
   sending any application data. This document outlines how to derive
   the keys used to encrypt traffic for a session and shows how session
   information is exchanged between a consumer and producer using
   message encapsulation.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   DISCLAIMER: This is a WIP draft of CCNxKE and has not yet seen
   rigorous security analysis.

   Ephemeral sessions similar to those enabled by TLS 1.3 [TLS13], QUIC
   [QUIC], and DTLS 1.2 [DTLS12] are needed for some CCN exchanges
   between consumers and producers. Currently, there does not exist a
   standard way to establish these sessions. Thus, the primary goal of
   the CCNxKE protocol is to provide privacy and data integrity between
   two CCN-enabled peers (e.g., a consumer and producer engaged in
   session-based communication). It is built on the CCNx 1.0 protocol
   and only relies upon standard Interest and Content Objects as a
   vehicle for communication. The CCNxKE protocol is used to bootstrap
   session-based communication, wherein traffic is encapsulated and
   encrypted using symmetric-key cryptography for transmission between
   two endpoints (i.e., a consumer and producer). The CCNxKE protocol
   enables this form of communication by establishing shared state,
   i.e., shared, ephemeral, and forward-secure symmetric keys. This
   protocol has the following four main properties:

   - Each peer's identity can be authenticated using asymmetric, or
      public key, cryptography (e.g., RSA [RSA], ECDSA [ECDSA], etc.).
      Server authentication is mandatory whereas mutual authentication
      is optional.

   - The negotiation of a forward-secure shared secret is protected
      from eavesdroppers and man-in-the-middle (MITM) attacks.

   - The negotiation is reliable: no attacker can modify the
      negotiation communication without being detected by the parties to
      the communication.

   - The state of a CCNxKE session can be securely migrated between an
      endpoint performing authentication and that which provides content
      using a "move token." This allows authentication and
      authorization to be separated from encryption for a session,
      enabling different systems to complete these steps.

Usage of CCNxKE is entirely independent of upper-layer application
protocols. Session-based communication via encapsulation and
encryption enables secure, confidential, and authenticated
communication between two peers. One advantage of this protocol is
that it facilitates the creation and use of ephemeral CCN Interest
and Content Objects.

CCNxKE also introduces a new type of cookie based on reverse hash
chains [HASHCHAIN] to help limit the amount of significant server
work done in response to a client or consumer Interest. TCP-based
protocols, such as TLS [TLS13], use the TCP 3-way handshake for such
proof. UDP-based protocols, such as QUIC [QUIC] and DTLS 1.2
[DTLS12], use an optional session address token or cookie that must
be presented by the client (consumer) to prove ownership of an
address during a key exchange procedure. Without source addresses,
our cookie technique ensures that the same entity which requested
server information, e.g., the public configuration data, is the same
entity that wishes to complete a key exchange.

The main contribution of this work is adapting key exchange
principles to the pull-based CCNx communication model. CCNxKE only
assumes that a consumer knows a first name prefix to initiate the key
exchange. The first Interest does not need to be a CCNxKE packet --
the producer can signal back to the consumer that it requires CCNxKE
before progressing.

This specification does not subsume other ICN-compliant key exchange
protocols. Nor does its existence imply that all encryption in an
ICN must be based on sessions. It was designed specifically to solve
the problem of session-based encryption in ICN.

1.1. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and
"OPTIONAL" in this document are to be interpreted as described in RFC
2119 [RFC2119].

The following terms are used:

Consumer/Client: The CCN consumer initiating the CCNxKE key exchange
via a first Interest.

Producer/Server: The CCN producer receiving or accepting the CCNxKE
key exchange request request Interest.

Sender: An endpoint that originates a message.

Receiver: An endpoint that is receiving messages.

Peer: An endpoint. When discussing a particular endpoint, "peer" refers to the endpoint that is remote to the primary subject of discussion.

Connection: A network path of n >= 1 hops between the consumer and producer.

Endpoint: Either the consumer or producer of the connection.

Handshake: A series of message exchanges between two peers that is used to perform a task (e.g., perform key exchange and derivation).

Session: An association between a consumer and a producer resulting from a CCNxKE handshake.

DH: A Diffie Hellman key exchange procedure [RFC2631] [DH].

Key Share: One half of the shared-secret provided by one peer performing a DH key exchange.

Forward-secure: The property that compromising any long-term secrets (e.g., cryptographic keys) does not compromise any session keys derived from those long-term secrets.

CONFIG information: A data structure created by a producer which contains long-term cryptographic material and associated information needed by a client to initiate a key-exchange with the producer.

HELLO exchange: An exchange between a consumer and producer wherein the consumer retrieves the CONFIG information from the producer.

Payload: The payload section of a CCNxMessage as defined in [CCNxMessages].

KEPayload: A payload for information used in the CCNxKE protocol which is a generic key-value store. The KEPayload is _not_ the CCNxMessage payload.

CCNxName: A CCNxName as defined in [CCNxMessages].

Semi-static: Short-term.

Short-term Secret (SS): A secret which is derived from the server's semi-static DH share and the client's fresh DH share.

Forward-secure Secret (FSK): A secret which is derived from fresh
(i.e., generated on demand at random) DH shares from both the
consumer and producer for the given connection.

HKDF: Hash-based key-derivation function [RFC5869].

2. Goals

The goals of the CCNxKE protocol, in order of priority, are as
follows:

1. Cryptographic security: CCNxKE should be used to securely
   establish a session and all related shared secrets between two
   peers. Cryptographic properties of interest include: (a)
   forward-secure session key derivation and (b) (state and
   computational) denial-of-service prevention at the producer (see
   [RFC4987]) that is no worse than DTLS 1.2 [DTLS12]}. For property
   (a), different keys (and relevant algorithm parameters such as
   IVs) are established for each communication direction, i.e., from
   consumer to producer and producer to consumer. For property (b),
   we use a new type of stateless cookie inspired by that of DTLS
   1.2.

2. Interoperability: Independent programmers should be able to
   develop applications utilizing CCNxKE that can successfully
   exchange cryptographic parameters without knowledge of one
   another's code.

3. Extensibility: CCNxKE seeks to provide a framework into which new
   public key and symmetric key methods and algorithms can be
   incorporated without breaking backwards compatibility or
   requiring all clients to implement new functionality. Moreover,
   the protocol should be able to support a variety of peer
   authentication protocols, e.g., EAP-TLS, EAP-PWD, or a simple
   challenge-response protocol.

4. Relative efficiency: CCNxKE tries to create sessions with minimal
   computation, bandwidth, and message complexity. In particular,
   it seeks to create sessions with as few end-to-end round trips as
   possible, and also provide support for accelerated session
   establishment and resumption when appropriate. At most 2 round-
   trip-times (RTTs) should be used to establish a session key, with
   the possibility of 1-RTT accelerated starts and resumption.

3. Scope

This document and the CCNxKE protocol are influenced by the TLS 1.3 [TLS13], QUIC [QUIC], and DTLS 1.2 [DTLS12] protocols. The reader, however, does not need a detailed understanding of those protocols to understand this document. Moreover, where appropriate, references to related protocols are made for brevity and technical clarity. This document is intended primarily for readers who will be implementing the protocol and for those doing cryptographic analysis of it. The specification has been written with this in mind and it is intended to reflect the needs of those two groups.

This document is not intended to supply any details of service definition or of interface definition, although it does cover select areas of policy as they are required for the maintenance of solid security.

4. Presentation Language

This document uses a presentation language of remote calls (i.e. packet messages) similar to the format used by TLS [TLS13].

5. CCNxKE Overview

5.1. Connection Establishment Latency

CCNxKE operates in three rounds, where each round requires a single RTT to complete. The full execution of the protocol therefore requires 2 RTTs before a session is fully established. The full version is used when consumers have no a priori information about the producer. An accelerated one round version is used when the consumer has valid configuration information and a source cookie from the producer; this variant requires 1 RTT before a session is established.

5.2. Connection Migration and Resumption

CCN end hosts lack the notion of addresses. Thus, the producer endpoint for a given execution of the CCNxKE protocol is one which can authoritatively serve as the owner of a particular namespace. For example, a consumer may wish to establish a session with a producer who owns the /company/foo namespace. The specific end host which partakes in the protocol instance is not specified, by virtue of the fact that all CCNxKE messages are based on well-defined names. This enables the producer end-host which partakes in the protocol to change based on the name of the CCNxKE messages. Consequently, to maintain correctness, it is important that a single execution of the protocol operates within the same trusted context; this does not mean

that the same producer end-host is required to participate in all
three steps of the protocol. Rather, it means that the end-host
responding to a CCNxKE message must be trusted by the consumer to
complete the exchange. CCNxKE is designed to enable this sort of
producer migration.

For example, a consumer may use an initial name like '/parc/
index.html' that works like an IP any cast address and could got to
one of several systems. CCNxKE allows the responding endpoint to
include a localized name to ensure that subsequent messages from the
consumer come back to the same producer. CCNxKE also allows the key
exchange peer to securely hand-off the session to a content producer
peer via another name and session token once the client is
authenticated and keying material is exchanged.

5.3. Re-Transmissions, Timeouts, and Replay Prevention

CCNxKE timeouts and retransmissions are handled using the approach in
[RFC6347]. One primary difference is that timer values may need to
be adjusted (elongated) due to prefix shifts and the need for a
producer to transfer security information between different machines.

Replay attack prevention is also an optional feature, and if used,
MAY be done using one of the following two approaches at the receiver
(producer):

- IPSec AH [RFC4302] and ESP [RFC4303] style replay detection based
   on sliding windows and monotonically increasing sequence numbers
   for windows. Note that the sliding window inherently limits the
   performance of the protocol to the window size, since only a
   finite number of messages may be received within a given window
   (based on the window size).

- The optimized anti-replay algorithm of [RFC6479].

5.4. Loss Sensitivity

CCNxKE messages are transferred using standard CCN Interest and
Content Objects and are therefore subject to loss as any datagram.
This means that traffic encrypted with keys derived from CCNxKE must
be stateless. They cannot depend on in-order arrival. This problem
is solved by two mechanisms: (1) by prohibiting stream ciphers of any
kind and (2) adding sequence numbers to each message that allow the
receiver to identify and use the correct cryptographic state to
decrypt the message. Moreover, sequence numbers permit anti-replay
mechanisms similar to those used in DTLS [DTLS12] as mentioned above.

[6](#). The CCNxKE Protocol

This section describes the CCNxKE protocol in detail at the message
level. The specific encoding of those messages is given later.
CCNxKE could be adapted to different wire format encodings, such as
those used by the NDN protocol.

The following assumptions are made about peers participating in the
CCNxKE protocol:

- Consumers know the namespace prefix of the producer for which they
  wish to execute the CCNxKE protocol.

- CCNxKE protocol information is carried in a distinguished field
  outside of the payload of CCN messages. This is done to
  distinguish key exchange material with application data in a
  message. This is necessary for 0 RTT packets that carry both
  keying material and application payload.

- CCNxKE does not require any special behavior of intermediate
  systems to forward packets.

- CCNxKE packets generally should not be cached for significant
  periods of time, as use normal protocol methods to limit caching.
  Part of this is achieved through the use of consumer-specific
  nonces in names.

[6.1](#). Round Overview

CCNxKE is composed of three rounds. The purpose of each round is
described below.

- Round 1: Perform a bare HELLO exchange to obtain the extensions
  (parameters) for the key exchange provided by the producer and a
  source cookie to prove ownership of the "source" of the request.

- Round 2: Perform the initial FULL-HELLO exchange to establish a
  forward-secure key used for future communication, i.e., Interest
  and Content Object exchanges in the context of the newly
  established session.

- Round 3: Send the first bit of application data and (optionally)
  transfer resumption cookie(s) from the producer to the consumer.

Conceptually, there are two secrets established during a single
execution of CCNxKE:

   - Static Secret (SS): A secret which is derived in one of two ways:
      (a) from the client and server ephemeral key shares and (b) from
      the server's semi-static share and the client's ephemeral key
      share. Keying material derived from SS in option (a) is not
      forward secure.

   - Ephemeral Secret (ES): A secret which is derived from both the
      client and server ephemeral key shares.

   Depending on the mode in which CCNxKE is used, these secrets can be
   established in a variety of ways. Key derivation details are
   outlined in Section Section 9.

   All secrets are derived with the appropriate amount of randomness
   [RFC4086]. An overview of the messages sent in each of the three
   rounds to establish and use these secrets is shown in Figure Figure 1
   below. This diagram omits some parts of each message for brevity.

```
     Consumer                                         Producer

     HELLO:
     + SourceChallenge
                    I[/prefix/random-1]
                        -------->
                                               HELLO-REJECT:
                                                   + Timestamp
                                                + SourceCookie
                                               + pinned-prefix*
                                              + ServerChallenge*
                                           + ServerConfiguration*

                    CO[/prefix/random-1]
                         <---------
     FULL-HELLO:
     + ClientKeyShare
     + SourceCookie
     + SourceProof
     + Timestamp
                    I[/pinned-prefix/random-2]
                        -------->
                                              HELLO-ACCEPT:
                                             + ServerKeyShare
                                                   + SessionID
                                          + [CertificateRequest*]
                                            + [CertificateVerify*]
                                       + [MovePrefix*, MoveToken)*]
                                                    + [Finished]
                    CO[/pinned-prefix/random-2]
```

```
                                 <--------
                       **key exchange complete**
          Payload:
          + MoveToken*
          + MoveProof*
          + [ConsumerData]

                         I[/prefix/SessionID/[...]]
                               -------->
                                                    + NewSessionID*
                                                 + NewSessionIDTag*
                                                          Payload:
                                                     [ProducerData]
                         CO[/prefix/SessionID/[...]]
                               <--------


          Repeat with data        <-------->        Repeat with data


                *   Indicates optional or situation-dependent
                    messages that are not always sent.

                {} Indicates messages protected using keys
                    derived from the short-term secret (SS).

                () Indicates messages protected using keys
                    derived from the ephemeral secret (ES).

                [] Indicates messages protected using keys
                    derived from the traffic secret (TS).
```

        Figure 1: High-level message flow for full CCNxKE protocol with a
                             maximum 2-RTT delay.

   In the following sections, we will describe the format of each round
   in this protocol in more detail.

   We do not specify the encoding of CCNxKE data sent in Interest and
   Content Object payloads. Any viable encoding will suffice, so long
   as both parties agree upon the type. For example, the payload could
   be structured and encoded as a JSON object, e.g.,

   { "ClientKeyShare" : 0xaa, "SourceCookie" : 0xbb, "SourceProof" :
   0xbb, ... }

   For now, we assume some valid encoding mechanism is used to give
   structure to message payloads. Moreover, we assume that these
   payloads are carried in a distinguished CCNxKE payload field
   contained in the Interest and Content Objects.

6.2. Round 1

The purpose of Round 1 is to acquire a cookie to binding the exchange
to the initial consumer and the public configuration information
contained in the ServerConfiguration structure. This information is
used in the second round when performing the actual key exchange. To
that end, the format of the Round 1 message is trivial. First, the
client issues an Interest with the following name

    /prefix/random-1

where random-1 is a randomly generated 64-bit nonce. This interest
carries a KEPayload with the following information:

HELLO Field	Description	Optional?
SourceChallenge	A random value generated to prove	No
	ownership of the consumer's	
	"source"	

Upon receipt of this interest, the producer responds with a HELLO-
REJECT Content Object whose KEPayload has the following fields:

HELLO-REJECT Field	Description	Optional?
Timestamp	Current server timestamp	No
SourceCookie	A cookie that binds the	No
	consumer's challenge to the	
	current timestamp	
PinnedPrefix	A new prefix that pins the key	Yes
	exchange to a particular server	
ServerConfiguration	The public server configuration	Yes
	information	
ServerChallenge	A random value for the consumer	Yes
	to include in its	
	CertificateVerify if the server	
	requires client authentication	

The Timestamp and SourceCookie are used in Round 2. Their derivation
is described later. If the server provides a PinnedPrefix then the

consumer must use this prefix in Round 2 in lieu of the Round 1 name
prefix. (This is because the PinnedPrefix identifies a particular
endpoint that is capable of completing the key exchange.)

The ServerConfiguration information is a semi-static catalog of
information that consumers may use to complete future key exchanges
with the producer. The fields of the ServerConfiguration information
are shown below.

ServerConfiguration Field	Description	Optional?
KEXS	Supported elliptic-curve key-exchange algorithms	No
AEAD	Supported AEAD algorithms	No
PUBS	List of public values (for key exchange algorithm) encoded appropriately for the given group	No
EXPRY	Expiration timestamp (i.e., longevity of the ServerConfiguration structure)	No
VER	Version of the CONFIG structure	Yes
CERT	Server certificate	No
SIG	Signature produced by the server over the entire ServerConfiguration message	No

The KEXS is a data structure that enumerates the elliptic curve key-
exchange algorithms that are supported by the producer (see [QUIC]
for more details). Currently, only the following curves are
supported:

- Curve25519

- P-256

Selection criteria for these curves is given at
http://safecurves.cr.yp.to/.

The AEAD structure enumerates the supported AEAD algorithms used for
symmetric-key authenticated encryption after the session has been
established. Currently, the only supported algorithms are:

- AES-GCM-(128,192,256) [GCM]: a 12-byte tag is used, where the
  first four bytes are taken from the FSK key-derivation step and
  the last eight are taken from the initial consumer nonce.

- ChaCha20+Poly1305 [RFC7539].

The key sizes and related parameters are provided with the AEAD tag
in the CONFIG structure.

The PUBS structure contains the public values for the initial key
exchange. Both Curve25519 and P-256 provide their own set of
accepted parameters. Thus, the only values provided here are the
random curve elements used in the DH operation.

The EXPRY value is an absolute timestamp that indicates the longevity
of the ServerConfiguration.

The CERT and SIG values contain the server's certificate and a
signature generated over the entire ServerConfiguration field. This
signature is generated with the corresponding private key.

6.3. Round 2

The purpose of Round 2 is to perform the initial FULL-HELLO exchange
to establish a forward-secure key used for future communication. It
is assumed that the consumer already has the ServerConfiguration
information that is provided from the producer in Round 1. It is
also assumed that the consumer has a

Moreover, assume that nonce2 is a ephemeral nonce provided by the
producer in Round 1. Then, the consumer issues an Interest with the
following name:

    /prefix/random-2

and a KEPayload with the following information:

FULL-HELLO Field	Description	Optional?
ClientKeyShare	The client's key share for the key exchange	No
SourceCookie	SourceCookie provided by the	No

	server in Round 1	
SourceProof	The SourceCookie construction proof provided by the client	No
Timestamp	The timestamp provided by the server in Round 1	No
ConsumerPrefix	The consumer's prefix that can be used for the producer to send interests to the consumer	Yes
PreSharedKey	A pre-shared key that can be configured between a consumer and producer	Yes
ResumptionCookie	The ResumptionCookie derived from a past session	Yes
{MoveChallenge}	A move challenge generated identically to the SourceChallenge	Yes
{AlgChoice}	Algorithm (KEXS and AEAD) options choice (a list of tags echoed from the ServerConfiguration)	No
{Proof}	Proof of demand (i.e., a sorted list of types of proof the consumer will expect)	No
{CCS}	Compressed certificate set that the consumer possesses	No
{ConsumerData}	Application data encrypted under a key derived from SS (in a 1-RTT exchange)	Yes
ServerNameIndication	A server name indication (as a CCNxName) defined in Section 3 of [RFC6066]	Yes
Certificate	The client's certificate	Yes
CertificateVerify	A signature generated over the entire FULL-HELLO message	Yes

((TODO: provide more details about each of these fields))

Upon receipt of this interest, the producer performs the DH
computation to compute ES and SS, decrypts all protected fields in
the consumer's KEPayload, and validates the algorithm choice
selection (AlgChoice). If any of these steps fail, the producer
replies with with a HELLO-REJECT Content Object whose KEPayload
contains a REJ flag and the reason of the error. The REJ flag and
value are encrypted by the SS (if possible).

If the above steps complete without failure or error, then the
producer responds with a Content Object whose KEPayload has the
following fields:

HELLO-ACCEPT Field	Description	Optional?
SessionID	Cleartext session identifier	No
ServerKeyShare	Server's key share for the ES derivation	No
{ServerExtensions}	Additional extensions provided by the server, encrypted under ES	Yes
[ResumptionCookie]	Resumption cookie encrypted under a TS-derived key	Yes
{(MovePrefix,MoveToken)}	Third CCNxName prefix and token to use when moving to session establishment	Yes
CertificateRequest*	Server certificate that matches the type of proof provided by the client	Yes
CertificateVerify*	Signature generated over the entire HELLO-ACCEPT message	Yes

If a MovePrefix and MoveToken tuple is provided then in the HELLO-
ACCEPT message then a CertificateVerify (signature) MUST also be
provided in the response.

6.4. Round 3

   In Round 3, the consumer sends interests whose name and optional
   Payload are encrypted using one of the forward-secure keys derived
   after Round 2. In normal operation, the producer will respond with
   Content Objects whose Payloads are encrypted using a different
   forward-secure key. That is, interests and Content Objects are
   encrypted and authenticated using two separate keys. The producer
   may also optionally provide a new resumption cookie (RC) with a
   Content Object response. This is used to keep the consumer's
   resumption cookie fresh and to also support 0 RTT resumption. In
   this case, the producer's Content Object response has the following
   fields:

   - Payload: the actual Content Object payload data encrypted with the
      producer's forward-secure key.

   - ResumptionCookie: A new resumption cookie to be used for resuming
      this session in the future.

   The producer is free to choose the frequency at which new resumption
   cookies are issued to the consumer.

   The producer may also reply with a new SessionID. This is done if
   the client presented a MoveToken and MoveProof. A NewSessionID must
   be accompanied with a NewSessionIDTag, which is equal to the HMAC of
   NewSessionID computed with the traffic-secret key. A client MUST
   then use NewSessionID instead of SessionID after verifying the
   NewSessionIDTag.

7. Alternative Exchanges

   CCNxKE also supports one-round key exchange and session resumption.
   These variants are outlined below. The key material differences are
   described later. In these variants, we use message
   ExchangeSourceCookie to denote the following exchange:

```
   Consumer                                          Producer

   HELLO:
   + SourceChallenge
                    I[/prefix/random-1]
                        -------->
                                           HELLO-REJECT:
                                               + Timestamp
                                             + SourceCookie
                                           ServerChallenge*
                                         ServerConfiguration*


                    CO[/prefix/random-1]
                        <---------
```

        Figure 2: SourceCookie exchange -- ExchangeSourceCookie.

7.1. One-RTT Exchange

```
       Consumer                                        Producer
                              -------->
                        ExchangeSourceCookie
                              <---------
       FULL-HELLO:
       + ClientKeyShare
       + SourceCookie
       + SourceProof
       + Timestamp
       + Certificate*
       + CertificateVerify*
       + {ConsumerData*}
                        I[/prefix/random-2]
                              -------->
                                                     HELLO-ACCEPT:
                                                  + ServerKeyShare
                                                       + SessionID
                                               + [ServerExtensions]
                                               + [ResumptionCookie]
                                              + [CertificateRequest*]
                                               + [CertificateVerify*]
                                           + [MovePrefix*, MoveToken*]
                                                       + [Finished]
                        CO[/prefix/random-2]
                              <--------
                        **key exchange complete**
       Send encrypted data   <-------->   Send encrypted data


            *  Indicates optional or situation-dependent
               messages that are not always sent.

            {} Indicates messages protected using keys
               derived from the short-term secret (SS).

            () Indicates messages protected using keys
               derived from the ephemeral secret (ES).

            [] Indicates messages protected using keys
               derived from the traffic secret (TS).


                    Figure 3: Exchange with 1 RTT.
```

   As with TLS, the initial application data is protected with the

8. **Resumption and PSK Mode**

   In this mode, the client uses its ResumptionCookie to re-create a
   previous session. The client also provides a key share in case the
   server opts to fall back and establish a fresh key. If the server
   accepts the ResumptionCookie then it MUST issue a new SessionID and
   ResumptionCookie for future use with the client.

```
        Consumer                                         Producer
                            -------->
                       ExchangeSourceCookie
                            <---------
        FULL-HELLO:
        + ClientKeyShare
        + SourceCookie
        + SourceProof
        + Timestamp
        + PreSharedKey
        + ResumptionCookie
                       I[/prefix/random-2]
                            -------->
                                                    HELLO-ACCEPT:
                                                  + ServerKeyShare
                                                       + SessionID
                                              + [ServerExtensions]
                                              + [ResumptionCookie]
                                         + [MovePrefix*, MoveToken*]
                                                       + [Finished]
                       CO[/prefix/random-2]
                            <--------
                       **key exchange complete**
        Send encrypted data   <-------->   Send encrypted data

                *  Indicates optional or situation-dependent
                   messages that are not always sent.

                {} Indicates messages protected using keys
                   derived from the short-term secret (SS).

                () Indicates messages protected using keys
                   derived from the ephemeral secret (ES).

                [] Indicates messages protected using keys
                   derived from the traffic secret (TS).

                     Figure 4: Exchange with 1 RTT.
```
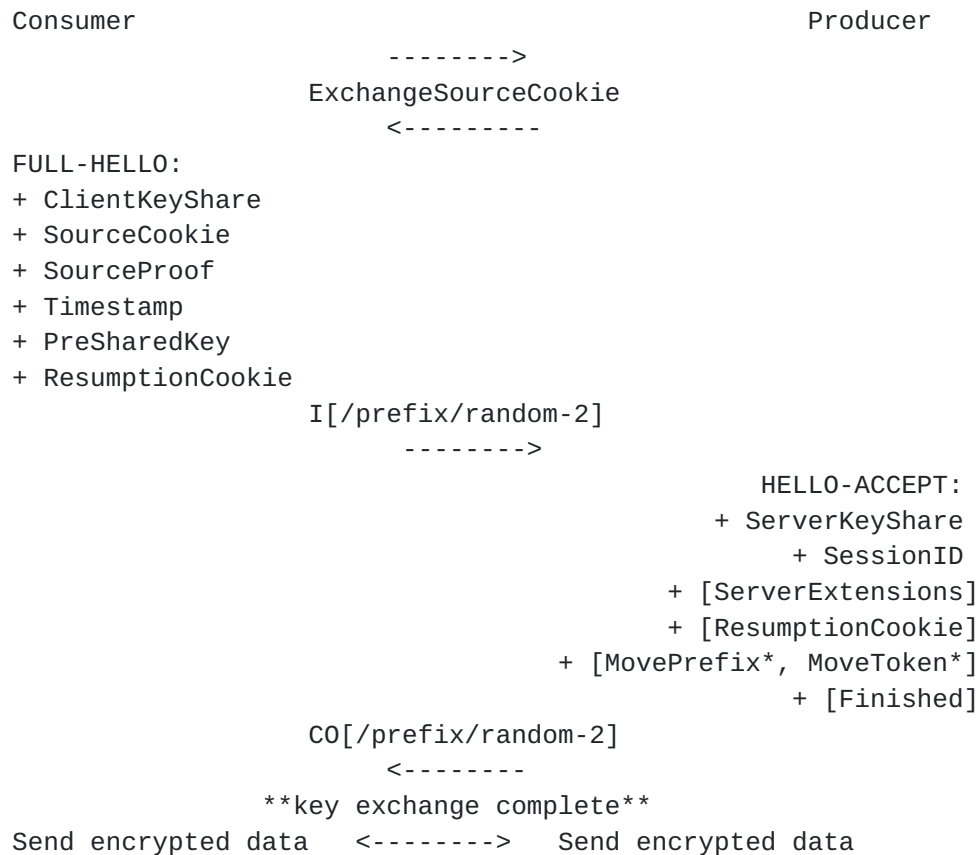
9. Secret Derivation

   In this section we describe how secrets used in the protocol are
   derived. We cover the SourceCookie, MoveToken, SessionID,
   ResumptionCookie, and the actual traffic keys.

9.1. SourceCookie Derivation

   The intention of the SourceCookie is to prove that a client is
   sending interests from a legitimate location before any server
   computation is done. Without this, a Denial of Service attack could
   be carried out by sending interests to the server with the intention
   of triggering wasted computation. TCP-based protocols prevent this
   with the SYN-flood cookie mechanism. Protocols based on UDP use
   cookies that bind to the client address [DTLS12]. Since CCN lacks
   any notion of a source address, these cookie mechanisms do not apply.
   Instead, we need a way for clients to prove that they initiated a key
   exchange from the "same origin." We now describe the cookie
   mechanism that gives us this guarantee.

   Instead of a source address, a SourceCookie is computed using a
   challenge provided by a consumer. To create this challenge, a
   consumer first generates a a randomly generated 256-bit string X.
   The consumer then computes SourceChallenge = SHA256(X). Upon receipt
   of this challenge, the producer generates a SourceCookie as follows:

   SourceCookie = HMAC(k, SourceChallenge || timestamp)

   where timestamp is the current server timestamp and k is the server's
   secret key. To prove ownership of the "source," the consumer then
   provides the SourceCookie and a SourceProof in the round 2 Interest.
   The SourceProof is set to the value X used to derive the
   SourceChallenge. Upon receipt of the SourceProof, the server
   verifies the following equality:

   SourceCookie = HMAC(k, SHA256(SourceProof) || timestamp)

   If this check passes, then the server continues with the
   computationally expensive part of the key exchange protocol.

   To avoid replays of the SourceProof and SourceCookie, a producer
   SHOULD keep a sliding window of previously received tuples.

9.2. Move Derivation

   The MoveChallenge and MoveProof are computed identically to the
   SourceChallenge and SourceProof. The MoveToken, however, is left as

   an opaque bit string. Extensions may be specified to describe how to
   compute this value.

9.3. SessionID and ResumptionCookie Properties, Derivation, and Usage

   The purpose of the session identifier SessionID is to uniquely
   identify a single session for the producer and consumer. A Producer
   MAY use a random bit string or MAY use the method described in this
   section or MAY use another proprietary method to distinguish clients.

   We provide a more secure creation of the SessionID since it is used
   with the ResumptionCookie derivation (defined later). Specifically,
   the SessionID is derived as the encryption of the hash digest of a
   server secret, TS, and an optional prefix (e.g., MovePrefix).

   Encryption is done by the using a long-term secret key owned by the
   server used for only this purpose, i.e., it is not used for consumer
   traffic encryption. Mechanically, this derivation is:

   SessionID = Enc(k1, H(TS || (Prefix3))),

   where k1 is the long-term producer key.

   For the resumption cookie, we require that it must be able to be used
   to recover the TS for a given session. Without TS, correct session
   communication is not possible. We derive it as the encryption of the
   hash digest of the server secret, TS, and the optional (MovePrefix,
   MoveToken) tuple (if created for the session). The producer must use
   a long-term secret key for this encryption. Mechanically, this
   derivation is:

   ResumptionCookie = Enc(k2, TS || ((Prefix3 || MoveToken))),

   where k2 is again a long-term producer key. Note that it may be the
   case that k1 = k2 (see above), though this is not required.

   With this SessionID and ResumptionCookie, the consumer then resumes a
   session by providing both the SessionID and ResumptionCookie to the
   producer. This is done to prove to the producer that the consumer
   who knows the SessionID is also in possession of the correct
   ResumptionCookie. The producer verifies this by computing

   (TS || ((Prefix3 || MoveToken))) = Dec(k2, ResumptionCookie)

   and checking the following equality

   SessionID = Enc(k1, H(TS || (Prefix3)))

   If equality holds, the producer uses the TS recovered from
   ResumptionCookie to re-initialize the previous session with the
   consumer.

9.4. Key Derivation

   CCNxKE adopts the key schedule and derivation techniques defined in
   TLS 1.3 [TLS13]. Specifically, it uses the SS and ES to establish a
   common master secret (MS) and, from that, the traffic secret (TS).
   These dependencies are shown below.

```
   +------+              +------+
   | KE-1 |              | KE-2 |
   +------+              +----+-+
       |                      |
       |                      |
       |                      |
   +---v--+              +----v-+
   |  SS  +---+     +--+  ES  |
   +------+   |     |  +------+
              |     |
              |     |
          +-v----v-|
          |   MK    |
          +---+----+
               |
               |
               |
            +-v--+
            | TS |
            +----+
```
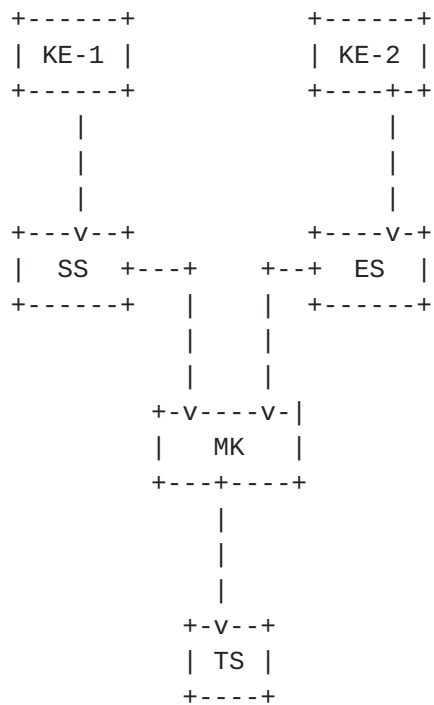
   In this figure, KE-1 and KE-2 are two "sources" of keying material.
   The following table shows what these two sources are in different key
   exchange scenarios.

```
+-------------+----------------------------+---------------------+
| Key         | KE-1                       | KE-2                |
| Exchange    |                            |                     |
+-------------+----------------------------+---------------------+
| Full        | ClientKeyShare and         | ClientKeyShare and  |
| handshake   | ServerKeyShare DH          | ServerKeyShare DH   |
|             |                            |                     |
| Handshake   | ClientKeyShare and         | ClientKeyShare and  |
| with 1-RTT  | ServerConfiguration public | ServerKeyShare DH   |
|             | share DH                   |                     |
|             |                            |                     |
| PSK         | Pre-shared key             | Pre-shared key      |
+-------------+----------------------------+---------------------+
```

Given the values for SS and ES, the remaining derivation steps are
below as defined in [TLS13]. They are repeated here for posterity.

1. xSS = HKDF-Extract(0, SS). Note that HKDF-Extract always
    produces a value the same length as the underlying hash function.

2. xES = HKDF-Extract(0, ES)

3. mSS = HKDF-Expand-Label(xSS, "expanded static secret",
    handshake_hash, L)

4. mES = HKDF-Expand-Label(xES, "expanded ephemeral secret",
    handshake_hash, L)

5. master_secret = HKDF-Extract(mSS, mES)

6. traffic_secret_0 = HKDF-Expand-Label(master_secret, "traffic
    secret", handshake_hash, L)

In all computations, the value "handshake_hash" is defined as the
SHA256 hash digest of all CCNxKE messages contained up to the point
of derivation. More details are given in Section 7.3.1 of [TLS13].

Updating the traffic secret using the re-key message (defined later)
increments traffic_secret_N to traffic_secret_(N+1). This update
procedure works as follows:

traffic_secret_N+1 = HKDF-Expand-Label(traffic_secret_N, "traffic
secret", "", L)

9.5. Secret Generation and Lifecycle

   The secrets (keys and IVs) used to encrypt and authenticate traffic
   are derived from the traffic secret. The explicit derivation
   formula, as is defined in [TLS13], is as follows:

   secret = HKDF-Expand-Label(Secret, phase + ", " + purpose,
   handshake_context, key_length)

   In this context, secret can be a key or IV. This formula is used
   when deriving keys based on a non-forward-secure SS and the forward-
   secure TS. The following table enumerates the values for "phase",
   and "handshake_context" to be used when defining keys for different
   purposes.

```
+-------------+--------+-----------------+------------------------+
| Record Type | Secret | Phase           | Handshake Context      |
+-------------+--------+-----------------+------------------------+
| 1-RTT       | xSS    | "early handshake | HELLO +               |
| Handshake   |        | key expansion"  | ServerConfiguration +  |
|             |        |                 | Server Certificate     |
|             |        |                 |                        |
| 1-RTT Data  | xSS    | "early          | HELLO +                |
|             |        | application data | ServerConfiguration + |
|             |        | key expansion"  | Server Certificate     |
|             |        |                 |                        |
| Application | TS     | "application    | HELLO ... Finished     |
| Data        |        | data key        |                        |
|             |        | expansion"      |                        |
+-------------+--------+-----------------+------------------------+
```

   Moreover, the following table indicates the values of "purpose" used
   in the generation of each secret.

```
+------------------+-------------------+
| Secret           | Purpose           |
+------------------+-------------------+
| Client Write Key | "client write key" |
|                  |                   |
| Server Write Key | "server write key" |
|                  |                   |
| Client Write IV  | "client write IV" |
|                  |                   |
| Server Write IV  | "server write IV" |
+------------------+-------------------+
```

   ((TODO: should we add examples for each of the above variants?))

10. Re-Key Message

Either the client and server can trigger a key update by sending an
Interest or Content Object with a KEPayload field containing the flag
KeyUpdate. The KEPayload will be encrypted by the traffic key. Upon
receipt, the recipient MUST update the traffic secret as defined
above and re-compute the traffic encryption and authentication keys.
The previous traffic key must be securely discarded.

11. Application Data Protocol

Once traffic keys and the associated IVs are derived from the CCNxKE
protocol, all subsequent Interest and Content Object messages are
encrypted. Packet encryption uses the TLV encapsulation mechanism
specified in [ESIC]. For Interest encryption, the Salt in [ESIC] is
set to the packet sequence number. The same substitution is done for
Content Object encryption. Similarly, the KeyId field is substituted
with the SessionID derived by the CCNxKE protocol. Packet sequence
numbers are 64-bit numbers initialized to 0 when after the traffic
secret is calculated. Each message increments and uses the sequence
number when sending a new datagram (Interest). The sequence number
for an Interest matches that of the Content Object response.

12. Security Considerations

For CCNxKE to be able to provide a secure connection, both the
consumer and producer systems, keys, and applications must be secure.
In addition, the implementation must be free of security errors.

The system is only as strong as the weakest key exchange and
authentication algorithm supported, and only trustworthy
cryptographic functions should be used. Short public keys and
anonymous servers should be used with great caution. Implementations
and users must be careful when deciding which certificates and
certificate authorities are acceptable; a dishonest certificate
authority can do tremendous damage.

13. References

13.1. Normative References

[CCNxMessages]
          Mosko, M., Solis, I., and C. Wood, "CCNx Messages in TLV
          Format", January 2016, <https://datatracker.ietf.org/doc/
          draft-irtf-icnrg-ccnxmessages/>.

   [DH]   Diffie, W. and M. Hellman, "New Directions in
               Cryptography", IEEE Transactions on Information Theory,
               V.IT-22 n.6 , June 1977.

   [DTLS12]   Rescorla, E. and N. Modadugu, "Datagram Transport Layer
               Security Version 1.2", January 2012,
               <https://tools.ietf.org/html/rfc6347>.

   [ECDSA]   American National Standards Institute, "Public Key
               Cryptography for the Financial Services Industry: The
               Elliptic Curve Digital Signature Algorithm (ECDSA)",
               ANSI ANS X9.62-2005, November 2005.

   [ESIC]   Mosko, M. and C. Wood, "Encrypted Sessions In CCNx
               (ESIC)", n.d., <https://datatracker.ietf.org/doc/draft-
               wood-icnrg-esic/>.

   [GCM]   Dworkin, M., "Recommendation for Block Cipher Modes of
               Operation: Galois/Counter Mode (GCM) and GMAC",
               NIST Special Publication 800-38D, November 2007.

   [QUIC]   Iyengar, J. and I. Swett, "QUIC: A UDP-Based Secure and
               Reliable Transport for HTTP/2", December 2015.

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
               Requirement Levels", BCP 14, RFC 2119,
               DOI 10.17487/RFC2119, March 1997,
               <http://www.rfc-editor.org/info/rfc2119>.

   [RFC2631]   Rescorla, E., "Diffie-Hellman Key Agreement Method",
               RFC 2631, DOI 10.17487/RFC2631, June 1999,
               <http://www.rfc-editor.org/info/rfc2631>.

   [RFC4086]   Eastlake 3rd, D., Schiller, J., and S. Crocker,
               "Randomness Requirements for Security", BCP 106, RFC 4086,
               DOI 10.17487/RFC4086, June 2005,
               <http://www.rfc-editor.org/info/rfc4086>.

   [RFC4302]   Kent, S., "IP Authentication Header", RFC 4302,
               DOI 10.17487/RFC4302, December 2005,
               <http://www.rfc-editor.org/info/rfc4302>.

   [RFC4303]   Kent, S., "IP Encapsulating Security Payload (ESP)",
               RFC 4303, DOI 10.17487/RFC4303, December 2005,
               <http://www.rfc-editor.org/info/rfc4303>.

   [RFC4987]   Eddy, W., "TCP SYN Flooding Attacks and Common
               Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007,
               <http://www.rfc-editor.org/info/rfc4987>.

   [RFC5869]   Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand
               Key Derivation Function (HKDF)", RFC 5869,
               DOI 10.17487/RFC5869, May 2010,
               <http://www.rfc-editor.org/info/rfc5869>.

   [RFC6066]   Eastlake 3rd, D., "Transport Layer Security (TLS)
               Extensions: Extension Definitions", RFC 6066,
               DOI 10.17487/RFC6066, January 2011,
               <http://www.rfc-editor.org/info/rfc6066>.

   [RFC6347]   Rescorla, E. and N. Modadugu, "Datagram Transport Layer
               Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347,
               January 2012, <http://www.rfc-editor.org/info/rfc6347>.

   [RFC6479]   Zhang, X. and T. Tsou, "IPsec Anti-Replay Algorithm
               without Bit Shifting", RFC 6479, DOI 10.17487/RFC6479,
               January 2012, <http://www.rfc-editor.org/info/rfc6479>.

   [RFC7539]   Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF
               Protocols", RFC 7539, DOI 10.17487/RFC7539, May 2015,
               <http://www.rfc-editor.org/info/rfc7539>.

   [RSA]   Rivest, R., Shamir, A., and L. Adleman, "A Method for
               Obtaining Digital Signatures and Public-Key
               Cryptosystems", Communications of the ACM v. 21, n. 2, pp.
               120-126., February 1978.

   [TLS13]   Rescorla, E., "The Transport Layer Security (TLS) Protocol
               Version 1.3", December 2015, <https://tools.ietf.org/html/
               draft-ietf-tls-tls13-11>.

13.2. Informative References

   [HASHCHAIN]
               L. Lamport, "Password Authentication with Insecure
               Communication", ANSI Communications of the ACM 24.11, pp
               770-772, November 1981.

   [RFC5077]   Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig,
               "Transport Layer Security (TLS) Session Resumption without
               Server-Side State", RFC 5077, DOI 10.17487/RFC5077,
               January 2008, <http://www.rfc-editor.org/info/rfc5077>.

Authors' Addresses

    M. Mosko
    PARC

    EMail: marc.mosko@parc.com

    Ersin Uzun
    PARC

    EMail: ersin.uzun@parc.com

    Christopher A. Wood
    University of California Irvine

    EMail: woodc1@uci.edu