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.

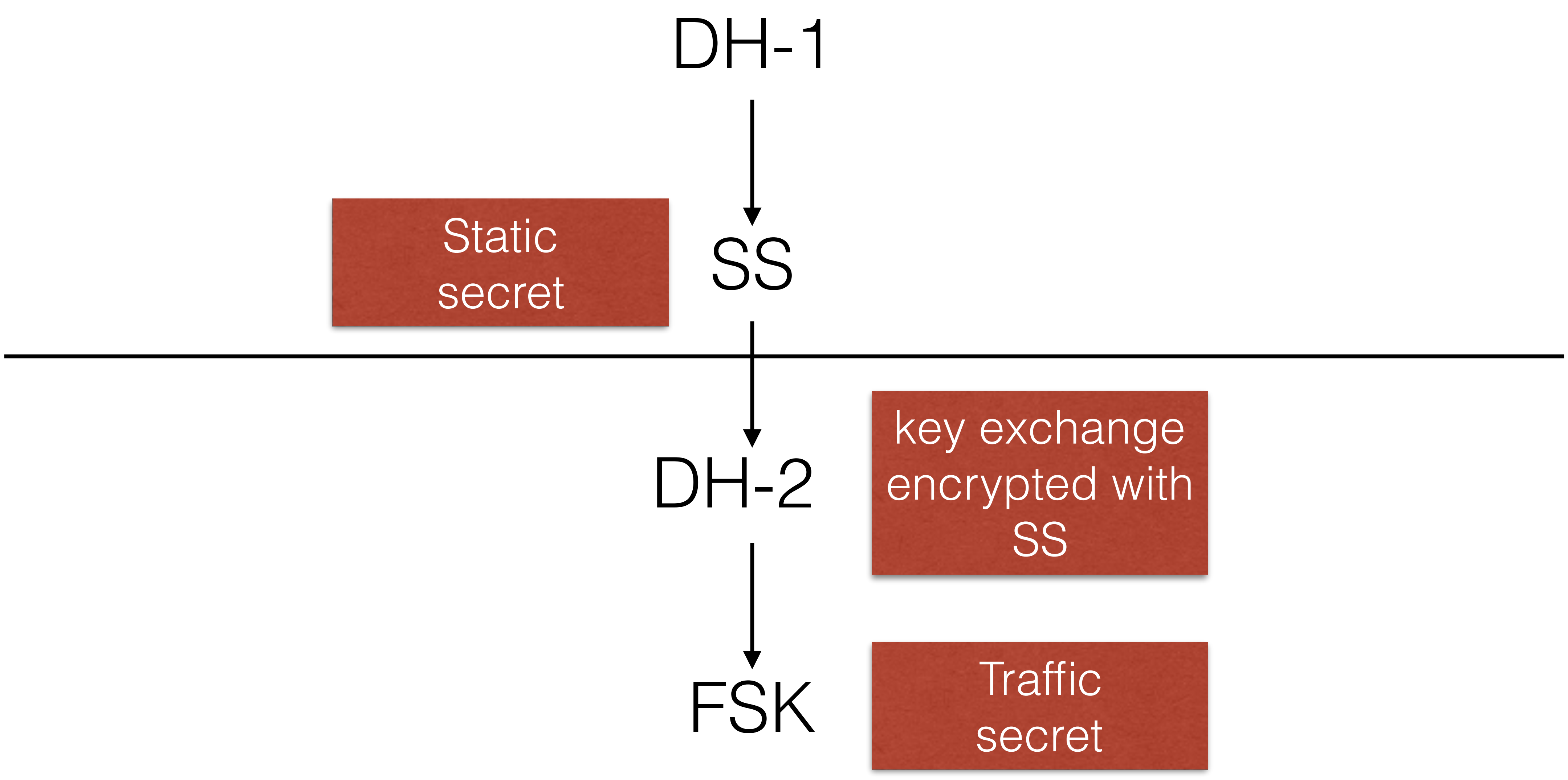
Goals

- 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.

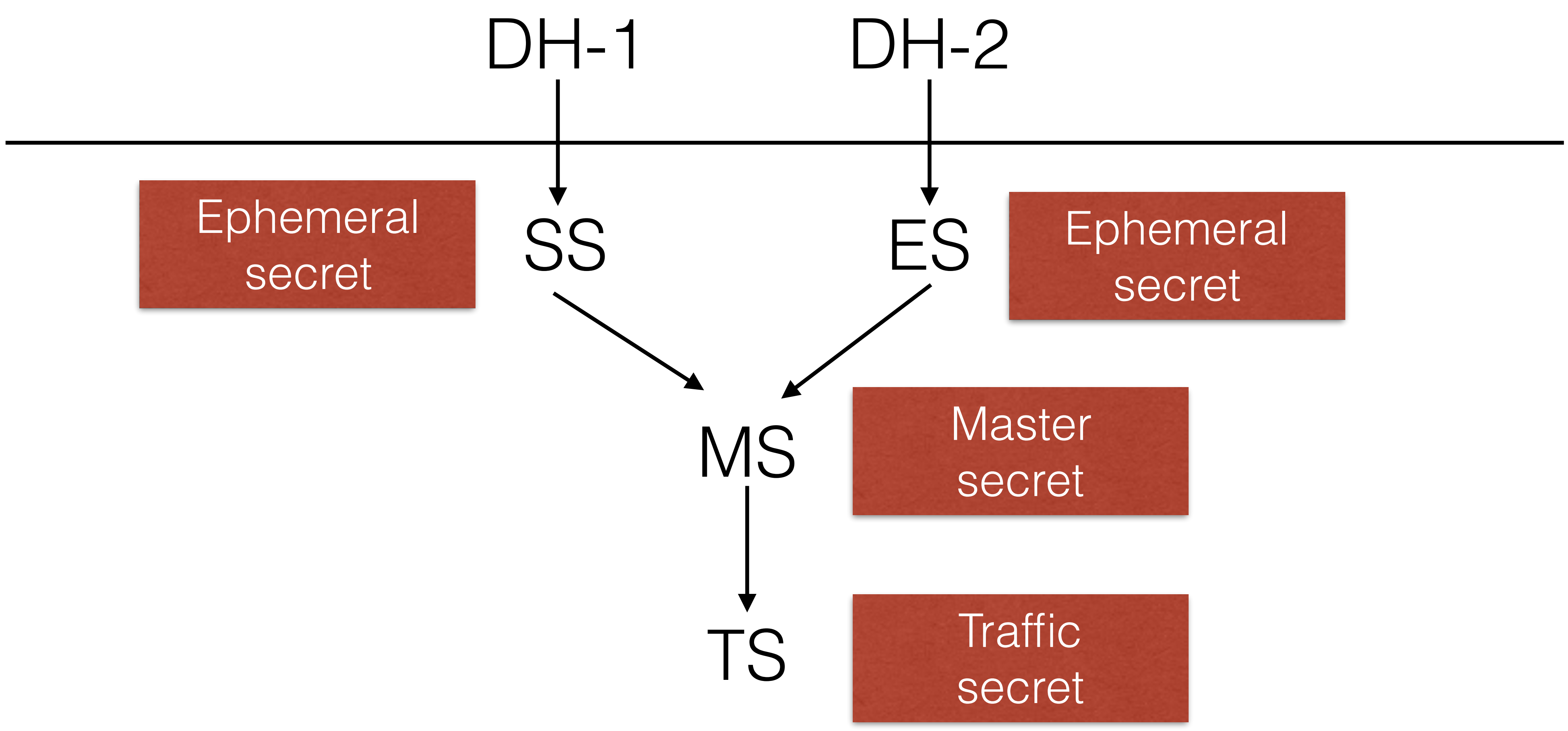
Updates

- 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



New Traffic Key Calculation



Keying Material Sources

	SS	ES
Normal exchange	(cleartext) client and server ephemeral shares	(cleartext) client and server ephemeral shares
Resumption (1-RTT)**	(cleartext) client ephemeral and server (CONFIG) static shares	(cleartext) client and server ephemeral shares
PSK	PSK	PSK

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]

[1] Langley, A., and W. T. Chang. "QUIC crypto." (2014).

CCNx-KE vs TLS 1.3

Feature	CCNx-KE	TLS 1.3
Record layer	Interests and content objects are encapsulated and assigned explicit sequence numbers	Streamed record layer
Session secret usage	Handoff to other parties	Pinned to the server
Transport mechanism	Unreliable datagrams	TCP
Resumption	Stateful resumption cookies	Opaque labels
DoS prevention	DTLS-like cookies	Defers to TCP

TLS: Encrypted Streams

TLS record layer:

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

```
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:

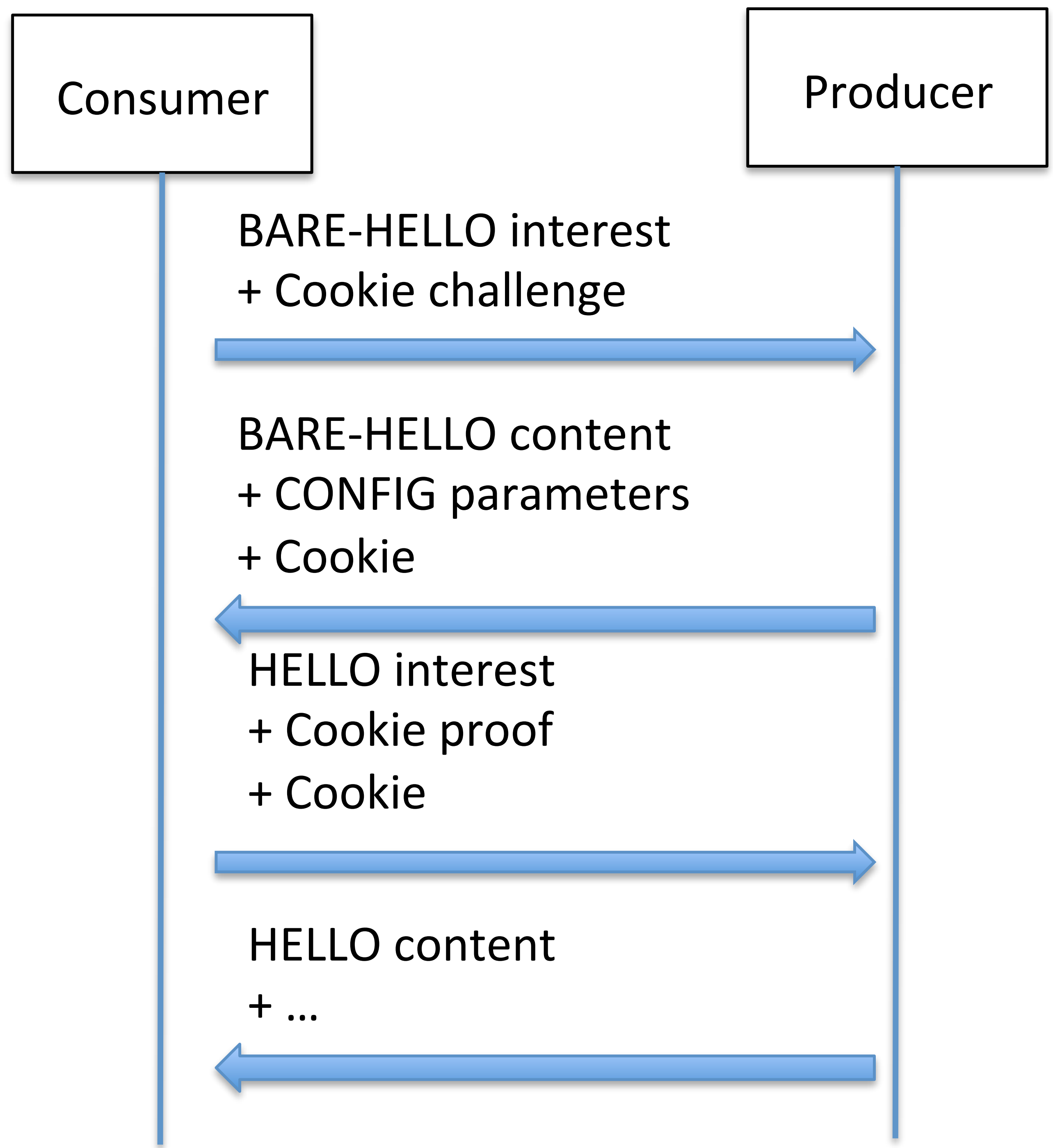
```
 +-------------------+
 | T1 | L1 |   V1    |
 +-------------------+
                ||
                \/
+-------------------------------+
| T2 | L2 |  Enc((T1,L1,V1))    |
+-------------------------------+
```

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



Cookie Generation

- Cookie challenge:

  $Y = H(X)$ 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:

  $Y = H(X)$ where $X \leftarrow \{0,1\}^{128}$

- Cookie:

  $P = \text{timestamp} \,||\, HMAC_K(Y, \text{timestamp})$

- Cookie proof:

  timestamp || X

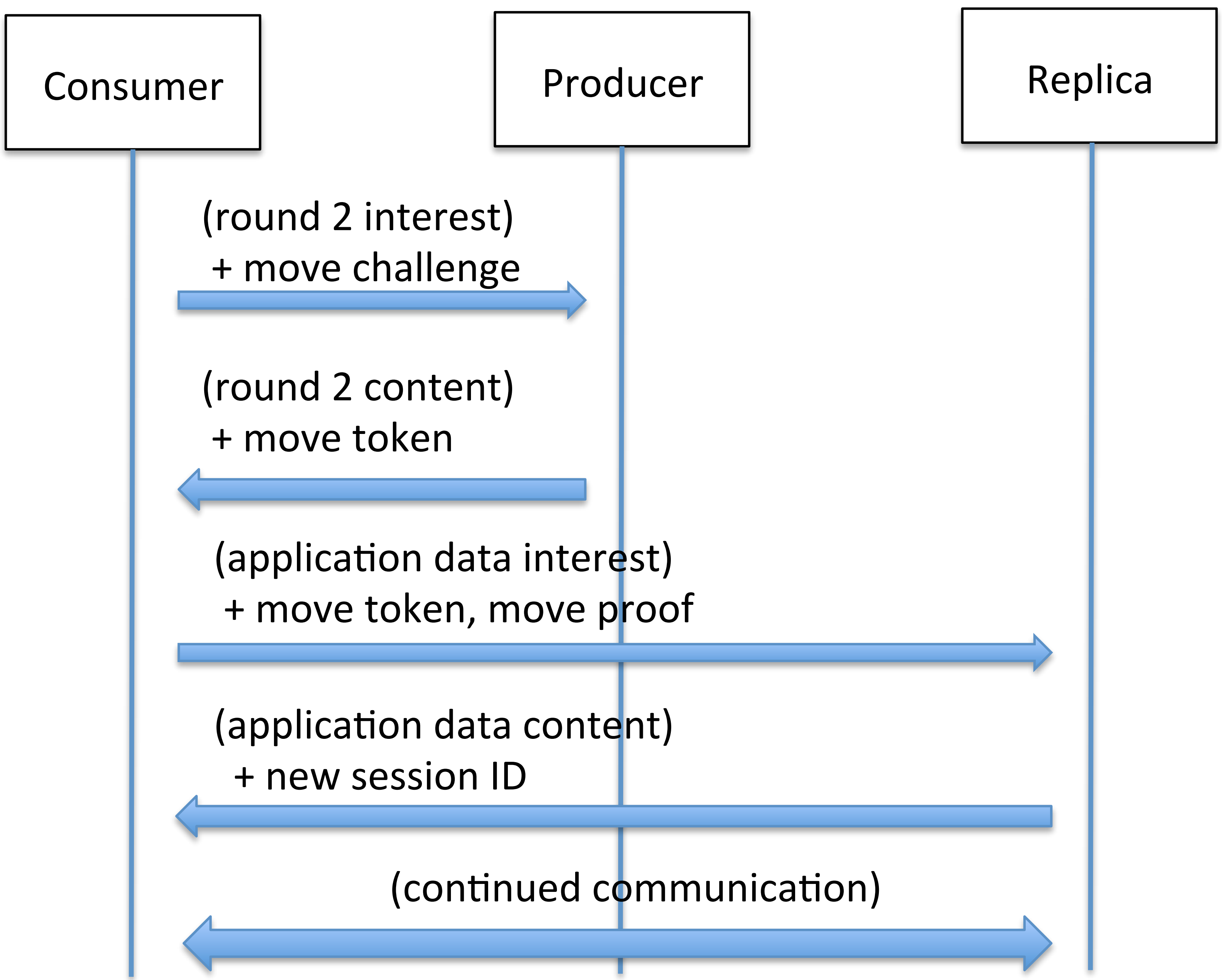
Cookie check:

- Verify freshness of the timestamp
- Verify HMAC tag using X and timestamp

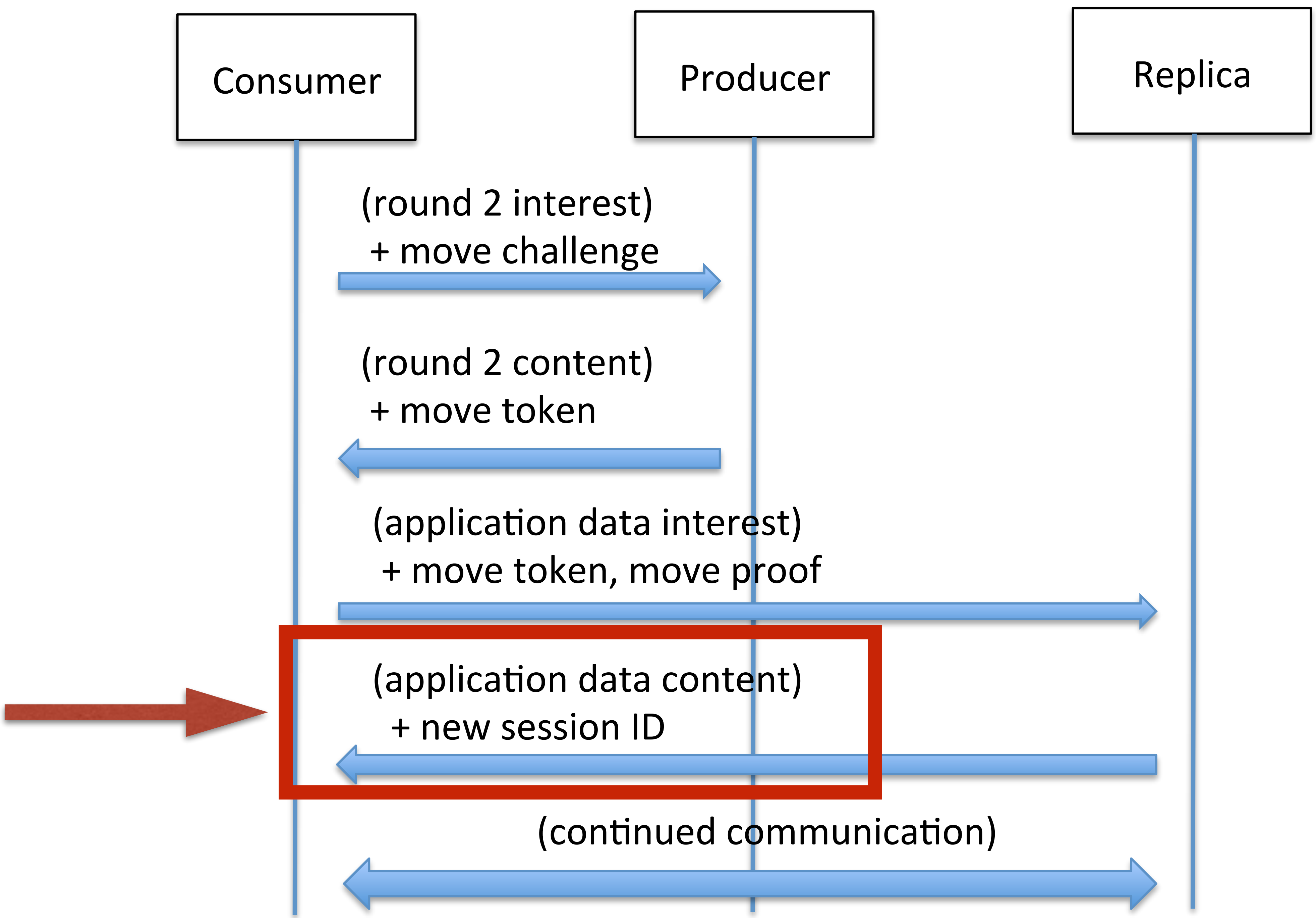
Session Mobility

- 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



Mobility Example



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

Feature	CCNx-KE	DTLS 1.2
DoS Prevention Cookie	Cookie derived client-provided hash premiere and timestamp	Undefined
Cipher suite options	AES-GCM (mandatory) with options for Salsa20+Poly1305	Stream ciphers (RC4), non-AEAD block ciphers
Timeout and retransmission	TBD	Grouped message retransmissions

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