Mobile Sessions in Content-Centric Networks

Marc Mosko, Ersin Uzun
Palo Alto Research Center
{mmosko, euzun}@parc.com

Christopher A. Wood+
University of California Irvine
woodc1@uci.edu

***Abstract*—Content-centric networking (CCN) is a network architecture for transferring named data from producers to consumers upon request. This shifts security from that of a connection or channel to the content itself. There remains, however, many critical uses for the traditional client-server communication model with secure sessions. For instance, in many CCN applications, producers need a way to transfer key material or secret information to consumers. Not only does caching this content fail to serve multiple consumers, encrypting it under long-term, static keys does not afford them any forward secrecy. Consequently, there is a real and present need for a CCN-friendly protocol whose security properties meet or exceed similar transport security protocols in IP networks. In this paper, we present the design and implementation of the CCNx Key Exchange Protocol – CCNxKE – the first protocol design for bootstrapping encrypted service-centric sessions in CCN. We compare our design to that of existing IP-based transport security protocols to highlight important differences, discuss several important use cases for CCNxKE and secure sessions in CCN, and present a preliminary performance assessment. Our experiments indicate that session encryption adds, on average, a 30% data transfer latency compared to unencrypted traffic using our prototype implementation.**

I. Introduction

Information-centric networking (ICN) architectures such as Content-Centric Networking (CCN) define the fundamental unit of communication as named and addressable data. Rather than moving data in the network by addressing the endpoint to which it should be delivered, data is obtained (transferred) by explicitly requesting it. Such a request, or *interest*, carries a data name and is routed based on this name towards some authoritative source or producer for the data. Once the request is satisfied, the response, called a *content object* or content, is forwarded along the requests' reverse path towards the consumer(s).

Since data is identified by a name, routers may opportunistically cache content object responses so as to satisfy future requests for the same content. This is done to reduce upstream bandwidth consumption and minimize the consumer data retrieval latency. As a consequence of separating data from its origin, there must be some means by which consumers can verify the authenticity of content. Content objects are authenticated in one of two ways: (1) they either carry some form of authenticator, such as a Message Authentication Code (MAC) or digital signature, which allows the recipient or an intermediate node with the appropriate context information to verify the content, or (2) they are requested with and subsequently verified by their cryptographic fingerprint (i.e., hash).

In CCN, confidentiality, unlike integrity and authenticity, has long been treated as an application-layer access control problem (see Section II-B). It is *not* a feature of the network protocol. The majority of existing approaches to provide content confidentiality are based on content encryption – the core idea of which is to encrypt content under some key such that only authorized consumers can obtain the decryption key(s) and thus consume the content. This approach has one crucial property that sets it apart from the transport security protocols that are ubiquitous in modern IP-based applications: Two legitimate requests for the same encrypted content from two authorized consumers will yield an *identical* response from the network if they are satisfied from a cache. This means that an adversary can link both the requests and responses together and therefore conclude that two consumers asked for the same data. This utility, if coupled with auxiliary information about the data that is requested, can lead to privacy breaches [1], [2].

Beyond this correlation issue, we claim that there is a clear and present need for data protected by session-based, ephemeral encryption keys in CCN. Exemplary applications that would benefit from this include online banking, payroll, and e-commerce applications wherein sensitive transactions or information should only be viewable by the endpoints engaged in the exchange of data. Moreover, this type of secure session protocol is often necessary to bootstrap the aforementioned encryption-based access control systems by securely transferring keying material.

For these reasons, we advocate for a session-based encryption protocol which offers the same benefits of modern protocols such as TLS 1.3 [3], QUIC [4], [5], and DTLS 1.2 [6]. Not only would such a protocol enable applications which deal with sensitive data and require forward-secrecy [7], which is the property that past traffic is protected in the event that long-term private keys are compromised, but it would also fill a void that has existed in CCN and related ICN architectures. To this end, we present the CCNx Key Exchange Protocol – CCNxKE – and show how it can be used to create secure sessions in CCN. CCNxKE allows a *consumer* to establish a shared, forward-secure secret with a name prefix, i.e., a *service*, which can then be used to bootstrap a secure session with said service. Importantly, it is *not peer-to-peer* as in IP-based transport security protocols. CCNxKE exploits the native multicast and authenticity properties of CCN to create sessions associated with a routable prefix. This allows the network to transparently handle issues such as load balancing sessions across different service endpoints. It also allows nodes participants to migrate freely around the network without continually tearing down and restarting sessions.

To execute CCNxKE, the consumer *does not require a routable prefix* for itself. CCNxKE will authenticate the service, similar to TLS, and may optionally authenticate the consumer. (For bi-directional traffic, the consumer may provide

+Supported by the NSF Graduate Research Fellowship DGE-1321846. Work was done while this author was at PARC.

ISBN 978-3-901882-94-4

its routable prefix, if it has one, to allow interests to flow both ways.) Additionally, CCNxKE borrows ideas from Kerberos [8] to allow the service to migrate a session to another name that actually provides data, such as a trusted content replica or repository. This allows client client authentication and authorization to be decoupled from the service that provides data. (See [9] for more details of this approach.) Lastly, CCNxKE introduces a novel technique to mitigate volumetric denial-of-service (DoS) attacks on a service in CCN.

From a performance perspective, CCNxKE is designed to minimize the number of rounds required in safely establishing a shared key to begin sending application data. (Here, application data can be sent from the producer to the consumer in content objects or, if needed, from the consumer to the producer in interests.) In the worst case, it requires two round-trip-times (RTTs) but in most cases only one RTT is required before sending application data.

We implemented CCNxKE with encrypted session support in the CCNx software stack [10] to assess its overhead. Our experiments indicate that session encryption adds, on average, a 30% data transfer latency compared to unencrypted traffic using our prototype implementation.[1]

Collectively, our contributions are as follows: (i) the first forward-secure key exchange protocol for CCN that follows the same driving security principles used in TLS 1.3, QUIC, and DTLS 1.2, (ii) additional DoS mitigation and mobility support beyond TLS and QUIC, (iii) a description of how to run encrypted CCNx sessions bootstrapped by CCNxKE, (iv) a comprehensive discussion of how and why CCNxKE deviates from modern transport security protocols, (v) an expose of important use cases enabled by CCNxKE, and (vi) an experimental assessment of its performance as implemented in the CCNx software stack.

II. Preliminaries

This section presents an overview of the CCN architecture[2] and work related to confidentiality, privacy, and transport security. Those familiar with these topics can skip it without loss of continuity.

A. CCN Overview

In contrast to IP networks, which focus on end-host names and addresses, CCN [11], [12] centers on content by making it named, addressable, and routable within the network. A content name is a URI-like string composed of one or more variable-length name segments, each separated by a '/' character. To obtain content, a user (consumer) issues a request, called an *interest* message, with the name of the desired content. This interest can be *satisfied* by either (1) a router cache or (2) the content producer. A *content object* message is returned to the consumer upon satisfaction of the interest. Moreover, name matching in CCN is exact, e.g., an interest for /ifip/networking/2017/cfp can only be satisfied by a content object named /ifip/networking/2017/cfp.

In addition to a payload, content objects include several fields. In this work, we are only interested in the following three: `Name`, `Validation`, and `ExpiryTime`. The `Validation` field is a composite of (1) validation algorithm information (e.g., the signature algorithm used, its parameters, and a link to the public verification key), and (2) validation payload (e.g., the signature). We use the term "signature" to refer to this field. `ExpiryTime` is an optional, producer-recommended time after which a content object should not be cached. Conversely, interest messages carry a mandatory name, optional payload, and other fields that restrict the content object response. The reader is encouraged to review [12] for a complete description of all packet fields and their semantics.

Packets are moved in the network by routers or forwarders. A forwarder is composed of at least the following two components:

- *Forwarding Information Base* (FIB) – a table of name prefixes and corresponding outgoing interfaces. The FIB is used to route interests based on longest-prefix-matching (LPM) of their names.
- *Pending Interest Table* (PIT) – a table of outstanding (pending) interests and a set of corresponding incoming interfaces.

A forwarder may also maintain an optional *Content Store* (CS) used for content caching. From here on, we use the terms *CS* and *cache* interchangeably.

Forwarders use the FIB to relay interests from consumers to producers and the PIT to forward content object messages along the reverse path to consumers. More specifically, upon receiving an interest, a router R first checks its cache (if present) to see if it can satisfy this interest locally. If the content is not in the cache, R then consults the PIT to search for an outstanding version of the same interest. If there is a PIT match, the new incoming interface is added to the PIT entry. Otherwise, R forwards the interest to the next hop according to its FIB (if possible). For each forwarded interest, R stores some amount of state information in the PIT, including the name of the interest and the interface from which it arrived, so that content may be sent back to the consumer. When content is returned, R forwards it to all interfaces listed in the matching PIT entry and said entry is removed. If a router receives a content object without a matching PIT entry, the message is deemed unsolicited and subsequently discarded.

B. Confidentiality and Privacy in CCN

Content-based encryption is arguably the most popular technique for protecting CCN content from unauthorized disclosure. This strategy permits content to be disseminated throughout the network since it cannot be decrypted by adversaries without the appropriate decryption key(s). Many variations of this approach have been proposed based on general group-based encryption [13], broadcast encryption [14], [15] and proxy re-encryption [16]. Kurihara et al. [17] generalized these specialized approaches in a framework called CCN-AC, an encryption-based access control framework that shows how to use manifests to explicitly specify and enforce other encryption-based access control policies. Consumers use information in the manifest to (1) request appropriate decryption keys and (2) use them to decrypt content object(s). The NDN NBAC [18] scheme is similar to [17] in that it allows decryption keys to be flexibly specified by a data owner. However, it does this based on name engineering rules instead of explicit configuration. Interest-based access control [19] is a different type of access control scheme wherein content was optionally encrypted. Access was protected by making the names of content derivable by only authorized consumers. NDN-ACE [20] is a recent access control framework for IoT environments which includes a key exchange protocol for

[1] We believe there is ample room for improvement with a more optimized implementation focused on procedures such as, e.g., packet encoding and decoding.

[2] Named-Data Networking [11] is an ICN architecture related to CCN. However, since CCNxKE was designed for ICNs that have features which are not supported by NDN (such as exclusively exact name matching), we do not focus on NDN in this work. However, CCNx could be retrofitted to work for NDN as well.

distributing secret keys to sensors. We revisit NDN-ACE in Section VI-A.

III. CCNxKE Design

In CCN, there is a trend to treat both confidentiality and privacy as problems of the application. This leads to diverse approaches for different niche applications. These ad hoc solutions may be suitable for some applications, but we claim that content-based encryption alone is not appropriate for a variety of applications whose primary objective is *not* content distribution, e.g., online banking and payroll, where *privacy* is equally important to confidentiality. Encryption-based access control does not offer privacy, primarily because it only protects the content payloads and, by default, services more than one consumer at a time. (See [2] for more details about the challenges of privacy in CCN.) Such applications need transport security akin to what is provided by TLS and related protocols in today's Internet.

The primary goal of the CCNxKE protocol is to enable transport security by deriving forward secure traffic encryption keys between a consumer and service. With respect to key exchange protocols, forward secrecy is the property that session keys are not compromised even in the event that either party's long-term private keys are leaked. This is a much sought-after property in communication protocols in recent years [21]. As such, CCNxKE adopts the core exchanges in the TLS 1.3 protocol to derive all forward-secure secrets. These secrets are then used to bootstrap session-based communication, wherein traffic is encapsulated and encrypted using authenticated encryption (AEAD) for transmission between two endpoints (i.e., a consumer and service) under a session associated with a single name. With regards to DoS, the address-less nature of CCN packets raises the need for a new mechanism to prevent volumetric DoS attacks on the producer. CCNxKE introduces a novel approach to address this problem that induces minimal computational and constant storage overhead at the producer.

CCNxKE also goes beyond most IP-based transport security protocols in its feature set. One important new feature is the ability to migrate sessions from an authentication and authorization service to one that serves content. To see why this is useful, consider the following. Assume a consumer wishes to access content within the the /parc/ namespace, but this content is actually hosted under by a trusted content store under the /xerox/cdn prefix. To begin, the consumer would initiate a key exchange with the former name. Upon authenticating the client and completing the key exchange, the producer who executed this protocol with the consumer would provide a special migration (move) token to the client along with the prefix /xerox/cdn. The client could then use this move token to resume the session under the new namespace to securely fetch the desired content. Today, the equivalent method is to establish a TLS session to a server, receive a redirect (e.g. an HTTP 3xx), then establish a second TLS session with the second server.

A. Protocol Description

Having motivated the goals and outlined the major properties of this protocol, we now describe its technical details. CCNxKE is built on the CCNx 1.0 [12] protocol and uses standard interest and content objects as a vehicle for data transfer. It relies upon on the following minimal assumptions about peers participating in the CCNxKE protocol: (1) Consumers know the namespace prefix of the desired service to which to connect; (2) Interests and content objects carry distinguished fields (separate from the payload) that contain CCNxKE signaling information (this is done to not pollute application data with key exchange signaling information); (3) Forwarders have no participation in the protocol.

CCNxKE operates in (at most) two rounds, where each round requires a single RTT to complete. These two rounds are used to derive the session secrets (and keys). Application data can be sent in either interest or content objects in the third (and beyond) rounds (as described in Section IV). The exact purpose of each round is described below.

- R-1 Perform a bare hello exchange to obtain the public producer parameters and a source cookie. The source cookie is used to prevent DoS at the producer by pinning the key exchange to the same consumer which initiated the first interest. In this round, the consumer sends a BareHello interest and the producer responds with a HelloReject content object.
- R-2 Perform the key exchange to establish the shared traffic secret. In this round, the consumer sends a FullHello interest and the producer responds with a HelloAccept content object. The consumer may also optionally send non-forward-secure data encrypted to the producer in this round.
- R-3 Send the first bit of application data to the destination service and (optionally) migrate the session to a new service. All messages sent in this round (and those after) are protected with forward-secure authenticated encryption.

Each round is a single interest and content object exchange. Unless otherwise specified, all content objects are signed by the producer(s) and verified by consuers in the normal way. The choice of trust model by which the consumer verifies the producer's signatures is orthogonal to the actual authentication mechanics in CCNxKE. A consumer must use a trust model through which the producer is trusted, else a session cannot be created. In the first and second rounds, the interest name includes the service's prefix *with a random nonce appended as the suffix*. (The nonce prevents cache hits and ensures that the interest reaches the service). For example, if the service prefix is /parc/service, then an interest for the first round might be /parc/service/01234ABCDE. In the third round and beyond, the interest name additionally includes a session ID and other context information (see Section IV).

An overview of the full protocol with the standard messages sent in each of the three rounds is shown in Figure 1 below. The key derivation information is outlined in Section IV-A. In the following sections, we describe the rounds of this protocol in more detail.

B. Round 1: DoS Prevention

Similar to QUIC and DTLS, CCNxKE operates over a stateless channel. Volumetric DoS attacks aimed at wasting CPU cycles on a victim producer are notoriously difficult to protect against because the server has no prior relationship with the client. In IP-based protocols, the server cannot trivially filter incoming packets as the attacker can easily spoof the source address. QUIC and DTLS deal with this attack by forcing the client to present a cryptographic token which binds the incoming packet to the address presented. These tokens are typically coupled with IPSec ESP style replay detection based on sliding windows that prevent replay attacks for messages carrying these tokens [22].[3]

[3]Neither DTLS nor QUIC have solutions that prevent legitimate clients from flooding the server.

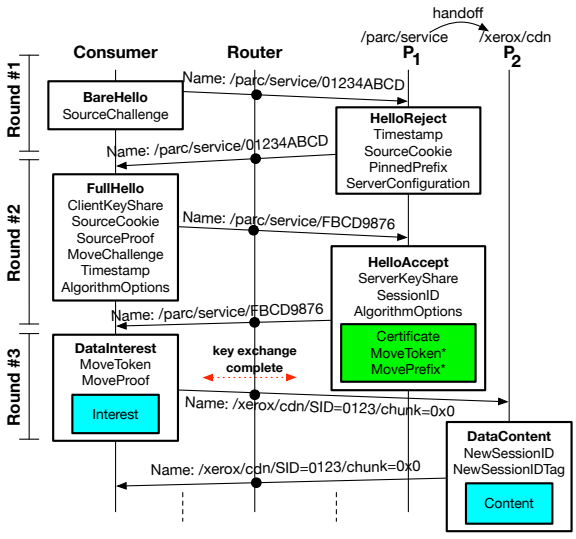

Fig. 1. The base CCNxKE protocol showing a session migration between producers P_1 and P_2. The green fields are encrypted with a non-forward-secure secret (see section IV-A) and the blue fields are encrypted with the forward-secure keys.

In CCN, interests have no source address. This means we need another way to bind an interest to its originator. Our solution relies on a simple proof-of-knowledge protocol that proves the origin of the interest is that with whom the producer had previously interacted. The proof works as follows: A consumer will present a random challenge to the server during the first round. The producer computes the stateless source cookie from this value and returns the result to the consumer. The consumer must then, in the subsequent round, present proof that it generated the challenge to the producer. Once the producer verifies the proof in this second interest, it concludes that the originator of said interest must be the same entity for whom the cookie was constructed. This is functionally equivalent to the source address cookie used in DTLS and QUIC.

To construct this source cookie and complete the originator proof, we make use of a strong cryptographic hash function H that is preimage resistant and a keyed message authentication code (MAC) F_k. Cryptographically, a hash function H is preimage resistant if, given an output $y = H(x)$, an adversary find an input x' such that $H(x') = H(x) = y$. The originator proof works as follows:

1) Cr generates a random $x \leftarrow \{0,1\}^{256}$ (SourceProof) and computes $y = H(x)$.
2) Cr sends y to the producer in the BareHello message.
3) P computes SourceCookie $= F_k(y||t)$ where t is the current timestamp and k is the producer's secret key.
4) P returns SourceCookie and t to Cr in the HelloReject message.

To prove ownership of SourceCookie, Cr must present SourceCookie, the SourceProof x, and Timestamp t to the producer in the subsequent (round two) interest. Before processing the cookie, the producer first checks to see that the interest is not a replay. (This can be done using the sliding window technique of [22].) After, to verify that the cookie is correct, P checks that SourceCookie $= F_k(H(x)||t)$. If equality holds, the source cookie is accepted and the producer proceeds with the protocol. Based on the properties of H, an adversary can only succeed in forging a source cookie or proof with negligible probability.

The remainder of the round one traffic is catalog information for the rest of the exchange, e.g., the signed server parameters (ServerConfiguration) that contain the supported algorithms (key exchange algorithms and ciphersuites) and other miscellaneous information. One important response from the producer is a *pinned prefix*. This allows the producer to *pin* the key exchange to a specific end-host serving a namespace. For example, suppose the client attempts to create a session with the namespace /parc/service. The round one interest to create this session may be multicast to several producers who can provide this service. Suppose that a service node in control of the /parc/service/nodeA prefix elects to handle the session. It would then respond with the pinned prefix /parc/service/nodeA to ensure that the subsequent interest would go only to that node and not forwarded to others in the /parc/service namespace.

C. Round 2: Key Exchange

The second round serves several purposes: (1) choose the session algorithm options, (2) exchange material to create session secrets, (3) authenticate the server and (optionally) the client, and (4) provide a move token and prefix to migrate the session (if necessary). The following sections explain each of these steps in more detail.

1) Algorithm Options: The consumer selects its preferred cipher suites, signature algorithms, and key-exchange methods from the server's parameters provided in the ServerConfiguration of the HelloReject message. These options are conveyed in the AlgorithmOptions field in the consumer FullHello message. The producer will echo the chosen options back to the consumer in an identical field if the FullHello is accepted.

2) Handshake Variants: CCNxKE supports two handshake variants, including (1) a full key exchange wherein the consumer has no prior information about the service beyond its prefix and (2) a resumption variant. The full variant is outlined in Figure 1. In it, the consumer and producer exchange fresh Diffie Hellman pairs carried in the KeyShare fields which are used to derive the session secrets (as detailed in Section IV-A). (This exchange is what enables the session to have forward secrecy; session secrets are not derived based on static information.) Notice also that the server provides a MoveToken and MovePrefix. These are used to migrate the session in a Kerberos-like [8] fashion. The details of which are described in Section III-C4. Lastly, the session ID is generated and provided by the service in the HelloAccept message. To applications, the session ID is an opaque identifier that only refers to a session. However, the service may embed state within it if desired, similar to a TLS session ticket. CCNxKE does not prescribe the generation of this value.

The resumption variant is designed to reduce the number of round trips before sending data. Specifically, it allows the consumer to resume a previously generated session with a pre-shared key (PSK) and, simultaneously, send data after 1-RTT encrypted under that PSK. In addition to sending the PSK optionally encrypted data, the consumer may send a KeyShare field in the FullHello so that they can derive a forward-secure traffic secret for encrypting data after the first message. Note that the producer always sends a new session ID to the consumer in the HelloAccept message so that the same session ID is not re-used across resumed sessions.

3) Peer Authentication: As always, the producer must always authenticate itself. Unlike the first round, the HelloAccept message, and all subsequent content objects sent from

the producer, carry a MAC authenticator instead of a digital signature. In the HelloAccept message, this MAC is generated from the secret key derived from the key exchange algorithm. In content objects after this round, this MAC is the tag from the output of the authenticated encryption algorithm used to encrypt (or encapsulate) data from the producer. We expand on this in Section IV-C. This is necessary to prove knowledge of the shared secret. See [23] and Section V for more details. Also, the producer may supply its public key certificate (Certificate) in the HelloAccept message. However, this is not necessary since the same producer must have supplied this in the HelloReject message.

To support mutual authentication, the consumer must be able to present its certificate and a signature to the producer. However, this can only be done after the key exchange is complete. This means that mutual authentication takes place *after* the second round. (Again, see [23] for details.) To authenticate itself, the consumer mirrors the producer's behavior *after* receiving the HelloAccept: it provides its public key certificate (Certificate), signs the interest carrying the message, and computes a MAC over the bundle which is inserted into the interest validation field.

Related to mutual authentication, the consumer is able to specify its own routable prefix in this post-handshake message so that the service may send interests to the consumer for bidirectional communication. This is an optional feature and is enabled at the consumer's discretion.

4) Session Migration: The CCNxKE session migration feature is inspired by Kerberos and the recent LURK (limited use of remote keys) BOF in the IETF [24]. The goal is to allow a service to securely move a session from the node that performed the handshake computations to another node or service that can provide content for the consumer. This effectively decouples the authentication and authorization steps in CCN. To do so, we *do not want* this different service, called the replica, to have access to the private key associated with the primary service. Under this constraint, the migration step must allow the session secrets (or more specifically, the initial traffic secret TS_0, described in Section IV-A) to be recovered at the replica; this the minimal information necessary to derive the keying material used to encrypt traffic. Moreover, we desire this recovery to also prove that the consumer had previously authenticated and performed the handshake with authentication service. This prevents the replica from servicing unauthorized consumers. Lastly, we want the recovery to not be itself another form of computational DoS that can be exploited by malicious consumers.

To achieve these goals, we use a proof technique similar to that which is done for the SourceCookie. We assume that the authentication service is able to create and update a shared symmetric key k with ID k_{ID} with the replica. With this information in place, the session migration protocol works as follows.

1) Cr generates a random $x \leftarrow \{0,1\}^{256}$ and computes $y = H(x)$.
2) Cr sends y to the producer in the FullHello message.
3) P computes the $\mathsf{MoveToken} = Enc_k(y||TS_0||t)$ where t is the current timestamp and k is the producer's secret key. Here, Enc is an authenticated encryption algorithm such as AES-GCM [25] that produces ciphertext and an authentication tag.
4) P returns the tuple $(\mathsf{MoveToken}, k_{ID}, t)$ to Cr in the HelloAccept message.

To prove ownership of MoveToken, Cr must present this same tuple along with x to the replica in the third round interest. The replica must then verify the token and confirm that it was produced by a trusted service. To do so, it performs the following:

1) Check to see if the interest is a replay. If not, then discard the interest.
2) Check to see if k_{ID} corresponds to a shared key with one of its trusted services. If not, then discard the interest.
3) Attempt to compute $Dec_k(\mathsf{MoveToken})$. If the decryption fails, e.g., because the authentication fails, then discard the interest.
4) Compute $H(x)$ and verify that it equals the challenge y. If not, discard the interest.

If all of these steps pass, then the replica knows that a trusted service previously authorized the consumer who generated the incoming interest to access stored content. As with the SourceCookie, the MoveToken can only be forged with negligible probability by the CCA-secure properties of the authenticated encryption algorithm. As a technical matter, the replica must provide a new session ID for the consumer after authenticating its MoveToken. The consumer then uses this new session ID for all subsequent communication. Optionally, when migrating, Cr and the replica could perform another key exchange to update their traffic secret. (This would be done by sending mutual key shares in the interest and content messages. These are not shown in the protocol shown in Figure 1.) Moreover, the producer and service must update their shared key after a move token is issued. This prevents the compromise of the producer from exposing previously migrated sessions.

We comment that we the MoveToken can be generated by the producer at any point after the key exchange is complete. However, if client authentication is required prior to issuing a move token, then the producer cannot send one until after the third round. This is due to the fact that client authentication cannot safely happen until after the key exchange is complete.

IV. Secure Sessions

After the key exchange is complete, the consumer and producer derive the keying material used to encrypt traffic. In this section, we describe the key derivation steps[4] and then show the mechanics necessary to perform interest and content encryption between a consumer and service.

A. Session Secret and Key Derivation

The forward-secure secret derived in CCNxKE is the *traffic secret* (TS). This is ultimately derived from a *master secret* (MS). In the standard handshake, TS is derived from the consumer and producer DH key shares DH_C and DH_P, respectively. Specifically, let $s = DH(DH_C, DH_P)$ be the output of a DH exchange using the two shares DH_C and DH_P. MS is then computed as $\mathsf{HKDFExtract}(0, H(s))$, where H is a hash function. (See [26] for the HKDF details.) In the resumption variant, the 1-RTT data key is derived from the PSK k and the subsequent forward secure MS is derived from the pair of DH shares *and* k. Here, the 1-RTT key k_e is computed as $k_e = \mathsf{HKDFExtract}(0, H(k)$ and MS is subsequently computed as $\mathsf{HKDFExtract}(k_e, H(s))$. After this computation, MS is then used to create the initial TS (TS_0) as follows:

$$TS_0 = \mathsf{HKDFExpandLabel}(MS, \text{"traffic secret"}, \mathsf{ke\_hash}, L)$$

Here, the parameter L is the desired length of the secret to be created and is usually 256 bits. Also, ke_hash is the hash

[4]The key derivation material is based on TLS 1.3 [3].

of the concatenated CCNxKE messages received up to the point of derivation. (In this case, it is the hash of the entire transcript.) TS_i is updated by simply performing another KDF operation. Moreover, all symmetric keys and IVs are generated by expanding TS_i.

B. Message Encapsulation

With the cryptographic keying material, both the consumer and service can encrypt messages to exchange with one another. Message encryption is done by *total encapsulation*: full CCNx interests and content objects with (or without) names and provenance information are encapsulated in "wrapper" interests and content objects that are transferred between the consumer and service. We use *total encapsulation* because it leaks the least amount of information about the encrypted plaintext and allows for easy padding. The outer wrapper has a normal CCN name that identifies (a) the prefix of the service, (b) the session ID (SID), and (c) a sequence number (expressed as a chunk number). For example, this name could be /parc/gateway/SID=0x1234/chunk=0x12341234. This name is carried by the interests and content objects exchanged between the consumer and service. The payload of these messages contain the actual encapsulated messages (interest or content object) to be sent between the two parties. We refer to these encapsulated messages as the *inner messages* and the transport messages with the outer name as the *outer messages*. For brevity, we refer to the names associated with inner and outer messages as NI and NO, respectively.

It is possible (and likely) that each NI is unrelated to each NO. For example, it is valid for a message with the previously state outer name to carry an encapsulated message with a name /irtf/icnrg/agenda96. Also, it is not a requirement for the inner messages to "correspond" to the outer messages. For example, if a consumer sends three interests with outer names NO_1, NO_2, NO_3 and inner names NI_1, NI_2, and NI_3, the producer can return these names in any order. It could, for example, put content objects with name NI_3 in NO_1, NI_1 in NO_2, and NI_2 in NO_3. We do, however, require that one outer message request return one outer message response as per the usual symmetric nature of CCN. It is possible for the outer message response to be an Interest Return (NACK) if the producer cannot handle the corresponding inner message.

The point of separating inner and outer messages is that inner messages have only end-to-end meaning, i.e., between the consumer and service. Inner messages may themselves be protected with a different form application-specific encryption, e.g., broadcast encryption. The encapsulation method does not preclude any inner message format – it only places a cap on the total size of the message.[5]

C. Processing Chain

The processing chain that performs encapsulation and decapsulation from a consumer source to a service sink is shown in Figure 2. The compression and decompression stages are optional and are not strongly tied to the encrypted session. If used, we assume compression is done so as to avoid attacks similar to CRIME [28].

The encryption and decryption logic happens in the encryption and framing stages of the chain. Given an interest with the inner name NI to be sent as the n-th message in session SID=0x1234, the encryptor performs the following

[5]In CCN, the total packet length of the inner message is bounded by the maximum packet size of 64KB [12], [27].

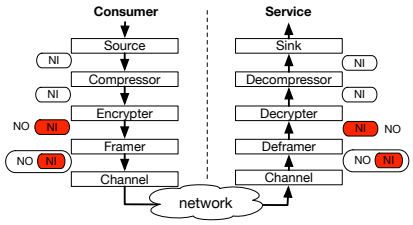

Fig. 2. The encrypted CCNx session pipeline.

steps: First, the AEAD nonce is derived from the local state and unique packet number. Then, the wire format encoding of the inner message is encrypted with the chosen AEAD algorithm using the IV and the client or service write key. (The additional authenticated data for the AEAD algorithm is set to the fixed contents of the to-be-created outer message that do not include the payload.) The output ciphertext is used as the encapsulated message in the outer message with an NO such as /service/prefix/SID=0x1234/chunk=n. Moreover, the authentication tag from the encryption algorithm is inserted into the `Validation` payload of the outer message.

The decryptor performs the inverse operation given the NO and its own copy of the consumer write key and nonce. If an error occurred in the replica decryption chain then an interest return (NACK) with an appropriate error code is encrypted and returned to the consumer in an outer content object. Once the sink receives the inner message it may respond with a satisfying content object or an interest return. The processing chain does not distinguish between these responses and encrypts them equally.

V. Security Analysis

To assess the security of the CCNxKE protocol, we adopt the attacker model of the OPTLS protocol used in [29], which originates from [30]. The attacker here is a man-in-the-middle adversary Adv whose goal is to learn information about active sessions. Thus, security means that compromising a given session should have no adverse affects on other *non-matching* sessions in the network. By non-matching we refer to active and in-progress sessions that are not equal to the session that was compromised. Adv is assumed to have complete control over all links between active parties in a CCNxKE protocol execution. In particular, Adv can intercept, modify, and inject messages into exchange when desired, as well as control the timing of messages in the protocol. To capture these capabilities, among others, Adv is given access to the following functions:

- StateReveal(): This function returns all the state information associated with an incomplete session, i.e., one that is still in performing a handshake.
- Reveal(): This function returns the traffic secret associated with an already-completed session.
- Corrupt(): This function compromises a given entity such that any and all long-term term secrets are disclosed.

For uniqueness, we refer to a session identifier as the concatenation of all messages transferred thus far in one invocation of the protocol. Two sessions are therefore matching if their session identifiers are equal. Any session which is subjected to any of the above functions is heretofore referred to as *exposed*. Under this notion, the security of a CCNxKE session (i.e., its traffic secret) is defined as the advantage Adv has in distinguishing the traffic secret of a *chosen* session, migrated

(moved) or not, and a random value. Moreover, it must be the case that Adv learns nothing about the traffic secrets of non-exposed sessions, even if other non-matching sessions were exposed.

We are also concerned with forward secrecy. We capture this by relaxing the definition of un-exposed sessions to those which can be corrupted even after a session is complete. (Corruption cannot occur during the session establishment phase.) Lastly, we strive for the protocol to be secure even in the event of an ephemeral secret disclosure. This occurs when the StateReveal() query is invoked for the producer.

In our analysis we consider two variants of the CCNxKE protocol: stationary and multi-homed. The stationary variant is one where the producer *does not* provide a MoveToken to the consumer. However, this does not prevent Adv from injecting in the handshake. Multi-homed CCNxKE is one in which the producer provides a MoveToken. This variant requires more care when assessing the security since it involves multiple, possibly corrupt, producers.

Definition 1: Stationary CCNxKE is secure if for all probabilistic, polynomial-time adversaries Adv it holds that:

1) If two uncorrupted parties complete matching sessions then they derive the same traffic secret.
2) The probability of Adv winning at the aforementioned distinguishing game is negligible.

Definition 2: Multi-homed CCNxKE is secure if for all probabilistic, polynomial-time adversaries Adv it holds that:

1) Stationary CCNxKE is secure with respect to Adv.
2) Exposure of migrated sessions does not compromise other non-exposed sessions.

Using these definitions, we now assess the security of the CCNx-KE handshake.

Claim 1: Stationary CCNxKE is secure.

Proof: (Sketch) Most of the handshake protocol in stationary CCNxKE reduces to the OPTLS (and SIGMA) protocol in [29]. Notable exceptions are, e.g., the use of the SourceCookie to prove the origin of an interest. These are extra inputs into the handshake transcript that are used when stretching the *master secret* to the traffic secret. ■

Claim 2: Multi-homed CCNxKE is secure.

Proof: (Sketch) The core problem here is that a session is migrated to another producer. To be secure, it must be true that compromising the original producer does not reveal any information about other active sessions or the migrated session. Recall how session migration occurs. The consumer presents a MoveProof and MoveToken to the new service, the latter of which contains TS encrypted for the service. For Adv to distinguish this from a random value, as in the stationary case, it must learn some information about TS. It can get this from (a) the original producer, (b) the MoveToken, or the new service. First, corruption of the new service will reveal the TS, but this will not compromise any other non-exposed sessions. Second, since the MoveToken is encrypted with a key shared between the original producer and new service, Adv can only access the information if it corrupts one of these parties. By compromising either one, the shared key is learned and the token can be decrypted. However, this also does not reveal information about past sessions since (a) the shared key is always updated after every move token and (b) the consumer and replica derive fresh traffic secrets during the migration step. Lastly, compromising the producer to acquire the TS is ineffective since this state is removed after the MoveToken is generated and returned to the consumer. Therefore, revealing the migrated TS only harms a targeted session, and this TS is indistinguishable from a random value. ■

VI. Discussion

In this section we compare CCNxKE to modern transport encryption protocols to support key design decisions. We also describe several use cases for secure sessions in CCN.

A. Protocol Comparison

For obvious reasons, the design of CCNxKE tracked that of modern transport security protocols. Deviations were only made when some feature of these IP protocols did not map to the CCN communication model. We will compare CCNxKE to TLS 1.3 [3], DTLS 1.2 [6], and NDN-ACE [20].[6]

With respect to TLS 1.3, CCNxKE distinguishes itself in four important ways:

1) CCNxKE sessions can be migrated to other producers (services), whereas TLS sessions are pinned to the same server which completed the handshake. This feature was added to CCNxKE for reasons outlined in Section III.
2) Communication is *only* one-way unless the consumer supplies its own routable prefix to the producer in round 2. Conversely, TLS enjoys the benefits of full-duplex TCP streams. CCNxKE does not specify mechanisms for providing transport semantics similar to TCP (as DTLS does). This is the responsibility of some component above the session encryption layer, which is outside the scope of this work.
3) CCNxKE uses custom source cookies and move tokens to minimize computational DoS on services. Conversely, TLS enjoys DoS protections from TCP via, e.g., SYN-cookies [32].
4) CCNxKE encrypted sessions transfer *regular CCN messages* which have their own associated provenance and authenticity information. This is in contrast to TLS which enables secure byte streams. This encapsulation method was chosen to retain the CCN message authenticity proofs and permit recursive (nested) sessions with CCNxKE.

The remaining *features* between CCNxKE and TLS 1.3 are isomorphic, such as the supported cipher suites and key exchange algorithms (modulo encoding schemes).

With respect to DTLS 1.2[7], CCNxKE uses its own DoS cookie mechanism to bind a key exchange request to a single originator. This is in contrast to the mechanism supported by DTLS which binds these packets to source IP addresses. Also, CCNxKE is not peer-to-peer as DTLS. Rather, it's between a consumer and *some service that owns the authoritative namespace*. This makes CCNxKE a one-to-many protocol due to the multicast nature of interests and routing in CCN.

NDN-ACE is a framework for access control in IoT environments that includes its own key exchange and message transmission protocol for distributing secret keys and then using them. However, these have significant weaknesses from a security and performance perspective. First, it is highly vulnerable to DoS attacks since there is nothing to prevent

[6]We omit QUIC since, at the time of writing, it is moving towards adopting the TLS 1.3 handshake for secret derivation [31].

[7]We only comment on the differences between CCNxKE and DTLS 1.2 *for features that were not updated or fixed in TLS 1.3* and those which were not mentioned in comparison to TLS 1.3.

malicious consumers from overloading the authorization server (AS) with fake interests that cause it to perform unnecessary signature verification and DH operations. Second, the actual key that is distributed (the access key) is not forward-secure. Third, this key is only used to authenticate interests sent after the exchange; interest messages sent as part of the primary message transmission protocol contain cleartext names and are therefore not private. The NDN-ACE framework therefore offers no communication privacy since the intent of every action can be discerned from its name. In sumation, NDN-ACE is a protocol to distribute a *static* secrets to consumers, not a protocol to afford them any confidentiality with forward-secrecy.

B. Session Use Cases

Traditional Client-Server Applications: Many traditional client-server applications such as banking and payroll require a client to securely communicate with a single server (or service) to perform sensitive operations or transfer private data. Moreover, in many existing CCN applications that employ encryption-based access control, the process by which consumers securely obtain their private keys from the producer is often overlooked. For example, in applications using broadcast encryption schemes such as [14], a trusted key generator is responsible for producing client secret keys. Therefore, there must be a way to securely transfer these keys to the client. Since this type of information is meant for a single, specific consumer, there is no benefit to caching the data (beyond transport retransmission use). Furthermore, the transfer should use best-practice cryptography for data in transit, i.e., use forward-secure keys to encrypt all traffic. CCNxKE enables a way to address these problems. For applications where the service must send data to the consumer, the latter can authenticate themselves in the second round of the handshake to prevent any unauthorized information disclosure. Using the previous key distribution example again, if the service authentication and key generation agents are physically disjoint then it would be possible for the the service to issue a MoveToken to allow the consumer to securely communicate with the key generator.

Secure Content Distribution: A powerful use case for CCNxKE is in secure content distribution. In modern web applications that use CDNs to distribute popular content, the client application almost always obtains data over a TLS session from the CDN node. However, the client only does so because the content producer has granted the CDN access to a private key that allows CDN nodes to masquerade as the actual server during the session establishment step. In this way, the client remains oblivious to whether or not it is communicating with the actual server or a CDN. Recently, the IETF LURK (Limited Use of Remove Keys) [24] BOF was created to solve this problem of offloading TLS without exposing any private keys to the CDN. CCNxKE solves this problem through the use of the MoveToken. In particular, the producer can grant the consumer secure access to content stored at a CDN by issuing a move token for a said CDN, or replica. The consumer would then migrate its session to the replica using this move token and continue to fetch the required content.[8]

The MoveToken allows a consumer to create a secure session with the replica to securely fetch content. However, in many modern applications, the content stored in the replica is itself encrypted under some form of group encryption, e.g., broadcast encryption. This prevents the service from disclosing the contents of the its data to the replica and therefore uses the replica as a distribution medium. The consumer may obtain the secret keys necessary to encrypt said content from the server using a separate session (as described in the prior use case).

[8]Since, today, services and replica have a mutually trusting relationship, it is not egregious to believe that the service and replica could generate and maintain a shared secret to compute move tokens.

VII. Performance Analysis

We implemented a minimal version of the CCNxKE protocol and processing chain to test the amount of overhead it adds to standard CCNx traffic. Our implementation consists of the full handshake protocol without session migration. The key exchange algorithm uses the secp256r1 (NIST P-256) curve and encryption uses a 256-bit key with AES-GCM. To compare the added overhead, we augmented a CCNx file transfer application to transfer data over a secure session. We are concerned with two primary metrics: the key exchange overhead (the time required to complete the first two rounds at the consumer) and the data transfer computation overhead (the additional amount of time added by encrypting and decrypting packets). Thus, we configured the consumer and producer in a simple single-hop topology to perform the experiment, i.e., the consumer and producer are connected to the same forwarder.[9] All experiments were conducted on a machine with an 2.8 GHz Intel Core i7 processor with 16GB of DDR3 memory using OpenSSL.

The file transfer application works as follows. A producer pre-processes a file to produce a manifest [33]. The consumer then requests this root manifest and uses it to recursively resolve the rest of the content object chunks that make up the file. The exact fetching procedure is outlined in [33]. The consumer uses a simple stop-and-wait protocol when issuing interests and retrieving content objects. (This is because there was no transport protocol to use for the experiment.) This is not a requirement for CCNxKE, however, since the outer name for each interest uniquely identifies the state needed to decrypt the packet. In other words, the producer can handle out-of-order outer interests. In the secure session variant of this application, the consumer establishes a session with the producer and then issues all of its normal interests over this session. No other changes were made.

Our experiments indicate that the key exchange part of the application takes an average of 2.127ms with a standard deviation of 0.474ms. This exchange includes the round one BareHello message, the round two key derivation step wherein the producer verifies the cookie and, upon validation, generates and returns a key share for the consumer, and the subsequent consumer traffic secret generation step.

Figure 3 shows the cryptographic overhead at the consumer. (The same data, except inverted, was observed at the producer.) We observe that interest encryption is an efficient process that takes at most 5us to complete (since interests are small), whereas decryption varies based on the content object chunk size. In general, we observed that the decryption was bounded by approximately 30us. The encryption and decryption steps had an obvious effect on the data transfer time. Figure 4 shows the percentage increase in transfer time (in microseconds). Our results indicate that this increased the time to transfer the file by at most 50% and on average by approximately 30%.[10]

[9]The topology was kept small so as to amplify the computational overhead incurred by the use of CCNxKE. Moreover, forwarders have no role in CCNxKE, so we wanted to minimize their role in the experiment.

[10]This number could be substantially reduced with a better implementation. Problem areas in the current code include packet encoding and decoding.

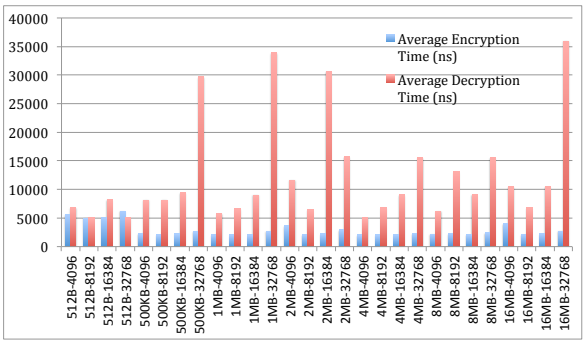

Fig. 3. Packet encryption and decryption overhead at the client. Each x-axis label indicates the size of the file transferred and the producer chunk size used to create the manifest, e.g., 500KB-4096 indicates a 500KB file transferred with a 4096B chunk size.

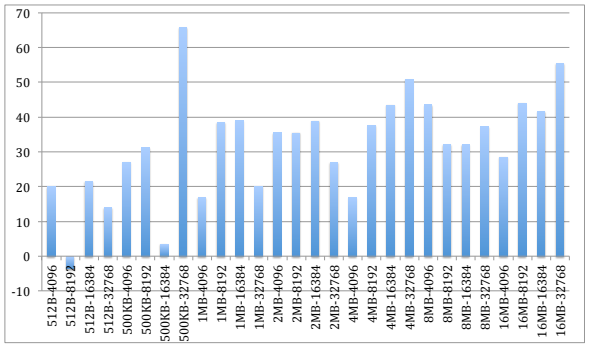

Fig. 4. Data transfer percentage increase

VIII. Conclusion

We presented CCNxKE, the first session-centric key exchange protocol that enables forward secure communication between a consumer and service producer. We described its design and discussed how it can be used to encrypt CCN packets (interests and content objects). We also compared the protocol to modern TCP/IP protocols, presented an expose of use cases for secure sessions in CCN, and conducted a preliminary performance assessment. Given the clear need for secure sessions for certain applications, this work is both timely and important in formally defining a protocol optimized for a prominent ICN architecture to achieve it. For future work, we plan to integrate support for multiple encryption contexts in a given CCNxKE session and integrate consumer deniability as a core feature of the protocol.

References

[1] M. Backes, G. Doychev, and B. Köpf, "Preventing side-channel leaks in web traffic: A formal approach." in *NDSS*, 2013.

[2] C. Ghali, G. Tsudik, and C. A. Wood, "(the futility of) data privacy in content-centric networking," in *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*. ACM, 2016, pp. 143–152.

[3] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-tls13-18, Oct. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13-18

[4] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk, "Quic: A udp-based secure and reliable transport for http/2," *IETF, draft-tsvwg-quic-protocol-02*, 2016.

[5] A. Langley and W. Chang, "Quic crypto," 2014.

[6] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," Internet Requests for Comments, RFC Editor, RFC 6347, January 2012.

[7] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Advances in Cryptology EUROCRYPT*. Springer, 2000.

[8] B. C. Neuman and T. Ts' O, "Kerberos: An authentication service for computer networks," *IEEE Communications Magazine*, vol. 32, no. 9, 1994.

[9] M. Mosko and C. A. Wood, "Secure Off-Path Replication in Content-Centric Networks," in *IEEE ICC 2017 Next Generation Networking and Internet Symposium (NGNI 2017)*. IEEE, 2017.

[10] PARC, "Ccnx distillery," https://github.com/parc/CCNx_Distillery, 2016.

[11] V. Jacobson *et al.*, "Networking named content," in *CoNext*, 2009.

[12] M. Mosko, I. Solis, and C. Wood, "CCNx semantics," *IRTF Draft, Palo Alto Research Center, Inc*, 2016.

[13] D. K. Smetters, P. Golle, and J. D. Thornton, "CCNx access control specifications," PARC, Tech. Rep., Jul. 2010.

[14] S. Misra, R. Tourani, and N. E. Majd, "Secure content delivery in information-centric networks: Design, implementation, and analyses," in *ICN*, 2013.

[15] M. Ion, J. Zhang, and E. M. Schooler, "Toward content-centric privacy in ICN: Attribute-based encryption and routing," in *ICN*, 2013.

[16] C. A. Wood and E. Uzun, "Flexible end-to-end content security in CCN," in *CCNC*, 2014.

[17] J. Kurihara, C. Wood, and E. Uzuin, "An encryption-based access control framework for content-centric networking," *IFIP*, 2015.

[18] Y. Yu, A. Afanasyev, and L. Zhang, "Name-based access control," *Named Data Networking Project, Technical Report NDN-0034*, 2015.

[19] C. Ghali, M. A. Schlosberg, G. Tsudik, and C. A. Wood, "Interest-based access control for content centric networks," in *International Conference on Information-Centric Networking*. ACM, 2015.

[20] W. Shang *et al.*, "Ndn-ace: Access control for constrained environments over named data networking."

[21] D. Adrian *et al.*, "Imperfect forward secrecy: How diffie-hellman fails in practice," in *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[22] S. Kent, "Ip encapsulating security payload (esp)," Internet Requests for Comments, RFC Editor, RFC 4303, December 2005.

[23] H. Krawczyk, "Sigma: The sign-and-macapproach to authenticated diffie-hellman and its use in the ike protocols," in *Annual International Cryptology Conference*. Springer, 2003, pp. 400–425.

[24] Limited Used of Remote Keys (LURK), Tech. Rep. [Online]. Available: https://datatracker.ietf.org/wg/lurk/charter/

[25] J. Salowey, A. Choudhury, and D. McGrew, "Aes galois counter mode (gcm) cipher suites for tls," RFC 5288 (Proposed Standard), Tech. Rep., 2008.

[26] H. Krawczyk, "Cryptographic extraction and key derivation: The hkdf scheme," in *Annual Cryptology Conference*. Springer, 2010, pp. 631–648.

[27] M. Mosko, "CCNx Messages in TLV Format," Internet Engineering Task Force, Internet-Draft draft-irtf-icnrg-ccnxmessages-02, Apr. 2016.

[28] J. Rizzo and T. Duong, "The crime attack," in *EKOparty Security Conference*, vol. 2012, 2012.

[29] H. Krawczyk and H. Wee, "The optls protocol and tls 1.3," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 81–96.

[30] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2001, pp. 453–474.

[31] M. Thomson and S. Turner, "Using Transport Layer Security (TLS) to Secure QUIC," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-tls-02, Mar. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-tls-02

[32] D. J. Bernstein, "Syn cookies," 1996.

[33] C. A. Wood and C. Tschudin, "File-Like ICN Collection (FLIC)," Internet Engineering Task Force, Internet-Draft draft-tschudin-icnrg-flic-02, Jan. 2017, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-tschudin-icnrg-flic-02