An Encryption-Based Access Control Framework for Content-Centric Networking

Jun Kurihara[†]
KDDI R&D Labs., Inc
kurihara@ieee.org

Ersin Uzun
Palo Alto Research Center
ersin.uzun@parc.com

Christopher A. Wood
Palo Alto Research Center & UC Irvine
woodc1@uci.edu

***Abstract*—This paper proposes a comprehensive encryption-based access control framework for content centric networking (CCN), called *CCN-AC*. This framework is both flexible and extensible, enabling the specification, implementation, and enforcement of a variety of access control policies for sensitive content in the network. The design of CCN-AC heavily relies on the concept of secure content object manifests and leverages them to decouple encrypted content from access policy and specifications for minimum communication overhead and maximum utilization of in-network caches. To demonstrate the flexibility of framework, we also describe how to implement two sample access control schemes, group-based access control and broadcast access control, within CCN-AC framework.**

I. Introduction

Information-centric networking architectures (ICNs) [8], [13] are quickly becoming an attractive alternative to the current host-to-host Internet design in both research and industrial communities. Several novel networking architectures [2], [5]–[7], [19], [22], [30], [31] have been recently proposed as instances of the ICN. The most common and fundamental features in these ICN instances are: (1) *interest-based content retrieval*, (2) *content oriented naming and routing at the network layer*, and (3) *in-network caching*. Features (1) and (2) imply that users acquire content *from the network* via explicit queries for *uniquely named* content, rather than by establishing point-to-point connections between endpoints. In-network caching permits a router to cache any content for predetermined lengths of time such that subsequent requests for the same content can be satisfied from the cache, rather than by forwarding the interest upstream. These architectural features enable many foreseeable benefits such as improved performance [28] and lower network cost [32].

Due to in-network caching, content objects may not always arrive from their original producer(s). Consequently, the content security – defined with respect to confidentiality and authenticity – cannot be considered in the traditional Internet model based on secure point-to-point channels. This implies that content must be encrypted so as to prevent invalid disclosure or modification within the network by unauthorized parties [34], [36].

Several encryption-based access control schemes in ICNs have already been proposed [18], [26], [29], [37] (We will summarize them in Section VI), with substantial differences in each design. Rather than require each producer application to implement their own form of application-specific access control specification and enforcement mechanisms, we advocate a unified framework by which *any* access control scheme may be constructed. In this paper, we present such a framework for the latest revision of *content-centric networking* (CCN) architecture, CCN 1.0 [1]. The framework is called *CCN-AC* (CCN access control) and it is the first of its kind for ICN architectures.

In Section III, we describe the basic design of the CCN-AC and how the CCN-AC works based on content object *manifests* [30] (we will briefly explain manifests in Section II). We show how manifests decouple encrypted content from the access control and decryption information via *access control specifications* (ACS) and *key-chains*. Key-chains are themselves manifests containing a list of decryption keys necessary to access content protected under arbitrary group-based hierarchies. The keys themselves are based on *principals* in order to realize fine-grained access control schemes, where a principal is an individual user or a group of principals.

In Section IV, we demonstrate the detailed configurations of the CCN-AC framework that realize two typical types of access control applications over CCN: group-based access control [29] and broadcast access control [12], [27], [33]. Other potential extensions of CCN-AC, along with relevant security claims and arguments, are presented in Section V. Although our presentation in this paper is tailored towards CCN, we note that our framework is compatible with any ICN design that supports manifests and secure links or constructions of equal functionality.

The rest of this paper is organized as follows: Section II gives a brief introduction of the latest design of CCN (CCN 1.0) [1]. Section III describes our comprehensive framework of access control in CCN based on manifests - CCN-AC. Section IV presents some specific instances of access control scheme realized over CCN-AC. Section V gives discussions on CCN-AC and analyzes its security. Finally, Section VI summarizes existing studies related to access control for ICNs, and Section VII concludes this paper.

II. CCN Overview and Manifest-Based Content Retrieval

In this section, we first introduce *content-centric networking* (CCN) [19], [30]. We then explain its new content retrieval

[†] This work has been done while the author was at PARC.

ISBN 978-3-901882-68-5 © 2015 IFIP

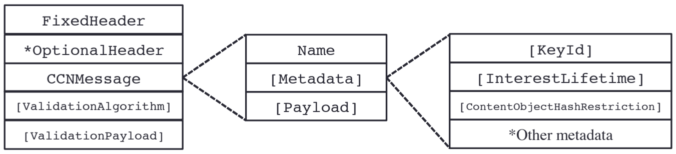

Fig. 1. CCN 1.0 interest message format: [·] means an optional field and ∗· means zero or more repeated fields

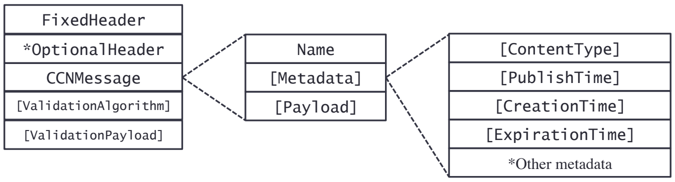

Fig. 2. CCN 1.0 content object message format: [·] means an optional field and ∗· means zero or more repeated fields

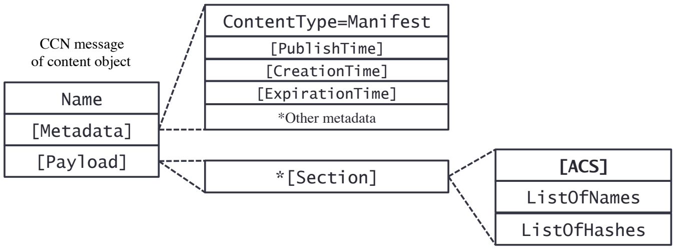

Fig. 3. CCN 1.0 manifest format: [·] means an optional field, ∗· means zero or more repeated fields, and ·|· means "or"

method based on *manifests*.

A. Content-Centric Networking (CCN) Architectural Elements

Content-centric networking (CCN) [19] is an instance of ICN, and CCN 1.0 [30] is the latest protocol design of CCN. There are four basic parties in CCN: (1) *content producers*, (2) *content publishers*, (3) *content consumers*, and (4) *routers*. For the sake of simplicity, we may refer to these parties as producers, publishers, consumers, and routers, respectively.

Content producers are responsible for generating content, e.g., photos, sensor-collecting data, etc. Publishers convert data from producers to named data objects with desired security bindings and protections, and publishes them on the network. Routers are responsible for forwarding requests for data objects and corresponding responses through the network between the consumers to the publishers.

Routers are composed of three primary elements: (1) a *forwarding information base* (FIB), (2) *pending interest table* (PIT), and (3) *content store* (CS). The FIB is used to route incoming interests to the appropriate output port towards the desired content producer. Much like traditional IP routing tables, the FIB is populated using standard routing protocols or static routes and matches content names in interest packets to FIB entries using longest prefix match. The PIT serves as a cache of interest state such that content objects that satisfy interests may follow the reverse interest path back to the requester. This preserves upstream and downstream network flow. Finally, the CS is an optional cache for content objects that, if present, is first searched prior to forwarding an interest upstream. These caches serve to reduce content object retrieval latency and bandwidth consumption in the network. A routing protocol is responsible for asynchronously populating the FIB based on the most probable locations of published data objects.

1) Network Messages in CCN 1.0: CCN 1.0 supports a two main kinds of network messages: interests and content objects, and a number of well-defined content object types that are understood and interpreted at the network layer such as manifests. Each network message in CCN 1.0 consists of three components: (1) a *message header* (consisting of FixedHeader and OptionalHeader fields), (2) *CCN message* (CCNMessage field), and (3) *validation data* (consisting of ValidationAlgorithm and ValidationPayload fields). The message headers provide information about the message structures and forwarding information. The CCN message is the message body, which includes metadata and the actual payload (for both interest and content object). Lastly, the validation data field is an *optional* field containing the information necessary to validate the authenticity and/or the integrity of the payload (e.g., the signature algorithm, public key and the actual signature bytes).

A content object is a named data object that is requested and located in the network via its name. An interest is a message issued by a consumer to request a content object with a particular name. Interests are composed of the name of the desired content, optional metadata and optional payload. Figures 1 and 2 briefly describe the format of interest and content objects, respectively.

The Metadata field of interest, especially its KeyId and ContentObjectHash subfields, in CCN 1.0 allows interest messages to carry additional restrictions that are used to determine which content objects may satisfy (match) the interests. The KeyId limits the match to a specific publisher by checking the key ID (i.e., the cryptographic hash digest of the public key) used to sign the content object. The ContentObjectHash subfield will only allow the network to return a content object whose cryptographic digest equals to the indicated value, where the digest is computed over the CCNMessage, ValidationAlgorithm and ValidationPayload (shown in Figure 2). Note that an interest message with a non-empty ContentObjectHash would uniquely point to one content object (i.e., within the limitations of the collusion space of the hash function).

We also emphasize that interest messages are allowed to carry additional information within the payload field of the network message. This payload information is not stored in the PITs of routers, as it is only used by producers to generate dynamic content to be distributed by a publisher.

The manifests are one of the recent enhancements in the CCN protocol that was introduced to efficiently handle fragmentation and distribution of large content. As previously

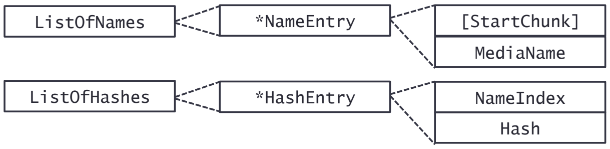

Fig. 4. The fields of list of names and list of hashes in CCN 1.0 manifest

mentioned, the manifest is defined as a type of content object. Conceptually, a manifest is used to describe a logical collection of a set of content objects. This is done by providing meta information about the collection and enumerating the ordered pointers to every constituent piece of the collection. This format enables consumers to issue interests for all of the content objects enumerated within a manifest.

Figure 3 shows the message format of manifest[1]. In the `Payload`, the meta information and ordered names are given in the format of `Section` that consists of `ACS` (access control specification), `ListOfNames` and `ListOfHashes`. The `ListOfNames` and `ListOfHashes` fields jointly give secure and ordered pointers to constituent content objects. If an ACS field exists in the section, it gives all of the access control information applicable to the listed content objects in the same section. We emphasize that this paper defines the specification of ACS, upon which the CCN-AC framework is heavily based.

B. Manifest-Based Content Retrieval

In this section we briefly describe how to handle the (non access-controlled) large content through the manifest and explain the merits of its usage. Since we are not yet dealing with access control specifications or semantics, assume that manifests here do not have ACS in any of its sections.

Figure 4 shows the `ListOfNames` and `ListOfHashes` fields in the payload of manifest. The `ListOfNames` field enumerates name entries that indicate the content name prefix in the `MediaName` subfield and its first chunk number in the `StartChunk` subfield. In the `ListOfHashes`, each hash entry indicates a content object by a number in the `Index` subfield and the hash value of its CCN message in the `Hash` subfield, where the number in the `Index` subfield corresponds to an entry of `ListOfNames`. To illustrate this binding, consider the case where the first entry of the `ListOfNames` is

`{StartChunk = 3, MediaName = lci:/parc/obj},`

and the first and second entries of the list of hashes are

`{NameIndex = 1, Hash = 0x123},`

`{NameIndex = 1, Hash = 0xABC},`

respectively. Then, the first entry of the `ListOfHashes` represents a content object with name `lci:/parc/obj/chunk=3` and hash value `0x123`, and the second entry represents one with the name `lci:/parc/obj/chunk=4` and hash value `0xABC`.

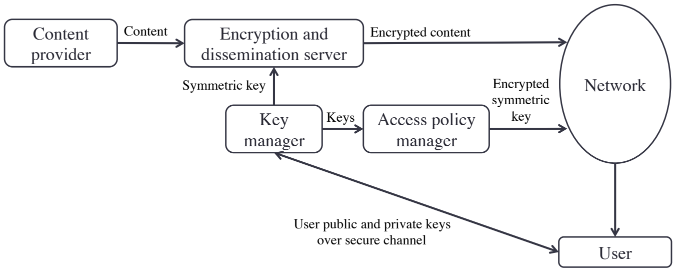

Fig. 5. An example distributed content-dissemination system that leverages the CCN-AC design.

With manifests, a consumer first sends an interest for a content objects and retrieves a manifest as the response. The manifest is then parsed to obtain content object names and hash values as shown in the above example. Using these names and hash values, interest messages are created and issued to the network sequentially or simultaneously to obtain all content objects encapsulated by the manifest. Note that this general content retrieval strategy is always followed even if an ACS exists in a section of manifest.

Aside from the ease of simultaneously requesting chunks of fragmented objects, manifests are useful to circumvent the need to individually sign and verify all content objects in CCN. Since each component content object is obtained via an interest that restricts the responses to a particular object by providing its cryptographic digest, a manifest can serve as a tool for batch signing/verification and only the signature of the manifest needs to be verified for any set of content objects a manifest provides the hashes, i.e, cryptographic fingerprints, for. This benefit can be exploited by creating and leveraging manifest trees, in which the name of a component content object within a manifest is itself the name of another manifest. If the signature of the root manifest is verified and the associated verification key is trusted, then all content objects encapsulated by said manifest are also trusted.

III. A General Manifest-Based Access Control Framework

A. Design Goals

One of the primary goals of this work is to design a comprehensive and scalable framework for encryption-based access control in CCN. The framework should be flexible enough to support arbitrary access control policies enforced by appropriate cryptographic algorithms. We aim to make CCN-AC flexible and scalable enough to support an arbitrary number of consumers for any given producer.

To meet these requirements, consider a distributed system design similar to the one shown in Figure 5. In this example, the key manager uses a symmetric key for content encryption and an asymmetric public key to encrypt the symmetric key. Additionally, a user obtains his public-private key pair (or help the producer to learn and authenticate its public key) through a secure channel to the producer. The content provider then generates content, either on-demand or by reading data from persistent storage, and sends it to the encryption and

[1] Note that the manifest type is indicated by the `ContentType` subfield of the `Metadata` field.

dissemination server for encryption under the symmetric key and making it available in the network. The access policy manager encrypts the symmetric key under the user's public key, and publishes the encrypted symmetric key over the network as well. Finally, the user obtains the encrypted content and symmetric key, decrypts the symmetric key, and then uses the symmetric key to decrypt the content. This example consists of these five reasonable entities working in a decentralized manner; they could be collocated on a single machine or distributed across different places within a network.

With regards to the underlying cryptographic algorithms used to enforce access control, we also aim to support arbitrary types of encryption that are appropriate for different access control policies. For example, we leverage attribute-based encryption [10], [14] for attribute-based access control. Another goal we seek to achieve is maximized usage of in-network caches so as to increase the efficiency of encrypted content retrieval and access. In other words, we aim to maintain the efficiency of the CCN content retrieval in terms of the computational and communication overhead, even if the content is confidential and needs to be access-controlled.

B. Basic Access Control Overview

Cryptographic access control in CCN-AC is based on hybrid encryption of content data. Specifically, content objects are encrypted under a cryptographically random symmetric key, called the *nonce key*. The nonce key is then encrypted under another encryption algorithm that is appropriate for the desired form of access control (e.g., broadcast encryption for group-based access control). Nonce keys are encrypted for *principals*, which can be groups of or individual consumers. Technically, a principal is defined as an individual consumer or an *arbitrary* group of principals. Each consumer is treated as a user of the access control system; consequently, we use the terms interchangeably without loss of generality in the remainder of this paper.

Recall the manifest-based content retrieval described in Section II-B. In CCN-AC, access-controlled content objects that are specified in one manifest are encrypted under a *nonce key* by a certain encryption algorithm, e.g., AES-256-CTR. The names and hashes of the encrypted content objects are given in the ListOfNames and ListOfHashes fields of the manifest as the normal content objects. Hence, once a user obtains the manifest, he can issue interests for all the encrypted content objects sequentially or simultaneously. The ACS field of the manifest provides the required information to decrypt the encrypted content objects; specifically, the name of the encrypted nonce key, parameters of encryption algorithms, and any other information necessary for successful decryption by authorized consumers. We will explain the structure of the ACS field in Section III-C.

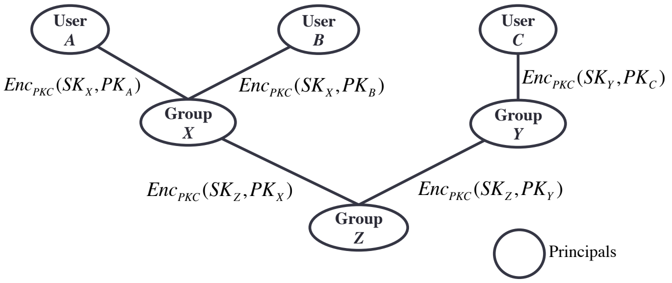

Fig. 6. Principal hierarchy and encryption of principals' private keys, where Enc_{PKC} is the function of encryption via a certain public key cryptographic algorithm, PK_i is the public key of principal i, and SK_i is the private key of i.

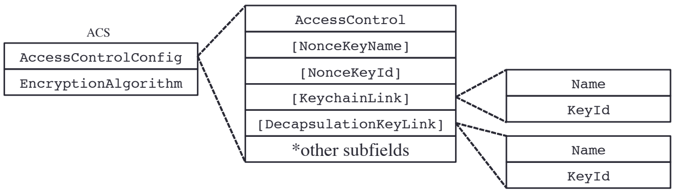

Fig. 7. Description of access control specification (ACS) field in a manifest: [·] means an optional field and $*\cdot$ means zero or more repeated fields.

CCN-AC encrypts the nonce key used in the encryption of content objects by another encryption algorithm that is appropriate to realize the desired access control structure, e.g., broadcast encryption, attribute-based encryption, session-based encryption, etc. We refer to the key required to decrypt the encrypted nonce key as the *decapsulation key*. The decapsulation key itself is encrypted such that only qualified and authorized principals can decrypt the decapsulation key.

Consider the following example illustrated in Figure 6. First, each principal i is assigned a public-private key pair (PK_i, SK_i). Individual users initially have their own principal key pairs. We denote by $Enc_{PKC}(SK_i, PK_j)$ the encryption of SK_i by some public key cryptographic algorithm under PK_j. In this example, when the principal group X is authorized for access, the decapsulation key DK_X to X is encrypted as $Enc_{PKC}(DK_X, PK_X)$. User A can then obtain the decapsulation key DK_X by retrieving the ciphertexts $Enc_{PKC}(SK_X, PK_A)$ and $Enc_{PKC}(DK_X, PK_X)$ using only their own private key SK_A.

As we see, the decapsulation of nonce key and decryption of principal key along with the principal tree are essentially separated. This separation implies that the private key set is flexible enough to enable sophisticated access control group hierarchies for any principal(s). We expand on the private key specification and retrieval procedures in Sections III-C and III-D, respectively.

C. Access Control Specification in Manifest

In this section, we provide a detailed description of the *access control specification* (ACS) field of the manifest. Functionally, it contains all of the information needed to decrypt the nonce key to obtain access to encrypted content objects. Figure 7 briefly describes the ACS field in the manifest, composed of two subfields: (1) the EncryptionAlgorithm and (2) the AccessControlConfig (ACC).

The EncryptionAlgorithm field simply describes an encryption algorithm and its parameters that are used to encrypt the content objects listed in the manifest[2]. The field value is encoded in JSON [3], e.g.,

{{encryptionAlgorithm:AES-128-CTR, initialCounter:***}}.

The AccessControlConfig (ACC) field provides all the information required to execute the decryption algorithm specified in the EncryptionAlgorithm field. Unlike the EncryptionAlgorithm, the ACC has a nested structure and consists of four basic subfields: (1) AccessControl, (2) NonceKeyName, (3) NonceKeyId, and (4) KeychainLink. It can have other additional subfields depending on the access control instance, as shown in Figure 7.

The AccessControlConfig field explains the type of access control scheme that handles the symmetric key used in the EncryptionAlgorithm field in ACS. It is encoded in JSON, e.g.,

{type:NonceKey,
 encapsulationAlgorithm:Naor-Pinkas-BE, ...},

where type: NonceKey means that access control to the content objects is done through the encapsulation of the nonce key, and encapsulationAlgorithm: Naor-Pinkas-BE implies that the Noar-Pinkas broadcast encryption algorithm given in [27] is used to encapsulate the nonce key. The type is usually NonceKey in CCN-AC. It can however take different values in order to realize different types of access policies[3]. Decryption process of the content objects listed in a manifest starts by first referring to this field to determine which type of cryptographic primitives are required.

The NonceKeyName field of the ACC simply provides the (interest) name of the encapsulated nonce key. This name is used to issue an interest for the decapsulation key(s). We note that there exist cases where the nonce key name field has *aliases*. Aliases are variables of name components and introduced to represent state-dependent names in one name field. For example,

lci:/parc/obj/noncekey/<principal_public_key_hash>,

where <principal_public_key_hash> is an alias of a principal-specific value, the hash value of the qualified principal's public key[4]. By setting aliases in the nonce key name field, one can query the nonce key encrypted with principal-specific values, where the required types of principal-specific values are described by the aliases themselves and/or the AccessControl field of the ACC.

The NonceKeyId field of the ACC equals the hash value of the (non-encrypted) nonce key. This enables one to easily verify the integrity of the decrypted nonce key; with a proper collision-resistant hash function (e.g., SHA-256), integrity and unforgeability are guaranteed since the hash value is covered by the manifest's digital signature.

The KeychainLink field of the ACC provides the link to another manifest called a *key-chain*, where the link consists of its name and signer's key ID (KeyId to the content object of key-chain). The key-chain can be viewed as an ordered list of all the (encrypted) principal private keys required to decrypt the decapsulation key. Since the key-chain is itself a manifest, one can retrieve all required keys for the decryption of the decapsulation key in parallel by parsing the embedded name/hash lists. We defer a detailed description of the key-chain manifest to Section III-D. Just as with the NonceKeyName, the name of KeyChainLink usually has aliases to provide a key-chain personalized to each user. In the ACS field of the key-chain manifest, the specific algorithm for each of the principal key-chain hierarchy nodes is specified. For example, if a key-chain contains three keys, then the ACS of the key-chain has three ordered encapsulationAlgorithm fields in the EncryptionAlgorithm field.

The DecapsulationKeyLink field of the ACC provides the link to the encrypted decapsulation key, where the link consists of its name and its signer's key ID (KeyId to the content object of the decapsulation key), and the name usually has the aliases, just as the nonce key name, to assign a decapsulation key personalized to each qualified principal. Note that the key-chain provides the name of the decapsulation key to the principal. However, we do not need to retrieve the key-chain when a user already stores the decapsulation key. The purpose of this field is just to check if we need to fetch the key-chain to recover the decapsulation key, which is used to successively obtain the nonce key, by using the name and the signer's key ID of the decapsulation key.

D. Key-Chain to Retrieve All the Required Keys

As explained in Section III-B, our framework provides principal-level access control to content objects, and each user needs obtain his ancestors' private keys (leading to the decapsulation key) by recursively traversing the principal tree from their respective leaf to the root. In order to efficiently fetch these (encrypted) keys without $O(\mathsf{pathlength})$ round trip messages [29], we introduce a new key retrieval method using the *key-chain*.

The key-chain is a manifest that provides an ordered list of the private keys required for a user (principal). Although the key-chain is itself a normal manifest, it has only one ordered list of encrypted keys and has the following values in the fields of its ACS.

- AccessControl of ACC: {type: Keychain},
- EncryptionAlgorithm: The encryption algorithm with parameters that are used to encrypt keys listed in the manifest, e.g., RSA-1024. This can be repeated for each different algorithm applied to each listed key.

The ListOfNames fields provides the ordered list of names of encrypted (decryption) keys required for a user.

We now provide an example of how to use the key-chain to decrypt the nonce key given the principal hierarchy shown

[2] Note that this field can also specify multiple algorithms. In fact, when we apply CCN-AC to instantiate TLS-like one-to-one secure communication between a producer and consumer, this field describes multiple encryption algorithms and their parameters in JSON format.

[3] In fact, we use the value type: SecureSession to realize TLS-like one-to-one secure communication over CCN-AC.

[4] Any other valid principal value may be used in lieu of the public key hash.

in Figure 6. Consider the situation where the group Z is qualified to access content objects listed in a manifest and user B tries to access the content objects by retrieving all the keys. Then, the user needs to retrieve the encrypted private keys (decryption keys) of his ancestors, $Enc_{PKC}(SK_X, PK_B)$ and $Enc_{PKC}(SK_Z, PK_X)$. User B also needs the encrypted decapsulation key assigned to the principal Z, $Enc_{PKC}(DK_Z, PK_Z)$. To retrieve these keys, user B first issues interests for the key-chain personalized to him by referring to `KeychainLink` of the ACC. The name of the key-chain will typically have aliases of a user-specific value, e.g.,

`lci:/parc/obj/key-chain/<user_public_key_hash>`.

As the response to the interest, user B obtains the key-chain with the ordered key list in its `ListOfNames` fields as

{`StartChunk=1`, `MediaName`=name of $Enc_{PKC}(SK_X, PK_B)$}

{`StartChunk=1`, `MediaName`=name of $Enc_{PKC}(SK_Z, PK_X)$}

{`StartChunk=1`, `MediaName`=name of $Enc_{PKC}(DK_Z, PK_Z)$},

where the `ListOfHashes` field enumerates the hash values of their (chunked) content objects. At this point, user B needs to retrieve the content objects of encrypted keys and sequentially decrypt them, starting with the leaf. Note that if the user's private key can be used to decrypt the nonce key, then the list of keys in the key-chain will be empty.

Note that in a group-based access control setting where a principal hierarchy exists, the encapsulation algorithm of any decapsulation key is *not* necessarily coupled to the algorithms used to encrypt other keys in the key-chain. In other words, RSA may be used to encrypt all non-root nodes in the key-chain, and a different algorithm may be used to decrypt the root key. This means that different encryption algorithms may be applied at each level in the root-to-leaf path given in the key-chain. The ACS specifies which algorithm applies to each level in the tree so that consumers may apply the appropriate decryption algorithm when traversing the path. This decoupling allows *any* group-based access control policy to be imposed over a group-based principal hierarchy.

The integrity and unforgeability of the listed keys in a key-chain can be verified and guaranteed as long as the key-chain has a valid signature. Also, since the user is given the set of all decryption keys necessary to decrypt the nonce key, he can issue an interest for each key in parallel, thereby maximizing his usage of the available network bandwidth.

IV. Instances Realized over CCN-AC

With the foundation of our framework in place, we now show how to instantiate and enforce common access control policies over CCN. We first present an analog to the *group-based access control* that was implemented in the old version of the CCN protocol [29]. We then show how to instantiate *broadcast access control* over CCN by implementing *broadcast encryption* [12], [27], [33] in CCN-AC.

A. Group-based Access Control

1) Access Control Model: Consider the following content dissemination system to be used to enforce group-based access control in CCN-AC. Assume that there exists one content publisher, e.g., Netflix. Also assume a network topology in which all published content flows from the content publisher to individual consumers by traversing core-network and Internet service provider nodes (CCN routers). Assume that any node may cache content as it flows from the producer to the consumer. Furthermore, assume that each user of the Netflix-like service is already assigned their public-private key pair to be used when traversing the principal hierarchy for key-chains. These keys may be distributed offline or over a secure-session. Note that this system model can be easily extended to multiple publisher cases.

To begin, the content publisher encrypts and publishes named content objects to the network. The random nonce key used to encrypt each content object is then encrypted (encapsulated). The content publisher in the group-based access control setting allows certain groups of users, i.e., principals, to access the content simply by generating multiple (different) ciphertexts of the nonce key encrypted for different principal groups. Then, each ciphertext can be decrypted only by qualified principals in each group using the key-chain retrieval method described in Section III-D. This means that the qualified principals' public keys are directly used to encrypt the nonce key and hence the decapsulation keys are qualified principals' private keys.

As explained in Section III, each user initiates the retrieval of a piece of content by issuing an interest to the manifest with the name associated with the content. Upon receiving and parsing this manifest, the user unambiguously obtains all the encrypted content objects using the names and the hash values listed in it, and can decrypt them according to the ACS in the manifest after fetching the key-chain and all of the required keys. The next subsection gives the detailed description of the ACS in this manifest.

2) Configuration of ACS in the Manifest: For the sake of simplicity, we assume that the name of content objects generated by the content publisher in this instance have prefix `lci:/parc/GE/`. Also suppose that RSA encryption with a 1024-bit key is used as the encryption and encapsulation algorithm of keys. Using the example ACC subfield in Table I, we shall explain the configuration of the ACC of the ACS in the manifest by using these prefix and assumptions in each of the respective fields.

- `AccessControl`: This field may include any parameters related to the encapsulation algorithm.
- `NonceKeyName`: The name of the nonce key encrypted under the qualified principals' public keys, including the alias that is specific for each qualified principal.
- `NonceKeyId`: The hash value of the (non-encrypted) nonce key.
- `KeychainLink`: The link to the key-chain.
  - `Name`: The name of the key-chain including the alias

TABLE I
An Example ACC of a Group-Based Access Control ACS

AccessControl	= { type:NonceKey, encapsulationAlgorithm:RSA-1024,...}
NonceKeyName	= lci:/parc/GE/NonceKey/<principal_private_key_hash>
NonceKeyId	= 0x********
KeychainLink	{ Name = lci:/parc/GE/Keychain/<user_private_key_hash> { KeyId = 0x********
DecapsulationKeyLink	{ Name = lci:/parc/GE/PrincipalPrivateKey/<principal_private_key_hash>/<member_private_key_hash> { KeyId = 0x********

of the user-specific value.

  - KeyId: The signer's key identifier for key-chains.

- DecapsulationKeyLink: The link to the encrypted decapsulation key.
  - Name: The name of the decapsulation key, i.e., qualified principal's private key, encrypted under its member's public key, including the aliases that are specific for the qualified principal and its members.
  - KeyId: The signer's key identifier for encrypted decapsulation keys.

Note that the DecapsulationKeyLink field of the ACC in the ACS can be blank when it is capable of always retrieving the key-chain without checking if the user's key store has the decapsulation key for access control.

Also, the EncryptionAlgorithm field of ACS describes the encryption algorithm used to encrypt the content objects listed in the manifest, such as:

$$\texttt{EncryptionAlgorithm}=\{\texttt{encryptionAlgorithm:AES-128-CTR}, \\ \texttt{initialCounter:0x********},\ \ldots\}. \tag{1}$$

As long as all the encrypted content objects are specified in the lists of the manifest and in the ACS, a user belonging to the qualified principal can retrieve and decrypt them correctly.

B. Broadcast Access Control

1) Access Control Model: As in the model described in Section IV-A1, we assume a simple centralized setup in the broadcast access control model in which all network nodes are CCN routers and there exists one content publisher who encrypts and disseminates content objects over the network. Also suppose that each user is already assigned his public-private key pair in the principal hierarchies, generated by the content publisher.

The content publisher qualifies a certain set of principals to access his content objects by encrypting the content objects under a random nonce key and then by encapsulating the nonce key with a group-specific (principal-specific) key using broadcast encryption. Each private key qualified to decrypt the decapsulation key is associated with a qualified principal.

After retrieving and analyzing the manifest including the ACS in this model, each user starts to access the content objects as we explained in Section III. The next subsection gives the detailed description of the ACS in this manifest.

2) Configuration of ACS in the Manifest: For the sake of simplicity, assume that the name of content objects generated by the content publisher in this instance has the prefix lci:/parc/BE/. Also suppose that the Naor-Pinkas broadcast encryption [27] is used as the encapsulation algorithm of the nonce key. We shall explain the configuration of the ACS in the manifest by using these prefix and assumptions in each of the respective fields.

Table II presents an configuration example of the AccessControlConfig ACC field in the broadcast access control under the assumed settings. In this instance of CCN-AC, the ACS in the manifest that each user first retrieves has these values in the ACC field. Note that the following fields in Table II are specifically different from those in the case of group-based access control described in Section IV-A2.

- AccessControl: This field may include any parameters related to the encryption of [27].
- NonceKeyName: The name of the nonce key encrypted by the broadcast encryption [27], where it is common to all qualified principals and therefore does not contain aliases.
- Name subfield of DecapsulationKeyLink: The name of the (encrypted) decapsulation key including the alias that is specific for each qualified principal.

Also, the EncryptionAlgorithm field of ACS has the description of the encryption algorithm used to encrypt the content objects listed in the manifest, similar to the encoding specified in (1).

Note that the configuration of the ACC is very flexible and one can easily demonstrate that the broadcast access control schemes given in [9], [10], [14], [16], [17], [20], [24], [35] and the role-based access control scheme of [39] can be directly realized over CCN-AC as an exercise.

V. Discussion

We now discuss several topics pertinent to CCN-AC. Section V-A briefly summarizes the security of CCN-AC, and Section V-B discuss about user (principal) revocation in CCN-AC. Lastly, Section V-C shows some potential extensions of our basic framework to realize other access control structures.

A. Security of CCN-AC

The security of correctly implemented CCN-AC can be considered as a combination of manifest-based content retrieval and that of the access control or encryption schemes utilized. Since CCN-AC invokes any access control procedure by using a properly-signed manifest, we can guarantee the integrity and unforgeability of any objects that are listed or linked with their names and hash values in the manifest as long as the

TABLE II
An Example ACC of a Broadcast Access Control ACS

AccessControl	= { type:NonceKey, encapsulationAlgorithm:Naor-Pinkas-BE, ... }
NonceKeyName	= lci:/parc/BE/NonceKey
NonceKeyId	= 0x********
KeychainLink	{ Name = lci:/parc/BE/Keychain/<user_private_key_hash> KeyId = 0x********
DecapsulationKeyLink	{ Name = lci:/parc/BE/PrincipalPrivateKey/<principal_private_key_hash> KeyId = 0x********

hash function used is secure. Moreover, although some objects linked from the manifest, e.g., objects personalized to each principal, cannot be described with their hash values, it is enforced that they have valid signatures that would be trusted by specifying a `KeyId` field in the interest and they are verified once received. Hence, we can guarantee the integrity and authenticity of all the objects, including encrypted content and decryption-related information, retrieved (recursively) through the manifest.

This implies that for an access control instance over CCN-AC, the only security consideration is that of the access control scheme and the encryption scheme (e.g., specific broadcast encryption algorithm like [12]). In other words, any access control scheme implemented in CCN-AC will be secure as long as the underlying encryption algorithms, protocols, the access control scheme and their implementations are secure.

B. Revocation in CCN-AC

Achieving both rapid access revocation and high cache utilization within a network is not easy. Even if the publisher changes his local content objects to revoke some principals from the access control structure, the revoked user might still be able to retrieve the older copies of content objects from various caches in the network. Since caching is a fundamental feature of CCN (and ICNs in general), whenever possible, we recommend using the *lazy revocation* approach in CCN-AC to make the most out of the caches in the network. For applications that require rapid revocation capability, CCN-AC can easily support session-based access control schemes, however at the expense of low cache utilization within the network.

In lazy revocation, when a principal is revoked, old content objects listed in a manifest are not required to be updated. Note that in CCN-AC, content objects are encrypted under a symmetric key that is unique and random for each manifest. Hence, as long as the revoked users cannot obtain valid (plaintext) decapsulation keys, they cannot access content objects that are listed in new manifests generated after the revocation. However, all the public-private key pairs of the revoked principal's ancestors in the principal hierarchy needs to be updated. Moreover, the decapsulation key have to be re-encrypted under the updated qualified principal's key. Note that as long as the key-chain is up-to-date, a user can easily check if his ancestors' private keys and decapsulation keys stored by him or cached nearby are valid or expired by their hash values given in the key-chain.

C. Potential Extensions

We claim that various types of encryption-based access control schemes can be easily realized in CCN by using CCN-AC, as shown in Section IV. For instance, the proxy re-encryption access control scheme proposed in [37] can be realized over CCN-AC by introducing some optional ACC subfields associated with proxy re-encryption [11], [15], [25]. Moreover, by using several additional fields in the ACS of a manifest, CCN-AC can be easily applied to interactive security protocols. In fact, TLS-like end-to-end secure communication can be realized over CCN-AC, as discussed in [23], with no changes to the framework; it only requires users and publishers to support the interactive protocol.

VI. Related Work

This section summarizes existing works related to our access control framework. There exist several existing approaches of specific access control schemes for ICNs. Smetter et al. [29] gave a group-based access control scheme as a default access control scheme based on the old version of the CCN software (CCNx 0.x) [1]. This corresponds to the instance of CCN-AC given in Section IV-A. Misra et al. [26] proposed a broadcast-based access control procedure in CCN that uses broadcast encryption [12], [27], [33] to encrypt the nonce key, which is exactly the instance of CCN-AC presented in Section IV-B. Ion et al. [18] gave an attribute-based access control in ICNs by applying an attribute-based encryption [10], [14], and proposed a routing scheme based on user's attributes. Wood et al. [37] presented an access control schemes using proxy re-encryption to personalize cached content objects to each user, as mentioned in Section V-C.

Each of these works are merely instances of access control in ICNs, and, to our knowledge, there has been no comprehensive access control framework for ICNs on which various access control schemes can work. Consequently, we designed CCN-AC in such a way that existing and potential ICN access control schemes can work as similar, simple instance of the framework.

Outside of information-centric networking, there have been many proposed access control frameworks. Recently, access control of content in shared cloud storage or social network services, e.g., Google Drive, Dropbox, Facebook, etc., is attracting many researchers [20], [35], [38], [39]. For instance, Kamara et al. [21] modeled encryption-based access control framework for cloud storage. Microsoft PlayReady

[4] is another popular access control framework for content dissemination over the Internet. Such services, along with social network services, consist of multiple entities similar to our distributed system design, e.g., a key manager, encryption server, storage server, client proxy, etc. Recall that the network in ICNs can be viewed as cloud storage from the producer, publisher, and consumer (user) perspective. Consequently, these ICN-agnostic access control frameworks inspired our design of CCN-AC.

VII. Conclusion

This paper described the design of a comprehensive access control framework for CCN, called CCN-AC. The design is heavily based on CCN-specific manifests, which enables maintaining the efficiency and security of standard content retrieval strategies using the new features in CCN 1.0. CCN-AC provides the much needed flexibility for implementing access control in CCN applications over a common framework, where various types of access control schemes can be easily implemented and enforced. To illustrate this flexibility, we showed how to realize two common access control schemes: (1) group-based access control, and (2) broadcast access control, using CCN-AC.

This paper presents our initial effort in designing a flexible and comprehensive access control framework for CCN. There exists many possible avenues for future work. For example, more sophisticated access control schemes that are based on interactive and non-interactive protocols for establishing secure communication and can be designed and instantiated using the framework. Similarly, we also believe that CCN-AC facilitates rapid prototyping and analysis of new security protocols in CCN, which is critical for an emerging networking protocol with such grand ambitions.

References

[1] "CCNx," http://ccnx.org/.

[2] "The FP7 4WARD project," http://www.4ward-project.eu/.

[3] "Introducing JSON," http://www.json.org/.

[4] "Microsoft PlayReady," http://www.microsoft.com/playready/.

[5] "Named-data networking," http://named-data.net/.

[6] "Pursuing a pub/sub internet (PURSUIT)," http://www.fp7-pursuit.eu/PursuitWeb/.

[7] "Scalable and adaptive internet solutions (SAIL)." http://www.sail-project.eu/.

[8] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 26–36, Jul. 2012.

[9] N. Attrapadung and H. Imai, "Conjunctive broadcast and attribute-based encryption," in *Proc. Paring 2009*, Aug. 2009, pp. 248–265.

[10] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE S&P 2007*, May 2007, pp. 321–334.

[11] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. EUROCRYPT 1998*, May–Jun. 1998, pp. 127–144.

[12] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Proc. CRYPTO 2005*, Aug. 2005, pp. 1–19.

[13] A. Ghodsi, S. Shenker, and T. Koponen, "Information-centric networking: seeing the forest for the trees," in *Proc. ACM HotNets 2011*, Nov. 2011, pp. 1–6.

[14] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. ACM CCS 2006*, Oct.–Nov. 2006, pp. 89–98.

[15] M. Green and G. Ateniese, "Identity-based proxy re-encryption," in *Proc. ACNS 2007*, Jun. 2007, pp. 288–306.

[16] J. Hur, "Improving security and efficiency in attribute-based data sharing," *IEEE Trans. Knowledge Data Eng.*, vol. 25, no. 10, pp. 2271–2282, Oct. 2013.

[17] J. Hur and D. K. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 7, pp. 1214–1221, Jul. 2011.

[18] M. Ion, J. Zhang, and E. M. Schooler, "Toward content-centric privacy in ICN: Attribute-based encryption and routing," in *Proc. ACM SIGCOMM ICN 2013*, Aug. 2013, pp. 39–40.

[19] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. ACM CoNEXT 2009*, Dec. 2009, pp. 1–12.

[20] S. Jahid, P. Mittal, and N. Borisov, "EASiER: Encryption-based access control in social networks with efficient revocation," in *Proc. ACM ASIACCS 2011*, Mar. 2011, pp. 411–415.

[21] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Proc. FC 2010*, Jan. 2010, pp. 136–149.

[22] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, Oct. 2007.

[23] J. Kurihara, C. A. Wood, and E. Uzun, "An encryption-based access control framework for content-centric networking (full version)," 2015, to appear on ArXiv.

[24] K. Liang, L. Fang, W. Susilo, and D. S. Wong, "A ciphertext-policy attribute-based proxy re-encryption with chosen-ciphertext security," in *Proc. INCoS 2013*, Sep. 2013, pp. 552–559.

[25] M. Mambo and E. Okamoto, "Proxy cryptosystems: Delegation of the power to decrypt ciphertexts," *IEICE Trans. Fundamentals*, vol. 80, no. 1, pp. 54–63, Jan. 1997.

[26] S. Misra, R. Tourani, and N. E. Majd, "Secure content delivery in information-centric networks: Design, implementation, and analyses," in *Proc. ACM SIGCOMM ICN 2013*, Aug. 2013, pp. 73–78.

[27] M. Naor and B. Pinkas, "Efficient trace and revoke schemes," in *Proc. FC 2000*, Feb. 2000, pp. 1–20.

[28] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and evaluation of CCN-caching trees," in *Proc. IFIP Networking 2011*, May 2011, pp. 78–91.

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

[30] I. Solis and G. Scott, "CCN 1.0 (tutorial)," in *ACM ICN 2014*, Sep. 2014.

[31] D. Trossen and G. Parisis, "Designing and realizing an information-centric internet," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 60–67, Jul. 2012.

[32] G. Tyson, S. Kaune, S. Miles, Y. El-khatib, A. Mauthe, and A. Taweel, "A trace-driven analysis of caching in content-centric networks," in *Proc. IEEE ICCCN 2012*, Jul.–Aug. 2012, pp. 1–7.

[33] W.-G. Tzeng and Z.-J. Tzeng, "A public-key traitor tracing scheme with revocation using dynamic shares," in *Proc. PKC 2001*, Feb. 2001, pp. 207–224.

[34] M. Walfish, H. Balakrishnan, and S. Shenker, "Untangling the web from DNS," in *Proc. USENIX NSDI 2004*, Mar. 2004, p. 17.

[35] G. Wang, Q. Liu, and J. Wu, "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services," in *Proc. ACM CCS 2010*, Oct. 2010, pp. 735–737.

[36] A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the scale and performance of cooperative web proxy caching," *ACM SIGOPS Oper. Syst. Rev.*, vol. 33, no. 5, pp. 16–31, Dec. 1999.

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

[38] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. IEEE INFOCOM 2010*, Mar. 2010, pp. 1–9.

[39] L. Zhou, V. Varadharajan, and M. Hitchens, "Achieving secure role-based access control on encrypted data in cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 12, pp. 1947–1960, Dec. 2013.