NDN, Technical Report NDN-0034, 2015. http://named-data.net/techreports.html
Revision 2: January 11, 2016. Adding comparison against Attribute Based Encryption
~~Revision 1: October 27, 2015~~. (http://named-data.net/wp-content/uploads/2015/11/ndn-0034-nac.pdf)

Name-Based Access Control

Yingdi Yu
UCLA
yingdi@cs.ucla.edu

Alexander Afanasyev
UCLA
afanasev@cs.ucla.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

This paper presents a content-based access control access control model for content stored in network storage. The model enforces the access control directly over content through encrypting content at the time of production, rather than relying on a third party (such as data storage) as traditional perimeter-based access control model. We present the design of Name-based Access Control (NAC), which implements the content-based access control model in Named Data Networking (NDN). We demonstrate how to make use of naming convention to explicitly convey access control policy and efficiently distribute access control keys, thus enabling effective access control. We evaluate the scalability of NAC against CCN-AC, another encryption-based access control scheme. The results suggest that NAC is more suitable for large scale distributed data production and consumption.

1. INTRODUCTION

Sharing private content among multiple parties requires end-to-end confidentiality, to ensure that no one but those authorized parties can see the shared content.

Due to a number of reasons (scalability, availability, economy, etc.), today's content sharing applications, by and large, rely on a third party to host their contents, and the security in content sharing is provided through encrypted channels. However these channels are not directly between content producers and consumers, but between producers and host, and host and consumers. This practice fails to provide end-to-end confidentiality because it allows a third party, the content host, to see the shared content in plain text. This failure not only causes the potential privacy concern, but also introduces liability on content hosts, or make them an attractive attack target since a single host may store content for a large number of customers.

To achieve true end-to-end confidentiality, one must decouple the content confidentiality from any hosting party by securing the content directly. More specifically, a content producer should encrypt content at the time of production, then it can control the sharing of its content by controlling the distribution of the corresponding decryption keys. In this way, supporting content-based confidentiality means addressing the following two questions: (1) *how to encrypt content*, and (2) *how to securely distribute the decryption keys.*

To address these two problems, we designed NAC, a name-based access control scheme which provides secure, distributed content sharing by encrypting content directly. Although the design described in this report is for use by applications running over an NDN network, we believe that the solution can be generally applicable to other applications which use a data-centric communication model, e.g., web, file sharing.

In the rest of this report, Section 2 provides a briefing of NDN, Section 3 introduces our basic access control model, Section 4 described how to use name in access control, Section 5 does security analysis, and Section 6 discusses a number of design considerations. We discuss related work in Section 7 and conclude our work in Section 8.

2. NAMED DATA NETWORKING

Named Data Networking (NDN) is a proposed Internet architecture which provides data-centric communication primitives. NDN changes the Internet's communication model from *delivering packets to an end host* to *retrieving content for a given name*. A host requests content by sending an *interest packet*, which specifies the name of the desired content, and the network responds by sending back a *data packet* containing the requested content. Since the requester only specifies what it wants (i.e., the data name), the network has the freedom to make intelligent decisions on where to forward an interest packet. It could be sent toward a replica of the data hosted by the original data producer or by a third-party storage provider, or get satisfied enroute by an router cache containing the requested data.

NDN uses a content-based authenticity model by requiring every data packet be signed. Besides the signature, each data packet also carries additional metadata including the signing key name as shown in Figure 1.

To authenticate a data packet, one needs a trust model that defines which keys are authorized to sign which data (*trust rules*), and specifies one or more trusted

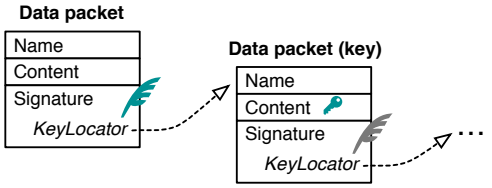

Figure 1: Authentication elements in NDN data packet

keys to bootstrap the trust (*trust anchors*). Any entity–applications, dedicated network storage elements, and even network routers–that learns the trust model for a given piece of content can verify its authenticity, and may perform necessary actions when the authentication fails (e.g., discard the packet, or try an alternative path to retrieve). Keys in NDN are just another type of data, thus they also have unique names and can be authenticated in the same way as other data packets; data packets carrying public keys are effectively NDN certificates.

NDN defined the content-based authenticity into its architecture. However this makes only one part of the content-based security model. The other part of the model, content-based confidentiality must also be provided. In the rest of this paper, we will demonstrate how to use content-based confidentiality to secure content sharing application over NDN.

3. ACCESS CONTROL MODEL

To facilitate explanation and discussion in the rest of this paper, we first introduce a simple wellness application example (Figure 2). A user Alice uses a wellness application to collect her heart rate data and activity data, which is produced by two sensors respectively. The activity sensor can produce two types of data every minute: the number of steps and the user location, while the pulse sensor only produces Alice's heart rate data.

Alice may share her data with different people at different granularities. For example, Alice may share the daily activity data with her husband Bob and occasionally share the location data with her friend Cathy when she is doing outdoor running exercise. Alice also wants to share her heart rate data with her personal physician David.

In order to facilitate data sharing, Alice may upload all the wellness data to a data storage (e.g., cloud), so that the authorized consumers can retrieve data at anytime. Alice assumes that the data storage will guarantee the availability of her wellness data, but does not rely on the data storage to enforce access control.

3.1 Content-Based Access Control

Figure 2 highlights a specific access control model, which we call *content-based access control*. In this model, the *data owner* (e.g., Alice) directly controls both data

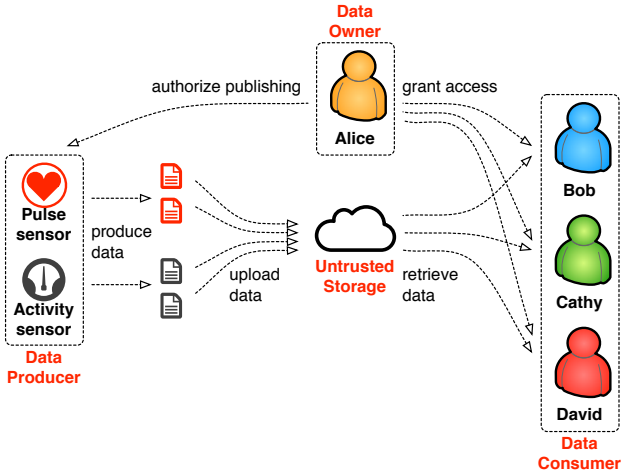

Figure 2: Example of health data sharing

production and access, i.e., who is authorized to produce data under specific branches of her namespace, and who is authorized to read the data. To avoid reliance on any intermediate device (e.g., data storage, firewalls, and routers) to enforce the access control, *data producer* must produce data in an encrypted format, so that only authorized *data consumers* (e.g., Bob) can decrypt the data.

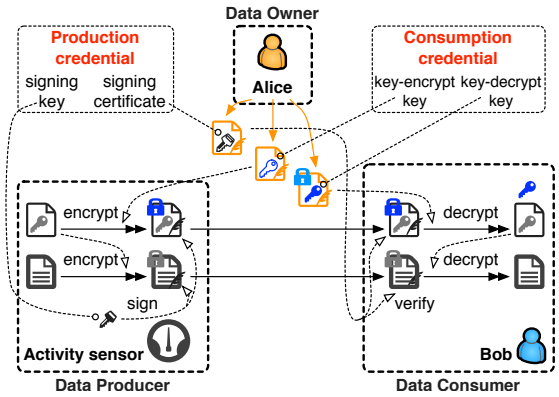

Figure 3: Production credential and consumption credential in content-based access control.

To achieve content-based access control, the data owner needs to provide two types of credential: *production credential* and the *consumption credential*, as shown in Figure 3. A production credential lets an authorized producer authenticate itself to data receivers. A producer generates a pair of public/private keys for data signing and verification. The data owner issues a public key certificate for the producer. In this certificate, the data owner explicitly specifies the production privilege, i.e., the data set that the producer is authorized to produce. The data producer signs data with the corresponding private key. Any data receiver (e.g., end consumers and data storage) can verify, with the producer

certificate, if the producer is authorized to produce the data.

A producer encrypts content using a symmetric key (*content key*), which is generated by the producer. A data owner enforces read access control by controlling the delivery of content keys. A data owner generates a pair of public/private keys, which we call *consumption credential*. As shown in Figure 3, all authorized consumers will obtain the private key (key-decrypt key, or KDK), while data producers retrieves the public key (key-encrypt key, or KEK) and use it to encrypt content key. The data owner explicitly specifies the privilege of consumption credential, i.e., the data set that a consumer is authorized to read, so that producers know which content keys should be encrypted using a particular KEK.[1]

3.1.1 Design Issues

To achieve practically usable content-based access control, we must address the following design issues.

A data owner must be able to explicitly specify privilege in both production credential and consumption credential. Expressing privilege explicitly in credentials is an important premise to provide fine-grained access control. For example, when Alice can explicitly specify in the activity sensor certificate that the sensor can only produce activity data, data consumers or the data storage can reject any non-activity data produced by the activity sensor. When Alice can explicitly specify the readable data set for each consumption credential, a consumer with the KDK can read data only within the data set, because data producers will not use the corresponding KEK to encrypt content key whose corresponding content is beyond the data set.

A data owner must be able to deliver the credentials to the corresponding entities. None of data owner, data producers, and data consumer will be online all the time. The only always-online entity in the system is the data storage. Similar to normal data, credentials will also be stored in the untrusted storage and delivered through the untrusted network. This implies that producers and consumers must directly authenticate credentials, independent from any retrieval mechanism. Sensitive credentials, such as decryption keys, must be properly encrypted, so that they are only visible to authorized consumers.

A data owner must be able to revoke the access of producer and consumer. A data owner must retain the ability of preventing a producer (or a consumer) at any time from further producing (or reading) data.

[1]Note that public/private key pair is only one of possible implementations for consumption credential. Other implementation may include identity-based encryption, attribute-based encryption, and etc.

In the next section, we demonstrate how to leverage named data and keys, together with *naming conventions*, to solve the above three problems.

4. NAMED ACCESS CONTROL

In this section, we first explain how to name data, followed by the explanation on how to name production credential (signing/verification keys) and consumption credential (key-decrypt/decrypt keys) to specify different access privileges. We will also show how to distribute keys in a distributed system to achieve the content-based access control and discuss how to revoke the access.

4.1 Naming Data

In NDN, data is named under a hierarchical namespace. This allows us to group data with the same property into the same namespace. As an illustrative example, Figure 4 shows the naming hierarchy for Alice's health data. Alice can put all her health related data (including keys as we will show later) under a namespace "`/alice/health`". Under this namespace, Alice allocates a sub-namespace "`/alice/health/samples`" for the data produced by sensors. Alice can further sort her health data into two categories: "`activity`" and "`medical`", and give each of them an individual namespace: "`/alice/health/samples/activity`" and "`/alice/health/samples/medical`". The activity namespace covers two types of data: steps ("`alice/health/samples/activity/steps`") and location ("`/alice/health/samples/activity/location`"). Each piece of data is named under the namespace for its own type, with a suffix that can describe additional information of the data. For example, the data under name "`/alice/health/samples/activity/steps/2015/08/27/16/30`" refers to the step data that is produced during 16:30 to 16:31 on August 27th, 2015 (assuming activities are measured in the time unit of one minute).

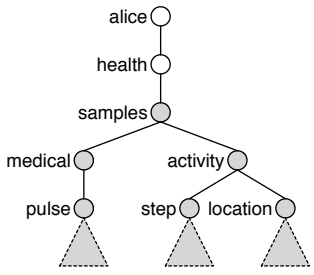

Figure 4: An example of naming hierarchy for Alice's health data

4.2 Naming Production Credential

Since data is organized under the hierarchical naming structure, a data owner can express the privilege of a

production credential as the namespace under which the producer is authorized to produce data. For example, a namespace "`/alice/health/samples/activity`" represents the privilege of producing Alice's activity data, including both step and location.

Figure 5: The naming convention of signing key

To authorize a data producer to produce data under a given data namespace, a data owner can issue a signing key certificate which associates the producer's signing key with the authorized namespace. Figure 5 shows the naming convention of signing key. A signing key name consists of three parts: 1) a prefix indicating the data namespace; 2) a key ID that with the prefix uniquely identifies the signing key; and 3) a special name component "`KEY`" that distinguishes the key from normal data under the same name.

We mentioned in Section 2 that NDN requires data producer to put its signing key name into the "`KeyLocator`" field of each produced data packet. With this information, a data consumer can construct a chain of signing keys from the producer signing key to a trusted key it already knew (e.g., Alice's root key). Consumers can check the authentication chain against a pre-defined trust model, which can be described in a trust schema [6], to decide whether a producer has been authorized to produce a particular data packet.

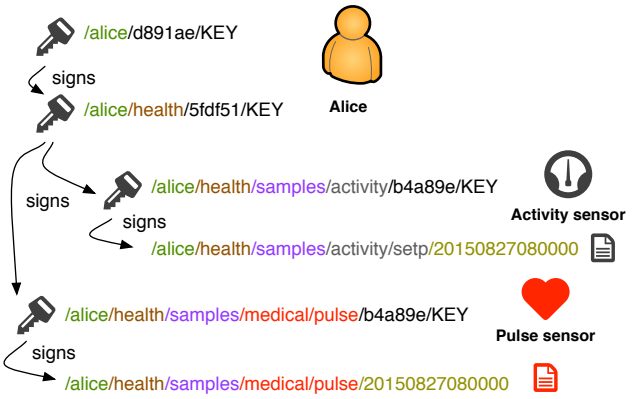

Figure 6: Signing hierarchy of Alice's health data

Figure 6 shows an example trust model for Alice's production credential authentication. This hierarchical trust model has the root key of Alice's own namespace "`/alice`" as the trust anchor. A separate key, "`/alice/health/5fdf51/KEY`", is created to manage the wellness sub-namespace ("`/alice/health`"), which is used to sign the certificate of each authorized data producer (pulse sensor and activity sensor).

A data owner may adopt different trust model for the production credential authentication. Discussions on alternative trust models are beyond the scope of this paper. Interested readers are referred to our previous work [6] about trust management in NDN.

4.3 Naming Consumption Credential

In our design consumption credential is another public key pair (KEK/KDK) for content key encryption. We use well defined naming convention to help a data owner explicitly specify the privilege of a consumption credential. We mentioned earlier that the privilege of a consumption credential is a data set that a consumer with the key-decrypt key (KDK) can access (indirectly through decrypted content keys). With the privilege encoded in the corresponding key-encrypt key (KEK) name, a data producer can tell which content key should or should not be encrypted through the key-encrypt key.

The naming of key-encrypt/decrypt key (KEK/KDK) must accommodate four facts. First, key-encrypt/decrypt keys have different usage than the signing/verification keys introduced above. A signing key is possessed by an authorized producer while a KDK is held by an authorized consumer. The roles of these two pairs of keys are different, and the key name must explicitly reflect such differences.

Second, consumption credential is created and managed by data owner. The naming convention of consumption credential should prevent other entities (e.g., producers or consumers) from issuing valid consumption credential.

Third, a data owner may want to delegate the consumption credential management to a third party. The consumption credential naming convention should facilitate such management delegation, at the same time, a data owner must be able to restrict the privilege of this third party to consumption credential management only. In other word, the third-party entity should not be able to produce wellness data on behalf of the data owner.

Last, the data set that a consumption credential covers may need additional description that cannot be explicitly encoded as the data namespace. For example, a data owner may want to create a consumption credential that allows consumers to access data produced during certain time period, e.g., from 6pm to 10pm on every workday. Therefore, the consumption credential name must be able to carry additional information to enforce a variety of access restrictions beyond the data naming hierarchy.

We will present a naming design that can address the four issues above.

4.3.1 Consumption credential namespace

To distinguish consumption credentials apart from production credentials, we allocate a separate namespace for consumption credential, which is parallel to the data namespace as shown in Figure 7. Take Alice's health data as an example, Alice can create a namespace "`/alice/health/read`" for consumption credentials. The naming hierarchy of the consumption credential namespace mirrors that of the data namespace, except that data under this hierarchy are all consumption credentials.

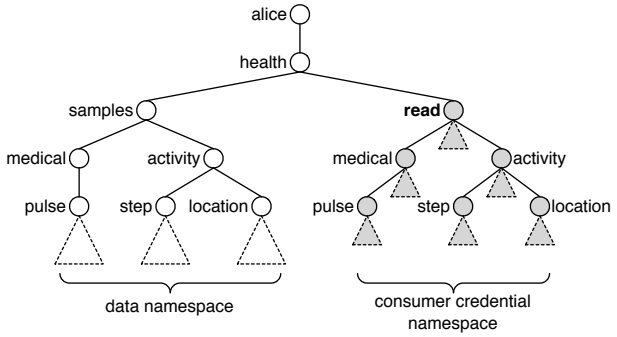

Figure 7: An example of consumption credential namespace along with data namespace

With a separate consumption credential namespace, a data owner can delegate the whole or part of the consumption credential management to a third party. The data owner can publish the delegation as a certificate which binds the third party's public key to the delegated consumption credential namespace or sub-namespace. Figure 8 shows an example of consumption credential delegation in which Alice delegated her physician to control the read access to her medical data. As restricted by the certificate name, the delegated entity (e.g., Alice's physician) can only issue consumption credential for certain type of data (e.g., medical data), and cannot issue any production credential nor produce any data, including medical data.

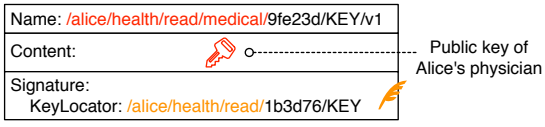

Figure 8: An example of consumption credential delegation.

The naming hierarchy under the consumption credential namespace also enables multi-level delegation. For example, Alice's physician can further delegate a cardiology expert to manage the read access to Alice's heart rate data.

4.3.2 *Consumption credential name convention*

Under the consumption credential namespace, a data owner can name a consumption credential at any level of the naming hierarchy (e.g., "`/alice/health/read/medical`", and "`/alice/health/read/medical/pulse`") with the meaning that consumers with the credential can only access data under the corresponding data namespace. We mentioned earlier that our design uses a public key pair (KEK/KDK) to construct a consumption credential. Both keys need to be named properly to convey the privilege of a consumption credential.

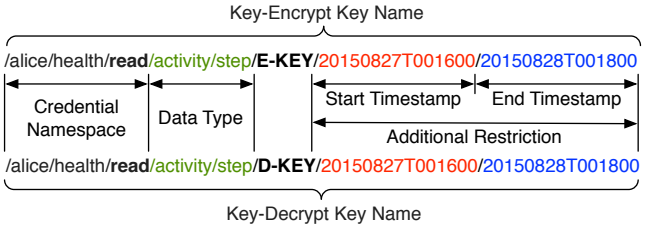

Figure 9: The key naming convention of consumption credential

Figure 9 shows the naming convention for the keys of a consumption credential. Both key-encrypt key (KEK) and key-decrypt key (KDK) share the same naming structure. They all start with a particular prefix under the consumption credential naming hierarchy. After the prefix, each type of keys has a key-tag component that distinguishes the usage of these keys: "`E-KEY`" for key-encrypt key and "`D-KEY`" for key-decrypt key. After the key-tag, a data owner can append other additional restrictions that have not been explicitly encoded in the data namespace. For example, the key names in Figure 9 says that a consumer with this corresponding credential can access Alice's step data produced between 4pm to 6pm on August 27, 2015.

4.4 Credential Delivery

For producer credential, we assume that a producer creates and retains signing key and data owner issues signing key certificate using conventional certificate issuing mechanism. Only consumers need to retrieve signing key certificates for data authentication. Since a signing key certificate is an NDN data packet, data owner can simply upload the issued certificate to a data storage. Potential data consumer can follow "`KeyLocator`" in data packet to retrieve the certificate later.

The key-encrypt/decrypt key of a consumption credential, however, are created by data owner and should be delivered to related data producers and authorized consumers respectively. We will explain how to deliver these keys to related entities.

4.4.1 *Key-encrypt key delivery*

As mentioned earlier, the key-encrypt key (KEK) in this design is a public key. A data owner can name a KEK as we mentioned above (Figure 9), and publish the key as a data packet. Since each encryption key has the

data owner's signature, they can be safely uploaded to the data storage, retrieved and verified by data producers and consumers. As long as a data producer knows the key-encrypt key naming convention, it can infer the name of the key-encrypt key to retrieve. The naming convention can be tailored for specific applications to facilitate key retrieval.

Let's consider the wellness application as an example. In this application, producers (e.g., sensors) produce wellness data continuously. With the naming convention defined in Figure 9, a data owner can specialize the "additional restriction" as a time interval, and create a sequence of KEKs. The time interval of these KEKs can be concatenated together to cover a continuous time period as shown in Figure 10. Note that this naming convention implies that the ending timestamp of a KEK is the starting timestamp of the next KEK.

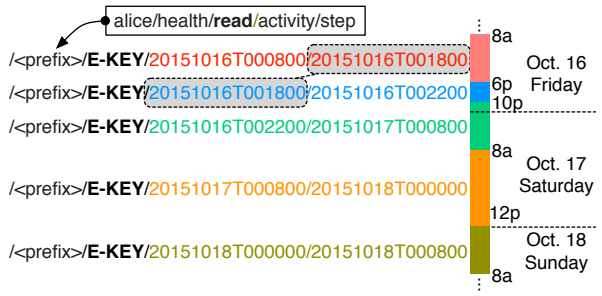

Figure 10: A sequence of key-encrypt keys cover a continuous time period.

To construct an interest to retrieve a KEK, a data producer must first determine the credential prefix. Note that there could be multiple prefixes which a data producer can infer from the name of produced data. For example, given Alice's step data "`/alice/health/samples/activity/step`", the activity sensor can derive the most specific credential prefix "`/alice/health/read/activity/step`" corresponding to the step data namespace. Since every parent credential prefix of the most specific prefix also covers the step data, the data producer can determine all the possible credential prefix by tracing back to the root of the credential namespace (e.g., "`/alice/health/read`"). In the example above, the activity sensor can deterministically derive three prefixes: "`/alice/health/read`", "`/alice/health/read/activity`", and "`/alice/health/read/activity/step`".

For each derived credential prefix, a data producer needs to determine the starting timestamp for the KEK to retrieve. We mentioned earlier that the ending timestamp of a KEK is the starting timestamp of the next KEK. When a data producer has already obtained a KEK, it can construct an interest for the next KEK by specifying the starting timestamp using the ending timestamp of the obtained key. Routers and data storage can apply the longest prefix match to pick the next KEK and satisfy the interest.

If a data producer has not received any KEK before, the data producer can express an interest with the credential prefix (e.g., "`/alice/health/read/activity/E-KEY`"). The interest can bring back a KEK which can serve as a starting point for the KEK enumeration as described above.

When a retrieved KEK's ending timestamp is much earlier than current timestamp, KEK enumeration may become inefficient. In this case, a data producer can use "`Selectors`" to speed up the key enumeration process. More specifically, a data producer may use "`Exclude`" filter to exclude any KEK whose starting timestamp is earlier than the latest one among all the received KEKs.[2] A data producer may also specify "`ChildSelector`" to select the "rightmost" KEK under the credential prefix.

4.4.2 *Key-decrypt key delivery*

Key-decrypt key (KDK) should be visible only to authorized consumers. Note that a data owner may not be online all the time, it would be desirable for the data owner to leave the KDKs in a data storage for authorized consumers to retrieve it whenever needed. Since the data storage is untrusted, a data owner can encrypt a KDK using the public key of each authorized consumers. Each encrypted copy makes an individual data packet.

Figure 11 shows the format of encrypted data. The data content consists of two components: "`EncryptionAlgorithm`" which the meta-information about the encryption scheme and "`EncryptedContent`" which contains the cipher text of content. Note that the format is general enough to carry any content which is not restricted to KDKs but also include content key and normal content.

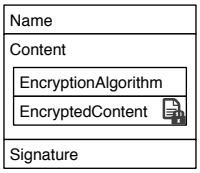

Figure 11: Data packets carrying encrypted data and keys

Since each consumer has its own encrypted copy of KDK, each copy must have a unique name. To distinguish different copies, we define the naming convention for encrypted data as shown in Figure 12. For each encrypted copy, we append a special name component "`FOR`" and the encrypting key name after the

[2]In case clock is not synchronized, one may also set "`Exclude`" filter to exclude any KEK whose starting timestamp is later than current timestamp.

content name. For example, a decryption key for Alice's activity data that is encrypted using Bob's public key is named as "`/alice/health/read/activity/D-KEY/20151016T000800/20151016T001800/FOR/bob/health/access/E-KEY`".

/<ContentName>/**FOR**/<EncryptionKeyName>

Figure 12: Naming convention of encrypted data

The name of encrypting key in each data packet can help a data consumer to construct a decryption chain to access the original content as shown in Figure 13. When a consumer retrieves an encrypted data packet, it can extract the content key name from the data name. We assume that an authorized consumer should know its authorized credential prefix[3]. With the content key name, a consumer can construct an interest by appending the consumption credential prefix to the content key name. With longest prefix match, routers and data storage can satisfy the interest with the encrypted content key. After receiving encrypted content key, the consumer can extract the key-encrypt key (KEK) name and construct an interest for the corresponding key-decrypt key (KDK) by appending the consumer's own name to the KDK name. In the end, the consumer can retrieve the encrypted KDK and recursively decrypt all the intermediate keys and the original content.

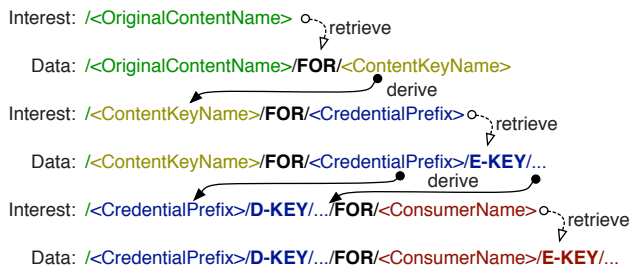

Figure 13: A chain of keys to decrypt wellness data

4.5 Fine-Grained Access Control

With consumption credential, a data owner can control the read access to content from two dimensions: specifying the privilege of individual consumption credential and restricting the set of credentials that a consumer can obtain. For example, Alice may want to share her location information with her husband Bob all the time, but with her colleague Cathy only during working hours. In this case, Alice can specify a sequence of consumption credentials which cover her location data for 9am-5pm and 5pm-9am every day. Alice can encrypt the KDKs for 9am-5pm from Monday to Friday for both Bob and Cathy, and encrypt all the other decryption keys for Bob only, as shown in Figure 14.

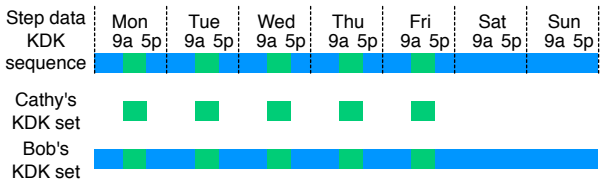

Figure 14: Different read privilege in terms of KDK set.

In fact, a data owner can divide the data set arbitrarily into multiple consumption credentials (KEK/KDKs), so that the data owner can create different combination of KDKs to represent different privilege of individual consumers. When the read privilege can be pre-defined, consumption credentials can be automatically created and published in the network.

4.5.1 *Post-Facto Access Granting*

The model we discussed so far focus on controlling the access to data as they are being produced. The name-based access control also allows a data owner to grant a new consumer the access to the data that is produced long time ago. When the granted access is covered by one or more KDKs that were generated earlier, the data owner can simply encrypt KDKs directly using the new consumer's public key.

Note that a data owner can always create a top-level consumption credential (e.g., "`/alice/health/read`") and retain the KDKs to itself only. Since every producer will publish a copy of content key encrypted using the KEK of the top-level credential, the data owner can obtain all the content keys. As a result, when the granted access cannot be expressed as a combination of existing KDKs, the data owner can re-encrypt the granted content key directly for the new consumer.

4.6 Access Revocation

With content-based access control, revoking write access is equivalent to revoking the producer's public key certificate, so that neither data storage nor end consumers will accept data of the revoked producer. A data owner can also easily prevent a previously authorized consumer from reading any new data by stopping publishing consumption credentials for the consumer. However, revoking data access that has been granted requires strict control on the availability of KDKs, i.e., preventing a revoked consumer from accessing these KDKs. For example, a data owner may delete from a data storage the KDKs encrypted for a revoked consumer.

A more effective solution can be provided at the application layer. For example, to control the access to video with copy right, a video provider (e.g., Netflix, Hulu) may ship its own video player as a blackbox to

[3]An authorized consumer does not have to know the complete credential name, i.e., the full name of each decryption key.

user. The video player not only decrypts the video stream, but also can prevents users from retaining a copy of the decrypted video. We assume the same techniques can be applied here to control the access to the KDKs. More specifically, the blackbox can negotiate an ephemeral keys for KDK distribution and throw the ephemeral keys and KDKs away after data decryption. For a revoked consumer, the blackbox will fail to obtain an ephemeral key, thus preventing the consumer from accessing content.

5. EVALUATION

We perform a comparative evaluation on NAC. We first compare NAC with CCN-AC [4], another encryption-based access control scheme using regular public key cryptography, about the overhead of crypto computation and key retrieval. We examine the difference between NAC and Attribute-Based Encryption [3], another encryption scheme which is often mentioned as an easy-to-use solution for encryption-based access control.

5.1 NAC vs. CCN Access Control

CCN-AC [4] is an encryption-based access control scheme where each data producer has complete knowledge about the access control policy, i.e., who are the authorized consumers and what the consumers are allowed to access. We first consider the total number of encryption/decryption operations that both NAC and CCN-AC must perform to distribute content keys. We compare the two schemes under two scenarios.

The first scenario includes m producers under the same namespace and n consumers that are authorized to access data under the namespace for one day. For both schemes, we assume that each producer will generate per-hour content key to encrypt data it produces within each hour.

In CCN-AC, each producer needs to explicitly encrypt the content key for each authorized consumer. Therefore, the total number of encryption operations in one day can be calculated as:

$$N_{ccn-ac} = 24mn \tag{1}$$

For NAC, a data owner can create a consumption credential for all the data produced in one day and encrypt the credential KDK for each authorized consumer. Each producer only needs to encrypt each content key with the credential KEK.

$$N_{nac} = 24m + n \tag{2}$$

Clearly, the number of encryption operations that NAC has to perform is much less than CCN-AC, especially in cases of a large number of producers or consumers.

In the second scenario, we also consider m producers but 24 consumer groups, and each group has n consumers. Each group of consumers can access data produced in a particular hour of a given day. In CCN-AC, each producer will encrypt a content key for all the authorized consumers:

$$N_{ccn-ac} = 24m + 24mn \tag{3}$$

For NAC, a data owner needs to create 24 credentials. A data owner will encrypt each KDK for authorized consumers, while each data producer encrypts the content key using the corresponding credential KEKs:

$$N_{nac} = 24m + 24n \tag{4}$$

The result suggests that NAC scales better than CCN-AC when there is more than one producers in the system. Note that the scalability comes from the one-level indirection of data owner, which aggregates multiple consumers into a group, so that data producers only need to be aware of the group's KEK rather than the key of each group member.

Our second evaluation metric is the overhead in key retrieval. CCN-AC puts all encrypted content keys into a single data blob, called *manifest*. A consumer must retrieve the whole manifest to extract the content key encrypted for it. When the size of the data blob is larger than the network's maximum transmission unit (MTU), the data blob must be segmented. Assume that MTU is 1500 bytes and that consumers use RSA keys, one data segment can carry about 4 encrypted content keys. Assuming that a content key is granted to n consumers, the average number of data packets that a consumer must retrieve is:

$$N_p = \lceil \lceil n/4 \rceil \cdot 2 \rceil \tag{5}$$

This analysis suggests that a CCN-AC consumer only needs to retrieve one key packet when there are fewer than 4 authorized consumers for a content key, but will need to retrieve multiple key packets when the number of authorized consumers increases, and the number increases linearly with the number of authorized consumers. In contrast, NAC requires a consumer to retrieve a content key and a key-decrypt key, of which each is carried by a separate data packet, therefore only two data packets are needed to retrieve these two keys.

5.2 NAC vs. Attribute-Based Encryption

Before we make a brief comparison between NAC and Attribute-Based Encryption (ABE) [3], it is helpful to understand the working mechanism of ABE (Figure 15(b)). Unlike traditional encryption techniques, ABE encrypts data using a set of predefined, descriptive attributes instead of crypto keys, eliminating the need for data producers to fetch encryption keys. To enable such a scheme, ABE requires a key authority that knows the attributes of all the receivers and can generate a *master key* and its corresponding *public params*. An ABE receiver (data consumer) must obtain its private key from this key authority. The key authority derive

a user's private key from the master key together with the user's attributes. Users with the identical attribute set will obtain the same private key. An ABE sender (data producer) generates ciphertext using the public params together a set of attributes. A receiver can decrypt a ciphertext only if the receiver's attributes match the attributes with which the ciphertext is generated.

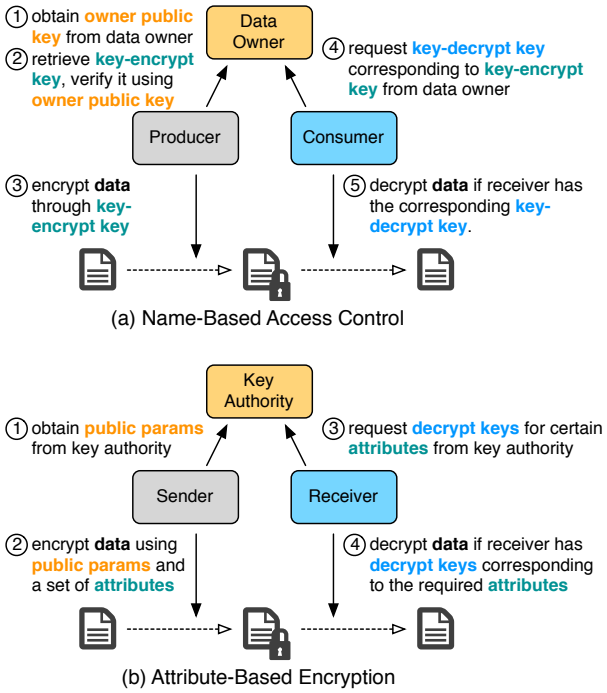

Figure 15: Comparison between attribute-based encryption and name-based access control

Figure 15 shows the working flow of both NAC and ABE. Next, we will compare ABE based access control and NAC on four aspects: setup, encryption, decryption, and revocation.

5.2.1 *Setup*

Both NAC and ABE requires an authority (the data owner or key authority) that determine how encryption and decryption keys match up. In NAC, the encryption and decryption keys are generated by the data owner. The data owner names the encryption/decryption keys by their encryption scope and signs the keys, both producers and consumers must learn the *owner public key* at the setup phase, so that they can authenticate the encryption/decryption keys and use them properly.

In ABE, an encryption key can be constructed by any sender, and the corresponding decryption key is derived by the key authority. In order to pair up encryption and decryption keys, a sender must learn the *public parameters* of the key authority at the setup phase, and use them to generate encryption key correctly. Receivers must learn certain information (e.g., key authority's public key) so that they can securely request decryption keys from the key authority.

The ABE scheme, however, also requires each sender to be configured with the knowledge about all the supported attributes, i.e., which attributes should be used to encrypt which data. In contrast, NAC does not require such configuration, but relies on the encryption naming convention to determine which keys should be used to encrypt a particular piece of data.

5.2.2 *Encryption*

NAC and ABE differ most significantly in encryption. NAC requires producers to periodically retrieve KEKs from data owners, thus introducing the overhead of key retrieval. While failure in retrieving a KEK does not block data production, it may prevent consumers from accessing the data. In contrast, ABE waives the need of key retrieval. A sender can directly encrypt data with a set of descriptive attributes. In order to control the time granularity of encryption, a sender may add time as one of encrypting attributes.

The computational overhead of two schemes are determined by different factors. In NAC, a producer encrypt data using a content key, which in turn is encrypted by one or more key-encrypt keys. In the worst cases where there is a key-encrypt key at each level of the key hierarchy, the computational overhead is proportional to the depth of the key hierarchy. However, in ABE, a producer may need to process each attribute used in encryption, the computational overhead will linearly increase with the number of attributes used in the encryption.

5.2.3 *Decryption*

Both NAC and ABE relies on the authority (data owner or key authority) to generate decryption keys and distribute the keys to corresponding consumers if they have authenticated themselves to the authority. In both scheme, the generated decryption keys must be securely delivered to the consumers.

NAC creates a decryption key hierarchy and assign consumers into the different levels in the hierarchy according to the consumer's privileges. ABE expresses a consumer's privilege in terms of the consumer's attributes and crafts decryption keys according to the consumer's attributes. While ABE waives the need of maintaining the decryption key hierarchy, the process of key generating and decryption is non-trivial, and may introduce significant computation overhead.

5.2.4 *Revocation*

Both NAC and ABE handle revocation through restricting the livetime of encryption keys. In both schemes, consumers have to periodically "renew" the corresponding decryption keys. In NAC, data owner explicitly

specifies the lifetime of KEKs in the key name. As a result, the data owner controls the temporal granularity of keys. In contrast, senders in ABE specify the validity period of the encryption key as an attribute involved in the encryption. When the validity period of a receiver's decryption key does not satisfy the required attributes, the receiver must request a new decryption key with appropriate validity period from the key authority. Therefore, the lifetime of encryption keys is usually determined by the senders in ABE.

6. DISCUSSION

6.1 Consumption Credential Implementation

Besides public/private key, there are several other design options of implementing key-encrypt/decrypt key in consumption credential. The first one is symmetric key. In this case, a data owner not only needs to encrypt the symmetric key for each authorized consumer, but also needs to encrypt the symmetric key for each related producers. Therefore, this design introduces more encryption overhead.

Another option is attribute-based encryption. In this case, a data owner does not need to publish key-encrypt key for data producers. Instead, a data producer can encrypt content key with well-known attributes. This option requires delicate design of attribute set. We plan to compare the complexity of attribute-based encryption against NAC, and investigate the feasibility of implementing consumer credential using attribute-based encryption.

6.2 Emergent Revocation

A data owner needs to pre-specify the effective time interval of each consumption credential, but it is possible that the data owner may want to revoke a consumer's access before the end of the time interval. A data owner can publish a new consumption credential with a new starting timestamp as an indication of revoking the previous KEK/KDK, so that producers will use the new KEK/KDK to encrypt content key. However, this solution requires producers to pro-actively retrieve KEK all the time. Another solution is to specify short-lived consumption credentials. This solution alleviates the burden of data producers, but it requires data owners to publish credential more frequently.

6.3 Forward Secrecy

Forward secrecy requires past communication to be free from compromise of a long lived key. Since our design directly encrypt KDKs using a consumer's public key, compromise of a consumer's private key may allow an attacker to access all the data that the consumer has accessed before.

A possible solution is to encrypt KDKs using an ephemeral key which is negotiated through a plain-text key exchange protocol, such as Diffie-Hellman key exchange [1]. Since the ephemeral key will be thrown away once the KDK is decrypted, compromise of a consumer's private key cannot help an attacker to recover ephemeral keys.

An online key distribution service, however, must exist to run the key exchange protocol and negotiate ephemeral keys with consumers. In this paper, we assume the only always-online entity is an untrusted data storage. For applications or systems that requires forward secrecy, we may relax the restriction on the online entity and enable key exchange service on it.

6.4 Content-Based v.s. Perimeter-Based Access Control

Both models have their own advantages and disadvantages. Content-based access model eliminates the trust over data storage and middle boxes. However, it requires additional mechanism to control the content availability. As a results, revocation in content-based access control cannot prevent a consumer from reading data whose read access was granted to the consumer previously, as long as both the data and keys are available. In contrast, perimeter-based access model directly controls data availability, but it requires the enforcement of access control policy in every device on the perimeter.

6.5 Key-Encrypt Key Distribution

In Section 4, we explained how to leverage naming conventions to facilitate the key-encrypt key distribution when the additional restriction has only one dimension (e.g., time). However, when there are more than one dimension of additional restriction (e.g., geofencing), data producers need a name-independent mechanism to retrieve key-encrypt keys (KEKs).

We consider data synchronization as a promising approach to distribute KEKs with various additional requirement. For example, a data owner can create several data sets (one for a particular consumption credential prefix) and publish new KEKs in the corresponding data set. Producers can synchronize the KEK set that is related to their interest, so that they can get notification at first time when a new KEK is published.

6.6 Automated Consumer Authorization

In this paper, we assume that data owner manually authorizes each consumer. It is also possible to automate the consumer authorization. For example, a data owner may accept read requests from consumers. After a consumer's key is authenticated, the consumer's public key can be automatically used to deliver the key-decrypt key (KDKs) for the authorized data set. Dif-

ferent data owner may apply different trust model to authenticate consumers. Trust schema [6] can help data owner to customize their trust model and automate consumer authorization.

6.7 User Key Management

The NAC design requires participants to have their own public/private key pairs. How to educate users to correctly manage their keys remains a challenging security problem. In NDN, we assume key management has become a normal practice for users.

7. RELATED WORK

There are several existing works on enforcing access control over information-centric network. Ghali [2] proposed an interest-based access control solution. However, the solution requires every router in the network to enforce the data producer's access control policy. Our work made a weaker assumption that network is not trustworthy, and we aimed at minimize dependency on intermediate devices to enforce access control.

Kurihara [4] proposed an encryption-based access control framework. The framework enforce access control by encrypting content directly. However, the framework assumes that each producer has full knowledge about the access control policy. In contrast, our work consider a more general scenario in which multiple producers may collectively produce content under the same namespace. In this scenario, it may be infeasible to notify each producer of any change in the access control policy, e.g., adding a new consumer or removing an existing one. Our work introduces one-level indirection by explicitly dividing the information about access control policy into two parts, thus having better scalability.

Misra [5] proposed another encryption-based access control scheme which used broadcast encryption to achieve large scale content delivery.

8. CONCLUSION

Content-based access control model provides a new perspective for end-to-end confidentiality. By requiring content encryption at the time of production, the model minimize the dependency on any intermediate device for access control. Therefore, the model can achieve true end-to-end confidentiality at the application level, thus can be naturally fit into the data-centric architecture, such as NDN.

The hierarchical namespace of NDN can convey rich contextual information about access control. Our design reveals that by defining correct naming conventions for encryption/decryption keys and signing/verification keys, one can achieve effective access control at fine granularity, even with simple public key cryptography. We also demonstrate in this report that a well-designed naming convention can convey access control policy clearly, thus significantly reducing the number of crypto operations and facilitate encryption key distribution in certain scenarios.

Name-based access control is our first step to achieve content-based access control. It introduced several interesting open questions as we discussed above. We will address these questions in the future to provide a usable solution for content-based access control.

9. REFERENCES

[1] W. Diffie and M. E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 1976.

[2] C. Ghali, M. A. Schlosberg, G. Tsudik, and C. A. Wood. Interest-based access control for content centric networks. In *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015.

[3] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, 2006.

[4] J. Kurihara, C. Wood, and E. Uzun. An encryption-based access control framework for content-centric networking. In *IFIP Networking Conference*, 2015.

[5] S. Misra, R. Tourani, and N. E. Majd. Secure content delivery in information-centric networks: design, implementation, and analyses. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013.

[6] Y. Yu, A. Afanasyev, D. Clark, kc claffy, V. Jacobson, and L. Zhang. Schematizing trust in Named Data Networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, September 2015.