ICN Content Security Using Encrypted Manifest and Encrypted Content Chunks

Dante Pacella
Verizon Labs
dante@verizon.com

Mani Tadayon
Verizon Labs
mani.tadayon@verizon.com

Ashish Sardesai
Verizon Labs
ashish.sardesai@verizon.com

Venkat Josyula
Verizon Labs
venkat.josyula@verizon.com

Abstract

ICN defines a new paradigm for networking recognizing the fact that modern communication is primarily focused around retrieving objects via the network. ICN allows an end host to ask the network for a named object and receive it from any source while being able to trust the content authenticity regardless of the source. This is a very powerful feature and is made possible by content signatures. This feature enables ubiquitous/opportunistic caching of the content throughout the network.

Traditional content distribution networks require an abstraction layer that resides above the network to facilitate optimizations in content delivery. In many cases, content producers require multiple CDNs to deliver content, each with their own independent abstraction layer. These abstraction layers move traffic without direct knowledge of the underlying network and can enact traffic distribution policies that are orthogonal to the underlying network. The decision to cache content and the decision to deliver content is entirely managed within the closed ecosystem of each CDN, as contractually guided by the content producer.

This ubiquitous/opportunistic caching ability within ICN in turn means that content is available throughout the network and when any consumer of content (hence for referred to as consumer) issues an Interest for content it receives the content from the nearest network node which has the content in cache. This poses a challenge for any content owner interested in maintaining control over distribution of the content specifically in areas of authentication, authorization, and usage analytics.

This is a proposal for solving aforementioned issues by the content owner generating encrypted content, encrypted manifest per consumer, and modifying the Namespace in initial Interest message for identification, authentication, and authorization.

Keywords

Information Centric Networking, Longest Match, Named Data Networking, Content Security, Chunk Groups, Nameless Objects, User Identifier, Encrypted Manifest, Scalability

1. Background

The creative design of ICN as a protocol for information / content distribution, solves the problem of content authenticity regardless of location from which the content is obtained. While this feature of ICN makes it very powerful for content distribution, it also introduces challenges for maintaining control over authorized access to content. If a consumer obtains the name of the content (or chunks that collectively constitute the content), this name can be used to retrieve the content for future playback, or if shared, even used by other consumers to retrieve the content. This means that content owners lose control of the distribution of their content.

2. Design

2.1 Chunking

For optimal utilization of network and storage resources, the first step in preparing large scale content for distribution is to chunk the content into smaller pieces. The chunk size is determined by content-type, receiving application, and delivery environment.

There are multiple methods of naming a Content Chunk:

- Name structure derived from a predetermined schema
- Name structure based on computed hash of a content chunk which results in a Nameless Object

With the first option chunk names can easily be inferred and thereby the schema lowers privacy and security, hence we propose using the second option i.e., Nameless Objects.

2.2 Encryption

Once content has been divided into chunks, the chunks are encrypted using unique key per chunk. This approach results in proliferation of keys and key management overhead. In order to simplify key management and optimize content distribution, we propose using Chunk Groups and encrypting all chunks in each group with a single group key.

To further enhance owner's control over the content we propose using encryption keys with limited validity period. The expiration of encryption keys triggers regeneration of the encrypted chunks with the new key(s) and invalidates the existing cache entries in the network. The key validity period per content can be gauged based on a trade-off between cache lifetime and bandwidth or performance of the network, with compute impact to key generation as a function of overall asset catalog size factored.

Key regeneration would most likely occur wherever content encryption occurs. Each asset would likely be processed from a single location, though the processing of a catalog can be distributed between assets. Each key can be associated with a portion of an overall catalog. Regeneration events can be staggered such that only a portion of a catalog is processed at a given time interval. These processing event windows can be constructed in rolling windows so that a catalog is continually having keys regenerated and refreshed.

2.3 Manifest Generation

A Manifest is generated associated with the name of each Content.

The Manifest is created as one or more Manifest Objects that each contain information associated with a Chunk Group and a link to the next Manifest Object (except in the last Manifest Object in the chain). Currently each Manifest Object has a locator associated with a Chunk Group and a list of corresponding Nameless Content Objects. Each Nameless Content Object reference is derived from the hash of that Chunk.

This method proposes including encryption credentials/ keys associated with each Chunk Group in the Manifest Object along with the locator and each Nameless Content Object reference being derived from the hash of that encrypted Chunk as depicted below.

```
+
| + +--------------------------------------------
----------------+
| | Locator: Name of the Chunk Group
| |  Key: <Encryption Credentials associated with
Chunk Group>
| | +
| | | { Hash(Encr[Chunk1]) }
| | | { Hash(Encr[Chunk2]) }
| | | { Hash(Encr[Chunk3]) }
| | | { Hash(Encr[Chunk4]) }
| | | { Hash(Encr[Chunk5]) }
| | | { Hash(Encr[Chunk6]) }
| | | { Hash(Encr[......]) }
| | +
| |
| + +--------------------------------------------
----------------+
|
| + +--------------------------------------------
----------------+
| | Locator: Name of the Next Manifest Object
| |
| | +
| | | { Hash(Next Manifest Object Pointer) }
| | +
| |
+ + +--------------------------------------------
---------------+
```

Figure 1: Manifest with keys

2.4 Delivery

Existing ICN mechanisms provide for consumers requesting content by issuing an Interest message. In response to the interest, producer retrieves the manifest for that content, encrypts it using the requesting consumer's public key, and responds with encrypted manifest. On reception of encrypted manifest, the consumer uses the corresponding private key to decrypt the manifest and issues requests (Interest messages) for content chunks as per the manifest. Interests for content chunks are satisfied by the nearest node that has the content chunk available. The consumer uses chunk group key listed in the manifest to decrypt associated chunks.

2.5 Initial Interest Forwarding

By default Interest messages requesting a specific content are identical for different users. This allows for Interest aggregation but it prevents the producer from receiving per consumer Interest messages resulting in inability to authenticate and authorize access to content on a per consumer basis. To provide these controls to producers, two potential solutions were considered:

- Adding an indicator (such as a ""DO NOT AGGREGATE" flag) in the Interest message to prevent Interest aggregation on intermediate nodes.
- Modifying the Namespace of Interest messages requesting a specific content by inserting a consumer Identifier to make the Interest unique and prevent Interest aggregation

For the first option to work, in addition to the indicator, a consumer identifier would also need to be passed within the Interest message. Intermediate nodes would then be required to implement special processing to achieve uniqueness per PIT entry, as namespaces would be identical.

In case of the second option, Namespace modification provides the producer with consumer identification and makes the Namespace unique without requiring intermediate nodes to implement special processing to achieve uniqueness per PIT entry.

Namespace modification approach is selected as it is the better of the two options for achieving uniqueness per Interest and PIT entry, due to its minimal impact on intermediate nodes.

An example of the modified namespace structure is depicted in the figure below:

```
        /<content_namespace>/<content_name>/ID=<c
onsumer_identifier>
```

Performance and scaling impacts of implementing this proposal on PIT and FIB tables were analyzed.

In case of PIT, the entries are transitory in nature; therefore, having unique interests for content per consumer is not an issue of scale. Once the unique Interest is satisfied, the PIT entry related to the modified namespace is purged.

In case of FIB, support for longest match is required as without it, there is a large increase in FIB entries (<number of items in content catalogue> X <number of service subscribers)>. Longest match allows for a much more efficient FIB size.

The section below illustrates the proposal using an example.

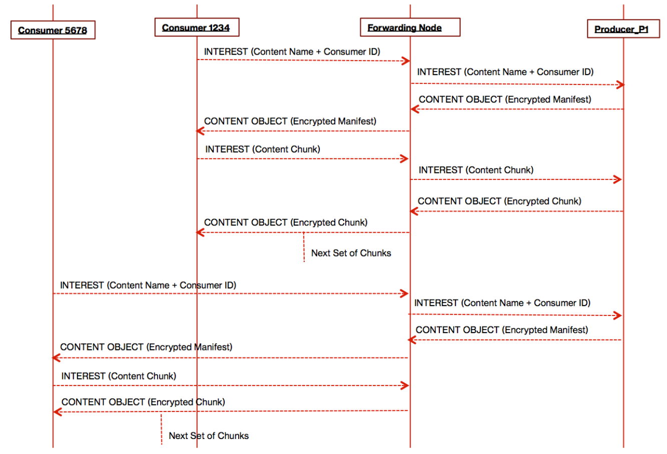
```
consumer1234
 +
 |
 |                +---------------+
 |                |               |
 |                |               |
 +--------------+ |               |
                  |               |FIB: Next Node
                  |     NODE1     |+------------>
                  |               |
 +--------------+ |               |
 |                |               |
 |                |               |
 |                +---------------+
 |
 +
consumer5678
```

Figure 2: Leaf node with two consumers attached

Example base structure of namespace modification for Interest messages from different consumers making concurrent requests for same content.

```
        /foo/bar/content1/ID=consumer1234

        /foo/bar/content1/ID=consumer5678
```



Consumer 5678
Consumer 1234
Forwarding Node
Producer_P1
INTEREST (Content Name + Consumer ID)
INTEREST (Content Name + Consumer ID)
CONTENT OBJECT (Encrypted Manifest)
CONTENT OBJECT (Encrypted Manifest)
INTEREST (Content Chunk)
INTEREST (Content Chunk)
CONTENT OBJECT (Encrypted Chunk)
CONTENT OBJECT (Encrypted Chunk)
Next Set of Chunks
INTEREST (Content Name + Consumer ID)
INTEREST (Content Name + Consumer ID)
CONTENT OBJECT (Encrypted Manifest)
CONTENT OBJECT (Encrypted Manifest)
INTEREST (Content Chunk)
CONTENT OBJECT (Encrypted Chunk)
Next Set of Chunks

Figure 3: Interest and Content Flow

In its simplest form, Namespace modification could be achieved by adding a consumer_ID or the value of the public key of the consumer to the content name. However, in order to obfuscate subsequent consumer requests from intermediate nodes such that the transit carrier cannot perform usage tracking, we propose using a nonce to randomize this field. The Namespace modification takes the form of a consumer_ID plus nonce encrypted with the public key of the producer/provider.

Note: *synchronizing the randomization scheme between producer and consumer is outside of the scope of this proposal.*

Example structure of namespace modification for Interest messages from different consumers for the same content.

```
        /foo/bar/content1/ID=dfdec888b72151965a34
b4b59031290 --encrypt(<random> + consumer1234)

        /foo/bar/content1/ID=21596697d99734b8ac40
4c4baa3988a --encrypt(<random> + consumer5678)
```

Example structure of namespace modification for Interest messages from same consumer for any content.

```
        /foo/bar/content1/ID=22f65b72888151965a90
3129034b1b5 --encrypt(<random> + consumer1234)

        /foo/bar/content2/ID=855c3697d9979e78ac40
4c4ba2c6653 --encrypt(<random> + consumer1234)
```

Transit forwarding nodes use longest match against FIB entries to forward the interest towards the producer/provider

2.6 Manifest Delivery

At the producer/provider node, the consumer identifier is extracted from the Namespace and is matched with the provisioned data associated with that consumer. Upon successful match, the Manifest associated with the requested content is sent to the consumer.

As this Manifest contains encryptions keys, we propose a uniquely encrypted Manifest for each consumer using their public key to prevent unauthorized access to encryption keys in the manifest. This enables producers to maintain control over distribution of the content as well as discrete security per consumer. Any producer participating in the generation of encrypted Manifests for the same consumer community must have access to the consumer provisioned data to aid in Authentication, Authorization and gathering Usage Analytics.

***Note:** exchanging consumer's public key is assumed to be a part of initial service setup and is outside of the scope of this proposal.*

Only the consumer that has the paired private key can decrypt this encrypted Manifest.

In Figure 1, Consumer1234's Manifest can only be decrypted with its private key. Consumer1234's Interest and Consumer5678's Interest must each have separate PIT entries, so that the unique Manifests are delivered to their respective clients.

Normally, Manifests are cacheable, just like any other Object, however due to the unique Manifest Namespace, each node in the path will not overwrite the Manifest in cache when the node receives the next request for the same content, as a result, caching of the Manifest has validity only to recover from data loss for the original requesting consumer (i.e., if Consumer1234 does not receive the manifest because of data corruption or momentary loss of connectivity).

Due to this per consumer encryption, Manifests are single use and not cache worthy for intermediate nodes.

2.7 Content Delivery

After receiving the encrypted Manifest, the Consumer decrypts the Manifest with its Private Key.

The Consumer uses locator and nameless reference within decrypted Manifest to form the Interest to retrieve the content chunk. Any intermediate node along the Interest forwarding path that contains the chunk may satisfy the Interest. If none of the intermediate node caches contain the chunk, the producer satisfies the Interest. Cache fill operations can occur on any intermediate nodes at this stage.

Once the consumer obtains the chunk, the corresponding Chunk Group Credentials/Key is used to decrypt the corresponding chunk. Each piece of content can have multiple Chunk Groups and Keys associated with the content, as previously described.

While the Manifest is encrypted such that it becomes unique per consumer, content chunks associated with a given content are identical for a consumer community. Interest messages by multiple consumers from a consumer community for a given chunk are identical, allowing ubiquitous/opportunistic caching of the chunks. As this represents a majority of the byte count being transferred between cache and consumer, with Manifest being an incidental portion, bandwidth savings is still achieved in this proposed model.

3 Usage Analytics

This proposal allows producers to have access to consumer identification and, hence, enables another source for collection of data on consumer usage patterns and generate analytics that facilitates producers to enact appropriate subscription-based revenue-generating models. This proposal also prevents profiling of individual consumers by intermediate nodes unassociated with the producer or consumer.

4 Summary

ICN Content Security provides a scalable and distributed method for content access controls while still maintaining the maximal amount of ICN paradigm. With every chunk being uniform, all chunks can be cached ubiquitously. The chunks represent the majority of data transmitted under this method; therefore, ICN's objective of bandwidth savings is still achieved.

By using the modified namespace to insert a consumer identifier, this method guarantees uniqueness for the Manifest Interest, allowing for discrete controls and usage analytics. Because the majority of Interests are generated for retrieving uniform chunks, these Interests are aggregated in the PIT. Longest prefix match results in efficient and manageable FIB sizes across the network.

5 References

[CCN] PARC, Inc., "CCNx Open Source", 2007, <http://www.CCNx.org>.

[CCN] Jacobson, V., et al., "Networking Named Content", Proc. CoNEXT, ACM, 2009.

[CCNx] Mosko, M., Solis, I., and E. Uzun, "CCN 1.0 Protocol Architecture", Project CCNx documentation, Xerox Palo Alto Research Center, 2013, <http://ccnx.org/pubs/ICN_CCN_Protocols.pdf>.

[CCNMessages] Mosko, M., Solis, I., and M. Stapp, "CCNx Messages in TLV Format (Internet draft)", 2015, <http://tools.ietf.org/html/draft-mosko-icnrg-ccnxmessages-00>.

[CCNTlv] Mosko, M. and I. Solis, "CCNx Messages in TLV Format (Internet draft)", 2015.

[FLIC] Tschudin, C. , Wood, C., "File-Like ICN Collection (FLIC)", 2016, <https://www.ietf.org/archive/id/draft-tschudin-icnrg-flic-01.txt>.

6 Acknowledgements

We thank Jean McManus, Verizon Labs Executive Director, for comments that greatly improved this paper.