CCNx 1.0 Collection Synchronization with Secure Catalogs

Marc Mosko[1]*

Abstract
We describe methods to secure Sync with Exact Names. The first method uses hash chains. The second method uses secure catalogs.

Keywords
Content Centric Networks – Named Data Networks

[1] *Palo Alto Research Center*
***Corresponding author**: marc.mosko@parc.com

Contents

Introduction

Sync with Exact Names uses advertisements the cryptographic hash of a manifest within a namespace that identifies the collection of data. However, this method does not inform a receiver of the cryptographic identity who created the manifest (hereafter known as the "keyid") or of the Content Object Hash needed to retrieve an exact content object. We describe two methods to secure the process by hash chains and by secure catalogs.

1. Hash Chaining

The manifest advertisement also carries the Content Object Hash of the manifest's first segment:

(1) `/prefix/adv/<ManifestHash>`
`/<ContentObjectHash>`

`<ContentObjectHash>` is the Content Object Hash of `/prefix/data/<ManifestHash>/s0`. This would allow the network to disambiguate all potential segment 0's of the manifest to the one given by the content object hash. Inside each content object of the manifest is the hash of the next manifest segment. This allows secure chaining of manifest segments from the initial advertisement.

Fig. 1 illustrates the hash chaining approach. We have simplified the hash codes for 16-bits represented as four hexadecimal digits for ease of illustration. An initial manifest has a hash of 0x1234. The producer creates the content object

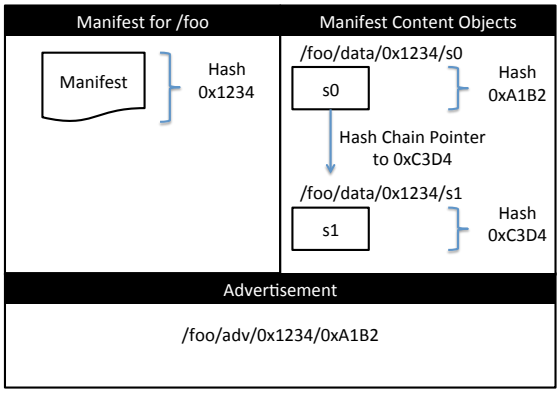

Figure 1. Hash Chaining manifests

representation of the manifest, which results in two objects. Working backwards from the final object, it inserts the content object hash of the next object in to the previous object in a distinguished field. Once it finishes with the first object, it knows the first hash of the hash chain. In this example, the final content object has a Content Object Hash of 0xC3D4, which it puts in the hash chain pointer of the previous object. The first object has a Content Object Hash of 0xA1B2, which covers the content object and the hash chain pointer. Therefore, the advertisement is `/foo/adv/0x1234/0xA1B2`.

If each party in the sync network has a different identity (keyid), then each party would produce a unique hash chain, even for the same manifest. This means that all the advertisements for Name 1 (above) are unique and one would not achieve interest aggregation at the forwarder. However, if a node already knows `<ManifestHash>` it would not need to retrieve each instance of it, so long as it has at least one instance from a trusted source.

It may be that an attacker fabricates `<ManifestHash>` and floods it with one or more fabricated content object hashes under it. There is no defense against this apart from admission

control or secure routing not pointing to the attacker for the prefix. A client device would need to retrieve the first segment of each fabricated advertisement to look at the keyid and determine if its an acceptable participant.

It may be that an attacker uses a true `<ManifestHash>` but fabricates the `<ContentObjectHash>`. There is no defense against this apart from admission control or secure routing not pointing to the attacker for the prefix. A client device would need to retrieve the first segment of each advertisement to look at the keyid and determine if its an acceptable participant. It may stop iteration after the first acceptable advertisement and follow its hash chain.

Because a node must follow a hash chain, pipelining the download is limited by the fanout of the hash chain.

1.1 Hash Chain Convergence

If two nodes produce different hash chains for the same manifest, they can use a distributed election to pick the one hash chain both use, thus reducing the multiplicity of content object hashes used to describe one manifest. For example, the hash chain with the largest hash value should be used. In this example, the node with a lower hash value would retrieve the first segment of the larger hash chain. If it is a valid publisher, it transfers the entire hash chain then begins advertising the larger hash chain. If it is not a trusted publisher, it continues advertising its hash.

A client, seeing multiple advertisements for a manifest would prefer the largest content object hash first. If that hash chain is not a trusted publisher, it should try the other chains, perhaps in order or perhaps in a random order to avoid an attacker front-loading an attack.

2. Secure Catalog

Rather than advertise the Content Object Hash of the first segment of the manifest, a node may advertise the name of a secure catalog that then enumerates all segments. The secure catalog may be segmented too. This allows a faster ramp-up for pipelining a download because a device may retrieve many more segments of the catalog after one round trip.

The content objects of the manifest could be unsigned, and thus identical for every publisher with the same manifest. The only difference is the signature of the secure catalog.

If the secure catalog of the manifest is too large for a single content object, the subsequent objects after the first may be unsigned and identical between publishers. Only the first segment of the secure catalog would contain publisher specific information, such as a signature and timestamps.

Fig. 2 shows one method to use a secure catalog. The sync manifest has hash 0x1234. A system breaks the manifest in to two content objects, with hashes 0x2222 and 0x3333. There is no publisher-specific data in the content objects and they are unsigned. The system creates a secure catalog with entries that point to the 0x2222 and 0x3333 content objects. The catalog is signed and the final object has a Content Object Hash of 0xFEDC. The resulting advertisement has the name `/foo/adv/0x1234/0xFEDC`.

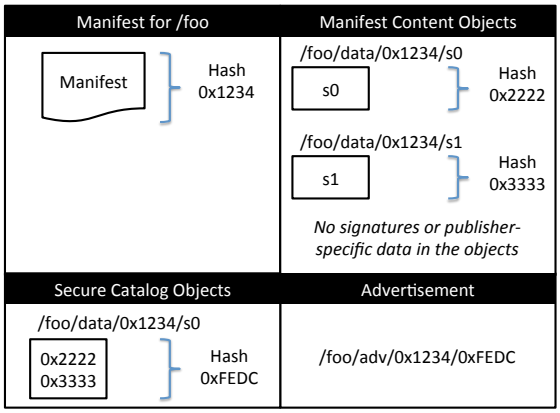

Figure 2. Secure Catalog manifest

Secure catalogs from multiple publishers for the same manifest may use the same technique of Sec 1.1, to converge on one hash chain via a distributed election. One advantage of the secure catalog approach is that the content objects of the catalog are identical between all publishers, so the distributed election is only for the secure catalog name to use. The contents of the catalog should be identical between all publishers for the same manifest hash.

3. Conclusion

We have described two methods to secure Synchronization with Exact Match Names. The first method modifies a manifest hash advertisement to include the Content Object Hash of the first segment of the manifest. Each segment, except the last, contains a hash chain pointer to one or more subsequent objects. The second method uses a secure catalog. The manifest hash advertisement contains the Content Object Hash of the first segment of the secure catalog. The secure catalog then contains hash pointers to subsequent segments of the catalog and content object hashes of the constituent segments of the data.

We also described a method where nodes use a distributed election to pick one out of many equivalent hash chains, so advertisements will converge to one name for a given manifest hash even with multiple manifest publishers.