CCNx 1.0 Elements of Trust

Ersin Uzun[1]*, Cesar Ghali[2], Gene Tsudik[2]

Abstract
In contrast to today's IP-based host-oriented Internet architecture, Information-Centric Networking (ICN) emphasizes content by making it directly addressable and routable. CCNx is an instance of ICN designed by PARC as a reference implementation. By opportunistically caching content within the network (in routers) and providing security at the network layer, CCNx appears to be well-suited for large-scale content distribution, internet of things and for meeting the needs of increasingly mobile and bandwidth-hungry applications that dominate today's Internet.
One key feature of the original CCNx design is the requirement for each content object to be digitally signed by its producer. Thus, CCNx should be, in principle, immune to distributing fake (aka "poisoned") content. However, in practice, this poses two challenges for detecting fake content in CCNx routers: (1) overhead due to signature verification and certificate chain traversal, and (2) lack of trust context, i.e., what public key(s) is/are trusted to verify which content.
In this document, we introduce a trust management architecture for CCNx, elements of which we construct while carefully justifying specific design choices. This document represents the initial effort towards comprehensive trust management for CCNx v1.0.

Keywords
Content Centric Networks – Network Level Trust Management

[1] *Palo Alto Research Center*
[2] *UC Irvine*
*__Corresponding author__: ersin.uzun@parc.com

Contents

1. Introduction

The Internet usage model has changed considerably over the last two decades. Limitations of the current Internet are becoming more pronounced as network services and applications become increasingly mobile and data-centric. In recent years, a number of research efforts have sprung up aiming to design the next-generation Internet architecture. Some are based on the notion of Information-Centric Networking (ICN) which emphasizes efficient and scalable content distribution. CCNx is one such effort lead by PARC.

One of the main tenets of CCNx is to name content, instead of communication end-points. It also stipulates in-network content caching, by routers. To secure each content, the original CCNx design requires it to be (cryptographically) signed by its producer. This way, globally addressable and routable content can be authenticated by anyone, which allows CCNx to decouple trust in content from trust in entities that store and disseminate it. CCNx entities that request content are called *consumers*. A consumer is expected to verify content signatures in order to assert:

- *Integrity* – a valid signature (computed over a content hash) guarantees that signed content is intact;
- *Origin Authentication* – since a signature is bound to the public key of the signer, anyone can verify whether content originates with its claimed producer;
- *Correctness* – since a signature binds content name to its payload, a consumer can securely determine whether delivered content corresponds to what was requested;

Although any entity can verify the authenticity of any content, it could be a challenge for CCNx routers. This is not only because of the overhead stemming from the actual cryptographic verification of the signature itself but also due to two other, more important, reasons:

1. First, a router must be aware of the specific trust model for each content-producing application. Although many applications use a similar hierarchical trust model, it is clear that they will not adhere to a uniform one. Some

will use trust hierarchies, while others might adopt a flat peer-based trust model or hybrid versions thereof. Furthermore, CCNx-supported applications are very likely to be a dynamic set – new ones will be added and older ones might be phased out. Also, a given application's trust model might not be fixed.

2. Meanwhile, depending on the trust model of a given application associated with a particular content, a router needs access to (and thus might need to fetch) multiple public key certificates (or similar structures) in order to trust the public key that verifies a content signature. For example, if an application uses a hierarchical PKI, an entire root-to-leaf path might have to be traversed and all intermediate certificates would need to be separately verified. This would need to include ancillary activities for each such certificate, i.e., expiration and revocation checking.

These issues can greatly complicate trust management/enforcement in CCNx routers. One easy alternative – adopted by NDN, an academic branch out from and earlier CCNx design – is to make it optional for routers to verify content signatures. Unfortunately, this decision leaves NDN vulnerable to content poisoning attacks on router caches. To make matters worse, NDN does not provide any definitive mechanism for a consumer to request genuine desired content. Instead, a consumer that receives fake content can explicitly exclude the latter (by referring to its hash) in subsequent requests. This does not guarantee eventual success, due to the potentially unbounded number of fake content objects sharing the same name.

On the other hand, CCNx 1.0 has abandoned selectors and such exclusion capabilityin its core protocol design due to its obvious penalty on forwarding performance and its ineffectiveness in helping consumers to acquire the content they want. This also means that CCNx 1.0 requires an efficient way to enforce trust at the network layer so that consumer will get a content they would trust on their first ask. In this document, we analyze the anatomy of content poisoning attacks that strongly motivates network level trust management/enforcement and postulate some intuitive goals for CCNx routers to support it. We then present simple rules that allow all parties (consumers, producers and routers) in CCNx 1.0 to mitigate content poisoning and trust enforcement at the network layer, while minimizing trust-related complexity for routers.

2. CCNx Overview

Unlike IP which focuses on end-points of communication and their names/addresses, CCNx ([5, 8]) emphasizes content and makes it named, addressable and routable at the network layer. A content name is composed of one or more variable-length components opaque to the network. Component boundaries are explicitly delimited by "`/`" in the usual path-like representation. For example, the name of CNN home-page content for May 20, 2013 might be: `/cnn/news/2013may20/index.htm`. Large content can be split into segments with different names, e.g., fragment 37 of Alice's YouTube video could be named: `/youtube/alice/vids/video-749.avi/37`.

CCNx communication mainly adheres to the *pull* model and content is delivered to consumers only following an explicit request. [1] There are two types of packets in CCNx: interest and content. A consumer requests content by issuing an *interest* packet. If an entity can "satisfy" a given interest, it returns a corresponding *content* packet. Content delivery must be preceded by an interest. If content C with name n is received by a router with no pending interest for n, C is considered unsolicited and is discarded. Name matching in CCNx 1.0 is exact match (unlike some other architectures and previous CCNx design, where it is prefix based). For example, an interest for `/youtube/alice/video-749.avi` can only be satisfied by content named `/youtube/alice/video-749.avi`.

CCNx content includes several fields. In this paper, we are only interested in the following:

- `Signature` – a public key signature, generated by the content producer, covering the entire content, including all explicit components of the name and the public key needed to verify it.
- `Name` – a sequence of explicit name components followed by an implicit digest (hash) component of the content that is recomputed at every hop. This effectively provides each content with a unique name and guarantees a match with a name provided in an interest.
- `PublisherPublicKeyDigest (PPKD)` – an SHA-256 digest of the public key needed to verify the content signature.
- `Type` – content type, e.g., data, encrypted content, key, etc.
- `Freshness` – a recommendation of the lifetime of the content in the cache.
- `KeyLocator` – the public key (in the form of a raw key or a certificate) required to verify the signature.

Each content producer is required to have at least one public key, represented as a *bona fide* named content of `Type`$=$ *key*, signed by its issuer, e.g., a certification authority (CA).[2].

A CCNx interest includes the following fields:

- `Name` – name of requested content.
- `PublisherPublicKeyDigest (PPKD)` – the SHA-256 digest of the publisher public key. If this field is present in the *interest*, a matching content objects must have the same digest in its `PPKD`.

There are three types of CCNx entities[3]: (1) *consumer* – an entity that issues an interest for content, (2) *producer* – an entity that produces and publishes (as well as signs) content, and (3) *router* – an entity that routes interest packets and forwards corresponding content packets. Each entity (not just routers) maintains the following three components [2]:

- *Content Store* (CS) – cache used for content caching

[1] Push model is possible in CCNx via interest packets, but it is not the preferred model for mainstream applications.

[2] Recall that CCNx is agnostic as far as trust management, aiming to accommodate peer-based, hierarchical and hybrid PKI approaches.

[3] Note that a physical entity (a host, in today's parlance) can be both consumer and producer of content.

and retrieval. From here on, we use the terms *CS* and *cache* interchangeably. Recall that a recommended time to cache each content is specified in the freshness field.

- *Forwarding Interest Base* (FIB) – routing table of name prefixes and corresponding outgoing interfaces used to route interests. CCNx does not mandate any routing protocol. Forwarding is done via longest-prefix match on names.
- *Pending Interest Table* (PIT) – table of outstanding (pending) interests and a set of corresponding incoming and outgoing interfaces.

When a router receives an interest for content named *n* (which is not in its cache), and there are no pending interests for the same name in its PIT, it forwards the interest to the next hop(s), according to its FIB. For each forwarded interest, a router stores some amount of state information, including the name in the interest and the interface on which it arrived. However, if an interest for *n* arrives while there is already an entry for the same content name in the PIT, the router collapses the present interest (and any subsequent interests for *n*) storing only the interface on which it was received. If and when content is returned, the router forwards it out on all incoming-interest interfaces and flushes the corresponding PIT entry. Since no additional information is needed to deliver content, an interest does not carry any *source address*. (If a content fails to arrive before some router-determined expiration time, the router can either flush the PIT entry or attempt interest retransmission over the same or different interfaces.) An CCNx router's cache size is determined by local resource availability. Each router unilaterally determines which content to cache and for how long. Upon receiving an interest, a router first checks its cache to see if it can satisfy the interest locally. Therefore, CCNx in general (and interests, in particular) lacks any notion of a *destination address*, since cached content can be fetched from any CCNx entity. Producer-originated content signatures allow consumers and routers to authenticate received content, regardless of the entity serving it.

3. Problem of Content Poisoning

CCN's key design objective is efficient and scalable distribution of content. This is facilitated by routers opportunistically caching content. Whenever an CCNx router receives an interest for a name that matches some content in its cache, it satisfies the interest with that content. However, the match of only the name does not mean the delivered content is guaranteed to be authentic. That is because there might be many content objects sharing the same name. A consumer would detect it if a fake content is received since it is required to verify signatures of all returned content and assumed to have the necessary application-specific trust context to decide which public keys to trust. However, routers should not be assumed to have that context as discussed above and even if they mechanically verify signatures, this doesn't solve the problem.

This would lead to a fundamental problem of network delivering consumers content that they would discard. The recourse for a consumer to issue another interest that specifically excludes the unwanted content as in some ICN architectures is not the solution either due to the unbounded number of unwanted content. Even if the exclusion technique were to be used strictly for flagging poisoned content, the result would still be undesirable, for the following reasons:

The entire notion of consumers (i.e, end-systems or hosts) informing routers about poisoned content is full of pitfalls. Suppose a consumer complains to a router about specific content. If this is done without consumer authentication (whether via an interest, e.g, using exclusion, or via a separate packet type), the router would have two choices: (1) immediately flush referenced content from its cache, or (2) verify the content signature and flush content only if verification fails. The former (1) is problematic, since it opens the door for anyone to cause easy removal of popular content from router caches, which can be considered as a type of a denial-of-service attack. Even if this were not an issue, there would remain a more general problem: as noted in [3], the adversary mounting a content poisoning attack could continue *ad infinitum* to feed new invalid content in response to interests that exclude previously consumer-detected invalid content. The second option (2) is also problematic, because, besides the cost of verifying a signature (which can lead to a denial-of-service attack), it brings back the problem of routers having to understand potentially complex trust semantics of many diverse content-producing applications.

Another possibility is to require consumers to authenticate themselves when complaining about poisoned content. This would entail signing the interest (or another new packet type) that complains about allegedly bad content. One unpleasant privacy consequence is that the signer (consumer) would be exposed by the signature, since it would need to be bound to a public key, contained in a certificate. (This certificate would have to be communicated along with each complaint message, and auxiliary information that the router would need to trust the certificate.) More generally, signing would violate one of the key elements of CCNx architecture – consumer opacity. Recall that producers sign content while consumers do not sign interests, or any other messages.

Another reason why consumer signing of "complaint" messages is a bad idea is because it can be abused to mount DoS attacks on routers by flooding them with junk complaints and forcing expensive signature verification.[4] Note that, even if the router successfully authenticates a consumer complaint, this is no guarantee that the accused content is fake; in order to be sure, the router would have to verify the content signature as well. Moreover, authentication of consumers by routers would require identity management and verification systems to be in place at the network layer, thus adding significant overhead.

Finally, the preceding discussion applies not only for con-

[4] The same type of attack does not work with flooding of routers with junk content since a router would only attempt signature verification of incoming content for which it has a pending interest entry in its PIT.

tent *cached* by routers. Since CCN only recommends, and does not mandate, content caching, it is entirely *legal* for a router not to cache some, or all, content it forwards. If a router does not cache *C*, then complaining about *C* being fake is clearly useless. At this point, it becomes clear that dealing with fake content requires trust enforcement at the network layer in CCNx architecture. In the remainder of this paper we postulate some simple rules that pave the way towards the elements of practical network-layer trust management for CCNx.

3.1 Goals

As a first step in addressing the content poisoning problem, it is necessary to recognize the obvious, i.e., that **network-layer trust management and content poisoning are inseparably tied to each other**. Since content is the basic unit of network-layer "currency" in CCNx, trust in content (and not in its producers or consumers) is the central issue at the network layer.

Second, trust-related complexity (activities, state maintenance, etc.) must be minimized at the network layer. Specifically, as part of establishing validity of content, **a router should not: fetch public key certificates, perform expiration and revocation checking of certificates, maintain its own collection of certificates, or be aware of trust semantics of various applications**.[5]

On a related note, **a router should verify at most one signature per content**. This upper-bounds the heavier part of content-related cryptographic overhead; the other part is computing a content hash. Ideally, a router would not perform any signature verification at all. However, as discussed below, this might be possible for some, yet not all, content. Also, although verifying a signature given an appropriate public key is a mechanical operation, a router would still need to support multiple signature algorithms as it is improbable that all applications would adopt the same signature algorithm.

The above discussion implies that CCNx entities other than routers, i.e., **producers and consumers of content, should bear the brunt of trust management**.

4. The Interest-Key Binding Rule

Our approach to network-layer trust – that adheres to all desired goals outlined above – is based on one simple rule, that we denote as *Interest-Key Binding (IKB)*:

IKB: An interest must reflect the trust context of the consumer in a form enforceable at the network layer.

Recall that CCNx interest format (Section 2) includes two optional information `PPKD` field and the `digest` of the requested content object as the last component of the name. Our approach suggests to have at least one of them in each interest packet.

[5]This is separate from trust management for routing.

An CCNx public key is usually distributed via a special type of content in the form of a certificate signed by the issuing CA. Each certificate contains a list of all name prefixes that it is authorized to sign/verify. The name of the certificate-issuing (content-signing) CA and the name of the key contained in a certificate (content) are not required to have any common prefix. This is part of CCNx's philosophy of leaving trust management up to the application, e.g., signed content *C* can be verified with public key *PK* with *C* and *PK* having no common prefix requirement. For instance, content containing the public key `/cnn/usa/web/key` could be issued and verified by the key `/verisign/key`. Of course, an application is free to impose all kinds of restrictions, as long as routers remain oblivious.

4.1 Implications for Producers and Routers

We now examine the implications of IKB on content producers and routers, respectively.

For content producers, IKB has very few consequences. In particular, it simplifies construction of content by asking a producer to include the public key itself in the `KeyLocator` field of content. The exception to this rule are content objects that will be consistently requested by using self-certifying names (such as content distributed via secure catalogs as described in Section 5), where no key information is necessary.

For routers, the implications of IKB are overwhelmingly positive due to the simplification. First, a router needs to perform no fetching, storing or parsing of public key certificates, as well as no revocation or expiration checking. All such activities are left to consumers.

Upon receiving a content and identifying its corresponding PIT entry (corresponding to one or more pending interests) a router does one of the following:

- If the PIT entry specifies a `PPKD`, it simply hashes the public key from the content `KeyLocator` field of the received content and checks whether it matches. If so, the content signature is verified and, if found valid, the content is forwarded and optionally cached. Otherwise, the content is discarded.
- If the PIT entry specified a self-certifying name (SCN) by providing a `digest` as its last component, the router hashes the content (along with its name, except for the last implicit digest component) and checks whether all components of the name including the `digest` matches. In case of a match, the content is forwarded and optionally cached. Otherwise, the content is discarded.

Note that routers are required to do only one signature verification operation in the first case, and the latter case does not require any signature verification operation within the network. For a favorable traffic composition, where most interests use SCNs, this would allow an off the shelf router today to reach multi gigabit forwarding speeds by processing SCN-based traffic on fast and signature verification on slow path. As discussed in Section 5, such favorable traffic

composition can easily be accessible via the use of secure catalogs.

4.2 Implications for Consumers

For consumers, IKB does not increase complexity; it only forces them to codify desired consumer behavior.

The most immediate IKB consequence for a consumer is the need to either **(1) obtain and validate the producer's public key before issuing an interest for any content object from that producer.**, or **(2) obtain the digest (i.e., SCN) of a content before issuing an interest for it**. At the first glance, either might appear to be an example of the proverbial "chicken-and-egg" problem. However, we show below that this is not the case.

For the first case, a consumer that wants to fetch certain content C is doing so as part of some CCNx application, APP_C. We assume that a consumer must have already installed this application. APP_C must have a well-defined trust management architecture that is handled by its consumer-side software. However, the remaining question is: how to bootstrap trust and how to obtain initial public keys?

We consider three non-exclusive alternatives:

(A) One possibility is that APP_C client-side software comes with some pre-installed root public key(s), perhaps contained within self-signed certificates. Without loss of generality, we assume that there is only one such key – PK_{root}. Armed with it, a consumer can request lower-level certificates, by issuing an interest referencing the hash of PK_{root} in the `PPKD` field. [6]

(B) Alternatively, we could envision a global CCNx Key Name Service (KNS), somewhat akin to today's Domain Name Service (DNS). In response to consumer-issued special interests referencing public key names (and/or name prefixes), KNS would reply with signed content that would contain one or more public key certificates (i.e., embedded content) corresponding to requested names.

(C) Another similar possibility is to imagine a global search-based service, i.e., something resembling today's Google. A consumer would issue a search query (via an interest) to the search engine which would reply with signed content representing a set (e.g., one page at a time) of query results. One or more of those results would point to content corresponding to the public key certificate of interest to the consumer.

In cases (B) and (C), consumers would still need to somehow securely obtain the root public keys for KNS and the search engine, respectively. This can be easily done via (A).

Once a public key certificate for a publisher is retrieved, obtaining SCNs for content objects from that producer becomes trivial. Interests with `PPKD` can be used to request signed lists of SCNs (i.e, content objects listing SCNs for chunks of a file).

[6] If APP_C comes with several root public keys, it would need to send multiple simultaneous interests referencing the hash of each key in `PPKD` if it does not know which key would have signed the lower-level certificate.

4.3 Security Arguments

We now return to the original motivation for this work – mitigation of content poisoning attacks. We need to show that global adherence to the IKB rule leads to security against content poisoning.[7]

If we assume that:

1. The consumer requesting content C is not malicious.
2. Each router R that is one hop away from the consumer is not compromised.
3. The links between a consumer and its adjacent routers are not compromised.

We can briefly argue security by contradiction: Suppose that a consumer receives some fake content C from R. Let *Int* denote the interest (issued earlier by that consumer) that was satisfied by C. According to IKB, *Int* must either contain the digest of a public key of producer P in its `PPKD` field or the digest of the content object as the last component of the name.

For the former case, let PK denote the public key for P. Consequently, R must have made sure that: (1) C is signed with a public key PK' with a hash matching `PPKD` of *Int*, meaning that $H(PK') = H(PK)$ and (2) the signature itself is correct, i.e., valid. Also, since R is not malicious and all communication between R and the (also not malicious) consumer is secure, the only remaining possibility is a hash collision, i.e., $PK' \neq PK$ while $H(PK') = H(PK)$. However, collisions in cryptographic hash functions are assumed to occur with negligible probability. It is easy to see that a similar argument also follows for the latter case: receiving a fake content would only be possible if the digest of the fake content is equal to the original content implying a hash collision. This is due to the fact that each router calculates the `digest` and verifies its match before forwarding content objects.

This does not yet conclude our security discussion. As noted in [3], content poisoning attacks can originate with malicious routers. What happens if a malicious router R' feeds poisoned content C' to its non-malicious next hop neighbor R, towards some consumer(s)? Since R is honest and implements IKB, before forwarding and (optionally) caching C', it verifies, as before, that either the `digest` in the corresponding PIT entry matches the content hash or the signature of C' is successfully verifiable using PK that matches the hash in the corresponding PIT entry, i.e., the value of the `PPKD` field of the original interest *Int* that triggered creation of this PIT entry.

5. Optimizations

As mentioned before, IKB rule implies that routers will perform at most one signature verification using the public key provided (by the producer) in the content and specified (by the consumer) using the `PPKD` field in the interest. Instead of including the public key in the content, it could be directly

[7] Note that, as mentioned in Section 1, cache pollution attacks are an entirely different matter.

included by the consumer in the interest. This would require storing the key alongside the interest in the PIT entry, to be used later for signature verification of the content. Since it is fair to assume that cache entries have longer lifetime than PIT entires, this approach can be beneficial in terms of storage. Its main drawback, however, is that the current interest format would need to be modified to include public keys and average size for a PIT entry would increase.

For backbone routers that process and forward tens of gigabits per second, performing even a single signature verification imposes a huge overhead. One approach to overcome this problem is to take advantage of the network structure. The current Internet is divided into Autonomous Systems (AS-s), each representing an administrative entity. In this architecture, only border routers of consumer-facing AS-s might implement the IKB rules by verifying signatures and/or SCNs of all received contents. Alternatively, each router in an AS might probabilistically verify signatures on a subset of packets. One drawback of these approaches are that fake content could still be cached by some routers. However, in either method, fake content would be detected and discarded with high probability, before reaching the consumer.

A more dramatic optimization (which is employed in the current implementation of CCNx 1.0) is to only require the router that responds to an interest with `PPKD` from its cache verify the signature of that content. Given the significant majority of forwarding routers will be benign, even such a dramatic optimization will still provide effective security against content-poisoning attacks. This is due to the fact that fake content will not be served from router caches in response to benign interests, and as long as a robust routing is in place, such interests will reach to the correct nodes that can authoritatively answer with an authentic content.

For favorable traffic mix, where most content is requested using SCNs without requiring any signature verification at routers, we advocate the use of *catalogs* (a.k.a., Aggregated Signing Objects). Technically, a catalog is an "authenticateable" data structure providing a list of self-certifying names. This list consists of references to content objects containing data, public keys, or even other catalogs. The structure of catalogs could be application-specific and can vary from a single a list of self-certifying names, to several lists in different content objects forming a Merkle tree [7]. For securely fetching catalogs, consumers can use the `PPKD` field of the interest, as discussed in Section 4.

6. Proposed Model in Practice

The goal of designing and implementing a new network architecture, such as CCNx, is to have a replacement candidate for the current IP-based network. In order for this migration to be smooth and successful, CCNx should be able to adapt to application specific requirements, such as trust. In this section we discuss how the aforementioned trust model and its optimizations could be applied in practice. We start by identifying different CCNx traffic types.

6.1 Content Distribution

This type of traffic corresponds to client-server communication in today's network and accounts for well over 90% of the current Internet traffic[4]. Since most requested content is static, creating secure catalogs is straightforward. For their part, consumers request catalogs and then use included self-certifying names to request desired content. We consider two common examples of content distribution traffic:

Audio/Video Streaming: A typical audio/video is a large content split into several segments with different names (as mentioned in Section 2). If a catalog containing the self-certifying names of all the segments can be provided, consumers can use these names in subsequent interests to retrieve all segments of the content.

Internet Browsing: We anticipate that most HTML files would fit into a single content object [9, 1]. A typical HTML file contains reference links to other static and dynamic content, such as images, audio or other HTML pages (sub-pages). While rendering HTML files, Internet browsers parse all reference links and download corresponding content. Therefore, if an HTML file uses self-certifying names as references, it can be viewed and treated as a secure catalog. Of course, self-certifying names can only be used for static content, since the hash of dynamic (e.g., generated upon request) content cannot be known *a priori*.

Internet browsing provides a good example of content that can be requested by either supplying `PPKD` or via self-certifying names. Suppose that a web page *A* contains a reference link to sub-page *B* and this link is expressed using a SCN. Once a consumer requests and obtains page *A*, the client browser can request *B* using the SCN within *A*. Whereas, other consumers might wish to directly request page *B* (not as part of *A*) using its producer's public key digest. Moreover, SCNs cannot be applied to HTML pages with circular references, e.g. page A has a link to page B, and page B has a link to page A. Thus, both links (with and without SCNs) are needed to support circular references on the Web.

6.2 Interactive Traffic

The second type of traffic in CCNx is interactive communication where content is generated on demand. All traffic generated by applications such as voice and video calls, SSH sessions, and gaming fit into this category. Such applications benefits from caching only in case of packet loss, where reissued interests for retransmission are (likely) satisfied from the first hop router. This reduces latency and improves quality of service. Since interactive traffic is generated at run-time, building big catalogs in advance might not be feasible. Instead, consumers should request content by specifying the `PPKD` field in interest messages or utilizing dynamically generated small catalogs whenever small delays are tolerable.

7. Conclusion

In this document, we postulated some intuitive trust management goals needed to support content validation in an ICN network in general, and CCNx in particular. To enable trust enforcement at the network layer and mitigate content poisoning attacks, we presented simple rules that allow all CCNx nodes to determine whether received content is valid. We also presented some overhead-lowering optimization techniques.

References

[1] The average web page has almost doubled in size since 2010. http://www.webperformancetoday.com/2013/06/05/web-page-growth-2010-2013/. Accessed: 2014-01-31.

[2] CCNx 1.0 Protocol Specifications (Rev 2). PARC 2014.

[3] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang. DoS DDoS in named-data networking. In *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, 2013.

[4] Sandvine global internet phenomena report. https://www.sandvine.com/trends/global-internet-phenomena/.

[5] V. Jacobson et al. Networking named content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 1–12, 2009.

[6] J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC Press, 2008.

[7] R. C. Merkle. Method of providing digital signatures, Jan. 5 1982. US Patent 4,309,569.

[8] Named Data Networking project (NDN). http://named-data.org.

[9] S. Ramachandran. Web metrics: Size and number of resources. *Online a t https://developers. google. com/speed/articles/web-metrics*, 2010.

APPENDIX A: Formal Security Argument

Definition 7.1. A hash function $\mathscr{H}$ is *second pre-image resistant*, if for any given x, no probabilistic polynomial-time (PPT) adversary $\mathscr{A}$ can find a value $x' \neq x$ such that $\mathscr{H}(x) = \mathscr{H}(x')$. In other words, $\Pr[\mathscr{H}(x) = \mathscr{H}(x')] \leq \varepsilon(n)$, where $\varepsilon(n)$ is negligible and n is the security parameter. A formal definition of probabilistic polynomial-time adversaries and negligible functions can be found in [6]

Definition 7.2. A signature scheme Π is unforgeable if for any message m, no PPT adversary $\mathscr{A}$ (given a public key PK) can generate a valid signature without knowing the corresponding private key. We denote the success of $\mathscr{A}$ as $\mathscr{A}^{\textbf{forge}}(m) = 1$, i.e., if Π is unforgeable, there exists a negligible function $\varepsilon(n)$ such that: $\Pr\left[\mathscr{A}^{\textbf{forge}}(m) = 1\right] \leq \varepsilon(n)$.

Definition 7.3. For any interest message *Int* with $\mathscr{H}(PK)$ (the digest of the verifying public key for the corresponding content) assigned to the `PPKD` field, and for any $\mathscr{A}$, the CCNx cache poisoning experiment is defined as follows:

Given *Int* as input to $\mathscr{A}$, it outputs a content object C' containing: (1) a public key PK' in the `KeyLocator` field, (2) a digest of this key $\mathscr{H}(PK')$ in `PPKD`, and (3) a signature σ' in the `Signature` field. The output of this experiment is defined to be 1 if one of the following holds:

- $PK \neq PK'$ and $\mathscr{H}(PK) = \mathscr{H}(PK')$,
- or, $PK = PK'$ and σ is valid.

In other words, $\mathscr{A}$ can either violate the second pre-image resistance of $\mathscr{H}$ (we denote this event as **collision** which occurs with some probability p_c and succeeds with $\Pr[\mathscr{H}(x) = \mathscr{H}(x')]$), or forge the signature (we denote this event as **forge** which occurs with some probability p_f and succeeds with $\Pr\left[\mathscr{A}^{\textbf{forge}}(m) = 1\right]$). We denote the success of $\mathscr{A}$ as $\mathscr{A}^{\textbf{pois}}(Int) = 1$.

Theorem 7.4. *Given $\mathscr{H}$, Π (as defined above), $\mathscr{A}$ succeeds in injecting a fake content object C' into a network that abides by the IKB rule with a negligible probability $\varepsilon(n)$.*

$$\Pr\left[\mathscr{A}^{\textbf{pois}}(Int) = 1\right] \leq \varepsilon(n)$$

Proof. We show the above by contradiction:

Assume that $\mathscr{A}$ succeeds in injecting C' with a non-negligible probability. Then, we can construct a reduction $\mathscr{A}'$ (another PPT adversary), that uses $\mathscr{A}$ to break second pre-image resistance of $\mathscr{H}$, or unforgeability of Π:

Adversary $\mathscr{A}'$
1. Is given a hash value x.
2. Creates an interest message *Int* and sets $\mathscr{H}(x)$ as its `PPKD` field value.
3. Runs $\mathscr{A}(Int)$ to obtain C'.
4. Extracts from C' and outputs:
   (a) PK' as a collision with x, if $x \neq PK'$,
   (b) or σ' as a forged signature for C', if $x = PK'$.

We now determine the probability of success of $\mathscr{A}'$. Whenever either **collision** or **forge** event occurs $\mathscr{A}'$ succeeds. Therefore,

$$\begin{aligned}\Pr\left[\mathscr{A}' \text{ succeeds}\right] &= \Pr[\textbf{collision} \cup \textbf{forge}] \\ &= p_c \cdot \Pr\left[\mathscr{H}(x) = \mathscr{H}(PK')\right] \\ &+ p_f \cdot \Pr\left[\mathscr{A}'^{\textbf{forge}}(C') = 1\right] \\ &> \varepsilon(n)\end{aligned}$$

The last inequality holds because $\mathscr{A}'$ succeeds with the same probability as $\mathscr{A}$, which is non-negligible. If the result of adding two functions is non-negligible, at least one of them must be non-negligible [6]. Moreover, since both p_c and p_f cannot be exponential functions, then either
$\Pr[\mathscr{H}(x) = \mathscr{H}(PK')] > \varepsilon(n)$ or $\Pr\left[\mathscr{A}'^{\textbf{forge}}(C') = 1\right] > \varepsilon(n)$.
□