# Data Authentication for NDN Using Hash Chains

Tamer Refaei, Mark Horvath, Michelle Schumaker, Creighton Hager
The MITRE Corporation
McLean, VA 22102

***Abstract*—Named Data Networking (NDN) is considered to be a viable replacement of the traditional IP networking for the next generation Internet architecture. NDN is a data centric, rather than host centric, approach to discovery and retrieval of information distributed across a network. This paradigm shift results in a host of new challenges, one of which is authentication of data and data-sources. NDN requires each content producer to digitally sign each data packet it produces. While the use of digital signatures certainly accomplishes data and data source authentication, it imposes significant overhead on the whole network. In this paper, we propose a novel data and source authentication mechanism that relies primarily on hashing. The mechanism utilizes hash chaining to ensure that unauthorized modification, insertion, or deletion of data can be easily detected by routers and content consumers.**

***Keywords—Named Data Networking, Hash Chains, Data Authentication***

## I. INTRODUCTION

Named Data Networking (NDN) is a proposed architecture for the next generation Internet. NDN considers the shortcomings of the traditional IP-based networks as well as the evolution of communication paradigms over the Internet. Realizing that the most prominent communication paradigms in today's Internet focus on data access (e.g. peer-to-peer, streaming, social networking), NDN shifts the networking paradigm away from the traditional host-centric model (adopted by IP) to a data-centric model. By changing the semantics of the network from "deliver a packet to a given destination address" to "fetch data identified by a given name" [1], NDN makes the networking paradigm more organic to today's most prominent communication paradigms.

Data and data source authentication are built natively into the design of NDN. Data available for distribution is explicitly named and signed by its producer. Routers and content consumers can then validate the signatures to ensure data and data source authenticity. While the use of a digital signature (which relies on public key encryption) certainly accomplishes the desired goals of ensuring data and data source authentication, it imposes significant computational overhead on the network. Moreover, the lifetime of data may well exceed the lifetime of public key certificates [2].

In this paper, we address this problem by introducing a hash chaining based mechanism for data and data source authentication. The mechanism leverages hash chains to interconnect chunks of data to ensure that any unauthorized modification, insertion, or deletion of any chunk is reflected in the chain. This allows routers and data consumers to easily and efficiently detect unauthorized attempts to corrupt the data and discard the affected chunks of data from their caches. A single digitally signed data chunk suffices to verify the authenticity of the data and the data source across the entire chain.

Our contribution in this paper is two-fold. First, we illustrate how the currently adopted data and data authentication mechanism in NDN exerts significant computation overhead. Second, we show that a hash chaining mechanism can be utilized to realize secure and efficient data and data source authentication in NDN.

This paper is organized as follows. In section II, we provide an overview of NDN as well as a summary of related work. In section III, we describe our mechanism and a prototype that we built using the *ndn-cxx* library [3]. In section IV, we evaluate the performance of our mechanism in comparison to the standard NDN signature based mechanism and the manifest mechanism in [4]. We conclude this paper in section V.

## II. BACKGROUND

### A. NDN Overview

The motivation behind NDN is to move away from today's IP-based and host centric networks and towards more dynamic and intelligent distribution networks. By naming data, certain network functions are introduced such as automatic in-network caching that in turn supports content distribution, multicast, and delay-tolerant networking. Clips of a video stream, position location information (PLI), or text messages are some examples of data objects that can be cached at a router and then fetched at a later time based on interest by users.

NDN is similar in nature to peer-to-peer (P2P) networking and publish-subscribe software architectures. Routing and forwarding packets in NDN are performed on names rather than IP addresses, eliminating the need for IP address changes in a mobile environment, as the data object names on a router remain constant. This distribution strategy mitigates the risk of data loss due to network disruptions and alleviates traffic loads on network nodes.

NDN communication fundamentally relies on the communication between three entity types: content consumers, content producers, and routers [5]. To enable communication, NDN implements two types of packets [1]: Interest and Data. Named content are requested via Interest packets. A router that receives an Interest packet and has a corresponding data object cached in its Content Store can respond with a Data packet. By design, one Interest packet matches one Data packet to maintain flow balance. For large content, the content producer



978-1-4673-7194-0/15/$31.00 ©2015 IEEE

may segment the data into a manageable number of Data packets such that each Data packet (chunk) can be requested by a separate Interest packet. A suggested NDN naming convention for segmentation is provided in [6].

In the NDN architecture, data and source authenticity are enabled through the use of cryptographic signatures. While Interest packets are unsigned, Data packets are signed by the content producer. This allows an NDN entity to verify the authenticity of the data and the data source.

### B. Related Work

A representative walkthrough of the NDN signature scheme and a mechanism of implementing a signature-based trust model are demonstrated in [7]. In addition to the native signature scheme provided by NDN, several additional mechanisms of ensuring data and data source authentication have been investigated.

Recognizing the computation cost of digital signatures, the authentication mechanism introduced in [2] adds a notary entity to the architecture. The notary entity exposes a content producer-signed manifest file, which can be used to validate a specified subset of unsigned content Data packets. The manifest file contains a set of digests, or cryptographic hashes of the content available for distribution. The consumer can validate the signature on the manifest Data packet to authenticate the source, and use the digests inside the manifest Data packet to validate the authenticity of the given set of content Data packets. This mechanism reduces the number of signatures needed in order to fully authenticate the content.

In [4], an alternative mechanism for manifest distribution is offered. In this mechanism, there is no notary entity. Instead, the manifest files are periodically distributed by the content provider to the consumer in a cryptographically signed data packet. This mechanism removes the need for the notary architecture entity and reduces the number of signatures needed in order to fully authenticate the content. The size of the manifest is dependent on the maximum Data packet size. A large Data packet size can accommodate a large number of hashes in a single manifest. Moreover, the number of manifests required to represent named content is a function of the size of the content and the maximum Data packet size (which can be up to eight octets). In [8], a mechanism was introduced that attempts to provide both data integrity as well as access control while limiting the use of digital signatures. However, the authors assume that a content provider has complete identity information of authorized NDN nodes along with their public keys, which may not be possible all the time.

One-way and collision resistant hash functions provide a mean to enable, among other cryptographic applications, verification of data authenticity. An overview of hashing mechanisms is covered in [9]. Hash chains further build on this concept, enabling the creation of a series of one-time use keys that, due to their one-way and collision resistant attributes, cannot be reverse engineered to produce any preceding part of the used hash chain. Hash chain theory and implementation are covered in [10], [11]. Our proposed mechanism expands on these concepts to provide an alternate method for data and source authentication.

## III. Data Authentication using Hash Chains

### A. Scope

Our protocol *only* considers attacks against data and data source authentication. We consider that an attacker can be a rogue content producer or a compromised router. The attacker can have access to any Data packet produced by any content producer. The objective of the attacker is to cause damage to the network through modification, insertion, or re-ordering of Data packets. We assume the existence of a trust model that allows a content consumer to validate signatures of Data packets it receives.

Most of our discussion in this section will focus on non-real-time data (e.g. images). We will discuss applicability to real-time data (e.g. video conferencing) in section IV. For either, our proposed mechanism is applicable to data objects that are too large to be sent in one chunk, and hence needs to be sent out in multiple chunks. For small data objects (a few chunks), the standard NDN mechanism of using a digital signature will suffice.

### B. Design Details

For each data object, the content producer generates a hash chain that is as long as the number of Data packets in the object. For instance, the content producer will generate a chain of 10 hashes for a data object that is composed of 10 Data packets. We denote the Data packets of a data object $M$ as $\{m_1, m_2, \ldots, m_n\}$ and the hash chain as $\{k_1, k_2, \ldots, k_n\}$.

The hash chain can be calculated in one of two ways: a *Backward Chain* and a *Forward Chain.* The design of both chaining approaches limits the use of the computationally expensive public key cryptography operation in favor of the less expensive hashing operation to provide data and data source authentication. Data packets are chained using cryptographic hashing and the chain is sealed with a digital signature. The seal figuratively ties the two loose ends of the chain. We detail the two approaches below and discuss their respective pros and cons in the next section.

- *Backward Chain*

The *Backward Chain* assumes that Data packets arrive in a reasonable order. It is most applicable to network environments where bandwidth is the most valuable resource and can be traded for delayed authentication of Data packets (e.g. tactical network environments). The chain is calculated as follows:

$$(BC\text{-}1) \qquad k_1 = H(m_1 \,||\, k_n)$$

$$(BC\text{-}2) \qquad k_i = H(m_i \,||\, k_{i-1}), \ \forall \ n > i > 1$$

$$(BC\text{-}3) \qquad k_n = H(m_n),$$

where $H$ is a cryptographic hash. Each message $m_i \ \forall \ n > i > 1$ is connected to the chain by including the hash $k_{i-1}$ from the previous message in the chain as part of the calculation of $k_i$. The chain is then sealed by a digital signature $S$ and is disclosed along with the first Data packet. The seal $S$ is calculated as follows:

$$(BC\text{-}4) \qquad S = E(PR_{CP}, H(k_n \,||\, k_{n-1})) \,||\, k_n \,||\, k_{n-1} \,||\, C_{cp},$$

where $E$ is a public key encryption function, $PR_{CP}$ is the private key of the content producer, and $C_{cp}$ is the certificate of the content producer signed by a mutually trusted certificate authority. Only after the generation of the full hash chain can Data packets be sent from the content producer (we will discuss how to relax this requirement for real-time data in section IV). Each Data packet $m_i$ may or may not include its corresponding hash $k_i$. The decision to include or not include the hash value $k_i$ is a trade between communication overhead and authentication delay (we will discuss that in the next section). Fig. 1 shows an example where a hash is included along with each message.

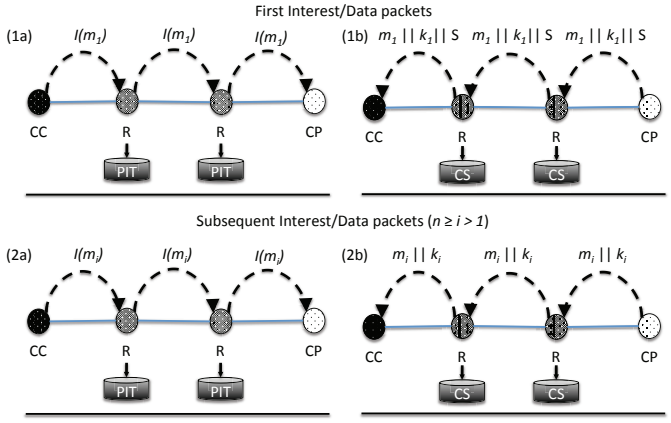


Fig. 1. Propagation of Interest and Data packets for the *Backward Chain*.

- *Forward Chain*

The *Forward Chain* also assumes that Data packets arrive in a reasonable order. It is most suitable for applications that require immediate authentication of Data packets at the consumer side (or router) once they are received. The chain is calculated as follows:

$$(FC\text{-}1) \qquad k_1 = H(m_1 \,||\, k_2)$$

$$(FC\text{-}2) \qquad k_i = H(m_i \,||\, k_{i+1}),\ \forall\ n > i > 1$$

$$(FC\text{-}3) \qquad k_n = H(m_n),$$

where $H$ is a cryptographic hash function. Each message $m_i$ $\forall$ $n > i > 1$ is connected to the chain by including the hash $k_{i+1}$ from the next message in the chain as part of the calculation of $k_i$. The chain is then sealed by a digital signature $S$ and is disclosed along with the first Data packet. The seal $S$ is calculated as follows:

$$(FC\text{-}4) \qquad S = E(PR_{CP}, H(k_n \,||\, k_1)) \,||\, k_n \,||\, k_1 \,||\, C_{cp},$$

where $E$ is a public key encryption function, $PR_{CP}$ is the private key of the content producer, and $C_{cp}$ is the certificate of the content producer signed by a mutually trusted certificate authority. The generation of the full hash chain occurs before any Data packets are sent from the content producer. When a Data packet $m_i$ is sent, it includes its corresponding hash $k_i$ as well as the value of $k_{i+1}$ as shown in Fig. 2.

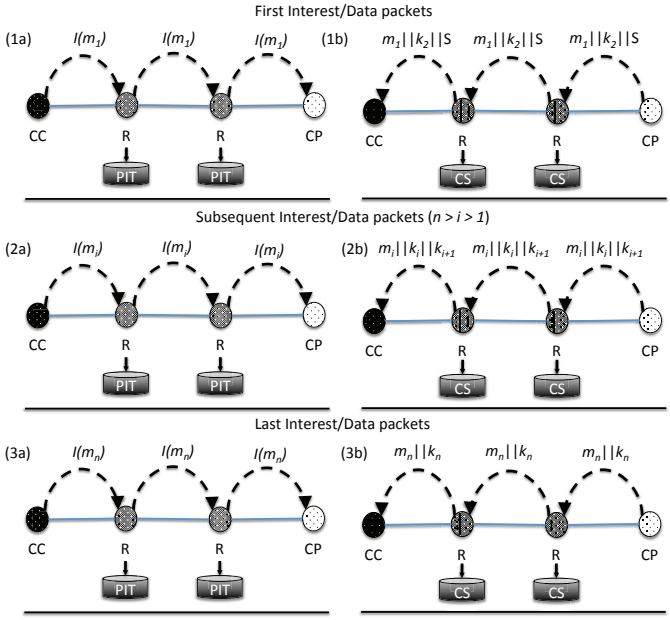


Fig. 2. Propagation of Interest and Data packets for the *Forward Chain*.

## C. Resilience Analysis

In this section we discuss the resilience of the *Backward Chain* approach to modification, insertion, and re-ordering attacks. A similar analysis will hold for the *Forward Chain*.

The *Backward Chain* approach ensures that an unauthorized change in one Data packet propagates forward affecting all subsequent Data packets in the chain. The seal $S$ facilitates the detection of that change. To illustrate the resilience of the mechanism against modification, insertion, and re-ordering attacks, we will discuss each separately:

- *Data modification*: Suppose that an attacker modifies a Data packet $m_i$, where $n \geq i \geq 1$. The modification of $m_i$ results in a modified value of $k_i$ as shown in (BC-1) through (BC-3). If $i=n$ or $i=n-1$, the chain is invalidated immediately since the seal $S$ signs both $k_n$ and $k_{n-1}$ as shown in (BC-4). If $n-1 > i \geq 1$, the attacker will have to advertise for a modified Data packet $m'_i \,||\, k'_i$. The modified hash $k'_i$ will result in the modification of the value of $k_{i+1}$ as shown in (BC-1) and (BC-2). The attacker would then have to advertise for a modified Data packet $m'_{i+1} \,||\, k'_{i+1}$. This pattern continues for all Data packets until $m'_{n-2} \,||\, k'_{n-2}$. Since the value $k'_{n-2}$ affects the calculation of $k_{n-1}$, there will be a mismatch between the calculated $k'_{n-1}$ and the one signed in $S$. Hence, the chain will be invalidated.
- *Data insertion*: Since the last two Data packets are protected by a digital signature, the attacker can only attempt to insert Data packets at position $n-1 > j \geq 1$. In order for the attacker to successfully insert a Data packet $m_j$ in between $m_i$ and $m_{i+1}$, the attacker will need to generate a valid hash $k_j$ for it and also ensure that $k_{i+1}$ remains valid. From (BC-1) and (BC-2), the attacker will have to generate $m_j$ and insure that $k_j = H(m_j \,||\, k_i)$ and also that $k_{i+1} = H(m_{i+1} \,||\, k_j)$. The first part is easy but the

second is infeasible with respect to cryptographic hash functions since it requires finding two different messages $(m_{i+1} \| k_j)$ and $(m_{i+1} \| k_i)$ that produce the same hash value $k_{i+1}$.

- *Data re-ordering*: Data re-ordering will result in invalidation of the chain since the calculation of a hash depends on the hash before it. However, a data consumer can potentially reorder Data packets correctly based on their names.

### D. Authentication Process

In this section we discuss the authentication process of the two hash chaining mechanisms. The authentication process within both mechanisms starts with the receipt of the first Data packet. The first Data packet is critical since it includes the value of *S*. Upon receiving the first Data packet, a content consumer or a router will perform the following steps in order:

(1) *Validate the certificate of the content producer* by ensuring that it is signed by a trusted certificate authority.

(2) *Validate the seal S* by ensuring its integrity (i.e. verify the digital signature on the seal).

The next step after that is handled differently between the *Backward Chain* and the *Forward Chain* approaches. We will discuss each separately.

- *Backward Chain*

In the *Backward Chain,* the consumer can easily validate Data packets $m_n$ and $m_{n-1}$ since their respective hashes $k_n$ and $k_{n-1}$ are signed by the seal. Validating a Data packet $m_i$ $\forall$ $n\text{-}1 > i \geq 1$ on the other hand is possible if and only if $k_i$ is included along with the message and there is a contiguous path along the chain from $k_i$ to $k_{n-1}$ as shown in Fig. 3. This means that the consumer must be in possession of all Data packets from $m_i$ to $m_{n-1}$ along with their respective hashes. The validation process ensures that both (BC-1) and (BC-2) are correct all the way up to $m_{n-1}$. This is due to the fact that a sequence of message modifications by an attacker can only be detected once the end of the chain is reached as discussed in section III.C. As a result, a consumer will be able to validate the entire chain once all the Data packets are received.

- *Forward Chain*

In the *Forward Chain,* the consumer can easily validate Data packets $m_n$ and $m_1$ since their respective hashes $k_n$ and $k_1$ are signed by the seal. Validating subsequent Data packet $m_i$ $\forall$ $n\text{-}1 \geq i > 1$ can happen immediately without the need to complete the chain. The fact that $k_1$ is calculated over $k_2$ together with the fact that $k_1$ is signed validates the value of $k_2$. Accordingly, it is not possible for an attacker to alter $k_2$. Once $m_2$ is received it can be validated right away by verifying that $H(m_2||k_3)$ equal $k_2$. Recall that the value of $k_3$ is disclosed along with $m_2$. Once $k_2$ is verified, that automatically validates $k_3$ (since $k_2$ is calculated over $k_3$). Accordingly, $m_3$ can be immediately validated once it is received as well. This pattern continues as long as Data packets are received in order. Similar to the *Backward chain*, a gap in the chain will only allow for the validation of sub-chains that have a path starting at the seal.

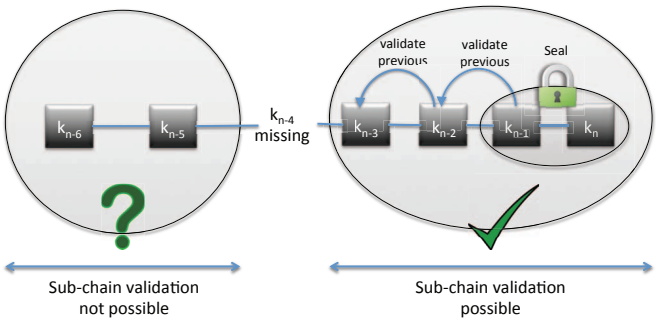


Fig. 3. Backward Chain validation: validation of a contiguous sub-chain is only possible when the sub-chain ends at $k_{n-1}$.

## IV. Performance Analysis

In this section, we compare our proposed hash chaining mechanism to the standard NDN mechanism of including a digital signature in every Data packet as well as the signed manifest mechanism in [4]. We have integrated all three mechanisms into the *ndn-cxx* library in order to compare their performance. In comparing the three mechanisms, we consider computation, communication, and storage overhead.

The evaluation was done using a network of 2 nodes, a producer and a consumer, in CORE [12]. The CORE engine was hosted in an Ubuntu virtual machine that was assigned three 2.4GHz cores on an Intel Xeon E5645 processor and was assigned 8GB of memory. All digital signatures utilized RSA/SHA1. The manifest and hash chain mechanisms utilized SHA256 for hashing. Recognizing the implications of the value of the maximum Data packet size on the performance of the manifest mechanism, we used two different values: 1500bytes (standard Ethernet MTU) and 9000bytes (jumbo Ethernet MTU). We also utilized different data object sizes that ranged from 1.5Mbytes to 15Mbytes. Finally, all results noted were averaged over 100 runs.

- *Computation Overhead*

In comparing the computation overhead of our proposed hash chaining mechanism to that of the standard NDN and the signed manifest mechanisms, we consider the metric *generation time*, which is the time during which the data authentication credentials (i.e. hash chain, digital signature, or manifest) are prepared at the content producer. We also consider the metric *verification time*, which is the time taken to verify these credentials at a router or content consumer.

As shown in Fig. 4, the standard digital signature mechanism performs very poorly in comparison to the hash chain and manifest mechanisms with respect to the generation time. Within the context of NDN, about 15ms is needed to generate a digital signature for a single Data packet in comparison to about 0.015ms to generate a hash value (in other words, generating a single digital signature is equivalent to generating 1000 hashes). On the other hand, the computation time of the hash chain mechanism is always less than that of the manifest mechanism. The manifest mechanism generation time ranges between 10 to 20 times that of the hash chain mechanism at 1500byte maximum Data packet size. At

9000byte maximum Data packet size, the difference is much smaller (manifest mechanism generation time ranges between 1 to 1.5 times that of the hash chain mechanism).

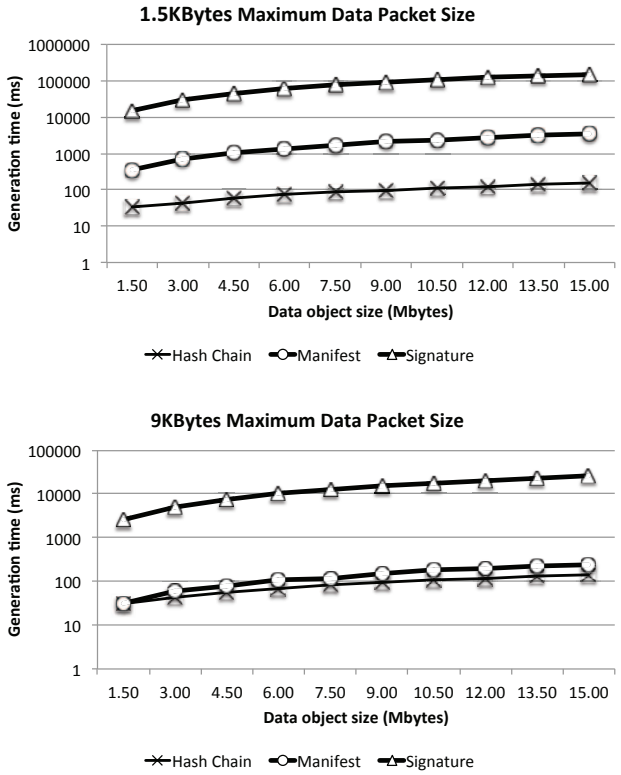


Fig. 4. Generation time of data authentication credentials

For any given data object size, both the hash chain and the manifest mechanisms produce *n* hashes, where *n* is the number of Data packets. The main difference in generation time is that the hash chain mechanism produces a single digital signature while the manifest mechanism may produce more than one. Since the manifest size is bounded by the maximum Data packet size [4], multiple manifests may be generated and signed for large data objects. For a 1500byte maximum Data packet size, a single manifest can hold 46 hashes. For a data object of size 1.5Mbytes, 22 manifests were generated. For a data object of size 15Mbytes, 218 manifests were generated. These numbers are reduced to 4 and 36 manifests respectively for 9000byte maximum Data packet size since at that size a manifest can hold 281 hashes. In comparison, the hash chain mechanism only requires one digital signature.

Verifying a digital signature is a much less expensive computational operation in comparison to generating a digital signature. Nevertheless, the standard NDN digital signature mechanism still performs poorly in comparison to the hash chain and manifest mechanisms with respect to verification time as shown in Fig. 5. This is due to the larger number of signature verifications that must be done in the standard NDN mechanism in comparison to the other two mechanisms. About 0.5ms is needed to verify a digital signature for a single Data packet in comparison to about 0.015ms to verify a hash value (verifying 33 hashes is equivalent to verifying 1 digital signature). The hash chain mechanism remains superior to the manifest mechanism. At 1500byte maximum Data packet size, the hash chain mechanism verification time is about half of that of the manifest mechanism irrespective of the data object sizes. At 9000byte maximum Data packet size, that ratio goes to almost one with a slight advantage to our mechanism.

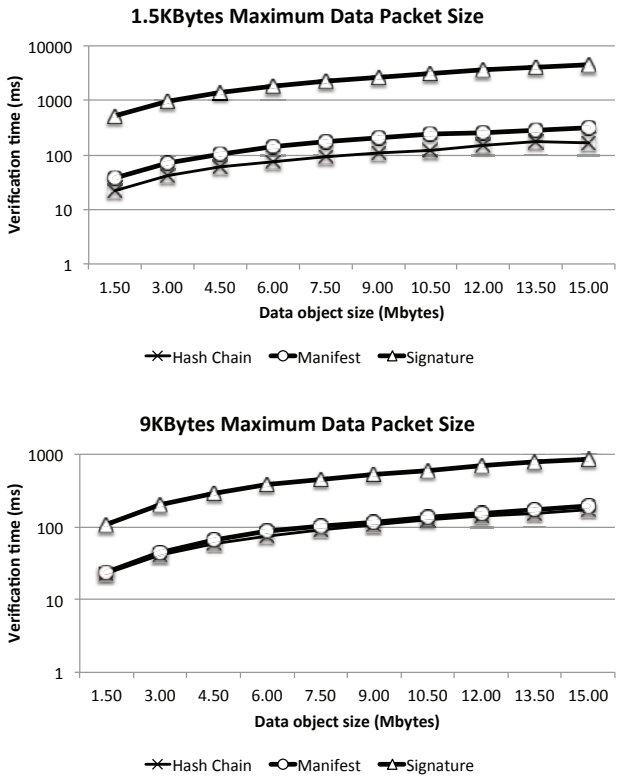


Fig. 5. Verification time of data authentication credentials

- *Communication Overhead*

Communication overhead stems from the authentication credentials that are sent along with Data packets. One component of the authentication credentials that is common across all three mechanisms is the digital signature. The size of the public/private keys we used in our experiments is 2048bits long, which generates 256byte digital signatures. As a result, every Data packet in the standard NDN authentication mechanism will have to carry along an additional 256bytes to allow for its authentication. For a 1500byte maximum Data packet size, that is a 17% overhead. For a 9000byte maximum Data packet, that value is reduced to about 3%.

The communication overhead of our mechanism is a result of the hash values included along with every Data packet as well as the seal. Each hash value is 32bytes and the seal is 256bytes. Combined, this results in an overhead of about 2% when a 1500byte maximum Data packet size is used. Note that unlike the seal value, which must be sent along with the very first Data packet, the hash values in the *Backward Chain* approach can be eliminated. This reduces the communication overhead to a constant 256bytes irrespective of the chain size or the value of the maximum Data packet. The downside to this

approach is delayed authentication. The hash chain calculation will remain the same but since a hash value will not be sent along with each message (imagine Fig. 1 without the *k* values being sent), a content consumer (or a router) will only be able to authenticate the chain as a whole. This means that the consumer has to possess all of the Data packets in the chain before proceeding to authenticate it (as discussed in section III.D).

Finally, the communication overhead of the manifest mechanism originates from the manifest files. Each manifest file is its own Data packet that has a hash list along with a digital signature calculated over the hash list. For a 1500byte maximum Data packet size, this results in a 2.5% overhead. The ratio goes down to 0.37% for a 9000byte maximum Data packet size.

- *Storage Overhead*

There is no storage overhead on the part of the content producer in the process of generating signatures or generating the hash chain. The manifest mechanism on the other hand has a slight storage overhead. The mechanism requires holding in memory the hashes of Data packets in order to generate a signed manifest. At a 1500byte maximum Data packet size, the mechanism is required to store the hashes of the last 46 Data packets (1472bytes). At a 9000byte maximum Data packet size, the number goes to 218 Data packets (6976bytes).

- *Real-time Application Support*

To support real-time applications, we consider a modification to the *Backward Chain* mechanism. Instead of computing a single seal consisting of a signature over the hashes of the last two Data packets within a data object, we instead compute a number of intermediate seals. In the context of a *Backward Chain,* an intermediate seal signs the hash value $k_i$ disclosed with a given Data packet $m_i$ in order to enable the verification of all received contiguous Data packets preceding it. To enable timely verification, a real-time application could include an intermediate seal at a periodic interval *t*. This mechanism has the advantage of enabling verification of sub-chains within the full data chain. The only downside to this mechanism is that a sub-chain will have to be buffered until an intermediate seal is disclosed to enable its verification. However, we can adapt the *Forward Chain* mechanism to operate in the same manner and eliminate the need for buffering at the receiver/router side (since the *Forward Chain* enables immediate verification of a Data Packet). However, because the *Forward Chain* looks ahead rather than backwards, will have to buffer a sub-chain at the producer side, generate an intermediate seal for it, and disclose it with the first Data packet.

## V. Conclusion and Future Work

In this paper, we introduced a hash chaining mechanism for providing data and source authentication in NDN. Our mechanism is lightweight in comparison to the standard NDN mechanism. We evaluated the performance of our mechanism and showed how it exerts low communication, computation, and storage overhead. In the future, we plan to investigate the applicability of our mechanism to DIL (Disrupted, Intermittently connected, Low-bandwidth) environments.

## Acknowledgement

This work was sponsored by the MITRE Technology Program as a MITRE Sponsored Research (MSR) Project.

## References

[1] L. Zhang, V. Jacobson, S. Diego, P. Crowley, S. Louis, and L. Wang, “Named Data Networking,” vol. 44, no. 3, pp. 66–73, 2014.

[2] M. Baugher, B. Davie, A. Narayanan, and D. Oran, “Self-verifying names for read-only named data,” *2012 Proc. IEEE INFOCOM Work.*, pp. 274–279, Mar. 2012.

[3] “http://named-data.net/doc/ndn-cxx/current/.” .

[4] I. Moiseenko, “Fetching content in Named Data Networking with embedded manifests,” pp. 0–2, 2014.

[5] C. Ghali, “Network-Layer Trust in Named-Data Networking,” vol. 44, no. 5, pp. 13–19, 2014.

[6] N. D. N. N. Components, “NDN Technical Memo : Naming Conventions Value of NDN Name Components Functional Name Component : Special Markers,” pp. 1–5, 2014.

[7] J. Burke, A. Horn, and A. Marianantoni, “Authenticated Lighting Control Using Named Data Networking,” no. October, 2012.

[8] Q. Li, X. Zhang, Q. Zheng, R. Sandhu, and X. Fu, “LIVE: Lightweight Integrity Verification and Content Access Control for Named Data Networking,” *IEEE Trans. Inf. Forensics Secur.*, vol. 6013, no. ii, pp. 1–1, 2014.

[9] I. Mironov, “Hash functions : Theory , attacks , and applications Theory of hash functions,” pp. 1–22, 2005.

[10] Y. Hu, M. Jakobsson, and A. Perrig, “Efficient Constructions for One-way Hash Chains,” pp. 1–20.

[11] I. Syamsuddin, T. Dillon, E. Chang, and S. Han, “A survey of RFID authentication protocols based on Hash-chain method,” *Proc. - 3rd Int. Conf. Converg. Hybrid Inf. Technol. ICCIT 2008*, vol. 2, pp. 559–564, Nov. 2008.

[12] “http://cs.itd.nrl.navy.mil/work/core/.” .