NDN, Technical Report NDN-0040, 2016. http://named-data.net/techreports.html
Revision 1: May 24, 2016

NDN DeLorean: An Authentication System for Data Archives in Named Data Networking

Yingdi Yu
UCLA
yingdi@cs.ucla.edu

Alexander Afanasyev
UCLA
aa@cs.ucla.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

Named Data Networking (NDN) enables data-centric security in network communication by mandating digital signatures on network-layer data packets. This change introduces a new issue with data authentication: the lifetime of data can be longer than the lifetime of the signatures which is limited by the validity periods of the corresponding certificates. In this paper, we introduce a new authentication system for archived NDN data, *NDN DeLorean*, which uses a *look back* validation model that authenticates data considering the time point when the data was produced instead of the time it is being retrieved. As long as the archived data received a valid signature at the time of its production, it can stay valid perpetually. We designed NDN DeLorean as a publicly audited timestamp service that maintains a historical evidence of the data's validity. NDN DeLorean creates permanent existence proofs of data (and certificates) upon request at a time when the original data signatures are valid. With both data and its signing key certificates being time stamped, DeLorean frees data producers from necessity to periodically re-sign archived data in order to keep it valid.

1. INTRODUCTION

Named Data Networking (NDN) changes the network communication model from "delivering packets to an end host" to "retrieving (immutable) data by name," enabling and integrating many of the long sought-after functions into a unified network delivery framework, including efficient data distribution via multicast, delay-tolerant communication, ad hoc communication, and many more. This change in communication semantics relies on a data-centric security model, which is in part realized through digital signatures on every network-level data packet. Regardless from where a data packet is retrieved, it can always be authenticated directly, i.e., without trusting either the data storage or delivery channels.

Unlike physical signatures, digital signatures may not be considered trustworthy over prolonged time periods: given enough computation power and time, it is possible to reconstruct the corresponding private key and issue impersonated signatures.[1] In addition, each created signature "weakens" the privacy of the private key [4], and there is also a chance that the keys get accidentally or maliciously leaked to adversaries. As a result, the current practices recommend the use of relatively short-lived signatures/certificates (from serval months to couple years) [2]. This limited lifetime span works well for channel-based security model since communication channels have a limited duration, but not so well for data-centric security model of NDN. The lifetime of an NDN data packet can outlive the lifetime of its signature, especially in cases of historical data archives.

In this paper, we propose an authentication system for NDN data archives, dubbed NDN DeLorean, which uses a *look back* data authentication model: data authentication is performed with the clock rolled back to the time of the data creation. In order to allow consumers to securely rollback the reference time for data authentication, we designed a publicly auditable timestamp service that issues proofs of data creation times by logging the fingerprints of archived data in the form of Merkle tree (Section 5). Given a data packet, the certificates that authenticate its signature (certification chain), and the proof of the creation time of data and certificates, one can always authenticate the data, regardless of the signature expiration and even the fact that the private key may have become known to everybody.

Our main contributions in this work include a) the look back validation model as the solution to long-lived data maintenance in NDN (Section 4), and b) the design of the first publicly auditable timestamp service over NDN (Section 5). They represent a significant step toward effective authentication of long-lived data. We also identified a number of remaining issues (Section 3) to be addressed in our future work to complete the construction of a fully functional validation system for long-lived data.

2. BACKGROUND

[1]Luckily, with the current computation technology and reasonably strong keys, it would take many years to do so [1].

Named Data Networking (NDN) is a proposed networking architecture that uses data fetching as a communication primitive. Producers secure data chunks as soon as they are created, assigning each a unique and hierarchically structured name. Consumers can then use names to fetch specific data chunks and authenticate them.

NDN defines two types of network packets to support the data-centric communication: *interest packets* as request to retrieve desired data, and *data packets* that carry the actual data. Data consumers send interests and NDN routers forward interests based on the names carried in the interests toward potential data sources, setting up a state of "pending interests" along the way. Upon receiving an interest, a data producer returns the corresponding data packet which may already exist or is created on demand. This data packet is returned back to the consumer or consumers following the breadcrumb path of pending interest states. Because each data packet is immutable and identified by a unique name, and can be authenticated on its own, NDN routers can cache data packets to satisfy subsequent interests for the same data.

For illustrative purposes, in the rest of the paper we use an electronic version of "USA Today" newspaper as an example. We assume that USA Today publishes all its articles under a namespace with the name pattern "`/UsaToday/[Date]/[Category]/[ArticleID]`". For example, a data packet with the headline story "Youth Jailed" from October 22, 2015 would be named "`/UsaToday/2015/10/22/headline/YouthJailed`".[2] All the data packets of an articles are signed by the keys of its author, who should have a unique name under the namespace "`/UsaToday/journalist`". For example, Compu Fax, the author of "Youth Jailed" article,[3] would sign the above article with the key "`/UsaToday/journalist/CompuFax/10/KEY`".

Note that NDN eliminates the requirement that data producers (USA Today) and consumers (its readers) have to be online at the same time to communnicate. A data producer can move its produced data packets to a third party storage to meet future requests for the data, or the data can be carried by some devices to meet future consumers as in a DTN scenario.

2.1 Data-Centric Authenticity

Since the data-centric communication model enables consumers' interests "to pick data from anywhere possible", traditional security solutions, e.g., TLS [8], IPSec [10], are no longer applicable as they are designed to create secure channels between two fixed end nodes. NDN embraces a *data-centric security* which ensures integrity, provenance, and secrecy of data itself, instead of relying on the delivery channel security.

[2]A large article may need to be split into multiple data packets, each getting a unique name with this prefix and a suffix that represents the segment number, e.g., "`/_s=1`", "`/_s=2`", etc.

[3]Compu Fax is indeed a program that can write automated stories for USA Today in "Back to the Future" film.

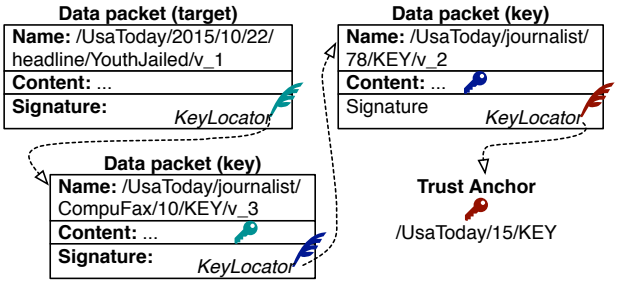

Figure 1: An example of certification chain consists of target data, intermediate keys, and trust anchor.

NDN data producers attach digital signatures to data packets and consumers who have the producer's public key can directly authenticate such data, not worrying about from where this data came. If consumers do not have the producer's public key at hand, they can apply trust schema [19] to recursively retrieve the signing keys (*certification chain*) following the information in the `KeyLocator` field of data packets, validate the chain according the trust model defined by the schema (i.e., that names of data and keys are as expected and chain terminates in a pre-trusted key—*trust anchor*), and verify validity of all signatures in the chain. Figure 1 shows an example of certification chain for our USA Today example. Note that keys are just another type of general NDN data, and given the data packet carrying a public key binds the key to its name through the packet's signature, it is effectively a public key certificate.

All digital signatures have a limited lifetime. NDN data packet signatures carry explicit validity period fields, which define periods when the signatures are considered valid. Moreover, the *effective* validity period of a signature can be shorter than its defined validity period. For a USA Today article in our example, the effective validity period will be defined as the intersection of the validity periods of all the keys along the certification chain. Therefore, to be able to verify the validity of data packets over prolonged time periods, one has to either set the validity periods of all signature to unrealistic values, or keep refreshing signatures of the data and keys along the certification chain, or use NDN DeLorian framework introduced in this paper to be able to look back in time to check the packets.

2.2 Data Immutability

All data packets in an NDN network are immutable. Any modification to the content of a data packet leads to a new version of the data packet, which must have its own unique name. A common naming convention is

to embed an additional name component that specifies the data's version number, e.g., a modified or re-signed version of the USA Today article, would be named as "`/UsaToday/2015/10/22/headline/YouthJailed/_v=2`".

3. THREAT MODEL AND ASSUMPTIONS

In this paper we focus on *long-lived* data, i.e., the data packets or data collections that need to be preserved for a long period of time. Typical examples of such data include newspaper articles, library archives, historical records, experimental results, etc. Although DeLorean could be used in all scenarios, it may be considered prohibitive expensive in terms of processing and storage overheads if one were to use DeLorean for unbounded volumes of data. The key security issue we address in this paper is to ensure that the long-lived data can stay authenticatable, potentially many years after the data producer ceased to exist. Note our focus is on the authentication aspect of the data; ensuring long-term secrecy of confidential data is outside the scope of the current work.

We assume that the cryptographic keys in the the corresponding trust chains of long-lived data have limited validity periods in order to restrict key exposure and potential harm of the key compromise. We also assume that the consumers know which trust schema should be used to authenticate data and that the trust anchors do not change over time. We plan to address these assumptions in the next milestone of our research.

In order to authenticate the data with the above assumptions, NDN DeLorean implements a notary service that "certifies" the existence of data at particular points of time. To ensure that this third party service behaves correctly, there must be continuous audit of its consistency by either or both dedicated parties and volunteers (*auditors*). Potential misbehaviors of the notary service include timestamp *denial*, *repudiation*, *reordering*, and *injection*: the timestamp notary should not be able to deny access to the previously issued proofs, pretend that the previously issued proof is invalid, alter timestamp of the existing proof, or inject a new proof for a past timestamp. Any such misbehavior can be noticed by the public auditors, who can then take actions to remediate the issue: request immediate correction of the timestamp service behavior or switch to alternative timestamp service provider. The design described in this paper assumes the existence of a single timestamp service; we briefly discuss how multiple alternative timestamp services can co-exist in Section 7.

In this paper we assume that the key used to sign data is valid during timestamp certification and is not leaked during its validity period, i.e., there is no *producer impersonation* while the trust chains are within their validity. For example, if a USA Today article is timestamped at time t, we assume this article has a valid signature at the time, and only USA Today can create the signature during the validity period of the corresponding trust chain. As part of our future work we plan to extend the design to incorporate revocation of the archived data to address potential producer impersonation problem during validity periods of the keys.

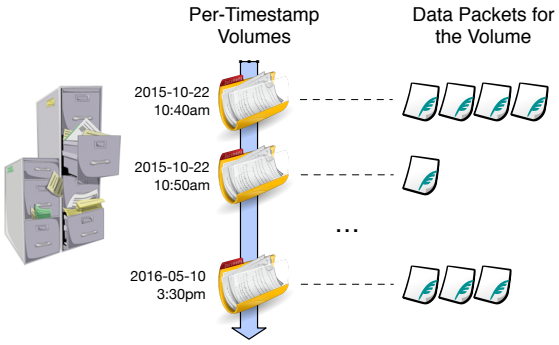

Figure 2: DeLorean's data chronicle

However, an attacker may launch impersonation attack after the timestamp creation. In this case, an attacker may pre-produce data signed by an uncertified key and record the data in DeLorean chronicle (described below). The attacker may recover the key of the victim's certificate issuer after certain amount of time (through key leaking or brute force computation). At this point, the attacker can create a certificate for its previously uncertified key and claim that the data was valid when it was produced. Our counter measure to this is to timestamp both data and their signing keys, and ask consumers to verify the existence of both data and keys. In this way, consumers can reject the falsified certificate because it cannot proof its existence before the data production.

4. OVERVIEW

In this section, we present a high-level overview of DeLorean, a verifiable and publicly audited timestamp service, as the solution to the threats described above.

DeLorean is an always-on service that publishes a data "chronicle" (Figure 2). The chronicle consists of a sequence of volumes, each containing fingerprints of the witnessed data packets, such as specific USA Today articles, within a fixed timeslot, e.g., 10 minutes. The existence of a data packet (its fingerprint) in a particular volume is a *timestamp proof* that the data packet has existed before the end of the corresponding time slot. Each volume is finalized at the end of each time slot and published as a set of data packets, given the volume's information may not fit into a single data packet. After the volume is finalized, it cannot be changed without invalidating consistency with any future volumes.

At any time, a data producer (article's author) or an archive service on the producer's behalf (USA Today

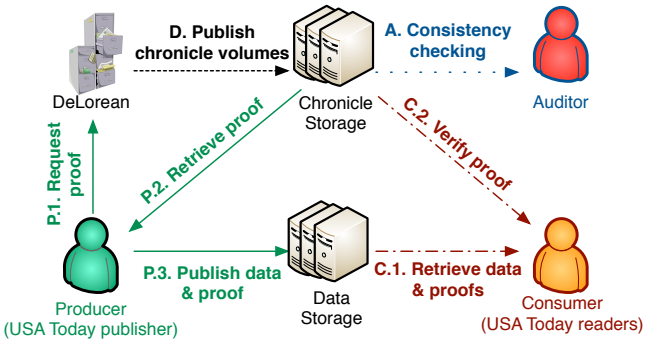

Figure 3: DeLorean workflow

publisher) can request a timestamp proof for data (articles) from DeLorean (Flow P.1 in Figure 3), supplying a fingerpint of the archived data in form of a hash digest of an individual data packet or a digest of the manifest that represents a data collection. The response to this request is a name of the chronicle volume that will be published by DeLorean at the end of the current cycle (Flow D) and the index of the fingerprint in the volume. After waiting until the volume is ready (on average a 5 minute wait in our example), the producer can retrieve the volume to verify whether DeLorean has included the data fingerprint in the volume (Flow P.2). In the end, the producer can publish the timestamp proof, which includes the full name of the volume and the index of data fingerprint, alongside the data (Flows P.3).

To verify data independently of its signature validity, consumers need to "look back" to the timepoint when data was produced (or time stamped). A consumer first obtains the corresponding timestamp proof, which can be stored alongside the data (Flow C.1), and verifies the data existence by retrieving several additional DeLorean volumes (see Section 5). Similarly, the consumer verifies the existence of the data's signing key certificates.[4] With all certificates proving their existences, the consumer can verify the data signature as if it was at the time of production or time stampping.

In order to ensure the correct and truthful operations of DeLorean, a set of third-party auditors continuously check the consistency of the chronicle (Flow A), i.e., checking that DeLorean has not modified the previously published volumes. If auditors detect that DeLorean has modified the chronicle, the users of the service (auditors, data producers, and consumers) will take immediate actions to either fix the issue or abandon the specific instance of DeLorean service. In order to guarantee consistency, each DeLorean volume has to be retrieved at least by one auditor around the time it is published. The more auditors are involved in the process, the less frequently each individual auditor needs to perform consistency checking. Note that although consumer and producer roles are separated from the auditor role in Figure 3, they can be (and, from security perspective, should be) combined.

[4]Certificate issuers request timestamp proofs for the issued certificates. Alternatively, a data producer can request and publish the timestamp proofs of the data and the corresponding certificates as a bundle, similar to our previous certificate bundle proposal [18].

4.1 Design Objectives

The first DeLorean's design objective is to minimize the storage and verification overheads, as overwhelming overheads would prohibit any use of system. For example, a naïve solution to consistence verification is to ask each auditor to maintain a local copy of the whole chronicle by retrieving volumes at the end of each time slot. Such solution not only requires impractical storage at the auditor side, but also requires each auditor to timely retrieve each volume. On the other hand, if consistency verification cost is trivial, it can encourage more users to audit DeLorean, improving fidelity and overall usefulness of the system.

The second design objective is to prevent DeLorean from knowing identities of auditors and knowing if audit requests are coming from the same auditor(s) or not. If DeLorean could do that, it would be able to present one consistent chronicle to a group of auditors while presenting a completely different consistent chronicle to another. In this case, none of auditors in the two groups can detect any modification, while the timestamp service would be obviously inconsistent.

The third design objective is to minimize the maintenance overhead. Given the number of volumes increases over the time, the amount of data stored in the volume should be sufficiently concise yet faithfully record all evidence in the corresponding time periods. Moreover, consistence and existence verifications involve continuous retrieval of the published volumes, desiring as low overhead as possible.

For the these design objectives, we will present the solutions in detail in Section 5. However, there are several design objectives that we have not addressed yet. For example, how to prevent a single party from monopolizing the recording of chronicle; how to increase the robustness of DeLorean; and how to reboot DeLorean in case of inevitable failures. We will discuss the potential solutions to these objectives in Section 7.

5. DELOREAN

In this section, we present the design details of DeLorean. At the end of every time slot, DeLorean publishes a volume in the chronicle in a form of one or multiple data packets, to archive data packets recorded during the time slot. For the sake of simplicity, we first assume that DeLorean publishes a volume as a single data packet and explain how to expand the capacity of

a single volume with multiple data packets in In Section 5.3.

The volumes per se however are also archive data. A simple solution to ensure authenticity of old volumes would be inclusion of crypto hash digest of the previous volume in a new volume, effectively constructing a hash chain of chronicle volumes. In this case, to authenticate any historical volume, one needs to authenticate the latest volume which should have a valid signature, and then verify the authenticity of previous volumes by checking their hashes one by one.

This hash chain based chronicle however results in large overhead, as authentication time would require $O(n)$ volume retrievals, where n is the number of time slots between the current time slot and the time slot of the volume under verification. For example, with 10-minute time slot, one has to retrieve a prohibitive amount of records (about a million) to authenticate a volume 20 years ago.

Inspired by Certificate Transparency [11], we design DeLorean chronicle as a Merkle tree [13] to minimize the authentication overhead. As we explain in detail in Section 5.1, the state of the chronicle is represented as a Merkle tree with volumes represented as leaves of the tree, where the root node effectively stores a hash of all volumes published so far. With this structure, the verification overhead of an old volume can be reduced to a much smaller number of operations $O(\log n)$. For example, the same 20 year old record can be authenticated by DeLorean with the binary Merkle tree using just several dozens of retrievals, which can be even further reduced by selecting a more optimal tree structure.

Note that the chronicle volume authentication by itself does not provide guarantee of the DeLorean chronicle consistency, i.e., DeLorean can still arbitrarily inject data into past volumes and then re-create subsequent volumes. To provide the guarantee, a set of auditors (dedicated entities, consumers, and producers) is required to periodically retrieve and authenticate the current volume (current state) and check consistency with previously fetched volumes (past state). With the trivial verification overhead, made possible by the use of Merkle trees, the consistency check between the current and any previously verified state is a trivial task (Section 5.2), which can be performed by a large number of auditors. With chronicle under public audit, consumers can simply assume all the historical volumes are consistently covered by the latest chronile state.

5.1 Chronicle Construction

Next, we describe how to construct a chronicle using Merkle tree, and how to efficiently verify the existence of a volume.

Merkle tree is a k-ary tree, where the value of each node is the hash of the concatenation of the value of its children. Similar to hash chains in which the last node fixes all the previous nodes, root node of the Merkle tree fixes all the leaves in the tree. Any change of any leaf leads to the change of the root hash. Figure 4a shows a binary Merkle tree with three leaves.

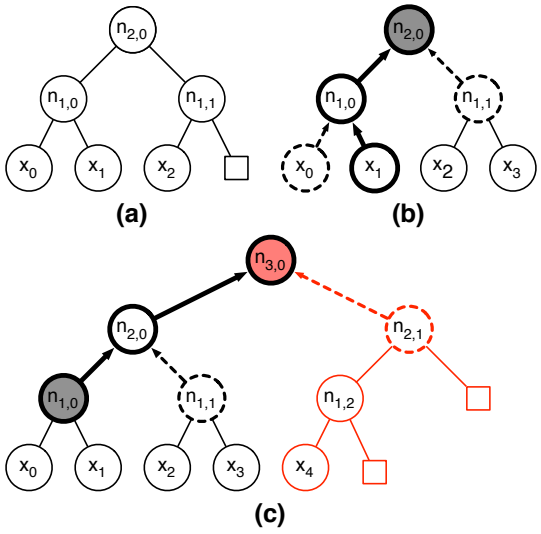

Figure 4: Merkle tree examples: (a) a Merkle tree with three leaves; (b) the evidence proof for leaf x_1 in a four-leaf Merkle tree; (c) the consistence proof between a tree with two leaves (x_0 and x_1) and a tree with five leaves (x_1 to x_4).

To construct a chronicle using a Merkle tree, we align volumes as leaf nodes in a Merkle tree, adding a new leaf to the tree whenever a new volume is published. This addition leads to change of hash values in all of the ancestors up to the root of the tree. If the tree is full, it can grow one level up to accomodate more leaves (or volumes). For example, the three-level tree in Figure 4c grew from the two-level tree in Figure 4a and can cover at most 8 leaves.

Whenever a new volume is added, in addition to updating the Merkle tree, DeLorean also signs the root hash and publishes it as a separate data packet, described in the next section. The signature of this data packet can be used to transitively authenticate all previously published volumes.

In order to verify existence of a particular volume in the chronicle (e.g., that the volume x'_1 exists), one needs to reconstruct a part of the tree along the path from the corresponding leaf node to the current root node of the tree ($x'_1 \rightarrow n'_{1,0} = \text{hash}(x_0, x'_1) \rightarrow n'_{2,0} = \text{hash}(n'_{1,0}, n_{1,1})$ in Figure 4b). The volume x_1 can be proved to exist in the chronicle if the reconstructed value of root node ($n'_{2,0}$) matches ($n_{2,0}$), the one recorded in the tree. Therefore, if we use a k-ary Merkle tree that has n leaves, a single verification only requires $O(\log_k n)$ hash computations in total.

To verify consistency of the Merkle tree evolution, one needs to know the root digest represented some old state and the most recent root digest. For example, to verify consistency between states x_1 and x_4 in Figure 4c, one needs to know past root digest $n_{1,0}$ and the current digest $n_{3,0}$. Similar to the existence verification, one can reconstruct nodes along the path from old to new root: $n'_{2,0} = \text{hash}(n_{1,0}, n_{1,1}) \rightarrow n'_{3,0} = \text{hash}(n'_{2,0}, n_{2,1})$. The tree evolution can be declared consistent if all reconstructed values match the one stored in the tree.

5.1.1 *Proof Publishing*

To verify existence of the volume in DeLorean requires knowledge of all sibling nodes along the path to the root (nodes circled by dashed line in Figure 4b). In order to allow it, DeLorean publishes each node of the Merkle tree as an individual data packet, including the hash values of all its children (Figure 6). Note that with a 1500-byte MTU and SHA-256 hash algorithm being, a single data packet can safely carry 32 SHA-256 hashes (1024 bytes in total), leaving enough space for other fields in the data packet. For that reason, we chose

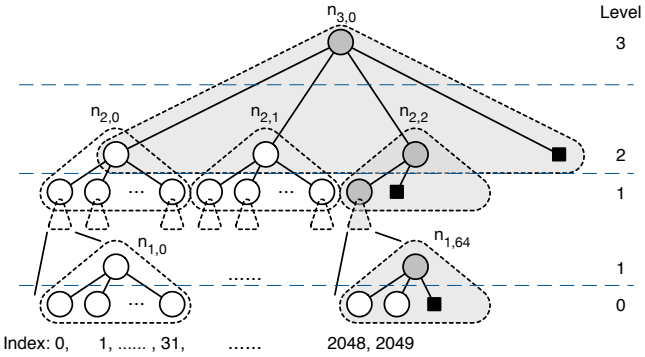

Figure 5: 32-ary Merkle tree example

32-ary Merkle tree to construct DeLorean's chronicle, exemplified in Figure 5. In this case, a chronicle with volumes for each 10 minute time interval will require only four levels of the Merkle tree to record 20 years worth of state.

Note that retrieving nodes individually leverages efficient data distribution of NDN: requests from multiple auditors can be efficiently joined or served from in-network caches. Because nodes at higher layers are involved in more verifications, they are more frequently requested and have a high chance of being universally cached in the network.

5.1.2 *Node Naming Convention*

For the Merkle tree nodes we defined the naming convention as shown in Figure 6a, which consist of five parts. The first two parts specifies the tree prefix and the hash algorithm used to construct the tree.

The third part of the name is a component indicating a state of the node: "complete" when a node has the full set of descendents (e.g., white nodes $n_{1,0}$, ..., $n_{1,63}$, $n_{2,0}$, and $n_{2,1}$ in Figure 5), or "incomplete" when one more descendents do not yet exist (gray nodes in Figure 5). The hash values of the incomplete nodes are keep changing until all leaves added to the corresponding subtree, after which the node becomes complete with the perpetually fixed hash value. Given a node can have as many incomplete states as the half of the leaf nodes it covers, to disambiguate the name for different states, we included the sequence number of the next leaf node that can be added to subtree. In Figure 5 example, all incomplete nodes would have "`/incomplete-2050`" as a node state component (e.g., the first name in Figure 6a).

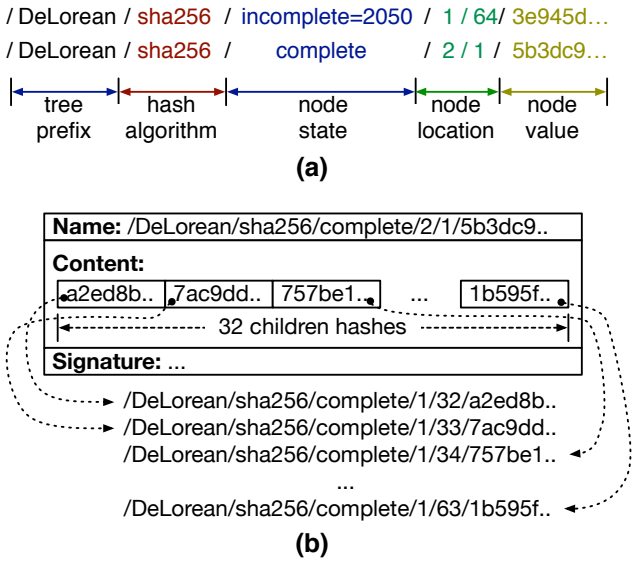

Figure 6: (a) Naming convention of 32-ary chronicle tree node; (b) An example of 32-ary chronicle tree node data.

Note that all nodes in all states are represented as immutable data packets, and can be easily replicated in the network. At the same time, only the latest state of the node is needed to perform the volume existence or chronicle consistency verifications: the content of the data packet that represents a new state includes all information that existed in previous data packets. Therefore, as soon as the new state of the node is created, the old state data can be safely removed from the system.

The fourth part of a node name includes two parts: the level of the node and the index of the node at the specified level. Given the total number of nodes in the Merkle tree n, the sequence number s of the leaf node, and the level l ($0 \leq l \leq \lceil \log_{32} n \rceil$) of the intermediate node, its index in the level can be calculated as $i_l = \lfloor s \times 32^{-l} \rfloor$. Using these simple conversions, consumers and auditors can request intermediate nodes for any desired level, e.g., requesting them simultaneously.

The last part is the hash value of the node, which is also the digest of data content. During existence and consistency verifications, consumers and auditors can explicitly request a node using the expected hash digest value, calculated from the pre-verified parts of the tree.

5.2 Public Audit

A chronicle, once being detected as inconsistent (i.e., a previous volume being modified), immediately loses its trustworthiness. As we explained before, by building DeLorean chronicle over Merkle tree, the consistence verification overhead is on the order of $O(\log n)$. With this trivial overhead, a large number of auditors can occasionally and effortlessly check the consistence of the DeLorean chronicle. The collective behavior of the auditors can ensure that at each time slot the consistence of DeLorean chronicle is checked by at least one auditor. Therefore, it is difficult for the chronicle publisher to modify previous volumes without being caught, thus effectively deterring the chronicle publisher from modifying the history. For example, it would be impossible for DeLorean to modify the record for October 22, 2015 issue of USA Today or deny its existence without actually using the time machine and altering the reality. Moreover, since the producers and consumers of data recorded by chronicle rely on it to provide the existence proofs, they have strong motivation to audit the consistence of the chronicle.

5.2.1 Consistence Verification

To audit the consistence of the Merkle tree based chronicle, auditors occasionally retrieve the root hash of the tree and check whether it "covers" a root hash that the auditor has retrieved before. Once an auditor verifies the consistence between the two hashes, the auditor can keep the new root hash and discard the old one. Therefore, the auditor's storage overhead is constant.

Similar to existence verification, the consistence verification is to re-compute the new root hash from the old one, other nodes retrieving nodes along the way. With all the nodes of the chronicle tree being published, an auditor can retrieve the nodes that are necessary for consistence verification in at most $O(\log n)$ number of iterations.

Incomplete Node Issue.

Note that the previously recorded root hash is most cases would be represented as an incomplete node, whose status will change to complete or a different incomplete state. For example, an incomplete node $n_{2,0}$ in Figure 4a captures state of volumes x_0, x_1, x_2, while the same node becomes complete when capturing state for volumes $x_0, \ldots, x_3$ in Figure 4b. This fact, while complicating the process, does not impact the ability to perform the tree reconstruction. The auditor will need to retrieve the latest state of the node and check that it is a superset of the old state. The exact current state of any node in the Merkle tree is determined by the number of leaves. For a 32-ary tree with the largest volume sequence number s, for a node at the level l,

$$\text{node state is } \begin{cases} \text{“}\texttt{complete}\text{”}, & \text{if } s \geq 32^l \\ \text{“}\texttt{incomplete-(s+1)}\text{”}, & \text{otherwise} \end{cases}$$

Multiple History Issue.

The only way that the chronicle publisher can modify the history without being detected is to present a different chronicle consistently to the same group of auditors, which is usually called a multiple history issue. The stateful data retrieval and in-network caching of NDN architecture, however, intrinsically eliminate the possibility of creating multiple histories that target different auditors.

Interests that request node data do not reveal any information about the requesters, or in this case auditors. Therefore, it is impossible for DeLorean to craft the auditor-specific responses.

In addition to that, DeLorean will not receive all interests for node data, as they can be aggregated (when multiple auditors request state at the same time) or served from in-network caches. The higher-level nodes of the DeLorean chronicle can be used to verify many different individual states, Therefore, we expect that the data packets that correspond to these states will be universally cached throughout the NDN network, further reducing any possibility of DeLorean to crafting individual responses.

5.3 Volume Construction

In the previous description we focused on verification of the state of the chronicle volumes. Consumers, however, would want also to verify the existence of a data packet in the volume, which represent a collection of data packet fingerpints submitted within the corresponding time interval. In the simplest case, the volume is represented as a single data packet which records all submitted fingerprints for the corresponding time period. However, a volume may need to record a large number of data packet fingerprints, exceeding capacity of a single data packet. Therefore, a volume needs to be constructed as a set of data packets, but in a way to minimize overhead for data existence verification.

To accommodate a large number of data fingerprints and yet provide efficient verification, we construct the volumes with the help of Merkle trees. Leaves of the volume-specific Merkle tree represet hashes of data (the recorded fingerprints), and root hash of the tree "fixes" all fingerprints in the volume (Figure 7). Recoding the volume tree's root hash as a leaf node of the chronicle's

Merkle tree, effectively "fixes" this volume in time.

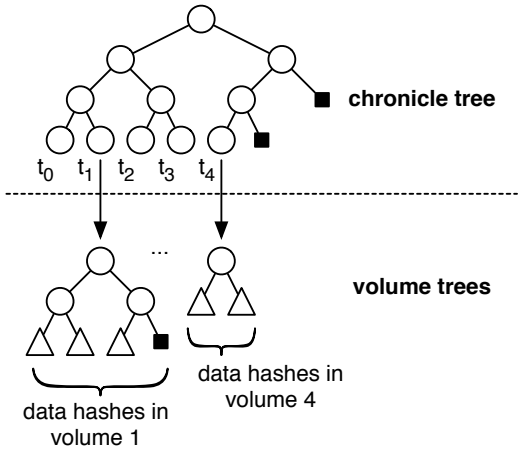

Figure 7: Two-level Merkle tree hierarchy of the timestamp service.

Given a volume tree, a consumer can quickly locate a record according to the local record index. Based on the local index, a consumer can compute a verification path from the record back to the volume tree root and verify the existence of the record in a volume, same way volume existence verification described in Section 5.1.

In summary, to assure that a specific data packet existed at a specific time point, the consumer needs to have: (1) volume hash, (2) volume index, and (3) local record index within a data volume. The first two are used to reconstruct the relevant portions of the chronicle tree; and the last one along the fingerprint of the data (obtained from data directly) can reconstruct and verify consistency of the volume tree.

5.4 Hash Rollover

For the sake of simplicity, we used only one hash algorithm (SHA-256) in the description above. However, no hash algorithm can be secure forever. Once a hash algorithm is broken (though not very often), all the records in the chronicle are no longer secure.

A proper hash algorithm rollover is the key for DeLorean to prevent hash breaking. More specifically, before the hash algorithm in use is broken,[5] DeLorean can publish another chronicle tree with the same volume sequence but using a stronger hash algorithm. Auditors can verify the new chronicle tree against the existing one to ensure the correctness. Note that hash algorithm breaking happens rarely, the overhead of verifying a new tree though expensive is still affordable.

In some rare case, a hash algorithm may break unexpectedly. In order to survive from the "hash crisis", DeLorean can always publish two sets of chronicle tree, of which each is constructed using a hash algorithms with different crypto strength. Since it is really rare (if not impossible) that the two hash algorithms will be broken at the same time, the stronger hash algorithm offers the publisher to find another hash algorithm with more crypto strength and rebuild the chronicle trees.

Note that hash rollover cannot address the broken hash issue completely. Although it can secure the chronicle tree and volume tree, it cannot prevent an attacker to utilize hash collision to modify data content if the hash algorithm of the original data signature is broken. However, given the chance of falsifying a meaning content with the same digest is really rare, we will address this issue in our future work.

[5]In most cases, a hash algorithm does not completely break immediately.

6. STORAGE REQUIREMENTS

Since DeLorean chronicle is a permanent record of archived data and is growing over time, it is important to evaluate the storage requirements. We consider a 20-year chronicle with 10-minute time slots, with both chronicle and volumes represented as 32-ary Merkle trees.

The storage overhead of DeLorean consists of two parts: the chronicle tree and volume trees. A 20-year chronicle involves about 1 million (i.e., $\simeq 32^4$) volumes. Therefore, the chronicle tree has four levels and 32259 intermediate nodes.[6] Assume the size of each node packet is 1500 bytes, the total storage capacity required to save 20 years of chronicle tree is about 48 MB.

If chronicle volumes contain on average 1024 fingerpints of data packets, the corresponding volume trees would have two levels, with the leaf nodes as 32-byte hashes. Therefore, a volume tree involves 33 intermediate nodes (32 nodes at level 2 and one root node). A single volume tree would take about 50 KB, and the total storage requirement for 1 million volumes would be about 50 GB. If we increase the average capacity of a volume to 32768 (i.e., 32^3) data packets, the storage overhead of a single volume would increase to 1.6 MB, renderring the total storage overhead for all volumes to 1.6 TB. And for volumes with 1 million data packet capacity (i.e., 32^4), the total storage would be only 50 TB, which is still modest compared to the current commercial servers.

7. DISCUSSION

7.1 Scaling DeLorean Storage

Although our analysis above suggested that DeLorean may potentially record a large amount of data over the time, the storage however will still be overwhelmed if all the produced packets in the world would require timestamp certification. Therefore, there has to be a limit on

[6]The leaf nodes of chronicle tree is also the root node of volume tree, so they are included in the volume tree storage calculation.

the number or frequency of the certifications. This limit can be enforced, for example, by business relationships between data producers and DeLorean provider where producers pay for each certification using real money or time working as system auditor (e.g., auditing DeLorean for an hour gives a credit for one timestamp certification).

The limit on number of certifications does not mean that only a limited number of data packets can be timestampped. Using manifest-based aggregation techniques [3, 16, 14] producers can request a single certification for a large collection of data packets. For example, USA Today publisher does not need to request timestamp proofs for every single article published on October 22, 2015, instead it can create a manifest linking all October 22 articles (in a simple list or a tree of manifests) and request proof just for that manifest.

As part of our future work we will investigate how to support safe deletion or compression volume records that are no longer needed [7]. This will allow further reduction of the storage overhead and relaxing of the enforced certification limit.

7.2 Recovery from Audit Failures

Whenever the auditors detect that the DeLorean provider is not consistent with the previously recorded states, they need to take actions to remediate the issue. The first course of actions would be to publicly contact the provider and attempt to correct the problem, which highlights necessity of an open channel to the provider through which problems can be reported. This way, the issue can be confirmed by multiple auditors, forcing the provider to immediately address the problem.

In an unlikely case when the DeLorean provider does not respond to the reported issues or no longer wishes to provide the service, it is possible to transition to a new provider. Recall that the authenticity of previous volumes can be implicitly verified through Merkle tree state. The new provider can pick up the service from a state under the consensus of the auditors and obtain copies of the existing volumes that represent the last consistent state. After updating the publisher public key, users of the timestamp service can keep using the same historical volumes.

7.3 Resiliency & Multiple DeLorean Providers

The example presented in this paper only involves a single instance of DeLorean. However, it is important to avoid a single point of failure, which can be implemented in part using a set of hot backup instances. As soon as there is an issue with a primary DeLorean instance, another timestamp service instance can immediately resume from any mirror. Note that all the data structures (e.g., Merkle trees and hash chains) of DeLorean are publicly audited, thus it is trivial to keep them in-sync among backup instances.

It is also possible to run multiple independent DeLorean instances: it would be producer's decision on which instance to use. The only change to the described protocol would be in the naming: instead of a single "`/DeLorean`" prefix, the chronicle volumes will be published under "`/google/DeLorean`", "`/apple/DeLorean`", and similar prefixes. However note that consistency guarantee of a single DeLorean instance depends on the quality of the public audit. In case a consumer does not have much confidence about public audit, it can still audit a particular instance by itself at a frequency that satisfies the consumer's own need.

7.4 Impact Timestamping on Data Production

Data aggregation and timestampping does not block data production and consumption. It is an additional procedure to ensure data can still be authenticated after the signature expires. In other words, before the original signature on data packets expires, consumers can directly verify the data without needing the timestamp. For example, "Youth Jailed" article of USA Today can be directly authenticated on October 22, 2015 or several days after, until the original signature is still valid. Only when readers access that article year or so later, they may need to use the timestamp proof in order to ensure authenticity of the article at the time it was originally published.

8. RELATED WORK

To the best of our knowledge, Haber and Stornetta [9] were the first to propose the use of the timestamp service to secure digital documents. They built the service by linking documents in a time order using a crypto hash function, allowing users to check the existence of a document by checking against a set of documents along the timeline. Buldasi et al. [5] later proposed a binary linking timestamp that simplified implementation of the timestamp service. Additional information and history of the timestamp service designs is available in the survey by Vigil et al. [17].

The timestamp service work that is most related to DeLorean design is KASTS [12]. KASTS not only timestamps signed documents, but also keeps a secure storage of verification keys. Compared to KASTS that builds the timestamp service over hash chains, DeLorean uses Merkle tree hierarchy to allow efficient public auditing. Moreover, KASTS is focused on a single trust model, i.e., the PKI model, while DeLorean supports data signing under arbitrary trust models, as long as consumers know the corresponding trust schema.

The foundation of DeLorean design is a work of Crosby and Wallach [7] that proposes the use of Merkle tree to implement a tamper-evident logging system. They conducted the detailed performance analysis to prove effi-

ciency of the Merkle tree based logging system. They also proposed a scheme to safely delete log entries that are no longer needed, which we plan to investigate in the next revisions of the DeLorean design.

Certificate Transparency (CT) [11, 6] offers one of the most important use cases of Merkle tree based logging system and inspired the design of DeLorean. CT is designed to mitigate the certificate mis-issuance problem through a "security through publicity" approach. CT uses Merkle tree to build a public board, on which certificate authorities are required to post all the issued certificates. Using this board, the legitimate owners of the domain names can easily detect the mis-issued certificates. DeLorean borrows the same "security through publicity" concept, but applies it to verification of absolute time of the chronicle volumes: any attempt to "back-publish" or modify volume will be detected by a set of public auditors. Because of the nature of IP protocol, CT instance will always know the source address of the requester. To avoid problem of multiple consistent views to different users, CT design includes an additional gossip protocol [6]. DeLorean intrinsically avoids this problem by being an NDN-based system: data retrieval in NDN does rely on source addresses, but uses states set up by the incoming requests.

BitCoin [15] represents another example of "security through publicity" approach to support a consistent append-only log based on hash chain. However, BitCoin requires an efficient peer-to-peer overlay multicast network and also requires each peer in the system to keep a copy of the history, thus making it unsuitable for maintenance of a large amount of long-lived data.

9. CONCLUSION

In this paper we presented the design of DeLorean. After the data is archived using DeLorean, it can be validated through "look back" authentication by using the issued time proofs to safely roll back the clock to check the validity of signatures. The heart of DeLorean is the timestamp service that publishes the data chronicle which consisting of volumes of data fingerpints, each volume corresponding to a specific time interval. We structure the chronicle as a multi-ary Merkle tree which enables quick and efficient public audit of the record consistency, enforcing the operation correctness through publicity. To our best knowledge, it is the first ICN system whose deisgn applies the concept of security-through-publicity.

DeLorean is our first step toward securing long-lived data in NDN and lays the foundation for our future work in certificate revocation and trust bootstrapping. DeLorean effectively decouples lifetimes of long-lived data and their signatures. The decoupling of signature lifetime from the data lifetime encourages the use of short-lived signatures for important data (as long as their lifetime is long enough to obtain proofs), reducing dependency on key revocation mechanisms.

Although DeLorean is designed for signature validation, we believe the design is general enough to accommodate other append-only logging systems over NDN. We expect to see more uses of the same idea in other applications, such as transaction systems, databases, etc., in the future.

10. REFERENCES

[1] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management. NIST Special Publication 800-57, July 2015.

[2] R. Barnes, J. Hoffman-Andrews, and J. Kasten. Automatic certificate management environment (ACME). https://tools.ietf.org/html/draft-ietf-acme-acme-02, October 2015.

[3] M. Baugher, B. Davie, A. Narayanan, and D. Oran. Self-verifying names for read-only named data. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*. IEEE, 2012.

[4] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In *Advances in Cryptology—ASIACRYPT 2001*, 2001.

[5] A. Buldasi, P. Laud, H. Lipmaai, and J. Villemson. Timestamping with binary linking schemes. In *CRYPTO'98*, 1998.

[6] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie, and E. Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 415–423. IEEE, 2015.

[7] S. A. Crosby and D. S. Wallach. Efficient data structures for tamper-evident logging. In *Security Symposium*, pages 317–334. Usenix, 2009.

[8] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5245, 2008 August.

[9] S. Haber and W. Stornetta. How to time-stamp a digital document. In *CRYPTO'90*, 1990.

[10] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301, December 2005.

[11] B. Laurie. Certificate transparency. *Queue*, 2014.

[12] P. Maniatis and M. Baker. Enabling the archival storage of signed documents. In *the USENIX Conference on File and Storage Technologies (FAST) 2002*. Usenix, 2002.

[13] R. C. Merkle. Protocols for public key cryptosystems. In *Security and Privacy, 1980 IEEE Symposium on*, pages 122–122. IEEE, 1980.

[14] I. Moiseenko. Fetching content in Named Data Networking with embedded manifests. Technical Report NDN-0025, NDN, 2014.

[15] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[16] C. Tschudin and C. Wood. File-like ICN collection (FLIC). Internet-Draft, draft-tschudin-icnrg-flic-00, 2016.

[17] M. Vigil, J. Buchmann, D. Cabarcas, C. Weinert, and A. Wiesmaier. Integrity, authenticity, non-repudiation, and proof of existence for long-term archiving: a survey. *Elsevier Computers & Security*, 2015.

[18] Y. Yu. Public key management in Named Data Networking. Technical Report NDN-0029, NDN, 2015.

[19] Y. Yu, A. Afanasyev, D. Clark, k. claffy, V. Jacobson, and L. Zhang. Schematizing trust in named data networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015.