# LIVE: Lightweight Integrity Verification and Content Access Control for Named Data Networking

Qi Li, *Member, IEEE*, Xinwen Zhang, *Member, IEEE*, Qingji Zheng, Ravi Sandhu, *Fellow, IEEE*, and Xiaoming Fu, *Senior Member, IEEE*

***Abstract*— Named data networking (NDN) is a new paradigm for the future Internet wherein interest and data packets carry content names rather than the current IP paradigm of source and destination addresses. Security is built into NDN by embedding a public key signature in each data packet to enable verification of authenticity and integrity of the content. However, existing heavy-weight signature generation and verification algorithms prevent universal integrity verification among NDN nodes, which may result in content pollution and denial of service attacks. Furthermore, caching and location-independent content access disables the capability of a content provider to control content access, e.g., who can cache a content and which end user or device can access it. We propose a lightweight integrity verification (LIVE) architecture, an extension to the NDN protocol, to address these two issues seamlessly. LIVE enables universal content signature verification in NDN with lightweight signature generation and verification algorithms. Furthermore, it allows a content provider to control content access in NDN nodes by selectively distributing integrity verification tokens to authorized nodes. We evaluate the effectiveness of LIVE with open source CCNx project. Our paper shows that LIVE only incurs average 10% delay in accessing contents. Compared with traditional public key signature schemes, the verification delay is reduced by over 20 times in LIVE.**

***Index Terms*— Next generation networking, access control, data security.**

## I. Introduction

NAMED data networking (NDN) is recently proposed to solve several fundamental problems of the existing IP networks, e.g., using in-network caching to optimize bandwidth use, and location-independent content access for multi-path forwarding and mobility management [1]. The NDN design has many security advantages. For example, each data packet in NDN is digitally signed by an entity (e.g., its publisher), such that its integrity and authenticity can be verified by network nodes and end users, no matter where they retrieve the data packet. However, the NDN design also faces several important security challenges. First, existing signature generation and verification algorithms are heavy-weight such that universal content integrity verification is hard to achieve for network nodes, especially for Internet-scale content routers. Secondly, the current NDN design allows arbitrary content caching and accessing such that any network node of a domain (e.g., an Internet Service Provider) that enables NDN can arbitrarily cache contents when the contents are delivered by them, without any approval from Content Providers (CP). Similarly, users can arbitrarily request and access any content that they want from network caches, which is also out of CP's control.

Traditionally, content caching and access control are performed in application-level services, such as encryption-based access control or delegation-based services [2]. We seek to address efficient integrity verification and content access (including caching) control with a single solution in the network layer, by leveraging the existing security mechanism in NDN with a minimal extension. Particularly, our design is based on the fact that the current NDN design requires a signature field in each content packet for content integrity verification [1]. Intuitively, NDN nodes, i.e., content routers and end users (devices), are willing to cache or consume a data packet only after its integrity is successfully verified, which means that the packet is not tampered or faked. By controlling the capability of verifying the integrity of a content object in network nodes and end devices, our solution achieves lightweight content access control with efficient integrity verification.

Towards these, we propose LIVE, a lightweight integrity verification architecture for NDN. It controls the verification capability of content integrity and authenticity for NDN nodes (content routers and end users) with an efficient key update mechanism, such that unauthorized nodes cannot successfully verify and thus drop content packets. With such a selective integrity verification mechanism, to prevent unauthorized content access, a CP can generate integrity status for each content packet with respect to the content name and the NDN nodes

Manuscript received March 9, 2014; revised June 22, 2014 and September 16, 2014; accepted September 20, 2014. Date of publication October 31, 2014; date of current version January 13, 2015. This work was supported in part by a gift from Cisco, in part by the Air Force Office of Scientific Research, Arlington, VA, USA, through the Multidisciplinary University Research Initiative Programme under Grant FA9550-08-1-0265, in part by the 973 Program of China under Grant 2012CB315803, in part by the National Natural Science Foundation of China under Grant 61133015, in part by the European Union (EU)-Japan GreenICN Project through the EU FP7 Programme and National Institute for Information and Communications Technologies, Tokyo, Japan, and in part by the Simulation Science Center through the Volkswagen Foundation and State Lower Saxony. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Dinei A. Florencio.

Q. Li is with the Graduate School at Shenzhen, Tsinghua University, Shenzhen 518055, China (e-mail: qi.li@sz.tsinghua.edu.cn).

X. Zhang is with Samsung Research America, Santa Clara, CA 95134 USA (e-mail: xinwenzhang@gmail.com).

Q. Zheng is with the Huawei Research Center, Santa Clara, CA 95050 USA (e-mail: qingji.zheng@huawei.com).

R. Sandhu is with the Institute for Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: ravi.sandhu@utsa.edu).

X. Fu is with the Institute of Computer Science, University of Göttingen, Göttingen 37073, Germany (e-mail: fu@cs.unigoettingen.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2014.2365742

requesting it. Thereby, LIVE can ensure that content access performed by each NDN node is under the CP's control since NDN nodes cannot access corrupted contents.

There are several challenges to realize efficient integrity verification in our architecture. Traditional signature schemes impose three important challenges in NDN. (i) Lightweight: Traditional signature schemes (e.g., RSA and DSA) are heavyweight, and introduce significant computation overhead, which may not be acceptable for NDN nodes serving content packets for large-scale traffic. Specially, routers have limited computation resources that are used primarily for content routing and forwarding. (ii) Practicality: In traditional signature schemes, it is not easy to revoke public keys so that it may not be possible to revoke content verification permissions assigned to content routers or users at run time. (iii) Simplicity: Traditional signature schemes require public key management infrastructures which require verifying the "trust chain" of public keys before verifying signatures. This complexity impedes their deployment.

To address these challenges, LIVE adopts one-way hash functions [3], [4] to produce content signatures such that integrity verification is done by verifying hash-based signatures. In particular, it uses Merkle Hash Tree algorithm [3] to generate tokens to sign and verify contents, and uses standard hash functions, such as SHA-1/SHA-2, to generate final content signatures. Tokens are generated for NDN nodes according to a CP's security policies. In this setting, content integrity verification performed by NDN nodes are completely controlled by the CP. Thus, LIVE achieves: (i) Lightweight: one-way hash functions are lightweight enough for NDN nodes to verify signatures and content integrity; (ii) Practicality: content caching can be easily revoked by CPs by changing tokens used to sign contents; (iii) Simplicity: Tokens are easily generated and flexibly distributed by CPs with a flat architecture that does not require "trust chain" among different tokens.

The contributions of this paper are three-fold:

- We propose a lightweight integrity verification architecture (LIVE) that emphasizes controlling content access in NDN by leveraging content integrity verification. We also analyze CPs' security policies for content access in NDN nodes to ensure secure content access.
- We propose a lightweight signature scheme by leveraging hash functions to realize lightweight, practical, and simple integrity verification in LIVE. Therefore, it effectively minimizes the computation overhead in verifying content integrity and authenticity so as to enforce CPs' security policies in a lightweight manner.
- We implement LIVE in CCNx [5]. Our experimental results show that it introduces acceptable overhead. In particular, LIVE only incurs 350$\mu s$ delays in content access, and achieves more than 20 times improvements compared to traditional signature schemes.

The paper is organized as follows. We briefly review the NDN design and discuss its security challenges in Section II. Section III presents the overview of LIVE. Section IV presents the lightweight integrity verification schemes. Implementation and performance evaluation are presented in Section V. Related work is discussed in Section VI. The paper concludes in Section VII.

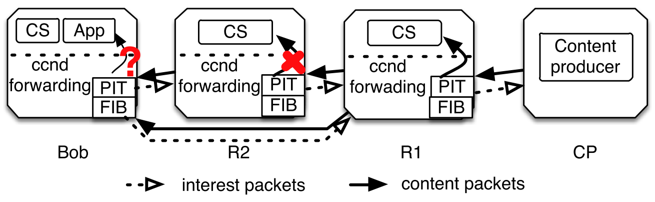

Fig. 1. An example of NDN communication among nodes.

## II. Background

### A. Basics of NDN

NDN is a new network architecture that delivers packets by content names but not packet addresses [1]. An end user or device sends an *interest* packet with desired content name to NDN. NDN routers forward the interest via a name prefix-based routing table called *forwarding information base (FIB)*, and record the request in a table called *pending interest table (PIT)*. When the interest reaches the content owner or publisher, it responds with a *data* packet. The intermediate routers forward the data packet to the original requester by checking their PITs. At the same time, each router caches the data packet in its local *content store (CS)*, which can be used to satisfy future interests with the same content name. In general, a content object (e.g., a video file) may be split into multiple data packets, each of which has the same name prefix but different full packet name, e.g., with block id as part of the packet name [1]. Open source project CCNx implements basic data structures and protocols of NDN [5]. A daemon program `ccnd` is implemented on each network node to forward interest and data packets, and cache data packets. Also, `ccnd` can be set up as a client to process interests and deliver data packets to applications.

In this paper, we understand an *NDN router* to be a general network node running `ccnd`. *Users* are nodes requesting and consuming contents. An *NDN node* is either an NDN router or a user device which runs `ccnd` and implements NDN functionalities.

Figure 1 shows an example of communication in NDN. User Bob's device sends a content request by sending out an interest packet with the content name. According to the setup of the FIB in Bob's `ccnd`, the interest is forwarded to router R2, which first checks if its local CS has the data packet with the same name, and if so it returns the data. Otherwise, R2 checks if the same request has been recorded in its PIT, if so it adds the requesting interface to the PIT entry and drops the request. Otherwise, it forwards the interest to next hop by checking its FIB which indicates which interfaces it should forward the interest to, based on the content name prefix carried in the interest packet. In Figure 1, R2 forwards the interest to R1, which does similar checks and forwards the interest to the CP.

When a data packet is returned from the CP or any NDN router, the `ccnd` forwarding logic in a router along the down-stream path forwards the packet to all interfaces that

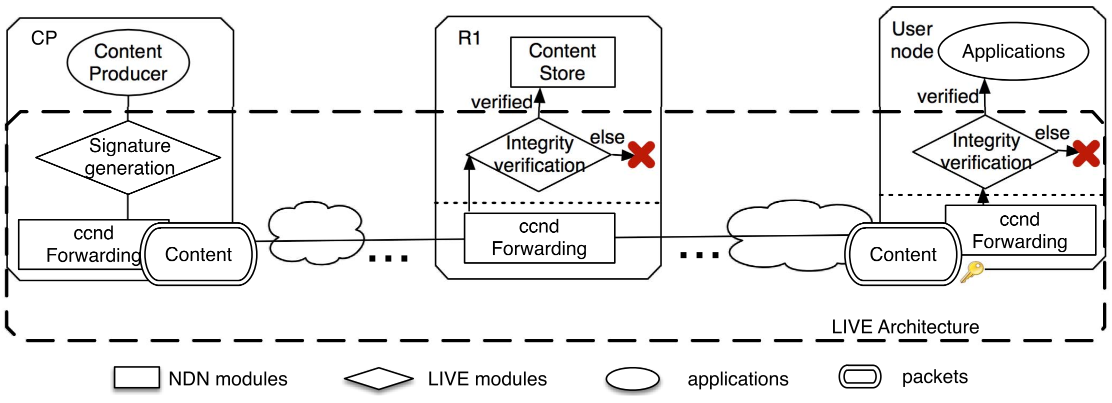

Fig. 2. Securing NDN with the LIVE architecture.

have sent requests for that content, by checking its local PIT, and flushes the corresponding PIT entries. In the meanwhile, the `ccnd` delivers the data packet to its local CS for content caching. At the end, the data packet is forwarded back to the requesting device, where the `ccnd` forwards it to the requesting application that consumes the data. In this paper, we understand *content accessing* to mean content caching in NDN routers and content consumption in end devices.

### *B. Content Protection Challenges in NDN*

Previous work has identified several security benefits and challenges in different aspects of NDN design [1], [6], [7]. For example, an NDN router directly uses content names to request contents and content request packets do not have address notations. This achieves certain level of privacy protection for client by default. Furthermore, each content data packet is digitally signed such that any NDN node can verify the integrity and authenticity of the content, no matter where the content is retrieved, e.g., from its original CP or any other NDN router. However, towards content protection, we identify that NDN has two important security challenges.

*1) Uninsured Content Integrity:* NDN nodes cannot efficiently verify content integrity and authenticity because of the verification cost, though NDN design demands signature provisions in data packets [7]. Therefore, malicious users or routers can arbitrarily inject corrupted and fake contents into the networks with the names of benign contents so that NDN routers or users cannot obtain the correct contents. For example, R1 in Figure 1 can respond to the requests from R2 with fake data packets whose name matches that specified in interest packets. As discussed in [7], implementing a heavyweight integrity verification mechanism in NDN routers introduces DoS threats to the network infrastructure.

*2) Unauthorized Content Access:* For many cases, e.g., business conflicts or privacy reasons, a CP usually has policies on which network domains or routers can cache its contents in network. Similarly, a CP usually has strong incentive to control the end users which can access its contents, although the contents are cached in the network. In the current NDN design, a CP loses the control of content access after the contents are delivered. For example, R2 in Figure 1 can arbitrarily cache contents when it delivers the contents, though the CP does not approve it. In actual practice CPs may only allow limited network routers to cache contents that are delivered over NDN routers, e.g., only R1 is allowed to cache the contents from the CP in Figure 1. Also, the user in Figure 1 can easily access contents that are cached by NDN routers by specifying the content names in interest packets. Thus, NDN should enable controllable content access performed by different NDN nodes.

For an intuitive approach to prevent unauthorized content access, one can leverage end-to-end content encryption to enable content access control in NDN [6], [8], [9]. However, end-to-end content encryption may greatly restrict the benefits to NDN, since NDN aims to improve content forwarding performance by providing caching mechanisms. Contents may be encrypted with different keys along with time to ensure temporal access control. Hence, the end-to-end encryption may disable the mechanism because the cached contents can only be decrypted in some specific nodes and cache retrieval by other nodes is in vain. Moreover, these approaches can be only used to secure contents with high confidentiality because enabling encryption for all contents introduces significant delays during content forwarding. Specially, routers may not have enough computation resources to process encrypted contents in NDN [7]. Therefore, it is important to design a generic and lightweight content access control architecture for NDN to prevent unauthorized content access in NDN and enhance its security.

## III. LIVE Overview

### *A. Overview*

LIVE aims to achieve the following design goals:

- *Lightweight content integrity and authenticity verification* to prevent NDN routers and users from accessing "corrupted" or "fake" contents.
- *Lightweight content security policy enforcement* to prevent unauthorized content accessing performed by NDN routers and users.

The main mechanism of LIVE lies in generating different content integrity status for a single content object, which allows a CP to control content access performed by NDN nodes. As shown in Figure 2, it introduces a signature generation module in CP to generate content signatures, which express content integrity and authenticity status, and an integrity verification module in NDN nodes, i.e., NDN routers and end devices, to verify the content status. With LIVE, CPs can efficiently generate signatures for contents, and NDN nodes will mandatorily verify content signatures before accessing the contents. LIVE leverages one-time signature schemes [3], [4] to realize lightweight content verification. This section gives a brief overview, and next section presents the detailed design.

*1) LIVE Initiation and Token Retrieval:* A CP classifies different NDN nodes into two categories for a content object (or a collection of content objects) according to its security policies, and generates different tokens for them. Specifically, NDN nodes that are authorized to access the content are in one category, which obtain private tokens, and others retrieve public tokens For example, as Figure 2 shows, assume R1 is not authorized to access the content from the CP, thus it obtains public token $P^\star$ from the CP, and the user node attached to R1 that is authorized to access the content can retrieve private key $P$ from the CP. Different NDN nodes need to explicitly request tokens from the CP before accessing contents.

*2) Content Signing:* A CP generates one-time content signatures [3], [4] with different tokens using the signature generation module. Normally, the CP generates two signatures for each content data packet, with the tokens $P^\dagger$ and $P$ that are assigned to routers and users, respectively. For example, as shown in Figure 2, the CP uses the tokens corresponding to $P^\dagger$ and $P$ to sign the outgoing content. The content is then forwarded back to the content requesting user with the signatures piggybacked.

*3) Content Verification:* An NDN router forwards content data packets to requesters according to its PIT. In the meanwhile, the verification module of the node verifies content status by verifying the attached content signatures before delivering them to content store (CS) or user applications. If a signature is verified, it means that the content packet is not corrupted and the node is authorized to cache the data. Otherwise, integrity verification module drops the packet to prevent corrupted or unauthorized content accessing. For example, in Figure 2, R1 forwards a data packet to its next hop. In the meanwhile, it uses $P^\star$ to verify the content signature in the integrity verification module. Based on the signature generation in the CP, R1 cannot successfully verify it with $P^\star$ since the content signature is generated with $P$. Therefore, the verification shows that the packet is "corrupted", and R1 does not save it in its local CS. Similarly, on a user node, the integrity verification module verifies integrity with received token before the packet is delivered to a local application. Here, the user node can verify the content signature with $P$ and then the packet is delivered to the application.

We note that content integrity verification is not performed in `ccnd` forwarding logic (see Figure 2) but before content packets are delivered to CS or user applications. Therefore, LIVE does not incur any content forwarding delay introduced by the integrity verification processes. If a data packet is not retrieved from an NDN router's Content Store, the content integrity verification is left to end users, which eventually prevents the user accessing corrupted contents. We present the details of our signature scheme in Section IV.

*4) Content Confidentiality:* For highly sensitive content, confidentiality is a desired requirement, i.e., only authorized end users can obtain the content. Access control relying on integrity verification is not sufficient for this requirement. LIVE adopts a lightweight encryption mechanism, where encryption keys are derived from integrity verification tokens. With this option, a CP can seamlessly support strong content access control for confidentiality by controlling who can obtain the tokens.

## *B. NDN Security Polices*

In LIVE, CPs realize content access control by enforcing security policies. Security policies for each content object (or objects with same name prefix) can be defined by assigning different security levels with respect to the nodes in the network. Content objects in NDN can be at the following three security levels.

- *Non-Cacheable:* Content should not be cached by any NDN router, and only authorized users are allowed to access the content.
- 1-*Cacheable:* Content can only be cached by one NDN router[1] after it is sent out to one of its CPs' neighbors, and only authorized users are allowed to access the content.
- *All-Cacheable:* Content can be cached by all NDN routers or accessed by users. Content at this level is publicly available in the network.

Note that, according to the NDN design, a CP cannot know requesting users and delivery paths, and it also cannot know any remote NDN routers that are requesting the contents. Therefore, for 1-Cacheable level, what the CP can do is only to allow content access performed by users and its first hop neighbor NDN routers. It is usually meaningless for a CP to make contact with any remote NDN router and allow it to cache contents in networks. If any NDN router wants to make contract with the CP and cache contents from the CP, it is required to build a peering link with the CP, which is similar to the practice of inter-domain routing operations in the current Internet [10]. Since the security level of 1-cacheable is not for universal control over all content objects but for sensitive ones that need protection, it does not violate the design goal of NDN. Actually, CPs can easily extend the content security level of 1-cacheability to $k$-cachability, where $1 \leq k \leq m$ and $m$ is the number of NDN nodes in the forwarding path between the CP and the destination, by distributing private tokens to the corresponding NDN nodes (see Section IV). In this paper, for simplicity but without loss of generality, we only discuss the enforcement of above three security levels.

Here, we assume that a CP has complete identity information of authorized NDN nodes. Since each NDN node has an assigned public key, which is specified in NDN design [5], we can directly use the hash value of the public key as the identity of the node. The authorized NDN nodes are classified

[1]In practice, a set of NDN routers within one domain will be authorized to access the contents, which will be discussed in Section IV-C.

into two sets, one set for authorized NDN routers and another set for authorized users. For a given content name (or a name prefix) $C$, its CP can include the authorized user identities in $R_C$, and include authorized NDN routers into $R_C^\dagger$. These are the set of nodes that are authorized to access content $C$ and thus have to receive private tokens. Other unclassified NDN nodes can receive public tokens by default, which means that these routers are not authorized to cache content $C$. The private tokens retrieved by NDN routers in $R_C^\dagger$ are used to enable 1-cacheability of $C$. In reality, a CP may only need to know the identity of a domain of authorized nodes, e.g., one router in the domain, since usually all routers in one domain have the same privilege to cache contents.

Now we have security policies specifying content security levels that are defined with respect to neighbor requesters. Content security level information is embedded in content packets and covered by content signatures that are used to authorize NDN nodes using the corresponding tokens owned by the nodes, which is similar to that in capability-based systems [11], [12]. Therefore, LIVE shares the same benefits with the capability-based system [11], [12]. Each NDN node can be directly authorized to access the packets by using simple cryptographic operations, e.g., signature verification, with the packets themselves, and it does not need to contact the packet producers to retrieve the authorization information during runtime packet delivery. Logically, a security policy for content $C$ can be expressed as $(R_C^\dagger \mapsto P^1,\ R_C \mapsto P^2)$, which states that the set of NDN routers $R_C^\dagger$ and the set of users $R_C$ are authorized to access $C$ using tokens $P^1$ and $P^2$, respectively. $C$ is non-cacheable if $R_C^\dagger = \emptyset$, and $C$ is all-cacheable if $R_C^\dagger = \cup^{R_i}$. Note that, $P^1$ and $P^2$ are two different tokens generated by a CP, and are not differentiable for different NDN nodes. NDN nodes in the same set will be assigned with the same token to access the contents produced by the CP. The token will be updated and synchronized during content retrieval (see Section IV).

## IV. Design of LIVE

This section presents the detailed design of content integrity verification in LIVE. Normally, signature algorithms built upon asymmetric cryptography algorithms, e.g., RSA and DSA, are used to verify data integrity and authenticity. These algorithms introduce significant computation overhead. Hash functions are lightweight for integrity verification. However, pure hash-based message authentication codes (MAC) are forgeable and thus cannot be used to verify content authenticity. Although keyed-hashing for MAC (HMAC) [13] is used for authenticity purpose, it does not address how to refresh shared keys. Inspired by one-time signature algorithms [3], [4], we extend hash functions to generate tokens, produce signatures, and verify signatures, which ensures that all operations are lightweight and could be implemented by hardware.

In particular, we leverage the Merkle Hash Tree (MHT) algorithm [3], a hash tree based signature algorithm, to produce tokens for signature generation. CPs realize their security policies by producing different signatures with different tokens for a single content packet such that the content access can be controlled by distributing different tokens to different NDN nodes. An NDN node can simply check content integrity with a received token and determine if the content packet that it is accessing is not "corrupted".

### A. LIVE Initialization and Token Generation

For a name prefix or a collection of content names, a CP generates the following tokens:

- *Public tokens:* All unauthorized nodes use this token to verify the integrity of the data packets of a content with 1- or non-cacheability, and the verification process fails.
- *Private tokens:* The CP generates two private tokens, for authorized NDN routers and end users, respectively. With a valid signature, the integrity verification with either private token succeeds.

Note that only the CP knows whether a token is public or private, while NDN nodes cannot distinguish them – they are obtained by NDN nodes with same protocol and used to verify integrity of content packets. With a private token, an NDN node will successfully verify the integrity and access the data; while with a public token, the NDN node will fail the verification, and thus consider the packet is corrupted or faked and drop it.

*1) Token Generation and Distribution:* To initialize LIVE, the CP needs to produce initial tokens for different nodes. LIVE generates a random number to construct a key vector $X$ that consists of $n$ key components, and uses the key vector to generate the tokens using standard hash algorithms, such as SHA-1. Note that, for the purpose of reducing communication overheads for content integrity verification, we use function $f(\cdot)$ to extract lower $l$-bits of hash values as the seed for token generation. In this paper, we set $l$ and $n$ to 32.

```
Stage (I): Token generation
1. Generate key vectors $X^k = \{x_1^k, \cdots, x_j^k, \cdots, x_n^k\}$, where
   $x_j^k \in \{0,1\}^\lambda$, given $n \in \mathcal{Z}$, where $k = 0, 1, 2$;
2. $P^k \leftarrow h(f(h(x_1^k))||f(h(x_2^k))||\cdots||f(h(x_n^k)))$, where $P^0$
   is in the category of public token $P^\star$, and $P^1$ and $P^2$ are
   in the category of private tokens $P^\dagger$ and $P$, respectively;
Stage (II): Policy generation
3. Generate two categories of the router sets $R_C^\star$ and $R_C^\dagger$ with
   respect to content $C$, where $R_C^\star$ denotes the set of routers
   receiving public token $P^0$ to verify $C$, and $R_C^\dagger$ denotes the
   router set receiving private token $P^1$ to verify $C$;
4. Generate the categories of authorized user sets $R_C$ receiving
   private token $P^2$ to verify content $C$;
Stage (III): Token distribution
5. Receive token request from node $i$ that are in $R_C^\dagger$ or $R_C$;
6. $P^k \leftarrow \mathrm{ENC}_{PK_i}(P^k)$, where $k$ is equal to 1 if $i$ is in $R_C^\dagger$,
   otherwise it equals to 2;
7. Send out $P^k$ to node $i$;
Stage (IV): Token refreshment
8. Generate new key vectors $X^{k\prime} = \{x_1^{k\prime}, \cdots, x_j^{k\prime}, \cdots, x_n^{k\prime}\}$,
   for $R^\star$, $R^\dagger$, and $R$, where $x_j \in \{0,1\}^\lambda$, given $n \in \mathcal{Z}$;
9. $P^{k\prime} \leftarrow h(f(h(x_1^{k\prime}))||f(h(x_2^{k\prime}))||\cdots||f(h(x_n^{k\prime})))$;
10. $P^{k\prime} \leftarrow ENC_{P^k}(P^{k\prime})$ if $k = 1$ or 2;
11. Embed $P^{k\prime}$ in the content and distribute them to different
   requesters.
```

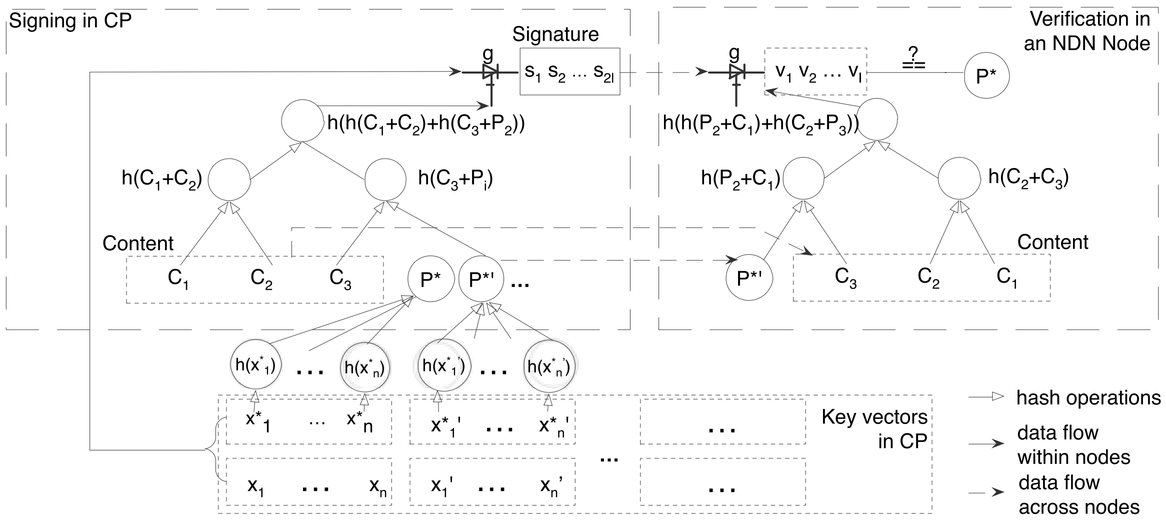

Fig. 3. A signature on a sample content consisted of three content blocks.

The CP needs to generate two private tokens for authorized routers and users, e.g., R1 and Bob in Figure 1, and a public token for the other NDN nodes, e.g., R2 in Figure 1.

As we have discussed in Section III, for a content object $C$, the CP should classify different NDN nodes into two sets, $R_C^{\star}$ and $R_C^{\dagger}$. The CP should build a list of all identities of nodes composing a given set of content nodes, e.g., authorized users. The authorized user nodes to access content $C$ are classified into set $R_C$. Two private tokens are distributed to the router nodes in $R_C$ and the user nodes in $R_C^{\dagger}$, respectively, if these two set are not empty. The public token is distributed to the set of unauthorized nodes in $R_C^{\star}$.

Before verifying the integrity of $C$, a node needs to retrieve its token from the CP, by sending out a token requesting interest packet to the CP. The packet includes the public key of the requesting node that is already embedded in the NDN node [1], [5]. Since the CP knows the identity (hash of the public key) of the authorized node, it responds the interest with the corresponding token encrypted by the requesters' public key. A faked request (unauthorized node requesting token with authorized node's public key) will fail since the requesting node cannot decrypt the token without corresponding private key. This ensures that only authorized NDN nodes can obtain the corresponding tokens. We note that this public-key based token retrieving is one-time for a content name or a CP on each NDN node, since the token update for data packets is conducted by encrypting new tokens with previous tokens and embedding them in data packets directly (cf. Section IV-C).

Each NDN node only needs to store at most two tokens from one CP, i.e., the public token and private one. Moreover, in real deployment, routers in a domain, e.g., an ISP, do not need to retrieve tokens by themselves. Normally, a domain can have a designated router that can retrieve tokens from different CPs and then distribute to other routers in the domain [14], [15].

### B. Signing and Verifying Data Packets

Figure 4 illustrates the basic procedure of content signing and verification. A CP uses $P^{*\prime}$ (which can be $P^{\star\prime}$ or $P^{\dagger\prime}$) and the content to generate a hash value $g$ by using the MHT algorithm. The MHT algorithm constructs a binary tree where each node is associated with a bit string. The bit strings of the leaves are the hash values of the contents or the new token, and the bit strings of internal nodes are the hash values of their left and right nodes' bit strings. Figure 3 shows the signing procedure for a content object which is divided into three data blocks (packets). The content blocks together with the tokens are the leaf nodes of MHT. Normally in NDN, the same content may be sent from or to different neighbors with different tokens. With MHT, NDN nodes only need to construct a complete tree for a neighbor, and can quickly re-construct $g$ for other neighbors by replacing part of the leaf nodes, i.e., the token nodes in the tree [3]. For example, in MHT shown in Figure 3, to generate a new token, only the right part of the tree will be recomputed. Thus, MHT can effectively reduce computations in constructing tokens during signature signing and verification processes. Note that, the CP can use the precomputed tokens to efficiently generate signatures. In particular, it can reuse the tokens generated during token refreshment. Note that, token $P$ is not directly used to generate a signature. Therefore, any NDN node except CPs cannot regenerate a signature using $P$. To prevent compromising the tokens, the tokens will be encrypted (see Section IV-C), and then the MHT will be built upon the encrypted tokens.

In the content signing procedure, $g$ determines whether part of each component in the key vector $\{x_1^*, \cdots, x_n^*\}$ and $\{x_1, \cdots, x_n\}$ or part of the hash value of the component is used to produce each signature component, i.e., $\{s_1, \cdots, s_{2l}\}$. Note that, the key vectors $\{x_1^*, \cdots, x_n^*\}$ and $\{x_1, \cdots, x_n\}$ were previously used to generate $P^*$ that was received by the corresponding authorized nodes.

The content verification procedure is similar to the signing procedure. When the signature gets to a NDN node, the node will use the received $P^{*\prime}$ and the content blocks to generate $g$. A new bit vector $\{v_1, \cdots, v_l\}$ will be produced by using $g$ to determine if using part of $s_i$ in $\{s_1, \cdots, s_{2l}\}$ or part of the hash value of $s_i$ as the vector component. If the concatenated

**Algorithm 1** LIVE Signing Algorithm

```
Input: Content $C$, Router Set $R_C^\dagger$, Key vector $X^\star$ for the normal routers, Key vector $X^\dagger$ for CR, Key vector $X$ for authorized user nodes, content requester router $i$;
Output: Content signature $S$;
 1: Generate a token key vector $X^{*\prime}$, where $X^{*\prime} = \{x^{*\prime}_1, x^{*\prime}_2, \cdots, x^{*\prime}_n\}$;
 2: $P^{*\prime} \leftarrow h(f(h(x^{*\prime}_1))||f(h(x^{*\prime}_2))||\cdots||f(h(x^{*\prime}_n)))$;
 3: if ($C$ is non-cacheable) then
 4:     $\{y_1, y_2, \cdots, y_{2l}\} \leftarrow X \,||\, X$;
 5: else if (($i \in R_C^\dagger$) && ($C$ is 1-cacheable)) then
 6:     $\{y_1, y_2, \cdots, y_{2l}\} \leftarrow X^\dagger \,||\, X$;
 7: else if ($C$ is all-cacheable) then
 8:     $\{y_1, y_2, \cdots, y_{2l}\} \leftarrow X^\star \,||\, X$;
 9: end if
10: $g \leftarrow$ MHT$(C + P^{*\prime})$;
11: $g \leftarrow f(g) \,||\, f(g)$;
12: for ($j = 1 \rightarrow 2l$) do
13:     if ($g_j = 0$) then
14:         $s_j \leftarrow f(h(y_j))$;
15:     else
16:         $s_j \leftarrow y_j$;
17:     end if
18: end for
19: $S \leftarrow s_1||s_2||\cdots||s_{2l}$;
```

value of each component in $\{v_1, \cdots, v_l\}$ is equal to $P^*$, which means that the signature is successfully verified, $P^{*\prime}$ will be used to update $P^*$ and verify the signature of the next content from the CP.

*1) Signing Procedure:* Algorithm 1 illustrates the signing algorithm in LIVE performed by a CP. To generate a signature for a content, the CP needs to generate a new token key vector $X'$ and uses the hash algorithm to produce a new token (steps 1-2). If a content is non-cachable, the CP generates a new bit vector $\{y_1, y_2, \cdots, y_{2l}\}$ by concatenating two $X$s, where $X$ is the key vector associated with tokens delivered to users (step 4), which ensures that only users can verify the generated signatures. The new bit vector is used as the seeds for signature generation. That is, the first part of the bit vector is used to generate signatures for routers, and the second part is used to generate signatures for users.

Similarly, if the content is 1-cachable and the content requester is CR, the CP produces the bit vector $Y$ by concatenating $X^\dagger$ and $X$, where $X^\dagger$ was generated for router $i$ (step 6). If the content is all-cachable, the CP needs to produce the bit vector by concatenating public tokens $X^\star$ and $X$. In step 10, the CP uses the MHT algorithm to generate a bit vector $g$. Then, it generates signature $S$ according to bit vector $g$. Since each $g$ generates one part of the signatures, either for routers or for users, we need to concatenate $g$ to generate the signature for both NDN routers and users (step 11). The CP uses lower $l$ bits of $h(y_i)$ as part of a signature if the bit in $g$ is equal to 1 (step 14); otherwise it directly uses $y_i$ as part of the signature (step 16). Finally, a signature produced by concatenating all signature parts (step 19). After the computation, a content packet under delivery is consisted of the content payload, the content signature, one token that is used to verify the signatures and refresh the token in the NDN router node, and one token updating the token in the user nodes (see Section IV-C).

**Algorithm 2** LIVE Verification Algorithm

```
Input: Content $C$, Content Signature $S = s_1||s_2||\cdots||s_{2l}$ , Content public key $P^{*\prime}$, Local token set $\mathcal{P}$;
Output: true: accepting $C$; false: rejecting $C$;
 1: $g \leftarrow$ MHT$(C + P^{*\prime})$;
 2: if ($S$ is verified by routers) then
 3:     $m \leftarrow 0$;
 4: else
 5:     $m \leftarrow l$;
 6: end if
 7: for ($j = m + 1 \rightarrow m + l$) do
 8:     if ($g_j = 1$) then
 9:         $v_j \leftarrow f(h(s_j))$;
10:     else
11:         $v_j \leftarrow s_j$;
12:     end if
13: end for
14: $V \leftarrow h(v_{m+1}||v_{m+2}||\cdots||v_{m+l})$;
15: if (V matches P $\in \mathcal{P}$) then
16:     return true;
17: else
18:     return false;
19: end if
```

*2) Verification Procedure:* Algorithm 2 shows the signature verification algorithm performed in an NDN node, which is similar to the signing algorithm. Firstly, the NDN node computes the bit vector $g$ with the MHT algorithm according to the received content and the token $P^{*\prime}$ embedded in the content packet (step 1). The node directly uses each signature $s_i$ in $S$ to construct a new bit vector $V$. Note that if the NDN node is a router, it only needs to use the first part of signature components in $S$, i.e., $s_1||s_2||\cdots||s_l$, to generate the components in $V$ (step 3). Similarly, if the NDN node is a user node, it only needs to use the second part of signature components in $S$, i.e., $s_{l+1}||s_{l+2}||\cdots||s_{2l}$, to compute the components in $V$ (step 5). Similar to Algorithm 1 (see steps 12-18), the NDN node computes all required signature parts $s_j$ according to the bits in $g$ (see step 7-13).

After constructing all required bit components, the algorithm concatenates $v_j$ and generates the hash value to obtain the final signature $V$ (step 14). Now we can compare the signature with the tokens that the NDN node previously received from the CP. If V matches the token in $\mathcal{P}$, i.e., $P^\star$, $P^\dagger$, or $P$, it means the content integrity is successfully verified and the node is authorized to access the content. That is, the content can be delivered to the Content Store (if the NDN node is a NDN router) or to user applications (if the NDN node is a user). Otherwise, the content is simply dropped.

### C. Token Refreshment

LIVE uses one-time content signature (OTS) [3], [4] to realize lightweight content verification. Therefore, a token has to be refreshed once it has been used. In general, the CP generates new tokens to produce content signatures (see Algorithm 1). These new tokens can be piggybacked in content packets and updated to different nodes during content delivery so that different NDN nodes can update their tokens. As discussed in Section III, new generate token $P^{*\prime}$ is encrypted with the previous token $P^*$ that were received in the last round of content delivery. For example, if $P^* = P^0$, $P^{*\prime}$ is equal to $P^{0\prime}$ and used to update public token $P^0$ in NDN routers. Otherwise, $P^{*\prime}$ is used to update the private token $P^1$. Moreover, $P^{2\prime}$ is directly piggybacked in the content packets to update $P^2$ in the user nodes. Authorized NDN nodes can securely refresh the local tokens by decrypting the received tokens after successfully verifying the content signature. As discussed above, public tokens normally are public available and may not be encrypted during content delivery. Note that, tokens embedded in the contents are delivered with token version information, which indicate whether the tokens in the NDN nodes have been expired. If authorized NDN nodes do not receive contents from a CP for a period, their tokens may expire. They need to synchronize their tokens by retrieving the latest tokens from the CP.

## *D. Content Encryption and Decryption*

The goal of LIVE is to enable a generic and lightweight content verification and cache access control in NDN nodes. However, it cannot prevent attacks from malicious NDN nodes in network. For instance, an attacker can access sensitive contents by compromising NDN nodes and launching man-in-the-middle (MITM) attacks. In particular, MITM attacks can be launched by malicious nodes between two benign NDN nodes, and the malicious nodes can capture contents by content packet sniffing.

To address this type of attacks, we extend the basic verification scheme and incorporate content encryption in LIVE. If a content is sensitive, CP can encrypt the content before it is distributed, such that the whole content cannot be read without a decryption key even if an unauthorized node obtains it. The encryption key is embedded in the private token. Therefore, an attacker cannot access the content, even if he can obtain the content by bypassing the content verification logic. However, an authorized NDN node can still correctly decrypt the correct content blocks during signature verification according to the information embedded in the tokens. the performance does not reduce significantly. Only users who have correct tokens can verify and decrypt the contents. Therefore, the mechanism enables ensured content access control.

Different from the basic verification scheme discussed in Section IV-B, before generating content signatures, the CP encrypts the content blocks using authorized users' tokens. Contents can be partially or fully encrypted, e.g., only a subset of blocks are encrypted, which is by authorized users' tokens as well. Algorithm 3 shows the signing algorithm with content encryption. The CP uses token $P$ to decide whether a content is partially or fully encrypted and which content blocks will be encrypted if the content will be partially encrypted. The number of 1-bits in token $P$ is divided by $l$, and the remainder is assigned to $i$ (step 5). If $i$ is less than $l/2$, the $i$th content block will be encrypted with token $P$ as the encryption key (steps 6-7). If $i$ is larger than $l/2$, the entire content will be encrypted (steps 8-10). The rest of the algorithm is similar to Algorithm 1. Similarly, in the verification algorithm (i.e., Algorithm 4), the content will be decrypted according the computed $V$ if $V$ matches $P$ (steps 1-11). Note that, the encrypted content will be decrypted only in NDN user nodes.

**Algorithm 3** LIVE Signing Algorithm With Content Encryption

```
Input: Content $C$, Router Set $R_C^{\dagger}$, Key vector $X^{\star}$ for the normal
    routers, Key vector $X^{\dagger}$ for CR, Key vector $X$ and the corre-
    sponding token $P$ for authorized user nodes, content requester
    router $i$;
Output: Content signature $S$;
 1: Generate a token key vector $X^{*\prime}$, where $X^{*\prime} = \{x^{*\prime}_1, x^{*\prime}_2, \cdots, x^{*\prime}_n\}$;
 2: $P^{*\prime} \leftarrow h(f(h(x^{*\prime}_1))||f(h(x^{*\prime}_2))|| \cdots ||f(h(x^{*\prime}_n)))$;
 3: if ($C$ is non-cacheable) then
 4:   $\{y_1, y_2, \cdots, y_{2l}\} \leftarrow X \;||\; X$;
 5:   $i$ = count($P$) mod $l$;
 6:   if ($i < l/2$) then
 7:     $C_i \leftarrow \text{ENC}_P(C_i)$;
 8:   else if $i > l/2$ then
 9:     $C \leftarrow \text{ENC}_P(C)$;
10:   end if
11: else if (($i \in R_C^{\dagger}$) && ($C$ is 1-cacheable)) then
12:   $\{y_1, y_2, \cdots, y_{2l}\} \leftarrow X^{\dagger} \;||\; X$;
13: else if ($C$ is all-cacheable) then
14:   $\{y_1, y_2, \cdots, y_{2l}\} \leftarrow X^{\star} \;||\; X$;
15: end if
    {The rest of the algorithm (steps 16-25) is the same as steps 10-19
    in Algorithm 1.}
```

**Algorithm 4** LIVE Verification Algorithm With Content Decryption

```
Input: Content $C$, Content Signature $S = s_1||s_2|| \cdots ||s_{2l}$ , Content
    public key $P^{*\prime}$, Local token set $\mathcal{P}$;
Output: true: accepting $C$; false: rejecting $C$;
    {The first 14 steps are omitted because they are the same as that
    in Algorithm 2.}
 1: if if (V matches P $\in \mathcal{P}$) then
 2:   $i$ = count($P$) mod $l$;
 3:   if (local is a user node) && ($i < l/2$) then
 4:     $C_i \leftarrow \text{DEC}_V(C_i)$;
 5:   else if (local is a user node) && ($i > l/2$) then
 6:     $C \leftarrow \text{DEC}_V(C)$;
 7:   end if
 8:   return true;
 9: else
10:   return false;
11: end if
```

## *E. Security Analysis*

LIVE generates MHT values to produce token $P$ and signs contents with the signature scheme $(P, h)$, where $P$ is the generated token and $h$ is a collision-resistant hash function. We can obtain the following theorem.

*Theorem 1:* Given a collision-resistant hash function $h$ and a token $P$, the proposed signature scheme $(P, h)$ in LIVE is unforgeable.

*Proof:* We prove the theorem by showing that any polynomial time algorithm breaks the unforgeability of the proposed signature scheme only with negligible probability in the random oracle model [16].

Recall that the proposed signature scheme is a one time signature. Without loss of generality, we assume that in some state the token key is $X = \{x_1, \ldots, x_n\}$ and the public key is $P = h(f(h(x_1))|| \ldots ||f(h(x_n)))$, and the adversary is intending to forge a signature scheme corresponding to the token key $X$.

Suppose that the adversary presents $(C', S')$, where $C'$ is the content, and $S'$ is the forged signature corresponding to $C'$. Given content $C'$, with the token $X$, we can generate the signature $S$. We are interested in the probability that $S = S'$ given $C'$, because in the verification algorithm the adversary makes the verification algorithm output true only if $S = S'$.

Moreover, in order to achieve $S = S'$, then for those $g_j = 0$, the adversary has to correctly compute $s'_j = f(h(x_j))$, where $x_j$ is unknown to the adversary. Therefore, the probability of generating output $s'_j = f(h(x_j))$ by adversary is $\frac{1}{2^l}$ where $h$ is modeled as a random oracle. In addition, for $g_j$ where $g_j = 1$, the adversary has to correctly compute $s'_j = x_j$, where $x_j$ is unknown to the adversary. Therefore, the probability of generating output $s'_j = x_j$ is $\frac{1}{2^{|x_j|}}$. Since $|x_j| \geq l$, we can obtain that the probability of generating output $S = S'$ by the adversary is less then $(\frac{1}{2^l})^l$.

We can conclude that $(\frac{1}{2^l})^l$ is negligible, where $l = 32$. Therefore, the probability of breaking the unforgeability of the signature scheme by the adversary is negligible. ■

### F. Discussion

*1) Token Update and Refreshment:* LIVE introduces key management complexity for NDN routers. However, it does not require verifying each content packet. In real deployment, the security mechanism and key management is only enabled by CPs producing sensitive contents. Therefore, LIVE can be a NDN working mode, which is used for a value-added service for CPs. Moreover, CPs can have some strategies to update and synchronize tokens with different NDN nodes. For example, CPs can update tokens according to per content delivery path so that the NDN nodes in the path can always update their tokens as long as a NDN node in the path closer to the CP caches the content. Therefore, token distribution triggered by token expiration does not often occur in LIVE, and token update and refreshment will not introduce significant computation and communication overheads to synchronize tokens. Note that, public tokens can be cached while private tokens are encrypted and protected.

*2) Scalability of LIVE:* As we discussed in Section III, for a given content, a CP needs to maintain three tokens for all NDN nodes, i.e., two private tokens for authorized NDN routers and users, respectively, that are used to verify content integrity and authenticity, one public token for other NDN nodes to verify content integrity and authenticity. Each NDN node needs to store one token to communicate with a CP. Therefore, for an NDN node, the entire cost of storing tokens is proportional to the number of CPs that the node communicates with. For the purpose of fine-grained access control, a CP can build different private tokens for different NDN nodes, and group-based key management scheme can be adopted to achieve this.

## V. Implementation and Evaluation

### A. Implementation

We have implemented LIVE in CCNx [5], an open source project implementing basic NDN data strictures and protocols. As Figure 4 shows, the `ccnd` implements the main NDN logic and sends all received data packets to CS or user applications by default. Our implementation extends `ccnd` with

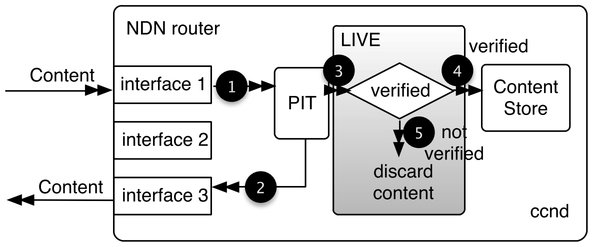

Fig. 4. LIVE implementation in CCNx.

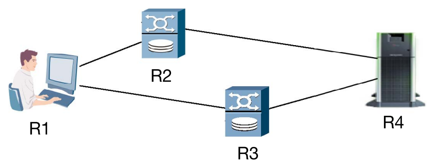

Fig. 5. A topology of LIVE testbed including a user node $R1$, a CR node $R2$, a normal router node $R3$, and a CP node $R4$.

a LIVE signing and verification library and LIVE Runtime. We implement the LIVE signing and verification library with C code. The LIVE Runtime engages in verifying content signatures and integrity status by calling the library functions at runtime, and generates token requesting interest packets to retrieve tokens from the CP. If content packets cannot be verified by the LIVE Runtime, they are directly dropped. Note that, the LIVE Runtime does not affect `ccnd` forwarding modules via flows (1) and (2) since it operates only when the contents are delivered to content store or user applications (flow (4)). Before content packets are delivered to content store, they will go through the LIVE module (flow (3)) to integrity and authenticity verification. Hence, LIVE does not introduce any overhead to content delivery procedures. In the next subsection, we will show that the LIVE Runtime introduces very low overheads in accessing contents.

Note that, LIVE does not detach caching from forwarding. With LIVE, NDN daemon will evaluate content integrity only when it decides if the content can be cached. It will not impact cache lookup upon an incoming interest packet.

### B. Performance Evaluation

We use our prototype to determine the performance of LIVE. Since content access delay incurred by verification is proportional to length of content packet delivery, for simplicity, we only evaluate the delay of two-hop content forwarding with and without caching. Figure 5 shows the testbed of our experiments, including one user node $R1$ and one CP node $R4$. Also, it includes two machines acting as NDN routers: $R2$ is a CR of the CP and can cache the contents from the CP, and $R3$ is the normal router that cannot cache the contents from CP. $R3$ forwards content requests received from $R1$ to $R4$.

We evaluate the performance of LIVE at $R1$ and $R4$ using Mac laptops with 2.53 GHz Intel CPU, 4GB RAM, and Mac OS 10.6.8. We investigate the token generation performance in the CP and measure the computation and content delivery delays introduced by LIVE in the user node R1. Also, since LIVE increases the content packet size by piggy-backing signatures, we measure the communication overhead

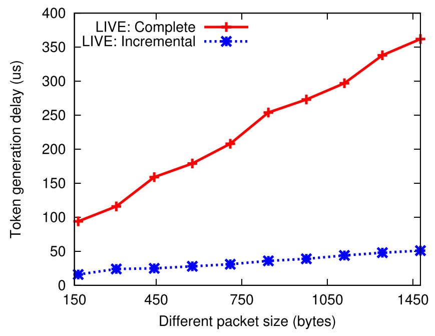

Fig. 6. Performance of token construction for content signing.

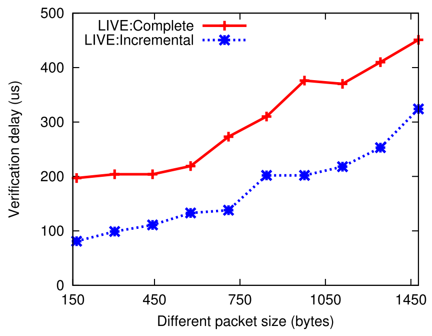

Fig. 8. Verification performance w/ token reconstruction.

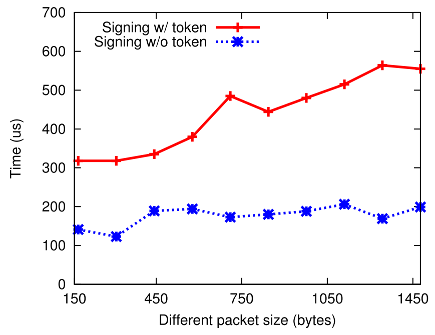

Fig. 7. Content signing performance w/ and w/o token generation.

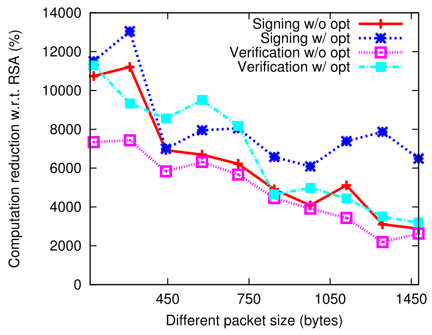

Fig. 9. Signing and verification delay reduction by LIVE.

incurred by LIVE. We evaluate the LIVE performance with different content sizes range from 150 bytes to 1480 bytes. To demonstrate the benefits of lightweight signatures in LIVE, we also implement a signing and verification library with the RSA algorithm with 1024-bit RSA keys by extending the OpenSSL library (OpenSSL-1.0.1c) [17].

*1) Token Generation Performance:* We measure the key generation performance in the CP. As Figure 5 shows, the CP needs to use different tokens to generate signatures for $R2$ and $R3$. LIVE leverages MHT hash algorithm to quickly reconstruct tokens for different routers. CP only needs to have a complete key generation process for the first router requesting the content, i.e., $R2$, and then reconstructs the token for the other routers, e.g., $R3$. Figure 6 shows the key generation delays for the two routers. The delays increase with increasing content size. On average, the delay to generate tokens for $R1$ is around 224$\mu s$. Since the CP only needs to change a leaf of the MHT tree to generate a new token, the key generation delay for the second router has a significant reduction. The delay is about 34.2$\mu s$ which is roughly 5 times less than for the first router. Hence, token generation in LIVE is efficient.

*2) Computation Overhead:* We measure the overheads for different content packet sizes. Figure 7 and 8 show the performance of content signing and verification. LIVE only takes averagely 439.4$\mu s$ and 386.10$\mu s$ to sign and verify contents. In particular, as we have discussed in Section IV, The CP can generate tokens in advance so that the content signing delay is significantly reduced. The content signing delay without token generation is about 176.2$\mu s$. Similar to the token generation process, the user node $R1$ can quickly reconstruct tokens for content verification when it receives the same contents from different routers. The content verification delay can be reduced by 70% (see Figure 8).

We also observe that the lightweight signature scheme introduces much smaller delays than RSA-based scheme. For example, RSA introduces 14,133.90$\mu s$ and 10,186.40$\mu s$ to sign and verify contents during content accessing, respectively. Figure 9 shows that LIVE improves about 62 times in the signing and verification computation overheads. Note that, since the number of the hash operations in LIVE is determined by the MHT values of contents and tokens, it is not deterministic. The signing and verification delays in RSA are relatively stable. We observe that delay reductions vary with different content packet sizes. LIVE has over 50 times delay reduction in verifying small content packets compared to RSA. In particular, as Figure 9 illustrates, LIVE obtains more reduction in signing and verifying contents if it uses pre-computed tokens in content signing and leverages incremental verification in content verification.

*3) Content Delivery Performance:* CPs produce different signatures to realize their security policies for the contents. It is clear that LIVE introduces delays in accessing contents because it checks content status by verifying the signatures. We measure the overhead in accessing contents in user applications in $R1$ with different distances, i.e., 1-hop distance from $R2$'s cache store and 2-hop distance from CP $R4$ via $R3$. We observe the similar delivery delay. Figure 10 illustrates the overhead incurred by LIVE, which indicates that LIVE only introduces around 7.8% delay in accessing contents. However, the RSA-based scheme incurs average 68% delay. We believe LIVE will introduce similar overheads if it is deployed on

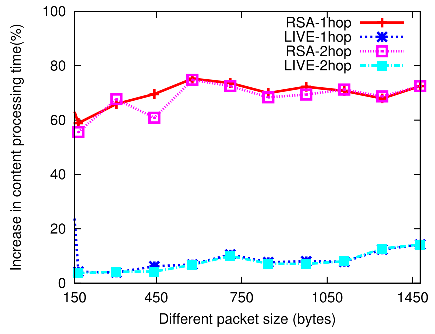

Fig. 10. Content delivery overhead: LIVE vs. RSA.

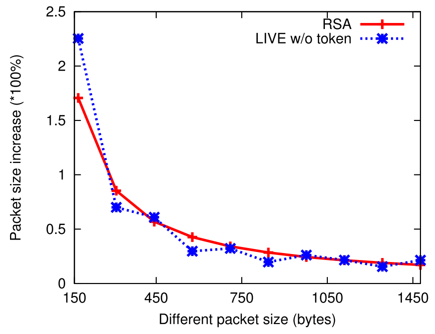

Fig. 11. Introduced communication overhead.

large scale networks because the incurred delays are bounded and not impacted by the length of content delivery paths. To further reduce the computation overhead, we can use high speed hardware implementations of MAC computation [18] in content signing and verification.

*4) Content Encryption/Decryption Performance:* Since intermediate NDN nodes do not decrypt content for content caching and forwarding, the content encryption and decryption only impact the CP and end users. In our scheme, we adopt symmetric key algorithm for content encryption, therefore we expect no significant performance overhead for CP and end users with content encryption and decryption.

*5) Communication Overhead:* LIVE requires piggybacking signatures in each content packet and incurs extra communication overhead. The signatures generated by RSA have a constant size. However, signature sizes in LIVE vary with respect to the sizes of the contents and the tokens. It consumes similar bandwidth cost to RSA and introduces similar communication overhead. Figure 11 illustrates the communication overheads introduced by RSA and LIVE. On average, RSA and LIVE incur about 50% communication overheads. Note that, since NDN is designed for named content delivery, the content packets normally have a larger packet size than IP. Fortunately, we can observe that both RSA and LIVE incur less communication overheads by increasing content packet sizes. In particular, LIVE only introduces average 15% communication overheads when the content packet sizes are larger than 750 bytes. We believe that the communication overhead is acceptable.

## VI. Related Work

Security has been considered in the design of content-centric network (CCN) and NDN [1] (which are essentially synonymous for our purpose). Gasti *et. al.* [7] explore the DoS and DDoS attacks to contents in NDN, and discuss that the current heavyweight integrity verification mechanisms cannot prevent them but introduce new threats. Recently Afanasyev *et. al.* [19] mitigate the DDoS attack to interest tables by maintaining the interest limit in each node and punishing malicious interests. In order to address the timing attacks on cache, Mohaisen *et. al.* [20] and Acs *et. al.* [21] take the approach of randomly introducing delays in responding to content requests and mimicking a cache hit and a cache miss for each request.

Efficient integrity verification mechanisms have been desired from the beginning of CCN design [1]. DiBenedetto *et al.* [6] apply onion routing to NDN, which uses multiple cryptogram operations to provide end-to-end content privacy protections. Nabeel *et al.* apply Paillier homomorphic cryptography to secure different messages in publish-subscribe networks [9]. Although these approaches can be used to prevent unauthorized content access, they may impair the performance benefits of NDN. The NDN designs improve the network performance using content caching mechanisms, but end-to-end content encryptions make content caching less efficient. Moreover, these approaches introduce significant overheads in processing and forwarding contents. It extends the NDN architecture and leverages content integrity verification to realize the security goals, i.e., verifying content integrity and authenticity and preventing unauthorized content access. Specially, LIVE only incurs processing overheads during content access but not in content forwarding procedures, which is different from these cryptography-based approaches that require encrypting and decrypting all content packets multiple times.

Fotiou *et. al.* [2] propose a delegation-based access control architecture for ICN. However, the solution relies on state maintenance and synchronization between content routers and access control policy servers, therefore scalability becomes a major issue for large-scale content networks. LIVE simplifies the access control protocol by integrating it into integrity verification, which is more scalable and efficient.

Lauinger *et al.* [22] discuss different attacks to NDN. Bianchi *et al.* [23] and Detti *et al.* [24] analyze the cost of content integrity verification with the Least Recently Used (LRU) caching strategy. Goergen *et al.* [25] propose a CCN/NDN firewall to filter CCN packets according to content names. Cache pollution attacks and defenses are extensively discussed in [26]–[28]. These schemes are orthogonal to LIVE that is the first scheme that implements distributed content access authorization in the network layer.

The security of clean-slate Internet designs is extensively discussed in the literature [29], [30]. Some features proposed in these designs are similar to LIVE. For example, Koponen *et al.* [30] propose Data-Oriented Network Architecture (DONA) that allows contents to piggyback signatures for the purpose of integrity verification. They only consider integrity verification by users by leveraging traditional public-key cryptography. Arianfar *et al.* [29] leverage cooperation between CPs, and use unrelated contents, called "cover" contents, to mix sensitive contents. Although the approach eliminates the computation overhead incurred by cryptographic operations, it requires CPs to cooperate and store a large volume of "cover" contents, which may introduce heavy communication overheads between CPs. Moreover, users are still required to retrieve some meta data about cover contents before accessing the requested contents.

## VII. Conclusion

In this paper, we propose LIVE, a lightweight integrity verification mechanism for Named Data Networking to enable universal content integrity and authenticity verification. We further leverage LIVE to achieve efficient and scalable content access control, which allows a content provider to enforce flexible security policies on content caching and access. In particular, we incorporate random encryption in the mechanism such that LIVE can prevent or mitigate unauthorized content access. We prototype LIVE in CCNx, and demonstrate its benefits by experimental study, which shows that it introduces acceptable overhead in content access. In future, we will investigate a more efficient token refresh scheme, and leverage group key management schemes to enable token management for sensitive contents and to implement CP authentication during token refreshment.

## References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking named content," *Commun. ACM*, vol. 55, no. 1, pp. 117–124, 2012.

[2] N. Fotiou, G. F. Marias, and G. C. Polyzos, "Access control enforcement delegation for information-centric networking architectures," in *Proc. ACM SIGCOMM Workshop Inf.-Centric Netw.*, 2012, pp. 85–90.

[3] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proc. CRYPTO*, 1987, pp. 369–378.

[4] K. Zhang, "Efficient protocols for signing routing messages," in *Proc. NDSS*, 1998, pp. 1–7.

[5] *The Content Centric Networking (CCNx) Project*. [Online]. Available: http://www.ccnx.org/, accessed 2012.

[6] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun, "ANDaNA: Anonymous named data networking application," in *Proc. NDSS*, 2012. [Online] Available: http://www.internetsociety.org/andana-anonymous-named-data-networking-application

[7] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang, "DoS and DDoS in named data networking," in *Proc. ICCCN*, 2013, pp. 1–7.

[8] A. K. Maji and S. Bagchi, "$\upsilon$-CAPS: A confidentiality and anonymity preserving routing protocol for content-based publish-subscribe networks," in *Security and Privacy in Communication Networks*. Berlin, Germany: Springer-Verlag, 2011.

[9] M. Nabeel, N. Shang, and E. Bertino, "Efficient privacy preserving content based publish subscribe systems," in *Proc. 7th ACM SACMAT*, 2012, pp. 133–144.

[10] H. Chang, S. Jamin, and W. Willinger, "To peer or not to peer: Modeling the evolution of the internet's as-level topology," in *Proc. 25th IEEE INFOCOM*, Apr. 2006, pp. 1–12.

[11] L. Gong, "A secure identity-based capability system," in *Proc. IEEE Symp. Secur. Privacy*, May 1989, pp. 56–63.

[12] J. S. Shapiro, J. M. Smith, and D. J. Farber, "EROS: A fast capability system," in *Proc. 17th ACM SOSP*, 1999, pp. 170–185.

[13] H. Krawczyk, M. Bellare, and R. Canetti. (Feb. 1997). *HMAC: Keyed-Hashing for Message Authentication*. RFC 2014, IETF. [Online]. Available: https://tools.ietf.org/html/rfc2104

[14] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "SCION: Scalability, control, and isolation on next-generation networks," in *Proc. IEEE Symp. Secur. Privacy*, May 2011, pp. 212–227.

[15] G. Huston and R. Bush, "Securing BGP and SIDR," *IETF J.*, vol. 7, no. 1, 2011. [Online]. Available: http://www.internetsociety.org/articles/securing-bgp-and-sidr

[16] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. 1st ACM Conf. CCS*, 1993, pp. 62–73.

[17] *The OpenSSL Project*. [Online]. Available: http://www.openssl.org/

[18] X. Zhang, Z. Zhou, G. Hasker, A. Perrig, and V. Gligor, "Network fault localization with small TCB," in *Proc. 19th IEEE ICNP*, Oct. 2011, pp. 143–154.

[19] A. Afanasyev, P. Mahadevany, I. Moiseenko, E. Uzuny, and L. Zhang, "Interest flooding attack and countermeasures in named data networking," in *Proc. IFIP Netw. Conf*, May 2013, pp. 1–9.

[20] A. Mohaisen, X. Zhang, M. Schuchard, H. Xie, and Y. Kim, "Protecting access privacy of cached contents in information centric networks," in *Proc. 8th ACM ASIACCS*, 2013, pp. 173–178.

[21] G. Acs, M. Conti, P. Gasti, C. Ghali, and G. Tsudik, "Cache privacy in named-data networking," in *Proc. IEEE 33rd ICDCS*, Jul. 2013, pp. 41–51.

[22] T. Lauinger, N. Laoutaris, P. Rodriguez, T. Strufe, E. Biersack, and E. Kirda, "Privacy risks in named data networking: What is the cost of performance?" *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 5, pp. 54–57, Sep. 2012.

[23] G. Bianchi, A. Detti, A. Caponi, and N. Blefari-Melazzi, "Check before storing: What is the performance price of content integrity verification in LRU caching?" *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 3, pp. 59–67, 2013.

[24] A. Detti, A. Caponi, G. Tropea, G. Bianchi, and N. Blefari-Melazzi, "On the interplay among naming, content validity and caching in information centric networks," in *Proc. IEEE GLOBECOM*, Dec. 2013, pp. 2108–2113.

[25] D. Goergen, T. Cholez, J. Francois, and T. Engel, "A semantic firewall for content-centric networking," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2013, pp. 478–484.

[26] L. Deng, Y. Gao, Y. Chen, and A. Kuzmanovic, "Pollution attacks and defenses for internet caching systems," *Comput. Netw.*, vol. 52, no. 5, pp. 935–956, 2008.

[27] M. Conti, P. Gasti, and M. Teoli, "A lightweight mechanism for detection of cache pollution attacks in named data networking," *Comput. Netw.*, vol. 57, no. 16, pp. 3178–3191, 2013.

[28] A. Compagno, M. Conti, P. Gasti, and G. Tsudik, "Poseidon: Mitigating interest flooding DDoS attacks in named data networking," in *Proc. IEEE 38th Conf. LCN*, Oct. 2013, pp. 630–638.

[29] S. Arianfar, T. Koponen, B. Raghavan, and S. Shenker, "On preserving privacy in content-oriented networks," in *Proc. ACM SIGCOMM Workshop Inf.-Centric Netw.*, 2011, pp. 19–24.

[30] T. Koponen *et al.*, "A data-oriented (and beyond) network architecture," in *Proc. SIGCOMM*, 2007, pp. 181–192.

**Qi Li** received the B.Sc. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China, in 2003 and 2012, respectively, where he is currently an Associate Professor with the Graduate School at Shenzhen. He was a Researcher with the Swiss Federal Institute of Technology, Zurich, Switzerland, and a Post-Doctoral Fellow with the Institute for Cyber Security, University of Texas at San Antonio, San Antonio, TX, USA. His research interests include system and network security, Internet, and large-scale distributed systems.

**Xinwen Zhang** received the Ph.D. degree in information security from George Mason University, Fairfax, VA, USA, in 2006. He is currently a Principal Engineer with Samsung Research America, San Jose, CA, USA. His current research interests include security policies, models, architectures, mechanism in general computing and networking systems, secure and trusted network architecture, cloud computing, mobile platforms, and storage systems.

**Qingji Zheng** received the Ph.D. degree in computer science from the University of Texas at San Antonio, San Antonio, TX, USA. He is currently a Research Scientist with the Huawei Research Center, Santa Clara, CA, USA. His research interests lies in the security technologies in cloud computing with applied cryptography.

**Ravi Sandhu** (F'02) is currently the Executive Director of the Institute for Cyber Security with the University of Texas at San Antonio, San Antonio, TX, USA, where he is also the Lutcher Brown Endowed Chair in Cyber Security. He was with the faculty of George Mason University, Fairfax, VA, USA, from 1989 to 2007, and Ohio State University, Columbus, OH, USA, from 1982 to 1989. He received the B.Tech. degree from IIT Bombay, Mumbai, India, the M.Tech. degree from IIT Delhi, New Delhi, India, and the M.S. and Ph.D. degrees from Rutgers University, Piscataway, NJ, USA. He is a fellow of the Association for Computing Machinery (ACM) and the American Association for the Advancement of Science. He has received awards from the IEEE, ACM, NSA, and NIST. As a prolific and highly cited author, his research has been funded by NSF, NSA, NIST, DARPA, AFOSR, ONR, AFRL, and private industry. His seminal papers on role-based access control established it as the dominant form of access control in practical systems. His numerous other models and mechanisms have also had considerable real-world impact. He served as the Editor-in-Chief of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and a founding Editor-in-Chief of the *ACM Transactions on Information and System Security*. He was the Chairman of the ACM Special Interest Group on Security, Audit and Control, and founded the ACM Conference on Computer and Communications Security, the ACM Symposium on Access Control Models and Technologies, and the ACM Conference on Data and Application Security and Privacy. He has served as the General Chair, the Steering Committee Chair, the Program Chair, and a Committee Member for numerous security conferences. He has consulted for leading industry and government organizations, and has lectured all over the world. He is an inventor of 30 security technology patents and has accumulated over 27 000 Google Scholar citations for his papers. At the Institute for Cyber Security, he leads multiple teams conducting research on many aspects of cyber security, including secure information sharing, social computing security, cloud computing security, secure data provenance, and bonnet analysis and detection, in collaboration with researchers all across the world.

**Xiaoming Fu** received the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2000. He was a research staff with Technical University Berlin, Berlin, Germany, until joining the University of Göttingen, Göttingen, Germany, in 2002, where he has been a Professor and the Head of the Computer Networks Group since 2007. His research interests lie in architectures, protocols, and applications for networked systems, including information centric networking, mobile networking, cloud computing, and social networks.