A Cache-Aware Mechanism to Enforce Confidentiality, Trackability and Access Policy Evolution in Content-Centric Networks

Michele Mangili[a,b], Fabio Martignon[a,c,*], Stefano Paraboschi[d]

[a]*LRI, Université Paris-Sud, Bat. 650, rue Noetzlin, 91405 Orsay, France.*
[b]*DEIB, Politecnico di Milano, via Ponzio 34/5, 20133 Milano, Italy.*
[c]*IUF, Institut Universitaire de France*
[d]*Dip. di Ingegneria, Università degli Studi di Bergamo, Via Marconi 5, 24044 Dalmine (BG), Italy.*

Abstract

The Content-Centric Networking (CCN) paradigm introduces a novel communication model where any node in the network can implement caching functionalities to directly serve incoming content requests. However, such a radical change in the protocol stack poses new security challenges since the content producer loses control over the data he provides to the network.

Our contribution is to propose *ConfTrack-CCN*, an efficient encryption-based extension to the CCN proposal, designed to enforce *confidential* data dissemination, *trackable* content access and seamless support of *policy evolution*. *ConfTrack-CCN* jointly enforces all these three requirements by protecting the data with two layers of encryption, the latter of which evolves to reflect access privilege updates. A forced consumer-producer interaction makes consumers fetch keying materials, while sending back logging data on the accessed objects.

To evaluate the traffic reduction that *ConfTrack-CCN* can guarantee, we perform thorough simulation campaigns with real network topologies, and we further study the computational overhead introduced by the encryption primitives we use to secure the communication. The results clearly show that, on average, *ConfTrack-CCN* ensures a 20% higher hit-rate than other security schemes, while introducing a negligible computational overhead.

Keywords: Content-Centric Networks, Security Architectures, Confidentiality, Trackability, Access Policy Evolution.

*Corresponding author, Tel: (+33) 01.69.15.68.16, Fax: (+33) 01.69.15.65.86
Email addresses: michele.mangili@lri.fr (Michele Mangili), fabio.martignon@lri.fr (Fabio Martignon), parabosc@unibg.it (Stefano Paraboschi)

1. Introduction

The objective that drove the development of the Internet when it was originally conceived in the late '60s was resource sharing [1]. In fact, the network was designed to let the scientific community remotely exploit and share the available computational power, which was very expensive at that time [2]. As a result of this need, the infrastructure was used mostly to interconnect two remote machines and make them exchange data by creating an end-to-end communication pipe. The presence in IP packets of the source and destination addresses of the machines involved in the dialog reflects exactly the above-mentioned objective. However, nowadays this end-to-end communication model does not seem to be appropriate anymore. As a matter of fact, in terms of traffic, Internet is currently mostly used as a means to perform *content distribution* of static data: users care about retrieving the content that they are looking for, no matter where it is located and stored [3].

Content-Delivery Networks (CDNs), such as Akamai [4], are a common technology used nowadays to serve the ever increasing bandwidth demands of Internet users [5]. CDNs are built as overlay networks on top of the TCP/IP protocol stack. In this architecture the publisher (i.e., the *origin* server) is assisted by a set of *replica* servers scattered all over the world, to serve clients' demands; clients requesting a given content will be targeted to one of the available replica servers, which will in turn transparently help the origin server to push the content towards the destination.

Such a content-oriented usage of the network is motivating the research community to propose brand new architectures for the Future Internet known under the name of *Content-Centric Networks* (CCNs). Despite that many CCNs have been proposed in the literature [6, 7], they all share the common goal of changing the protocol stack to make the network become an efficient content-distribution infrastructure, without having the need to implement overlay approaches as done in CDNs. For the sake of clarity, in this paper we focus our attention on the proposal known under the name of *Named-Data Networking (NDN)* [8], since it is, to the best of our knowledge, the CCN architecture that has so far received most of the attention from the scientific community[1].

[1]For this reason, throughout the paper we will use the terms *CCN* and *NDN* interchangeably.

The CCN paradigm suggests to provide universal in-network caching as a default feature implemented right inside the network protocol stack [9, 10]. Due to the presence of the distributed caches, any node in the network can potentially serve the requested content without contacting the origin server (i.e., *the publisher*) [11]. However, this feature acts as a double-edged sword since, on one side, it clearly improves the performance of the network, reducing the costs faced by ISPs and content producers to operate the infrastructure, lowering also the response time for users when accessing large resources. On the other hand, it demands for novel mechanisms to enforce *confidentiality*, *content access trackability* and to support *access policy evolution* at the same time.

Enforcing these three requirements in CCN demands for specific solutions. In particular, in order to make the design as simple as possible, the current proposal for CCN poses the following constraints:

- **Content is immutable (C1)**: it is impossible to update the content already published under a given name. If a new version of the content is created, a new name will be attributed to it, thus ensuring cache-coherency.

- **Caches are autonomous (C2)**: each cache can choose the content replacement strategy to adopt, thus making it possible for administrators to further customize the network behavior.

From the security perspective, however, the joint presence of **(C1)** and **(C2)** introduces the challenging issue that once the content has been published in the network, it is impossible for the producer to make sure that such content is completely removed from it. In fact, according to **(C2)** some caches might hold (and serve) the given object for an indefinite period of time, whereas **(C1)** negates the possibility to overwrite the given data. Not to mention, policy evolution clashes with **(C1)**, whereas **(C2)** negates the possibility to trust caching nodes or demand their cooperation, since they could act independently from the content producer's will. Therefore, in current proposals for CCN, a consumer whose access privilege to a popular content has already been revoked might still fetch that data directly from the caches, and, worst of all, without even making the original producer be aware that the access violation occurred.

For all these reasons, in this paper we propose *ConfTrack-CCN*, a novel cache-aware and encryption-based mechanism tailored for Content-Centric Networks, designed to be integrated as an underlying layer of CCN. Data is protected by two layers of symmetric encryption, the latter of which evolves to match access privilege updates, thus ensuring *confidentiality*. Moreover, we force the consumers to retrieve keying material directly from the original producers, an interaction that provides access *trackability* feedback. Lastly, we efficiently enforce *access policy evolution* by making the nodes refresh only a small fraction of the cached objects, a strategy that reduces the operational costs of network operators, since it leads to higher overall hit-rates when compared to those obtained with custom security solutions implemented at the application level.

In summary, our paper provides the following contributions:

- We discuss the problems that arise in a CCN, regarding confidential and trackable access to resources, and propose *ConfTrack-CCN*, a novel security layer that enforces confidential and trackable content dissemination in CCNs in a cache-friendly manner, while providing seamless support to the evolution of access policies.

- We perform a thorough analysis of the security of our proposal, by showing that the mechanism ensures confidentiality when both the policy evolves and the access privilege of a user is revoked. We further provide evidence that the encryption scheme we design can be effectively used to detect and protect against malicious users that cooperate forming a coalition.

- We measure the performance of our proposed architecture by a simulation campaign in real network topologies. Numerical results show that *ConfTrack-CCN* always performs better than user-based encryption schemes, leading to a higher overall hit-rate, while introducing a negligible performance penalty.

- We analyze the cryptographic overhead of our proposal by implementing a Java prototype of *ConfTrack-CCN*. Collected results show that computationally demanding encryption procedures are executed only once on the content producer side, i.e., when a new content is published on the network; a negligible performance overhead is instead required on the consumer side.

The paper is structured as follows: Sec. 2 provides an overview of our proposal and introduces the features of the CCN architecture relevant for our design. Sec. 3 describes in detail *ConfTrack-CCN*, the mechanism we propose to enforce confidential and trackable content dissemination in CCN. Sec. 4 is dedicated to the security analysis of our proposal and provides insights regarding the protection offered in the presence of users' collusion. A thorough performance evaluation is presented in Sec. 5, using a mathematical analysis, simulation results as well as a prototype implementation of the proposed encryption layer in different, realistic network topologies. Sec. 6 discusses related works, and finally, Sec. 7 concludes the manuscript.

2. Overview of the Proposal

We begin this section by providing an overview of our proposal (Sec. 2.1), then we present Named-Data Networking (NDN, Sec. 2.2), the particular instance of Content-Centric Network (CCN) that we consider in our design. In Sec. 2.3 we describe our network model and complementary assumptions, and we finally conclude by illustrating User-Based Encryption (Sec. 2.4).

2.1. Security Challenges and Proposed Solution

The problem that we tackle in this paper is to support *confidential* and *trackable access* to resources distributed using a CCN while fully supporting *policy evolution*. The design principles of Content-Centric Networks leave the responsibility to manage these properties to the applications. However, despite being technically feasible, our analysis shows that an implementation of these features with no intervention on the underlying protocol jeopardizes the correct realization of such security requirements. In fact, as extensively discussed in Sec. 5.1, the overall cache hit-rate we observed while using our proposed solution (*ConfTrack-CCN*) is on average 20% higher than the one obtained with classic security architectures, meaning that our proposal leads to higher network efficiency. To achieve this goal, the technique we propose applies a limited variant to the current CCN protocol, and guarantees a correct enforcement of *confidentiality* and *trackability* that otherwise could discourage content providers to adopt this novel communication paradigm.

We settle our design on top of the solid ground of standard, state of the art, symmetric encryption and hash functions. By doing that, we can furthermore experience all the benefits of leveraging efficient hardware implementations of these primitives that are commonly provided in standard commercial CPUs since the Intel Westmere microarchitecture (launched in January 2010) [12].

The rationale of our proposal starts from the necessity to control the access to resources, which is needed since: (1) only specific users (e.g., paying subscribers) might have acquired the privilege to access given data *(i.e., Confidentiality requirement)*, (2) the content provider wants to keep a precise track of the number and profile of access requests *(i.e., Trackability requirement)*, and (3) it should be possible to add and revoke access privileges to users *(i.e., Policy evolution requirement)*. Indeed, most content providers today, (e.g., YouTube) show great interest in keeping track of every access and do not directly offer to end users the possibility to download a local copy of the data for postponed fruition. However, in current implementations of CCNs, consumers might fetch contents directly from the caches without sending any type of access feedback to the original producer. For this reason, our idea is to let each user access an encrypted version of the data, taking advantage of network caching for the efficient download of the content, but forcing them to retrieve the decryption key directly from the origin server, according to a specified policy.

The access policy may evolve, with the addition and removal of access privileges to the resources. The addition of an access privilege only requires to authorize a new user in retrieving the decryption key. The revocation of a privilege instead requires a more complex solution: the resource is encrypted with a new key that is then made accessible to the set of users who remain authorized after the evolution of the policy, but the correct realization of this strategy requires an adaptation of the CCN infrastructure, which will be reviewed hereafter for the sake of clarity.

2.2. Overview of Content-Centric Networks

Named-Data Networking (NDN) [8] is the Content-Centric Network (CCN) design we consider in this work. The long term objective of the NDN project is to propose an alternative protocol for the “thin-waist” of the hourglass architecture of the network: NDN should implement only the minimal functionalities needed to ensure global connectivity, similarly to what the universal network layer of IP is currently doing in the Internet.

The fundamental feature consists in replacing *host addresses* with *content names*. Each NDN packet does not contain anymore the source and destination IPs, but instead carries the name of the content that has to be retrieved. From the syntactical perspective, NDN proposes a hierarchical structure of names: they are composed of a sequence of variable-length components, each of which is separated by the "/" symbol. At the same time, NDN does not impose any component semantic: the chosen namespace is completely transparent to the network, thus it is up to every application to determine the best namespace conventions according to the type of service that it is going to deliver. Despite that, the NDN proposal suggests to add version and segment numbers by default to every name. As an example of an NDN name, consider: `/organization.com/arch/Work.zip/_v<Timestamp>/_s5`, where `organization.com` is a globally routable name, `arch/Work.zip` is the object name, whereas the version and the segment number are appended to the two final components of the name, respectively.

As depicted in Fig. 1, in NDN there are two types of packets: (1) Interest and (2) Data packets. Demands govern the interactions between nodes in the network, therefore the communication model is *pull*-driven. A node sending *interests* acts as a *consumer*, whereas a node that can provide *data packets* behaves as a *producer*. Routers have a twofold responsibility: (1) they perform packet forwarding and (2) they can also behave as distributed caches.

In NDN, network interfaces are abstracted using the concept of "Faces". A "Face" can either be a physical network interface, but it can also be an application operating on a given node.

Each node in NDN is characterized by three data structures:

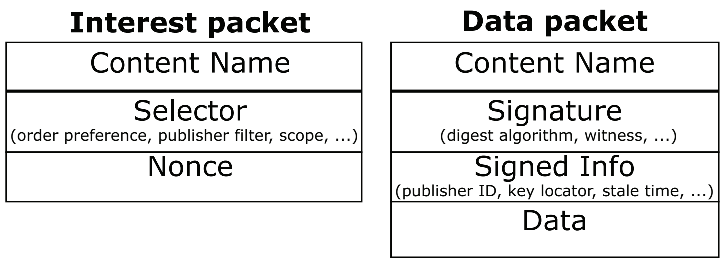

Interest packet
Content Name
Selector
(order preference, publisher filter, scope, ...)
Nonce
Data packet
Content Name
Signature
(digest algorithm, witness, ...)
Signed Info
(publisher ID, key locator, stale time, ...)
Data

Figure 1: Interest and Data packets in NDN.

1. The Pending Interest Table (PIT);
2. The Content Store (CS);
3. The Forwarding Information Base (FIB).

The PIT is responsible for tracking the list of interests previously forwarded, but not yet answered. This table stores the *faces* from which interests were originally received, in order to implement *reverse path forwarding*: as soon as a router receives a data packet, it checks the PIT and forwards the packet on the same faces from which interests for that object arrived. The CS acts as a persistent caching storage for the node, in order to implement universal in-network caching. When an interest arrives, the router will initially query the CS; in the case of a cache hit, the router will be able to directly serve the data. The FIB comes into play when a cache miss happens: it contains the next-hop information for prefix names.

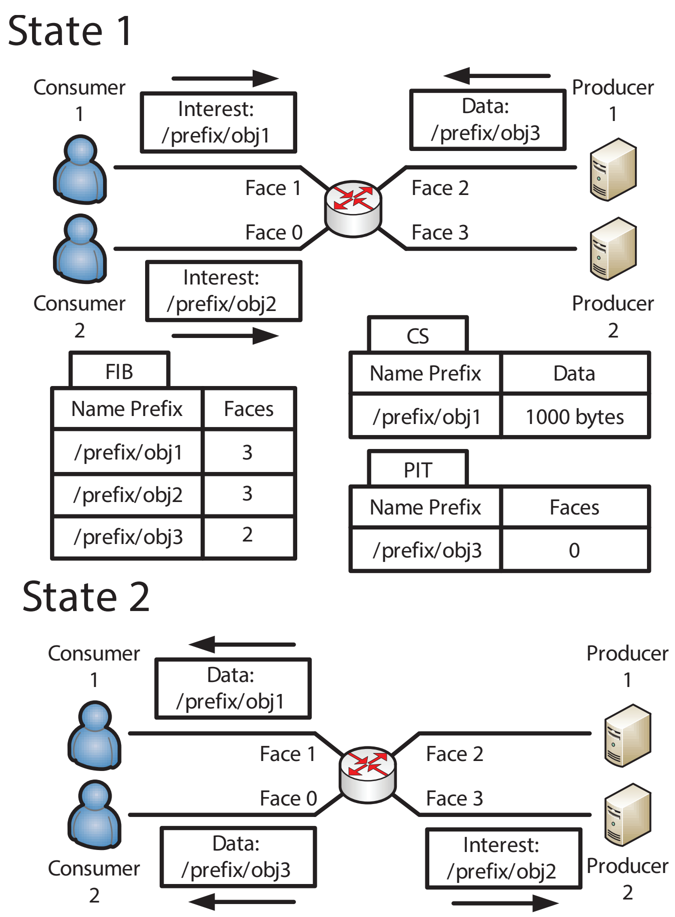
An example showing the behavior of a CCN router is depicted in Fig. 2. In *State 1*, the router receives two interests and one data packet. As shown in Fig. 2, the interest for object `/prefix/obj1` is directly served by the router since it is available in the CS. The interest for `/prefix/obj2` will be forwarded to *Face 3* since it is the destination available in the FIB. Lastly, the data packet for `/prefix/obj3` will be forwarded to *Face 0*, as written in the PIT. In *State 2*, the transition to the next state is represented: as described, the router forwards one interest and two data packets.

In terms of security, due to the fact that interests do not contain the identity of the sender, it is difficult for the provider to assess whether the given request came from a legitimate host or not, thus violating *confidentiality* requirements. On top of that, no access feedback is provided to the producer when data packets are served by intermediate caches, thus violating *access tracking* requirements. For simplicity reasons, CCNs do not provide cache coherency features, which means that they do not natively support *policy evolution* requirement, since the content published in the network is immutable.

However, at the same time, interesting security features are already provided by default in NDN: due to the fact that data packets can potentially be served by any node in the network, they will be signed by the original producer in order to enforce packet integrity and authentication. Creating a binding between the name of a content and the corresponding bits of data permits to place trust in the packets themselves rather than trusting the host from which the data was retrieved. The same requirement does not apply to interest packets: as shown in Fig. 1, they do not possess the “Signature” field. In addition to that, the applications will have to deal with trust and key management issues since NDN provides no native support for their enforcement.

2.3. Network Model and Assumptions

Having revised the NDN paradigm we build upon, we now describe our network model and assumptions. We consider a scenario where the content producer is selling digital



FIB

Name Prefix	Faces
/prefix/obj1	3
/prefix/obj2	3
/prefix/obj3	2

CS

Name Prefix	Data
/prefix/obj1	1000 bytes

PIT

Name Prefix	Faces
/prefix/obj3	0

Figure 2: Example illustrating the behavior of a CCN router. Two interests and one data packet are received by the router in *State 1*. Given the information contained in the *Pending Interests Table* (PIT), the *Content Store* (CS) and the *Forwarding Information Base* (FIB), in *State 2* the router forwards two data packets and one interest.

contents on the network: consumers have to purchase the right to access the data before being able to make use of the corresponding bits. Since many different types of content are nowadays distributed through the Internet, we formulate our proposal as to ensure independence from the particular application considered, meaning that *ConfTrack-CCN* can operate regardless of the particular type of content that is going to be delivered.

Some of the most popular digital marketplaces available today (such as iTunes Store, Google Play, Amazon Store, Netflix, Windows Store) very well exemplify the characteristics of the content distribution service we take into account to design our proposed solution. While these services usually provide many types of content, ranging from movies and e-books, to music and applications, the common feature shared by all of them is that these files are quite large (from few Mbytes, to Gbytes or even more). For this reason, the producer might want to offload the traffic demand on his technological infrastructure, by leveraging the services provided by in-network caching.

A common type of contract that these services offer is the subscription plan: the customer pays for a periodic subscription which gives her/him the right to access a given subset of the content catalog for the whole duration of the purchased period. As soon as the right to access the catalog expires, the user should be denied any further access to the available collection of files. The rising popularity of Netflix provides evidences of the remarkable importance that these particular types of service play nowadays, therefore our design for *ConfTrack-CCN* specifically accommodates this precise need.

Finally, to keep the presentation as clear as possible, we make the assumption that the particular application considered does not have stringent delay requirements. Despite the fact that *ConfTrack-CCN* can also be used for live-streaming (by setting a small size for the content block), in this paper we are mostly interested in evaluating the security properties of our scheme and the network performance impact in terms of the hit-rate that the application is going to experience.

2.4. User-based Encryption Scheme

In this subsection we illustrate the characteristics of user-based encryption and we show that, despite the fact that it provides interesting security properties, it also completely jeopardizes the potential benefits introduced by in-network caching mechanisms.

Let $\mathcal{U}$ be the set of consumers, $U \subseteq \mathcal{U}$ denote a subset of users, while $\mathcal{O}$ is the set of objects published by a content provider. We define a *user-based* encryption scheme as follows: the content provider generates a new encryption key K_o^U, for each content $o \in \mathcal{O}$ that a given subset of users $U \subseteq \mathcal{U}$ is entitled to access. By restricting the cardinality of the subset of users considered (i.e.: $|U|$), we can make this encryption mechanism be very secure, at the price of jeopardizing the in-network caching efficiency. As a matter of fact, if we consider a very small subset of users, ideally $|U| = 1$, many different copies of the same content (encrypted with different keys) will be published in the network. This feature not only makes the cardinality of the real object catalog processed by intermediate routers rise to the very large $|\mathcal{U}| \cdot |\mathcal{O}|$ size, but it also completely disrupts the overall object popularity statistics, severely affecting the efficiency of the caching infrastructure. However, one such scheme has interesting security properties: *confidentiality* is enforced by making the producer disclose the decryption keys only to the authorized users. Instead, *trackability* and *access policy evolution* are jointly supported by the fact that a very low cache hit-rate will be experienced and the producer can easily re-encrypt the content and publish new versions that fully enforce the new access policy.

On the other hand, user-based encryption can also be used as a mechanism that generates a high hit rate, but can only provide weak protection. In particular, one such configuration can be achieved by selecting large $|U|$ values, for instance using only one encryption key per object (i.e.: $U = \mathcal{U}$).

In the next sections we extensively describe our proposed *ConfTrack-CCN* mechanism. In particular, the numerical results in Sec. 5 show that *ConfTrack-CCN* outperforms a user-based encryption scheme in terms of network efficiency, by leading to a much higher overall hit-rate, while correctly implementing the security requirements of *confidentiality*, *trackability* and *access policy evolution*.

3. Confidential and Trackable Content Dissemination Mechanism

We now introduce *ConfTrack-CCN*, the novel mechanism we propose to enforce *confidential and trackable* content dissemination in CCNs, thus representing a fundamental layer of the Content-Centric Network paradigm. Our vision is based on the following principles:

Table 1: Summary of the notation used to illustrate the confidentiality mechanism.

Summary of the Notation	
$\mathcal{B}$	The set of content blocks
$\mathcal{C}$	The set of chunks
n	The number of content blocks, $n = \|\mathcal{B}\|$
m	The number of chunks per content block, $m = \|\mathcal{C}\|$
K_b	The First-Layer encryption key used for content block $b \in \mathcal{B}$
s	The First-Layer encryption key size expressed in bits
$r_i, \forall i \in \{2, ..., m\}$	Sequences of s random bits
$\mathbb{K}$	Keying materials appended to the first chunk. $\mathbb{K} = K_b \oplus r_2 \oplus r_3 \oplus ... \oplus r_m$, where $\oplus$ is the bitwise XOR
$SK(h, l)$	The Second-Layer encryption key
$stp(h, l)$	Producer state of the key regression algorithm
$stc(h, l)$	Consumer state of the key regression algorithm
h	The version of the key
l	The seed used to initialize key regression

- **Encryption enforces confidentiality**. The producer provides encrypted content to the network. Caches store content without the right to access the plaintext.

- **Key management enforces trackability**. Consumers authenticate and directly retrieve decryption keys from the original publisher.

- **Access policy evolution is based on key-derivation and re-encryption**. The evolution of the access policy is enforced by the producer issuing a new version of the content, encrypted with a new key. There exists a key derivation mechanism to let the consumer compute keys for previous versions of a chunk.

In order to enforce the above-mentioned requirements, a producer encrypts the content *twice*, as detailed in the following two subsections: the First Layer of encryption forces consumers to retrieve all content chunks (Sec. 3.1), whereas the Second Layer ensures that only authorized users can retrieve the current version of keying material (Sec. 3.2). In Sec. 3.3 we describe the authenticated key-retrieval protocol used to exchange encryption keys securely while sending content access trackability feedback. Finally, in Sec. 3.4 we show how policy evolution is seamlessly enforced by *ConfTrack-CCN*.

For the sake of clarity, throughout this section we use the notation summarized in Table 1.

3.1. First Layer of Encryption

The purpose of the First Layer of encryption is to force the consumer to retrieve *all* the content blocks before having access to the plaintext version of the requested data. In order to publish the content, the producer will (1) partition and segment the content, (2) encrypt the data and (3) piggyback keying material as depicted in Fig. 3 and further described below.

Content partitioning and segmentation. The original content, whose size is usually quite large[2], is initially partitioned into a set of *content blocks* denoted with $\mathcal{B}$, of cardinality $n = |\mathcal{B}|$. Each content block $b \in \mathcal{B}$ is of fixed size (e.g., 1 Mbyte) and is segmented according to the CCN specification into a set of *chunks* denoted with $\mathcal{C}$, significantly smaller than content blocks (e.g., 1 kbyte each). Each chunk will be transmitted through the network with a CCN data packet, and for this reason a CCN name is associated to each $c \in \mathcal{C}$. We denote with $m = |\mathcal{C}|$ the number of segmented packets per content block, which is computed as a function of the chosen chunk size (e.g., $m = 1000$).

Content encryption. As shown in Fig. 3, the First-Layer encrypted chunks, denoted by *FL-Enc Chunks*, are produced by encrypting the segmented *chunks*. The same *First-Layer encryption key*, K_b, is used to encrypt all the chunks belonging to the same content block $b \in \mathcal{B}$. Such key is randomly generated by the content producer, and is stored in a secure way in the FL-Enc chunks as described hereafter. Well-known symmetric encryption algorithms, such as AES in Cipher-Block Chaining (CBC) mode with PKCS#7 padding, can be used to secure the First Layer of encryption; in addition to that, we assume that the chosen key size is $s = 128$ bits (note that longer keys can be seamlessly used). We would like to point out the fact that only Second Layer encrypted chunks will be transmitted, and eventually cached, into the network.

Piggybacking keying material. As shown in Fig. 4, we force the consumers to retrieve all the encrypted packets in order to gain access to the plaintext version of the content block. We reserve the last s bits of FL-Enc chunks to transfer keying material.

The producer generates $m - 1$ random sequences of s bits $r_2, r_3, ..., r_m$ and then uses classical secret splitting techniques to compute:

[2]We focus our description on data with size of many Mbytes, that well represents multimedia content, like videos.

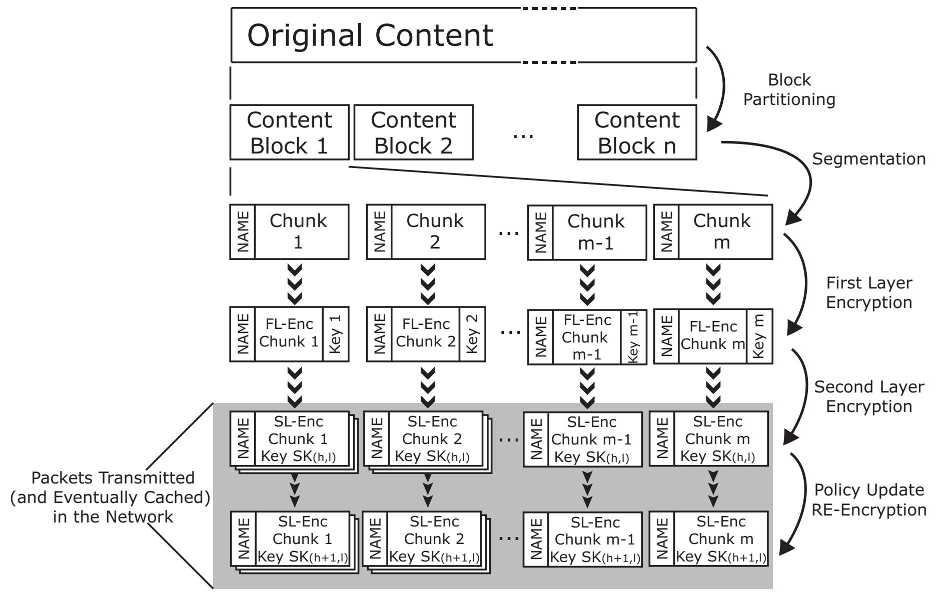

Original Content
Block Partitioning
Content Block 1
Content Block 2
Content Block n
Segmentation
NAME
Chunk 1
Chunk 2
Chunk m-1
Chunk m
First Layer Encryption
FL-Enc Chunk 1
Key 1
FL-Enc Chunk 2
Key 2
FL-Enc Chunk m-1
Key m-1
FL-Enc Chunk m
Key m
Second Layer Encryption
SL-Enc Chunk 1 Key SK(h,l)
SL-Enc Chunk 2 Key SK(h,l)
SL-Enc Chunk m-1 Key SK(h,l)
SL-Enc Chunk m Key SK(h,l)
Policy Update RE-Encryption
SL-Enc Chunk 1 Key SK(h+1,l)
SL-Enc Chunk 2 Key SK(h+1,l)
SL-Enc Chunk m-1 Key SK(h+1,l)
SL-Enc Chunk m Key SK(h+1,l)
Packets Transmitted (and Eventually Cached) in the Network

Figure 3: High-level packet structure defined by *ConfTrack-CCN*. FL-Enc and SL-Enc denote the 1st and 2nd-layer encrypted chunks, respectively.

- $K_b \oplus r_2 \oplus r_3 \oplus ... \oplus r_m = \mathbb{K}$, which is appended to the first chunk;
- $r_2, r_3, ..., r_m$, which are appended to the other chunks.

The joint utilization of CBC and the random values forces the client to retrieve all the encrypted packets in order to gain access to the plaintext. On the other hand, to implement access control and deny access to the entire content block to a given user, we make sure that he will not be able to decrypt at least one SL-Enc chunk, by not disclosing the corresponding Second Layer key. If the user cannot compute at least one FL-Enc chunk in the content block, he will not be able to compute the First Layer key K_b, and therefore, he will not be able to access the plaintext version of the content block.

3.2. Second Layer of Encryption

The purpose of the Second Layer of encryption is to guarantee confidentiality and prevent collusion, while serving as a basis to ensure trackability.

As depicted again in Fig. 3, after having been processed with the First Layer of encryption, chunks are further encrypted using one of the available Second-Layer encryption keys. Such keys are generated using the “Key Regression” derivation algorithm

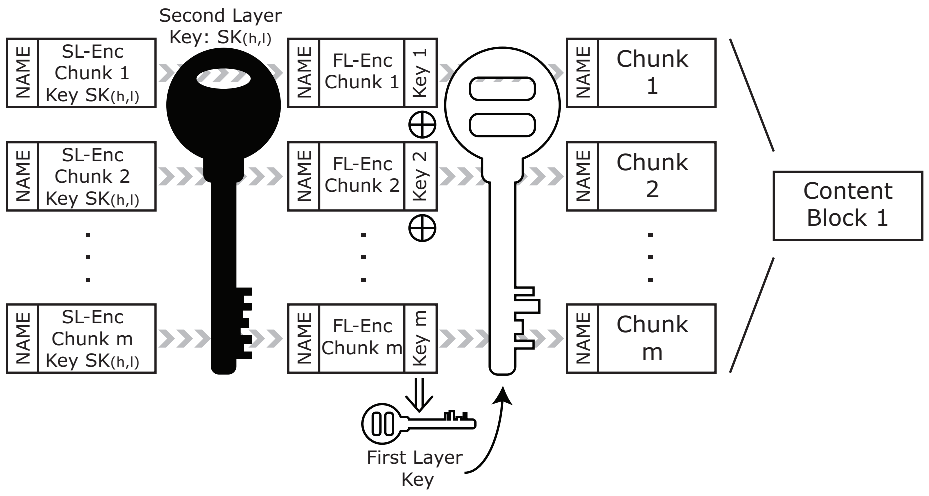


Figure 4: The plaintext can be accessed only if the user can decrypt all the SL-Enc Chunks belonging to the content block, by using the corresponding Second Layer keys. Once that the FL-Enc Chunks are obtained, by computing the XOR of the pseudo-random values piggybacked to the FL-Enc Chunks, the user can compute the First Layer Key, which is then used to decrypt all the chunks belonging to the content block.

in its KR-RSA formulation [13]. We denote each Second-Layer key with $SK(h,l)$, where h is the key version, whereas l represents the original seed used to initialize the KR-RSA algorithm. A given FL-Enc chunk can be encrypted using many Second-Layer keys, each of which is generated using a different seed l.

As illustrated in Fig. 5, four operations (setup, wind key, unwind key, key derivation) define the KR-RSA algorithm. The detailed specification of all these operations is available in Appendix A. The initial *producer state*, $stp(0,l)$, is computed by the producer using the *setup* operation, whereas the *wind key* is used to derive $stp(h+1,l)$ given $stp(h,l)$. The consumer retrieves the consumer state $stc(h,l)$ from the producer, by using the *authenticated key-retrieval protocol* described in Sec. 3.3. Given $stc(h,l)$, the consumer can therefore compute $SK(h,l)$ and all the previous versions $SK(h',l), 0 \leq h' < h$ by using the *unwind key* and the *key derivation* operations. We underline that, compared to well known key derivation algorithms (such as S/KEY [14]), KR-RSA provides two advantages: (1) the *wind key* operation can be executed an unbounded number of times, and (2) it is KR-secure, meaning that if two derived keys are released, a third party cannot discern whether they are linked or not.

Protection against collusion can be provided by means of encrypting the same FL-

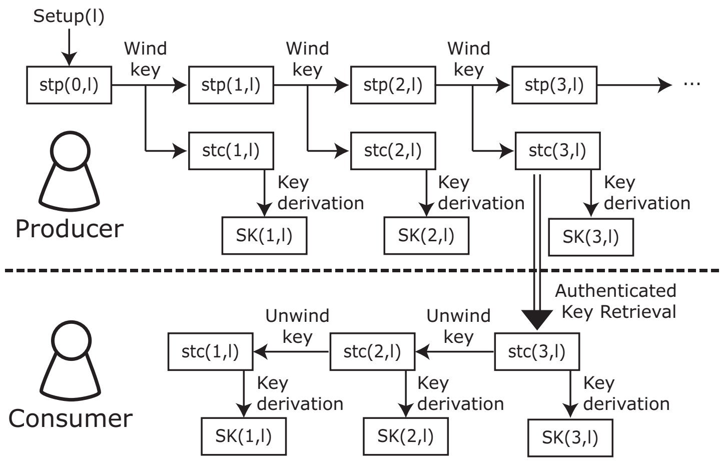


Figure 5: *Key Regression - RSA*: *stp*(h, l) denotes the producer state at version h initialized with seed l, whereas *stc*(h, l) is the consumer state. The *wind key* operation lets the producer compute a new encryption key $SK(h + 1, l)$, whereas the *unwind key* operation is used by the consumer to compute previous key versions.

Enc chunk with many Second-Layer keys generated using different seeds: in this way, by disclosing different key sets to the users, we can detect malicious nodes that disclose their secret decryption keys even in the presence of collusion, as discussed in Sec. 4.2.

3.3. Authenticated Key-Retrieval Protocol

Hereafter, we describe the authenticated key-retrieval protocol used by the consumer to fetch the state $stc(h, l)$ from the producer, while sending content access trackability feedback. The data exchanged during such interaction comprises: (1) the ID of the object that the customer is willing to retrieve, $objID$; (2) the seed used to initialize KR-RSA, l, and (3) the version of the needed key, h.

We assume that the producer is reachable through the globally routable name `/prod.com/`, and that he generated the asymmetric key-pair $KeyPair_p = (Pub(p), Priv(p))$, whereas the consumer publishes content under the name `/cons.com/` and owns the keys $KeyPair_c = (Pub(c), Priv(c))$. We denote with || the concatenation operator, with E_k(data) the encryption of "data" with key k; finally, the hash of "data" is represented by $hash(data)$.

Fig. 6 summarizes the sequence of messages exchanged. MSG_1 is used to communicate to the content producer the namespace `/cons.com/rnd1` under which authentication data was published. MSG_1 is at the same time confidential and authentic, since it is

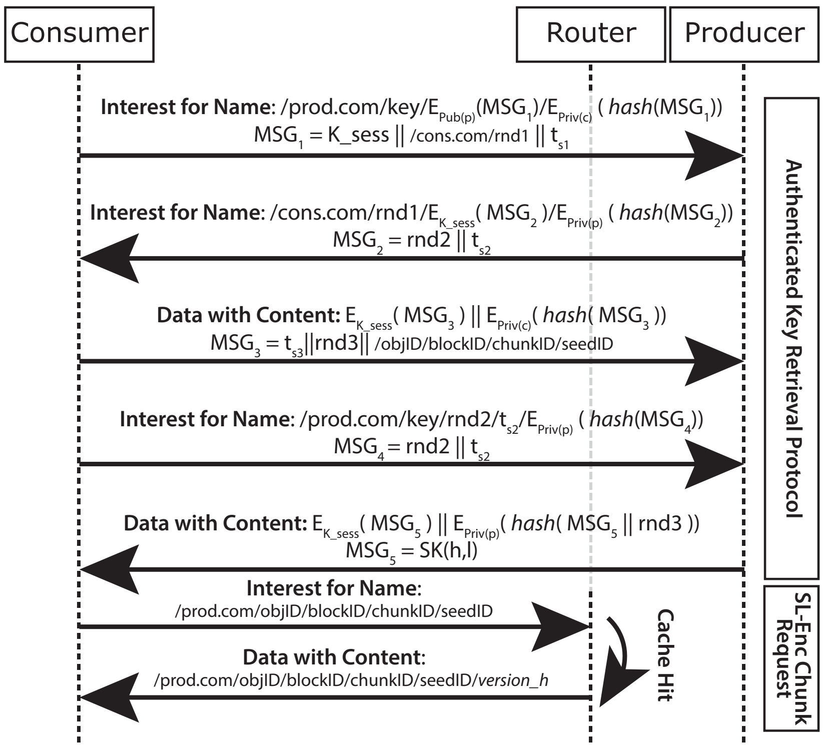

Consumer
Router
Producer
Interest for Name: /prod.com/key/E_Pub(p)(MSG_1)/E_Priv(c) (hash(MSG_1))
MSG_1 = K_sess || /cons.com/rnd1 || t_s1
Interest for Name: /cons.com/rnd1/E_K_sess(MSG_2)/E_Priv(p) (hash(MSG_2))
MSG_2 = rnd2 || t_s2
Data with Content: E_K_sess(MSG_3) || E_Priv(c)(hash(MSG_3))
MSG_3 = t_s3||rnd3|| /objID/blockID/chunkID/seedID
Interest for Name: /prod.com/key/rnd2/t_s2/E_Priv(p) (hash(MSG_4))
MSG_4 = rnd2 || t_s2
Data with Content: E_K_sess(MSG_5) || E_Priv(p)(hash(MSG_5 || rnd3))
MSG_5 = SK(h,l)
Interest for Name:
/prod.com/objID/blockID/chunkID/seedID
Data with Content:
/prod.com/objID/blockID/chunkID/seedID/version_h
Cache Hit
Authenticated Key Retrieval Protocol
SL-Enc Chunk Request

Figure 6: Messages exchanged in the *Authenticated Key-Retrieval Protocol* when the consumer downloads the Second-Layer encryption key, while jointly authenticating and providing access trackability feedback.

encrypted with $Pub(p)$, and signed with $Priv(c)$. MSG_2 is used to retrieve key information and to communicate the random number (rnd2) that will be used by the customer to generate MSG_4 and retrieve the Second-Layer encryption key. MSG_3 contains the basic information `/objID/blockID/chunkID/seedID`, whereas MSG_4 and MSG_5 complete the protocol by permitting to obtain the Second-Layer encryption key, if the consumer is authorized to fetch it.

As shown in Fig. 6, cache hits can only be generated while retrieving SL-Enc chunks. This is due to the presence of timestamps in the naming structure we designed for the authenticated key retrieval protocol. Both the consumer and producer send CCN interest packets during the key retrieval dialog. Since the CCN model is pull driven, this type of dialog is necessary to let one host upload data to a remote server. In particular, the host will publish on the network the content to be uploaded while the server will issue interest packets to retrieve the corresponding data.

The amount of data that nodes exchange in order to transmit keying materials is practically negligible: less than 5 kbytes must be transmitted for each content block. The overhead introduced in the *Authenticated Key-Retrieval Protocol* for a content block size of 1 Mbyte (as considered in our numerical results) is therefore less than 1%.

3.4. Policy Evolution

ConfTrack-CCN lets the producer make the access policy evolve by revoking access to the customers not entitled anymore to make use of given resources. In order to enforce policy evolution, the producer: (1) computes new encryption keys $SK(h+1,l)$; (2) re-encrypts and publishes the new version of the data; (3) does not serve anymore the obsolete content and (4) does not disclose the new $stc(h+1,l)$ to the customers whose access privilege was revoked. More specifically, the rationale behind this approach is to make sure that some chunks of a *content block* will be served to the revoked user encrypted with the $SK(h+1,l)$ key, as enforced by (1)-(3). In this way, the user cannot obtain the corresponding decryption key $stc(h+1,l)$, as enforced by (4), and consequently he will not be able to decrypt the corresponding chunk. Finally, even though the user can retrieve all the chunks in one content block, he will still not be able to decrypt the whole content block.

Hereafter, we describe the policy evolution mechanism we designed, in order to make the Content-Centric Network quickly refresh contents as to ensure enforcement of confidentiality properties.

Even if caches can still provide obsolete content as a result of an incoming interest, the cache evolution mechanism we design is tailored to support such evolution in an efficient manner. To achieve this goal, we propose an innovative caching policy mechanism that, at regular intervals, forces the cache to forward an upstream interest for a “fresh” copy of a single chunk randomly chosen in a given content block. We implement such behavior by making the router mark cached content as *stale*, and forcing it to evict the data from the cache. Our mechanism and the joint presence of the two layers of encryption ensure that the network will propagate policy evolution changes very quickly, minimizing the amount of data that nodes exchange. It is important to note that our proposal is backward-compatible: caches not willing to utilize our proposal will work as usual, but they will experience higher operational costs due to lower hit rates.

We suggest to utilize the naming scheme shown hereafter: `/prod.com/objID/blockID/chunkID/seedID/`h. The consumer issues interests without specifying h, the version of the Second-Layer encryption key: by issuing an interest for `/prod.com/objID/blockID/chunkID/seedID/` the consumer is demanding any version of the given content chunk, a condition that ensures higher hit rates.

When receiving an interest packet, caches execute Alg. 1.

Algorithm 1: Cache Update Algorithm

```
Input  : prod.com, objID, blockID, chunkID, seedID
Output: DataPkt
1 if IsCached(prod.com, objID, blockID, chunkID, seedID) then
2     $t \Leftarrow$ GetTimeout(prod.com, objID, blockID);
3     if IsExpired($t$) then
4         $c \Leftarrow$ ChooseCachedRandomChunk(prod.com, objID, blockID);
5         MarkChunkAsStale($c$);
6         UpdateTimeout(prod.com, objID, blockID);
      end
7     DataPkt $\Leftarrow$ GetDataPkt(prod.com, objID, blockID, chunkID);
  else
8     ForwardInterestUpstream(prod.com, objID, blockID, chunkID);
  end
```

If the interest generates a cache-hit, the "freshness" timeout for `/prod.com/objID/blockID/chunkID` is checked (Steps 1-3). If such timeout is expired, the cache will select a random chunk already cached by the node and belonging to the same content block; such chunk will be refreshed by making the router mark the content as stale (Steps 4-5), while the timeout will be updated (Step 6). Finally, the data packet will then be provided to the customer, by reading it from the content store in case of a cache hit (Step 7), or by forwarding the interest upstream (Step 8).

4. Security Analysis

In this section we analyze *ConfTrack-CCN* by studying the security properties of the proposed mechanism. In particular, in Sec. 4.1 we prove that an adversary whose access privilege has been revoked cannot access the plaintext version of the content, if at least one SL-Enc chunk has been refreshed. In Sec. 4.2 we instead investigate the problem of user collusion, and show that ConfTrack-CCN provides strong security properties even in the presence of a coalition of adversaries.

4.1. Security Proof for the Proposed Cache Management Policy

In this section we prove that, as a result of an access policy evolution for a given content, our proposed scheme revokes access to a user who is not authorized to obtain the given content anymore. The rationale behind such property is to make sure that the user cannot decrypt *at least one* SL-Enc chunk of a given content block b. If this is the case, our mechanism guarantees that the user cannot access the plaintext of the *entire* content block, since he cannot obtain the First Layer decryption key. Hereafter we consider the scenario where the access privilege of an adversary $\mathbb{A}$ has been revoked and the caching policy mechanism proposed in Sec. 3.4 has refreshed at least one SL-Enc chunk of the block b.

Theorem 1. *The first encryption layer forces the users to retrieve all the FL-Enc chunks belonging to the same content block b to gain access to the plaintext version of the data.*

Proof. In order to prove the security of *ConfTrack-CCN*, we take into account the most adverse case where $\mathbb{A}$ could retrieve m SL-Enc chunks, but he was able to decrypt only $m-1$ of them; hence, $\mathbb{A}$ can access all the FL-Enc chunks of a block apart from the j-th. Since the FL-Enc chunks contain

$$\begin{cases} r_j & \text{if } j \neq 1 \\ K_b \oplus r_2 \oplus r_3 \oplus ... \oplus r_m = \mathbb{K} & \text{if } j = 1, \end{cases} \tag{1}$$

both cases must be considered in the proof.

Case $j \neq 1$: the adversary can access $\mathbb{K}$ and all the random values apart from the j-th (r_i, $\forall i : i \neq j$). Thus, he can compute $\mathbb{K} \oplus r_2 \oplus r_3 \oplus ... \oplus r_{j-1} \oplus r_{j+1} \oplus ... \oplus r_m = K_b \oplus r_j$. This cyphertext is the one obtained using the Vernam cipher, an information-theoretically secure encryption algorithm that provides perfect secrecy under the assumption that the sequence $r_2, r_3, ..., r_m$ is truly random and will be used only once. Therefore $\mathbb{A}$ cannot compute the decryption key K_b.

Case $j = 1$: $\mathbb{A}$ can access all the FL-Enc chunks, apart from the first. For this reason, he doesn't know the value of $\mathbb{K}$, but he can extract the sequence of random values $r_2, r_3, ..., r_m$. However, these random values are uncorrelated with the encryption key K_b, which cannot be computed by the adversary. $\square$

As a result of the access policy evolution, *ConfTrack-CCN* can prevent unauthorized accesses to the resources distributed in the network. Our mechanism only necessitates to make sure that at least one SL-Enc chunk cannot be decrypted by the revoked user. In this way, as Theorem 1 shows, access to the *entire* content block is denied.

4.2. Detecting Collusion Attacks

As we will show in Sec. 5, *ConfTrack-CCN* uses shared encryption keys to increase the overall hit-rate that the network can achieve while reducing also the computational overhead of the solution. Hereafter, we investigate the problem of user collusion aiming at disclosing the Second Layer decryption keys to let unauthorized users decrypt the content that the provider has published in the network. We will show that, by distributing different key sets to the users, *ConfTrack-CCN* can also provide *collusion prevention* and *detection* functionalities.

We do not take into account the scenario where an adversary $\mathbb{A}$ provides the plaintext version of the content, because, due to the large data size, we assume that it is cost ineffective for him to serve the plaintext. On the other hand, it is instead feasible for $\mathbb{A}$ to distribute his own set of decryption keys, due to their modest size.

The content provider may want to distribute the content using only one set of Second Layer keys, shared among all the users. In this case the overall hit-rate would be very high, since only one copy of the content is going to be distributed in the network, but the security of the mechanism would be very low since an adversary $\mathbb{A}$ can easily disclose the set of keys without revealing his own identity. On the other hand, the provider may want to use different sets of Second Layer encryption keys, making sure that each user has his own unique set. In this case, if an adversary discloses his keys, the content provider can easily detect his identity and therefore he will be able to punish the user for this unauthorized behavior. In this subsection, we consider a scenario in between these two extremes: the content provider assigns different key sets to the users; therefore, when

Table 2: Summary of the notation used for the Collusion Analysis.

Parameters of the model	
$\mathcal{M}$	Set of malicious users
$\mathcal{N}$	Set of non-malicious users
$\mathcal{U}$	Set of all users ($\mathcal{U} = \mathcal{M} \cup \mathcal{N}$)
$\mathcal{B}$	Set of content blocks
$\mathcal{K}$	Set of encryption keys
$y_{u,b,k}$	Key assignment matrix. $y_{u,b,k} = 1$ if user $u \in \mathcal{U}$ is assigned for block $b \in \mathcal{B}$ the key $k \in \mathcal{K}$; otherwise, $y_{u,b,k} = 0$.

Decision Variables of the Model	
$x_{u,b,k}$	0-1 Variable that indicates if the identity of user $u \in \mathcal{U}$ is disclosed for block $b \in \mathcal{B}$ with respect to the encryption key $k \in \mathcal{K}$
d^M	Identity disclosure for malicious users
d^N	Identity disclosure for non-malicious users

performing a key disclosure, the user is implicitly disclosing information on his own identity. In our model, the content provider should be able to identify the misbehaving user, given the disclosed keys, even in the challenging scenario where users collaborate and form a coalition.

With the ILP model presented hereafter we compute the best key disclosure strategy that the coalition of adversaries can choose. Their objective is to minimize the maximum identity disclosure of the users in the coalition, in order to make it difficult for the content provider to identify them as guilty. The key assignment strategy implemented by the content provider to allocate the Second Layer decryption keys has an important impact on the overall collusion-resistance properties of the proposed system. To perform one such analysis, we take into account the adverse scenario where the provider randomly allocates the Second Layer keys to its users. As reported in Fig. 7, even in this unfavorable case, *ConfTrack-CCN* exhibits strong security properties, detecting users' collusion with very high probability.

For the sake of clarity, Table 2 summarizes the notation used in this section. Let $\mathcal{M}$ be the set of malicious users, $\mathcal{N}$ the set of non-malicious users, while $\mathcal{U} = \mathcal{M} \cup \mathcal{N}$ is the set of all the user we consider in our analysis. We denote with $\mathcal{B}$ the set of content blocks, while $\mathcal{K}$ is the set of encryption keys.

We consider the most adverse case where malicious users collude. We make the realistic assumption that colluding members do not know which keys have been assigned to non-malicious users; hence, their best key disclosure strategy is to minimize the maximum identity exposure of every member in the coalition by publishing a *combination* of the decryption keys possessed by them.

We denote with $y_{u,b,k}$ the key assignment matrix: $y_{u,b,k} = 1$ if key k is given to user u for content b, while $y_{u,b,k} = 0$ otherwise. $x_{u,b,k}$ is a binary decision variable we use to model both the set of disclosed keys and user identities. In particular, if key $k \in \mathcal{K}$ for content $b \in \mathcal{B}$ is disclosed by some malicious user, all the users $u \in \mathcal{U}$ for which $y_{u,b,k} = 1$ will implicitly disclose their identities (and thus, their $x_{u,b,k}$ variable will be set to 1). In other terms, while disclosing a key for a given block, all the other users to whom that key was assigned will implicitly disclose their identities at the same time.

We denote with d^M the maximum number of times the identity of a *malicious user* is disclosed by the mechanism. On the other hand, d^N represents the maximum number of times a *non-malicious user* is deemed to be guilty. We say that the mechanism is collusion-resistant if the content provider can identify at least one colluding user as guilty of disclosing his secret decryption keys. In other words, the maximum number of times a malicious user is thought to be guilty by the content provider should be greater than the number of times the content provider thinks another user is responsible for disclosing the key sequence. Therefore, if $d^M > d^N$, we say that the mechanism is collusion resistant.

The optimal key disclosure strategy for $\mathbb{A}$ is formulated as an ILP model as follows:

$$\min d^M \tag{2}$$

subject to:

$$\sum_{\forall p \in \mathcal{U}} x_{p,b,k} \leq (x_{u,b,k} + 1 - y_{u,b,k})\,|\mathcal{U}| \qquad \forall (u,b,k) \in \mathcal{U} \times \mathcal{B} \times \mathcal{K} \tag{3}$$

$$\sum_{\substack{\forall m \in \mathcal{M} \\ \forall k \in \mathcal{K}}} x_{m,b,k} \geq 1 \qquad \forall b \in \mathcal{B} \tag{4}$$

$$x_{u,b,k} \leq y_{u,b,k} \qquad \forall (u,b,k) \in \mathcal{U} \times \mathcal{B} \times \mathcal{K} \quad (5)$$

$$\sum_{\substack{\forall b \in \mathcal{B} \\ \forall k \in \mathcal{K}}} x_{m,b,k} \leq d^M \qquad \forall m \in \mathcal{M} \quad (6)$$

$$x_{u,b,k} \in \{0,1\} \qquad \forall (u,b,k) \in \mathcal{U} \times \mathcal{B} \times \mathcal{K}. \quad (7)$$

The objective function (2) finds the best key disclosure strategy that malicious users can adopt. Since they do not know which encryption keys are possessed by non-malicious users, they will therefore minimize their maximum identity disclosure. This explains why d^N does not appear in the objective function.

The set of constraints (3) ensures that if a user in the coalition discloses his key, all the other users possessing the same key will disclose their identities as well.

In (4), we ensure that at least one key will be disclosed for each block, whereas the set of coherence constraints (5) forces a user to disclose only the keys he possesses.

Constraints (6) force d^M to represent the maximum identity exposure of a user. We add the binary constraints on the disclosure variable in (7).

Lastly we compute the maximum number of times the identity of non-malicious users is disclosed, d^N, as follows:

$$d^N = \max_{\forall n \in N} \left(\sum_{\substack{\forall b \in \mathcal{B} \\ \forall k \in \mathcal{K}}} x_{n,b,k} \right). \quad (8)$$

To evaluate the collusion resistance property of *ConfTrack-CCN*, we consider different scenarios with 20 or 25 encryption keys and up to 2000 blocks. We set the total number of users equal to 50, we uniformly generate the key assignment matrix $y_{u,b,k}$, and we perform the analysis with up to 80% colluding members. For each of these scenarios we consider 100 random collusion sets, then, using the model (3)-(7) and equation (8), we compute the *detection probability.* This latter is defined as the ratio between the number of scenarios where the content provider identifies a guilty user in the coalition and the total number of scenarios considered. Fig. 7 shows the detection probability as a function of the size of the colluding users set.

As expected, since encryption keys are uniformly allocated to the users, increasing the

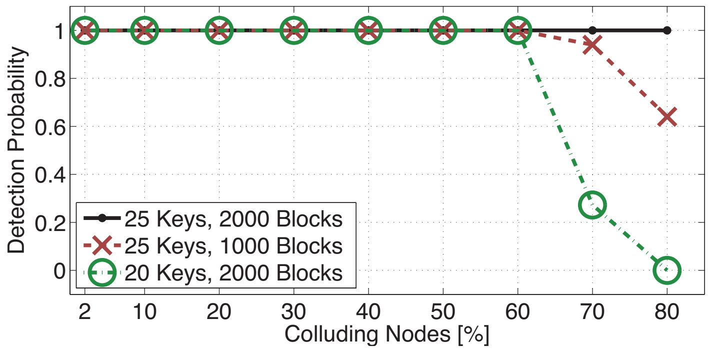


Figure 7: *Disclosure Detection Analysis*: detection probability as a function of the size of the collusion set, for 20 or 25 encryption keys and up to 2000 blocks. The total number of users is set to 50.

number of keys and/or chunks is beneficial in terms of resilience against users' collusion. In particular, 25 keys and 2000 blocks are sufficient to detect a coalition containing up to 80% of colluding users, while if we decrease the number of keys to 20, we can detect misbehaving consumers with very high probability even when 60% of the users are colluding.

In summary, our proposed mechanism proves to be very robust against collusion, even in adverse case when the content provider randomly distributes the keys to the users, and several of them are colluding.

5. Numerical Results

In this section we provide extensive numerical evaluations of our proposal through simulations as well as using the Java prototype we built. More specifically, Sec. 5.1 quantifies the *bandwidth benefits* in terms of cache hit-rate that our proposal can provide with respect to the alternative scenario where a *user-based* encryption model is used. In Sec. 5.2 we study the security properties of our solution, whereas Sec. 5.3 provides computational performance measurements on the prototype of *ConfTrack-CCN*.

5.1. Cache Hit Analysis

We first quantify the bandwidth benefits that our proposal can guarantee using an analytic, as well as a simulated model. We abstract the analysis with a hierarchical cache, an assumption that well represents a scenario where a producer is publishing content and

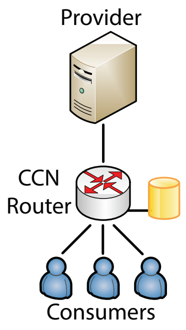


Figure 8: Single-level cache network topology used for numerical results.

consumers are requesting it through a network of caches [15]. The stochastic process of content requests expressed by consumers is filtered by each cache as if they were low-pass filters. Independently from the cache replacement policy chosen, the output process of a cache does not contain requests for frequently demanded contents, since they generate cache hits, and as a consequence they are not forwarded upstream.

We denote with $\mathcal{U}$ the set of consumers, whereas $\mathcal{O}$ denotes the set of objects. For the sake of simplicity, we assume that objects have the same size Θ; however, our results can be easily extended to the general case where objects have a variable size. Each consumer $u \in \mathcal{U}$ generates requests modeled as a Poisson process with aggregate rate λ_u, proportional to the object popularity p_o, where $o \in \mathcal{O}$.

We assume that the popularity distribution of objects is subject to the *Independent Reference Model* (IRM), since this leads to very realistic results when the requests are generated by a large population of users, as shown in [16]. We adopt the *Zipf* discrete distribution, which is frequently used in the literature to model the objects popularity in Internet [15]. Such model describes the popularity rank of each object $o \in \mathcal{O}$ by giving higher priority to objects with lower indices. Let α be the *Zipf* exponent of the distribution; the Zipf probability mass function for the j-th most popular object is defined as $p_j = P(X = j) = \frac{1/j^{\alpha}}{\sum_{i=1}^{|\mathcal{O}|}(1/i^{\alpha})}$.

We take into account the *least recently used* (LRU) cache replacement policy since, to our knowledge, it is the only one for which a realistic analytic model for cache hits has been proposed [17]. Under these assumptions, our performance analysis is based on the model proposed in [17], as it was shown in [16] that this is an extremely accurate

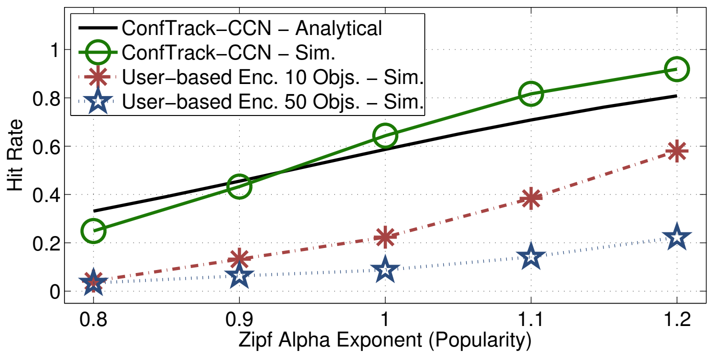


Figure 9: Hit-rate for a single-level LRU cache network with 10^8 objects of 1 Mbyte each and 100 Mbyte of storage, as a function of the Zipf popularity exponent α.

model for hierarchical caches. According to such model, the hit rate for object $j \in \mathcal{O}$ is given by $h(j) = 1 - e^{-p_j t_C}$, where t_C is the root of $\sum_{j=1}^{|\mathcal{O}|} (1 - e^{-p_j t_C}) = M$, M being the cache size, expressed in number of objects.

We compute the hit rate for a single-level cache network (shown in Fig. 8) for $\alpha \in [0.8; 1.2]$, because this range of popularity exponents well represents heterogeneous types of contents, as discussed in [15]. Fig. 9 illustrates the analytic solution of the model, as well as simulation results obtained using the ndnSIM network simulator [18] for the single-level cache network with 10^8 objects of 1 Mbyte each, given a 100 Mbyte cache. In particular, we consider the following cases:

1. *ConfTrack-CCN* is used, and 0.1% of each object is encrypted with 5 different keys;
2. User-based encryption is used (as described in Sec. 2.4), and each object is accessed by 10 users;
3. User-based encryption is used, and each object is accessed by 50 users.

As shown in Fig. 9, the hit rate obtained by *ConfTrack-CCN* is more than 20% higher than the one provided when user-based encryption is used, even when few users (i.e., 10) are accessing the object set.

We further investigated the hit rate considering the Abilene network topology (depicted in Fig. 10) [15], with 10^8 objects of 10 Mbytes each, a Zipf exponent $\alpha = 1.2$, a caching storage of 1 Gbyte per node, and varying the following parameters:

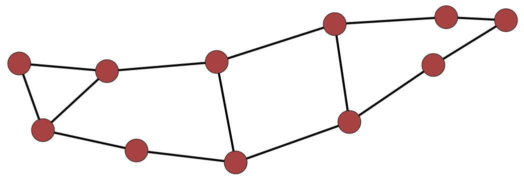


Figure 10: Abilene topology used for numerical results.

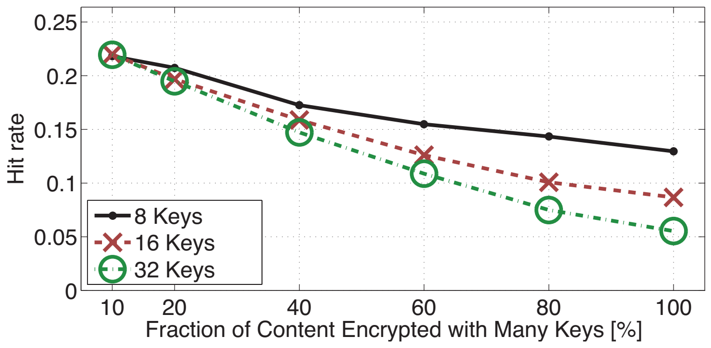


Figure 11: Hit-rate measured in the Abilene topology as a function of the percentage of contents encrypted with many keys.

1. The number of encryption keys used;
2. The fraction of objects encrypted with many keys;
3. The policy evolution delay.

As illustrated in Fig. 11, a higher hit rate is obtained by reducing the number of Second-Layer encryption keys used, as well as the fraction of objects encrypted with many of these keys. This result is expected, since there is a clear trade-off between the performance of the network and the security of the mechanism: if we increase the number of encryption keys, we increase also the possibility to detect colluding users.

Fig. 12 plots the hit rate as a function of the policy evolution delay. When we impose very frequent policy updates (e.g., once every 0.1 seconds), the network hit rate drops below 5%, whereas if the content is updated every 10 seconds or above (a more realistic assumption), we instead observe that a hit rate higher than 25% is obtained when 8 keys are used.

Similar results, omitted here due to space constraints, have been observed also in other network topologies, including the Géant network [19].

5.2. Security Analysis

In this section we numerically study the security properties enforced by our proposed *ConfTrack-CCN* solution, comparing it with a scenario where no other security mechanism is implemented in the network.

We make the realistic assumption that different traffic classes are handled by the network; in particular, we consider a traffic mix as in [20] where Video on Demand (VoD) and User-Generated Content (UGC) are delivered. As detailed in [20], we assume that UGC traffic is characterized by a content catalog of 10^8 objects whose average size is 10 MB, the Zipf exponent is $\alpha = 0.8$ and accounts for 38% of the overall amount of requests. On the other hand, VoD traffic is characterized by a catalog of 10^4 objects of 100 MB each, $\alpha = 1.2$, and accounts for the remaining 62% of the requests. We simulate such scenario in the Abilene topology, as described in the previous section, and study the security properties of our proposed mechanism where CCN routers implement caching functionalities using either the LRU or a random cache replacement policy. We assume that the content provider is offering a VoD service, and it wants to securely distribute the content into the network.

In order to provide evidences that *ConfTrack-CCN* successfully satisfies the security requirements, we measure security violations, as defined hereafter. In particular, a *"trackability violation"* arises whenever a consumer can acquire a given content without making the provider be aware that he did so. We assume that users do not disclose their decryption keys, therefore *"confidentiality violations"* happen only in the case of *access*

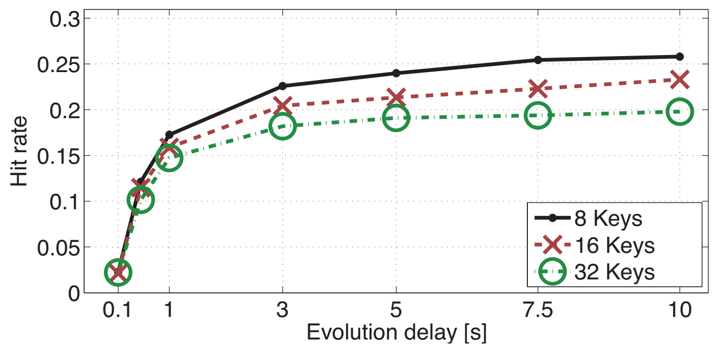
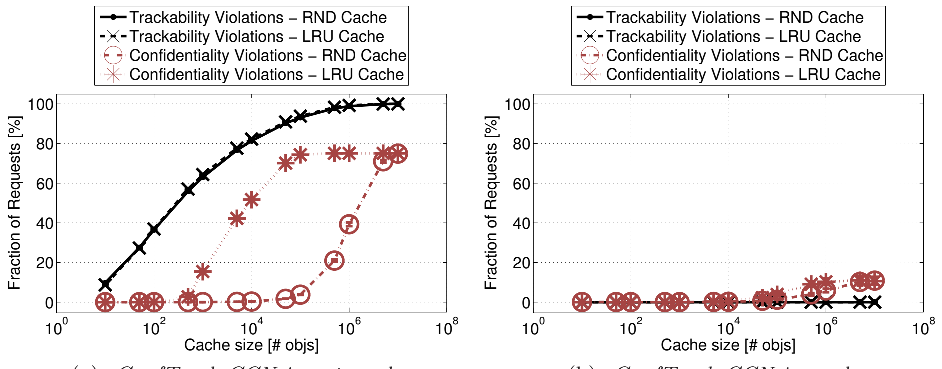


Figure 12: Hit-rate measured in the Abilene topology as a function of the policy evolution delay.

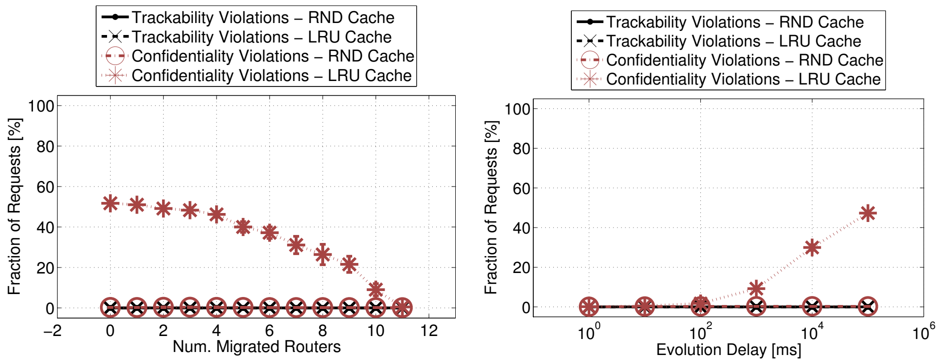


(a) *ConfTrack-CCN is not-used* (b) *ConfTrack-CCN is used*

Figure 13: *Security of ConfTrack-CCN - Effect of cache size.* Fig. 13a shows the trend of trackability and confidentiality violations as a function of the cache size, when ConfTrack-CCN is *not* utilized. Fig. 13b shows the trend for the same security properties, when ConfTrack-CCN is used and all network routers implement our proposed policy evolution mechanism with a policy evolution delay of 1 second.

policy evolution. For this reason, to simulate a scenario where the access policy changes, we assume that the producer removes the 100 most popular resources he is publishing, and we measure the number of successful requests the consumers can issue on the subset of these deleted contents. For each of the analysis we performed 20 different runs, and figures portray the very narrow 95% confidence intervals. Unless stated otherwise, we consider a cache size of 10^4 objects, all the routers implement our proposed policy evolution mechanism, and a policy evolution delay of 1 second is used.

In Fig. 13a we show the security properties of a network that does not implement any protection mechanism, as a function of the cache size. We observe that, in terms of *trackability violations*, random caching and LRU show practically the same performance; in particular, even considering small cache sizes (in the order of 10^3 objects), more than 60% of requests are served without making the producer be aware that users accessed the content. This behavior is caused by the high efficiency of the caching mechanism, perfectly in line with the results observed in [20]. On the other hand, as shown in Fig. 13a, random cache replacement is to be preferred in terms of confidentiality violations. In particular, LRU caches tend to persist popular contents even if the provider has deleted them from its catalog. Fig. 13b reports instead the performance observed in the same scenario simulated for Fig. 13a, but when *ConfTrack-CCN* is used, and all the routers implement our proposed cache update mechanism. In this case, trackability violations do

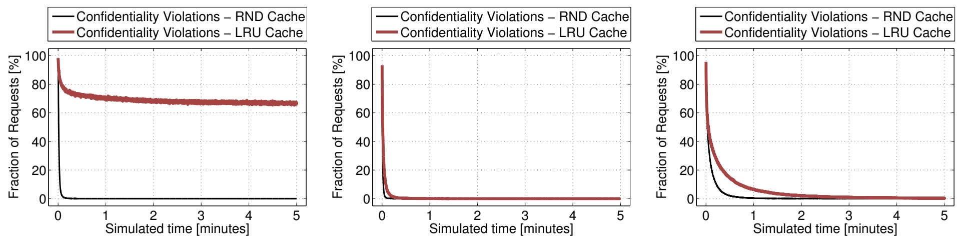


(a) *Effect of the Number of Migrated Routers* (b) *Effect of the Cache Evolution Delay*

Figure 14: *Security of ConfTrack-CCN - Effect of Number of Migrated Routers and Cache Evolution Delay.* Both Fig. 14a and 14b show the security properties of a network where ConfTrack-CCN is used. In Fig. 14a we portray the effect of the number of routers implementing our novel cache eviction policy, while in Fig. 14b we show the effect of the evolution delay parameter for the cache eviction policy.

not happen anymore, whereas a small number of confidentiality violations ($< 18\%$) are observed when caches are very large and can store up to 10^7 objects ($^1/_{10}$ of the entire object catalog). For the sake of conciseness, we are not reporting here similar results for the User-based encryption since, as thoroughly described in the previous section, *ConfTrack-CCN* outperforms user-based encryption in terms of the overall network hit-rate.

The effect of the number of routers implementing our proposed cache update policy, and the effect of the evolution delay parameter (i.e.: the timeout of Alg. 1, Sec. 3.4) are reported in Fig. 14a and 14b, respectively. Since *ConfTrack-CCN* is used, no trackability violations are observed, moreover the number of confidentiality violations when using a random cache replacement policy is negligible regardless of both these parameters. LRU, instead, is sensitive to both the number of routers implementing our policy and the evolution delay. In particular, as Fig. 14a shows, by migrating many routers to our proposed cache update policy, we can significantly reduce the number of observed confidentiality violations, dropping from 55% of confidentiality violations (when no router implements our policy), down to almost 0% (when all the routers implement it). On top of that, as observed from Fig. 14b, the cache evolution delay has a remarkable effect on the observed number of confidentiality violations, and, as expected, the lower the timeout, the more secure the network is.



(a) *ConfTrack-CCN not-used,* 10^4 *objs. cached* (b) *ConfTrack-CCN used,* 10^4 *objs. cached* (c) *ConfTrack-CCN used,* 10^5 *objs. cached*

Figure 15: *Temporal Evolution of Confidentiality Violations.* We portray the number of confidentiality violations in time, for different types of cache eviction policies. In Fig. 15a we show the behavior of a standard CCN network that does not use ConfTrack-CCN, while in Fig. 15b we show the effect of using our solution. Lastly, in Fig. 15c we consider a cache of one order of magnitude larger, while still using ConfTrack-CCN.

By adopting our *ConfTrack-CCN* mechanism, trackability violations do not happen anymore, moreover the number of confidentiality violations is significantly reduced. Hereafter, we study the transient behavior of the network in the interval immediately following the instant when contents are removed from the producers, as a result of an access policy evolution. In particular, since our proposed cache update algorithm does not require nodes' cooperation, a small delay is introduced to effectively revoke the access to the deleted contents, as shown in Fig. 15.

Fig. 15a and 15b show the advantages of using *ConfTrack-CCN* and our proposed cache update mechanism. The trends confirm previous observations on the LRU cache replacement policy: in terms of *confidentiality violations*, one such replacement policy does not provide adequate protection against confidentiality violations. However, by adopting *ConfTrack-CCN* as well as our cache update mechanism, we can make the network quickly remove stale content making the number of confidentiality violations drop to almost 0% in less than 1 minute. Lastly, in Fig. 15c we study the effect of the cache size in a scenario where *ConfTrack-CCN* is still used. In particular, even by increasing the caching storage of one order of magnitude, the number of confidentiality violations is still kept under control, and quickly reaches 0% in less than 3 minutes.

5.3. Encryption Performance Evaluation of the Prototype

As described in Sec. 3, *ConfTrack-CCN* requires the end-hosts to implement cryptographic primitives, in particular the producers encrypt the content, whereas the con-

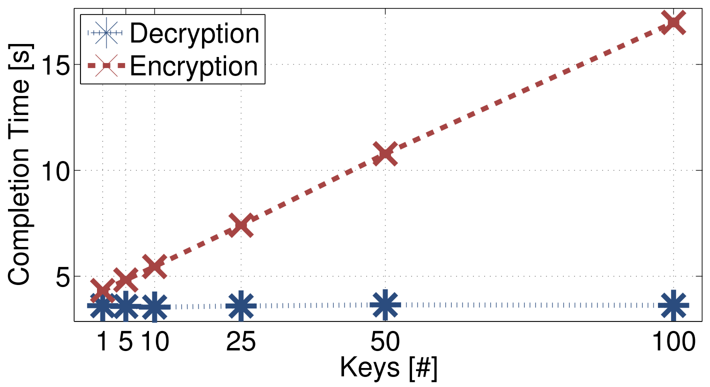


Figure 16: Encryption and decryption completion time, as a function of the number of keys. We consider 100 Mbytes of data and 10 policy updates.

sumers perform the corresponding decryption. Intermediate routers need not execute any of these cryptographic functions, and therefore this design choice ensures scalability of our security architecture, making it possible to operate at line speed. In this subsection, we quantify the computational overhead introduced on the end-hosts by *ConfTrack-CCN* to perform the cryptographic operations required to implement our mechanism.

To perform such evaluation, we implemented a Java prototype of the encryption primitives of *ConfTrack-CCN*, and performed extensive evaluations to understand the performance impact of these procedures. All tests have been executed on a dual Intel Xeon 2.2GHz machine, with 64GB RAM, running Ubuntu Linux 12.04.2 LTS, using Java SE 7u25 with the Bouncy Castle provider for the Java Cryptography Extension and the Java Cryptography Architecture. The performance metric considered in all these evaluations is the completion time of the cryptographic primitives, as a function of:

1. The *number of encryption keys*;
2. The *number of policy updates*;
3. The *size of the data*, expressed in Mbytes.

Unless otherwise stated, we performed the tests encrypting 100 Mbytes of data with 10 different keys, considering 10 policy updates. For each scenario that we took into account we performed 10 runs, and in Fig. 16-18 we further reported the corresponding (narrow) 95% confidence intervals.

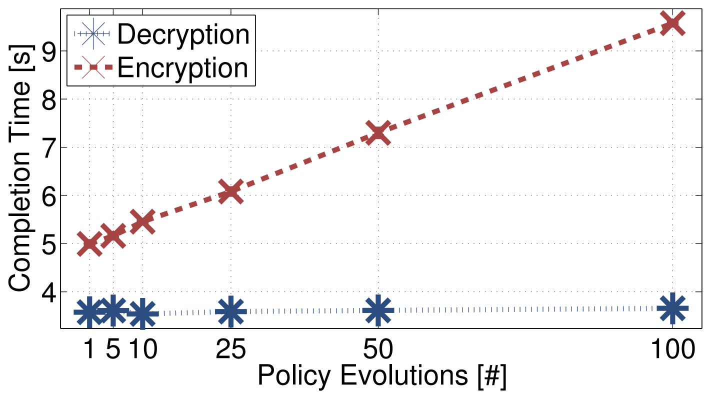


Figure 17: Encryption and decryption completion time, as a function of the number of policy evolution. We consider 100 Mbytes of data and 10 keys.

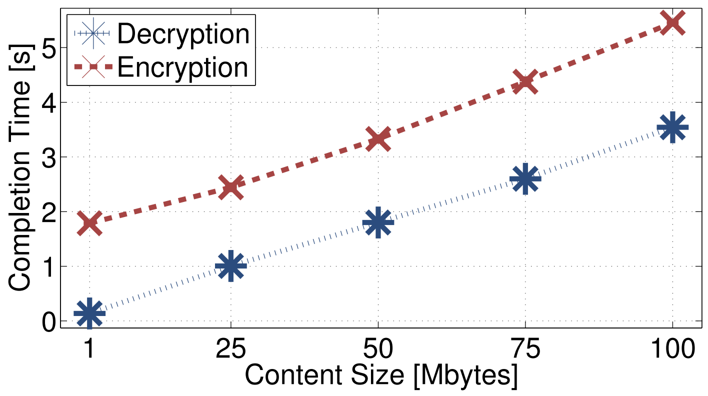


Figure 18: Encryption and decryption completion time, as a function of the size of the content that has to be processed. We consider 10 keys and 10 policy updates.

As described in Appendix A, the KR-RSA algorithm generates a public-private RSA key pair for each Second Layer encryption key. Despite the fact that such generation is computationally expensive, our proposed mechanism minimizes its performance impact in two ways: (1) the key-pairs have to be computed only once that new data is published on the network, and (2) the pairs can also be pre-computed offline.

When releasing a new version of the data by performing a policy update, modular exponentiation is used to *wind/unwind* the keys; the other operations require only to perform fast symmetric AES encryption. It is interesting to note that the fraction of re-encrypted content only affects the network hit-rate as discussed in Sec. 5.1, while it does not have any impact on the computational performance, since it only requires to choose an appropriate Second-Layer key among those already generated.

Fig. 16 shows the encryption/decryption completion time as a function of the number of Second-Layer keys. A linear time is required to compute the RSA key-pairs, and such computational cost is only paid on the content provider side. Furthermore, we observe that cryptographic primitives introduce only minor overhead on the consumer side.

The policy evolution frequency as well as the size of the data are positively correlated with the completion time of the algorithm, as depicted in Fig. 17 and 18, respectively. The gap between the encryption/decryption curves is due to the cost for initializing the RSA key-pairs, which is paid only by the content provider, once the data is originally published on the network.

6. Related Work

Content-Centric Networking aims at providing solid grounds for the Next-generation Internet architecture, not only by fostering efficient content distribution, but also by improving network security. However, novel issues arise when the addressing space changes from the host-based model to content-based naming; in particular, the following topics have already been investigated: *Integrity and Trust*, *Denial of Service*, *Privacy* and *Access Control*.

Integrity and Trust. Among the advertised advantages of CCN, *security* is one of the relevant achievements. In [8], Jacobson et al. describe the networking primitives offered by NDN/CCN that permit to improve network security in a scenario where contents can be retrieved from any node storing a copy of the given data. In particular, they formulate the proposal according to which, rather than securing the communication channel between the two end-points, CCN data packets will be digitally signed by content producers to enforce integrity and authenticity.

Due to the fact that asymmetric encryption techniques are used to produce the signatures, the relevant problem of determining trust of the exchanged keying material needs to be taken into account, as envisioned by Smetters and Jacobson in [21]. To foster openness, the NDN/CCN proposal does not force to adopt any trust management architecture for key distribution and, in the general case, many techniques will jointly be used.

A proposal for one such key distribution architecture was recently formulated by Mahadevan et al. in [22], where a Key Resolution Service (KRS) for NDN/CCN is presented. Inspired by the hierarchical structure of DNS, the proposed KRS system can be used to register, store and efficiently distribute keying materials associated with CCN namespaces, providing adequate support to help consumers check content integrity. Despite the fact that content integrity can be verified by nodes in the CCN, the presence of the distributed caches and the way consumers issue interest packets on the network let adversaries perform *content poisoning* attacks to distribute fake copies of data. Ghali et al. present in [23] a ranking algorithm for cached content, specifically designed to foster eviction from the caching storage of poisoned contents, by gathering statistics on consumers' actions as a result of previous content delivery.

Denial of Service. Additional security implications of this novel network model have also been studied in the literature. In [24], Gasti et al. do a preliminary evaluation of Denial of Service (DoS) attacks in NDN; in particular, they argue that interest flooding as well as content/cache poisoning represent the most serious threats. Performing such a kind of attacks is demonstrated to be possible in [25], where Compagno et al. present "Poseidon", a framework that uses local detection as well as collaborative techniques based on the push-back approach to limit the feasibility of interest flooding attacks in NDN. DoS attacks are again the topic addressed by Goergen et al. in [26], where they formulate a proposal for a monitoring architecture that can detect attack patterns by tracking recent activity happening over the basic data structure of an NDN node.

While content poisoning attacks aim at disrupting integrity by making adversaries provide novel copies of contents that do not correctly represent the real data requested, cache pollution attacks instead aim at disrupting cache efficiency by breaking the temporal correlation of requests. Since they can severely affect the overall network performance, we group them within the DoS class. As thoroughly discussed in [27], cache pollution attacks can be effectively performed even by an adversary that possesses limited resources. Since cache pollution attacks can have a network-wide effect and are not confined to a single node, Xie et al. present in [27] CacheShield, a pro-active mechanism to prevent cache pollution in CCN. An alternative approach is proposed by Conti et al. in [28], where the authors formulate a novel pollution detection algorithm. Compared to CacheShield, the solution envisioned by Conti et al. has a smaller computational footprint while ensuring at the same time a high overall hit-rate.

Privacy. Despite the fact that packets in NDN do not contain any information regarding the source host from which they were originally sent, this characteristic is not sufficient to protect the privacy of the users. Moreover, both Interest and Data packets leak the name of the contents that users are willing to retrieve, therefore posing remarkable privacy concerns for NDN. In order to mitigate this issue, DiBenedetto et al. in [29] propose ANDaNA, an adaptation of onion-routing [30] specifically tailored for NDN. By using multiple layers of encryption, they make sure that an eavesdropper cannot leak sensitive information. Another approach based on homomorphic encryption is instead proposed by Fotiu et al. in [31]. Despite the fact that one such proposal is specifically tailored for the Publish-Subscribe paradigm, the authors claim that it can also be applied to CCN and other infrastructures. They envision the implementation of a brokering system that consumers query to retrieve pointers to the requested information items, without making the content provider or the brokers know what type of content the user is willing to access. Another class of attacks is presented in [32], where Acs et al. focus on timing attacks on caches: in this case, the adversary learns whether a consumer has requested a given content by querying the neighboring routers and analyzing the delay. Cache privacy techniques are then analyzed with the precise aim to mitigate the effects of a timing attack on the distributed caches. Finally, a comprehensive analysis of privacy issues in ICN, attack categories, adversary models and potential remediation is provided in [33]. More specifically, authors develop a generic ICN model that can be applied to different ICN architectures, to map architectural design choices to potential privacy issues that arise in one such context.

Access Control. The topic of access control in CCN, as clearly recognized in [26], is extremely important and still deserves further attention. In [34], Zhu et al. propose a mechanism to enforce confidentiality for a conference tool in NDN. Their proposal is based on a centralized approach where the user hosting the conference also handles the generation of symmetric encryption keys to secure the voice stream, while using public-key cryptography to distribute keying materials. However, one such solution is specifically designed for live streaming applications, and is not suited for a more generic scenario where large files are distributed through the CCN. Burke et al. formulate in [35] a proposal to secure light control; however their work, like [34], can hardly be extended to other more general cases.

A thorough description of the encrypted access control mechanism implemented in the CCNx prototype [36] is described by Smetters et al. in [37]. Nonetheless, one such proposal lacks dedicated support for massive content distribution in the network, as well as adequate support for policy evolution and access feedback, which is instead the focus of our solution.

The works that most closely resemble ours are illustrated hereafter. Fotiu et al. study in [38] an access control enforcement delegation mechanism for an ICN, which guarantees consumers' privacy in the PURSUIT [39] architecture. However, one such proposal is not tailored for CCN; in fact, while in PURSUIT can be assumed that a caching node, called *Relaying Party* (RP), will provide the content only to the subset of authenticated users, one such requirement can instead hardly be enforced considering in-network caching as implemented in CCN. On top of that, their focus on privacy preservation makes it very difficult to jointly satisfy the *trackability* requirement.

Zhang et al. propose in [40] to enforce the confidentiality and integrity requirements by leveraging the services provided by Identity-Based Encryption (IBE). The main advantage gained by using such technique is that the sender does not need to obtain the public key certificate of the receiver to transmit data securely. However, the main disadvantage of their proposal is that they do not take into account in-network caching, making the security mechanism not suited to support efficient content distribution in CCN.

Proxy Re-Encryption (PRE) is used in [41] to support *access-control* in a content-centric network. Among the advantages of one such solution, the overall efficiency of the network is supported by the fact that in-network caching mechanisms can take advantage of this paradigm: like in our ConfTrack-CCN design, the cyphertext may securely be cached and served by the CCN routers, whereas keying materials are disclosed by the producer only upon verification of the access policy. To foster the large scale application of such paradigm, especially considering the relevant computational overhead introduced, the content will be encrypted using symmetric encryption, while PRE techniques will only be used to compute the symmetric decryption key. This proposal is therefore prone to users' collusion: an adversary can easily break the security of the overall system by publishing the symmetric decryption key.

Inspired by the works on secure video content streaming, Misra et al. present in [42] a security architecture specifically tailored to support content distribution in an ICN. By using Broadcast Encryption and, more specifically, a public-key traitor tracing variant of Shamir's threshold secret sharing scheme, their proposed security architecture provides support for confidential communications in CCN, while offering viable techniques to revoke access to any given user. On top of that, another positive advantage of this architecture is that the chosen cryptosystem provides traitor-tracing features, meaning that colluding users disclosing their private decryption keys will indirectly disclose information on their identity. Our proposed *ConfTrack-CCN* is instead designed on the solid grounds of standard symmetric encryption and hash functions, in order to reduce the computational overhead to execute the cryptographic primitives. Moreover, it also provides functionalities to effectively support content access trackability, a feature that is instead neglected by the system proposed in [42].

To cope with the large files distribution, the solutions reviewed so far use one layer of symmetric encryption to efficiently secure the data, while enforcing access control protection by using different versions of public-key encryption (such as IBE, PRE, BE), to securely distribute keying materials, and ensuring that only authorized users can retrieve the correct decryption key. However, all these systems can be easily bypassed by making users disclose the common symmetric encryption key used to initially protect the content. On top of that, since the symmetric key is shared by all the users, one such strategy does not reveal any type of information regarding the user's identity. Our *ConfTrack-CCN* mechanism, instead, is tailored to avoid this issue. In fact, in our design, contents are encrypted twice, using two symmetric keys. To break the system security, users must disclose the Second Layer decryption keys, but, doing that, they are also forced to expose their identity. Furthermore, by relying on standard hash functions and symmetric encryption, our solution significantly improves the overall efficiency of the mechanism. Finally, for the sake of clarity, Table 3 provides a thorough comparison

Table 3: Comparison of Related Works on Access Control.

	ConfTrack-CCN	**[38]**	**[40]**	**[41]**	**[42]**
Confidentiality	✓	✓	✓	✓	✓
Privacy Preservation	-	✓	-	-	-
Trackability	✓	✗	✗	-	✗
Access Policy Evolution	✓	-	Revocation is not trivial	✓	✓
Collusion Protection	✓	-	-	✗	✓
Cache Awareness	✓	In PURSUIT, not in CCN	✗	✓	✓
Performance	Sym. Enc. Hash Functions	Sym. Enc.	IBE	PRE	BE

of our ConfTrack-CCN scheme with respect to the most notable reviewed literature on access control enforcement.

7. Conclusions

We proposed *ConfTrack-CCN*, the first *cache-aware, encryption-based* mechanism designed to enforce confidential and trackable content dissemination in Content-Centric Networks, while seamlessly supporting policy evolution. With our mechanism, the consumers can take advantage of the distributed caches (one of the core elements of CCNs) to retrieve the encrypted data, while they are instead forced to contact directly the content producer to fetch keying material, thus authenticating themselves and providing access trackability feedback.

We evaluated our solution by both developing a mathematical model and through an accurate simulation analysis with real network topologies, showing that the proposed mechanism always performs better than user-based encryption. In the most adverse case, *ConfTrack-CCN* ensures a 25% hit rate, while user-based encryption schemes can barely reach 5%. A large gap is also observable in the best case, where our solution scores a 91% hit-rate, compared to the 58% of user-based schemes. By distributing different key sets to the users, our proposal provides robust collusion detection and prevention functionalities, even when 60% of the users are colluding.

On top of that, we also quantified the computational overhead introduced by *ConfTrack-CCN* by implementing a Java prototype of our solution and performing extensive measurements. We demonstrated that on the consumer side our solution has a negligible performance cost since it only executes fast operations (it decrypts 100 Mbytes of data in less than 4 seconds). At the same time, our design limits the impact of demanding cryptographic primitives such as the generation of the Second-Layer encryption keys: this operation has to be performed on the content provider side only once.

Appendix A. KR-RSA

In this appendix we describe the KR-RSA key derivation algorithm.

The key regression algorithm is the mechanism we use to generate a sequence of encryption keys that appear to be pseudorandom. While the content provider can compute subsequent versions of a key, the consumers can only derive previous versions. When compared to other key-derivation algorithms such as S/KEY [14], key regression (in its KR-RSA formulation [13]), has two advantages: (1) the content producer can generate an unbounded number of keys (i.e.: the maximum wind parameter MW is *infinite*), and (2) subsequent keys of the sequence appear as pseudorandom.

For the sake of clarity, we use the same notation as in Sec. 3.2. The content producer generates and persists securely the *producer state* $stp(h, l)$, while it discloses the *consumer state* $stc(h, l)$ to its customers. We denote with $SK(h, l)$ the encryption key, where h is the version, while l is the seed used to initialize the algorithm.

Given $stc(h, l)$, a consumer can efficiently compute all the previous states $stc(1, l), ..., stc(h-1, l)$ as well as the corresponding keys $K(1, l), ..., K(h-1, l)$, but it cannot compute subsequent states $stc(h+i, l), \forall i \geq 1$. The sequence of keys is pseudorandom, whereas the sequence of consumer states is not: in fact, it is possible for an adversary to distinguish future consumer states from random bits, but the same does not apply to keys.

The key regression scheme is completely identified by four algorithms:

1. **Setup**: this operation is executed by the content producer and is used to generate the initial producer state $stp(h, l)$ (Algorithm 2).

2. **Wind key**: this operation is executed by the content producer. Given the current

state *stp*(h, l), it returns the corresponding consumer state *stc*(h, l) as well as the subsequent producer state *stp*$(h + 1, l)$ (Algorithm 3).

3. **Unwind key**: this operation is executed by the consumer. Given the current consumer state *stc*(h, l), it returns the previous state *stc*$(h - 1, l)$ (Algorithm 4).

4. **Key Derivation**: this operation can be executed by both a content producer and the consumer. Given the current consumer state *stc*(h, l) it returns the corresponding encryption key $SK(h, l)$ (Algorithm 5).

In Step 1 of Alg. 2, we use an RSA key generator denoted with $\mathcal{K}_{rsa}$ to compute (N, e, d), where N is the RSA modulus, e is the public exponent, while d is the private key exponent. In Step 2 a random unsigned integer S in $\mathbb{Z}_N$, with standard, big endian encoding is generated.

Using the well known properties of RSA in Step 1 of Alg. 3, the new value for S is computed, whereas in Step 1 of Alg. 4 the previous value of S is derived. By keeping d secret, the one-way property is guaranteed. Lastly, in Step 1 of Alg. 5, by using the SHA1 hash function, key pseudorandomness is enforced by the mechanism.

Algorithm 2: KR-RSA - Setup

```
  Output: stp(h,l)
1 $(N, e, d) \Leftarrow \mathcal{K}_{rsa}$ ;
2 $S \Leftarrow \mathbb{Z}_N^*$ ;
3 $h \Leftarrow S$ ;
4 $l \Leftarrow \langle N, e \rangle$;
5 stp(h,l) $\Leftarrow \langle h, l, d \rangle$;
```

Algorithm 3: KR-RSA - Wind Key

```
  Input  : stp($h = S, l = \langle N, e \rangle$)
  Output: stp($h', l$), stc($h, l$)
1 $S' \Leftarrow S^d \bmod N$ ;
2 $h' \Leftarrow S$ ;
3 stp($h', l$) $\Leftarrow \langle h', l, d \rangle$ ;
4 stc($h, l$) $\Leftarrow \langle h, l \rangle$ ;
```

Algorithm 4: KR-RSA - Unwind Key

```
  Input  : $\mathrm{stc}(h = S, l = \langle N, e \rangle)$
  Output: $\mathrm{stc}(h', l)$
1 $h' \Leftarrow S^e \bmod N$ ;
2 $\mathrm{stc}(h', l) \Leftarrow \langle h', l \rangle$ ;
```

Algorithm 5: KR-RSA - Key Derivation

```
  Input  : $\mathrm{stc}(h = S, l = \langle N, e \rangle)$
  Output: $SK(h, l)$
1 $SK(h, l) \Leftarrow \mathrm{SHA1}(S)$ ;
```

References

[1] B. M. Leiner, V. G. Cerf, D. D. Clark, et al. The Past and Future History of the Internet. *Communications of the ACM*, 40(2):102–108, 1997.

[2] G Carofiglio, G Morabito, L Muscariello, I Solis, and M Varvello. From content delivery today to information centric networking. *Computer Networks*, 57(16):3116–3127, 2013.

[3] Yusung Kim and Ikjun Yeom. Performance Analysis of In-Network Caching for Content-Centric Networking. *Computer Networks*, 57(13):2465–2482, 2013.

[4] Akamai Website. `http://www.akamai.com/`, Last accessed: January 2014.

[5] George Pallis and Athena Vakali. Insight and Perspectives for Content Delivery Networks. *Communications of the ACM*, 49(1):101–106, 2006.

[6] Christian Dannewitz. NetInf: An Information-Centric Design for the Future Internet. In *Proc. 3rd GI/ITG KuVS Workshop on The Future Internet*, Munich, Germany, May, 2009.

[7] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192, 2007.

[8] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking Named Content. In *Proc. of the 5th Int.l conference on Emerging networking experiments and technologies (CoNEXT)*, pages 1–12. ACM, 2009.

[9] Guoqiang Zhang, Yang Li, and Tao Lin. Caching in information centric networking: A survey. *Computer Networks*, 57(16):3128–3141, 2013.

[10] HyunYong Lee and Akihiro Nakao. User-assisted in-network caching in information-centric networking. *Computer Networks*, 57(16):3142–3153, 2013.

[11] Jaime Llorca, Antonia Tulino, Kyle Guan, Jairo Esteban, Matteo Varvello, Nakjung Choi, and Daniel Kilper. Dynamic In-Network Caching for Energy Efficient Content Delivery. In *Proc. of IEEE INFOCOM*, pages 245–249, Turin, Italy, April, 2013.

[12] Ryad Benadjila, Olivier Billet, Shay Gueron, and MattJ.B. Robshaw. The intel aes instructions set and the sha-3 candidates. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*,

volume 5912 of *Lecture Notes in Computer Science*, pages 162–178. Springer Berlin Heidelberg, 2009.

[13] Kevin Fu, Seny Kamara, and Yoshi Kohno. Key Regression: Enabling Efficient Key Distribution for Secure Distributed Storage. In *Proc. of the 13th Annual Network & Distributed System Security Symposium (NDSS 2006), San Diego, CA, USA*, February, 2006.

[14] Neil Haller. The S/KEY One-Time Password System. In *Proc. of the Internet Society Symposium on Network and Distributed Systems*, pages 151–157, February 1994.

[15] Dario Rossi and Giuseppe Rossini. On Sizing CCN Content Stores by Exploiting Topological Information. *IEEE INFOCOM, NOMEN Workshop*, pages 280–285, 2012.

[16] Christine Fricker, Philippe Robert, and James Roberts. A Versatile and Accurate Approximation for LRU Cache Performance. In *Proc. of the 24th International Teletraffic Congress, Krakow, Poland*, September, 2012.

[17] Hao Che, Ye Tunk, and Zhijun Wang. Hierarchical Web Caching Systems: Modeling, Design and Experimental Results. *IEEE JSAC*, 20(7):1305–1314, 2002.

[18] Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM: NDN simulator for NS-3. Technical Report NDN-0005, NDN, October 2012.

[19] Géant Network Website. `http://geant3.archive.geant.net/Network/NetworkTopology/pages/home.aspx`, Last accessed: January 2014.

[20] Christine Fricker, Philippe Robert, Jim Roberts, and Nada Sbihi. Impact of traffic mix on caching performance in a content-centric network. In *IEEE NOMEN 2012, Workshop on Emerging Design Choices in Name-Oriented Networking*, Orlando, USA, Mar. 2012.

[21] Diana Smetters and Van Jacobson. Securing Network Content. Technical report, PARC, technical report TR-06-11, October 2009.

[22] Priya Mahadevan, Ersin Uzun, Spencer Sevilla, and JJ Garcia-Luna-Aceves. CCN-KRS: a key resolution service for CCN. In *Proc. of the 1st Int.l conf. on Information-centric networking (ICN)*, pages 97–106. ACM, Paris, France, Sep. 2014.

[23] Cesar Ghali, Gene Tsudik, and Ersin Uzun. Needle in a Haystack: Mitigating Content Poisoning in Named-Data Networking. In *Proc. of NDSS Workshop on Security of Emerging Networking Technologies (SENT)*, San Diego, CA, USA, Feb. 2014.

[24] Paolo Gasti, Gene Tsudik, Ersin Uzun, and Lixia Zhang. DoS and DDoS in Named-Data Networking. Technical report, University of California, Irvine, 08 2012.

[25] Alberto Compagno, Mauro Conti, Paolo Gasti, and Gene Tsudik. Poseidon: Mitigating Interest Flooding DDoS Attacks in Named Data Networking. Technical report, University of California, Irvine, 2013.

[26] David Goergen, Thibault Cholez, Jérôme François, and Thomas Engel. Security monitoring for Content Centric Networking. In *Data Privacy Management and Autonomous Spontaneous Security*, volume 7731, pages 274–286. 2013.

[27] Mengjun Xie, Indra Widjaja, and Haining Wang. Enhancing cache robustness for content-centric networking. In *Proc. of IEEE INFOCOM*, pages 2426–2434, Orlando, Florida, USA, March 2012.

[28] Mauro Conti, Paolo Gasti, and Marco Teoli. A lightweight mechanism for detection of cache pollution attacks in Named Data Networking. *Computer Networks*, 57(16):3178–3191, 2013.

[29] Steven DiBenedetto, Paolo Gasti, Gene Tsudik, and Ersin Uzun. ANDaNA: Anonymous named data networking application. In *Proc. of the 19th Annual Network & Distributed System Security Symposium (NDSS 2012), San Diego, CA, USA*, February 2012.

[30] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proc. of the 13th USENIX Security Symposium, San Diego, CA, USA*, August 2004.

[31] N. Fotiou, D. Trossen, G.F. Marias, A. Kostopoulos, and G.C. Polyzos. Enhancing information lookup privacy through homomorphic encryption. *Security and Communication Networks*, 2013.

[32] Gergely Acs, Mauro Conti, Paolo Gasti, Cesar Ghali, and Gene Tsudik. Cache Privacy in Named-Data Networking. In *Proc. of the 33rd Int.l Conference on Distributed Computing Systems (ICDCS 2013), Philadelphia, PA, USA*, July 2013.

[33] Nikos Fotiou, Somaya Arianfar, Mikko Särelä, and George C. Polyzos. A framework for privacy analysis of icn architectures. In Bart Preneel and Demosthenes Ikonomou, editors, *Privacy Technologies and Policy*, volume 8450 of *Lecture Notes in Computer Science*, pages 117–132. Springer International Publishing, 2014.

[34] Zhenkai Zhu, Jeffery Burke, Lixia Zhang, Paolo Gasti, Yanbin Lu, and Van Jacobson. A New Approach to Securing Audio Conference Tools. In *Proc. of the 7th Asian Internet Engineering Conference (AINTEC), ACM*, pages 120–123, Bangkok, Thailand, November 2011.

[35] Jeff Burke, Paolo Gasti, Naveen Nathan, and Gene Tsudik. Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control and NDN. In *Proc. of the 2nd IEEE Int.l Workshop on Emerging Design Choices in Name-Oriented Networking (NOMEN)*, Turin, Italy, April 2013.

[36] CCNx Website. `https://www.ccnx.org`, Last accessed: January 2014.

[37] Diana Smetters, Philippe Golle, and Jim Thornton. CCNx Access Control Specifications. Technical report, Technical report, PARC, 2010.

[38] Nikos Fotiou, Giannis F. Marias, and George C. Polyzos. Access Control Enforcement Delegation for Information-centric Networking Architectures. In *Proc. of the 2nd ICN Workshop on Information-centric Networking*, ICN '12, pages 85–90. ACM, Helsinki, Finland, August 2012.

[39] Nikos Fotiou, Pekka Nikander, Dirk Trossen, and George C. Polyzos. Developing Information Networking Further: From PSIRP to PURSUIT. In *Broadband Communications, Networks, and Systems*, volume 66 of *LNCS, Social Informatics and Telecommunications Engineering*, pages 1–13. Springer, 2012.

[40] Xinwen Zhang, K. Chang, Huijun Xiong, Yonggang Wen, Guangyu Shi, and Guoqiang Wang. Towards name-based trust and security for content-centric network. In *IEEE Int.l Conf. on Network Protocols (ICNP)*, pages 1–6, Vancouver, BC Canada, Oct. 2011.

[41] Christopher A Wood and Ersin Uzun. Flexible End-to-End Content Security in CCN. In *IEEE Consumer Communications and Networking Conference (CCNC)*, pages 1–8, Las Vegas, NE, USA, Jan. 2014.

[42] Satyajayant Misra, Reza Tourani, and Nahid Ebrahimi Majd. Secure Content Delivery in Information-centric Networks: Design, Implementation, and Analyses. In *Proc. of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ICN '13, pages 73–78. ACM, Hong Kong, China, Aug. 2013.