Bernardini, Cesar; Marchal, Samuel; Asghar, Muhammad Rizwan; Crispo, Bruno

PrivICN

Published in:
Computer Networks

DOI:
10.1016/j.comnet.2018.11.012

Published: 11/02/2019

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Published under the following license:
CC BY-NC-ND

Please cite the original version:
Bernardini, C., Marchal, S., Asghar, M. R., & Crispo, B. (2019). PrivICN: Privacy-preserving content retrieval in information-centric networking. *Computer Networks*, *149*, 13-28. https://doi.org/10.1016/j.comnet.2018.11.012

PrivICN: Privacy-Preserving Content Retrieval in Information-Centric Networking

Cesar Bernardini[a], Samuel Marchal[b], Muhammad Rizwan Asghar[c], Bruno Crispo[d]

[a]*Barracuda Networks, Austria*
[b]*Department of Computer Science, Aalto University, Finland*
[c]*Cyber Security Foundry, The University of Auckland, New Zealand*
[d]*Department of Information Engineering and Computer Science, University of Trento, Italy*

Abstract

Information-Centric Networking (ICN) has emerged as a paradigm to cope with the increasing demand for content delivery on the Internet. In contrast to the Internet Protocol (IP), the underlying architecture of ICN enables users to request contents based on their name rather than their hosting location (IP address). On the one hand, this preserves users' anonymity since packet routing does not require source and destination addresses of the communication parties. On the other hand, semantically-rich names reveal information about users' interests, which poses serious threats to their privacy. A curious ICN node can monitor the traffic to profile users or censor specific contents for instance. In this paper, we present *PrivICN*: a system that enhances users privacy in ICN by protecting the confidentiality of content names and content data. *PrivICN* relies on a proxy encryption scheme and has several features that distinguish it from existing solutions: it preserves full in-network caching benefits, it does not require end-to-end communication between consumers and providers and it provides flexible user management (addition/removal of users). We evaluate *PrivICN* in a real ICN network (CCNx implementation) showing that it introduces an acceptable overhead and little delay. *PrivICN* is publicly available as an open-source library.

Keywords: ICN, Information-centric networking, content names, privacy, confidentiality, proxy encryption

1. Introduction

The Internet was initially designed as a network providing host-to-host communication using IP addresses. However, it has dramatically evolved over time to become a global platform used by governments, corporations and individuals for sharing a large amount of data. According to Cisco [1], the global mobile data traffic will increase sevenfold in the next five years. Moreover, all forms of content delivery in the Internet are expected to represent 86% of the whole traffic [1]. In order to cope with this growing demand, researchers have proposed to replace the existing host-to-host communication network with an *information-centric* one [2]. Several Information-Centric Networking (ICN) architectures have been presented, such as PURSUIT [3], DONA [4], Named-Data Networking (NDN) [5] or Content-Centric Networking (CCN) [6]. Among them, NDN and CCN have received most of the attention from the community because of three features: hierarchical naming scheme at networking layer, coupling of name forwarding and data routing, and simple easy-to-manage caching features (for efficiency reasons) at every node of the network [7, 8].

In ICN architectures, contents are searched for using their names. These names are semantically tied to the content they address. Content names and content data are sent within the network in plaintext and accessible to any ICN node forwarding them. This leaks sensitive information about interests of users requesting and receiving contents. A curious ICN node can use this information to profile users or sell it to marketing companies for targeted advertisement. A malicious ICN node can prevent the delivery of specific contents in a censorship fashion. Therefore, the confidentiality of contents and their names must be protected to avoid these threats and enhance users privacy.

There exist solutions for protecting the confidentiality of names [9, 10, 11], content data [12] or both [13, 14] in ICN. However, no solution exists that protects both names and contents confidentiality while preserving some basic features of ICN. For instance, they remove [15, 16] or impair [9] in-network caching benefits, or require end-to-end communications and security association between consumers and providers [10]. This degrades ICN performance and corrupts users' anonymity. Moreover, existing solutions suffer from scalability and flexibility issues with respect to user management. Most schemes require to re-generate a large number of keys and/or the re-encryption of content names and content data stored on ICN nodes [13, 14], when handling the addition and removal of users.

To cope with these issues, we propose a system named *PrivICN* (PRIVacy-preserving content retrieval in ICN) that enhances users privacy in ICN by protecting the confidentiality of content names and content data. *PrivICN* relies on a proxy encryption mechanism [17, 18] in which content consumers, content providers and ICN nodes do not share any key. All contents and names are in an encrypted form while in-network and rout-

Email addresses: mesarpe@gmail.com (Cesar Bernardini), samuel.marchal@aalto.fi (Samuel Marchal), r.asghar@auckland.ac.nz (Muhammad Rizwan Asghar), bruno.crispo@unitn.it (Bruno Crispo)

ing operations are performed on these encrypted entities. Only content consumers and content providers can ultimately decrypt the content data, which is protected from any ICN node. The contributions of this paper are:

- a set of generic design goals that any privacy-preserving solution for ICN must meet to be deployable and usable while achieving its purpose (Section 2.3). These goals justify the design choices we made for *PrivICN* and they can help in designing future proposals for protecting privacy in ICN.
- a privacy-preserving system for content request and delivery over ICN: *PrivICN* (Sections 4 and 5). In contrast to previous solutions, it preserves all ICN features (including in-network caching and no requirement for end-to-end communications) and it provides flexible user management (addition/removal of users).
- an experimental evaluation of *PrivICN* in a real ICN network (Section 6). It shows that it is deployable and induces an acceptable overhead in terms of storage and computation. It introduces an acceptable delay in network communications (≈30%).
- an open-source library implementing *PrivICN* available at [19].

This paper is an extended version from a previous publication [20] and includes the following major additions. (1) We introduce a set of design goals for any privacy-preserving solution for ICN and we evaluate the extent to which our proposed solution (*PrivICN*) meet them. (2) We provide the implementation for the cryptographic primitives of *PrivICN* and make them available as a public library. (3) We present an extended evaluation of the proposed solution including its assessment in a real CCNx network deployment.

2. Background and Problem Description

2.1. ICN Overview

Information-Centric Networking is a clean-slate approach for the future Internet. Since there are many ICN architectures [7], we focus on the most popular ones in this paper: CCN [6] and NDN [5]. We present their principles and name them indifferently using the generic term *ICN* in the remaining of the paper. ICN communication is based on two primitives: *interest* and *data* messages. The interest expresses the will of a client for content, while a data message contains the answer for that content. The communication paradigm is led by names, known as *content names*, which are contained in the interest and data messages. These names are made of hierarchically organized and human-readable components. For instance, */it/Venice* and */org/wikipedia/Maradona* are two valid ICN names composed of 2 and 3 components, respectively. To refer to content stored in *data* messages we will use the term content data.

To support content retrieval and delivery, every ICN node holds three tables:

- *Content Store (CS)* is a caching structure that stores content data temporarily.
- *Pending Interest Table (PIT)* keeps track of the currently non-satisfied interests. It serves as a trace of the reverse path for a requested content data to be delivered through once found.
- *Forwarding Interest Base (FIB)* is a routing table used to determine the interface to forward interest messages through.

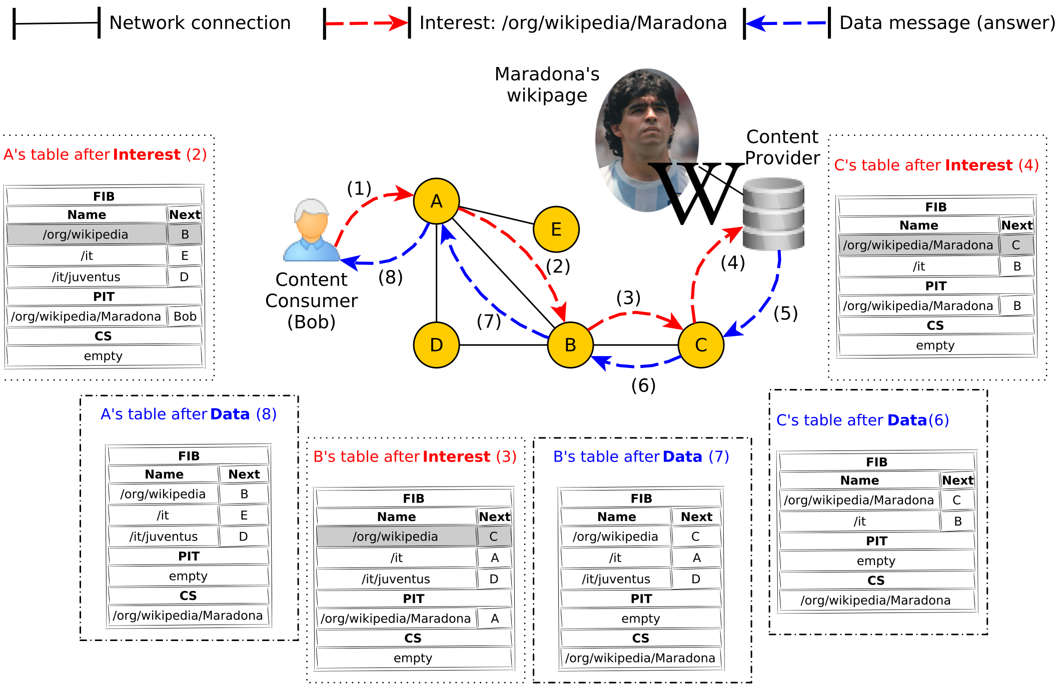
A client sends an interest to an *ICN edge node*, namely the first node it is connected to and is at the edge of the network. The ICN edge node receiving the interest extracts the content name and looks it up in its CS to determine if it is locally cached. If so, the content is sent back through the interface over which the interest arrived and the interest is discarded. Alternatively, the node lookups the PIT to determine whether it is already waiting for the requested content. If an entry is found, the PIT is updated to add a new incoming interface for the content name and the interest is discarded. If no match is found in both CS and PIT, the node operates the longest prefix match of the received content name against the FIB. The result of this operation determines the interface on which the interests must be forwarded. The next ICN node receiving the interest applies the same procedure until the requested content is eventually found.

Figure 1 depicts an example of content retrieval and delivery in an ICN network. It shows the CS, PIT and FIB table state after each step of the procedure. The client Bob is willing to retrieve the wikipedia page of the football player Maradona. Bob creates an interest message with the content name */org/wikipedia/Maradona* and sends it to the ICN edge node *A* (1). *A* checks whether the content name has any match in CS and PIT tables. It does not and as a result, the interest message is forwarded to node *B* according to the FIB table (2). A new entry is added to the PIT specifying the content name and incoming interface of the request: *Bob*. *B* runs the same process and forwards the interest to *C* (3). *C* finds a match in its CS (4). The content is encapsulated into a data message (5), which is forwarded through the reverse path using the entries recorded in the PIT (6–8). The intermediary nodes (*A* and *B*) store a copy of the content data in their CS. Finally, Bob receives the wikipedia page of Maradona (8).

2.2. Privacy Issues in ICN

ICN architectures distribute content in plaintext. Any ICN node on the path of an interest or data message can intercept it. Examining the content of an ICN data message can leak information about user interests. This information can be valuable and monetized by third parties, *e.g.,* marketing companies, to perform targeted advertisement campaigns. Similarly, gathering enough content delivered to a user can be used to profile and uniquely identify her. Malicious ICN nodes could also prevent the delivery of contents related to given topics in order to censor a certain type of information [21]. Thus content data needs to be protected and kept confidential to prevent such inference by ICN nodes.

However, protecting only content data does not solve the issue. As previously discussed, the major ICN architectures (CCN and NDN) use a URI like naming scheme for naming



A's table after **Interest** (2)

FIB	
Name	**Next**
/org/wikipedia	B
/it	E
/it/juventus	D
PIT	
/org/wikipedia/Maradona	Bob
CS	
empty	

C's table after **Interest** (4)

FIB	
Name	**Next**
/org/wikipedia/Maradona	C
/it	B
PIT	
/org/wikipedia/Maradona	B
CS	
empty	

A's table after **Data** (8)

FIB	
Name	**Next**
/org/wikipedia	B
/it	E
/it/juventus	D
PIT	
empty	
CS	
/org/wikipedia/Maradona	

B's table after **Interest** (3)

FIB	
Name	**Next**
/org/wikipedia	C
/it	A
/it/juventus	D
PIT	
/org/wikipedia/Maradona	A
CS	
empty	

B's table after **Data** (7)

FIB	
Name	**Next**
/org/wikipedia	C
/it	A
/it/juventus	D
PIT	
empty	
CS	
/org/wikipedia/Maradona	

C's table after **Data**(6)

FIB	
Name	**Next**
/org/wikipedia/Maradona	C
/it	B
PIT	
empty	
CS	
/org/wikipedia/Maradona	

Figure 1: Example ICN architecture (inspired from [7]) in which Bob requests Maradona's wikipage. His interest is disseminated within the network (red path). Once the content found, it is sent back through the same path taken by the interest message (blue path). CS, PIT and FIB table state is shown after each step of the process.

contents. Consequently, names are typically human-readable and have a semantic correlation with their content [22, 23, 24]. Content names contained in interest messages are also sent in plaintext and visible to all forwarders in ICN. Hence, a curious ICN node can easily infer what a content consumer or a group of consumers is interested in by monitoring requested content names. For instance, if we refer to the example illustrated in Figure 1, node A can discover that Bob is interested in Wikipedia and the football player Maradona. By transitivity, nodes B and C can infer the same interest for some content consumers located behind a given interface. The same threats previously presented remain if only the confidentiality of contents is guaranteed. Thus, both content data and content names must be confidential and protected from ICN nodes to enhance user privacy.

Threat model. The system is composed of content consumers, content providers and ICN nodes. We consider an attack model in which attackers are ICN nodes. Their goal is to infer interests for a given consumer based on the messages she sends and receives. ICN nodes are honest-but-curious, *i.e.*, they honestly participate in message routing and content retrieval but can passively monitor the traffic that goes through them. They can apply any kind of analysis on the traffic and collude with other ICN nodes to achieve their goal. We assume they can only perform passive attacks and do not cover active attacks. We do not make any assumptions on content consumers and content providers.

Our goal is to prevent content consumers from revealing their interests to ICN nodes, *i.e.*, preserving the confidentiality of their requested content data. We do not seek to protect the integrity of content data and their names though. This can be achieved by existing methods such as signatures from the content provider.

2.3. Design Goals

We define the following design goals for a deployable solution that provides confidential content delivery, which enhances content consumers privacy in ICN:

- **G1 Confidentiality of information**: ICN nodes must not be able to read contents delivered to a content consumer. In addition, content names must also remain confidential. ICN nodes must be able to perform lookup and longest prefix match on content names without inferring client interests from their requests.
- **G2 Ease of management**: The system must be able to manage a large and varying number of users. Especially, the addition and removal of content consumers and content providers from the system must be possible for a low cost. The replacement of ICN nodes must also be possible for a low cost to guarantee the quality of service.
- **G3 Performance & overhead**: The system must provide efficient content name lookup and fast content delivery. More specifically, the computational overhead and the communication delay must remain acceptable, e.g., $<50\%$ with respect to a plain ICN implementation that does not provide any confidentiality in content delivery.

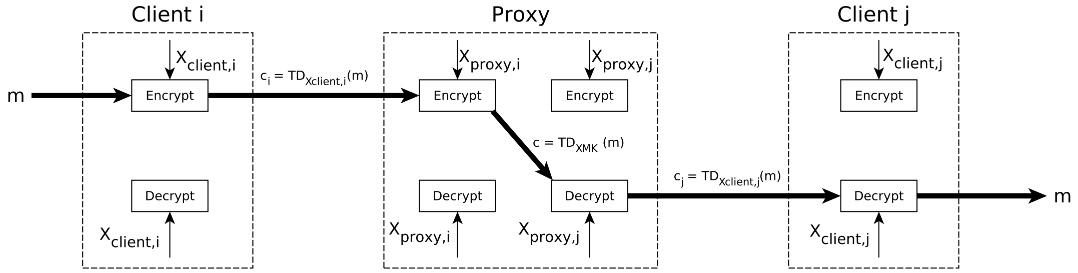


Figure 2: Proxy Encryption Cryptosystem: Clients i and j use a proxy to share encrypted information without leaking secrets.

G4 Preserve ICN features: ICN features that differentiate it from IP must be preserved. These include in-network content caching, name-based routing and abstaining from resorting to end-to-end communication. Also, the solution must not alter the forwarding process of messages, *i.e.,* it must preserve the CS-PIT-FIB workflow.

G5 No security association: Content providers and content consumers must not require to share information between them prior to any content request. The signature of contents can reveal the identity of a content consumer and compromise her anonymity. The establishment of a shared secret requires end-to-end communication and impairs ICN features.

3. Proxy Encryption

To address the proposed research challenges, we propose to adapt an existing cryptographic primitive [17] – proxy encryption – to ICN. Proxy encryption is an asymmetric encryption scheme that we present in this section.

The notion of proxy encryption was first introduced in [25] and solves the following problem. Two clients expect to communicate encrypted information through a proxy without sharing any secret. Every client holds a secret key, which allows it to encrypt/decrypt information. The proxy encryption transforms encrypted ciphertext from one client into encrypted ciphertext for another client, without sharing any information about the keys or the plaintext.

To build a proxy encryption system, we require:

- *Key Management Server.* A KMS is a fully trusted authority that is responsible for generation and revocation of keys. It generates one single master key X_{MK} for the system and several pairs of client/proxy key for every client i in the system such that: $\forall i, X_{MK} = X_{client,i} + X_{proxy,i}$
- *Client Key.* Every client i gets a private key $X_{client,i}$ and keeps it secret.
- *Proxy Keys.* The proxy holds a list of every client it serves and its corresponding proxy key (public key). For instance, if a proxy accepts requests from clients 3, 5 and 12, it holds a table that maps: $3 \rightarrow X_{proxy,3}$, $5 \rightarrow X_{proxy,5}$ and $12 \rightarrow X_{proxy,12}$.
- *Trapdoor.* A trapdoor function is a mathematical function that is easy to compute in one direction but difficult to compute in the opposite direction without having special information. If TD is a trapdoor function, there exists some information x, such that given $TD(m)$ and x, we can easily calculate m. $TD_x(m)$ is the calculus of $TD(m)$ while applying the information x. x is a security key. TD is a commutative function meaning that $TD_y(TD_x(m)) = TD_x(TD_y(m))$.

Figure 2 shows the encryption/decryption process in the proxy encryption cryptosystem. *Client i* intends to send a message m to *Client j*, without sharing information about their secret keys ($X_{client,i}$, $X_{client,j}$) or the plaintext m. To this end, *Client i* encrypts the message m with its secret key $X_{client,i}$ into $c_i = TD_{X_{client,i}}(m)$ and sends c_i to the *Proxy*. The *Proxy* holds a corresponding proxy key for *Client i and j* ($X_{proxy,i}$, $X_{proxy,j}$). It encrypts the received encrypted message c_i from *Client i* with $X_{proxy,i}$ such that $c = TD_{X_{proxy,i}}(c_i)$. The two subsequent trapdoor computations with client key $X_{client,i}$ and proxy key $X_{proxy,i}$ from any client i generate a ciphertext c from m such that $c = TD_{X_{MK}}(m)$. c can be subsequently decrypted with the proxy key corresponding to the target *Client j* to get $c_j = TD_{X_{client,j}(m)}$. The resulting ciphertext c_j is sent to *Client j* that decrypts it with its secret key $X_{client,j}$.

4. *PrivICN* at a glance

The main idea behind *PrivICN* is to employ proxy encryption [17, 18] for providing confidential lookup and retrieval of content data over ICN.

To protect user interest, content consumers transform each component of a content name, *e.g.,* $/c_1/c_2/c_3$, into a corresponding trapdoor $/TD(c_1)/TD(c_2)/TD(c_3)$ using their client keys. All $TD(c_i)$ have the same fixed length, which is independent from c_i. Trapdoors do not leak any information about their original components c_i and make the content name confidential. The content name is sent as part of an interest to an ICN edge node connected to content consumer i, which serves as a proxy and re-encrypts the trapdoors using the proxy key corresponding to the requesting content consumer. Next, the ICN node matches the encrypted content name against encrypted tables stored on ICN nodes (CS, PIT, FIB). All table entries are encrypted in the same form in any ICN node, *i.e.,* as if encrypted using the master key of the system. Content providers follow the same procedure as content consumers to announce content

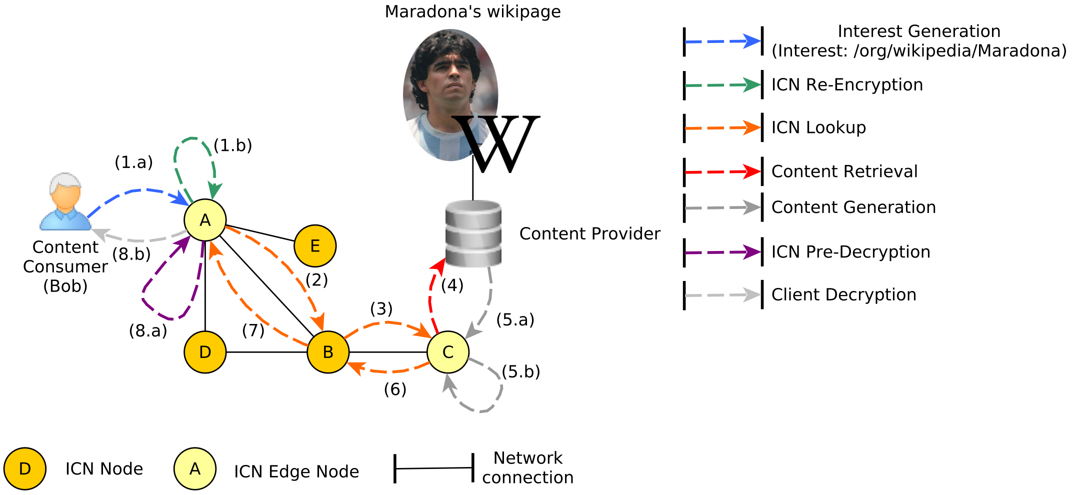


Figure 3: Content request and delivery using *PrivICN*. Additional operations are added to Steps 1, 5 and 8 compared to a typical ICN network.

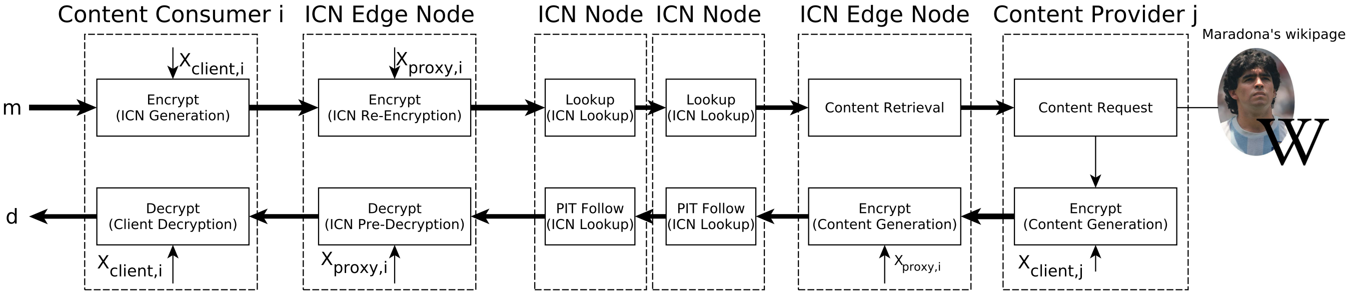


Figure 4: Implementation of PrivICN using Proxy Encryption.

names in the network. A content provider transforms components in content names into trapdoors that are re-encrypted by the ICN edge node it is connected to.

To protect content data, content providers encrypt them into trapdoors. They send the trapdoors to an adjacent ICN edge node that re-encrypts and stores them. The ICN edge node connected to a requesting content consumer acts as a proxy. It pre-decrypts the encrypted content data before sending it to the requesting content consumer that finally decrypts it.

Using *PrivICN*, the flow of messages mainly remains same as presented in Figure 1. There are only small additions to Steps 1, 5 and 8 that we show in Figure 3 and describe in details in Section 5:

- Step 1: the content consumer first generates a trapdoor from the content name it wants to request (1.a – interest generation). The trapdoor is sent to the ICN edge node (A) that re-encrypts it (1.b - ICN re-encryption).
- Step 5: the content provider generates a trapdoor for a content (5.a – content generation). The trapdoor is sent to the ICN edge node (C) that re-encrypts it (5.b - ICN re-encryption).
- Step 8: the content delivery is completed by pre-decryption of the content data by the ICN edge node (A) connected to the requesting content consumer (8.a - ICN pre-decryption). The content consumer can access the requested content by decrypting the pre-decrypted content data with its client key (8.b - Client decryption).

All remaining steps (Steps 2-4 and 6-7) are the same as already illustrated in Figure 1 albeit that matching operations are performed on encrypted content names and tables.

Figure 4 depicts how we adapt standard proxy encryption (depicted in Figure 2) in *PrivICN*. While all proxy encryption operations are traditionally delegated to a single host: the proxy (cf. Section 3), we distribute them among different ICN edge nodes in *PrivICN*. All ICN edge nodes know the content consumers and providers directly connected to them and they have the proxy key for each of them. A single ICN edge node is only in charge of re-encrypting content names and data using the proxy key $X_{proxy,i}$ of the requesting client. This re-encryption produces content names and a data as if encrypted with the master key X_{MK}. All routing operation in ICN core nodes are performed on content names and data encrypted using X_{MK}. At the receiving end of the request, an other ICN edge node will pre-decrypt content data using the proxy key $X_{proxy,j}$ of the receiving client. The receiving client will finally be able to decrypt the content using its client key $X_{client,j}$. This all process is further detailed in Section 5.

We envision *PrivICN* to be deployed in an ecosystem with three parties. The first party are content consumers and content providers of the ICN network. The second party is an Internet Service Provider (ISP) that provides the network infrastructure for content retrieval. The third party is the KMS that provides the client and proxy keys to ICN clients and ICN edge nodes respectively. The ISP wants to provide a privacy-preserving service to its clients (and claim to do so). This scenario is realistic considering the increasing attention given to privacy and the corresponding legal measures taken to protect it, e.g., EU General Data Protection Regulation (GDPR) [26]. One step towards enhancing clients privacy in ICN is by performing content retrieval over confidential data. The ISP contracts the KMS to enforce this confidentiality. The ISP controls all the routers of its network and knows which ones are ICN edge nodes. The ISP would reveal the list of known ICN edge nodes to the KMS, which in turn would only deliver proxy keys to these edge nodes. In such a deployment, any ICN client would be directly connected to an ICN edge node (part of the considered ISP). The ICN edge node acts as the proxy for proxy re-encryption in *PrivICN*.

PrivICN provides end-to-end encryption of both content names and content data between content consumers and content providers. Thus, the design goal **G1** from Section 2.3 is realized since confidentiality of both content names and content data is preserved from ICN nodes. The proxy encryption scheme allows *PrivICN* to store contents and names in a single encrypted form while in the network. This preserves the benefits of in-network caching and allows ICN nodes to perform name-based routing. By providing these features, we do not require end-to-end communication between content consumers and providers. Thus, it demonstrates that *PrivICN* preserves all ICN features and meets design goal **G4**. Moreover, it does not need any security association or key exchange between content consumers and content providers. They only need a private key to participate in the system, which satisfies **G5**.

5. Solution Details

We selected ElGamal encryption system [27] as the basis to implement our proxy encryption system. ElGamal is an asymmetric encryption algorithm where private keys will serve as client keys and public keys as proxy keys. It is worth noting that a client key (private key) is only known by the considered client (content consumer or requester) and the KMS while a proxy key (public key) is known to the KMS and to all ICN edge nodes. However, proxy keys remain private to all clients of the network. In this section, the terms *content consumer* and *content provider* are used interchangeably. While they have different roles in ICN, they have same *client* capabilities in *PrivICN*.

5.1. Initialization and Key Generation

The KMS generates a master secret key and public parameters using the **Init** method on a security parameter k. After setup, the KMS generates keys for every content consumer and content provider in the system. It runs **KeyGen**, which takes as input the identity of a client and generates a key pair including a client key (private key) and a proxy key (public key). The client key $K_{client,i}$ is securely sent to the client i and stored in its key store. The proxy key $K_{proxy,i}$ is provided to every ICN edge node the client i is connected to.

- **Init**(k)**:** Given a security parameter k, the KMS generates two prime numbers p and q such that $q|p-1$. It generates a cyclic group $\mathbb{G}$ of order q with generator g such that $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^*$. It chooses a random $s \in \mathbb{Z}_q^*$ and $X_{MK} \in \mathbb{Z}_q^*$, which is a master secret key and calculates $h = g^{X_{MK}}$. Then, our scheme relies on a pseudorandom function f that takes two arguments s and x (message to encrypt) and returns y. We refer to $f(s,x)$ as $f_s(x)$. The KMS publishes the public parameters $Params = (\mathbb{G}, g, q, h, f)$ and keeps securely the master secret key $MSK = (X_{MK}, s)$.
- **KeyGen**(MSK, i)**:** For each client i, the KMS chooses a random number $X_{client,i} \in \mathbb{Z}_q^*$ and computes $X_{proxy,i} = X_{MK} - X_{client,i}$ where $X_{proxy,i} \neq X_{client,i}$. Next, it sends $K_{client,i} = (X_{client,i}, s)$ securely to client i and $K_{proxy,i} = (i, X_{proxy,i})$ to any edge node adjacent to client i. The edge nodes append the new proxy key for client i to their proxy key store.

The FIB table of each ICN node remains empty at this point and is later populated as contents are published by content providers. Similarly, PIT and CS tables will be populated as content consumers start requesting content data.

5.2. Content & Interest Generation

Content data and content names are encrypted in a different manner in *PrivICN*. While content data needs to be decrypted and available in clear to clients, routing and matching operation in FIB, PIT and CS are operated on encrypted content names. Content names only require to be encrypted in a way that allows comparisons between them.

Content providers publish content data on the network using the **ContentTD** operation. This generates a trapdoor that contains a content data encrypted with the client key of the provider. On the other hand, content names are encrypted by both consumers and providers using the **NameTD** operation. This operation encrypts each component of a content name into a trapdoor using the client key. The encryption process for content data (**ContentTD**) and content name (**NameTD**) is different because of the requirement for decryption previously described. While data encrypted with **ContentTD** can be recovered by decryption, data encrypted with **NameTD** cannot. Content providers store content data and content names in their encrypted form as produced by **ContentTD** and **NameTD**. These are further transmitted to adjacent ICN edge nodes to announce their availability and location in the network.

ICN clients generate their interest for a content name using **NameTD**. This operation is performed before Step 1 in Figure 3 and the result is sent to an adjacent ICN node. The output of

NameTD is confidential because encrypted with a client key that only client i holds.

- **ContentTD**($D, K_{client,i}$)**:** A content provider i encrypts a content data element D using proxy encryption and its client key $K_{client,i} = (X_{client,i}, s)$. For each data element D, we choose a random number $r \in \mathbb{Z}_q^*$. Next, we compute $PE_i^*(D) = (\hat{e}_1, \hat{e}_2)$, where $\hat{e}_1 = g^r$ and $\hat{e}_2 = g^{rX_{client,i}}D$. $PE_i^*(D)$ is sent to the ICN edge nodes adjacent to i.
- **NameTD**($e, K_{client,i}$)**:** A content consumer (respectively content provider) i requests (respectively announces) a content name e to the network by generating a trapdoor. The trapdoor is encrypted using its client key $K_{client,i} = (X_{client,i}, s)$. e is composed of several components c_j. The client computes $\sigma = f_s(c_j)$ for each component c_j. It generates $TD_i^*(c_j) = (\hat{t_1}, \hat{t_2})$, where $\hat{t_1} = g^\sigma$ and $\hat{t_2} = \hat{t_1}^{X_{client,i}} = g^{\sigma X_{client,i}}$. **NameTD** returns $TD_i^*(e)$ being the concatenation of all $TD_i^*(c_j)$.

It is worth noting that **NameTD** is based on ElGamal encryption system with two minor changes. First, as we do not require to decrypt content names, we substitute D (data to encrypt) by the constant 1. Second, the random number r is replaced by the output of the pseudorandom function f_s applied to c_i. f_s is common to all clients and content providers.

5.3. ICN Re-Encryption

ICN edge nodes receive content data and content names encrypted with client keys $K_{client,i}$. In order to perform lookup operations in routing tables and deliver content data that can be decrypted by other clients $j \neq i$, these cyphertexts must be transformed. ICN edge nodes re-encrypt these cyphertexts into ciphertexts as they would be if encrypted using the master key X_{MK}, but without explicitly using it. Since the sum of the client i key and its associated proxy key are equal to the master key, the successive application of trapdoors with the client i key and its associated proxy key produces a ciphertext that can be decrypted with the master key X_{MK}. Encrypted content data $PE_i^*(D)$ are re-encrypted using **ICN-ContentTD** operation. Encrypted content names $TD_i^*(e)$ are re-encrypted using **ICN-NameTD**.

- **ICN-ContentTD**($j, PE_i^*(D)$): The ICN edge node retrieves the proxy key $K_{proxy,i}$ corresponding to content consumer i as initially provided by the KMS. It computes $(\hat{e}_1)^{X_{proxy,i}} \cdot \hat{e}_2 = (g^r)^{X_{proxy,i}} \cdot g^{rX_{client,i}}D = g^{rX_{client,i}+rX_{proxy,i}}D = g^{rX_{MK}}D$. This gives $PE(D) = (e_1, e_2)$ with $e_1 = g^r$ and $e_2 = g^{rX_{MK}}D$. $PE(D)$ is equal to the results of ElGamal encryption applied on D with the master key X_{MK}.
- **ICN-NameTD**($i, TD_i^*(e)$): The ICN edge node retrieves the proxy key $K_{proxy,i}$ corresponding to content consumer i. It computes: $TD(e) = T = \hat{t_1}^{X_{proxy,i}} \times \hat{t_2} = g^{X_{MK}\sigma}$. Note that the multiplication is performed over modular exponentiation. **ICN-NameTD** is applied to every pair $TD_i^*(c_i)$ corresponding to every component of e. $TD(e)$ is equal to the results of our modified ElGamal encryption scheme (**NameTD**) applied on e with the master key X_{MK}.

Once a content data and its content name are announced by a content provider and both undergo ICN re-encryption, the ICN edge node triggers a routing algorithm that will populate FIB tables [28, 29] and possibly cache the encrypted content in some nodes. Thus contents and content names are stored in ICN nodes (CS, PIT and FIB) in the same encrypted form, *i.e.,* as if encrypted with the master key X_{MK}, regardless of the client that initially generated them. Since only the KMS holds this key, no node or client can decrypt and access content names or contents in clear.

5.4. ICN Lookup

After the execution of **ICN-NameTD**, content names have been converted into ciphertext as if encrypted using the master key. Also, names populating the CS, PIT and FIB of ICN nodes are encrypted in the same manner. These tables must support two types of operations: the exact match and the longest prefix match (cf. Section 2.1). **MatchTDs** compares two encrypted ciphertexts in order to route interest and content messages in the network. To execute the longest prefix match or exact match, **MatchTDs** is executed several times with the different components c_j of the content name e.

- **MatchTDs**($TD(e_1), TD(e_2)$): The ICN node performs a simple equality match $T_1 = T_2$ that returns *true* in case of a match and *false* otherwise.

It is worth noting that each client has its own key for encrypting the request by generating a client-encrypted trapdoor. The second round of encryption by the ICN node using the proxy key re-encrypts the client-generated trapdoors. This results in trapdoors that eventually get encrypted under MK, which is revealed neither to the ICN nodes nor to the clients.

5.5. Content Retrieval

If a requested content is already cached in an ICN node, the retrieval is straightforward since it is already referenced (content name) and stored (content data) as encrypted with X_{MK}. However, if the interest message reaches the content provider, the content name is encrypted in the form $T = g^{X_{MK}\sigma}$, which is not understandable by the latter.

In ICN, when a content provider starts serving a content name, a link is created between the content provider and the ICN edge node. This link is implemented as an interface in the ICN edge node and another in the content provider. Only one content name is associated with one interface and if multiple contents are served, multiple interfaces are created. When an edge node receives a request for a content name, it knows which interface from the content provider serves it. Thus, there is no need for the content provider to understand the encrypted content name and only the ICN edge node has to request the right interface according to its mapping of the encrypted content name.

5.6. *ICN Pre-Decryption*

Once a content is found in the network, a data message follows the reverse path taken by its corresponding interest message. The payload of the data message is encrypted as a result of **ICN-ContentTD**$= PE(D)$. The message eventually reaches the ICN edge node directly connected to the content consumer j, which runs **ICN-ContentDec** to pre-decrypt the content data using the proxy key of j. It is worth mentioning that we only need to decrypt the payload of the content data message while the content name remains encrypted.

- **ICN-ContentDec**$(j, PE(D))$: The ICN edge node retrieves the proxy key $K_{proxy,j}$ corresponding to content consumer j. The ciphertext $PE(D)$ is decrypted as $e_2 \cdot (e_1)^{-X_{proxy,j}} = g^{rX_{MK}}D \cdot (g^r)^{-X_{proxy,j}} = g^{r(X_{MK}-X_{proxy,j})}D = g^{rX_{client,j}}D$. The ICN node sends $PE_j^*(D) = (\hat{e}_1, \hat{e}_2)$ to the requesting content consumer j, where $\hat{e}_1 = g^r$ and $\hat{e}_2 = g^{rX_{client,j}}D$.

5.7. *Client Decryption*

The content consumer j receives the data message containing the payload pre-decrypted by an ICN edge node: $PE_j^*(D)$. The content consumer decrypts it running **ContentDec** with its client key $K_{client,j}$. Content consumer j is the only one capable of decrypting the message because it has been pre-decrypted with its proxy key $K_{proxy,j}$. This step ends the content retrieval procedure without the content consumer revealing its requested content name nor the content itself to any ICN node.

- **ContentDec**$(PE_j^*(D), K_{client,j})$: The content consumer decrypts the ciphertext as $\hat{e}_2 \cdot (\hat{e}_1)^{-X_{client,j}} = g^{rX_{client,j}}D \cdot (g^r)^{-X_{client,j}} = D$.

5.8. *Revocation*

A client key may be compromised and need to be removed from the system. This operation is managed by the KMS that can revoke the key of any given client i. A list of revoked proxy keys is generated and broadcast (as a Certificate Revocation List - CRL) on the ICN network on a regular basis (*e.g.,* once per hour) for ICN nodes to run **Revoke** on each component of the list. Proxy keys of revoked clients are deleted from any node on the ICN network. No ICN node will re-encrypt/pre-decrypt any message from/to a revoked client after this operation.

- **Revoke**$(K_{proxy,i})$: Client i is revoked by removing its proxy key $K_{proxy,i}$ from the key store of any ICN nodes.

This section introduced the several operations needed to implement *PrivICN*. It is worth noting that initialization of the system and key generation for each client are independent operations. The latter is lightweight and only needs to generate two keys (client + proxy) for each new user. This is a one-time operation and the number of clients in the system does not have an impact on its running operations albeit the KMS stores all generated client/proxy key pairs. Similarly, the key revocation process consists in updating a CRL that is distributed at regular time interval. This does not depend either on the number of revoked clients. The addition/removal of users does not require any modifications to the encrypted content and names stored in the network. ICN core nodes can be replaced without any cost. New ICN edge nodes must request server keys from clients they serve to the KMS, as they receive interests from these clients. Thus, we can conclude that *PrivICN* meets the design goal **G2** providing easy and flexible user management for large number of users.

6. Evaluation

In this section, we evaluate the performance of *PrivICN*. First, we evaluate the computational cost and time required for each operation defined by *PrivICN* (Section 6.2). Some operations are one time operations while others need to be repeated. We seek to identify bottleneck operations and evaluate if *PrivICN* is theoretically applicable without incurring too high delay in communications and computational cost to ICN participants. Second, we evaluate the storage overhead in ICN routing tables and the computational overhead for performing routing operations on encrypted content names (Section 6.3). This set of experiments highlights the storage capabilities required by ICN nodes to implement *PrivICN* as well as the overhead on routing operations compared to using non-encrypted content names. Third, we evaluate the actual performance of *PrivICN* when deployed in a real ICN network, *i.e.,* CCNx implementation (Section 6.4). This experiment compares the theoretical performance results obtained in Section 6.2 to real world performance results when *PrivICN* is deployed. It also presents the overall time overhead generated by *PrivICN* when considering a real communication delay, and to compare it to a plain (non-confidential) CCNx implementation. Finally, Section 6.5 evaluates the size overhead for content names and content data with respect to the size of an ICN message. This shows the reduction in usable space for actual payload in ICN messages while using *PrivICN*. All experiments consider different level of provided security as represented by the size of keys k (in bits) used for encryption, *i.e.,* $k \in \{512, 1024, 2048\}$.

6.1. *Implementation Details*

The implementation prototype of *PrivICN* is developed in C++ programming language. The base ICN implementation we chose is CCNx from PARC [30]. All cryptographic operations are computed using the OpenSSL library. *PrivICN* is packaged into the *libprotector* library, which is publicly available [19]. It exposes wrappers for C and Python. It consists of three components: a module for the Key Management System (*KMS-module*), a module for ICN nodes (*node-module*) and a module for clients (*client-module*).

- *KMS-module* is responsible for initialization of the system (**Init**) and key generation (client key & proxy key) for each client (**KeyGen**). It also manages key distribution and generates the client revocation list.
- *Node-module* is responsible for content data and content name re-encryption (**ICN-ContentTD** & **ICN-NameTD**), for content data pre-decryption (**ICN-ContentDec**) and

for table lookup (**MatchTDs**). It also implements the **Revoke** function.

- *Client-module* is responsible for content data and interest generation (**ContentTD** & **NameTD**) and for decrypting contents (**ContentDec**).

6.2. Basic Operations Performance

We evaluate each operation presented in Section 5 individually in order to identify potential bottleneck operations and compute a theoretical overhead for *PrivICN*. Each module is simulated on a standard notebook with 2.7 GHz processor and 8.0 GB RAM. Results constitute averages over 10,000 independent executions. Table 1 summarizes the average time of each evaluated operation. Comprehensive experiments results are also available in the library [19].

We see that the key size increases exponentially the time of each operation except **MatchTDs**, which takes quite steadily $0.3\mu s$ either without using *PrivICN* or using it with a key of 512-bit or 1024-bit key. There is a small increase to $0.43\mu s$ when using a 2048-bit key though. **MatchTDs** is a mandatory operation for both a normal ICN network and one using *PrivICN*. It is performed several times per lookup on each ICN node. Experiments show that performing this operation on encrypted content names has a very low absolute overhead ($< 0.15\mu s$) compared to non-encrypted names.

On the other hand, all other operations are specific to *PrivICN*. The first three operations in the table are performed offline and only triggered when a new client join the network (**KeyGen**) or when a content is first published (**ContentTD**, **ICN-ContentTD**). These initial operations take very little time (<7 ms) regardless of the key size and thus would not impact the original performance of an ICN network.

The four next operations in Table 1 constitute the *static overhead*. These are operations that are performed for any content request and delivery, regardless of the location of a content to retrieve in the network. *Static Overhead (SO)* sums up this overhead as defined in Equation (1) with $t(x)$ being the computation time for the operation x. *SO* is between $1.2ms$ for a 512-bit key and around $40ms$ for a 2048-bit key.

$$\begin{aligned} SO = t(NameTD) + t(ICN\text{-}NameTD) \\ + t(ICN\text{-}ContentDec) + t(Content\text{-}Dec) \end{aligned} \qquad (1)$$

It is worth noting that **NameTD** and **ICN-NameTD** are computed on content names of diverse length (number of components). These are randomly picked from a public sample of ICN names containing 13 million content requests [31]. Content names in this file are composed of five components on average. Figure 5 shows the computation time for **NameTD** depending on the number of components c in a content name. Computation time is averaged over 20 runs using content names randomly selected from [31]. This time increases linearly with the number of components in the content name as each component is encrypted individually. The same observation applies to **ICN-NameTD** that takes a similar time.

	ICN	ICN + *PrivICN*		
Keysize	–	512-bit	1024-bit	2048-bit
KeyGen	–	$0.10ms \pm 0.02$	$0.46ms \pm 0.03$	$2.9ms \pm 0.1$
ContentTD	–	$0.17ms \pm 0.01$	$0.90ms \pm 0.04$	$6.2ms \pm 0.1$
ICN-ContentTD	–	$0.09ms \pm 0.01$	$0.47ms \pm 0.02$	$3.3ms \pm 0.1$
MatchTDs	$0.28\mu s$	$0.28\mu s$	$0.29\mu s$	$0.43\mu s$
NameTD	–	$0.48ms \pm 0.19$	$2.39ms \pm 0.09$	$16.3ms \pm 6.2$
ICN-NameTD	–	$0.41ms \pm 0.16$	$2.26ms \pm 0.09$	$15.9ms \pm 6.0$
ICN-ContentDec	–	$0.14ms \pm 0.01$	$0.61ms \pm 0.03$	$3.8ms \pm 0.1$
Content-Dec	–	$0.14ms \pm 0.01$	$0.61ms \pm 0.03$	$3.8ms \pm 0.1$
Static Overhead	–	$1.2ms \pm 0.2$	$6.0ms \pm 0.1$	$39.8ms \pm 6.1$

Table 1: Time for *PrivICN* operations using different encryption key sizes. Mean time (± standard deviation) obtained over 10,000 independent executions.

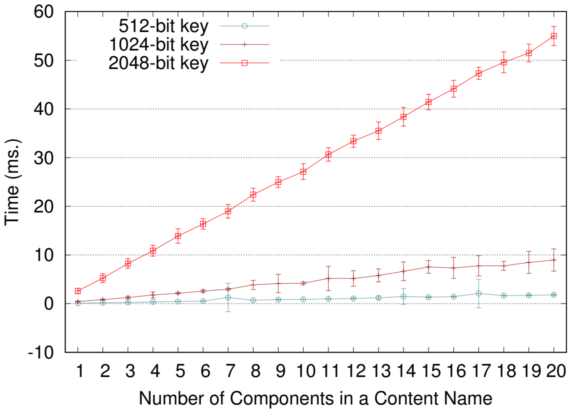


Figure 5: Increase in computation time for **NameTD** operation as the number of components in a content name increases. Mean time and standard deviation for 20 runs using random seed names [31].

If we consider the time consumed by **ICN-NameTD**, respectively **ICN-ContentDec**, they determine the throughput of an ICN edge node in term of interest messages, respectively data messages, it can process per time unit. Table 2 reports this throughput for different key sizes. We see that a short key (512-bit) allows an ICN edge node to process thousands of interest and data messages per second. The throughput is lower using a 2048-bit key with 62 processed interests per second and 263 data messages. This asymmetry is expected since content names require one encryption per component while a data message is encrypted at once. However, it is not problematic since one interest messages is sent for one content data that is likely fragmented into several data messages.

PrivICN presents an insignificant overhead of $0.43\mu s$ for name comparisons (**MatchTDs**). **MatchTDs** is the only *PrivICN*'s operation that is repeated for every hop of a packet in the network. Since this operation is very fast, we can expect a good scalability of *PrivICN* to large networks counting a high number of nodes. Its static overhead is lower than 40 ms with a key size of 2048-bit that guarantees the security of the system. Similarly, ICN edge nodes keep a satisfactory throughput for interest and data messages using any size of key. Thus, *PrivICN* shows a very low theoretical overhead that would ensure its deployability in an ICN network.

Key size	Interest message throughput	Data message throughput
512-bit	$\frac{1}{4.1\times10^{-4}} = 2439$ msg./s	$\frac{1}{1.4\times10^{-4}} = 7142$ msg./s
1024-bit	$\frac{1}{2.3\times10^{-3}} = 434$ msg./s	$\frac{1}{6.1\times10^{-4}} = 1639$ msg./s
2048-bit	$\frac{1}{1.6\times10^{-2}} = 62$ msg./s	$\frac{1}{3.8\times10^{-3}} = 263$ msg./s

Table 2: Interest and data messages throughput for an ICN edge node using *PrivICN*.

6.3. Lookup Time and Routing Table Size

We implemented FIB, PIT and CS on ICN nodes as hash tables. Hash table is a common data structure that meet the requirements specified by the ICN specification [32]. It is chosen because it has a static lookup time that does not depend on the number of entries in the table ($O(1)$ complexity). Since matching operations are similar for all of them, we only evaluate lookup performance and size for the FIB table. We used ICN content names randomly selected from [31] to fill the FIB table in experiment. Every experiment was repeated 20 times to obtain average lookup time and FIB table size.

Lookup in FIB table consists in the longest prefix match. We implemented it in the following manner: The complete encrypted content name $/TD(c_1)/TD(c_2)/.../TD(c_n)$ is first searched for in the FIB table. If no match is found, the last component $TD(c_n)$ is removed and the remaining prefix $/TD(c_1)/.../TD(c_{n-1})$ is searched for in the FIB table. This last component removal and search process continues until we find a match in the table.

Figure 6 depicts the time taken for lookup in FIB tables having an increasing number of entries. We see that lookup time is constant overall regardless of the size of the FIB table We actually observe that this time decreases as the size of the table increases. This is because with more entries in the FIB table, we get a higher probability to get a match for a long name prefix, reducing the number of comparisons required with **MatchTDs**. For example, assume we search for the content name $/org/wikipedia/Maradona$. First, we compute the hash of the full content name and look for a match into the FIB table. Remember that this matching operation has a constant time ($O(1)$) regardless of the FIB table size since we implemented the FIB, PIT and CS tables as hash tables. If it is not found, we continue by removing subsequently the latest component and do a longest prefix match again. As the number of entries in the FIB table increases, the probability of finding a longest prefix match becomes higher. This reduces the number of matching operation required and consequently reduces the overall lookup time.

In contrast to results observed in Table 1 for **MatchTDs**, Figure 6 shows a larger difference in the overall lookup time between the different key sizes. We can see an 8-fold difference between the time taken by *PrivICN* with a 2048-bit key compared to not using it. Nevertheless, lookup time remains lower than 10μs with *PrivICN* allowing an ICN node to process over 100,000 lookups per second.

Figure 7 depicts the increase in FIB table size (MB) according to the number of entries. There is a significant storage overhead when using *PrivICN*. The difference between a non-encrypted FIB table (0.8MB) and one with name encrypted with a 2048-bit key (42MB) is 50-fold. The use of shorter key length reduces this overhead: 1024-bit → 21MB, 512-bit → 10.7MB. *PrivICN* has a low overhead in lookup time but induces a significant overhead in table size. However, we can assume an ICN node can afford to store routing tables of a few tens of MB, ensuring the deployability of *PrivICN*.

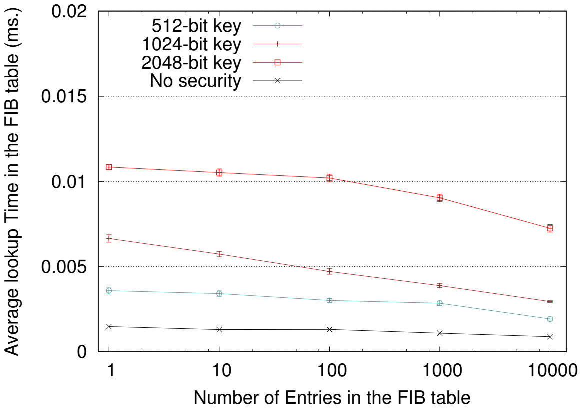


Figure 6: Lookup time in FIB tables according to the number of entries in the table. Mean time and standard deviation for 20 lookup on FIB tables filled with randomly selected names [31].

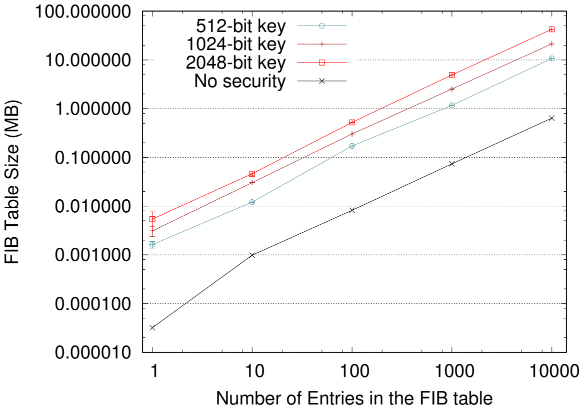


Figure 7: Increase in FIB table size as the number of entries in it increases for different key size. Average size and standard deviation computed over 20 FIB tables filled with randomly selected content names [31].

6.4. Network Performance

To evaluate the actual performance of *PrivICN* when deployed in a real ICN network, we built a CCNx network. We seek to evaluate the overhead for the retrieval of a content that fits (encrypted) into a single data message. We do not evaluate the impact of fragmentation in which a content is divided into several data message. Our CCNx network is depicted in Figure 8 and consists in a linear topology composed of one client, $n \in \{1,2,3,4\}$ ICN nodes and one content provider. We selected a linear topology because we want to measure the time to download a content data as the number of hops to reach it increases. This topology maximizes the number of hops for a message to be exchanged between a client and a provider.

The ICN nodes were implemented in virtual machine placed in the following locations: 1. Innsbruck (Austria); 2. Nancy (France); 3. New York City (USA); 4. Buenos Aires (Argentina). Every virtual machine ran in Virtual Box with 1 processor and 256MB of RAM. The client was always located in Innsbruck (Austria). The content provider was located at the same location as the last ICN node of the n-node topology. E.g., for a 2-node topology, the content provider is in Nancy (France).

Our CCNx implementation is based on the HelloWorld package from PARC [30] and our *PrivICN* library [19].

- *Content Consumer* is implemented based on the HelloWorld_Consumer application from PARC. It was modified to implement the *client-module* functionality.
- *CCNx Nodes* implement a CCNx prototype (Athena) from PARC. It was modified to support the *node-module*. The caching capabilities of Athena were disabled to force every request to reach the ICN edge node connected to the provider.
- *Content Provider* is implemented based on the HelloWorld_Producer application from PARC. It was modified to implement the *client-module* functionality.

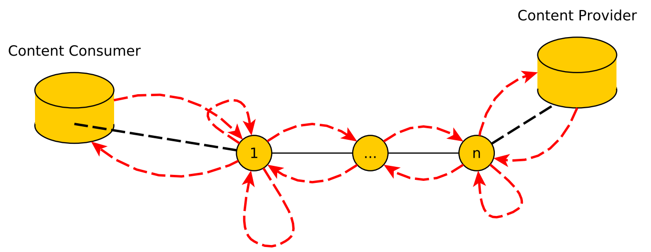


Figure 8: n-nodes topology used for experiments.

The scenario for the experiment is the following. The content consumer requests a list of 10,000 randomly chosen content names from a sample name file [31]. The matching contents are provided by the content provider located on the last node of the topology. No content is cached in ICN nodes. Contents have a small size and fit into a single data message. We measure the download time, from interest generation to content decryption, observed by the client for each request it makes.

Figure 9 depicts the average download time we measured empirically on several n-nodes topology ($n \in \{1,2,3,4\}$) and using different key sizes. It also depicts the theoretical download time (dashed line) we expected according to performance evaluation obtained in Section 6.2. The theoretical download time consists of the time of the required *PrivICN* operations plus a *Communication Delay (CD)*. Equation (2) defines this theoretical download time depending on the key size used (key) and the number of ICN nodes n to cross. It consists of the static overhead (SO), two lookups per node (one for interest message forwarding and one for data message routing), except the last node that performs only one lookup, and twice a communication delay (CD) between each ICN node ($n-1$). SO_{key} and $t(Lookup)_{key}$ are obtained from experimental results of Sections 6.2 and 6.3, respectively. We selected CD as the average of all communication delays observed between two successive ICN nodes during the empirical evaluation. We have $CD = 97ms$.

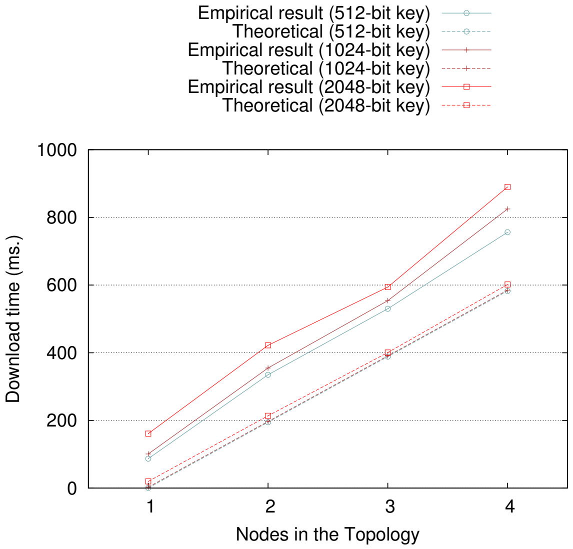


Figure 9: Theoretical and empirical download time using *PrivICN* in an n-node ICN network.

$$\begin{aligned} DownloadTime_{key}(n) = SO_{key} + (2n-1) \times t(Lookup_{key}) \\ +2(n-1) \times CD \end{aligned} \qquad (2)$$

Figure 9 shows that both theoretically and empirically the download time increases linearly with the number of nodes in the topology. We observe a significant difference between theoretical and actual download time though. This difference is present from a small topology with one node but increases only slowly as the topology grows. Thus, we can deduce that this difference is mostly generated by the static overhead (SO) of *PrivICN* that happens to be around $100ms$ higher in practice than in theory. Since this difference grows slowly as we add more nodes to the topology, we can conclude that the actual lookup time is also higher than the theoretical lookup time. This absolute difference in SO and $t(Lookup)$ is explained by the fact that we used virtual machines with far less resources during this experiment (256MB of RAM) than in Sections 6.2 and 6.3 (8GB of RAM). However, the linear growth in download time is mostly produced by the communication delay (CD) rather than *PrivICN*.

Figure 10 shows the ratio of delay due to *PrivICN* with respect to the overall download time: $\frac{DownloadTime - CD}{DownloadTime}$. It decreases as the size of the topology grows, starting from around 80% for a 1-node topology to reach less than 35% for a 4-node topology using any key size. Following this trend, we see that the delay generated by *PrivICN* becomes proportionally lower (<30%) with respect to the overall download time

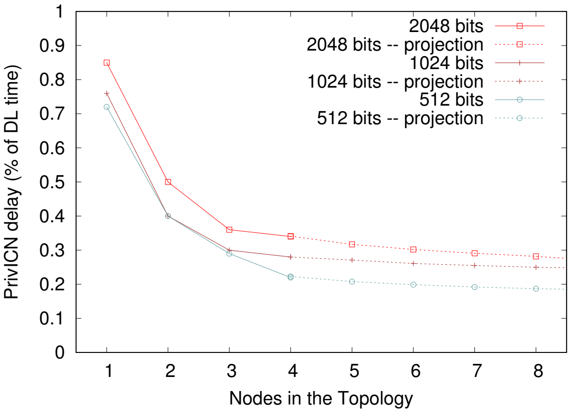


Figure 10: Ratio of delay due to *PrivICN* with respect to the overall download time. Values for 1-4 nodes topology are empirically measured, values for 5-8 nodes topology are theoretical projections.

as the network grows. This experiment shows that *PrivICN* induces an acceptable delay that proportionally decreases as the network grows. Consequently, we conclude that *PrivICN* has good performance and a acceptable delay showing that we meet the design goal **G3**.

6.5. Content Names / Content Size and MTU

The encryption of content names and content data increases the size of ICN messages. We evaluate the overhead of *PrivICN* in term of message size and its impact on the length of content names that can be requested.

An interest message is composed by a content name and a header. A data message is composed by a content name, a header and a payload. The ICN header has a fixed size that we assume to be 20 bytes (as for the Internet Protocol) and it is not affected by *PrivICN*. However, each encrypted content name component has a fixed size independent from its original size and equal to the key size: $\frac{keysize}{8}$ Bytes. Encrypted components are concatenated and separated by a 1-byte encoded character ($\#Components - 1$ separators) before being added to an ICN message. Equation (3) gives the size of a *PrivICN Content Name (PCN)*.

$$PCN\ (\text{Bytes}) = (\frac{keysize}{8} + 1) \times \#Components - 1 \qquad (3)$$

This size of interest and data messages must remain smaller than the Maximum Transfer Unit (MTU) value defined for ICN packets. Thus, the number of components in content names supported by *PrivICN* depends on the MTU and the encryption key size. This determines the space available for payload in data messages. CCNx defines an MTU of 8192 Bytes.

Figure 11 depicts the percentage of MTU available to deliver the payload of a requested content composed of n components. The point where the curves reach 0 gives the maximum length of a content name (number of components) that can be sent in an interest message (empty payload). We see that *PrivICN* allows one to request long content name of over 30 components for any considered key size. By requesting content names of average length (5 components) we see that from 84% to 96% of the packet size is dedicated to the payload, which is reasonably high.

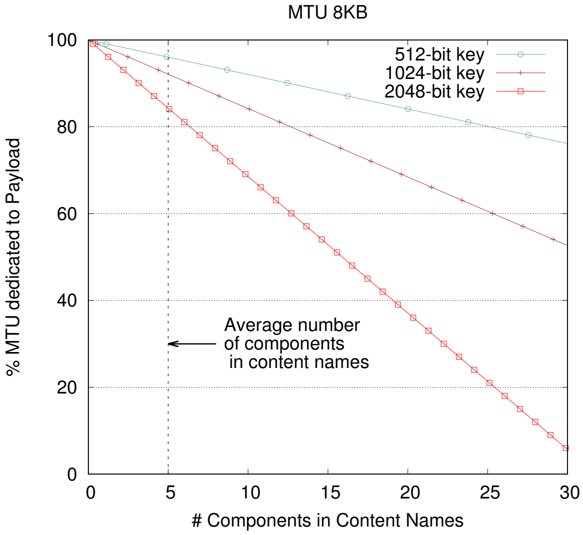


Figure 11: Maximum size of transported payload (in percentage of MTU) with respect to name length.

It is worth noting that encryption has also an impact on the payload size. *PrivICN* uses a cipher that allows to store a maximum number of bytes per block (i.e. 2048-bit can store 256 bytes, 1024-bit: 128 bytes, 512-bit: 64 bytes, etc.). For instance, when we transmit a large content, the content is not transmitted only in packets but is also divided into several blocks. After applying ICN-ContentTD, every block will generate two encrypted ciphertexts of $\frac{keysize}{8} Bytes \times 2$ by construction. For instance, for a 1MB file encrypted with our scheme and a 2048-bit keysize, we will require 4 MB to transmit the content. Thus *PrivICN* requires four times as many messages as a normal ICN network for content delivery. This overhead is mostly due to payload encryption that is not implemented by default in ICN: data is sent in plain text. However, traffic encryption is becoming a requirement to protect communications and is implemented in most protocols at different layer. Thus, we would assume that it may also become mandatory in ICN and the overhead we observe would apply to any ICN traffic.

The same overhead applies to any content in the ICN network, e.g., to content stored in caches. Content data cached in the network are also encrypted and require four times the space that they would require if stored in clear text.

7. Discussion

In this section we analyze the extent to which we meet the design goals formulated in Section 2.3 and discuss the potential limitations of *PrivICN*.

7.1. Meeting Design Goals

G1 - Confidentiality of information. Both contents and content names are encrypted in *PrivICN* as introduced in Section 4. No ICN node has access to this information. Only the client making the request and the potential content provider (if the content is not cached in the network) can know the subject of an interest. From a content provider perspective, there is no linkability between an interest and its author. Hence, the information about a requested content cannot be exploited and users privacy is guaranteed. From the ICN node perspective, confidentiality of content data and content names does not guarantee full users privacy. A curious ICN node could still perform dictionary attacks [33] (as done for password hashes) by encrypting some selected content names and comparing them to received encrypted interests. This attack is less scalable and has a high computational cost for the adversary, which would decrease the performance of the ICN node in content retrieval. We showed in Section 6 that *PrivICN* can efficiently use 2048-bit keys that meets the current minimal security recommendations for encryption [34]. Thus, *PrivICN* preserves the confidentiality of contents and content names, which enhances users privacy.

G2 - Ease of management. The whole management of *PrivICN* is centralized in the KMS. It performs addition, via key generation and distribution, and removal, via revocation list generation and distribution, of users from the system (cf. Section 5). These operations are independent from the current number of users in the system and do not induce any overhead at run time. The encrypted contents and names present in ICN nodes do not require any modification and remain encrypted in the master key space when adding/removing users. The cost of user key generation has been evaluated to be a few milliseconds in Section 6.2. In addition, ICN core nodes can be replaced without any cost in case of failure. When replaced, ICN edge nodes must request server keys to the KMS as they receive interests from new clients. Thus, *PrivICN* provides an easy management for a large and varying number of users and any modification is made at a low cost.

G3 - Performance & overhead. We have demonstrated in **G2** that *PrivICN* can manage a large number of users. Section 6 showed that its operations generate little delay that decreases in proportion as the network grows. The time for content retrieval is dominated by the network delay and *PrivICN* generates only around 30% additional delay in a large enough network (cf. Section 6.4). The overhead in lookup time for ICN nodes is negligible (a few microseconds) and *PrivICN* provides a significant throughput for both interest and data messages (cf. Section 6.2). The most significant overhead of *PrivICN* is on the size of tables and exchanged contents. The size of FIB, PIT and CS tables can be increased by a factor of 50. The traffic generated to transfer a file increases linearly with the size to the encryption key. Nevertheless, the tables storage remains affordable for ICN nodes since they represent a few tens of MB at most. The real world deployment of *PrivICN* demonstrated on a small network in Section 6.4 that *PrivICN* is deployable and generate an acceptable overhead.

G4 - Preserve ICN features. *PrivICN* stores contents and names in a single encrypted form while in the network using proxy encryption methods. Thus, encrypted content can be stored in the network on any ICN node and served to any client that can decrypt them. Similarly names are encrypted in a single form component by component, which allows ICN nodes to perform longest prefix match operations for routing. Thus, *PrivICN* preserves the benefits of in-network caching and name-based routing. Also, contents can be retrieved in a secure manner from any node on the network and there is no need for end-to-end communication between clients, pre-determined ICN nodes or content providers. Finally, the normal CS-PIT-FIB workflow is preserved as described in Section 5. *PrivICN* preserves all ICN features.

G5 - No security association. Client and proxy keys are managed by the centralized KMS that provides them when a client join the network. This is a one time operation. From then on, a client use its client key to communicate securely with any node of the network. No security association with a specific node is required and no message is signed. Thus no end-to-end communication occurs between users, which preserves their anonymity.

7.2. Limitations

Proxy encryption and client / ICN node collusion. Content names and contents are in the from of cyphertext encrypted with the master key X_{MK} while traversing the network of ICN nodes (cf. Section 4). The master key is held by the trusted KMS only. However, getting a client key $X_{client,i}$ and the corresponding proxy key $X_{proxy,i}$, one can recover the master key $X_{MK} = X_{client,i} + X_{proxy,i}$. Hence, we must assure that the distribution of client keys is limited to ICN clients only and ICN nodes only have access to proxy keys. To avoid key leakage and having a single entity holding both a client and a proxy key, the key management system can provision keys in a secure manner using trusted hardware [35]. This ensures that each key provisioned by the KMS will be tied to a physical hardware component on a given client / ICN node. This key cannot be extracted from it or revealed to an attacker. Nevertheless, the collusion of one client and one ICN node breaks the security of the system as an attacker could decrypt both contents and content names if it controls both. We assumed ICN nodes to be honest but curious in our attack model (cf. Section 2.2). Thus this collusion should not happen since every message to decrypt would need extra communications between the colluding client and node as client and proxy keys are tied to trusted hardware.

Centralized key management system. The use of a centralized key management system exposes a single point of failure for the system in case it gets compromised or is hosted by an untrusted entity. The master key can leak and be used to decrypt any contents and names forwarded on the network. However, several widely deployed systems, *e.g.,* credit cards, rely on such a centralized infrastructure and have proven to be secured. The delegation of key management to a trusted hardware entity [35] such as Intel SGX [36] can assure there is neither master key leakage nor misuse of it. We can constrain the usage of the master key to key generation only and prevent its usage for encryption/decryption. As previously presented the usage of trusted hardware on the KMS also guarantees the provisioning of client and proxy keys in a secure manner.

Inter-domain application. *PrivICN* is currently designed for intra-domain usage where a single KMS serves the whole domain. We gave the example of an ISP for such a domain where *PrivICN* can be applied (Section 4). *PrivICN* cannot be readily applied in inter-domain scenarios without breaking the confidentiality of content names and data. The ICN node at the junction of two domains would be required to decrypt and re-encrypt content names and data from domain 1 (master key X_{MK1}) to domain 2 (mater key X_{MK2}) to enable delivery of contents involving several domains. This *junction node* would have access to names and data in clear in the process. One solution to extend *PrivICN* for inter-domain applications is to enforce decryption and re-encryption to happen in a trusted execution environment (TEE) [35]. Content names and data would be decrypted using X_{MK1} and re-encrypted using X_{MK2} inside the TEE of the junction node without it being able to learn anything about the clear content data and names. TEEs become increasingly deployed and available as features for many processors, e.g., Intel SGX [36].

Client revocation delay. ICN nodes do not check every time they receive an interest if the client generating it is authorized to make a request *i.e.,* if its key is valid. This is a choice for sake of performance and scalability. The key revocation list is periodically generated and broadcast by the KMS (cf. Section 5). This process induces a delay before ICN nodes are notified about the revocation of one client. Meanwhile a revoked client can continue to participate in the ICN network. The interval of revocation list updates must be computed to provide a trade-off between responsiveness of the system and overhead of communications needed to broadcast the updates. The same problem is presented by widely deployed systems such as HTTPS and TLS for certificate revocation.

Impact of caches in ICN nodes. The ICN literature tries to address the problematic of caches in ICN. It is not clear yet the number of caches and their size required at every node of the network. With this limitation in mind, we have deployed experiments without in-network caching features enabled. Without caches, we have always worked with the worst-case scenario where in-network caching storage is be available to help reducing the download-time. We believe that *PrivICN* will benefit from the use of caches. We expect in the early future that measurements about number of caches and cache sizes will be available in the research literature (with respective tools) to address this point.

Space for content in data message. We have seen in Section 6.5 that encryption significantly increases the size of content names. This has limited effect on interest messages that do not contain payload. However, it significantly reduces the space for payload in data messages, increasing the number of required message to deliver a content. Currently, ICN couples name routing and data forwarding. Both interest and data messages contain the complete content name. Although we require each individual element of a content name for FIB longest prefix match, the PIT and CS table require exact content name match. Thus, a full content name can be encoded at once to be used for PIT and CS. This means that interest messages must contain the content name encrypted element by element as we defined it, since they are forwarded according to the FIB. However, data messages are routed according to the PIT and can contain a hash of the full content name. By using a hash, we can dedicate more space to the payload and use longer names in interest messages. Also the size of PIT and CS tables would be drastically reduced. However, it would add a small overhead for computing the hash of the content name contained in interest messages at each ICN nodes. Nevertheless, this solution modifies the original CCNx or NDN protocols and explains why we did not implement it.

7.3. Portability to other ICN architectures

PrivICN is designed for the most popular ICN architectures: CCN and NDN. We discuss in this section its applicability to other architectures.

7.3.1. Data Oriented Network Architecture (DONA)

In contrast to CCN or NDN, DONA does not store contents in the network. Content is only stored by content providers and clients, who can hold replicas of original contents. In DONA, a content is addressed by a hash computed on a set of keywords related to the content and the identity of its original content provider. There is no hierarchy in content names. Clients request a content using the hash that can be found in, *e.g.,* a search engine or a website. The request is sent to a *local resolve handler* (equivalent to an ICN edge node). The request is then forwarded until it reaches another local resolve handler connected to the content provider or a client holding a replica of the requested content. The last local resolve handler knows the source of the request and delivers the content found using IP.

PrivICN can provide confidentiality of content names to DONA. All requests and announcements use as a name a hash composed by the set of keywords and the identity of the content provider. *PrivICN* can provide confidentiality of these hashes, the DONA's content names by using **NameTD**. Then, local resolve handlers receiving these announcements and requests re-encrypt them using **ICN-NameTD**. DONA can operate the same normal routing operations on these encrypted names rather than on the original hashes. This would increase the size of names from 40 Bytes hashes to fix-length content names (*e.g.,* 256 Bytes encrypted names for a 2048-bit key – content names are not decomposed in components DONA). Content names would be confidential since in contrast to hashing, this encryption cannot be performed by a single node/client alone and needs the collaboration of one client and one local resolve handler.

Contents are delivered over IP. The content provider or the client holding a replica can know the destination of the requested content through the local resolve handler. Thus, contents can be sent encrypted using, *e.g.,* a TLS session that guaranties end-to-end confidential communications between clients and content providers.

7.3.2. Publish Subscribe Internet Technology (PURSUIT)

PURSUIT proposes an architecture based on a publish/subscribe overlay. Content providers announce their content to a *rendez-vous* node (equivalent to an ICN edge node) using a hash computed on a list of keywords. The rendez-vous node is in charge of announcing the presence of the new content inside the network. Clients wishing to download a content, subscribe to it through their nearest rendez-vous node. They use the hash of keywords to do so. The rendez-vous node is in charge of finding the path to the nearest replica of the requested content and provides a way to send the data back to the client.

For names, PURSUIT can use *PrivICN* the same way as for DONA. Clients and content providers request and announce content names encrypted using **NameTD** instead of hashes. The rendez-vous node acts as an ICN edge node and re-encrypt them using **ICN-NameTD**.

When a content is requested, the rendez-vous node contacts the nearest content provider and sends the path to deliver the content. The content provider encrypts the content the same way as in *PrivICN*'s content generation using **ContentTD** . The path to deliver the content starts by another rendez-vous node that re-encrypts the content with **ICN-ContentTD**. Successive rendez-vous nodes perform the ICN lookup operations on encrypted names until they reach the last rendez-vous node. This rendez-vous node pre-decrypts the content and sends it to the client that can decrypt it.

PrivICN can be applied to other ICN architecture than CCN and NDN as we showed. This only require small adaptation and modifications of the considered architectures.

8. Related Work

Although Information-Centric Networking addresses existing problems of the current Internet, it introduces an array of new privacy threats. These include cache privacy, content privacy, name privacy and signature privacy [9, 37, 38]. *PrivICN* addresses the privacy of content data and content names by providing confidentiality for both. We first discuss proposals that provide content data and content names confidentiality independently and then jointly as we do.

8.1. Content Data Confidentiality

The protection of contents in ICN can be accomplished using asymmetric encryption. An obvious solution is to use a similar mechanism as TLS. Every client generates a session key and encrypts it using the content provider public key. All further communications between the client and content provider are encrypted using the session key. Along this line, a session-based access control mechanism (SAC) for ICN has been introduced [16]. In SAC, a session key is negotiated between any client and provider and used for specific communications between them during the session. As such, same content data is encrypted with different session keys before being sent in the network. Similarly Chen *et al.* [15] introduce an access control model where clients and content providers each hold a public/private key-pair. Every time a content consumer requests a content name, the provider sends an encrypted version of the content data specifically created for the content consumer. Once the encrypted content data is sent, the content consumer negotiates a symmetric key with the provider to decrypt the received content data. Although these solutions protect the confidentiality of content data, they have major drawbacks. First, all content requests must reach the content provider and require to build an end-to-end secure channel between content consumer and content provider. Second, content data is encrypted with different keys for each consumer and session. Thus an encrypted content data cannot be served to different consumers and there is no need to cache it in the network loosing the benefit of in-network caching, which is one of the main ICN feature.

An alternative to asymetric encryption is broadcast encryption. It allows a content provider to generate an encrypted content data that can be decrypted by a set of clients [39]. A group of consumers interested in a same content data pre-calculate a common key, based on information given by a KMS. This key will be used by content providers and the group of clients to exchange requests and contents. This solution is proposed in [9] without providing any implementation or assessment though. While a content data can be encrypted and serve a group of consumers, this scheme suffers from the same shortcomings as the previous one. There will be several groups of consumers with different keys, prohibiting in-network caching of contents in a single encrypted format and producing a large caching overhead. Most interests would likely reach the content provider and would not be resolved by ICN nodes. Furthermore, content providers each requires a long list of group keys to serve any requesting group.

Wood *et al.* [40] present two alternatives to provide content data confidentiality in ICN. A first solution relies on El-Gamal and Schnorr's signature and a second one on Identity-Based Encryption (IBE) [41] . Fotiou *et al.* [42] also propose a scheme based on IBE and assume that every content consumer has an identification in the network. IBE schemes allow to encrypt content data by using the public parameters of a content provider. Intermediate consumers receiving the content data can re-encode it such that other clients can decrypt it by negotiating a key with the original content provider. These solutions present nevertheless several drawbacks since a secure channel between the content consumer and the content provider must be created in order to retrieve the identity of content providers. Such end-to-end tunnel is against the principle of name-based routing, which is one of the most interesting features of ICN. Moreover, any user can sign messages with the identity of the content provider. Any available content data in the network could be provided by any other node in the network. Finally, if the private key of any content provider of the network is compromised, every content data encrypted with this key could be decrypted.

In contrast to these solutions, *PrivICN* does not need end-to-end communications between consumers and providers, it allows in-network caching with content data and content names encrypted in a single space. In addition, compromised keys can be revoked and client keys or proxy keys alone are useless if only one gets compromised. Since both are hosted on different machines, this is less likely to happen, providing better security. Some previous work suggested, as we do, to protect the confidentiality of content data using proxy re-encryption. This solution was first proposed by Chabanee *et al.* [9]. Their proposal relies on generating a pair of public/private key for each content data in the network. When a content provider publishes a new content data, it creates a new key pair to encrypt specifically this new content data. This solution has two major drawbacks: 1) if different content providers provide the same content data, it will not have the same representation in the network, which prevents an optimal caching of contents. 2) The key distribution process is not discussed in the paper. Since one key pair is used for a unique content, witnessing which client retrieves which key leaks information about the content data it is looking for. In contrast, *PrivICN* preserves full in-network caching capabilities and it is not exposed to side channel attacks by monitoring the public keys retrieved by a content consumer. We also propose a solution for ensuring the confidentiality of both content names and content data while only the confidentiality of content is tackled using proxy re-encryption in [9]. Da Silva *et al.* also proposed a proxy re-encryption system to control access to content in ICN [43]. As for [9], the paper only discusses how such a system could be implemented in different ICN architectures. However, no specific cryptosystem nor any concrete system design and implementation are proposed. The paper only evaluates primitives for encryption and decryption at ICN edge nodes and user ends as well as the memory consumption of these operations. In contrast, *PrivICN* is publicly available as a library and it has been evaluated in a real CCNx deployment.

8.2. Content Name Confidentiality

The protection of content names has different requirements than contents, namely that protected content names must support prefix match operations but the protection does not need to be reversible *i.e.,* names can be encoded with one way functions. Considering this fact, Chaabane *et al.* [9] propose to use bloom filters to provide name confidentiality. Bloom filters are space-efficient data structures to test the membership of one element in a set. CS, PIT and FIB tables are implemented as bloom filters as well as the requested content name, which is a hierarchical bloom filter $HB = (B_1, B_2, ..., B_n)$ where B_i is a bloom filter made of the first i components of the name. The match operations are operated between bloom filters, which preserves name confidentiality. However, a malicious ICN node can generate bloom filters with one element and test incoming interests against it to discover what a requested content name is. Thus, this solution is vulnerable to attacks by curious ICN nodes in contrast to *PrivICN*.

Alternatively, names can be secured using onion routing [44] schemes similar to Tor [45]. Content names components are each encrypted using a different key and thus can only be decrypted by specific ICN nodes holding the corresponding key [10, 46]. When receiving an interest, an ICN node can decrypt only the first component of the content name, which unveils the next destination of the interest. This first component is removed by the ICN node and a new interest is generated, which requests the truncated content name. The process is repeated until the last component is reached and the content retrieved. Tourani *et al.* [10] propose such an anti-censorship mechanism based on Huffman coding. A content consumer generates an interest with content name components encoded using Huffman coding and to which it appends a pre-defined plaintext prefix such as */anonym*. The content consumer sends the interest that is routed according to the plaintext prefix */anonym* to a specific ICN node of the network called *anonymizer*. The anonymizer acts as a Tor exit node. It can only read the content name requested by the content consumer and will serve it from its CS, if cached, or request it in its name if not. This proposition presents several drawbacks. First, there is a need for extra communication to reach the anonymizer before being able to fetch any content even though the content data would be cached on the path to the anonymizer. Thus some main benefits of ICN, including the full exploitation of in-network caching, are lost. Second, there is a security association between the content consumer and the anonymizer, which induce an overhead. Finally, if the anonimyser is curious, it can infer consumers' interests, since interest are linked to their authors based on the security association that was performed. Tsudik *et al.* [47] proposed a similar scheme relying on a more efficient cryptographic construction. Although more efficient than previous solutions, the scheme presents the same issues as other Tor alike schemes.

A last solution to protect content names in ICN relies on homomorphic encryption as proposed in [11] for enhancing lookup privacy. The problem is tackled as Private Information Retrieval (PIR). The scheme provide satisfactory security and does not present similar flaws as other proposals. However, the major issue with PIR based solutions [48, 49, 50, 51] and Oblivious RAM (ORAM) [52, 53, 54] is the large computational and communication overhead they produce as well as their inability to scale to large deployment.

In contrast to Bloom filter based solutions [9] for name lookup or Tor based proposals [10, 46, 47], *PrivICN* provides better security since no ICN node can perform attacks to recover requested content names. Moreover, Tor based solutions require security association between client and nodes, give different roles to ICN nodes and reduce the performance of ICN preventing the optimal use of in-network caching. *PrivICN* does not present these drawbacks and preserves the full features of ICN. Finally, in contrast to the solution based on homomorphic encryption, *PrivICN* has a low overhead and it also protects content data.

8.3. Content and Name Confidentiality

The problem of protecting both content data and content names has been addressed in a limited manner in the literature. However, we explained in Section 2.2 that both are necessary to efficiently enhance users privacy.

One solution relies on Attribute Based Encryption (ABE) [55, 14]. Attribute Based Encryption (ABE) is a type of public-key encryption in which the decryption of a cyphertext depends on a set of content consumers attributes [56]. Ion *et al.* [55] propose a scheme in which content providers encrypt contents using an encryption key derived from name components and identifiers of consumers allowed to access the content. This way only authorized clients, and no ICN node, can decrypt a specific content data. ICN nodes route interests using lookup operations on encrypted content names. These can only be decrypted by the content provider because based encrypted using its public key.

An issue of this scheme is that lookup operation based on ABE is based on bilinear pairing, which induces a high computational overhead [57, 58]. A solution to improve the efficiency of the routing algorithm [14] uses ontologies that combine several encrypted attributes. Despite reducing the cost of lookup operations, they still rely on bilinear pairing and remain more costly than *PrivICN*. It is also worth noting that content consumer's revocation is complicated in ABE systems, since decryption keys depend on attributes and not on the content consumer. To revoke a content consumer, we must remove from the network all contents encrypted with its identity as possible recipient. We must also re-encrypt all these content data using a key that does not depend on its identity. *PrivICN* provides an easy and low overhead revocation procedure that is handled by the KMS and requires only to remove a specific proxy key from ICN nodes key store.

A solution was proposed to handle revocation in ABE based systems [13]. It uses a proxy server that holds a list of blacklisted consumers. All interest requests made on the network are treated by the proxy server which forward them to the network if emitted by authorized consumers and drop them otherwise. The mechanism keeps several drawbacks such as forcing all interest messages to go through a single entity: the proxy server. Moreover, it just improve the response time of the system to a client revocations and as for previously described schemes, if a consumer gets blacklisted all the contents it was authorized to access must be eventually re-encrypted.

As discussed before, *PrivICN* keeps the original features of ICN and does not require to send interests to specific nodes *e.g.,* proxy. Moreover, it provides easy user management, especially for addition and removal of content consumers from the system.

9. Conclusion

Privacy is a major concern in Information-Centric Networking (ICN). ICN nodes can monitor, profile and censor clients if the confidentiality of content data and content names is not properly protected. To address this problem, we introduced *PrivICN*, a privacy preserving system for requesting and retrieving contents in ICN. *PrivICN* relies on a modified proxy-encryption scheme to provide confidentiality for both content data and content names. In contrast to previous work, it preserves the full features of ICN and introduces a small computational overhead on clients and network edges only. It presents several features that have been extensively evaluated both theoretically and empirically in a real ICN network (CCNx implementation). These include ease of user management and especially the addition and removal of participants at a low cost. It presents an acceptable communication delay and a low computational overhead on the client (consumer and provider) and the ICN edge nodes while requiring an acceptable storage on ICN nodes. The security of our scheme has been assessed since all content data and content names are encrypted with a required level of security. In addition, it is resilient to most attacks ICN nodes can perform. It is deployable with small adaptations to an ICN network and all components needed for deployment are publicly available as an open-source library. As future work we will propose solutions to reduce the traffic generated by *PrivICN* since this currently grows linearly with respect to the size of the key used for encryption.

Acknowledgements

This work was supported in part by the SELIoT project funded by the Academy of Finland under the WiFiUS program (grant 309994). We thank the anonymous reviewers for their helpful comments.

[1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update, 2016-2021," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf, Cisco, Tech. Rep., March 2017, Cisco White Paper.

[2] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, 2012.

[3] N. Fotiou, P. Nikander, D. Trossen, G. C. Polyzos *et al.*, "Developing information networking further: From psirp to pursuit." in *Broadnets*. Springer, 2010, pp. 1–13.

[4] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 181–192.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[6] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[7] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[8] C. Bernardini, "Stratégies de cache basées sur la popularité pour content centric networking," Ph.D. dissertation, University of Lorraine, May 2015.

[9] A. Chaabane, E. De Cristofaro, M. A. Kaafar, and E. Uzun, "Privacy in content-oriented networking: Threats and countermeasures," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 3, pp. 25–33, July 2013.

[10] R. Tourani, S. Misra, J. Kliewer, S. Ortegel, and T. Mick, "Catch me if you can: A practical framework to evade censorship in information-centric networks," in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 167–176.

[11] N. Fotiou, D. Trossen, G. Marias, A. Kostopoulos, and G. Polyzos, "Enhancing information lookup privacy through homomorphic encryption," *Security and Communication Networks*, vol. 7, no. 12, pp. 2804–2814, 2014.

[12] Mannes et al., "Controle de acesso baseado em reencriptação por proxy em redes centradas em informação," in *SBseg*, 2014.

[13] R. S. da Silva and S. D. Zorzo, "An access control mechanism to ensure privacy in named data networking using attribute-based encryption with immediate revocation of privileges," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, Jan 2015, pp. 128–133.

[14] B. Li, D. Huang, Z. Wang, and Y. Zhu, "Attribute-based access control for ICN naming scheme," *IEEE Transactions on Dependable and Secure Computing*, 2016.

[15] T. Chen, K. Lei, and K. Xu, "An encryption and probability based access control model for named data networking," in *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International*. IEEE, 2014, pp. 1–8.

[16] Y. Wang, M. Xu, Z. Feng, Q. Li, and Q. Li, "Session-based access control in information-centric networks: Design and analyses," in *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International*. IEEE, 2014, pp. 1–8.

[17] Dong et al., "Shared and searchable encrypted data for untrusted servers," *Journal of Computer Security*, vol. 19, no. 3, 2011.

[18] A.-A. Ivan and Y. Dodis, "Proxy cryptography revisited." in *NDSS*, 2003.

[19] "libprotector," https://github.com/mesarpe/libprotector, last accessed: July 5, 2017.

[20] M. R. Asghar, C. Bernardini, and B. Crispo, "Protector: Privacy-preserving information lookup in content-centric networks," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–7.

[21] N. Fotiou, S. Arianfar, M. Särelä, and G. C. Polyzos, "A framework for privacy analysis of ICN architectures," in *Privacy Technologies and Policy - Anual Privacy Forum*. Springer, 2014, pp. 117–132.

[22] C. Ghali, G. Tsudik, and C. A. Wood, "Network names in content-centric networking," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 132–141.

[23] J. Hong and et al., "Requirements for Name Resolution Service in ICN," 2018. [Online]. Available: https://tools.ietf.org/id/draft-jhong-icnrg-nrs-requirements-03.html

[24] S. Signorello, S. Marchal, J. François, O. Festor, and R. State, "Advanced interest flooding attacks in named-data networking," in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2017, pp. 1–10.

[25] M. Blaze, G. Bleumer, and M. Strauss, *Divertible protocols and atomic proxy cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 127–144. [Online]. Available: http://dx.doi.org/10.1007/BFb0054122

[26] P. de Hert and V. Papakonstantinou, "The new general data protection regulation: Still a sound system for the protection of individuals?" *Computer Law & Security Review*, vol. 32, no. 2, pp. 179–194, 2016.

[27] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of CRYPTO 84 on Advances in Cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 10–18.

[28] L. Wang, A. M. Hoque, C. Yi, A. Alyyan, and B. Zhang, "OSPFN: An OSPF based routing protocol for Named Data Networking," NDN Consortium, Tech. Rep. NDN-0003, 2012.

[29] J. Garcia-Luna-Aceves, "Name-based content routing in information centric networks using distance information," in *Proceedings of the 1st international conference on Information-centric networking*. ACM, 2014, pp. 7–16.

[30] "Ccnx helloworld," https://github.com/PARC/ccnxHelloWorld, last accessed: July 5, 2017.

[31] U. Schnurrenberger, "ICN names," http://www.icn-names.net/download/cisco-icn-names-2014-12_en.html, last accessed: October 10, 2018.

[32] J. Shi, "Named data networking in local area networks," 2017. [Online]. Available: http://hdl.handle.net/10150/625652

[33] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proceedings of the 12th ACM conference on Computer and communications security*. ACM, 2005, pp. 364–372.

[34] E. Barker, "Nist special publication 800-57 part 1 revision 4," NIST, Tech. Rep., 2016.

[35] J. S. Dwoskin and R. B. Lee, "Hardware-rooted trust for secure key management and transient trust," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. ACM, 2007, pp. 389–400.

[36] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.

[37] R. Tourani, T. Mick, S. Misra, and G. Panwar, "Security, privacy, and access control in information-centric networking: A survey," *arXiv preprint arXiv:1603.03409*, 2016.

[38] G. Acs, M. Conti, P. Gasti, C. Ghali, and G. Tsudik, "Cache privacy in named-data networking," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE, 2013, pp. 41–51.

[39] A. Fiat and M. Naor, "Broadcast encryption," in *Annual International Cryptology Conference*. Springer, 1993, pp. 480–491.

[40] C. A. Wood and E. Uzun, "Flexible end-to-end content security in ccn," in *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*. IEEE, 2014, pp. 858–865.

[41] A. Shamir *et al.*, "Identity-based cryptosystems and signature schemes." in *Crypto*, vol. 84. Springer, 1984, pp. 47–53.

[42] N. Fotiou and G. C. Polyzos, "Securing content sharing over ICN," in *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 176–185.

[43] R. S. da Silva and S. D. Zorzo, "On the use of proxy re-encryption to control access to sensitive data on information centric networking," in *Information Networking (ICOIN), 2016 International Conference on*. IEEE, 2016, pp. 7–12.

[44] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing," *Communications of the ACM*, vol. 42, no. 2, pp. 39–41, 1999.

[45] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.

[46] J. Kurihara, K. Yokota, and A. Tagami, "A consumer-driven access control approach to censorship circumvention in content-centric networking," in *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 186–194.

[47] G. Tsudik, E. Uzun, and C. A. Wood, "Ac 3 n: Anonymous communication in content-centric networking," in *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*. IEEE, 2016, pp. 988–991.

[48] N. Borisov, G. Danezis, and I. Goldberg, "Dp5: A private presence service," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 4–24, 2015.

[49] S. Yekhanin, "Private Information Retrieval," *Commun. ACM*, vol. 53, no. 4, pp. 68–73, April 2010.

[50] P. Williams and R. Sion, "Usable PIR," in *NDSS*. The Internet Society, 2008.

[51] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*. IEEE, 1995, pp. 41–50.

[52] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: an extremely simple oblivious RAM protocol," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 299–310.

[53] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431–473, May 1996.

[54] R. Ostrovsky, "Efficient computation on oblivious rams," in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 1990, pp. 514–523.

[55] M. Ion, J. Zhang, and E. M. Schooler, "Toward content-centric privacy in ICN: Attribute-based encryption and routing," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013, pp. 39–40.

[56] A. Sahai, B. Waters *et al.*, "Fuzzy identity-based encryption." in *Eurocrypt*, vol. 3494. Springer, 2005, pp. 457–473.

[57] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*. Acm, 2006, pp. 89–98.

[58] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE, 2007, pp. 321–334.