PROTECTOR: Privacy-Preserving Information Lookup in Content-Centric Networks

Muhammad Rizwan Asghar*, César Bernardini† and Bruno Crispo†‡
*The University of Auckland, 1142 Auckland, New Zealand
†University of Trento, 38123 Povo (TN), Italy
‡DistriNet, KU Leuven, 3001 Leuven, Belgium

***Abstract*—Content-Centric Networking (CCN) is an emerging paradigm that can anticipate growing demands of content delivery in coming years. The underlying architecture of the CCN enables users to search for content based on names. On one hand, this is a privacy-friendly feature that do not require source and destination addresses. On the other hand, semantically-rich names reveal sufficient information about users' preferences. Unfortunately, a curious CCN node may learn and sell sensitive information to third-parties, thus posing serious threats to users' privacy. In this paper, we present *PROTECTOR* that aims at protecting content names as well as content and allows a CCN network to add new users or remove existing ones without requiring any re-encryption of stored content and names. It is scalable and efficient as it incurs very limited overhead for required cryptographic operations. Our performance analysis reports that *PROTECTOR* can handle 34 and over 10 million requests per second at boundary and other CCN nodes, respectively.**

I. Introduction

The Internet was initially designed as a communication network providing host-to-host communication using IP addresses. However, it has dramatically evolved over time and become a global platform used by governments, corporations and individuals for sharing a huge amount of data. According to Cisco [1], the global mobile data traffic will increase nearly tenfold in the next five years. Moreover, all forms of content delivery in the Internet are expected to represent 86% of the whole traffic [2]. In order to cope up with growing demands of content delivery, researchers have proposed to replace the existing host-to-host communication network with a *content-centric* one.

Several content-centric architectures have been presented, such as PURSUIT [3], DONA [4], Content-Centric Networking (CCN) [5]. Among all of them, CCN has received most of the attention from the community because of three features: hierarchical naming scheme at networking level, coupling of name forwarding and data routing and simple easy-to-manage caching features (for efficiency reasons) at every node of the network [6], [7].

In content-centric architectures, content is searched using names. Like DNS in the Internet, CCN uses a hierarchical naming scheme. These names are semantically tied to user preferences and are disseminated within the network. Unfortunately, names in plaintext may leak sensitive information to untrusted CCN nodes because they can easily profile user interests. Even if names are protected, curious CCN nodes may deduce private information from the content returned to the users. Therefore, not only names but also content should be protected in order to preserve privacy of users in the CCN.

There are some solutions for protecting names [8]–[10], content [11] and both [12] in CCN. However, there is no solution for protecting both names and content while in-network caching benefits are respected. More importantly, existing solutions suffer from scalability issues when it comes to user key management and performance. Basically, there is no scheme that can handle addition of new users or removal of existing ones from CCN without requiring re-encryption of names or content stored on CCN nodes.

In this paper, we propose a privacy-preserving mechanism named *PROTECTOR* (Privacy-preserving infoRmation loOkup in conTEnt cenTric netwORks) that aims at protecting names and content in the CCN. *PROTECTOR* is based on proxy encryption [13], [14] in which users or CCN nodes do not share any key. The contribution of this paper is four-fold.

- First, CCN nodes can forward content within the network without learning interest of users.
- Second, CCN nodes do not gain access to content.
- Third, the system is efficient and scalable as it can handle a large number of requests in the fraction of a second.
- The system offers a scalable user key management that enables addition of new users or removal of existing ones without requiring re-encryption of names or content stored on CCN nodes.

The rest of the paper is organised as follows. Section II provides a brief overview of CCN and describes the problem statement and associated research challenges. Section III presents the threat model. Next, we explain our proposed approach in Section IV. Section V presents *PROTECTOR*. Sections VI and VII report on security analysis and performance analysis, respectively. Related work is reviewed in Section VIII. Finally, we draw some conclusions and highlight the direction for future work in Section IX.

II. Overview and Problem

A. CCN Overview

CCN is a clean-slate approach for the future Internet. Communication is based on two primitives: *Interest* and *Data* messages. The *Interest* expresses the will of a user for a content, while a *Data* message contains the answer for that

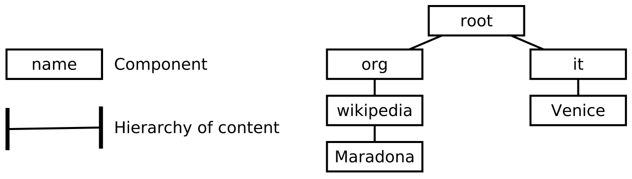


Fig. 1: Hierarchical representation of content names in CCN.

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 organised and human-readable components, as we can see in Figure 1. For instance, /it/Venice and /org/wikipedia/Maradona are two valid CCN names, composed of 2 and 3 components, respectively.

To support *Content Names*, every CCN node holds three tables:

- *Content Store (CS)* is a caching structure that stores content temporarily.
- *Pending Interest Table (PIT)* keeps track of the currently non-satisfied interests. Basically, it serves as a trace for the reverse path once the requested content is found.
- *Forwarding Interest Base (FIB)* is a routing table used to determine the interface to transmit content.

When an *Interest* arrives, the node extracts the *Interest* name and looks up into its CS for content stored that matches the full name. If the content is found, it is automatically sent back through the interface over which the Interest arrived and the *Interest* is discarded. Otherwise, we lookup the PIT to decide whether the CCN node is already waiting for the requested content. If an entry is found, the *Interest* is discarded. Otherwise, the FIB table is checked to decide the interface to redirect the *Interest* to. In the FIB table, a longest prefix matching operation is performed and the *Interest* is forwarded to another CCN node where same procedure is repeated until the requested content is eventually found.

Both PIT and FIB are illustrated in Figure 2. For brevity reasons, we do not show CS in Figure 2. In this example, user Bob is willing to retrieve wikipedia page of Maradona. Bob creates an *Interest* message with the name /org/wikipedia/ Maradona. As Bob is directly connected to *node A*, he sends it to *node A* (Step 1). *Node A* first checks whether the content is stored in its CS, which is not the case. Then, it verifies its PIT and there are no pending request for the requested content. As a consequence, the longest prefix match is searched in the FIB table and a new entry is added to the PIT specifying the *Content Name* and issuer of the request (*i.e., Bob*). As a result, the *Content Name* finds an exact match and forwards the content to *node B* as specifies the FIB (Step 2). At *node B*, the same procedure is repeated and the *Interest* is forwarded to *node C* (Step 3). At *node C*, the content is found and the content itself is encapsulated into a *Data* message (Step 4), which is forwarded – through the reverse path – using the entries of the PIT (Steps 5-8). The intermediary nodes stores a copy of the content in its CS. Finally, Bob receives wikipedia page of Maradona.

B. Problem Statement

As illustrated in Figure 2, CCN nodes see *Content Names* in cleartext. Unfortunately, this poses serious privacy threats to users because a curious CCN node can easily infer what a user or a group of users is interested in. For instance, if we refer to the example illustrated in Figure 2, nodes A, B or C can discover that Bob is interested in wikipedia or Maradona.

Protecting only *Content Names* will not solve the issue because a semantic correlation exists between the *Content Names* and the content itself. More specifically, even if *Content Names* are protected, the CCN nodes can deduce private information from the cleartext content returned to the user. CCN nodes may sell this information to third-parties, *e.g.,* marketing companies. Or, even worst, a malicious CCN node may censor certain types of content [15].

C. Research Challenges

In order to preserve privacy of users in the CCN that could be deployable in practice, the following research challenges must be addressed:

C1 In the presence of curious CCN nodes, how can users request for content without exposing their interest?

C2 How can a user receive content without providing any access to CCN nodes?

C3 In a network with a large number of users each one making multiple requests, how can we provide efficient lookups, *i.e.,* scalability of the protection scheme?

C4 How can we address aforementioned challenges without sharing any keys between content providers and users?

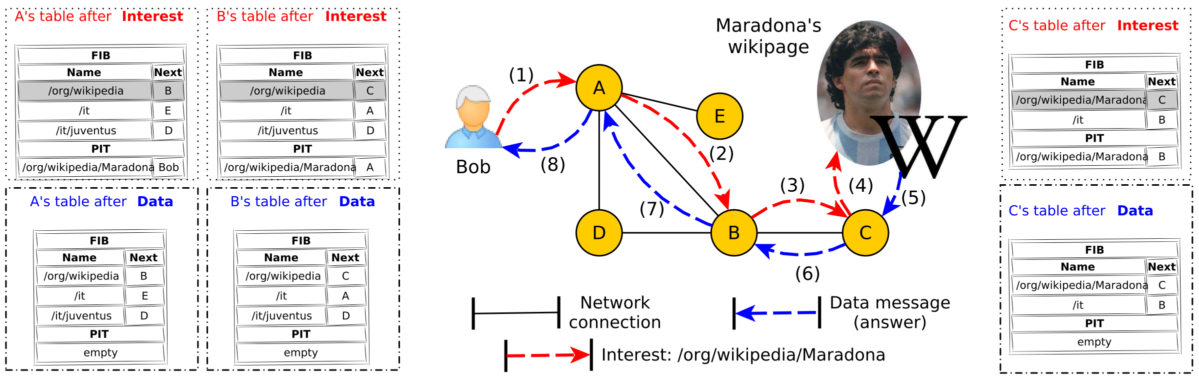
C5 How can we design the system where users (or CCN nodes) could be added or removed without requiring re-encryption of already stored content?

III. Threat Model

We first identify system entities, assumptions, potential adversaries and possible attacks.

There are four types of entities in our system. *Users* are entities that express their interest for content. *Content Providers* are entities that provide content. *CCN Nodes* are nodes of the network responsible for storing content and sending it to interested users. *Key Management Server (KMS)* is a fully trusted authority that is responsible for generation and revocation of the keys. The key material is distributed out of band.

The main adversary is a CCN node that is considered honest-but-curious, meaning it is honest to follow the protocol but curious to learn about interest of users or content. We assume that CCN nodes can collude, *i.e.,* even if they compromise, the information they might infer must be limited. We only assume passive attacks and do not cover active attacks by CCN nodes. We do not address integrity of *Content Names* or content.



A's table after **Interest**

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

B's table after **Interest**

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

A's table after **Data**

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

B's table after **Data**

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

C's table after **Interest**

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

C's table after **Data**

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

Fig. 2: A typical CCN architecture (inspired from [6]) in which Bob is interested in *Maradona's wikipage*. His interest is disseminated within the network. Once the content is found, it is sent back through the path as taken by the *Interest* message.

IV. PROTECTOR: The New Scheme

The main idea behind *PROTECTOR* is to employ proxy encryption [13], [14] for providing privacy-preserving lookups and protection of content. For protecting user interest, the user client transforms each component in *Content Name* into a corresponding trapdoor and then sends it to the CCN node. A trapdoor is a function that is easy to calculate in one direction and complicated in the opposite direction. While serving as a proxy, the CCN node re-encrypts the user generated trapdoors before performing any matching. Next, the CCN node matches the encrypted *Content Name* against encrypted tables stored on CCN nodes. The trapdoor generation does not leak information about components in *Content Name*. Content providers follow the same steps to disseminate the encrypted *Content Name* in the network. That is, a content provider transforms components in *Content Name* into trapdoors that are re-encrypted by the CCN node.

Like protection of user interest, a content provider encrypts content and sends it to the CCN node that re-encrypts it before storing it. As explained above, a CCN node performs encrypted matching. If the match is successful, the CCN node (as a proxy) pre-decrypts the content and sends it to requested users. The user client finally decrypts the content.

In *PROTECTOR*, a CCN node neither learns interest (*i.e.,* components in *Content Name*) of users nor gets access to content, thus tackling both *C1* and *C2*. At the same time, *PROTECTOR* offers efficient lookups (*i.e., C3*). In *PROTECTOR*, users, content providers or CCN nodes do not share any keys (*i.e., C4*) and even if a user or content provider is added or removed, the system is still able to function without requiring re-encryption of *Content Name* or content (*i.e., C5*).

Using *PROTECTOR*, the flow of messages mainly remains same as already explained in Figure 2. The adaptation is required for completing Steps 1, 5 and 8. Step 1 is decomposed into two: first, the user generates trapdoors and then the CCN node re-encrypts them. For completing Step 5, a content provider encrypts content while the CCN node re-encrypts it. Likewise, the delivery of content in Step 8 is completed first by pre-decryption by the CCN node and then finally decrypted by the user.

All other steps (Steps 2-4 and 6-7) are same as already illustrated in Figure 2 and explained in Section II.

V. Solution Details

A. Initialisation and Key Generation Phase

Given a security parameter, the system generates a master secret key and public parameters. The KMS runs **Init** for this purpose. After the system has been set up, the KMS generates keys for every user (or content provider) in the system. The KMS runs **KeyGen**, which takes as input an identity of the user and generates a key pair including a user key and a (proxy) server key. The user key is securely sent to the user while the server key is given to the CCN nodes that a user (or content provider) interacts with. This means that every node that is interacted by the user maintains a key store for storing server side keys.

- **Init(k):** Given a security parameter k, the KMS generates two prime numbers p and q such that $q|p-1$. It generates g such that $\mathbb{G}$ is the unique order q subgroup of $\mathbb{Z}_p^*$. It chooses a random $s \in \mathbb{Z}_q^*$ and $x \in \mathbb{Z}_q^*$, which is a master secret key and calculates $h = g^x$. It publishes public parameters $Params = (\mathbb{G}, g, q, h, f)$ (where f is a pseudorandom function) and keeps securely the master secret key $MSK = (x, s)$ The key store K_S is initialised as: $K_S \leftarrow \phi$.
- **KeyGen(MSK, i):** For each user i, the KMA chooses a random $x_{i1} \in \mathbb{Z}_q^*$ and computes $x_{i2} = x - x_{i1}$. Next, it transmits $K_{U_i} = (x_{i1}, s)$ securely to user i and $K_{S_i} = (i, x_{i2})$ to the CCN nodes that store all server side keys K_{S_i} in their key stores: $K_S \leftarrow K_S \cup K_{S_i}$.

The CCN network configures routing tables with all the available content in the network. This step is completed using a routing algorithm such as [16], [17].

B. Request Generation Phase

To preserve privacy of users, a user client runs **UserTD** for transforming every component in *Content Name* into a corresponding trapdoor using the user key provided by the KMS. As compared to the standard messaging flow illustrated in Figure 2, the additional step is that the user client prepares trapdoors before Step 1. Once these trapdoors are generated, they are sent to the CCN node in Step 1, which is same as a user client does in case of the traditional CCN architecture.

- **UserTD**(e, K_{U_i})**:** The user client computes: $\sigma = f_s(e)$. Finally, it generates $td_i^*(e) = (t_1, t_2)$, where $t_1 = g^{\sigma}$ and $t_2 = t_1^{x_{i1}} = g^{x_{i1}\sigma}$.

C. CCN Re-Encryption Phase

To preserve functionalities of the CCN, CCN nodes need to perform comparison operations based on *Content Names*. As users send encrypted *Content Names* in the form of trapdoors, CCN nodes must transform them in such a way they can still perform these comparisons. This is achieved by means of re-encrypting *Content Names*. CCN nodes run **CCNTD**. As a result, names are re-encrypted using the server side key corresponding to the user that issued the request. Due to the proxy encryption property, CCN nodes are able to perform queries without knowing user client keys. In other words, CCN nodes are not able to decrypt *Content Names* to learn any information.

- **CCNTD**($i, td_i^*(e)$)**:** The CCN node retrieves from the key store the key K_{S_i} corresponding to the user i. Next, it computes: $td(e) = T = t_1^{x_{i2}}.t_2 = g^{x\sigma}$.

D. CCN Lookup Phase

As we illustrated in Figure 2, CCN nodes maintain three tables: CS, PIT and FIB. All these tables must support two types of operations: the exact match and the longest prefix match, explained in Section II-A.

For performing the encrypted matching, a CCN node runs **MatchTDs**. As usual, the CCN algorithm is executed and CS, PIT and FIB tables are queried to retrieve content. First, the CS is revisited to check whether content is in the cache. For instance, it looks for an exact match between the encrypted *Content Name* and the encrypted entries of the table. Second, in case the content is not found, another exact match is searched, but now, in the PIT. If the response is still negative, a longest prefix match is performed in the FIB table and immediately after, the *Interest* is forwarded. All these comparisons are performed over encrypted operands, *i.e.,* CCN nodes neither learn about entries being queried nor the name being requested.

- **MatchTDs**($td(e_1), td(e_2)$)**:** The CCN node compares two encrypted elements by performing a simple equality match, *i.e.,* by evaluating $T_1 \overset{?}{=} T_2$. It returns *true* in case of a match and *false* otherwise.

E. Content Generation Phase

Content providers furnish content and *Content Names*. In *PROTECTOR*, a content provider encrypts content using proxy encryption. A content provider runs **Content-Provider-Enc**, which takes as input the user key of the content provider and encrypts the content. In order to benefit from the performance of symmetric cryptosystems, the content is encrypted using a randomly generated symmetric key. This symmetric key is then encrypted by running **Content-Provider-Enc**. Likewise, content providers protect components in content name by running **UserTD**. The client encrypted content and content names are sent to the CCN node. CCN nodes re-encrypt the content and *Content Names* by running **Content-Provider-Re-Enc** and **CCNTD**, respectively. For the re-encryption, CCN nodes fetch the server side key corresponding to the content provider from its key store. Once the content and *Content Names* are re-encrypted, they could be searched.

- **Content-Provider-Enc**(D, K_{U_i})**:** User i encrypts data using proxy encryption. For each data element D, it chooses random r from $\mathbb{Z}_q^*$. Next, it computes $PE_i^*(D) = (\hat{e}_1, \hat{e}_2)$, where $\hat{e}_1 = g^r$ and $\hat{e}_2 = g^{rx_{i1}}D$. Finally, $PE_i^*(D)$ is sent to the CCN node.
- **Content-Provider-Re-Enc**($i, PE_i^*(D)$)**:** The CCN node fetches the server side key K_{S_i} corresponding to user i and computes $(\hat{e}_1)^{x_{i2}} \cdot \hat{e}_2 = (g^r)^{x_{i2}} \cdot g^{rx_{i1}}D = g^{rx_{i1}+rx_{i2}}D = g^{rx}D$ in order to get $PE(D) = (e_1, e_2)$, where $e_1 = g^r$ and $e_2 = g^{rx}D$.

F. CCN Pre-Decryption Phase

Once content is found in the network, a *Data* message follows the reverse path taken by its corresponding *Interest* message. The *Data* message arrives eventually to the CCN node directly connected with the user who made the request. As the content is encrypted, the CCN node needs to translate it in such a manner that the user is able to understand content. Thus, the CCN node pre-decrypts the content using the server side key corresponding to the target user. CCN nodes run **CCN-Pre-Dec**. A CCN node cannot understand the content before and after the pre-decryption.

It is important to mention that our solution only needs to decrypt the *Payload* of the *Data* message. The *Data* name remains encrypted. However, to simplify the explanation, we have referred as *Data* message instead of *Payload* of the *Data* message.

- **CCN-Pre-Dec**($j, PE(D)$)**:** The CCN node fetches the server side key K_{S_j} corresponding to user j. The ciphertext $PE(D)$ is decrypted as $e_2 \cdot (e_1)^{-x_{j2}} = g^{rx}D \cdot (g^r)^{-x_{j2}} = g^{r(x-x_{j2}}D = g^{rx_{j1}}D$. The CCN node sends to the user $PE_j^*(D) = (\hat{e}_1, \hat{e}_2)$, where $\hat{e}_1 = g^r$ and $\hat{e}_2 = g^{rx_{j1}}D$.

G. User Decryption Phase

When a user receives the *Data* message, the message has been previously pre-decrypted by a CCN node. A user client runs **User-Dec**. The user proceeds to decrypt it using her key. It is important to highlight that this user is the only one capable of decrypting the message because she holds the corresponding user key. As a result of decryption, the symmetric key is extracted. This key is ultimately used to decrypt the content. Thus, the retrieval of content procedure ends without compromising privacy of *Content Names* and the content itself.

- **User-Dec**($PE_j^*(D), K_{U_j}$)**:** The user decrypts ciphertext as $\hat{e}_2 \cdot (\hat{e}_1)^{-x_{j1}} = g^{rx_{j1}}D \cdot (g^r)^{-x_{j1}} = D$.

H. Revocation Phase

Eventually, a user (or content provider) key may be compromised or she may ask to be removed from the system. In this case, all the server keys associated to this user could be revoked. This operation is managed by the KMS. The KMS runs **Revoke**. The KMS communicates securely to CCN nodes and request them to remove the server side key corresponding to requested user.

- **Revoke(i):** The CCN node revokes user i access the data by removing K_{S_i} from the key store as follows: $K_S \leftarrow K_S \backslash K_{S_i}$.

VI. Security Analysis

In this section, we analyse the security of *PROTECTOR* including protection of interest packets and data messages. In general, a scheme is considered secure if no adversary can break the scheme with probability significantly greater than random guessing. The adversary's advantage in breaking the scheme should be a negligible function (defined below) of the security parameter.

Definition 1 (Negligible Function): A function f is negligible if for each polynomial $p()$, there exists N such that for all integers $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.

We consider a realistic adversary that is computationally bounded and show that our scheme is secure against such an adversary. We model the adversary as a randomised algorithm that runs in polynomial time and show that the success probability of any such adversary is negligible. An algorithm that is randomised and runs in polynomial time is called a Probabilistic Polynomial Time (PPT) algorithm.

Our scheme relies on the existence of a pseudorandom function f. Intuitively, the output a pseudorandom function cannot be distinguished by a realistic adversary from that of a truly random function. Formally, a pseudorandom function is defined as:

Definition 2 (Pseudorandom Function): A function $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ is pseudorandom if for all PPT adversaries $\mathscr{A}$, there exists a negligible function $negl$ such that:

$$|Pr[\mathscr{A}^{f_s(\cdot)} = 1] - Pr[\mathscr{A}^{F(\cdot)} = 1]| < negl(n)$$

where $s \rightarrow \{0,1\}^n$ is chosen uniformly randomly and F is a function chosen uniformly randomly from the set of function mapping n-bit strings to n-bit strings.

Our proof relies on the assumption that the Decisional Diffie-Hellman (DDH) is hard in a group $\mathbb{G}$, *i.e.,* it is hard for an adversary to distinguish between group elements $g^{\alpha\beta}$ and g^{γ} given g^{α} and g^{β}.

Definition 3 (DDH Assumption): The DDH problem is hard regarding a group $\mathbb{G}$ if for all PPT adversaries $\mathscr{A}$, there exists a negligible function $negl$ such that:

$$|Pr[\mathscr{A}(\mathbb{G},q,g,g^{\alpha},g^{\beta},g^{\alpha\beta}) = 1] \\ -Pr[\mathscr{A}(\mathbb{G},q,g,g^{\alpha},g^{\beta},g^{\gamma}) = 1]| < negl(k)$$

where $\mathbb{G}$ is a cyclic group of order q $(|q| = k)$ and g is a generator of $\mathbb{G}$, and $\alpha,\beta,\gamma \in \mathbb{Z}_q$ are uniformly randomly chosen.

Theorem 1: If the DDH problem is hard relative to $\mathbb{G}$, then the trapdoor scheme TD is secure against PPT adversaries $\mathscr{A}$.

Proof. The trapdoor scheme TD consisting of **UserTD** and **CCNTD** could be realised as the DDH problem because of the following mapping: for $g, h = g^x, g^{\sigma}$ and $g^{x\sigma}$, there exists a negligible function $negl$ as follows:

$$|Pr[\mathscr{A}(\mathbb{G},q,g,g^{x},g^{\sigma},g^{x\sigma}) = 1] \\ -Pr[\mathscr{A}(\mathbb{G},q,g,g^{x},g^{\sigma},g^{\gamma}) = 1]| < negl(k)$$

Theorem 2: If the DDH problem is hard relative to $\mathbb{G}$, then the proxy encryption scheme PE is INDistinguishable under Chosen Plaintext Attack (IND-CPA) secure against the server S, *i.e.,* for all PPT adversaries $\mathscr{A}$ there exists a negligible function $negl$ such that:

$$Succ^{\mathscr{A}}_{PE,S}(k) = Pr\left[b' = b \;\middle|\; \begin{array}{c} (Params, MSK) \leftarrow Init(1^k) \\ (K_{U_i}, K_{S_i}) \leftarrow KeyGen(MSK, i) \\ d_0, d_1 \leftarrow \mathscr{A}^{Content\text{-}Provider\text{-}Enc(\cdot, K_{U_i})}(K_{S_i}) \\ b \xleftarrow{R} \{0,1\} \\ PE_i^*(d_b) = Content\text{-}Provider\text{-}Enc(d_b, x_{i1}) \\ b' \leftarrow \mathscr{A}^{Content\text{-}Provider\text{-}Enc(\cdot, K_{U_i})}(K_{S_i}, PE_i^*(d_b)) \end{array} \right] < \frac{1}{2} + negl(k)$$

Proof. See Theorem 1 in [13].

VII. Performance Analysis

A. Implementation Details and Evaluation Parameters

The implementation prototype is developed in Java. The implementation consists of three components: a module for the KMS (KMS-module), a module for the CCN (CCN-module) and a module for the user client (user-module).

The actual experiments are run on a standard notebook with 2.7 GHz processor and 8.0 GB RAM. The experimental results described below constitute the average over 1000 independent executions. Network latency is not considered in these experiments.

B. Evaluating the KMS-Module

Evaluating the performance of the KMS-module consists of measuring the overhead of key generation.

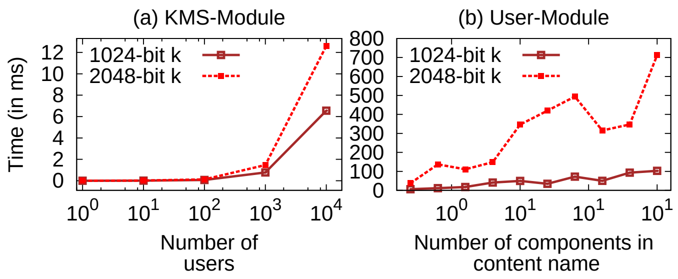


Fig. 3: Evaluation of (a) KMS and (b) User modules.

Key Generation Phase. Figure 3(a) illustrates the computational overhead of the key generation phase. In our experiment, we consider generating keys simultaneously for a number of users, logarithmically varying from 1 to 10000. The key generation time grows linearly with increase in number of users as shown in Figure 3(a). On average, it takes less than 2

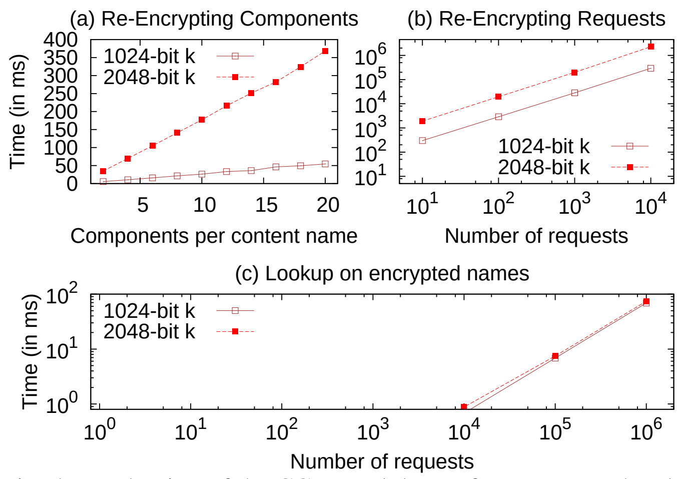


Fig. 4: Evaluation of the CCN-module: performance overhead of (a) re-encrypting trapdoors of components, (b) re-encrypting a large number of trapdoor requests and (c) performing encrypted matching at CCN nodes.

μ seconds per user. As we can see in Figure 3(a), we considered two different security parameters: 1024-bit and 2048-bit. For generating 2048-bit key pairs for 10000 users, the KMS-module just takes 12.61 milliseconds (ms). Asymptotically, the computational complexity of the key generation phase is $O(k)$, where k is the security parameter. The storage complexity is same as the computational complexity. For storing the server side keys at CCN nodes, we need a storage of size $k.U$ bytes, where U represents the number of users.

C. Evaluating the User-Module

Evaluating the performance of the User-module consists of measuring the overhead of generation of user requests, *i.e.,* user trapdoors, client encryption and decryption. For brevity reasons, we just focus on generation of user requests.

Request Generation Phase. Figure 3(b) illustrates the computational overhead of generating user requests, *i.e.,* client trapdoors. In *PROTECTOR*, we generate a trapdoor for each component in content name. In our experiment, we consider different number of components ranging from 2 to 20. The client trapdoor generation time grows linearly with increase in number of components as shown in Figure 3(b). For generating 20 trapdoors, the client takes 102 ms and 712 ms with keys generated using 1024-bit and 2048-bit security parameters, respectively. The curve in graph is not smooth due to random nature of the components used in the power. Asymptotically, the computational complexity of the request generation phase is $O(k.|C|)$, where $|C|$ indicates number of components in content name. The storage/bandwidth complexity is same as the computational complexity.

D. Evaluating the CCN-Module

For the performance evaluation of the CCN-module, we measure the overhead of looking up content, *i.e.,* re-encrypting trapdoors with varying number of components in content names, re-encrypting trapdoors from a large number of users and matching trapdoors at CCN nodes.

CCN Re-Encryption Phase. We conduct two experiments to observe effect on re-encryption of trapdoors. In the first experiment, we consider a variable number of components in content name. In the second experiment, we see effect of a large number of trapdoor requests.

Figure 4(a) shows the performance overhead of re-encrypting trapdoors in the request. For this experiment, we consider up to 20 components in content name[1] and run it with two different security parameters: 1024-bit and 2048-bit. For generating 20 server trapdoors, the CCN-module takes 55 ms and 369 ms with security parameters of sizes 1024-bit and 2048-bit, respectively. The computational complexity of re-encrypting trapdoors grows linearly with increase in number of components in content name or size of the security parameter. Asymptotically, the computational complexity of re-encrypting trapdoors is $O(k.|C|)$. The storage/bandwidth complexity is same as the computational complexity.

Figure 4(b) illustrates the performance overhead of re-encrypting trapdoors. In this experiment, we consider that each request contains 10 components in content name. Like other experiments, we observe the behaviour of CCN nodes while taking into account different sizes of security parameters. For processing 10000 requests, the CCN-module just takes 294 seconds and 2281 seconds with 1024-bit and 2048-bit security parameters, respectively. On average, a CCN node can process more than 34 re-encryption requests when we choose a security parameter of 1024-bit. As expected, the trapdoor re-encryption time grows linearly with increase in number of users. Asymptotically, the computational complexity of re-encrypting trapdoors is $O(k.|C|.U)$. The storage/bandwidth complexity is same as the computational complexity.

CCN Lookup Phase. This is the most important phase of *PROTECTOR*. In this phase, a CCN node performs encrypted match of requests against encrypted content names in the FIB table maintained by CCN nodes. This is the most critical operation in a sense that it will be performed most of the times at various CCN nodes in particular when requested content is not found at a CCN node and requests are disseminated within the network. For this experiment, we assume that a CCN node performs 20 comparisons to lookup the requested content. The rationale behind this assumption is the efficient data structure used for lookups. For instance, it could be a binary search tree using which components of content names are sorted, though they are encrypted using trapdoors. Figure 4(c) illustrates performance of encrypted lookups at CCN nodes. For processing 1 million requests, the CCN-module merely takes 69 ms and 74 ms with 1024-bit and 2048-bit security parameters, respectively. Thus, such level of efficiency makes *PROTECTOR* scalable. Like other experiments, the matching time grows linearly with increase in the security parameters or number of requests. Asymptotically, the computational complexity of re-encrypting trapdoors is $O(k.|C|.|L|.U)$, where

[1]According to Content Name datasets available at http://www.icn-names.net/, the average number of components in content name is no more than 6.

L represents entries (or components) of content names in the FIB table managed by a CCN node.

VIII. Related Work

Although the content-centric architecture addresses existing problems in the current Internet, it introduces an array of new privacy threats including cache privacy, content privacy, name privacy and signature privacy [8]. Ács *et al.* [18] study the problem of cache privacy. They present timing attacks used to learn whether nearby consumers have recently requested certain content. Mannes *et al.* [11] propose a proxy-encryption scheme for addressing the problem of content privacy. However, they do not consider name privacy. Chaabane *et al.* [8] are the first to solve the problem of name privacy. They propose a solution that is based on bloom filters. There are two main issues: first, they do not provide any security analysis; second, bloom filters may return false positive. Tourani *et al.* [9] propose an anti-censhorship mechanism based on huffman coding. The Huffman coding encodes and protects Content Names at a high speed rate. Tsudik *et al.* [12] protect Content Names and content with anonymous communication. Requests are wrapped into layers of encryption that are used to prevent monitoring activites. Both solutions tie requests to the end-user and are unusable to serve new requests, loosing all the benefits from in-network caching.

Fotiou *et al.* [10] propose a privacy-preserving lookup using homomorphic encryption. Basically, they model the problem as Private Information Retrieval (PIR). Unfortunately, the major issue with all PIR-based schemes [19]–[22] and Oblivious RAM (ORAM) [23]–[25] is that they are infeasible for large scale deployment due to their high computational complexity.

The problem of encrypted matching in CCN could be realised an instance of the wider problem of a search over encrypted data. Song *et al.* [26] propose a search scheme over encrypted data based on symmetric keys. The symmetric nature of the scheme rules out its applicability in CCN due to loosely coupled nature of users and content providers. The PEKS scheme [27] supports a search on encrypted data in the public key setting. The main drawback of this scheme is its computational cost. Similarly, the Attribute-Based Encryption (ABE) schemes [28], [29] or solutions based on them [30] use bilinear pairing that incurs high performance overheads.

IX. Conclusions and Future Work

User privacy is a major concern in content-centric architectures, which could be compromised by content names or content they are semantically tied with. To solve this problem, in this paper, we have presented *PROTECTOR*, a privacy-preserving scheme that protects content name and the content itself without compromising any underlying functionality of the CCN. The security analysis indicates that our proposed scheme is secure. Depending on the security parameter, *PROTECTOR* can handle over 34 requests per second at boundary nodes and over 10 million requests per second at other CCN nodes, thus making our proposal practically feasible.

As future work, we expect to investigate possible performance overheads of protecting content of various types including, but are not limited to, large documents, audio and video. Second, we would like to integrate our scheme with CCNx [31] and see the gain by using caching features.

References

[1] Cisco, “Cisco visual networking index: Global mobile data traffic forecast update, 2014-2019,” February 2015, cisco White Paper.

[2] ——, “Cisco visual networking index: Global mobile data traffic forecast update, 2013-2018,” *Cisco White Paper*, no. 7, February 2014.

[3] Fotiou et al., “Developing Information Networking Further: From PSIRP to PURSUIT.”

[4] Koponen et al., “A data-oriented (and beyond) network architecture,” in *ACM SIGCOMM 2007*.

[5] Jacobson et al., “Networking named content,” in *ACM CoNEXT 2009*.

[6] Xylomenos et al., “A survey of information-centric networking research,” *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 2, pp. 1024–1049, Second 2014.

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

[8] Chaabane et al., “Privacy in content-oriented networking: Threats and countermeasures,” *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 3, pp. 25–33, July 2013.

[9] 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 *ACM ICN 2015*.

[10] Fotiou et al., “Enhancing information lookup privacy through homomorphic encryption,” *Security and Communication Networks*, vol. 7, no. 12, pp. 2804–2814, 2014.

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

[12] G. Tsudik, E. Uzun, and C. A. Wood, “Ac 3 n: Anonymous communication in content-centric networking,” in *IEEE CCNC 2016*.

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

[14] Anca-Andreea et al., “Proxy cryptography revisited,” in *NDSS 2003*.

[15] Fotiou et al., “A framework for privacy analysis of ICN architectures,” in *Privacy Technologies and Policy - APF 2014*, pp. 117–132.

[16] 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.

[17] J. Garcia-Luna-Aceves, “Name-based content routing in information centric networks using distance information,” in *ACM ICN 2014*.

[18] Ács et al., “Cache privacy in named-data networking,” in *IEEE ICDCS 2013*.

[19] N. Borisov, G. Danezis, and I. Goldberg, “DP5: A Private Presence Service,” http://cacr.uwaterloo.ca/techreports/2014/cacr2014-10.pdf, last accessed: October 17, 2015.

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

[21] P. Williams and R. Sion, “Usable PIR,” in *NDSS*. The Internet Society, 2008.

[22] Chor et al., “Private information retrieval,” *J. ACM*, vol. 45, no. 6, pp. 965–981, November 1998.

[23] Stefanov et al., “Path ORAM: An extremely simple oblivious ram protocol,” in *ACM SIGSAC 2013*.

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

[25] R. Ostrovsky, “Efficient computation on oblivious RAMs,” in *ACM STOC 1990*.

[26] Song et al., “Practical techniques for searches on encrypted data,” in *IEEE SSP 2000*.

[27] Boneh et al., “Public key encryption with keyword search,” in *EUROCRYPT 2004*, 2004, vol. 3027, pp. 506–522.

[28] Goyal et al., “Attribute-based encryption for fine-grained access control of encrypted data,” in *ACM CCS 2006*.

[29] Bethencourt et al., “Ciphertext-policy attribute-based encryption,” in *IEEE SSP 2007*.

[30] Asghar et al., “PIDGIN: Privacy-preserving interest and content sharing in opportunistic networks,” in *ASIA CSS 2014*.

[31] “Project ccnx,” https://www.ccnx.org/, last accessed: October 17, 2015.