Multi-CDN: Towards Privacy in Content Delivery Networks*

Shujie Cui, Muhammad Rizwan Asghar, and Giovanni Russello

***Abstract*—A Content Delivery Network (CDN) is a distributed system composed of a large number of nodes that allows users to request objects from nearby nodes. CDN not only reduces end-to-end latency on the user side but also offloads Content Providers (CPs), providing resilience against Distributed Denial of Service (DDoS) attacks. However, by caching objects and processing user requests, CDN providers could infer user preferences and the popularity of objects, thus resulting in information leakage. Unfortunately, such information leakage may result in loss of user privacy and reveal business-specific information to untrusted or compromised CDN providers. State-of-the-art solutions can protect the content of sensitive objects but cannot prevent CDN providers from inferring user preferences and the popularity of objects.**

In this work, we present a privacy-preserving encrypted CDN system to hide not only the content of objects and user requests, but also protect user preferences and the popularity of objects from curious CDN providers. We employ encryption to protect the objects and user requests in a way that both the CDNs and CPs can perform the search operations without accessing objects and requests in cleartext. Our proposed system is based on a scalable key management approach for multi-user access, where no key regeneration and data re-encryption are needed for user revocation. We have implemented a prototype of the system and show its practical efficiency.

***Index Terms*—CDN, Multi-CDN, Confidentiality, Security, Privacy, Searchable encryption, Access pattern, Request pattern.**

I. Introduction

A Content Delivery Network (CDN) is a distributed system composed of a large number of nodes deployed across the world. Each node caches the replica of the most frequently or most recently requested objects, *e.g.,* files, images, and videos. When a user requests a certain object, the request will be forwarded to the nearby node, rather than the Content Provider (CP) (*i.e.,* origin server). CDN not only decreases the end-to-end latency on the user side but also reduces the load on CPs, ensuring availability in the face of Distributed Denial of Service (DDoS) attacks.

Problem. Despite these benefits, the use of CDNs also raises confidentiality and privacy issues for CPs and users, respectively. The CDN infrastructure could be compromised by attacks performed by hackers. For instance, in 2015, the Syrian Electronic Army (SEA) attacked the CDN providers of some major news media outlets [2]. By taking over the CDN servers, the SEA were able to send customised news feeds to the Washing Post's readers. Similarly, an attacker or a rogue employee of the CDN provider could also collect information regarding user's preferences. In some commercial sites, user requests and preferences are business-critical information.

Motivations. In the literature, several works [3]–[5] provide methods to encrypt objects and requests. Unfortunately, due to possible inference attacks [6], [7] that a CDN provider might be able to mount, encrypting the objects and requests only is not sufficient to ensure their confidentiality. For example, existing works [8], [9] have shown that even when web contents are encrypted, the retrieval of a collection of objects of various sizes can yield information about the web page that was being retrieved. Similarly, user requests can also be inferred from relative popularity of web pages or objects, even when the requests themselves are encrypted. As a matter of fact, CDN providers capture all the interests in order to provide a reliable service to their customers. However, disgruntled employees and hackers are always a risk that is outside the control of the CPs. From the CP's point of view, this risk can be mitigated if (i) the content of objects and requests, (ii) the popularity of objects, and (iii) user preferences are protected from the CDN providers.

There are also several other types of network infrastructures with cache functionalities, such as Content-Centric Networking (CCN) [10] and Named Data Networking (NDN) [11], that enable users to get the requested object from the nearest cache node. In the literature, there are many approaches to preserve privacy of requests and objects in these networks. For instance, in [12] and [13] the identifiers or names of objects and requests are protected using deterministic cryptographic hash functions, and [14]–[16] propose to ensure confidentiality of objects and support fine-grained access control over the objects. Although these approaches can also be leveraged to protect the objects and requests in CDNs, they are unable to hide the popularity of the objects and users preference from attackers.

Design Goals. In this work, we have three main goals.

- First, we attempt to ensure confidentiality of the objects and user requests while the CDN nodes are still able to return the request objects users correctly.
- Second, we aim at hiding the popularity of objects and user preferences from CDN nodes. The popularity of an object means the number of times it has been requested by the users. User preferences indicate a set of objects accessed by the user in a certain time period. If such information is revealed to the curious CDN providers, as

*This work is an extension of initial work appeared in the proceedings of the 42nd Annual IEEE Conference on Local Computer Networks (LCN) 2017 under the title "Privacy-preserving Content Delivery Networks" by Shujie Cui, Muhammad Rizwan Asghar, and Giovanni Russello [1].

Shujie Cui, Muhammad Rizwan Asghar, and Giovanni Russello are with the Cyber Security Foundry, The University of Auckland, New Zealand. They can be contacted by email: scui379@aucklanduni.ac.nz, r.asghar@auckland.ac.nz, and g.russello@auckland.ac.nz, respectively.

discussed in [6]–[9], it could be leveraged to infer the content of objects and requests. In this work, we aim to solve the issue using the multi-CDN strategy.
- Third, we also consider the key management for multi-user access. Specifically, our system could revoke compromised users efficiently without regenerating the secret key and re-encrypting the cached objects.

Contributions. We present a privacy-preserving encrypted CDN system to achieve above goals by combing Searchable Encryption (SE) with multi-CDN strategy and with a tradeoff on performance. In our approach, we encrypt the objects and user requests with SE such that both CDNs and CPs can perform the matching operation efficiently without performing any decryption. By distributing the objects and requests across multiple CDN providers, user preferences and object popularity are concealed from CDN providers as long as they do not collude. A scalable key management is also achieved due to the cooperation of multiple CDN service providers. Our main contributions are as follows:

- We propose a framework that enables CDNs to deliver the requested objects to users without learning users' interests.
- Our scheme does not leak the request and object access patterns, making it robust against the attacks described in [6]–[9], [17], [18], which also means our scheme protects the object popularity and user preferences from each single CDN provider.
- Our system is equipped with a scalable key management mechanism, where users can join or leave the system without requiring key re-generation and re-encryption of names or objects cached in CDNs.

The rest of the paper is organised as follows. In the next section, we cover some basic knowledge about CDN and Multi-CDN approaches. In Section III, we provide the overview of our system. Solution details can be found in Sections IV and V. In Section VI, we give the security definition and proof of our scheme. Complexity and performance analysis are reported in Sections VII and VIII, respectively. We discuss the limitations of our system in Section IX. The related work in the literature is reviewed in Section X. Finally, we conclude this paper in Section XI.

II. Background

A. CDN

There are two key issues that CDNs typically address. First, CDN nodes should return the latest objects to users. Basically, the node obtains objects from CPs either by a push or pull method. In the push mode, once the objects are updated or new objects are uploaded, the CPs will send them to the CDN nodes in advance. In the pull mode, if the requested object is already cached locally, the node just returns it to the user. In case if the object is not found in the cache or is out of date, the node will forward the request to the CP to retrieve the object. In general, both modes are integrated into most CDNs.

Second, users should be served by the 'closest' CDN node. The request-routing system in CDNs is responsible for directing users' requests to the 'closest' node. Technically, the 'closest' node is determined according to various policies and metrics, such as the geographical distance between the user and the node, the congestion of the network and the load on the node. The most commonly used request-routing techniques include the URL rewriting and DNS-based request routing. In URL rewriting based CDN, all the requests from users are direct to the CPs first, and then the CPs modify the URL of the request object such that the user will fetch the object in the closest node. In DNS-based request routing approach, a DNS server is employed to map domain names to a set of numerical IP addresses of CDN nodes. When receiving a user request, the DNS server of the CDN provider returns the IP addresses of the nodes. Then, the DNS resolver on the user side will choose the closest one for retrieving. More details of these two approaches and other request-routing techniques can be found in [19].

In this paper, we focus on the CDN systems that use the pull-based mode and DNS-based request routing.

B. Multi-CDN

The time to fulfil users' requests strongly depends on the availability of CDN nodes. Ideally, the more the nodes, the more efficient it is to retrieve the object. However, the nodes owned by one CDN provider are not everywhere, and their performance varies across regions, throughout the day. Multi-CDN is one of the strategies to ensure that objects are delivered to users as quickly as possible by combining a range of existing nodes in the same region and owned by different CDN providers into a single network known as a *cluster*. In this strategy, user requests can be distributed to the most optimal node within the cluster. Due to its benefits, the use of multi-CDN has risen in recent years for those organisations that find their sites experiencing huge amounts of traffic on a global scale, such as Netflix, Linkedin, and Twitter [20]. CDN Aggregator (e.g., Cedexis [21]) and Load Balancer (e.g., Amazon Route 53) [22] are two options for implementing a multi-CDN. Moreover, there are already many multi-CDN strategies in commence, such as [23]–[25]. In this work, we exploit multi-CDN to conceal user preference and object popularity from CDN providers.

C. Peering CDNs

In particular, in our system, the nodes provided by different vendors might communicate with each other. The architecture of *peering CDNs* [26] can be leveraged to build the CDN cluster and manage the communication among the nodes. Peering CDNs virtualise multiple CDN providers and allow flexible resource sharing and dynamic collaboration between autonomous individual CDNs. Specifically, the models for peering CDNs proposed in [19], [27]–[29] can be utilised in our system. Moreover, in case the peering CDNs is not available, we can employ a trusted proxy to manage the communication among the CDN nodes. That is, all the nodes in the cluster only communicate with the proxy, and the proxy forwards the messages to different nodes.

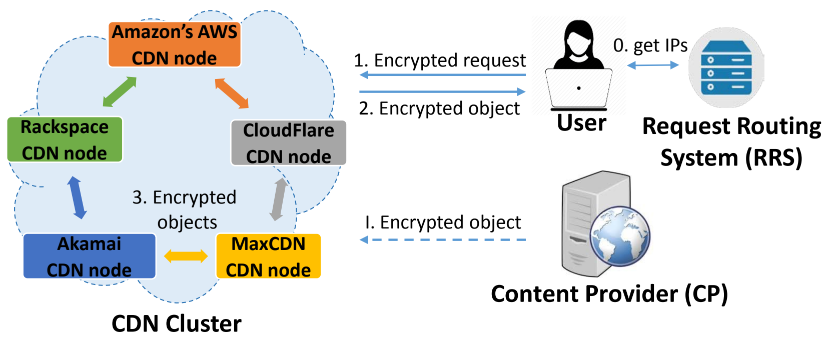


Fig. 1: Proposed architecture: A CDN cluster is composed of a set of CDN nodes located in the same region but owned by different CDN providers. To request an object, the user first gets the IPs of the 'closest' nodes from the RRS (Step 0), and then issues the encrypted request (Step 1). Starting from the 'closest node', the nodes in the cluster search their storage one-by-one until the requested object is found. When there is a hit, the matched object is sent to the user (Step 2). Meanwhile, the matched object is re-randomised and migrated to another node (Step 3). However, if the object is not found in the cluster, the request will be sent to the CP, the CP searches locally and uploads the matched object to the cluster (Step I).

TABLE I: Notations used and meaning.

Notation	Meaning
$O = \langle name, payload \rangle$	An Object consisting of $name$ and $payload$
$PEO = \langle PEN, PEP \rangle$	A Pre-Encrypted Object stored on the CP consisting of the Pre-Encrypted Name PEN and Pre-Encrypted Payload PEP
$EO = \langle EN, EP \rangle$	An Encrypted Object stored on CDN node, consisting of the Encrypted Name EN and Encrypted Payload EP
ER	Encrypted Request
n, η, α	Nonces included in EN, EP, ER, respectively
P_{pid}	The pid-th partition
W	The number of objects in each partition
$\mathcal{N}_i$	The i-th CDN node
N	The number of CDN nodes in a cluster

III. Solution Overview

A. System Model

As shown in Figure 1, our system involves four main entities:

- **User**: It represents the client that can request objects.
- **Content Provider (CP)**: The content provider is the object publisher that is basically a customer of the CDN. It stores and distributes its objects to CDN nodes.
- **CDN Cluster**: It is a set of CDN nodes located in the same geographical region, *e.g.,* a country, but provided by the CDN providers that should be in conflict of interest. Each CDN node can receive user requests and return the matched objects, or redirect the request to the CP.
- **Request Routing System (RRS)**: It is responsible for directing user requests to a high-performing available node.

B. Threat Model

We assume the CP and users are trustworthy. A CP encrypts the outsourced objects and authorises users to get objects from CDN nodes.

In general, the CDN nodes might perform properly as per specified protocol. However, they can be compromised and controlled by the attacker, like the SEA attack occurred in 2015 [2]. For simplicity, in this work, we assume the CDN provider is honest-but-curious, meaning it is honest to follow the designated protocol but curious about the cached objects and user requests. Specifically, the curious provider might try to infer the objects and user privacy by analysing the cached objects and user request history. Moreover, the CDN provider could take snapshots of the cached objects and requests at any time, and check if new requests match stale objects or new objects match previous requests.

Our design does not defend against an active attacker who attempts to modify objects, disrupt communications, or block requests. In [30], Levy et al. have introduced a method to handle an actively malicious adversary by preserving the integrity of object stored on CDN nodes. In this work, we assume the CDN providers do not attempt to tamper, modify, or delete any objects or requests.

Moreover, we assume that the CDN providers in conflict of interest do not collude with each other, or collude with revoked users. Note that the CDN nodes in our system can also be provided by one trusted CDN provider as long as its nodes are in conflict of interest; more specifically, we assume attackers can not compromise any two adjacent nodes simultaneously.

Considering there is no content stored on RRS, it could be managed by the CP or CDN provider.

C. Approach Overview

In this work, we propose a framework that provides CDN services while aim at ensuring user privacy. To protect sensitive data, we can employ state-of-the-art SE techniques [31], [32]. The challenge is to hide the popularity of objects and user preferences. When the CDN provider can tell whether any two users are requesting the same object or not, it can identify how many times each object has been requested by each user from the request history, even if the objects and requests are encrypted.

To hide object popularity and user preferences, we first use a semantically secure algorithm to encrypt user requests, making all the encrypted requests look different no matter if they are generated from the same query or not. However, a CDN provider could still infer whether two requests are the same if they match the same object. That is, based on the location of the requested object, a CDN provider can tell if the same object was requested twice or not. To address this issue, our basic idea is to migrate the matched object across a set of nodes owned by different CDN providers after each request. More specifically, we combine the nodes located in the same geographical region but owned by different CDN providers into a CDN cluster. When receiving a request, the nodes in a cluster search their storage one by one until the object is found. After returning the matched object to the user, the node caching the requested object migrates it to another node. In this way, whether this object will match future requests in other nodes is no longer known to the node.

Even with the idea of using multi-CDN nodes and migrating matched objects, a malicious node may still record the matched objects and check if they match future requests privately. Similarly, it could also keep previous requests and check if they match the new objects migrated from other nodes. So, the main challenge is how to ensure the forward and backward privacy (*i.e.,* a malicious node could neither check if the matched objects match a new request nor check if previous requests match the new objects migrated from other nodes) without increasing any overhead on users. To solve this problem, our idea is to encrypt each object with a one-off nonce and store the nonces and encrypted objects in two non-colluding nodes. For each request, the node could perform the search operation only when getting the one-off nonces from another node. After searching, the objects will be re-encrypted with new nonces before migrating to another node. This will protect matching information from potentially malicious nodes. More importantly, our design does not introduce any significant storage, computation, or bandwidth overheads to users. Because of the scheme used in our system, when a cluster forwards the encrypted request to the CP after a cache miss, the CP can perform the search over its local storage easily, just like searching over the objects in plaintext. With the use of the one-off nonces, we can securely revoke users without re-encrypting objects and distributing new keys.

For simplicity, Figure 1 illustrates the architecture of our system using only one CP and one multi-CDN cluster. It can be easily scaled to multiple CPs and clusters. First, the user gets the IP addresses of the closest node from RRS (Step 0). Next, the encrypted request is directed to the selected node (Step 1). Then, the node searches over the cached objects and returns the matched one if there is a hit (Step 2). Otherwise, the request will be forwarded to another node in this cluster until the object is found. However, if the object is not found, the request will be forwarded to the CP. The CP searches locally, encrypts the matched object and uploads it to the CDN node (Step I), and then the CDN node forwards this object to the user. After returning the matched object to the user, it will be re-encrypted and migrated to another CDN node (Step 3). The details of processing user requests are given in Section V.

IV. Data Representation

In this section, we explain how the object is represented and stored on the CDN cluster and the CP. For clarity, the notations used in this article are listed in Table I.

A. System Setup

We consider a CDN cluster with N nodes: $\mathcal{N}_1$, ..., $\mathcal{N}_N$, where $N \geq 3$ and $\mathcal{N}_i, \mathcal{N}_{i+1}$ $(N > i > 0)$, i.e., any two adjacent nodes, should be provided by the CDN providers in conflict of interest (discussed in detail in Section VI).

Let λ be the security parameter in our system. The system is set by the CP by generating the secret key k and configuring CDN clusters. k is shared among users and is used to protect the request and objects from CDN providers.

B. Data Representation

The object cached by CDN nodes (or hereafter nodes in short) could be a document, image, or video. In this work, we specifically consider the object as a digital document and model it as a 2-tuple $O = \langle name, payload \rangle$. Each object can be identified by its unique $name$, and both $name$ and $payload$ are represented as binary strings.

To hide the content of each object, it should be encrypted when caching in the CDN cluster. However, the CDN provider might still be able to distinguish the objects based on their sizes. This can be solved by padding (with fake data) all the objects up to a maximum size. However, holding the fake data might consume a large volume of storage on the nodes; in Section IX, we discuss how we can save storage. For clarity, in the following, we assume all the objects are of the same size.

Before uploading objects to CDNs, they should be encrypted by the CP. Formally, each object stored in CDNs is encrypted as:

$$EO = \langle EN \leftarrow H_k(name) \oplus n,\ EP \leftarrow f_k(payload) \oplus \eta \rangle$$

where $H_k : \{0,1\}^\lambda \leftarrow \{0,1\}^*$ is a keyed-hash function, $f_k : \{0,1\}^* \leftarrow \{0,1\}^*$ represents symmetric encryption primitives, such as AES, and n, η are two nonces. Here, we stress that, to achieve fine-grained access control over the objects, Attribute-Based Encryption (ABE) schemes, such as [33], [34], can be utilised to protect the payload.

The CP computes and locally stores:

$$PEO = \langle PEN \leftarrow H_k(name), PEP \leftarrow f_k(payload) \rangle$$

Once requested by a node, say $\mathcal{N}_1$, the CP XORs PEO with nonces $\langle n, \eta \rangle$ and sends it to $\mathcal{N}_1$. The pair of nonces $\langle n, \eta \rangle$ will be sent to $\mathcal{N}_2$. The size of nonce η is same as the payload of the object. To reduce the storage overhead on nodes, we could use a much shorter *seed* and a pseudo-random generator to generate a long bit-string as nonce η. In this case, $\mathcal{N}_2$ just needs to store the short *seed*, rather than the nonce.

At this stage, we re-encrypt objects with nonces for two main reasons. First, due to the nonces, our system achieves the

TABLE II: The storage on the CP and CDN nodes.

(a) Pre-encrypted objects on CP

id	objects
0	$\langle H_k(name_1),\, f_k(payload_1)\rangle$
1	$\langle H_k(name_2),\, f_k(payload_2)\rangle$
...	...

(b) Content Store (CS)

pid	id	objects
0	0	$\langle H_k(name_i)\oplus n_0,\, f_k(payload_i)\oplus \eta_0\rangle$
	...	...
	$W-1$	$\langle H_k(name_j)\oplus n_0,\, f_k(payload_j)\oplus \eta_0\rangle$
1	W	$\langle H_k(name_l)\oplus n_1,\, f_k(payload_l)\oplus \eta_1\rangle$
	...	...

(c) Nonce Store (NS)

pid	nonces
0	$\langle n_0,\, \eta_0\rangle$
1	$\langle n_1,\, \eta_1\rangle$
...	...

Table (a) stores the pre-encrypted objects. It is held by the CP. Table (b) stores the encrypted object, consisting of the encrypted name $EN \leftarrow H_k(name)\oplus n$ for search, and the encrypted payload $EP \leftarrow f_k(payload)\oplus \eta$ for data retrieval. The objects stored in this table are virtually divided into partitions. W is the size of each partition. The objects in the same partition are encrypted with the same nonces. Table (c) stores the nonces $\langle n,\eta\rangle$. Each node in a CDN cluster should store both Table (b) and Table (c). However, the nonces stored in $\mathcal{N}_{i+1}$'s Table (c) are those included in $\mathcal{N}_i$'s Table (b).

forward and backward privacy. Without the nonce provided by $\mathcal{N}_{i+1}$, the $\mathcal{N}_i$ can not perform any search operation independently (as described in Section V). Second, the nonce ensures that users can not recover the payload only with the secret key k. The nonces are generated by the CP and stored by $\mathcal{N}_{i+1}$, but they are unknown to users. Only with both the nonce η and the secret key k, the user could decrypt the object, which means we do not need to update the secret key k or re-encrypt all the cached objects in case of user revocation.

We assume that each object could be identified using a unique name. Hence, the objects can be encrypted with the same nonce. In order to reduce both storage and communication overheads (see Section V), every W objects cached in a node will be encrypted with the same nonce. The objects cached in each node are divided into partitions $\{P_0, P_1, \ldots\}$ based on their physical store location in the node, *e.g.,* the first W objects cached in a node is in the first partition P_1. As a result, $\mathcal{N}_{i+1}$ just needs to store a nonce pair $\langle n,\eta\rangle$ for each partition rather than each object.

C. Data Storage

Each node has two separate stores: Content Store (CS) and Nonce Store (NS), where the CS caches EO, and the NS caches the pair of nonces $\langle n,\eta\rangle$. Note that the nonces pairs associated with the EOs cached in N_i are stored in the NS of $\mathcal{N}_{i+1}$. For instance, $\mathcal{N}_2$ stores the nonce pairs associated with the objects stored in $\mathcal{N}_1$, and $\mathcal{N}_1$ stores the nonce pairs for $\mathcal{N}_N$.

Table II shows an example of the store in our system. Table II(a) is kept on the CP and stores all the pre-encrypted objects. Table II(b) and Table II(c) represent the CS and the NS cached in CDN nodes, respectively. More specifically, Table II(b) stores a set of the encrypted objects, consisting of the encrypted name EN for search and encrypted payload EP for data retrieval. Moreover, every W objects in order are in the same partition and encrypted with the same nonce. Table II(c) stores the pair of nonces $\langle n,\eta\rangle$ for each partition. The encrypted objects in CS are identified with unique ids, and the nonce pairs in the NS are identified with pids. Note that, the nonce pair $\langle n_{pid},\eta_{pid}\rangle$ stored in $\mathcal{N}_i$ is the one included in $P_{pid} = \{EO_{pid*W}, \ldots, EO_{pid*W+W-1}\}$ stored in $\mathcal{N}_{i-1}$, rather than $\mathcal{N}_i$.

V. Request Process

In this section, we show how the request is encrypted by users and processed by CDN nodes or the CP. Based on the cached location of requested objects, the requests are processed in different ways. Specifically, we give the details for the cases where the requested object is cached in the closest node to the user, not in the closest node and in the CP.

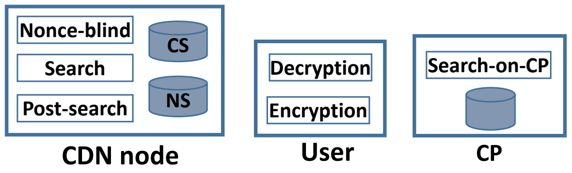


Fig. 2: The components of each entity: The user is provided with the *Encryption* and *Decryption* to encrypt requests and decrypt returned payloads. Each node $\mathcal{N}_i$ is equipped with *Nonce-blind*, *Search*, and *Post-search* to process encrypted requests. The CP is only provided with the *Search-on-CP* to provide the missing objects not stored by CDN clusters.

The components on each entity are shown in Fig. 2. Basically, the user is provided with the *Encryption* and *Decryption* to encrypt requests and decrypt returned payloads. Each node $\mathcal{N}_i$ is equipped with *Nonce-blind*, *Search*, and *Post-search* to process encrypted requests, where *Nonce-blind* is used to provide nonces for the search to be performed on $\mathcal{N}_{i-1}$, *Search* is used to search its CS, and *Post-search* is used to process the objects migrated from $\mathcal{N}_{i-1}$. The CP is only provided with the *Search-on-CP* to provide the missing objects not stored by CDN clusters. The implementation detail of each component is given below.

Algorithm 1 Encryption($name, \mathcal{N}_i, \mathcal{N}_{i+1}$)

```
1: $counter \leftarrow 0$
2: $ER_0 \leftarrow H_k(name)\oplus \alpha_0$, $\alpha_0$[1] $\xleftarrow{\$} \{0,1\}^\lambda$
3: Send the encrypted request $ER_0$ and $counter$ to $\mathcal{N}_i$, and send $\alpha_0$ to $\mathcal{N}_{i+1}$
```

Algorithm 2 Nonce-blind[2](α_{s-2})

```
1: $ENS \leftarrow \emptyset$
2: for each $n_{pid} \in NS$ do
3:     $ENS[pid] \leftarrow h(n_{pid}\oplus \alpha_{s-2})$
4: Send $ENS = \{h(n_1\oplus\alpha_{s-2}), h(n_2\oplus\alpha_{s-2}), \ldots\}$ to $\mathcal{N}_{s-1}$
```

A. Encryption on the User

Before issuing any request to the CDN cluster, the user first gets the IP addresses of the closest available nodes. In

[1] α_i means the nonce α is generated by $\mathcal{N}_i$. For instance, α_{s-2} is generated by $\mathcal{N}_{s-2}$. Particularly, α_0 is generated by the user.

[2] s identifies the host node running Algorithms 2, 3, and 4. For instance, when $\mathcal{N}_{i+1}$ running Algorithm 2, $s = i+1$.

Algorithm 3 Search($ER, counter$)

```
1: if $counter > 0$ then
2:     $ER \leftarrow ER \oplus \alpha_{s-2}$
3: Get $ENS$ from $\mathcal{N}_{s+1}$
4: for each $EO_{id} \in CS$ do
5:     $pid \leftarrow \lfloor \frac{id}{W} \rfloor$
6:     if $ENS[pid] = h(EN_{id} \oplus ER)$ then
7:         Send $EP_{id}$ to the user, and send $pid$ to $\mathcal{N}_{s+1}$
8:         $n' \xleftarrow{\$} \{0,1\}^{\lambda}$, $\eta' \xleftarrow{\$} \{0,1\}^{|payload|}$
9:         for each $EO_{id} \in P_{pid}$ do
10:            $EO'_{id} \leftarrow EO_{id} \oplus \langle n', \eta' \rangle$
11:        Send $P_{pid} = \{EO'_{pid*W}, \ldots, EO'_{pid*W+W-1}\}$ to $\mathcal{N}_{s+1}$
12:        Send $\langle n', \eta' \rangle$ to $\mathcal{N}_{s+2}$
13:        Remove all the objects in $P_{pid}$ from $CS$
14:        Exit
15: if the requested object is not found in CS then
16:     if $counter = N$ then
17:         Send $(ER, N)$ to the CP
18:     else
19:         $counter++$
20:         $ER_s \leftarrow ER \oplus \alpha_s$, $\alpha_s \xleftarrow{\$} \{0,1\}^{\lambda}$
21:         Send $(ER_s, counter)$ to $\mathcal{N}_{s+1}$
22:         Send $\alpha_s$ to $\mathcal{N}_{s+2}$
23:         if $counter = N$ then
24:             Send $\alpha_s$ to the CP
```

Algorithm 4 Post-search(P_{pid}, pid)

```
1: for each $EO'_{id} \in P_{pid}$ do
2:     $EO_{id} \leftarrow EO'_{id} \oplus \langle n_{pid}, \eta_{pid} \rangle$
3:     Insert $EO_{id}$ into the CS
4: Remove $\langle n_{pid}, \eta_{pid} \rangle$ from the NS
```

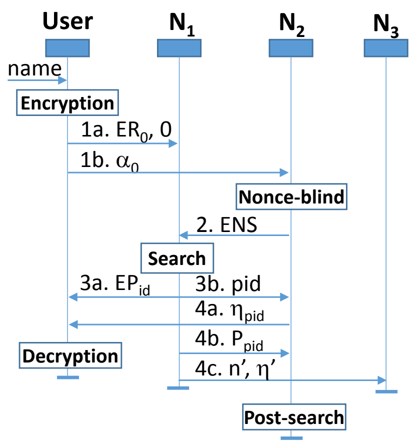


Fig. 3: The request process steps and the communications for the first round of search: The encrypted request ER_0 and $counter = 0$ are sent to the closest node $\mathcal{N}_1$ (Step 1a). The nonce used to randomise ER_0, α_0, is sent to $\mathcal{N}_2$ (Step 1b), with which $\mathcal{N}_1$ can generate ENS. The blinded nonces set ENS is sent to $\mathcal{N}_1$ (Step 2). Then, $\mathcal{N}_1$ performs the search. Once the requested object is found, it will be sent to the user (Step 3a). Meanwhile, the matched partition identifier pid is sent to $\mathcal{N}_2$ (Step 3b), with which $\mathcal{N}_2$ can return the related nonce η_{pid} to the user (Step 4a). At the same time, $\mathcal{N}_1$ re-randomises all the objects in the matched partition with new a nonce pair $\langle n', \eta' \rangle$ and migrates them to $\mathcal{N}_2$ (Step 4b). $\langle n', \eta' \rangle$ is sent to $\mathcal{N}_3$ (Step 4c).

this work, our system requires the cooperation of two adjacent nodes, and the user needs to communicate with both of them. Therefore, unlike traditional RRS, in our system IP addresses should be registered by pairs. One of them points to the node $\mathcal{N}_i$ storing the objects, and the other one points to the node $\mathcal{N}_{i+1}$ that stores the corresponding nonces.

The encryption algorithm implemented on the user end is described in Algorithm 1. To avoid an infinite loop, the user initialises a counter to record the number of times the request has been forwarded. The $name$ of the required object is hashed and encrypted with a nonce α_0. The nonce can ensure that the encrypted request ER_0 is semantically secure, which means the CDN provider can not tell whether the users are making requests for the same object or not. Finally, $(ER_0, counter)$ and the nonce α_0 are sent to $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$, respectively.

B. Search operation on $\mathcal{N}_i$

For clarity, we take $i = 1$ as an example, *i.e.,* assume $\mathcal{N}_1$ is the closest node to the user. Fig. 3 illustrates the request process steps and the communication between different entities when the requested object is cached in $\mathcal{N}_1$. In the following, we explain the procedures in details with Algorithms 2, 3, and 4.

Algorithm 5 Search-on-CP(ER, α_{i+N})

```
1: $ER \leftarrow ER \oplus \alpha_{i+N}$
2: for each $PEO_{id}$ do
3:     if $PEN_{id} = ER$ then
4:         Send $PEP_{id}$ to the user
5:         Ask $\mathcal{N}_{i+N}$ to send $lid$, the identifer of the last object in its CS
6:         if $(lid + 1)\%W > 0$ then
7:             Ask $\mathcal{N}_{i+N+1}$ to send the last $\langle n, \eta \rangle$ in its NS
8:         else
9:             $n \xleftarrow{\$} \{0,1\}^{\lambda}$, $\eta \xleftarrow{\$} \{0,1\}^{|payload|}$, send $\langle n, \eta \rangle$ to $\mathcal{N}_{i+N+1}$
10:        $EO_{id} \leftarrow PEO_{id} \oplus \langle n, \eta \rangle$, send $EO_{id}$ to $\mathcal{N}_{i+N}$
11:        Exit
```

First, node $\mathcal{N}_1$ receives the encrypted request ER_0 and $counter$ from the user (Step 1a, Fig. 3). Considering both the request and objects are encrypted with nonces, $\mathcal{N}_1$ alone would not be able to check if there is a match. It needs $\mathcal{N}_2$'s assistance. Specifically, after receiving α_0 from the user (Step 1b, Fig. 3), $\mathcal{N}_2$ executes *Nonce-blind* (Algorithm 2) to blind all the nonces stored in the NS. Formally, for each n_{pid}, it computes:

$$ENS[pid] \leftarrow h(\alpha_0 \oplus n_{pid})$$

where $h : \{0,1\}^* \rightarrow \{0,1\}^{\lambda}$ is a hash function. The set ENS is sent to $\mathcal{N}_1$ for search (Step 2, Fig. 3). If $counter < N$, $\mathcal{N}_1$ searches over its CS to check if there is a hit with *Search* component (Algorithm 3). Specifically, for each object EO_{id} stored in CS, $\mathcal{N}_1$ checks if it matches the request (Line 6, Algorithm 3). To be more specific, $\mathcal{N}_1$ checks if:

$$h(\alpha_0 \oplus n_{pid}) \stackrel{?}{=} h(H_k(name) \oplus \alpha_0 \oplus H_k(name_{id}) \oplus n_{pid})$$

It is clear that there is a match when $name = name_{id}$. Once there is a match, $\mathcal{N}_1$ stops the search, returns the matched EP_{id} to the user (Step 3a, Fig. 3), and notifies $\mathcal{N}_2$ to send the η_{pid} to the user by sending the matched pid to it (Step 3b and Step 4a, Fig. 3). Subsequently, all the records in the matched partition are re-encrypted with a new nonce pair $\langle n', \eta' \rangle$ (Line 10, Algorithm 3) and sent to $\mathcal{N}_2$ (Step 4b, Fig. 3). Meanwhile, the new nonce pair $\langle n', \eta' \rangle$ is sent to $\mathcal{N}_3$ (Step 4c, Fig. 3). Afterwards, all the records in the matched partition are removed from $\mathcal{N}_1$. Consequently, whether these objects will match future requests will be unknown to $\mathcal{N}_1$.

If $\mathcal{N}_2$ receives the migrated partition P_{pid}, it carries out *Post-search* (Algorithm 4). Specifically, $\mathcal{N}_2$ removes the old

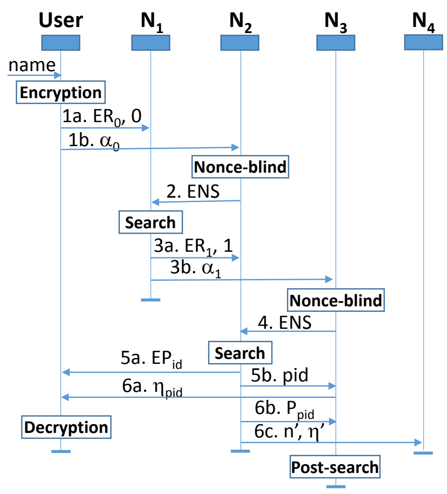


Fig. 4: The request process steps and the communication for the second round of search: When the requested object is not cached in $\mathcal{N}_1$, the request will be re-randomised into ER_1 with a new nonce α_1. ER_1 and $counter = 1$ are sent to $\mathcal{N}_2$ for the second round of search (Step 3a). Meanwhile, α_1 is sent to $\mathcal{N}_3$ for blinding the nonces stored in $\mathcal{N}_3$ (Step 3b). Then, $\mathcal{N}_2$ repeats the search over its NS as $\mathcal{N}_1$ did (Steps 4-6c). Note that when N=3, $\langle n', \eta' \rangle$ is sent to $\mathcal{N}_1$ (Step 6c).

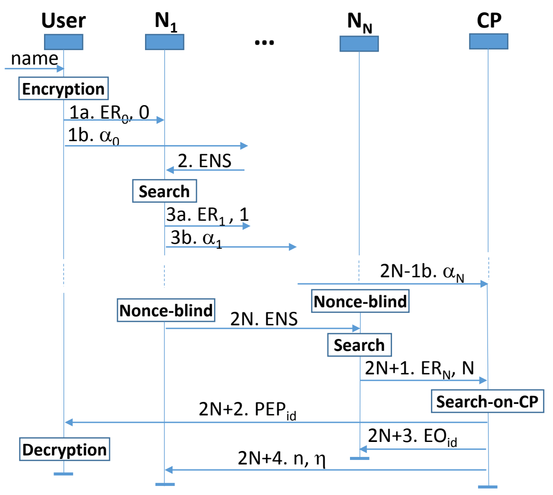


Fig. 5: The request process steps and the communication for the search on the CP: When the requested node is not cached in the CDN cluster, the request ER_N and the nonce α_N are forwarded to the CP (Step $2N+1$ and $2N-1$). The CP searches its local storage and returns the matched object to user (Step $2N+2$). Afterwards, the requested object is also re-randomised with a nonce pair and sent to $\mathcal{N}_N$ (Step $2N+3$), and the nonce pair is sent to $\mathcal{N}_1$ (Step $2N+4$).

nonces $\langle n_{pid}, \eta_{pid} \rangle$, which are stored in its NS, from each object in P_{pid} with XOR operation (Line 2, Algorithm 4). Consequently, they are only bound to the new nonce pair $\langle n', \eta' \rangle$. Finally, $\mathcal{N}_2$ inserts P_{pid} into its CS and removes $\langle n_{pid}, \eta_{pid} \rangle$ from its NS.

C. *Search operation on $\mathcal{N}_{i+1}$*

Instead, as shown in Fig. 4, if the requested object is not cached on $\mathcal{N}_1$, the request will be re-encrypted with a new nonce α_1 (Line 20, Algorithm 3) and forwarded to $\mathcal{N}_2$ for the second round of search (Step 3a, Fig. 4). Formally, the re-encrypted request is:

$$ER_1 = H_k(name) \oplus \alpha_0 \oplus \alpha_1$$

Meanwhile, if $counter < N$, α_1 is sent to $\mathcal{N}_3$ (Step 3b, Fig. 4), with which $\mathcal{N}_3$ could help $\mathcal{N}_2$ to perform the second round of search operation by running *Nonce-blind* and generating ENS (Step 4, Fig. 4).

Unlike $\mathcal{N}_1$, $\mathcal{N}_2$ first removes α_0 from ER_1 (Line 2, Algorithm 3). Recall that $\mathcal{N}_2$ has already received α_0 from the user. Then, with ENS, $\mathcal{N}_2$ could perform the second round of search over its CS as $\mathcal{N}_1$ did. The same operations will be performed on the rest nodes until the requested object is found or $counter = N$.

D. *Search operation on the CP*

If the requested object is not found when $counter = N$, *i.e.,* all the nodes have been searched, the request will be forwarded to the CP. The message flow for the search on CP is shown in Fig. 5.

Formally, the CP receives $ER_N = H_k(name) \oplus \alpha_N$ and α_N from $\mathcal{N}_N$ and $\mathcal{N}_{N-1}$ (Step $2N - 1b$ and $2N + 1$, Fig. 5), respectively. By XOR-ing α_N with ER_N, the CP could get $H_k(name)$. Then, the CP searches its pre-encrypted store. Considering the hash function H is deterministic, the search operation over pre-encrypted store can be performed like the search on plaintext domain, where the CP just checks if:

$$H_k(name) \overset{?}{=} H_k(name_{id})$$

The matched object is first sent to the user (Step $2N+2$, Fig. 5), and then it will be re-encrypted with nonces and sent to $\mathcal{N}_N$ (Step $2N+3$, Fig. 5). Depending on the location where the new replica will be cached in, the nonce pair used to encrypt it can be retrieved in two different ways. If the new replica will be part of an existing partition, the CP needs to get the corresponding nonce pair from $\mathcal{N}_1$ (Line 7, Algorithm 5). Otherwise, the CP generates a new pair and sends it to $\mathcal{N}_1$ (Line 9, Algorithm 5).[3] Moreover, the partition containing the matched object should be re-encrypted and migrated to $\mathcal{N}_1$ (Step $2N+4$, Fig. 5).

E. *Decryption*

If the requested object is found in the cluster, the user will receive $EP_{id} = f_k(payload) \oplus \eta_{pid}$ and η_{pid}. It can easily get $f_k(payload_{id})$ by XOR-ing them. On the contrary, if the requested object is found in the CP, the user could get $f_k(payload)$ directly. It can finally decrypt the matched payload by executing f_k^{-1} with the secret key k.

F. *Performance Optimisation using TTL*

A high cache hit rate means the requested objects are found after searching a certain number of nodes in a cluster. To achieve this, the CPs upload more replicas of the most popular

[3]The matched object and nonce can also be sent to the user by $\mathcal{N}_N$ and $\mathcal{N}_1$.

objects to the CDN. Here, we give an approach where the system could adaptively load multiple replicas of popular objects to the CDN cluster without revealing sensitive information to the CDN provider. Basically, to decrease response time, we set a Time To Live (TTL) value to limit the lifetime of requests. The TTL value determines the number of nodes a request should traverse. During a search, if the TTL expires before the object is found, the request will be forwarded to the CP. When $TTL < N$, over time, multiple replicas for the most popular objects will be cached in the CDN cluster. Consequently, the cache hit rate is improved.

G. User Revocation

Because of the nonces bound to EO, without the assistance of the CDN providers, the revoked user is unable to access and recover the $payload$. Therefore, for user revocation, we just need to manage a revoked user list at each node. Once a user is revoked, the CP informs them to add this user to their revoked user list. When receiving a request, the node will first authenticate the user and check if it has been revoked. If yes, they will reject the request. Over time, the revoked list might be very long and hard to manage. To solve this issue, we set a threshold, and when the list of revoked users reaching the threshold, the key k will be regenerated, and the objects will be re-encrypted with the new k.

VI. Security Analysis

In this work, we aim to provide confidentiality of objects and requests. Due to the encryption over the objects and request, the CDN provider can not learn the content of the object and request directly from their ciphertext. However, by analysing the request history, the CDN providers can get the popularity of objects and user preferences. According to [6]–[9], based on statistical information and given related background knowledge about the users and the CP, the CDN provider is still able to infer the content of the objects and requests when mounting inference attacks. To solve this issue, our solution is to hide the object popularity and user preferences by migrating the objects among the CDN nodes that are in conflict of interest and ensuring each object is only requested once in the view of each CDN node.

In this section, we show how our solution protects the object popularity and user preferences from CDN providers. Basically, we give formal definitions of object popularity privacy and user preference privacy, forward and backward privacy, and briefly analyse how our system achieves them. We also analyse possible collusion attacks in our system.

A. Object Popularity and User Preferences

Informally, given a period of time, user preferences mean which objects have been requested and what is the frequency of each one for each user, and the popularity of an object means the number of times it has been requested by users. User preferences mean which objects have been requested and what is the frequency of each one for each user. Formally, we define the user preference privacy and object popularity privacy as follows:

Definition 1. *(User Preference Privacy) At time t, assume $\mathcal{N}_i$ has received a set of requests $\{ER_1, \dots, ER_t\}$ issued by a user. We say that the user's preferences are protected if $\mathcal{N}_i$ can not infer whether any two requests, ER_μ and ER_ν, are generated from the same* name *with the probability greater than $\frac{1}{2} + \sigma(\lambda)$, where $0 < \mu < \nu \leq t$.*

Definition 2. *(Object Popularity Privacy) At time t, assume $\mathcal{N}_i$ has returned t requested objects $\{EO_1, \dots, EO_t\}$ to users. We say the object popularity privacy is achieved if $\mathcal{N}_i$ can not infer whether any two objects, EO_μ and EO_ν, are the same object with the probability greater than $\frac{1}{2} + \sigma(\lambda)$, where $0 < \mu < \nu \leq t$.*

Claim 1. *If our system achieves both forward and backward privacy and each node can only see a 1-time match for each object, it also hides user preferences and the popularity of objects.*

Proof. **User Preference Privacy.** According to the defined protocol in Section V, each node $\mathcal{N}_i$ gets the encrypted request ER from the user or $\mathcal{N}_{i-1}$, a set of nonces ENS from $\mathcal{N}_{i+1}$, and the matching object EO when processing a request. The CDN node can infer user preferences only based on the messages it receives and the data it stores. In the following, we will prove the CDN node can not infer if ER_μ and ER_ν are the same request or not from any one of them.

First of all, due to the nonces, ER_μ and ER_ν are semantically secure, and $\mathcal{N}_i$ learns nothing from the ciphertext of ER_μ and ER_ν.

Second, it can not infer whether ER_μ and ER_μ are generated from the same *name* or not by comparing their matching objects. Assume EO_μ and EO_ν are the matching objects of ER_μ and ER_ν, respectively. Likewise, because of the nonce pairs, $\mathcal{N}_i$ learns nothing from their ciphertext, but $\mathcal{N}_i$ could also compare their physical locations.

In terms of their store locations, there are 4 possible cases: (i) both EO_μ and EO_ν are found in $\mathcal{N}_i$; (ii) neither EO_μ nor EO_ν is found in $\mathcal{N}_i$; (iii) only EO_μ is found in $\mathcal{N}_i$; (iv) or only EO_ν is found in $\mathcal{N}_i$.

The first case can be divided into two sub cases: EO_ν is already cached in $\mathcal{N}_i$ before processing ER_μ and EO_ν is migrated to $\mathcal{N}_i$ after processing ER_μ. In the first sub case, $\mathcal{N}_i$ could check whether ER_μ matches EO_ν or not, and then infer the relationship between ER_μ and ER_ν. However, in the second sub case, $\mathcal{N}_i$ can not tell whether both ER_μ and ER_ν are same or not, since our scheme achieves both forward and backward privacy, *i.e.,* it can not check if ER_μ matches with EO_ν or ER_ν matches with EO_μ. Moreover, $\mathcal{N}_i$ is unable to tell whether EO_μ and EO_ν are the same object or not because of the nonce.

In the second case, after the searching on $\mathcal{N}_i$, both ER_μ and ER_ν will be re-encrypted and forwarded to other nodes for further search operations. Physical locations of both EO_μ and EO_ν are unknown to $\mathcal{N}_i$.

If only EO_μ is cached in $\mathcal{N}_i$, after the search for ER_μ, EO_μ will be re-encrypted and migrated to $\mathcal{N}_{i+1}$. On the contrary, after the search for ER_ν, ER_ν will be re-encrypted and forwarded to $\mathcal{N}_{i+1}$ since EO_ν is not cached in $\mathcal{N}_i$. Whether the re-encrypted EO_μ will match ER_ν on $\mathcal{N}_{i+1}$ is unknown to $\mathcal{N}_i$. $\mathcal{N}_i$ may keep the stale EO_μ, but it can not check if it matches ER_ν since our scheme achieves forward privacy. So $\mathcal{N}_i$ is unable to infer the relationship between ER_μ and ER_ν in this case.

In the last case (where EO_ν is cached in $\mathcal{N}_i$ but EO_μ is not), if EO_ν is already cached in $\mathcal{N}_i$ when processing ER_μ, $\mathcal{N}_i$ could learn if ER_μ and ER_ν are different requests since EO_ν matches EO_ν but does not match EO_μ. However, if EO_ν is migrated to $\mathcal{N}_i$ later after executing ER_μ, it is difficult to tell if ER_ν is same to ER_μ or not, since our system achieves backward privacy and $\mathcal{N}_i$ can not repeat ER_μ over ER_ν.

In summary, only in some special cases, $\mathcal{N}_i$ could learn if ER_μ and ER_ν are different. However, it can never learn if ER_μ and ER_ν are the same request.

Note that even if a malicious node actively injects an object into the system, it can only infer the content of one request. Once there is a hit, the injected object will be re-encrypted and migrated to other nodes, and then whether it will match other requests is unknown to the malicious node.

Object Popularity Privacy. Considering $\mathcal{N}_i$ could only see a 1-time match for each encrypted object, it can only try to infer the popularity information by checking if the returned encrypted objects have the same content or not. Unfortunately, due to the nonces, EO_μ and EO_ν are semantically secure, and $\mathcal{N}_i$ is unable to tell if EO_μ and EO_ν are generated from the same object or not. In $\mathcal{N}_i$, only the objects in the same partition are encrypted with the same nonce. However, if both EO_μ and EO_ν are found in $\mathcal{N}_i$, they should be located in different partitions. Otherwise, after EO_μ is found, EO_ν will also be re-encrypted and migrated to $\mathcal{N}_{i+1}$, which contradicts our assumption.

$\mathcal{N}_i$ could check if EO_μ and EO_ν are the same or not only by observing if they match the same request or not. As mentioned above, due to both forward and backward privacy, $\mathcal{N}_i$ could learn if EO_μ and EO_ν are different objects when EO_ν is already cached by $\mathcal{N}_i$ and EO_μ matches ER_μ. However, in all cases, $\mathcal{N}_i$ can not tell whether EO_μ and EO_ν are the same. □

Theorem 1. *When $W > 1$, in our system, each node can only see a 1-time match for each object.*

Proof. Based on the description in Section V, the node $\mathcal{N}_i$ only communicates with $\mathcal{N}_{i-2}$, $\mathcal{N}_{i-1}$, $\mathcal{N}_{i+1}$, and $\mathcal{N}_{i+2}$. $\mathcal{N}_i$ can only try to infer how many times the object has been requested based on the objects and nonce pairs migrated from or to these nodes, since only the matched partition should be migrated. However, from the messages get from other nodes, $\mathcal{N}_i$ is unable to infer the popularity of objects. In the following, we analyse the communication between $\mathcal{N}_i$ and the other four nodes one by one.

Recall the protocol described in Section V, (i) when there is a match in $\mathcal{N}_{i-2}$, $\mathcal{N}_i$ will get a pair of nonce from $\mathcal{N}_{i-2}$, which is used to re-encrypt the matched partition, say P_{pid_1}. (ii) when there is a match in $\mathcal{N}_{i-1}$, $\mathcal{N}_i$ will get the re-encrypted matched partition, say P_{pid_2}, from $\mathcal{N}_{i-1}$. (iii) When there is a match in $\mathcal{N}_i$, $\mathcal{N}_i$ could learn which object in the matched partition, say P_{pid_3}, has been requested. Moreover, P_{pid_3} will be migrated to $\mathcal{N}_{i+1}$, and the nonce pair used to re-encrypt P_{pid_3} is sent $\mathcal{N}_{i+2}$. In the first two cases, $\mathcal{N}_i$ could only know there is a match happened in P_{pid_1} and P_{pid_2}, but it can not learn which objects in these two partitions were requested. After the searching on $\mathcal{N}_i$, what will happen on P_{pid_3} is unknown to $\mathcal{N}_i$, since it has been migrated to $\mathcal{N}_{i+1}$. Therefore, $\mathcal{N}_i$ could only see a 1-time match for each object, and only when $pid_1 = pid_2 = pid_3$ and the partition size is 1, $\mathcal{N}_i$ could see a 3-time match. Therefore, when the partition size is 1, we should ensure $N > 3$ and $\mathcal{N}_i$, $\mathcal{N}_{i+1}$, $\mathcal{N}_{i+2}$, $\mathcal{N}_{i+3}$ are provided by different CDN providers. However, when setting a big partition size, we just need to ensure $N \geq 3$ and any two adjacent nodes are provided by different CDN providers. □

B. Forward and Backward Privacy

As mentioned in Section III-B, we assume the CDN node only honestly follows the specified protocol, but it could snapshot stale objects and requests, and try to execute stale requests over newly added objects or execute new requests over stale objects in private. To protect popularity of objects and user preferences, it is necessary to ensure that the CDN node can not check if stale objects match new requests or not or if newly added objects match stale requests or not. The formal definition for forward and backward privacy is given below:

Definition 3. *(Forward and Backward Privacy) Assume EO_t and EO_τ are the objects matching ER_t and ER_τ, respectively, where $\tau < t$ (i.e.,ER_τ is processed after ER_t). Moreover, assume EO_τ is uploaded to $\mathcal{N}_i$ after executing ER_t. We say forward and backward privacy is achieved if $\mathcal{N}_i$ can not tell whether EO_t matches ER_τ or whether ER_t matches EO_τ, respectively, with the probability greater than $\frac{1}{2} + \sigma(\lambda)$, where $\sigma(\lambda)$ is a negligible function.*

Theorem 2. *If all the nonces are sampled randomly, and if any two CDN providers do not collude together, our system achieves both forward and backward privacy.*

Proof. Formally, assume

$$\begin{aligned} EO_t =& \langle H_k(name_t) \oplus n_t, f_k(payload_t) \oplus \eta_t \rangle \\ EO_\tau =& \langle H_k(name_\tau) \oplus n_\tau, f_k(payload_\tau) \oplus \eta_\tau \rangle \\ ER_t =& H_k(name_t) \oplus \alpha_t \\ ER_\tau =& H_k(name_\tau) \oplus \alpha_\tau \end{aligned}$$

According to the scheme, once EO_t is found for ER_t, it will be re-encrypted into $EO_t' = H_k(name_t) \oplus n_t \oplus n'$ and migrated to $\mathcal{N}_{i+1}$, and the new nonce n' is sent to $\mathcal{N}_{i+2}$. After receiving EO_t', $\mathcal{N}_{i+1}$ removes n_t from EO_t' and its NS. When executing ER_τ, $\mathcal{N}_i$ gets ENS from $\mathcal{N}_{i+1}$, and then it can check which object matches ER_τ. However, $\mathcal{N}_i$ never gets $h(n' \oplus \alpha_\tau)$ or $h(n_t \oplus \alpha_\tau)$. Furthermore, because of the one-time pad encryption with the nonce, it can not tell if

$$H_k(name_t) \oplus n_t \overset{?}{=} H_k(name_\tau) \oplus \alpha_\tau$$

or

$$H_k(name_t) \oplus n' \overset{?}{=} H_k(name_\tau) \oplus \alpha_\tau$$

indicating forward privacy is achieved.

Similarly, when EO_τ migrates to $\mathcal{N}_i$, $\mathcal{N}_i$ can not check if it matches ER_t or ER'_t, since $\mathcal{N}_i$ never gets $h(\alpha_t \oplus n_\tau)$ or $h(\alpha_t \oplus \alpha' \oplus n_\tau)$, meaning backward privacy is achieved. □

C. Collusion Attacks

In our system, if $\mathcal{N}_i$ colludes with $\mathcal{N}_{i+1}$, they can remove the nonces from encrypted requests and objects with XOR operation. Although they can not recover the plaintext of the object without the secret key k, they can partially learn the object popularity and user preferences. Furthermore, if they collude with a malicious user, all the cached objects can be decrypted. To resist these attacks, we give two countermeasures as follows.

One possible solution is to hide the CDN providers from each other, and employ a trusted proxy in each cluster to manage the communication between any two different nodes. Specifically, each CDN node only communicates with the proxy, and the proxy forwards the message to other nodes. Another possible solution is to encrypt both the name and payload with two nonces. Formally, the encrypted object would be $EO = \langle H_k(name) \oplus n_1 \oplus n_2, f_k(payload) \oplus \eta_1 \oplus \eta_2 \rangle$. The nonces $\langle n_1, \eta_1 \rangle$ and $\langle n_2, \eta_2 \rangle$ will be stored in $\mathcal{N}_{i+1}$ and $\mathcal{N}_{i+2}$, respectively. Similarly, users also encrypt the request with two nonces, *i.e.,* $ER = H_k(name) \oplus \alpha_1 \oplus \alpha_2$. α_1 is sent to $\mathcal{N}_{i+1}$ for encrypting $\langle n_1, \eta_1 \rangle$, and α_2 is sent to $\mathcal{N}_{i+2}$ for encrypting $\langle n_2, \eta_2 \rangle$. In this case, $\mathcal{N}_i$ needs the assistance of $\mathcal{N}_{i+1}$ and $\mathcal{N}_{i+2}$ to process a request. Due to the nonces stored on the third CDN provider, the collusion between any two CDN providers could learn nothing.

Moreover, if the revoked or malicious users collude with one or more CDN nodes, they could access unauthorised objects. In our system, all the cached objects are encrypted with the secret key and nonces. Only when the colluded entities have both of them, they can recover the content of those objects. If the user colludes only with one node, she can recover a partition of objects. Specifically, after getting the requested object, the user also gets the nonce to decrypt the payload, which is also used to encrypt other objects in the same partition. If the CDN provider sends the partition to the user, she is able to recover all the objects in this partition. This issue can be solved by setting the partition size as low as possible (say 1), but with a performance tradeoff. In contrast, if the user colludes with both $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$, she can recover all the objects cached in $\mathcal{N}_i$. To resist this collusion attack, ABE [33], [34] schemes can be integrated into our system to protect the object payload.

VII. Computation Complexity Analysis

In this part, we empirically analyse the costs associated with each component. Let λ and ρ be the bit lengths of object name and payload, respectively. We use $\mathbf{H}_\lambda$, $\mathbf{X}_\lambda$ and $\mathbf{C}_\lambda$ to represent the computation overhead of hashing, XORing and comparing λ bits data. Moreover, $\mathbf{f}_\rho^{-1}$ represents the computation overhead of decrypting a ρ-bit object. The computation and communication overheads on each component are shown in Table III.

A. Encryption and Decryption

To issue a request, as illustrated in Algorithm 1 (see Section V), the user just needs to perform a hash and an XOR to encrypt the request, and sends the encrypted request (λ bits) and nonce (λ bits) to the CDN cluster.

Once the requested object (ρ bits) and the corresponding nonce η (ρ bits) are received, the user could recover it by performing an XOR and f^{-1} operations.

Hence, the overall computation overhead on the user side is: $\mathbf{H}_\lambda + \mathbf{X}_\lambda + \mathbf{X}_\rho + \mathbf{f}_\rho^{-1}$. Moreover, the user needs to send 2λ-bits data to CDN nodes, and get 2ρ bits payload and nonce from the CDN.

B. Nonce Blinding

To assist the search operation on $\mathcal{N}_i$, $\mathcal{N}_{i-1}$ needs to provide the blinded nonce for each partition by running the *Nonce-blind* component. Assume P is the number of partitions stored $\mathcal{N}_i$. From Algorithm 2, we can infer that the computation overhead of this component is: $P(\mathbf{H}_\lambda + \mathbf{X}_\lambda)$. That is, each nonce n is XORed with another nonce α and hashed.

Moreover, the blinded nonce ($|\mathbf{H}|$ bits) for each partition should be sent to $\mathcal{N}_i$. Therefore, the communication overhead is $P|\mathbf{H}|$ bits, where $|\mathbf{H}|$ represents the output size of the hash function.

C. Search

Recall that each partition contains W objects. When there are P partitions, PW objects are stored on $\mathcal{N}_i$. To check if an object is the requested one, a hash and a comparison operation should be performed. Currently, we only support linear search, but once the requested object is found, the search will stop. In the worst case, *i.e.,* the requested object is not cached or is the last one, the search computation overhead is $PW(\mathbf{H}_\lambda + \mathbf{X}_\lambda + \mathbf{C}_\lambda)$.

When there is a hit, the matched object (ρ bits) will be sent to the user. Moreover, the matched partition should be re-randomised with a new nonce pair and migrated to the next node. That is, one more XOR operation should be performed over each object in the matched partition. Meanwhile, the re-randomised partition ($W(\rho + \lambda)$ bits) and the new nonce pair ($(\rho + \lambda)$ bits) should be sent to $\mathcal{N}_{i+1}$ and $\mathcal{N}_{i+2}$, respectively. Then, the overall computation overhead on the search component is $PW(\mathbf{H}_\lambda + \mathbf{X}_\lambda + \mathbf{C}_\lambda) + W(\mathbf{X}_\lambda + \mathbf{X}_\rho)$, and the communication overhead is $(W+1)\lambda + (W+2)\rho$ bits.

On the contrary, if the requested object is not found, the request (λ bits) will be XORed with a new nonce and forwarded to the next node. Meanwhile, the new nonce (λ bits) is sent to $\mathcal{N}_{i+2}$. In this case, the overall computation overhead on the search component is: $PW(\mathbf{H}_\lambda + \mathbf{X}_\lambda + \mathbf{C}_\lambda) + \mathbf{X}_\lambda$, and the communication overhead is 2λ bits.

Note that, if the request is from a node, rather than the user, one more XOR operation is performed over the request (Line 2, Algorithm 3).

TABLE III: The computation and communication overhead on each component.

Operation	Computation	Communication
Encryption	$\mathbf{H}_\lambda + \mathbf{X}_\lambda$	2λ bits
Nonce-blind	$P(\mathbf{H}_\lambda + \mathbf{X}_\lambda)$	$P\vert H\vert$ bits
Search (found)	$PW(\mathbf{H}_\lambda + \mathbf{X}_\lambda + \mathbf{C}_\lambda) + W(\mathbf{X}_\lambda + \mathbf{X}_\rho)$	$(W+1)\lambda + (W+2)\rho$ bits
Search (not found)	$PW(\mathbf{H}_\lambda + \mathbf{X}_\lambda + \mathbf{C}_\lambda) + \mathbf{X}_\lambda$	2λ bits
Post-search	$W(\mathbf{X}_\lambda + \mathbf{X}_\rho)$	ρ bits
Search-on-CP (found)	$T\mathbf{C}_\lambda + \mathbf{X}_\lambda + \mathbf{X}_\rho$	$2\lambda + 3\rho$ bits
Search-on-CP (not found)	$T\mathbf{C}_\lambda$	0
Decryption	$\mathbf{X}_\rho + \mathbf{f}_\rho^{-1}$	0

λ and ρ are the bit length of object name and payload, respectively. P is the number of partitions in the node, *i.e.,* PW objects are stored on the node. T is the total number objects stored on the CP. $\mathbf{H}_\lambda$, $\mathbf{X}_\lambda$, and $\mathbf{C}_\lambda$ represent the computation overhead of hashing, XORing, and comparing λ bits data, respectively. Moreover, $\mathbf{f}_\rho^{-1}$ stands for the computation overhead of decrypting a ρ-bit object.

D. Post Searching

In *Post-search*, as shown in Algorithm 4, the main operation is to remove the old nonce from each object in the migrated partition. Thus, the computation overhead on this component is: $W(\mathbf{X}_\lambda + \mathbf{X}_\rho)$.

At the same time, the nonce η (ρ bits) of the matched partition should be sent to the user for decryption.

E. Search on the CP

Assume T is the number of objects stored on the CP. The search operation on the CP is also linear. Since the encrypted object is deterministic, the CP just needs to compare if the encrypted request equal to the encrypted names, and before that the CP also removes the nonces from the request with an XOR operation. As done in CDN node, the CP will stop the search once the requested object is found. In the worst case, the search overhead on the CP is: $T\mathbf{C}_\lambda$.

Once the requested object is found, it will be returned to the user. Meanwhile, its name and payload will be XORed with a pair of nonces and sent to the CDN. In this case, the computation overhead is: $T\mathbf{C}_\lambda + \mathbf{X}_\lambda + \mathbf{X}_\rho$, and the communication overhead is $2\lambda+3\rho$ no matter if the nonce pair is new or get from the CDN. Otherwise, no XOR operation and communication are required.

VIII. Performance Analysis

A. Experiment Setup

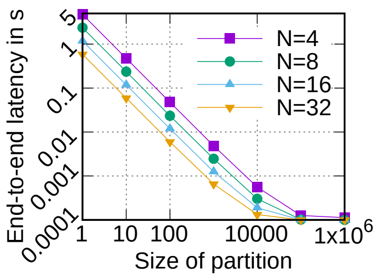
We implemented the scheme in C using MIRACL 7.0 library for cryptographic primitives. The *payload* of each object was 1024 bytes and encrypted using AES-CBC. The *name* of each object was 64 bytes and hashed using SHA-256, and the encrypted nonce was hashed with SHA-128. The performance was evaluated on a Sandy Bridge based cluster containing 236 nodes, where each node contains two Intel E5-2680 2.7GHz processors with 128GB of RAM. In the test, each CDN node was deployed on a separate node in the cluster. The network bandwidth between nodes was simulated with 7 Mbps, which is the latest global Internet average connection speed reported by Akamai [35]. Meanwhile, we simulated the round trip network latency between the node and the CP as 0.3 second (s), which is the largest measured value between Australia and UK reported by Verizon [36]. Considering the user and the CDN cluster are generally located in the same region, the latency between the user and CDN node is assumed to be 33 milliseconds (ms), which is the global average network latency in the US measured by AT&T [37].

In the experiments, the CP owns 1,000,001 different objects in total, and the encrypted replicas of the first 1 million are distributed to these nodes uniformly. There is only one replica cached in the CDN cluster for each object. According to the cached location of the requested object, we tested the performance of the system in three different cases: best, worst, and average. The best case means the first object stored in the selected node is the requested one. In the average case, 100 random requests were generated and the requested objects were cached in the CDN cluster uniformly. In the worst case, the requested object was not cached in the CDN cluster, but was the last object stored on the CP. Moreover, according to the realistic distribution of the nodes provided by existing CDN enterprises all over the world [38], the number of nodes currently deployed in each country ranges from 1 to 47. In the following, all the data points in the graphs were averaged over 100 trials.

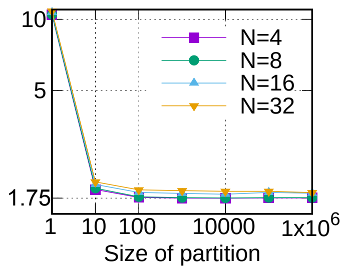
B. End-to-end Latency on the User

Our objective is to protect the objects cached by the CDN and to preserve user privacy while ensuring system usability. As for system usability, we should, first of all, ensure that the user should get the requested object in a reasonable time. So, we first tested the end-to-end latency on the user side. By the end-to-end latency, we mean the time spent by the user to get the requested object, including the request encryption time, the search time on nodes and/or the CP, and network latency between any two involved entities.

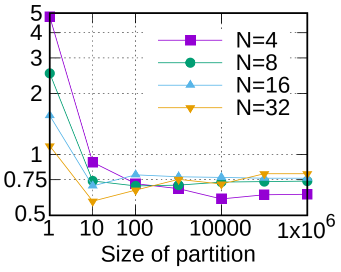
Figure 6 presents the effect on end-to-end latency in three (best, worst and average) cases when changing the size of each partition and the number of nodes deployed in the CDN cluster. Recall that a partition represents a group of objects that use the same nonce pair. In the best case (see Figure 6(a)), it is clear that the end-to-end latency goes down linearly with the increase in partition size. In our experimentation, we assume the selected node is $\mathcal{N}_i$. Although in the best case only one object is searched in $\mathcal{N}_i$, as described in Section V, $\mathcal{N}_{i+1}$ has to encrypt the nonces stored in its NS and send them to $\mathcal{N}_i$. That is, in the best case, the time is mainly consumed by *Nonce-blind* and transmission. Recall that $\mathcal{N}_{i+1}$ stores a pair of nonces for each partition stored in $\mathcal{N}_i$. When the number of objects cached in $\mathcal{N}_i$ is fixed, the bigger size of each partition means the fewer nonces should be stored in $\mathcal{N}_{i+1}$, and the fewer nonces should be encrypted and sent to $\mathcal{N}_i$. That is why,



(a) Best case



(b) Worst case



(c) Average case

Fig. 6: End-to-end latency on users.

the end-to-end latency becomes lower when the partition size gets bigger. Similarly, the end-to-end latency also decreases with the increase in the number of nodes when both the total number of objects owned by the CP and the partition size are fixed. This is because the more nodes we have, the fewer objects and nonces will be cached in each node, and the fewer nonces should be encrypted and transferred.

In contrast, in the worst case (see Figure 6(b)), all the nodes in the CDN cluster and the CP searched their storage one by one, meaning 2,000,001 objects were searched. So, the end-to-end latency in this case is much higher than that of the best case. From Figure 6(b), we can see that the end-to-end latency drops down significantly when changing the partition size from 1 to 10; however, there is no obvious decline when reducing the partition size further. That is because the partition size only affects the number of nonces that are encrypted and transferred, but it does not affect the number of objects that are searched. No matter what the partition size is, the number of objects that should be searched is always $2,000,001$. When the partition size ≥ 10, the time is mainly consumed by the search operation, rather than the nonce encryption and transmission. Another thing we can notice from the worst case is that the latency increases slightly when an increase in the number of nodes. The reason is after searching on $\mathcal{N}_i$, the request has to be re-encrypted before forwarding to $\mathcal{N}_{i+1}$, and more nodes means more request re-encryptions should be performed.

The best and worst cases are two extreme situations. In general, the end-to-end latency is in between the two cases, as shown in the average case as shown in Figure 6(c). It is also affected by the partition size. However, when the partition size ≥ 10, the latency is more likely to be affected by the cached location of the requested objects. On average, users can get the requested object in less than 0.75 s.

C. System Throughput

Generally, the CDN nodes are accessed daily by thousands of users. So, its throughput should be as high as possible. In our scheme, after sending the matched object to the user, the matched partition has to be re-encrypted and migrated to $\mathcal{N}_{i+1}$. Although this operation does not affect the end-to-end latency on users, it affects the throughput of the system since $\mathcal{N}_i$ can not process other requests until the matched partition has been migrated. In this test, we first measured the request process time, the time it takes to completely process a request, which also includes the time for re-encrypting and migrating the matched partition.

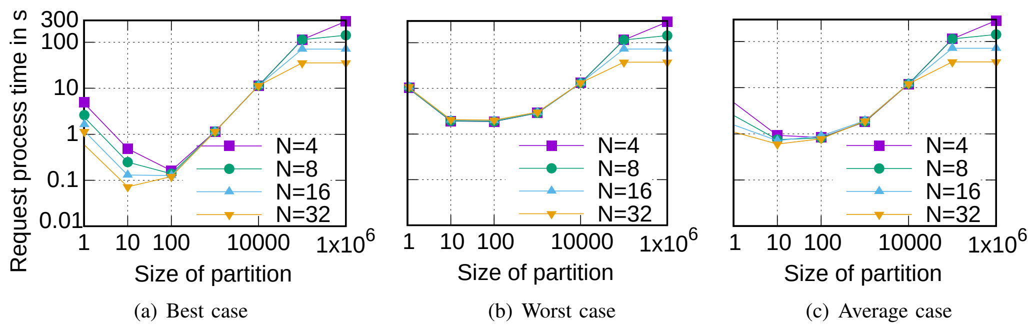
Figure 7 shows the effect on request process time when we vary the size of the partition and the number of nodes in the CDN cluster. From this graph, we can notice that and the system shows the best performance when the partition size is in between 10 and 100. However, when the partition size $>$ 100, the request processing time almost rises linearly with the increase of the partition size. That is because the partition size is proportional to the time spent on *Nonce-blind* and transmission, but inversely proportional to the time spent on object re-encryption and migration. In other words, the bigger the partition size is, the fewer nonces should be blinded and sent to $\mathcal{N}_{i+1}$, but the more objects should be re-encrypted and migrated to $\mathcal{N}_{i+1}$ after the search operation.

Using the request processing time, we computed the throughput of the system. To do so, we measured the time spent on executing 3200 requests and then calculated how many requests it can process per second. Note that the nodes in the CDN cluster are separate machines and could process requests in parallel. Even when $\mathcal{N}_{i+1}$ is encrypting the nonces stored in its NS for $\mathcal{N}_i$, it can search over its CS store for another request simultaneously, since both operations are performed over two separate datasets. Therefore, as shown in Figure 8, the throughput increases with an increase in the number of nodes. Furthermore, when the partition size is in between 10 and 100, the throughput reaches the peak in all the cases. When there are 32 nodes in the CDN cluster, it could process around 500, 15, and 90 requests in the best, worst, and average case, respectively. The throughput in the average case will get better by caching more replicas of the objects.

D. Performance Gain using TTL

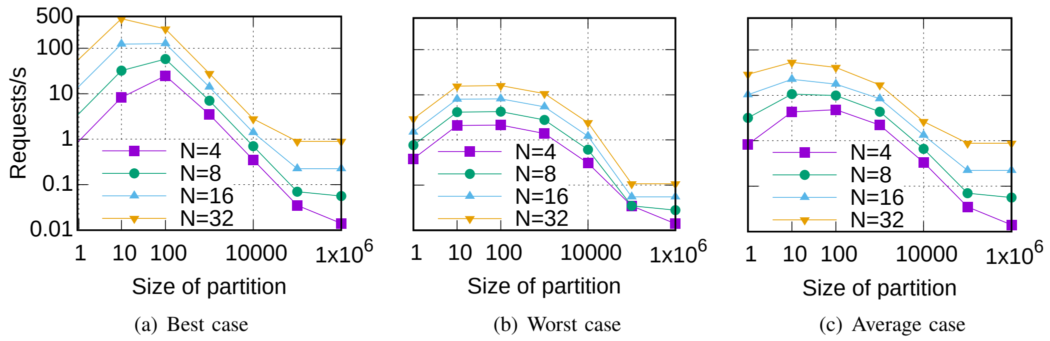
In this part, we show how the performance of the system can be improved by loading more replicas to the CDN cluster for the most popular objects. Moreover, we regard the time of getting the objects in plaintext from the CP directly as a baseline (which includes the round trip latency between the user and the CP, and the search time on the CP), and compare it with our scheme.

As mentioned before, more replicas for the popular objects can be adaptively loaded by setting a request TTL value. In this test, we fixed the number of nodes to 32 and the partition size to 10, changed the TTL value from $N/32$ to $N/2$, and



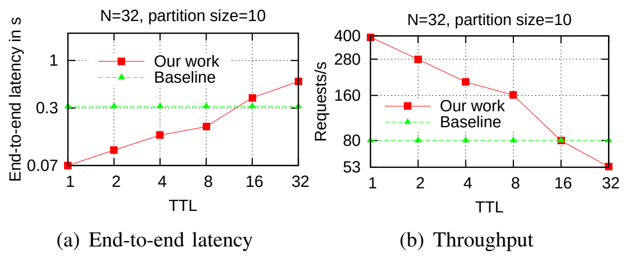
(a) Best case (b) Worst case (c) Average case

Fig. 7: The complete request processing time.



(a) Best case (b) Worst case (c) Average case

Fig. 8: The throughput of a CDN cluster.



(a) End-to-end latency (b) Throughput

Fig. 9: The performance of the system in average case with different TTL values.

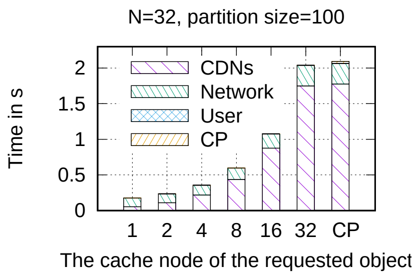


Fig. 10: The breakdown of end-to-end latency.

measured end-to-end latency, the request process time and throughput of the system. Moreover, the requested object was the last one in the TTL-th node. In Figure 9, it is clear that both the end-to-end latency and request process time rise with the increase of the TTL value. Specifically, when $TTL <= 8$, the performance of the system in the average case is better than getting the object from the remote CP directly.

E. Breakdown of the Latency

To better illustrate the overhead on each entity, in Figure 10, we disassembled the processing time into the time spent on the user (encryption time and decryption time), the CDN cluster (nonce blinded time, search time and partition re-encryption time) and the CP (search time), and network latency. In this experiment, we set the number of nodes to 32 and the partition size to 100, and tested the latency when the requested object is the last one stored in the 1^{st}, 2^{nd}, 4^{th}, 8^{th}, 16^{th}, 32^{nd} nodes and the CP.

When the payload size is 1024 bytes, the request encryption and payload decryption only take 0.0027 s in total. Comparing with the latency on other entities, as shown in Figure 10, the latency on the user side is almost negligible. Moreover, the search operation performed on the CP is also significantly more efficient than the CDN cluster. It only takes around 0.025 s when searching over 1,000,001 pre-encrypted objects. The main overhead is outsourced to the CDN cluster, and it increases with the number of searched objects. With a proper partition size, the network latency among nodes can also be controlled at an acceptable level, which is around 0.25 s in the worst case. Note that the network latency can be reduced a lot by sending a much smaller seed, rather than the blinded nonces.

F. The Performances with Different Payload Sizes

In the tests above, the payload size was set to 1024K. We also tested the performance of the system with larger payloads.

TABLE IV: The times of encrypting, re-encrypting, decrypting and delivering an object.

Payload size	Encryption time (ms)	Re-encryption time (ms)	Decryption time (ms)	Transmission time (s)
1K	0.0037	0.0029	2.70	0.0011
128K	0.385	0.338	3.09	0.142
256K	0.813	0.675	3.49	0.285
512K	1.54	1.39	4.29	0.571
1M	3.08	2.73	6.02	1.143

In our system, the payload size affects the performance of encryption on CPs, decryption on users, re-encryption on nodes, and the communication between nodes during the migration. In Table IV, we show the performance of these operations with different payload sizes. When the payload is 1 M, the object can be encrypted in 3 ms, re-encrypted in 2.73 ms, decrypted in 6 ms, and delivered in 1.14 s.

IX. Discussion

To hide the size of object payload, we could also divide the objects into smaller blocks, and upload them to different nodes. Each block should be of the same size. Moreover, the name of each block should be the object name appended with a serial number indicating its order. When the user requests an object, the user client can automatically retrieve serial numbers that could be included in the request. Furthermore, the requests with different serial numbers will be sent to different nodes. Because of the encryption, the node can not learn if the requests are for the blocks belonging to the same object or not. This way, the size of each object is concealed from CDN nodes.

To achieve a high cache hit rate, we could upload more replicas of the most popular and recently requested objects in the CDN cluster. However, to save storage, the replicas of outdated and unpopular objects should be removed periodically. In traditional CDN systems, there is an accounting component to provide the popularity information to the CPs. In our system, the popularity of objects is hidden from the CDN providers. We also give one possible method for the CP to manage outsourced replicas. Roughly, the CP could add a blinded counter, *i.e.,* a random integer, to each object and let the node increase it by one once it matches a request. Because of the migration and re-encryption, the original value of the counter is unknown to CDN nodes. However, it is recorded by the CP. The CP could also encrypt the counter with a fully homomorphic encryption primitive. So, after a period of time, the CP could send a request to the CDN and ask it to return counters of all the replicas of the requested object. Ultimately, the CP could learn how many times an object has been requested during a certain time period. Based on this information, the CP could decide which of them should be removed.

In our system, as shown in Section VIII, with proper TTL value and partition size on the CDNs, retrieving the objects from the CDNs is faster then getting it from the remote CP directly. In addition, the latency can be reduced further by combining with sub-linear index structures, such as B-tree, which will be investigated as our future work. The performance of our system is worse than that of traditional CDNs because it is performing more operations for getting the requested object, and this has been evidenced by similar proposals, such as [3]–[5], [30], [39]. However, our system ensures the confidentiality of objects and preserves user privacy, whereas traditional CDNs do not do so. There is always a tradeoff between performance and security. Moreover, we would like to stress that there are many other benefits of using CDN in addition to low latency, such as reducing the load on the CP and resistant DoS and DDoS attacks. Although our system increases the latency on the user, it maintains all the other benefits of CDN services.

X. Related work

In the literature, many approaches have been proposed to address different security issues in CDNs.

In [3], Edmundson *et al.* aim at solving the same security issues as we discuss in this work. They design and implement a system, called *OCDN*, that allows clients to retrieve web objects from one or more CDNs, while preventing the CDNs from learning the content of the cached objects and user requests. OCDN also breaks the link between users and the requested objects by forwarding the request through a set of proxies (user peers) such that which user makes the original request is unknown to the CDN. To hide the popularity of objects, they propose to use multiple CDNs to distribute the same content ensuring that no single CDN has access to the relative popularity distribution of all objects. Unfortunately, they fail to hide the request pattern from the CDNs, since the user request and object identifier are obfuscated using the deterministic Hash-based Message Authentication Code (HMAC). Moreover, each CDN could also learn which objects are the most popular ones among the accepted requests.

In [4], Leguay *et al.* also propose a security protocol that enables caching of encrypted content on untrusted CDN nodes. In their work, the CP encrypts objects and associates them with reusable pseudo-identifiers in order to enable the CDN node to return requested objects to the users. Unfortunately, the request pattern and the popularity distribution of objects are still revealed to CDN nodes.

In [39], Yao *et al.* consider the scenario where CDN nodes are authorised to perform modification on cached objects, *e.g.,*, transform a video to reduce its resolution. They propose a role-based model where CDN nodes can be authorised to transform objects while maintaining the integrity and confidentiality of objects in CDNs. That is, CDN nodes could securely adapt or transcode encrypted objects without requiring decryption, and users could check if they perform the operation properly. Their approach enhances the flexibility and scalability of data processing on CDNs.

In [5], Xiong *et al.* design a scheme that ensures end-to-end confidentiality of the object by encrypting the object using symmetric encryption. It also provides flexible access control policies for multi-user by combining proxy-based re-encryption with secret sharing and broadcast revocation mechanisms. As a result, the users can be revoked without generating a new key and re-encrypting the objects cached in CDN nodes.

In [30], Levy *et al.* introduce an architecture, called Stickler, that guarantees end-to-end integrity of the object in the face of malicious CDNs. In Stickler, the CP signs the object with its private key before uploading to CDNs, and provides a JavaScript-based boot loader for users. With the boot loader, users can download the requested objects from CDN and check the integrity of them without any modification in the browser.

As done in the CDN, the content-oriented networks, such as CCN [10] and NDN [11], also cache objects in routers or edge nodes and allow the users to access the objects from the nearest cache nodes. In the literature, several works have investigated the approaches to ensure the confidentiality of objects and preserve the privacy of users in content-oriented networks. Zhang *et al.* [16] propose an identity-based signature and encryption mechanism for integrity and trust verification and confidentiality protection of data in the content-oriented network. More recently, Hamdane *et al.* [14] propose a hierarchical identity-based cryptographic naming scheme to control the access to the objects in NDN.

Many other works focus on protecting the user privacy from malicious users. For instance, Baugher *et al.* [12] and Wong *et al.* [13] suggest to protect the requests and object identifiers with cryptographic hash functions. Acs *et al.* [40] investigate the timing and cache probing attacks in CCD/NDN, where attackers can learn if the requested object has been requested by other users by detecting the return time. Furthermore, if the attacker targets a specific victim, it can identify the content of the victim's requests [41]. To resist these attacks, Acs *et al.* propose to set a popularity threshold t and ask the routers to maintain the times being requested for each sensitive object, and the router replies the first t requests with random delay. In [42]–[44], researchers propose similar approaches. Unfortunately, these methods leak users preferences the popularity of objects to the untrusted nodes.

XI. Conclusions and Future Directions

In this work, we identified the limitations of traditional CDN systems and presented a multi-CDN system for protecting outsourced objects and users privacy. Our scheme not only protects the content of the objects and requests from CDN providers, but also protects user preferences and the popularity of objects. Meanwhile, our scheme offers a flexible key management method where revoking users does not require regeneration of keys and re-encryption of the objects. Moreover, when the requested object is not cached in CDNs, the CP can efficiently search over its local storage as without decrypting the request or storing encrypted objects. We also give the solution to improve the cache hit rate without revealing sensitive information to CDN providers.

The main idea in this work is combining SE with the multi-CDN system. Basically, the content of objects and requests are protected by SE, and the object popularity and user preference are hidden by re-randomising and migrating objects between CDN nodes after each request. Due to the usage of nonces, re-randomising objects can be performed efficiently. Moreover, both forward and backward privacy are achieved. We have implemented a prototype of the system and show its practical efficiency.

As for future work, we will design a sub-linear structure to further improve the search efficiency. We also plan to optimise the communication overhead between CDN nodes without sacrificing much security guarantees. Moreover, we will investigate more sophisticated methods to solve the collusion attacks in our system.

Acknowledgements

We would like to thank anonymous reviewers for providing useful comments that helped us in improving the quality of our work. This research is supported by STRATUS (Security Technologies Returning Accountability, Trust and User-Centric Services in the Cloud), a project funded by the Ministry of Business, Innovation and Employment (MBIE), New Zealand.

References

[1] S. Cui, M. R. Asghar, and G. Russello, "Privacy-preserving content delivery networks," in *The 42nd IEEE Conference on Local Computer Networks (LCN)*, 2017.

[2] "SEA attack2015." https://duo.com/blog/malicious-hackers-take-over-media-sites-via-content-delivery-providers. Last accessed: July 24, 2017.

[3] A. Edmundson, P. Schmitt, N. Feamster, and J. Rexford, "OCDN: oblivious content distribution networks," *CoRR*, vol. abs/1711.01478, 2017.

[4] J. Leguay, G. S. Paschos, E. A. Quaglia, and B. Smyth, "CryptoCache: Network caching with confidentiality," in *ICC 2017*, pp. 1–6, IEEE, 2017.

[5] H. Xiong, X. Zhang, D. Yao, X. Wu, and Y. Wen, "Towards end-to-end secure content storage and delivery with public cloud," in *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pp. 257–266, ACM, 2012.

[6] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *SIGSAC 2015*, pp. 668–679, ACM, 2015.

[7] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *SIGSAC 2015*, pp. 644–655, ACM, 2015.

[8] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: website fingerprinting attacks and defenses," in *CCS 2012*, pp. 605–616, ACM, 2012.

[9] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale," in *NDSS 2016*, The Internet Society, 2016.

[10] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. Braynard, "Networking named content," in *CoNEXT 2009*, pp. 1–12, ACM, 2009.

[11] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[12] M. Baugher, B. Davie, A. Narayanan, and D. Oran, "Self-verifying names for read-only named data," in *INFOCOM 2012*, pp. 274–279, IEEE, 2012.

[13] W. Wong and P. Nikander, "Secure naming in information-centric networks," in *Proceedings of the Re-Architecting the Internet Workshop*, p. 12, ACM, 2010.

[14] B. Hamdane, R. Boussada, M. E. Elhdhili, and S. G. E. Fatmi, "Hierarchical identity based cryptography for security and trust in named data networking," in *WETICE 2017*, pp. 226–231, IEEE Computer Society, 2017.

[15] B. Hamdane, R. Boussada, M. E. Elhdhili, and S. G. E. Fatmi, "Towards a secure access to content in named data networking," in *WETICE 2017*, pp. 250–255, IEEE Computer Society, 2017.

[16] X. Zhang, K. Chang, H. Xiong, Y. Wen, G. Shi, and G. Wang, "Towards name-based trust and security for content-centric network," in *ICNP 2011*, pp. 1–6, IEEE Computer Society, 2011.

[17] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *NDSS 2012*, The Internet Society, 2012.

[18] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *USENIX Security 2016*, pp. 707–720, USENIX Association, 2016.

[19] A.-M. K. Pathan and R. Buyya, "A taxonomy and survey of content delivery networks," *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, p. 4, 2007.

[20] "Multi-CDN strategy." https://www.bizety.com/2014/05/09/multi-cdn-strategy/. Last accessed: January 19, 2017.

[21] "Cedexis." https://www.cedexis.com/. Last accessed: January 19, 2017.

[22] "Amazon web services." https://aws.amazon.com/route53/. Last accessed: January 19, 2017.

[23] "Multi-CDN architecture." https://www.cedexis.com/solutions/multi-cdn/. Last accessed: August 21, 2017.

[24] "The Tealium iQ Multi-CDN Network." https://tealium.com/resources/multi-cdn/. Last accessed: January 20, 2018.

[25] "Multi-CDN content delivery." http://www.turbobytes.com/. Last accessed: January 20, 2018.

[26] R. Buyya, A.-M. K. Pathan, J. Broberg, and Z. Tari, "A case for peering of content delivery networks," *IEEE Distributed Systems Online*, vol. 7, no. 10, 2006.

[27] A. K. Pathan, J. Broberg, K. Bubendorfer, K. H. Kim, and R. Buyya, "An architecture for virtual organization (vo)-based effective peering of content delivery networks," in *Proceedings of the 2nd Workshop on the Use of P2P, GRID and Agents for the Development of Content Networks, UPGRADE-CN'07, jointly held with the 16th International Symposium on High-Performance Distributed Computing (HPDC-16 2007), 26 June 2007, Monterey, California, USA*, pp. 29–38, ACM, 2007.

[28] A. K. Pathan and R. Buyya, "Economy-based content replication for peering content delivery networks," in *CCGrid 2007*, pp. 887–892, IEEE Computer Society, 2007.

[29] A. K. Pathan and R. Buyya, "Performance models for peering content delivery networks," in *ICON 2008*, pp. 1–7, IEEE, 2008.

[30] A. Levy, H. Corrigan-Gibbs, and D. Boneh, "Stickler: Defending against malicious CDNs in an unmodified browser," *arXiv preprint arXiv:1506.04110*, 2015.

[31] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *S&P 2000*, pp. 44–55, IEEE Computer Society, 2000.

[32] S. Cui, M. R. Asghar, S. D. Galbraith, and G. Russello, "P-mcdb: Privacy-preserving search using multi-cloud encrypted databases," in *CLOUD 2017*, pp. 334–341, IEEE Computer Society, 2017.

[33] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *S&P 2007*, pp. 321–334, IEEE Computer Society, 2007.

[34] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *CCS 2006*, pp. 89–98, ACM, 2006.

[35] "Akamai report." https://www.akamai.com/us/en/about/our-thinking/state-of-the-internet-report/. Last accessed: April 20, 2017.

[36] "Verizon." http://www.verizonenterprise.com/about/network/latency/. Last accessed: April 20, 2017.

[37] "Global network latency averages." http://ipnetwork.bgtmo.ip.att.net/pws/global_network_avgs.html. Last accessed: September 19, 2017.

[38] "CDN locator." http://www.cdn-locator.com/pops. Last accessed: April 20, 2017.

[39] D. Yao, Y. Koglin, E. Bertino, and R. Tamassia, "Decentralized authorization and data security in web content delivery," in *SAC 2007*, pp. 1654–1661, ACM, 2007.

[40] G. Ács, M. Conti, P. Gasti, C. Ghali, and G. Tsudik, "Cache privacy in named-data networking," in *ICDCS 2013*, pp. 41–51, IEEE Computer Society, 2013.

[41] T. Lauinger, N. Laoutaris, P. Rodriguez, T. Strufe, E. Biersack, and E. Kirda, "Privacy implications of ubiquitous caching in named data networking architectures," *Technical Report TR-iSecLab-0812-001, ISecLab, Tech. Rep.*, 2012.

[42] A. Mohaisen, X. Zhang, M. Schuchard, H. Xie, and Y. Kim, "Protecting access privacy of cached contents in information centric networks," in *CCS '13*, pp. 173–178, ACM, 2013.

[43] A. Mohaisen, H. Mekky, X. Zhang, H. Xie, and Y. Kim, "Timing attacks on access privacy in information centric networks and countermeasures," *IEEE Trans. Dependable Sec. Comput.*, vol. 12, no. 6, pp. 675–687, 2015.

[44] T. Lauinger, N. Laoutaris, P. Rodriguez, T. Strufe, E. Biersack, and E. Kirda, "Privacy risks in named data networking: what is the cost of performance?," *Computer Communication Review*, vol. 42, no. 5, pp. 54–57, 2012.

VITAE

Shujie Cui is a Ph.D. candidate in Department of Computer Science, University of Auckland. She received her M.Sc. degrees in Computer Science from Shandong University, China and the University of Luxembourg, Luxembourg in 2013, and her B.Sc. degree from Shandong University, China in 2011. After obtaining her M.Sc. degree, she worked in the Department of Computer Science at City University of Hong Kong as a research associator. Her research interests include cloud computing, applied cryptography, security and privacy.

Muhammad Rizwan Asghar is a Senior Lecturer in the Department of Computer Science at The University of Auckland in New Zealand. In 2018, he received the Dean's Award for Teaching Excellence. Previously, he was a Post-Doctoral Researcher at international research institutes including the Center for IT-Security, Privacy, and Accountability (CISPA) at Saarland University in Germany and CREATE-NET in Trento Italy. He received his Ph.D. degree from the University of Trento, Italy in 2013. As part of his Ph.D. programme, he was a Visiting Fellow at the Stanford Research Institute (SRI), California, USA. He obtained his M.Sc. degree in Information Security Technology from the Eindhoven University of Technology (TU/e), The Netherlands in 2009. His research interests include access control, cyber security, privacy, and consent management.

Giovanni Russello is an Associate Professor in the Department of Computer Science at the University of Auckland, New Zealand. He received his M.Sc.(summa cum laude) degree in Computer Science from the University of Catania, Italy in 2000, and his Ph.D. degree from the Eindhoven University of Technology (TU/e) in 2006. After obtaining his Ph.D. degree, he moved to the Policy Group in the Department of Computing at Imperial College London, UK. His research interests include policy based security systems, privacy and confidentiality in cloud computing, smartphone security, and applied cryptography. He has published more than 60 research articles in these research areas and has two granted US Patents in smartphone security. He is an IEEE member.