Privacy-preserving Content Delivery Networks

Shujie Cui
The University of Auckland
Auckland, New Zealand
Email: scui379@aucklanduni.ac.nz

Muhammad Rizwan Asghar
The University of Auckland
Auckland, New Zealand
Email: r.asghar@auckland.ac.nz

Giovanni Russello
The University of Auckland
Auckland, New Zealand
Email: g.russello@auckland.ac.nz

***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 the 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 users' requests, CDN service providers could infer user preferences and the popularity of objects, thus resulting in information leakage. Unfortunately, such information leakage may result in compromising users' privacy and reveal business-specific information to untrusted or potentially malicious CDN providers. State-of-the-art Searchable Encryption (SE) schemes can protect the content of sensitive objects but cannot prevent the CDN providers from inferring users' preferences and the popularity of objects.**

In this work, we present a privacy-preserving encrypted CDN system not only to hide the content of objects and users' requests, but also to protect users' preferences and the popularity of objects from curious CDN providers. We encrypt the objects and user requests in a way that both the CDNs and CPs can perform the search operations without accessing those 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.

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 origin server. In recent years, a large number of Content Providers (CPs), such as Netflix, Youtube, and Facebook, use CDNs to deliver objects that are geographically closer to the users. 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.

Despite these benefits, the use of CDNs also raises confidentiality and privacy issues to CPs and users, respectively. In most cases, CPs may wish for their objects to be available only to a particular set of users. For example, only the paying users could watch pay-per-view movies on Netflix. However, once the object is outsourced to CDN nodes, it will be out of the control of CPs. Users could access unauthorised objects by colluding with malicious CDN service providers. In some commercial sites, user requests are analysed to extract user preferences and push targeted advertising. In e-commerce, the popularity of each product is business-critical information. The CDN service provider could get the object popularity by analysing the request history on each CDN node. It will seriously impact the business strategy of the CPs if a malicious CDN service provider sells this information to their competitors. Finally, users' privacy might also be impacted when contents are delivered through CDNs. Given that CDN providers serve the requests from users they are also able to profile users based on the requested content. Such profiling might put user privacy at risk.

Therefore, when distributing sensitive objects in CDNs, it is crucial to protect (i) the content of objects and requests, (ii) the popularity of objects, and (iii) user preferences from CDN service providers.

Searchable Encryption (SE) [1] supports search operations over encrypted data. In cloud computing, SE is widely used to protect the outsourced data from the cloud service provider. Such schemes allow the cloud service provider to perform encrypted search operations on encrypted data without revealing the data. In CDNs, encrypting the objects and requests with SE schemes could address the above security concerns. However, most of the SE schemes, such as [2]–[4], can not be directly applied to CDNs due to the following issues:

- First, they leak the *search pattern* and *access pattern* [2]. That is, with traditional SE schemes, CDN service providers could learn if any two encrypted requests are the same or not, and which objects match them. As discussed by Cash *et al.* [5], exploiting this information leads to breaking the SE encryption scheme and retrieve the data in cleartext.
- If an SE scheme is used, a scalable key management mechanism would be required. In CDN systems, the object is cached by a number of nodes and can be accessed by a large number of users. Ideally, the user should be able to join in or leave the system without affecting other users. In particular, a user should be revoked without any new key generation and re-encryption of the data. Otherwise, it would be significantly expensive to refresh all the objects cached in each node and distribute the new keys to the rest of users. Unfortunately, most of the SE schemes, such as [2], [4], fail to offer such a scalable key management method. Proxy re-encryption based SE schemes, like [3], [6], could solve this problem. However, they are based on asymmetric encryption and tend to be much slower than traditional symmetric encryption.

- Third, in traditional SE schemes, all the search operations are performed only by the cloud service providers. However, in CDNs, when the requested object is not found in one of the CDN nodes, the request will be forwarded to CPs for another round of searching. Hence, in using a tradition SE scheme, a CP has to store a searchable data structure and perform the encrypted search as the CDN node does. But this would require heavy storage and intensive computation on the CP end.

In this work, we present a privacy-preserving encrypted CDN system to address the aforementioned issues. In our approach, we encrypt the objects and user requests such that both the CDNs and CPs can perform the matching operation efficiently without performing any decryption. By distributing the objects and requests across multiple CDN service providers, user preferences and object popularity are concealed from each CDN service provider as long as they do not collude.

II. Approach Overview

A. Multi-CDN

In this paper, we focus on the CDN systems based on the pull-based mode and DNS-based request routing [7]. The time to process user requests strongly depends on the availability of CDN nodes. Ideally, the more the nodes, the more efficient is to retrieve the object. However, the nodes owned by one CDN service provider are not everywhere, and their performance varies across regions, throughout the day. Thus, the users may not get the best service anywhere, anytime. 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 owned by different CDN service providers into a single network known as *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 [8]. CDN Aggregator (*e.g.,* Cedexis [9]) and Load Balancer (*e.g.,* Amazon Route 53) [10] are two options for implementing a multi-CDN. In this work, we exploit multi-CDN to conceal user preference and object popularity from CDN providers.

B. System Model

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

- **User**: It represents the entity that can request objects.
- **Content Provider (CP)**: 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 offered by different CDN service providers. 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.

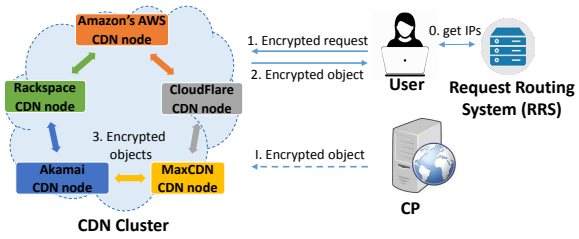

Fig. 1: The proposed architecture: A CDN cluster is composed of a set of CDN nodes owned by different CDN service providers. The CP uploads encrypted objects with the pull-based or push-based mode. With the IPs of the closest nodes, users could issue encrypted request and get the requested object.

C. Threat Model

We assume the CP is trustworthy. It encrypts the outsourced objects and authorises users to get objects from CDN nodes. Users are only supposed to securely keep their secret keys and objects properly. We assume each CDN service provider is honest but it could be victim of an insider or outsider attack. In our threat model, we assume that any two CDN service providers do not collude together. In other words, an attacker is not able to infiltrate the infrastructure of more than one CDN provider. Considering there is no content stored on the RRS, it does not matter if it is trusted or untrusted. If the RRS is provided by the CP, like in URL rewriting routing schemes, it would be trusted. On the contrary, if it is provided by a CDN provider, like in DNS-based routing schemes, it would be untrusted. In this work, we assume the RRS is provided by one of the involved CDN service providers.

III. Solution Details

We consider a CDN cluster with N nodes: $\mathcal{N}_1, \ldots, \mathcal{N}_N$, where $N > 3$ and $\mathcal{N}_i, \mathcal{N}_{i+1}, \mathcal{N}_{i+2}, \mathcal{N}_{i+3}$[1] $(i \geq 1)$ should be provided by different CDN service providers. 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 service providers.

A. Data Representation

The object cached in CDNs 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 $C = \langle name, payload \rangle$. Each object can be identified by its unique $name$, and both $name$ and $payload$ are represented as binary strings.

Before uploading to CDNs, objects 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,

[1] $\mathcal{N}_{i\pm a}$ is short for $\mathcal{N}_{(i\pm a) \bmod N}$, where a is an integer.

such as AES, and n, η are two nonces. Considering both H and f are deterministic, to avoid repetitive computation, the CP pre-computes and locally stores:

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

Once requested by a node $\mathcal{N}_i$, the CP just XORs the encrypted object with nonces $\langle n, \eta \rangle$ and sends it to $\mathcal{N}_i$. A pair of nonces $\langle n, \eta \rangle$ will be sent to $\mathcal{N}_{i+1}$, where $i \in [1, N]$. Therefore, each node $\mathcal{N}_i$ has two separate stores: Content Store (CS) and Nonce Store (NS). The CS caches EO, and the NS caches the pair of nonces $\langle n, \eta \rangle$. However, the nonces stored in $\mathcal{N}_i$'s NS are those included in the EOs cached in $\mathcal{N}_{i-1}$'s CS.

In order to reduce the storage and communication overhead among the CDN cluster (will explain in Section III-B), every W objects cached in a node is encrypted with the same nonce. In concept, the objects cached in each surrogate are divided into partitions $\{P_0, P_1, ...\}$ base on their physical order. 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. Specifically, each partition is identified with an unique identifier pid, and $\langle n_{pid}, \eta_{pid} \rangle$ is the nonce pair used to encrypt the objects in partition P_{pid}.

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

1: $counter \leftarrow 0$
2: $ER \leftarrow H_k(name) \oplus \alpha_0$, α_0[2] $\xleftarrow{\$} \{0,1\}^{2\lambda}$
3: Send the encrypted request ER and $counter$ to $\mathcal{N}_i$, and send α_0 to $\mathcal{N}_{i+1}$

B. Request Process

Here, we explain the request process details. Basically, the user uses the *Encryption* and *Decryption* components to encrypt requests and decrypt returned payloads, respectively. Each node $\mathcal{N}_i$ is equipped with *NonceEnc*, *Search* and *Post-search* to process encrypted requests, where *NonceEnc* 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 deployed with the *Search* components to provide the missed objects for CDN clusters. The implementation detail of each component is given below.

1) User Encryption: The user encryption algorithm is described in Algorithm 1. To avoid an infinite loop, the user initialises a counter to record the times the request has been forwarded. The $name$ of the required object is hashed and XOR-ed with a nonce α_0. The nonce can ensure that the encrypted request ER is semantically secure, such that the CDN service providers can not tell if users are requesting for the same object or not just from ER. Finally, $(ER, counter)$ and the nonce α_0 are sent to $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$, respectively.

[2]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.

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

Algorithm 2 NonceEnc[3](α_{s-2})

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

Algorithm 3 Search($ER, counter$)

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

2) Search Operation on $\mathcal{N}_i$: Considering both the objects and request are encrypted with nonces, $\mathcal{N}_i$ alone would not be able to check if there is a match. It needs $\mathcal{N}_{i+1}$'s assistance. Specifically, after receiving the nonce α_0 from the user, $\mathcal{N}_{i+1}$ executes *NonceEnc* (Algorithm 2) to encrypt 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 encrypted nonce sets ENS is sent to $\mathcal{N}_i$ for search.

If $counter < N$, $\mathcal{N}_i$ 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, $\mathcal{N}_i$ checks (Line 9) if: $ENS[pid] \stackrel{?}{=} h(ER \oplus EN_{id})$. where $pid = \lfloor \frac{id}{W} \rfloor$. That is, $\mathcal{N}_i$

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

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

Algorithm 5 Search-on-CP(ER, α_{i+N})

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

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, $\mathcal{N}_i$ stops the search, returns the matched EP_{id} to the user, and notifies $\mathcal{N}_{i+1}$ to send the η_{pid} to the user by sending the matched pid to it (Line 10). Subsequently, all the records in the matched partition are re-encrypted with a new nonce pair $\langle n', \eta' \rangle$ (Line 13) and sent to $\mathcal{N}_{i+1}$ (Line 14). Meanwhile, the new nonce pair $\langle n', \eta' \rangle$ is sent to $\mathcal{N}_{i+2}$ (Line 15). Afterwards, all the records in the matched partition are removed from $\mathcal{N}_i$ (Line 16). Consequently, whether these objects will match future requests will be unknown to $\mathcal{N}_i$.

If $\mathcal{N}_{i+1}$ receives the migrated partition P_{pid} and its pid from $\mathcal{N}_i$, it carries out *Post-search* (Algorithm 4). Specifically, it first sends η_{pid} to the user (Line 1). Next, $\mathcal{N}_{i+1}$ removes the old nonces $\langle n_{pid}, \eta_{pid} \rangle$, which are stored in its NS, from each object in P_{pid} with XOR operation (Line 3). Consequently, they are only bound to the new nonce pair $\langle n', \eta' \rangle$, which is unknown to $\mathcal{N}_{i+1}$, indicating $\mathcal{N}_{i+1}$ is unable to check if the mitigated objects match previous requests or not without the assistance of $\mathcal{N}_{i+2}$. Finally, $\mathcal{N}_{i+1}$ inserts P_{pid} into its CS and removes $\langle n_{pid}, \eta_{pid} \rangle$ from its NS (Lines 4 and 5).

3) Request Process on $\mathcal{N}_{i+1}$: Instead, if the requested object is not cached on $\mathcal{N}_i$, the request ER will be re-encrypted with a new nonce α_i (Line 18, Algorithm 3) and forwarded to $\mathcal{N}_{i+1}$ for another round of search. Formally, the re-encrypted ER is: $ER = H_k(name) \oplus \alpha_0 \oplus \alpha_i$. Meanwhile, if $counter < N$, α_i is sent to $\mathcal{N}_{i+2}$, with which $\mathcal{N}_{i+2}$ could help $\mathcal{N}_{i+1}$ to perform the second round of search operation by running $NonceEnc$ and generating ENS.

Unlike $\mathcal{N}_i, \mathcal{N}_{i+1}$ first removes α_0 from ER (Line 2). Recall that $\mathcal{N}_{i+1}$ has already get α_0 from the user. Then, with ENS, $\mathcal{N}_{i+1}$ could perform the second round of search over its CS as $\mathcal{N}_i$ did. The same operations will be performed on the rest nodes until the requested object is found or $counter = N$.

4) 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. Formally, the CP receives $ER = H_k(name) \oplus \alpha_{i+N}$ and α_{i+N} from $\mathcal{N}_{i+N+1}$ and $\mathcal{N}_{i+N}$, respectively. By XOR-ing α_{i+N} with ER, 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 $H_k(name) \stackrel{?}{=} H_k(name_{id})$. The matched object is first sent to the user (Line 4, Algorithm 5), and then it will be re-encrypted with nonces and sent to $\mathcal{N}_{i+N}$. 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 is part of an existing partition, the CP needs to get the corresponding nonce pair from $\mathcal{N}_{i+N+1}$ (Line 7). Otherwise, the CP generates a new pair and sends it to $\mathcal{N}_{i+N+1}$ (Line 9).[4] Moreover, the partition containing the matched object should be re-encrypted and migrated to $\mathcal{N}_{i+N+1}$.

5) User 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. It can finally decrypt the matched payload by executing f_k^{-1} with the secret key k.

[4]The matched object and nonce can also be sent to the user by $\mathcal{N}_{i+N}$ and $\mathcal{N}_{i+N+1}$.

C. User Revocation

Since the nonce is bound to an EO, without the assistance of the CDN service 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 CDN nodes to add this user to their revoked user list. When receiving a request, the node will first check if the user has been revoked. If yes, they will reject the request. In case a revoked user colludes with one of the CDN service providers, she cannot get the search results, since such operation requires the cooperation of at least two of them.

IV. Conclusion

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, but also conceals user preferences and the popularity of objects from CDN providers. It offers a flexible key management method for multi-user access. 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. As for future work, we plan to implement a prototype of the system and show its practical efficiency and give a comprehensive security analysis of the system.

References

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

[2] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *CCS 2006*, A. Juels, R. N. Wright, and S. D. C. di Vimercati, Eds. ACM, 2006, pp. 79–88.

[3] M. R. Asghar, G. Russello, B. Crispo, and M. Ion, "Supporting complex queries and access policies for multi-user encrypted databases," in *CCSW 2013*, A. Juels and B. Parno, Eds. ACM, 2013, pp. 77–88.

[4] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage." in *NDSS 2013*, vol. 71, 2014, pp. 72–75.

[5] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *SIGSAC 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 668–679.

[6] R. A. Popa and N. Zeldovich, "Multi-key searchable encryption," *IACR Cryptology ePrint Archive*, vol. 2013, p. 508, 2013.

[7] R. Buyya, M. Pathan, and A. Vakali, *Content delivery networks*. Springer Science & Business Media, 2008, vol. 9.

[8] "Multi-cdn strategy," last accessed: January 19, 2017. [Online]. Available: https://www.bizety.com/2014/05/09/multi-cdn-strategy/

[9] "Cedexis," last accessed: January 19, 2017. [Online]. Available: https://www.cedexis.com/

[10] "Amazon web services," last accessed: January 19, 2017. [Online]. Available: https://aws.amazon.com/route53/