OCD: Oblivious Content Distribution

Paper #XX – 9 Pages + References

Abstract

1 Introduction

Governments are increasingly using their authority to access data from their citizens and foreigners, even when this data may be stored overseas. For example, in a recent case, the United States government tried to compel Microsoft to surrender data about U.S. citizens, even when the data itself was stored abroad. Users may also face the converse problem, where access to their data may depend on the laws of the country where their data is stored. Recent work, for example, highlights the possibility that governments may move data across borders to facilitate surveillance.

The rise of content distribution networks (CDNs) makes the threats to citizen privacy more widespread, because data may be replicated in geographically diverse regions, and users may not have purview over where their data is stored, or where their traffic goes. One way to address this issue is to give users better control over the routes that either traffic takes between the client and a content provider; previous work has developed defenses that can better help users control such routing of traffic. Yet, control over routing is only part of the story: if content is *hosted* in a particular country, then user traffic might traverse that country simply to retrieve the content. Thus, protecting user privacy requires protecting not only the routes that traffic traverses, but also both the contents and the access patterns of data stored on CDN cache nodes.

In this paper, we design and implement a content distribution network that allows clients to retrieve web objects, while preventing the CDN cache nodes from learning either the content that is stored on the cache nodes or the content that clients are requesting. We call this system an *oblivious CDN* (OCDN), because the CDN itself is oblivious to both the content it is storing and the content that clients are requesting. OCDN allows clients to request individual objects with identifiers that are encrypted with a key that is shared by a client proxy and the origin server that is pushing content to cache nodes, but is not known to any of the CDN cache nodes. To do so, the origin server publishes multiple replicas of each object, each encrypted under a different shared key that is subsequently shared with a corresponding client proxy. To retrieve content, a client proxy transforms the URL that it receives from a client to an obfuscated identifier using the key shared with the origin server. The CDN cache node then returns the object corresponding to the object identifier; that object is also encrypted with a key that is shared between the origin and the proxy. This approach allows a user to retrieve content from a CDN without the cache node ever seeing the URL or the corresponding content. Users can use OCDN with minimal modification to their existing configuration: merely directing a web browser to an OCDN proxy allows a client to use the system.

Although the basic mechanisms for obfuscating the URL and the corresponding content are relatively straightforward, ensuring that the CDN operator never learns information about either (1) what content is being stored on its cache nodes or (2) which objects individual clients are requesting is more challenging, due to the many possible inference attacks that a CDN might be able to run. For example, previous work has shown that even when web content is encrypted, the retrieval of a collection of objects of various sizes can yield information about the web page that was being fetched. Similarly, URLs can often be inferred from relative popularity in a distribution of web requests, even when the requests themselves are encrypted. OCDN addresses these challenges by creating copies of the same object that are encrypted under different keys, and deploying multiple versions of the same object to the same CDN cache node, with each copy being encrypted with a different key. While this approach reduces the threat of various types of inference and also limits the number of OCDN proxies that share a key, each CDN cache node must store multiple copies of the same object, reducing both storage efficiency and cache hit rates. Our evaluation explores the implications of this tradeoff, as well as how OCDN performs relative to a conventional CDN.

We make the following contributions. First, we explore the problems that arise when data is stored across jurisdictional boundaries and highlight the need for a CDN that is oblivious to the content it is hosting and serving. Second, we design and implement OCDN, a CDN that can be oblivious to the content that it is hosting, as well as the content that clients are retrieving. Finally, we evaluate the performance of OCDN for both individual object retrieval and complete web page loads and show that OCDN incurs a performance overhead of only XX% relative to conventional CDNs.

The rest of the paper is organized as follows. Section 2 describes the typical operation of a CDN, the types of information that CDN operators typically know today, and the types of information we aim to obfuscate. Section 3 describes the threat model and security objectives for OCDN; based on these threats, Section 4 outlines various design decisions. Section 5 describes the protocol for both publishing and retrieving content, and Section 6 evaluates the performance overhead of OCDN. Section 7 describes various limitations and possible avenues for future work, Section 8 discusses related work, and Section 9 concludes.

2 Background

Before describing how Oblivious CDN works, we outline how a Content Delivery Network (CDN) typically operates, and what information it naturally has access to. CDNs provide content caching as a service to content publishers. A content publisher may wish to use a CDN provider for a number of reasons:

- CDNs cache content in geographically distributed locations, which allows for localized data centers, faster download speeds, and reduces the load on the content publisher's server.
- CDNs typically provide usage analytics, which can help a content publisher get a better understanding of usage.
- CDNs provide a high capacity infrastructure, and therefore provide higher availability, lower network latency, and lower packet loss.

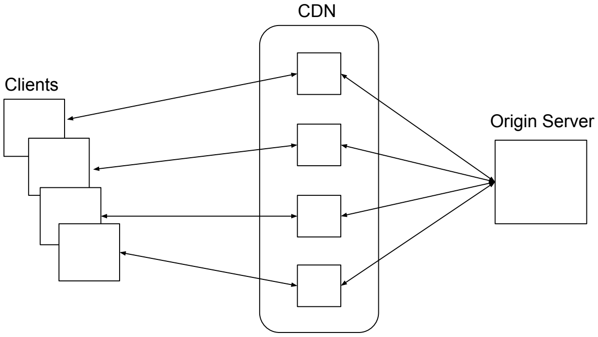


Figure 1: *The relationships between clients, the CDN, and content publishers in CDNs today.*

- CDNs' data centers have high bandwidth, which allows them to handle and mitigate DDoS attacks better than the content publisher's server.

CDN providers usually have a large number of edge servers on which content is cached; for example, Akamai has more than 216,000 servers in over 120 countries around the world [2]. Having more edge servers in more locations increases the probability that a cache is geographically close to a client, and could reduce the end-to-end latency, as well as the likelihood of some kinds of attacks, such as BGP (Border Gateway Protocol) hijacking. When a client requests a web page, the CDN serves it from its cache, one of its edge servers. If the requested page's content is not in its cache, then the request is forwarded to the content publisher's server, the CDN caches the response, and returns the content to the client.

Because the CDN interacts with both content publishers and clients, as shown in Figure 1, it is in a unique position to learn an enormous amount of information. CDN providers know information about all clients who access data stored at the CDN, information about all content publishers that cache content at CDN edge servers, and information about the content itself.

Knowing the content. CDNs, by nature, have access to all content that they distribute, as well as the content identifier, the URL. First, the CDN must use the URL, which is not encrypted or hidden, to locate the content. Therefore, it is evident that the CDN already knows what content is stored in its caches. And because CDNs provide analytics to content publishers, they keep track of cache hit rates, and how often content is accessed. But the CDN does not just know about the content identifier, it also has access to the plaintext content. The CDN performs optimizations on the content to increase performance; for example, CDNs minimize CSS, HTML, and JavaScript files, which reduces file sizes by about 20%. They can also inspect content to conduct HTTPS re-writes. In addition, requesting content via HTTPS does not hide any information from the CDN; if a client requests a web page over HTTPS, the CDN terminates the TLS connection on behalf of the content publisher. This means that not only does the CDN know the content, the content identifier, but also knows public and private keys, as well as certificates associated with the content it caches. Recently, the fact that CDNs know the content they are distributing has made its way into the legal system. A court order was given to Cloudflare that required the CDN to search out and block publishers who use a variation of a trademark held by a group of music labels [23]. Originally, the music labels went after the trademark infringing website, but later the order was extended to Cloudflare; the order "required CloudFlare to block all of its customers from using domain names that contained 'grooveshark,' regardless of whether those domains contained First Amendment-protected speech, or had any connection with the 'New Grooveshark' defendants who were the targets of the actual lawsuit." This case highlights the problem that CDNs face by knowing all the content that they distribute: it may burden them with the legal responsibility for the actions of their customers and clients.

Knowing client information. Clients fetch content directly from the CDN's edge servers, which reveals information about the client's location and what the client is accessing. Unique to CDNs is the fact that they can see each client's cross site browsing patterns. CDNs host content for many different publishers, which allows them to see content requests for content published by different publishers. This gives an enormous amount of knowledge to CDNs; for example, Akamai caches enough content around the world to see up to 30% of global Internet traffic [1]. And we have seen the implications of a CDN knowing this much information when Cloudflare went public with the years-long National Security Letters they had received [8]. These National Security Letters demand information collected by the CDN and also include a gag order, which prohibits the CDN from publicly announcing the information request.

Knowing content publisher information. A CDN must know information about their customers, the content publishers. Publishers must be billed, which causes the CDN to keep track of who the content publisher is, and what the publisher's content is. And the combination of the CDN seeing all content in plaintext and the content's linkability with the publisher, gives the CDN even more power. Additionally, as mentioned previously, the CDN often holds the publisher's keys (including the private key!), and the publisher's certificates. This has led to doubts about the integrity of content because a CDN can impersonate the publisher from the client's point of view [14].

3 Threat Model and Security Goals

In this section, we describe our threat model, outline the capabilities of the attacker, and introduce the design goals and protections provided by Oblivious CDN.

3.1 Threat Model

Our threat model involves two different passive, but powerful adversaries. Both adversaries wish to take advantage of the knowledge that a CDN has; the two adversaries are: 1) the CDN operator itself and 2) a government (or similar).

We address an attacker who wants to learn what content each client is accessing; this could mean learning either the identifier of the content, such as a URL, or the actual content of the web page. Additionally, we are concerned with a passive attacker who wishes to learn information that compromises the privacy of content publishers and/or Internet users. An active attacker that attempts to modify and/or delete data is out of the scope of this work.

In the case of the CDN operator being the adversary, he might try to make inferences. He can infer the popularity of content based on the number of accesses and infer the web page from the popularity. Addiitonally, this attacker may be able to infer a web page based on the length of the content. This adversary could be an inside attacker or an insider who is compelled to provide data.

In the case of an adversarial government or nation-state, the attacker could compel the CDN to divulge information, such as access logs or content. This adversary can serve an overreaching subpoena or National Security Letter. This is a realized attacker, as we know that this has actually already occured, and which was discussed in Section 2 [8].

3.2 Security and Privacy Goals for Oblivious CDN

To protect against the attackers described in Section 3.1, we highlight the design goals for Oblivious CDN. Each stakeholder, in this case the content publisher, the CDN, and the client, each have different risks, and therefore should have different protections. All three stakeholders can be protected by preventing CDNs from learning information, changing jurisdictions, and reducing the probability of attacks. Our design goals are listed in Table 1, and we further discuss our design decisions in Section 4.

Prevent the CDN from knowing information. First, and foremost, the CDN should not have access to all the information that it currently has access to (as described in Section 2). By limiting the information that the CDN knows, it limits the amount of information that an adversary can learn or request. Oblivious CDN should hide content, content identifiers, and remove links between clients and their content requests, as well as remove links between content (and content identifiers) and the content publisher. If the CDN does not know what content it is caching, who is requesting it, or who has created it, then an inside attacker will not be able to learn valuable information, and the CDN will not be able to supply a government adversary with the requested data.

Move jurisdictions. There have been many legal battles over which government is allowed access to which data; for example, data can be stored in Country X, but belonging to an organization in Country Y, and the data is about a person in Country Z. It is unclear which of these countries can legally demand the data with a subpoena or warrant. The issue becomes much more complex when the specific laws and policies of the different countries are conflicting. Perhaps Country X has much stronger data privacy guarantees and enforcement than Country Y or Z. A recent approach taken by Microsoft was to establish a datacenter in Germany, which is technically under the control of the Deutsche Telekom subsidiary T-Systems [17]. This prevents the United States government from serving Microsoft with a subpoena for data stored in Germany, where German citizens (or others) can request to have their data stored. To complement these legal battles, Oblivious CDN should take these conflicting jurisdictional issues into account. Additionally, the system should be able to protect the privacy of clients' locations; while addressing jurisdictional data privacy concerns, Oblivious CDN should not reveal more information about clients. It also provides the ability for clients to hide their fine-grained location, while still following the policies of the jurisdictions in which they reside.

Reduce the attack surface. Oblivious CDN should be able to achieve the previously mentioned security and privacy goals while not introducing new attacks. More specifically, it should aim to reduce the probability of other attacks occuring, and reduce the probability of any information leakage in the case of an attack.

A strength of Oblivious CDN is that it protects the origin server, the CDN itself, and the client, whereas existing systems, such as Tor, only protect the client.

4 Design

We describe the evolution of the design of Oblivious CDN starting with an initial strawman approach. We then alter the design as we address each of the design goals discussed in the previous section (and in Table 1), which brings us to a complete design.

4.1 Strawman Approach: Hiding Information From CDN

To prevent an inside attacker or government demanding data from learning information, the CDN must not have the knowledge of what content it is caching. Therefore, the content *and* the associated URL must be obfuscated before it enters the CDN.

Encrypt Content. The content can be obfuscated by encrypting it with a key that is not known to the CDN. Because this must be done prior to any caching, the content publisher has to generate some key k to encrypt the content with. Then this encrypted (and subsequently obfuscated) content is then sent to the CDN[1] and stored in its caches. Additionally, if the domain supports HTTPS requests, then the content publisher must also encrypt the associated certificate with the same key k. This content and certificate must be decrypted after it leaves the CDN; the client could decrypt the certificate and check its validity. If valid, then the client uses k to decrypt the content.

Obfuscate URL. Encrypting the content alone does not hide much from the CDN; the content identifier, or URL, must also be obfuscated, otherwise the CDN can still reveal information about which clients accessed which URLs (which is indicative of the content). In obfuscating the URL, the result should be a fixed, and relatively small size. Hashing the URL provides this property, and hides the actual URL from the CDN. With obfuscated content and URLs, the CDN does not know what content a given client is accessing.

Client Anonymization. The CDN may not know the content that a client is accessing, but it still knows information about the clients that are accessing any content at the CDN. It knows where the clients are located and how many times they are accessing any content. To address this, a client can simply use an anonymizing proxy or VPN when accessing content. This hides a certain amount of information about the client, including the client's location and a direct link of client to content request.

This strawman approach obfuscates content from the viewpoint of the CDN, which also allows the CDN to claim plausible deniability when served with a subpoena. Unfortunately, this approach raises many questions: are anonymizing proxies/VPNs trustworthy?, should clients be trusted to know k?, can any jurisdiction still subpoena the CDN for information (presumably the CDN has locations in many countries and clients in many countries)?, do malicious clients have more power as the CDN's attack detection and prevention capabilities are severely limited?

4.2 Moving Jurisdictions

As it is still unclear which jurisdiction is legally allowed to subpoena information from the CDN, this system should be a complement to the legal framework; it should make clear which jurisdictions are allowed which information, and also prevent an overreaching government from demanding data it should not have. This can be addressed by introducing a closed system of proxies, where a client uses a single proxy when requesting content — not only does this proxy provide a level of client anonymity, but also decides which jurisdiction the data falls under.

Proxy in Client's Jurisdiction. By replacing the use of an anonymizing proxy or VPN with the use of a specified proxy, the system can specify which juridiction each client falls under. For example, each country could have a set of proxies, where all clients in each country use one of the proxies that is located in their country.

[1] Most CDNs allow the publisher to decide on a push or pull model, but this makes no difference in our system design.

Design Goal	Design Decision
Prevent CDN from knowing information	(1) encrypt content, (2) obfuscate URL
Move Jurisdictions	decouple content distribution from decision of trust via proxies
Reduce Attack Surface	(1) n shared keys, for $1 < n < \lvert proxies \rvert$, (2) secrecy in URL obfuscation

Table 1: *Design goals and the corresponding design choices made in Oblivious CDN.*

This should allow the client's jurisdiction to serve a subpoena for information passing through the specific proxy that a client uses in the client's jurisdiction in necessary criminal investigations, but prevents any other jurisdiction from demanding data on the same client because the subpoena must be served for a specific proxy. While it appears that the source of trust is simply shifted from the CDN to the proxy, the use of this proxy is essential in the system design for a number of reasons. First, the use of proxies distributes trust; instead of a central entity, the CDN, having access to all information, it is now a set of proxies that each have access to some subset of information. Each proxy only knows information about a much smaller set of clients. Additionally, the proxy does not cache content or keep some state, whereas the CDN does both of these. Apart from addressing jurisdictional issues, proxies also improve usability; instead of each client having knowledge of each k for each domain, each proxy could have knowledge of each k. This means that clients would not have to keep track of secret knowledge, and overall, the number of entities that know this secret key is much smaller (as the number of proxies is significantly smaller than the number of clients). Lastly, the proxy can perform attack detection and prevention before the attack even reaches the CDN; more optimizations made possible by the use of proxies are discussed in Section 5.4.

Decouple Content Distribution from Decision of Trust. Using proxies also separates the issues of trust and content distribution. A client no longer needs to trust the CDN, which all other clients would need to trust as well. Now the client can simply trust the proxy, which interacts with the CDN; this proxy is a completely separate entity from the CDN. Therefore this design allows a client to decouple these issues, where she can trust only a proxy, not the CDN, but still access cached content at the CDN.

4.3 Reducing Attack Surface

The design has evolved into a system where: 1) the CDN does not know the content, the content identifier, or which clients are accessing any content, 2) the client's data is only subject to subpoenas in her jurisdiction (or the jurisdiction of the proxy), 3) trust has been distributed, as opposed to centralized. There are still some remaining security vulnerabilities that must be addressed. First, the shared key k for each content publisher is shared among all the proxies; therefore, if one proxy is compromised, then the key must be replaced on all proxies. Second, simply hashing the URL for obfuscation purposes leaves room for an attacker who simply wishes to guess whether the obfuscated URL is a specified plaintext URL or not. By increasing the number of shared keys k and introducing shared secrets to URL obfuscation, the design reduces the probability of attacks.

Increasing the Number of Keys. If a single shared key k associated with a single content publisher gets compromised, then all proxies have a compromised k for that content publisher. By increasing the number of shared keys that a content publisher uses to encrypt her content, this decreases the fraction of proxies that use a compromised key given that one of the keys is compromised. Instead of a content publisher encrypting content with a single key, she will encrypt a copy of the content with different keys. If there are n shared keys, then there will be n copies of the content, where each copy is encrypted with a different shared key. To maximize security, each proxy should share a different k with the content publisher; unfortunately, this defeats the purpose of CDNs caching content. If each proxy is requesting a unique copy of the content, then cache hit rates decrease and performance becomes worse. To make a tradeoff between security and performance, there should be n keys where $1 < n < \lvert proxies \rvert$, such that subsets of proxies share the same shared key k with the content publisher. If one of these shared keys gets compromised, only that subset of proxies that use that shared key must get a new key.

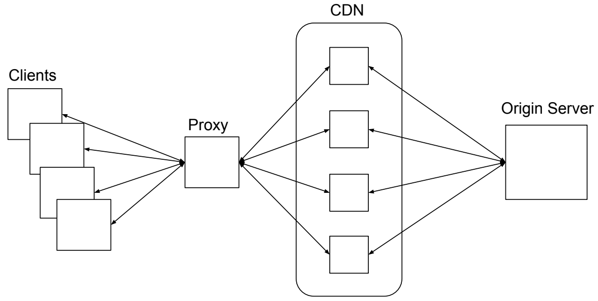


Figure 2: *The relationships between clients, the CDN, proxies, and content publishers in Oblivious CDN.*

Adding Secrecy to URL Obfuscation. According to the strawman approach, the URL is obfuscated via hashing. Unfortunately, an attacker could guess what the content identifier is by hashing his guesses and comparing with the hashes stored in the CDNs caches. To mitigate this vulnerability, the content publisher should incorporate the use of the shared key k in the hash of the URL by using an HMAC. This provides the property of fixed length identifiers, while obfuscating and preventing against guessing attacks.

The resulting system is the basis of Oblivious CDN, and the high-level view of the system is shown in Figure 2, which can be compared to what CDNs look like today (Figure 1). The next section describes Oblivious CDN in more detail and outlines the specific steps for publishing and retrieving content.

5 Oblivious CDN Protocol

Based on the design decisions discussed in the previous section, we specify the steps taken to publish and retrieve content in Oblivious CDN.

5.1 Publishing Content

In order to publish content such that the CDN never sees the content, the publisher must first obfuscate her content, as described in Section 4.1. Figure 3 shows the steps taken when publishing content.

The most important step in content publishing is obfuscating the data. We assume that the origin server already has a public and private key pair, as well as a certificate. To obfuscate the data the origin server will need to generate n shared keys, where n should be

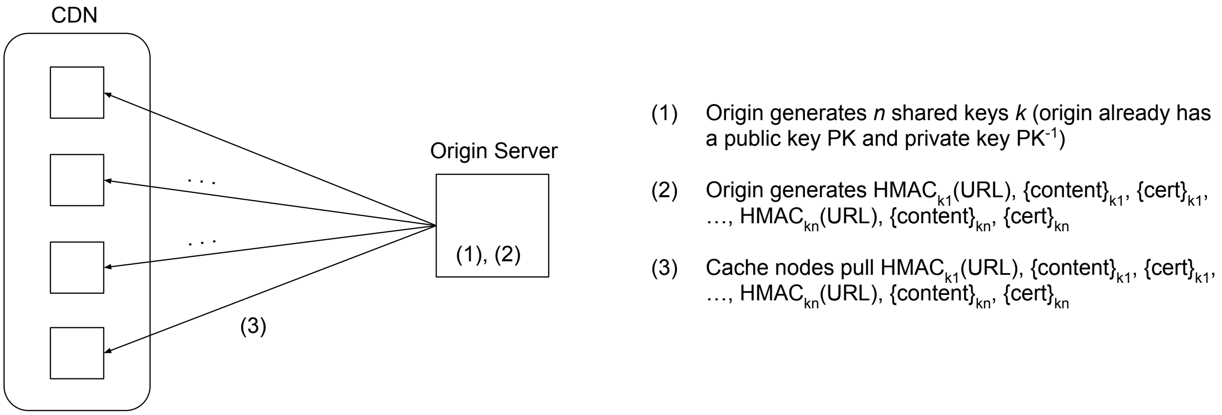

CDN
Origin Server
(1), (2)
. . .
. . .
(3)
(1) Origin generates n shared keys k (origin already has a public key PK and private key PK^{-1})
(2) Origin generates HMAC$_{k1}$(URL), {content}$_{k1}$, {cert}$_{k1}$, …, HMAC$_{kn}$(URL), {content}$_{kn}$, {cert}$_{kn}$
(3) Cache nodes pull HMAC$_{k1}$(URL), {content}$_{k1}$, {cert}$_{k1}$, …, HMAC$_{kn}$(URL), {content}$_{kn}$, {cert}$_{kn}$

Figure 3: *Step-by-step instructions on how content is published in Oblivious CDN.*

between 1 and the number of proxies. We reason about the security of different values of n in Section **??**.

Once all keys are established, the publisher must first pad the content to the same size for some range of original content sizes (i.e., if content is between length x and y, then pad it to length z). This content padding is done to hide the original content's length, as it may be identifiable simply by its length. After content is padded, then the content is divided into fixed size blocks and padded to some standard length. Then for each shared key k, each block is encrypted using the shared key, such that there are n sets of encrypted blocks. As long as the CDN does not have access to any of the n shared keys, then the CDN cannot see what content it is caching.

Now that the content is obfuscated, the publisher must also obfuscate the content's identifier. To do so, she computes the HMAC of the URL using the shared key k, for each shared key.

Once the identifier and the content replicas are obfuscated n times (with n keys), they can be pushed to the CDN. To increase reliability, performance, and availability, the publisher can push the content to multiple CDNs; a publisher can use a service, such as Cedexis [6], to load balance between CDNs. We discuss the use of multiple CDNs more in Section 5.3 on Oblivious CDN in partial deployment. Note that each proxy will only be able to fetch a specific replica of the content, that is a specific $\{\text{content}\}_{kn}$ for the n^{th} shared key that it holds. We discuss the security and performance trade offs associated with differing numbers of shared keys and proxies in Sections **??** and 6.

5.1.1 Updating Content

For a content publisher to update content, she must follow similar steps as described in publishing content. Once she has updated the content on her origin server, she must obfuscate it using the same steps: 1) padding the original content length, 2) divide the content into fixed size blocks, and 3) encrypt n copies of the content blocks with each of the shared keys. Because she is updating the content (as opposed to creating new content), the obfuscated identifier will remain the same. She must retain a copy of the obfuscated old content until after the new content has been updated on the CDN; this is to prove that the old content owner is the same as the new content owner. Only the origin and the proxy, both of which are outside the CDN, know the old obfuscated content, so an attacker cannot update the content that belongs to a legitimate publisher. The publisher must present the old obfuscated content to the CDN in order to also push her new obfuscated content to the CDN.

5.2 Retrieving Content

The steps taken for an end-user to retrieve a web page that has been cached by Oblivious CDN are shown in Figure 4. The end-user must first configure her browser to use an Oblivious CDN-designated proxy. A client is assigned to a specific proxy, and she configures her browser to use the assigned proxy. Then, once she sends a request for a web page, it goes to the proxy via a TLS connection.[2] The proxy then resolves the domain using it's local resolver, which will redirect it to the CDN's DNS resolver.

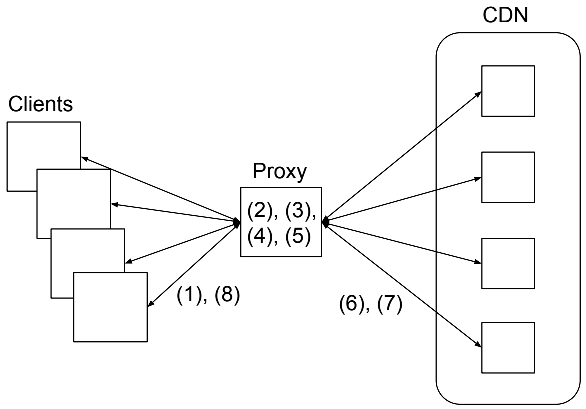
In order for the proxy to generate the obfuscated identifier to query the edge server for the correct content, it must have one of the n shared keys that the origin server generated and obfuscated the content and identifier with. The origin server publishes the shared key encrypted with the proxy's public key[3] in the DNS SRV record; therefore, when the proxy sends a DNS request to the origin server's authoritative DNS server, it will receive the encrypted shared key, which it can decrypt with it's private key.

Now that the proxy has obtained a shared key from the origin server, it can generate the obfuscated content identifier based on the request the client sent. It computes the HMAC of the URL with the shared key. The proxy then sends the (obfuscated) request to the edge server, where the CDN locates the content associated with the identifier. The CDN returns the associated obfuscated content, which we recall is the fixed size blocks encrypted with the same shared key that the identifier was obfuscated with. The proxy can decrypt the content blocks with the shared key from the origin server, assemble the blocks, and strip any added padding, to reconstruct the original content.[4] Finally, the proxy returns the content to the client over TLS.

[2] Comment: Add blinding here such that client1's request for foo.com and client2's request for foo.com looks different as it goes from the client to the proxy. So the proxy must be able to un-blind the request.

[3] Additionally, the origin server can learn the proxy's public key via DNS as well; for example, the proxy can publish it's public key in the DNS SRV record.

[4] Comment: Proxy can cache content in times of flash crowd to minimize correlation attacks if a provider has encrypted and unencrypted content on the same CDN. This raises the issue of charges for the origin (the CDN canâĂŹt charge as much if edge servers donâĂŹt see as many requests for the origin) — RFC 2227 describes a solution for this [19].



(1) Send GET foo.com request to proxy using TLS connection
(2) DNS lookup from proxy for foo.com
(3) CDN DNS lookup for a19.akamai.net (some Akamai ID that represents foo.com)
(4) Proxy sends DNS request to origin's authoritative server, and the origin publishes $\{k\}_{PK(proxy)}$ in the SRV record. Then the proxy decrypts the shared key with his own private key
(5) Proxy generates GET HMAC$_k$(URL) request
(6) Proxy sends request to cache node
(7) Cache node returns $\{content\}_k$, $\{cert\}_k$ to proxy. Proxy decrypts and validates the cert. Once the cert is validated, proxy decrypts the content with origin server's shared key
(8) Proxy returns decrypted content to client (using TLS)

Figure 4: *Step-by-step instructions on how content is retrieved in Oblivious CDN.*

5.3 Partial Deployment

Oblivious CDN should be partially deployable in the sense that if only some of the content publishers participate or only some of the CDNs participate, then the system should still provide protections. We have two different partial deployment plans, and both provide protections for those publishers, CDNs, and clients that use Oblivious CDN.

Plan 1. One option for deploying Oblivious CDN is to ensure there is some set S of content publishers the participate fully in the system. These publishers obfuscate their content, identifiers, and certificates, and most importantly, only have obfuscated data stored on the CDNs cache nodes. Recall that there are n shared keys, resulting in n replicas of the content that *appear* to the CDN as different content (because each replica is encrypted with a different key). This allows the minimum set of publishers S to be relatively small. We discuss the security tradeoffs with different values of S in Section **??**. This partial deployment plan protects the set of content publishers S and it partially protects the privacy of the clients accessing the content created by the set of publishers S. It does not protect the clients' privacy as completely as full participation of all publishers in Oblivious CDN because the CDN can still view cross site browsing patterns among the publishers that are not participating. It is important to note though, that because the clients are behind proxies, the CDN cannot individually identify users. The CDN can attribute requests to proxies, but not to clients.

Plan 2. It is reasonable to believe that some content publishers are skeptical of Oblivious CDN and prioritize performance and availability. Therefore, they should have the option to gradually move towards full participation by pushing both encrypted and plaintext content to the CDN. In this partial deployment plan, we see some set of publishers fully participating with only encrypted content, some other set of publishers partially participating with both encrypted and plaintext content, and some last set of publishers that are not participating. Unfortunately, if a publisher has both encrypted and plaintext content at a cache node, and some event causes a flashcrowd — the CDN sees a significantly larger spike in accesses to certain content — then the CDN can correlate the access spike on encrypted and plaintext content for the same publisher. In order to prevent this deanonymization of the content publisher, we can utilize multiple CDNs. The publisher can spread replicas over different CDNs such that the encrypted replicas are on one CDN and the plaintext replicas are on a different CDN. In this case the publisher is not susceptible to flashcrowds correlations and can still partially join the system.

5.4 Optimizations

While there are some optimizations that CDNs typically perform today that would not be possible with Oblivious CDN, the architecture of Oblivious CDN allows for new optimizations that are not possible in existing CDNs. Here we first outline some ways in which Oblivious CDN can be optimized in terms of performance, and then we point out what performance enhancements CDNs would not be able to do with Oblivious CDN.

Pre-Fetch DNS Responses. One way to increase the performance of Oblivious CDN is to pre-fetch DNS responses at the proxies. This would allow the proxy to serve each client request faster because it would not have to send as many DNS requests. Pre-fetching DNS responses would not take up a large amount of space, but it also would not be a complete set of all DNS responses. Additionally, if the content is moved between cache nodes at the CDN, then DNS response must also change; therefore, the pre-fetched DNS responses should have a lifetime that is shorter than the lifetime of the content on a cache node.

Load Balance Proxy Selection. As the proxy performs a number of operations on the client's behalf, it runs into the possibility of being overloaded. With Oblivious CDN, a client can be redirected to different proxies based on load; this can be implemented with a PAC file, which allows a client to access different proxies for different domains. In addition to being a performance benefit, this could also prevent a country from blocking the set of proxies that all of the country's citizens use; if this occurs, then the citizens can be redirected to a different proxy.

On the other hand, CDNs become more limited in some of their actions when following Oblivious CDN's design. For example, many CDNs perform HTTPS re-writes on content that they cache, but this can only be done if the CDN has access to the decrypted content. Similarly, the CDN needs the decrypted content to perform minimizations on HTML, CSS, and Javascript files. Any algorithms used internally to distribute content to certain caches based on what the content is can no longer be used in Oblivious CDN because the CDN does not know what the content is.

6 Performance Analysis

We evaluate the performance impact of Oblivious CDN in comparison to a CDN that does not perform extra cryptographic operations by simulating both systems using a series of Virtual Private Servers (VPSs). The design of Oblivious CDN will likely affect the time that it takes a content publisher to publish her data as the data has to obfuscated. Therefore, we measure the time it takes to obfuscate data. Additionally, Oblivious CDN may affect the performance of web page retrieval; therefore, we measure the time to retrieve a objects of varying sizes. Additionally, we simulate and analyze cache hit rates with varying numbers of shared keys k, as a higher number of shared keys will result in a lower cache hit rate, and thus negatively affect performance. Analyzing the relationship between cache hit rates and the number of shared keys can help determine the optimal number of shared keys that should be used in Oblivious CDN.

When implementing Oblivious CDN, we used SHA-256 for the HMAC implementation and AES-128 for symmetric key cryptographic operations. We used the `cryptography` library and implemented all cryptographic operations in Python.

6.1 Obfuscation Overhead: Publishing

As content publishers must compute the $\text{HMAC}_k(\text{URL})$ and $\{\text{content}\}_k$, the act of publishing potentially takes longer than not having to perform any cryptographic operations. In this experiment, we analyze the overhead the publisher faces when obfuscating her content.

6.1.1 Metrics

When comparing the time to publish using Oblivious CDN to a typical CDN, the only difference is the obfuscation of data (as discussed in Section 5.1). This results in our analysis simply of the obfuscation; the metric of interest is the total time it takes to encrypt content.

6.1.2 Experimental Setup

In our experiment, we setup a client, a proxy, and an edge server. For this experiment, the only machine necessary is the edge server — we are not concerned with the amount of time it takes for the content to be transmitted from the publisher's origin server to the edge server, only the amount of time the content publisher takes to obfuscate the data. Therefore, the content and URL are obfuscated on the edge server, and the time it takes to do these operations is measured. This measurement is taken for obfuscating the following file sizes: .000001MB, .001MB, .01MB, .1MB, 1.0MB, 10.0MB, and 100.0MB.

6.1.3 Results

The results of time to obfuscate data are shown in Figure 5. We can see that the overhead is relatively small for file sizes less than 100.0MB. In addition, this publishing overhead has no performance impact on a client's experience of retrieving, as the client never has to encrypt the content.

6.2 Obfuscation Overhead: Retrieving

There is also some overhead associated with retrieving objects when using Oblivious CDN due to the operations that the proxy must perform.

6.2.1 Metrics

We are interested in measuring the affect of cryptographic functions on the retrieval of single objects. The metric used to do so is the end-to-end time to download a single object.

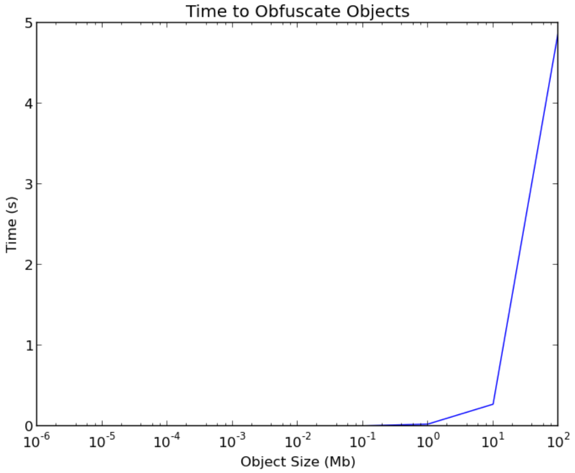


Figure 5: *The time it takes to encrypt/obfuscate different sizes of objects in Oblivious CDN.*

6.2.2 Experimental Setup

In our experiment, we setup a client, a proxy, and an edge server. The client is a Fujitsu CX2570 M2 servers with dual, 14-core 2.4GHz Intel Xeon E5 2680 v4 processors with 384GB RAM running the Springdale distribution of Linux machine, and both the proxy and edge servers are VPSs running Ubuntu 12.04 x64 LTS with 10GB of storage. The proxy is located in Montreal, Canada, and the edge server is located in Chicago, USA. To account for any additional latency on the links between the client and proxy, or proxy and edge server, we treat a traditional CDN as a system of client, proxy, and edge server, where the edge server acts as a simple proxy and does not perform any operations.

First, different sizes of objects are generated, encrypted, and obfuscated on the edge server; the sizes of objects we create are: .000001MB, .001MB, .01MB, .1MB, 1.0MB, 10.0MB, and 100.0MB. These are the objects that will be downloaded by the client, and we assume all objects are in the cache, as the time it takes the CDN to fetch the content from the origin server (if there is a cache miss) should not differ between Oblivious CDN and a traditional CDN.

Our proxy runs `mitmproxy`, which allows for interception and manipulation of HTTP requests and responses. We implement these modifications at the proxy.

In this experiment, the client machine requests the objects using `curl` with specific options to use the designated proxy, and to measure the transfer time.

6.2.3 Results

After conducting the experiment, we find the results shown in Figure 6. We can see that total download times are comparable for objects that are smaller than 10MB; objects larger than 10MB take slightly longer to download using Oblivious CDN than a traditional CDN, due to the cryptographic operations performed at the proxy.

Additionally, recent work has reported that the average object size retrieval from Akamai is only 335 Kb, and the average object size of popular objects (> 200,000 requests) is only 8.6 Kb [3]. Both of these object sizes are less than 10MB, resulting in negligible performance differences between Oblivious CDN and a traditional CDN.

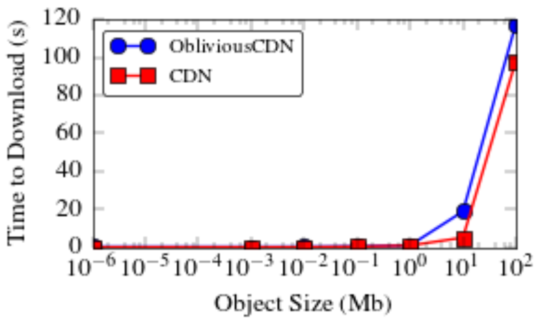


Figure 6: *The total transfer time for accessing different sizes of objects using Oblivious CDN and without Oblivious CDN's cryptographic operations.*

6.3 Cache Hit Rates

The other reason for lower performance in Oblivious CDN is because there are less copies of the content that a single proxy can request. As each set of proxies use a different shared key, and these proxies can only request the content encrypted with their shared key, this can lead to high cache misses. This would cause the CDN to fetch the object from the content publisher's server, which increases the latency, and degrades performance for the client. In this experiment, we measure just how much of an impact the number of shared keys has on the cache hit rate.

6.3.1 Metrics

In addition to the use of proxies and cryptographic operations, Oblivious CDN could suffer performance losses from lower cache hit ratios. As the number of shared keys (between the proxies and the content publishers' servers) increases, the cache hit ratio will likely decrease. The metric we are interested in to evaluate the performance impact of multiple shared keys is the cache hit ratio.

6.3.2 Experimental Setup

Simulate the cache hit ratio with different numbers of shared keys k ($k = 1$ to $k = 100$) when using a cache size of 1.2 GB [4] and a sequence of object requests from ??, where we assume each object to be 100 Kb [3].

6.3.3 Results

7 Discussion

7.1 Political Push Back

7.2 Law Enforcement

7.3 Private CDNs

Such as Netflix, Facebook, Google — Oblivious CDN doesn't work for them because they are the content publisher *and* the CDN.

7.4 Proxy Selection

To provide more mixing than the proxy already provides, the proxy that is used could be selected based on a Proxy Auto Configuration (PAC) file.

7.5 Active Measurements + Overreaching Subpoena

Spoofing origin server content updates. An attacker who spoofs origin server content updates.

Guessing attacks. An attacker who is attempting to learn if a client is accessing a specific URL or not by submitting requests to see if they match other clients' requests (this should be fixed by adding blinding).

8 Related Work

To our knowledge, there has been no prior work on preventing surveillance at CDNs, but there has been relevant research on securing CDNs, finding security vulnerabilities in CDNs, and conducting different types of measurements on CDNs.

Securing CDNs. Most prior work on securing CDNs has focused on providing content integrity at the CDN as opposed to content confidentiality (and unlinkability). In 2005, Lesniewski-Laas and Kaashoek use SSL-splitting — a technique where the proxy simulates an SSL connection with the client by using authentication records from the server with data records from the cache (in the proxy) — to maintain the integrity of content being served by a proxy [13]. While this work does not explicitly apply SSL-splitting to CDNs, it is a technique that could be used for content distribution. Michalakis et. al., present a system for ensuring content integrity for untrusted peer-to-peer content delivery networks [16]. This system, Repeat and Compare, use attestation records and a number of peers act as verifiers. More recently, Levy et. al., introduced Stickler, which is a system that allows content publishers to guarantee the end-to-end authenticity of their content to users [14]. Stickler includes content publishers signing their content, and users verifying the signature without having to modify the browser. Unfortunately, systems like Stickler do not protect against an adversary that wishes to learn information about content, clients, or publishers; Oblivious CDN is complementary to Stickler.

There has been prior work in securing CDNs against DDoS attacks; Gilad et. al., introduce a DDoS defense called CDN-on-Demand [9]. In this work they provide a complement to CDNs, as some smaller organizations cannot afford the use of CDNs and therefore do not receive the DDoS protections provided by them. CDN-on-Demand is a software defense that relies on managing flexible cloud resources as opposed to using a CDN provider's service.

Security Issues in CDNs. More prevalent in the literature than defense are attacks on CDNs. Recent work has studied how HTTPS and CDNs work together (as both have been studied extensively separately). Liang et. al., studied 20 CDN providers and found that there are many problems with HTTPS practice in CDNs [15]. Some of these problems include: invalid certificates, private key sharing, neglected revocation of stale certificates, and insecure back-end communications; the authors point out that some of these problems are fundamental issues due to the man-in-the-middle characteristic of CDNs. Similarly, Zolfaghari and Houmansadr found problems with HTTPS usage by CDNBrowsing, a system that relies on CDNs for censorship circumvention [27]. They found that HTTPS leaks the identity of the content being accessed, which defeats the purpose of a censorship circumvention tool.

Research has also covered other attacks on CDNs, such as flash crowds and denial of service attacks; Jung et. al., show that some CDNs might not actually provide much defense against flash events (and they differentiate flash events from denial of service events) [12]. Su and Kuzmanovic show that some CDNs are more susceptible to intential service degradation, despite being known for being resilient to network outages and denial of service attacks [21]. Additionally, researchers implemented an attack that can affect popular CDNs, such as Akamai and Limelight; this attack defeats CDNs' denial of service protection and actually utilizes the CDN to amplify the attack [22]. In the past year, researchers have found forwarding loop attacks that are possible in CDNs, which cause requests to be served repeatedly, which subsequently consumes resources, decreases availability, and could potentially lead to a denial of service attack [7].

Recently, researchers have studied the privacy implications of peer-assisted CDNs; peer-assisted CDNs allow clients to cache and distribute content on behalf of a website. It is starting to be supported by CDNs, such as Akamai, but the design of the paradigm makes clients susceptible to privacy attacks; one client can infur the cross site browsing patterns of another client [11].

Measuring and Mapping CDNs. As CDNs have increased in popularity, and are predicted to grow even more [26], much research has studied the deployment of CDNs. Huang et. al., have mapped the locations of servers, and evaluated the server availability for two CDNs: Akamai and Limelight [10]. More recently, Calder et. al., mapped Google's infrastructure; this included developing techniques for mapping, enumerating the IP addresses of servers, and identifying associations between clients and clusters of servers [5]. Scott et. al., develop a clustering technique to identify the IP footprints of CDN deployments; this analysis also analyzes network-level interference to aid in the identification of CDN deployments [18]. In 2017, researchers conducted an empirical study of CDN deployment in China; they found that it is significantly different than in other parts of the world due to their unique economic, technical, and regulatory factors [25].

Other measurement studies on CDNs have focused on characterizing and quantifying flash crowds on CDNs [24], inferring and using network measurements performed by a large CDN to identify quality Internet paths [20], and measuring object size distributions and request characteristics to optimize caching policies [4].

9 Conclusion

References

[1] Akamai Empowers Operators to Deploy Their Own Content Distribution Network. `https://www.akamai.com/us/en/resources/content-distribution-network.jsp`.

[2] Akamai: Facts & Figures. `https://www.akamai.com/us/en/about/facts-figures.jsp`.

[3] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter. Achieving high cache hit ratios for cdn memory caches with size-aware admission. 2016.

[4] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter. Adaptsize: Orchestrating the hot object memory cache in a content delivery network. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 483–498. USENIX Association, 2017.

[5] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan. Mapping the expansion of google's serving infrastructure. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 313–326. ACM, 2013.

[6] Cedexis. `https://www.cedexis.com/`.

[7] J. Chen, J. Jiang, X. Zheng, H. Duan, J. Liang, K. Li, T. Wan, and V. Paxson. Forwarding-loop attacks in content delivery networks. In *Proceedings of the 23st Annual Network and Distributed System Security Symposium (NDSS'16)*, 2016.

[8] CREDO and Cloudflare Argue Against National Security Letter Gag Orders. `https://techcrunch.com/2017/03/23/credo-and-cloudflare-argue-against-national-securit`

[9] Y. Gilad, A. Herzberg, M. Sudkovitch, and M. Goberman. Cdn-on-demand: an affordable ddos defense via untrusted clouds. In *Network and Distributed Security Symposium (NDSS)*, 2016.

[10] C. Huang, A. Wang, J. Li, and K. W. Ross. Measuring and evaluating large-scale cdns. In *ACM IMC*, volume 8, 2008.

[11] Y. Jia, G. Bai, P. Saxena, and Z. Liang. Anonymity in peer-assisted cdns: Inference attacks and mitigation. *Proceedings on Privacy Enhancing Technologies*, 2016(4):294–314, 2016.

[12] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *Proceedings of the 11th international conference on World Wide Web*, pages 293–304. ACM, 2002.

[13] C. Lesniewski-Laas and M. F. Kaashoek. Ssl splitting: Securely serving data from untrusted caches. *Computer Networks*, 48(5):763–779, 2005.

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

[15] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu. When https meets cdn: A case of authentication in delegated service. In *Security and Privacy (S&P), 2014 IEEE Symposium on*, pages 67–82. IEEE, 2014.

[16] N. Michalakis, R. Soulé, and R. Grimm. Ensuring content integrity for untrusted peer-to-peer content distribution networks. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, pages 11–11. USENIX Association, 2007.

[17] Microsoft Offers EU Customers Option to Store Data in Germany. `https://www.wsj.com/articles/microsoft-tightens-eu-clients-data-protection-14472`

[18] W. Scott, T. Anderson, T. Kohno, and A. Krishnamurthy. Satellite: Joint analysis of cdns and network-level interference. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 195–208. USENIX Association, 2016.

[19] Simple Hit-Metering and Usage-Limiting for HTTP. `https://www.ietf.org/rfc/rfc2227.txt`.

[20] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante. Drafting behind akamai: Inferring network conditions cased on cdn redirections. *IEEE/ACM Transactions on Networking (TON)*, 17(6):1752–1765, 2009.

[21] A.-J. Su and A. Kuzmanovic. Thinning akamai. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, pages 29–42. ACM, 2008.

[22] S. Triukose, Z. Al-Qudah, and M. Rabinovich. Content delivery networks: Protection or threat? In *European Symposium on Research in Computer Security*, pages 371–389. Springer, 2009.

[23] Victory for CloudFlare Against SOPA-like Court Order: Internet Service Doesn't Have to Police Music Labels' Trademark. `https://www.eff.org/deeplinks/2015/07/victory-cloudflare-against-sopa-court-order-internet-service-doesnt-have-police`.

[24] P. Wendell and M. J. Freedman. Going viral: flash crowds in an open cdn. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 549–558. ACM, 2011.

[25] J. Xue, D. Choffnes, and J. Wang. Cdns meet cn an empirical study of cdn deployments in china. *IEEE Access*, 2017.

[26] The Zettabyte Era âĂŤ Trends and Analysis âĂŞ Cisco. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html`.

[27] H. Zolfaghari and A. Houmansadr. Practical censorship evasion leveraging content delivery networks. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1715–1726. ACM, 2016.