Delegation of TLS Authentication to CDNs using Revocable Delegated Credentials

Daegeun Yoon
ETRI/KAIST
Daejeon, South Korea
dayoon@etri.re.kr

Taejoong Chung
Virginia Tech
Blacksburg, VA, USA
tijay@vt.edu

Yongdae Kim
KAIST
Daejeon, South Korea
yongdaek@kaist.ac.kr

ABSTRACT

When using a Content Delivery Network (CDN), domain owners typically delegate Transport Layer Security (TLS) authentication to the CDN by sharing their TLS certificate's private key. However, this practice not only delegates TLS authentication but also grants the CDN complete control over the certificate. To mitigate these concerns, Delegated Credential (DC) was proposed as a solution; DC, which contains both the CDN's public key and the domain owner's signature, allows the domain owners to delegate their own credentials for TLS authentication, thereby avoiding the need to share their private keys. However, the absence of a mechanism to distribute the revocation status of a DC renders it non-revocable, even when a compromise of a credential has been detected. DCs were thus designed to be short-lived, necessitating frequent renewal for continued use.

To overcome this limitation, we designed Revocable Delegated Credential (RDC), which provides a revocation method for DCs. With RDCs, there is no need for frequent renewals as they can be revoked, allowing for a longer validity period. The revocation status of RDCs is distributed via DNS, an essential component of web communication. RDCs utilize the NSEC record, a type of DNSSEC record, as a means to store, validate, and easily manage their revocation status. When domain owners no longer trust their CDNs or detect compromise in their RDCs, they can distribute the RDC's revocation status by simply creating a subdomain named with an RDC identifier. The browser then confirms the existence of this subdomain using the NSEC record to validate the revocation status. We implemented RDC in the go tls package and Firefox Nightly to demonstrate and evaluate its feasibility.

1 INTRODUCTION

A Content Delivery Network (CDN) is a distributed network of servers that domain owners use to improve website performance, reliability, and cost-efficiency. By caching website contents on multiple servers that are geographically closer to end-users, CDNs can reduce latency, leading to significant improvements in website performance. Additionally, CDNs use load balancing and redundancy mechanisms to provide reliable content delivery even during server failures; this enhances the user experience, improve website availability, and lower infrastructure costs.

However, the combination of HTTPS and CDNs can create challenges due to their inherent incompatibilities. HTTPS is a communication protocol that is authenticated and encrypted by Transport Layer Security (TLS). For a CDN to operate a service through HTTPS, the CDN needs to perform TLS authentication using the domain owner's TLS certificate. However, the TLS protocol assumes end-to-end communication and does not offer an explicit method of delegation, which makes it challenging for the domain owners to delegate their TLS certificates to CDNs. Consequently, domain owners delegate TLS authentication to CDNs either by allowing CDNs to generate TLS certificates and private keys on their behalf or by sharing their own TLS certificates and private keys with the CDNs. [1, 2].

Sharing the TLS certificate's private key with a CDN not only poses theoretical security risks but also creates practical challenges when a domain owner decides to stop using a CDN (i.e., due to loss of trust) or when the delegated key had been compromised. If the CDN generates the TLS certificate on behalf of the domain owner, which is a common method for sharing the TLS certificate's private key, the domain owner may not have control over the TLS certificate, preventing the domain owner from being able to revoke the certificate. Even if the domain owner generates the TLS certificate himself and then shares the certificate and its private key with the CDN (allowing the domain owner to have control over the TLS certificate and revoke it as needed), proper revocation of the certificate is still not guaranteed because the revocation status of the TLS certificate may not be correctly delivered to the browser [3–6]. In such cases, the domain owner must accept the potential risk of an untrusted third party possessing access to TLS authentication during the certificate's remaining validity period.

Several solutions [1, 7–18] have been proposed for delegating TLS authentication without sharing the TLS certificate's private key. Cloudflare has proposed Keyless SSL [7], which allows TLS authentication without sharing the TLS certificate's private key by introducing a key server under the domain owner's control. However, this necessitates that the domain owner maintain a highly available key server for uninterrupted service, adding an additional responsibility for domain name owners. Other methods [9–18] have been proposed to protect not only the TLS authentication key but also the TLS encryption key during delegation. However, such methods have trade-offs such as performance degradation, high deployment costs, and limitations in using the full functionality of CDNs, which ultimately reduce the benefits of using CDNs.

Mozilla, Facebook, Cloudflare, and Cisco have proposed a recent advancement called Delegated Credential (DC) [8], which allows domain owners to delegate TLS authentication by issuing their own credentials that contain the CDN's public key and the domain owner's signature. DC enables the signing and verification of TLS authentication using its key pair, and therefore, the domain owners do not need to share their TLS certificate's private key. However, because DCs do not have a storage that can be used to store and distribute their revocation status, they cannot be revoked. Inevitably, DCs were designed to be short-lived to allow for expiration at the end of their short lifespan. Since DCs are short-lived credentials,

domain owners are required to maintain an issuance server capable of issuing DCs frequently, often up to every 7 days, which diminishes the advantages of utilizing CDNs. Consequently, we observed that only 29 domains out of Tranco's top 100k [19] are utilizing DCs, signifying a relatively small fraction of delegations that use this method [20].

In this work, we propose Revocable Delegated Credential (RDC), which provides a revocation method for DC. Similar to DCs, RDCs can be utilized to sign and verify TLS authentication messages using their own key pair, eliminating the need for the domain owners to share their TLS certificate's private key. What sets RDCs apart from DCs is the added capability for the domain owners to revoke their RDCs in situations where they no longer wish to use a CDN (i.e., due to trust issues) or if the RDC's key had been compromised.

The primary challenge with RDCs is finding a reliable and secure method of distributing their revocation status. To address this challenge, we leveraged the DNS infrastructure. We ensured that the revocation status of RDCs are protected by Domain Name System Security Extensions (DNSSEC) [21] and DNS-over-HTTPS (DoH) [22] infrastructure during distribution, which in turn ensures reliability, integrity, and confidentiality. To avoid complications with deployment and distribution of the revocation status of the RDCs, we leveraged a NSEC record, a type of DNSSEC record, to store and validate the revocation status of the RDCs. Each RDC is assigned a unique identifier and its revocation status is determined by the existence of a subdomain with the same name as this RDC identifier. Domain owners have the convenience of determining the revocation status by simply adding or removing a subdomain that corresponds to the RDC identifier.

We Implemented RDC in a real-world environment to demonstrate its feasibility. First, we developed an RDC within the TLS package in Go and the NSS library in Firefox Nightly. This implementation allowed us to confirm that a TLS server can operate with an RDC's private key instead of the TLS certificate's private key (Of note, this is also a key feature of DC). Furthermore, we successfully validated that a domain owner is able to revoke his RDC by distributing its revocation status via DNS. During our evaluation, we observed that TLS handshake using an RDC introduces about 100 ms delay in both the TLS setup time and the page load time (PLT). This delay can be mitigated by caching of the revocation status in the DNS resolver, which allows the revocation status to be more easily accessed. It is worth noting that other solutions that protect the TLS encryption layer incur their overhead with each TLS communication; however, the RDC, which protects only the TLS authentication, incurs overhead only once during the entire TLS communication. Our contributions are summarized as follows:

- We propose a new design for delegation of TLS authentication called RDC, which has similar benefits to DCs but has an added feature of revocability, which is achieved by utilizing the DNS infrastructure, an essential part of web communication.
- We utilize a NSEC record, a type of DNSSEC record, to distribute the revocation status of RDCs while maintaining their integrity. Domain owners can easily determine the revocation status by adding or removing a subdomain named with the RDC identifier.
- We implemented a prototype of RDC into the Go tls package and Firefox Nightly, and publicly released the source codes[1].
- Our evaluation showed that RDC introduces an acceptable amount of delay to the TLS setup time and the PLT.

2 BACKGROUND

2.1 Content Delivery Network

CDNs are third-party vendor services that host domain owners' websites and the data on their servers on behalf of the domain owners. CDN infrastructures consist of regional data centers, known as PoPs (points of presence), that are responsible for communicating with nearby users. In general, each PoP contains several edge servers that act as reverse proxy web servers for caching, optimizing connections, and providing other content delivery capabilities. Delegating website operations to a CDN allows domain owners to provide end-users with reliable and high-performance web services.

2.2 Delegation to CDNs

Currently, the majority of HTTP packets are protected by TLS [23–25], and the protocol used to protect HTTP with TLS is called HTTPS. Thus, for domain owners who use HTTPS to protect their web services, delegation of the web service operation to a CDN also requires delegation of TLS authentication to the CDN.

TLS certificate. A TLS certificate is utilized for authentication of the identity of a domain owner. A TLS certificate binds a domain name to a public key, allowing the domain owner to prove ownership of the domain by demonstrating possession of the corresponding private key. Browsers store a list of their trusted certificate authorities (CAs) in the form of a root store and accept only these certificates that have been issued by the CAs registered in their root store as trustworthy [26–28].

TLS authentication. The TLS protocol consists of two stages: authentication and encryption. During the authentication phase, authentication is achieved by proving the domain owner's possession of the private key corresponding to the TLS certificate.

Here, we provide a concise summary of the TLS authentication procedure. First, the domain owner's TLS server sends an `Authentication` message in response to the browser's `ClientHello`. This Authentication message consists of a Certificate message containing the domain owner's TLS certificate and a `CertificateVerify` message signed with the private key of the TLS certificate. The browser then completes the authentication process in three steps. First, it verifies whether the TLS certificate in the Certificate message was issued by a trusted CA. Second, it checks whether the domain name specified in the certificate matches the domain name that the browser is trying to access. Finally, it validates the CertificateVerify message using the TLS certificate's public key to confirm that the domain owner possesses the private key of the TLS certificate.

Once the TLS authentication process is finished, both the origin server and the browser will create a symmetric encryption key (session key) based on the parameters negotiated during the TLS handshake, which will be used to establish a secure connection between them.

[1]The source code of the Go tls package and the NSS library are available at https://github.com/revtls/revtls.

Lack of delegation mechanism. CDNs are placed between the domain owner's origin server and the end-users' browsers to optimize content delivery. However, the communication model of CDNs is incompatible with TLS, which is designed to provide end-to-end authentication and encryption. Neither the X.509 standard [29] (the standard form of TLS certificate) nor the TLS standard [30] supports explicit delegation. As a result, many domain owners are resorting to abnormal means, such as sharing the TLS certificate's private keys, in order to delegate their TLS authentication to CDNs [2].

2.3 Current Practices of Delegation

Sharing TLS certificate's private keys. Sharing the TLS certificate's private key is a commonly used method for delegating TLS authentication to CDNs today [2]. There are two main methods by which CDNs and domain owners can share the certificate's private keys. The first method involves the CDN generating and managing the certificate and the private key on behalf of the domain owner. While this method may be convenient for domain owners, several CDNs providing this method may not authorize the domain owners to access their certificates and private keys, thereby making it challenging for the domain owners to revoke them. For instance, a CDN can provide the domain owner with an option to deactivate a shared certificate through its platform, rather than providing an option to revoke the certificate. To test whether the deactivation of a shared certificate successfully revokes the certificate, we conducted a simple test on Cloudflare's CDN [31], which is the largest CDN by customer count [32, 33]. We first generated a certificate via Cloudflare's CDN, then deactivated this shared certificate, and afterwards checked its status on the Censys database [34], which shows the status of every issuance of TLS certificate by trusted CAs. As expected, the test revealed that the deactivated certificate was not revoked until expired [2]. Therefore, in these situations, even if a domain owner chooses to stop using a particular CDN due to loss of trust, he would be unable to stop sharing the TLS certificate with the CDN, which would have become an untrusted third party.

A second method involves the domain owner generating his own TLS certificate and private key and uploading them to the CDN provider. In this case, the domain owner can directly request the revocation of the TLS certificate from the CA. However, several studies have demonstrated that revocation checking of the TLS certificate is not carried out properly [1, 3–5, 35]. Thus, when a domain owner stop using a CDN, or when a delegated private key becomes exposed by security breaks in the CDN, the domain owner may become vulnerable to exposure of the delegation key until the certificate expires. Fortunately, there has not yet been any reported incident of compromised private keys; however, attacks targeting CDNs or vulnerabilities of CDNs have been consistently discovered [36–39].

Keyless SSL. Cloudflare has proposed Keyless SSL [7], which enables domain owners to delegate TLS authentication without sharing the TLS certificate's private keys, by allowing domain owners to manage their private keys on a key server. When a CDN receives a TLS authentication request from a browser, it forwards the request to the domain owner's key server, which performs TLS authentication using the TLS certificate's private key and returns the result to the CDN. Accordingly, in order to ensure seamless web service, domain owners require a highly available key server capable of reliably processing every TLS authentication request sent by the browser via CDN. Thus, certain benefits of using CDNs, such as high web service availability and operational convenience, are lost. Contrary to its original purpose, Keyless SSL is currently being used within the CDN infrastructure to manage domain owners' TLS certificates and private keys [40].

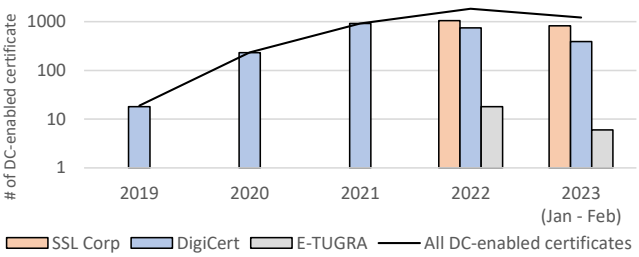

Figure 1: The number of DC-enabled TLS certificates issued by trusted CAs and registered in CT logs from 2019 to 2023.

Delegated Credentials. The IETF has developed an Internet Draft called Delegated Credential (DC) [8] that allows domain owners to issue their own credentials containing CDN's public key and the domain owner's signature, enabling TLS authentication without sharing of the TLS certificate's private key. In the process of TLS authentication, a key pair associated with the DC can be used to sign and verify a CertificateVerify message, replacing the TLS certificate's key pair. The TLS certificate's key pair is used to verify the signature in the DC, ensuring that the DC was legitimately issued by the correct domain owner.

Yet, DC has a significant drawback that limits its effectiveness: DC does not provide a mechanism to store its revocation status. Domain owners have no method of revoking the DC when they stop using a CDN or when a compromise of DC is detected. Implicit revocation of a DC is possible by revoking the TLS certificate that was used to sign the DC. However, if revocation checking of TLS certificates is not carried out properly, as previous studies have pointed out [1, 3–5, 35], there is a possibility that attackers will continue to exploit the DC. As a result, DC was designed to be short-lived to limit its exposure. Due to the short-lived nature of DCs, with a maximum validity period of 7 days, the domain owners must maintain an available server that is capable of issuing a new DC every 7 days to avoid disruptions in service. This requirement diminishes the benefits of using CDNs to improve web server availability and operational convenience.

In order to evaluate how DC has been used since its initial draft in 2017, we analyzed DC-enabled TLS certificates that contain a special X.509 certificate extension for DC issuance (data obtained from the Censys [34] database). Figure 1 shows the number of the DC-enabled TLS certificates that were issued by trusted CAs. We observed a consistent increase in the number of DC-enabled TLS certificates being used since DC has been introduced in 2017.

[2]Cloudflare uses the Google Trust Service (GTS) CA certificate for the service. We contacted GTS to request revocation of the certificate and received a response that we need to revoke it ourselves using the ACME protocol. We attempted to revoke the certificate using DNS-challenge supported by ACME, but revocation failed due to an unauthorized error in ACME.

However, these certificates are still used far less than CDNs [2, 20]. Moreover, we discovered that out of more than 4,000 DC-enabled TLS certificates, only 29 certificates were for domains in the Tranco top 100K [19]. Based on these observations, we could infer that DC is currently only being used for a very small portion of delegations. While there may be numerous reasons for why DC is not being extensively utilized, the operational cost of issuing DC at frequent intervals and the lack of support for revocation may be among the most notable ones.

3 REQUIREMENTS AND DESIGN GOALS

3.1 Requirements

In the current practice, domain name owners relinquish control of their private key and certificate when they authorize a CDN to manage their domain. This situation becomes critical if domain name owners choose to discontinue using their CDNs or if their certificate becomes compromised. In both scenarios, the certificate managed by the CA must be promptly revoked. However, under the current practice, domain name owners lack the ability to do so themselves [1, 3–5, 35].

To address these issues, we establish five goals as follows.

G1: No sharing of the domain owner's private key. One of the important design goals of RDC is to maintain a key feature of DC, ensuring the TLS certificate's private key "private" by avoiding the need to share it with CDNs. RDC aims to achieve delegation through sharing of revocable delegation keys instead of the TLS certificate's private keys.

G2: Retaining control of revoking delegation keys. As soon as a compromise of a delegation key is detected, the domain owner must have the capability to promptly revoke the delegation key. RDC aims to enable the domain owner to autonomously decide on revocation status of the delegation key as needed.

G3: Revoking the delegation key without revoking the TLS certificate. By providing a method to revoke the delegation key without revoking the TLS certificate, it is possible to minimize the impact of the revocation checking of the TLS certificate on the revocation checking of the delegation key. RDC aims to allow the domain owner to distribute the revocation status of the delegation key without revoking the TLS certificate via the DNS infrastructure and its security mechanism.

G4: Compliance of RDC with the current standards and infrastructure. RDC aims to comply with the current standards and infrastructure of HTTPS, ensuring that HTTPS ecosystem participants do not engage in non-compliant behavior. Furthermore, RDC is designed using only the existing infrastructure of the HTTPS ecosystem, without requiring any additional deployments such as Trusted Execution Environments (TEEs), CT-like servers or special security modules.

G5: Retaining benefits of using a CDN. RDC aims to minimize performance degradation and enables the use of full functionality of CDNs.

Based on these goals, we propose RDC (Revocable Delegated Credential) that satisfies these five goals to achieve secure delegation of TLS authentication. In (§6), we will examine whether all five goals of RDC are met when implementing RDC in real-world scenario.

3.2 Threat Model and Assumption

We assume that domain owners stop using a CDN when they no longer trust it. Domain owners may believe that a CDN has a software or configuration bug that could result in unintended exposure of the private key corresponding to the delegation key. Alternatively, domain owners may believe that a rogue employee in the CDN could use the private key for passive or active attacks to obtain their sensitive data.

Even if the CDN remains trustworthy and the domain owner continues to use it, the delegation key could be exposed by one or more edge servers due to security incidents in the CDN. In such cases, we assume that the CDN will immediately alert the domain owner as soon as it detects an exposure of the delegation key.

We adopt the Dolev-Yao model [41], which assumes that an active adversary has full control over the network. The adversary is capable of capturing messages in transit and performing actions such as message modification, dropping, reordering, and injection. The adversary who obtains the delegation key can also perform cryptanalysis of the domain owner's traffic and even launch attacks such as injecting malicious code on the victim. We consider the scenario where an active attacker can manipulate a victim's web and DNS traffic, such as through a man-in-the-middle (MitM) attack.

However, we assume that the attacker does not possess the capability to compromise authoritative DNS servers and DNS resolvers. We also assume that the adversary is not capable of breaking standard cryptographic primitives.

4 REVOCABLE DELEGATED CREDENTIAL

4.1 Design

At a high level, our approach is to delegate TLS authentication to a CDN by sharing a *revocable delegation key*. The challenge, however, is to provide a method for reliable revocation checking for the delegation key while ensuring its compatibility with the existing HTTPS ecosystem. To address this challenge, we make several design decisions. First, we design RDC by extending the structure of DC to support revocation, which allows to share the revocable delegation key with a CDN. Second, we utilize DNSSEC and DoH to enable the domain owner to distribute the revocation status of the delegation key. Lastly, we extend the TLS protocol to deliver revocable delegation key during the TLS handshake process while complying with a set of current TLS standards [29, 30].

Figure 2 provides an overview of the RDC's design. The first step involves a CDN generating a delegation key pair (including a public and private key) and sending the public key to the domain owner. Next, the domain owner generates an RDC that includes the public key generated by the CDN, a unique identifier (which we refer to as a `RDC_serial`), an expiration date, and a signature signed by the domain owner using the TLS certificate's private key. The domain owner then issues the RDC along with the TLS certificate to the CDN, which the CDN deploys to its edge servers.

A CDN edge server can perform TLS handshake using the RDC's private key to sign the `CertificateVerify` message instead of the TLS certificate's private key. After then, the edge server sends the `CertificateVerify` message, the RDC, and the TLS certificate to the browser so that it can verify the `CertificateVerify` signature using the RDC's public key, and can verify RDC's signature using the

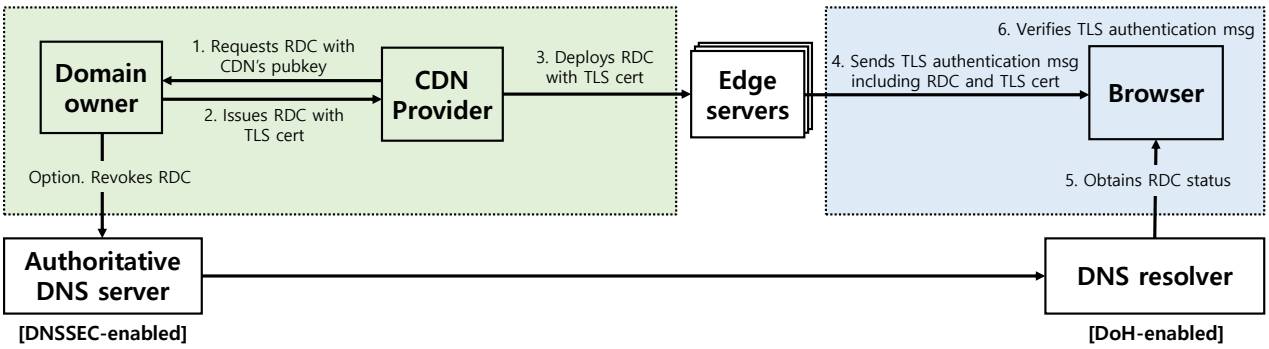

Figure 2: A high-level design of RDC. The domain owner signs an RDC (which contains the CDN's public key) using their TLS certificate's private key and sends it to the CDN. The CDN then deploys the RDC to its edge servers, which performs TLS authentication using the RDC's private key instead of the TLS certificate's private key. The domain owner can distribute the revocation status for the RDC by using an NSEC record, a type of DNSSEC record, and DoH to protect the revocation status.

TLS certificate's public key to confirm whether the RDC has been issued by the correct domain owner.

If the domain owner decides to stop using the CDN or detects a compromise of the private key corresponding to the RDC, the domain owner can revoke the RDC by creating a subdomain named with a specific `RDC_serial`. Existence of this subdomain indicates revocation of the RDC. During the TLS handshake, the browser can obtain the DNS record, which indicates existence or non-existence of this subdomain, and confirm the revocation status of the RDC. This process of delivering the revocation status of the RDC to the browser is performed securely by a combination of DNSSEC and DoH, which ensure integrity and confidentiality, respectively. As long as the domain owner chooses to revoke only the RDC and not the actual TLS certificate, the RDC can be safely revoked by its own revocation mechanism without being affected by the suboptimal revocation checking of the TLS certificate.

Next, we will describe the components of RDC and how they work together within the HTTPS ecosystem.

4.2 Properties of RDC

An RDC is a credential that extends the data structure of a DC to allow revocation. The properties of RDC are as follows:

- **RDC_serial** is a unique identifier used to verify the revocation status of an RDC. The browser can launch a DNS query for a subdomain named with the `RDC_serial` to obtain the specific RDC's revocation status.
- **ValidTime** is the validity period of an RDC. Since an RDC can be revoked any time, there is no pressure for the validity period to be short. A long validity period can be set as long as it falls within the validity period of the domain owner's TLS certificate.
- **PublicKeyInfo** is a CDN's public key used to verify a `CertificateVerify` signature, which is generated by the corresponding private key. It is the same data type as the `PublicKeyInfo` defined in the X.509 standard [29].
- **Signature** is the signature over the RDC generated by the private key corresponding to the domain owner's TLS certificate. The TLS certificate's private key can be used to sign the RDC since the X.509 standard with `digitalSignature` on the `X509 v3 KeyUsage` extension permits it to sign objects other than X.509 certificates [29].

Similar to the DC, the RDC is sent to the browser along with the TLS certificate during the TLS handshake. Adding an RDC_serial to the DC introduces a minor overhead of tens of bytes. Given that a DC is roughly a few kilobytes and the TLS 1.3 standard allows up to 16 MB of data in its handshake message, incorporating an RDC_serial into the DC and sending it during the handshake poses no issue[3]. Therefore, the RDC's secure delivery is also assured by the TLS handshake protocol.

4.3 RDC revocation

The revocation status of an RDC is determined by the existence or non-existence of a subdomain created and owned by the domain owner, named after the `RDC_serial`. In other words, if <RDC_serial>.<domain name> exists, the RDC is considered revoked. On the other hand, if it does not exist, the RDC is considered valid. Therefore, the domain owner can revoke an RDC by creating the subdomain with the `RDC_serial`. During TLS authentication, the browser can verify the revocation status of an RDC by issuing a DNS query for <RDC_serial>.<domain name> to check the existence of this subdomain.

The information on the existence of the subdomain is distributed using an NSEC record [43], which is a type of DNSSEC record. An NSEC record provides the proof of whether a domain exists or not, with its integrity guaranteed by DNSSEC. By utilizing the built-in record of DNSSEC, there is no need to worry about additional deployment or incompatibility issues when handling the revocation status of an RDC. According to the DNSSEC deployment statistics maintained by ICANN (Internet Corporation for Assigned Names and Numbers) [44], as of March 2023, over 90% of all Top Level Domains (TLDs) have been signed with DNSSEC. This suggests

[3]It is worth noting that the Go TLS package supports up to 64 KB during the TLS handshake [42].

that using NSEC records to handle the revocation status of RDCs should be feasible for most domains.[4]

To ensure confidentiality between the DNS resolver and the browser, the browser should use a DNS resolver that supports DNS-over-HTTPS (or DNS-over-TLS) so that it can prevent on-path active attackers from tampering with the revocation status of RDCs. Latency between DNS-over-UDP (Do53) and DoH has been reported to be negligible in previous measurement studies [45, 46], and depending on the DNS resolver and the location of the browser, latency performance may even be superior. Users who want to securely protect the data received from the DNS resolver can choose to use a browser that supports DoH. Of note, mainstream browsers such as Firefox [47], Chrome [48], and Edge [49] currently support DoH.

4.4 Security Implication of RDC Revocation

It has been known that revocation checking in TLS certificates poses a challenge since it requires either (1) browsers to fetch revocation status of a massive number of certificates during the certificate validation process (*i.e.* CRL [29]) or (2) trusted CAs to provide reliable revocation status for all certificates (*i.e.* OCSP [50]). Because of performance and reliability issues [3, 4], abnormal revocation checking methods are being used as practical alternatives [1, 5, 35]. Essentially, if a domain owner shares his TLS certificate's private key with the CDN, he must accept the risks associated with unreliable revocation checking of the TLS certificate.

RDCs help resolve this challenge because domain owners can revoke an RDC without asking CAs to revoke the TLS certificate. By revoking only the RDC, the domain owner can safely revoke the delegated key under the security guaranteed by the RDC design, without being affected by revocation checking of the TLS certificate. Note that RDC does not solve the revocation problems of the TLS certificate, but only the revocation problems of the delegation keys. When a TLS certificate is revoked, all associated RDCs must also be revoked, making the revocation of RDCs dependent on the TLS certificate revocation checking process.

In summary, RDC does not solve the problem associated with revoking TLS certificate, but rather, they allow the domain owners to safely revoke the delegated key in case they need to discontinue the use of the CDN or detect compromise of the delegation key.

4.5 TLS Handshake using an RDC

In this section, we describe how the TLS handshake works with RDC. Figure 3 illustrates the process of a browser initiating a TLS handshake using an RDC with a web service owned by a domain owner and served by a CDN's edge server. When the browser receives the `Authentication` message containing the RDC, it sends a DoH query to the DNS resolver, inquiring about the NSEC record for the subdomain named with the `RDC_serial` to determine its revocation status. If the NSEC record is already cached in the DNS resolver, it is promptly returned to the browser without further queries. Otherwise, the DNS resolver retrieves the record from the authoritative DNS server before passing it on to the browser.

[4]DNS resolvers should return `NXDOMAIN` responses to the clients when they receive NSEC responses from the DNS authoritative servers. Thus, clients should be able to interpret the `NXDOMAIN` response as `not-revoked`.

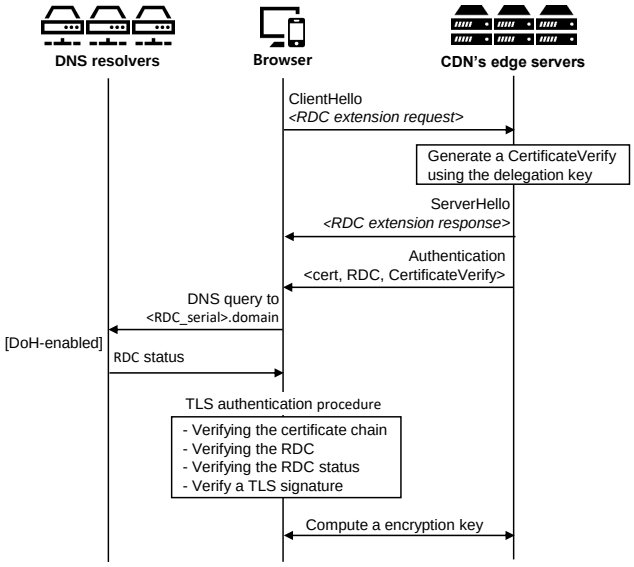

Figure 3: The sequential flow of TLS authentication using an RDC. The browser establishes a TLS handshake with a domain owner's web service that is being served by a CDN.

Upon receiving the NSEC record, the browser determines the revocation status of the RDC by verifying the existence of the subdomain associated with the `RDC_serial`. If the browser receives an NXDOMAIN response alongside the NSEC record, it interprets this as the subdomain's nonexistence, meaning the RDC has not been revoked. On the contrary, if the browser receives the NSEC record with a NOERROR response, it has to inspect the NSEC record to ascertain if it is a compact answer. This is because a compact answer, in both scenarios—whether the domain name exists or doesn't—returns an NSEC record with a NOERROR response. To determine if the response is a compact answer, the browser looks at the Next Domain Name field of the NSEC record. If the Next Domain Name field in the NSEC record starts with `\000.`, the answer is considered a Compact Answer. For such compact answers, the browser checks the Type Bit Map field in the NSEC record to decide the domain's existence. If this field contains only RRSIG, NSEC, and NXNAME (with RR type code 65281), the domain is deemed nonexistent, meaning the RDC has not been revoked. Otherwise, the domain is deemed existent, meaning the RDC has been revoked.

After determining the revocation status of the RDC, the browser follows steps similar to those with the DC. Firstly, it checks the RDC's signature to confirm that the domain owner issued the RDC. Next, it verifies the `CertificateVerify` using the RDC's public key to ensure that the CDN, acting under delegated authority, signed the TLS handshake message. Finally, the browser undergoes the traditional certificate chain validation procedure to ascertain that a trusted CA issued the TLS certificate.

5 IMPLEMENTATION

We implemented RDC into two libraries: the Go tls package (version 1.17.3) [42] to enable RDC in the HTTPS server, and the NSS (Network Security Service) [51] to enable RDC in the Firefox Nightly

browser (version 101.0a.1). As the first step in our process of implementing RDC in the TLS 1.3 protocol, we added an extension to the TLS extension type (assigned the unique identifier of the integer 10000), which serves to notify the browser's RDC support. We then extended the data structure of the TLS Certificate message to include the RDC along with the TLS certificate, allowing it to be delivered to the browser by the HTTPS server. Next, we made modifications to the TLS Certificate message, enabling the HTTPS server to use the RDC's private key instead of the TLS certificate's private key when signing the `CertificateVerify` message. We also added internal functions for the TLS authentication procedure to verify the `CertificateVerify` signature and the RDC's signature by using the RDC's public key and the TLS certificate's public key, respectively. Additional functions were also implemented to validate the revocation status of the RDC. Upon the browser receiving the RDC, the functions parse the `RDC_serial` contained in the RDC and initiate a DoH query for the subdomain named `RDC_serial` to retrieve the corresponding NSEC record. The browser detects the revocation status of the RDC by verifying the existence of the subdomain through the examination of the NSEC record.

6 EVALUATION

6.1 Experiment Setup

In order to evaluate the design goals outlined in (§3.1), we established a testbed consisting of an HTTPS web server, a client, and a DNS resolver. Our first step involved utilizing the Go tls package to develop an HTTPS web server that supports RDC. We then bought a test domain and obtained a TLS certificate of this domain from Let's Encrypt CA. After obtaining the TLS certificate, we created a key pair intended for use by the web server and generated an RDC that includes both the public key of the key pair and a signature that had been generated by the TLS certificate's private key. Subsequently, we provided the TLS certificate, the RDC, and the RDC's key pair to the web server, which we ran on an AWS t2.small instance[5]. We deployed the instance in three different geographic locations: Seoul, Virginia, and Paris

We then proceeded to the client setup and built a Firefox Nightly browser using the NSS library with RDC support on an Ubuntu 20.04 LTS desktop machine equipped with an Intel Core i9 processor clocked at 3.50 GHz and 8GB of memory. Of note, the Firefox Nightly browser was located in Seoul. We utilized Firefox's built-in Network Monitor tool [52, 53] to measure the TLS setup time and the page load time (PLT), which provided detailed information on various stages of network requests, including DNS resolution time (time taken to resolve a hostname), Connecting time (time taken to establish a TCP connection), TLS setup time (time taken to establish a TLS connection), Sending time (time taken to send the HTTP request to the server), Waiting time (waiting for a response from the server), and Receiving time (time taken to read the entire response from the server). The TLS setup time and the PLT were measured to examine how a DNS query for the NSEC record affects these values. To investigate the influence of DNS resolver caching on the TLS setup time and the PLT, we installed Bind 9 on an AWS t2.small instance located in Seoul and adjusted the max-cache-ttl, which allowed us to control the caching of the revocation status. The Firefox Nightly browser sent DoH queries to retrieve the DNS records. Lastly, we utilized Cloudflare's authoritative DNS service to distribute the revocation status of the RDC, as well as the IP address of the HTTPS server. To leverage the NSEC record for distributing the revocation status, we applied DNSSEC to our test domain.

[5]AWS t2 small instance has an Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz and 2GB RAM

6.2 Function Evaluation

Our evaluation focused on two main aspects: whether the HTTPS server can be operated using an RDC in the same manner as with a DC without the TLS certificate's private key, and whether the domain owner can revoke the RDC. First, we examined the operation of the HTTPS server utilizing an RDC without the TLS certificate's private key. The native TLS ListenAndServe function, which is necessary for running the HTTPS web server, requires both a TLS certificate and its corresponding private key.

```
func (srv *Server) \
ListenAndServeTLS("domainOwnerTLSCert", "cdnRDC", "cdnRDCKey")
```

Figure 4: ListenAndServeTLS function that integrates RDC. The function parameters do not require the TLS certificate private key; instead, only the TLS certificate of the domain, the CDN's RDC, and the RDC's private key are necessary.

However, as shown in Figure 4, the ListenAndServe function that supports RDC only necessitates a TLS certificate, an RDC, and the RDC's private key, rendering the TLS certificate's private key unnecessary. By employing our testbed, we successfully ran the RDC-supporting HTTPS web server and were able to access the test domain using the RDC-supporting Firefox Nightly browser. Through these results, we can confirm the accomplishment of design goal **G1** described in a prior section. Moreover, the successful operation of HTTPS communication within the existing infrastructure while adhering to the standard compliance confirms the fulfillment of design goal **G4**.

Next, we evaluated the revocation capability of the RDC. Revocation of the RDC can be achieved by creating a subdomain named with the `RDC_serial`. To test the revocation functionality, we utilized Cloudflare's authoritative DNS service, which served as our authoritative DNS service to create the subdomain and revoke the RDC. After creating this subdomain, we attempted to access the test domain using the Firefox Nightly browser. The browser successfully detected the RDC's revocation status and blocked us from accessing the test domain. Consequently, since the domain owner was able to autonomously revoke the RDC without revoking the TLS certificate, we can conclude that design goals **G2** and **G3** were successfully met as well.

Lastly, during the delegation process, the RDC does not impose any additional obligations on domain owners, such as running an additional server, as in Keyless SSL or DC. This allows domain owners to retain the operational benefits of using a CDN. Thus, we can affirm that design goal **G5** is achieved from an operational standpoint. A publicly accessible video demonstrating this function evaluation is available[6].

[6]https://github.com/revtls/revtls/video

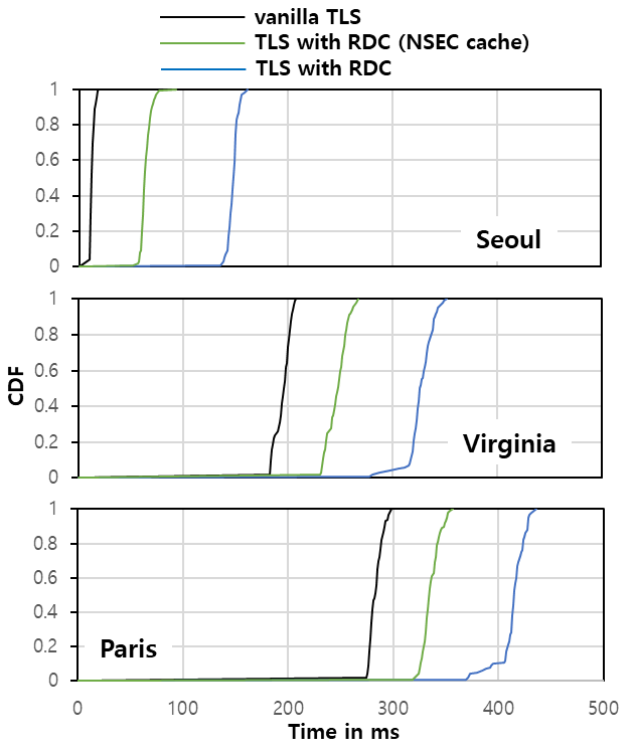

Figure 5: Comparison of the TLS setup times between vanilla TLS and TLS with RDC.

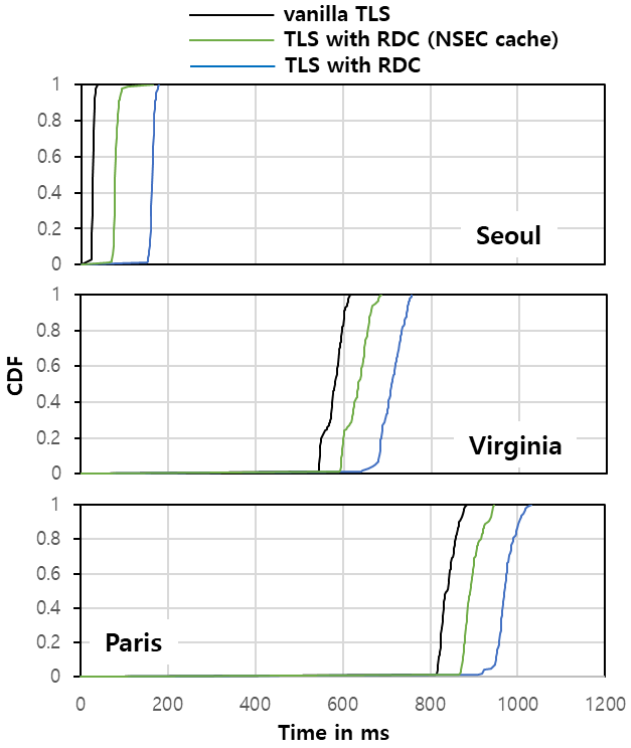

Figure 6: Comparison of the PLTs between vanilla TLS and TLS with RDC.

6.3 Performance Evaluation

In this section, we utilized Firefox's built-in Network Monitor tool [52, 53] to measure the TLS setup time and the PLT of vanilla TLS and TLS with RDC, aiming to evaluate the performance impact of using RDC during the TLS handshake.

Figure 5 presents the TLS setup time of vanilla TLS, TLS with RDC when the DNS resolver does not cache the NSEC record, and TLS with RDC when the DNS resolver caches the NSEC record. We observed that, regardless of the server's location, there is a consistent approximately 130ms difference between vanilla TLS and RDC with TLS. However, with the NSEC record cached in the DNS resolver, we observed that the difference between vanilla TLS and RDC with TLS is approximately 50ms, resulting in an overall reduction of approximately 80 ms when compared to setup times without the cached NSEC record. The difference in the TLS setup times between vanilla TLS and TLS with RDC can be attributed to the additional DoH query for the NSEC record during the TLS handshake process when RDC is used.

Moving on, we calculated the PLT by aggregating DNS resolution time, Connecting time, TLS setup time, Sending time, Waiting time, and Receiving time provided by Firefox's Network Monitor [52, 53].

Figure 6 presents the PLT of vanilla TLS, TLS with RDC without cached NSEC records, and TLS with RDC with cached NSEC records. Similar to the results of the TLS setup times, the primary difference between vanilla TLS and TLS with RDC can be attributed to the additional DoH query for the NSEC record during the TLS handshake process. Similarly, regardless of the server's location, when the NSEC record was not cached in the DNS resolver, we observed a consistent difference of approximately 130 ms between vanilla TLS and RDC with TLS. However, akin to the TLS setup time results, with the NSEC records cached in the DNS resolver, we observed an overall reduction of approximately 80 ms when compared to PLTs without cached NSEC records.

Unlike other solutions that introduce overhead for every communication to protect the TLS encryption layer, TLS with RDC focuses on protecting the TLS authentication layer, resulting in only a one-time delay during the TLS authentication procedure. Given that a one-time delay ranging from 50 ms to 130 ms is deemed acceptable for users, we can conclude that the utilization of TLS with RDC accomplishes the design goal of **G5** in terms of performance.

7 SECURITY ANALYSIS

Authenticating with a revoked RDC. An attacker who obtains an RDC's private key from a CDN, he may attempt to establish a TLS connection using this key. However, in our design, if the domain owner detects a compromise in the RDC, he can revoke the RDC by distributing its revocation status via DNS. The updated revocation status of the RDC will be delivered to browsers with a Time-to-Live (TTL) of the NSEC record to defeat the attacker's attempt. Thus, the attacker will not have enough time to carry out the attacks using the compromised private key of the RDC.

Cache poisoning of the NSEC record. An attacker may attempt to poison the NSEC record cache to mask the revocation status of

an RDC. However, an RDC's revocation status is distributed with a delay equal to the NSEC record's TTL (recommended to be 3 hours or less [54, 55]) This delay is significantly shorter than the 7-day validity period of DC and can be further reduced if domain owners operate their own authoritative DNS server to control the TTL.

Man-in-the-Middle attacks. An attacker may attempt to manipulate the revocation status of an RDC during its transmission between the authoritative DNS server and the DNS resolver. However, the use of DNSSEC signatures protects the NSEC record that indicates the RDC's revocation status, making it infeasible for the attacker to manipulate the RDC's revocation status during its delivery to the DNS resolver. As another potential form of attack, the attacker may try to tamper with the RDC's revocation status between the DNS resolver and the browser. However, DoH ensures confidentiality of the RDC's revocation status being delivered from a DNS resolver to the browser, making it infeasible for the attacker to access the revocation status.

Forging an RDC without a TLS certificate private key. RDCs are signed with the private key of the TLS certificate of the domain. During TLS authentication, the RDC signature is validated with the public key of the TLS certificate, which ensures that the RDC was issued by the correct domain owner. Moreover, the TLS certificate is validated through the certificate chain validation process that ensures that the TLS certificate was issued by a trusted CA. This process makes it impossible for an attacker to forge a valid RDC without possessing the TLS certificate's private key for the domain, which must be issued by a trusted CA.

Forging an RDC with a compromised TLS certificate private key. In the event that an attacker compromises a domain owner's TLS certificate, the attacker can create an RDC himself and use it for TLS authentication. However, using an RDC issued by the attacker for TLS authentication does not introduce any additional vulnerabilities compared to using a compromised TLS certificate. Once the domain owner detects a compromise of the TLS certificate, he must revoke the TLS certificate to invalidate both the TLS certificate and the RDC. However, when revoking the TLS certificate, it is necessary to rely on the abnormal revocation checking process. Therefore, it is crucial to securely manage TLS certificate's private keys to minimize their exposure, and thereby reduce the need for TLS certificate revocation.

Amplification attacks on DNS. An attacker can carry out an amplification attack if a DNS response is larger than its corresponding query. However, with RDC, no additional data is required to prove the RDC revocation status since the status is delivered using an NSEC record. The NSEC record is commonly used and has a relatively small size, making it unattractive for an attacker to launch an amplification attack.

8 LIMITATIONS AND DISCUSSION

Revocation checking of TLS certificates. In cases where the TLS certificate's private key is shared with a CDN, the risk of the private key's exposure will increase as the number of CDN's edge servers that share the private key increases. Given such increased risk, the domain owners are more likely to need frequent certificate revocations and encounter issues associated with these unreliable revocation mechanisms. However, if the domain owner shares the RDC instead of the TLS certificate's private key, the private key will be used only by the domain owner, and the risk of its exposure or compromise will be significantly reduced. As a result, there will be less situations that necessitate revocation of the TLS certificate, which in turn helps prevent encountering risks associated with the revocation checking of TLS certificates.

Session key protection. RDC offers a means to keep the TLS certificate's private key secure; however, an additional method is needed to protect the TLS encryption key (also known as the session key). However, previous studies [9–16, 18] have revealed that there are trade-offs that should be considered when aiming to protect the session key, including performance degradation, inability to use full functionalities of CDNs, and additional deployment. Depending on the security level needed for their web service, the domain owners can either choose to protect the TLS certificate's private key only or to protect both the private key and encryption key. Current technologies used by the CDNs primarily concentrate on protecting TLS certificate's private key. This circumstance indicates that many domain owners would opt to retain a moderate level of security (protection of their TLS certificate's private key only) and maintain the full benefits of using the CDNs, rather than to sacrifice those benefits of CDNs for the sake of a higher level of security (protection of both their TLS certificate's private key and session key).

Incremental deployment. The use of RDC requires support from the browser. As such, a CDN that only possesses the RDC's private key but not the TLS certificate's private key will not be able to perform TLS authentication with browsers that do not support the RDC mechanism. During the period of incremental deployment of RDCs when many browsers may not support the RDC mechanism, other alternative methods can be used to achieve TLS authentication using an RDC. First, the CDN determines whether the browser being used supports the RDC mechanism or not. This can be achieved by checking the inclusion of the RDC extension in the ClientHello. If the ClientHello shows that the browser does not support RDC, the CDN can forward the ClientHello to the domain owner's origin server or the Keyless SSL key server to request TLS authentication. Therefore, even in situations when the browser does not support the RDC mechanism, TLS authentication can be performed by the CDNs that only possess the RDC's private key without the TLS certificate's private key.

Managing the lifecycle of an RDC. Automating the management of the lifecycle of an RDC can help reduce the operational burden on the domain owner. To achieve this, a protocol that enables automated management of an RDC's lifecycle must be devised. This protocol would necessitate APIs between the domain owner and the DNS provider for RDC revocation, as well as APIs between the domain owner and the CDN for RDC issuance. Currently, DNS providers offer APIs for managing the domain owners' DNS records, one of which can be utilized for RDC revocation. Also, it can be reasonably assumed that CDNs will likely provide the necessary API between the domain owners and themselves to promote their business by implementing advanced security technologies. Based on these facts and assumptions, we believe that it will be possible to make a protocol for automation of RDC management, such as Automatic Certificate Management Environment (ACME) [56], which

streamlines TLS certificate management to alleviate the operational burden on domain owners in managing the RDC lifecycle.

9 RELATED WORK

TLS authentication solutions. Cloudflare introduced Keyless SSL [7] in which the domain owners maintain a key server that stores their private key, and CDNs forward requests to the key server for operations that require the private key. This approach eliminates the need for domain owners to share their private keys with the CDNs, but it does require the domain owners to run a highly available key server. At present, Keyless SSL is being used for managing customers' keys within the CDNs, which is far from its original intended purpose [40].

Cisco, Facebook, Cloudflare, and Mozilla have proposed the Delegated Credential (DC) [8], which enables domain owners to issue a credential that contains the CDN's public key and can be used for TLS authentication. The motivation behind DC is to minimize the risk of exposing the TLS certificate's private key. However, because DCs lack a method of distributing their revocation status, they must be short-lived, with a validity period of 7 days, to allow for expiration. Consequently, the domain owners who wish to continue using the DC must maintain a server capable of re-issuing the DC every 7 days before it expires. This reduces the operational benefits of using CDNs. Moreover, while the validity period of 7 days is short, it is still considered sufficient time for an attacker who has acquired a compromised private key to launch an attack. Shortening the validity period may reduce the chances of an attack but will increase the operational burden on the domain owners who will need to re-issue the DC more frequently.

DANE solutions. DNS-based Authentication of Named Entities (DANE) [57] is a protocol that leverages DNSSEC to associate a domain owner's TLS certificate with a domain name, facilitating validation of the domain owner's authenticity during TLS handshakes. In this protocol, the TLSA (TLS Authentication) record serves as a repository for storing and retrieving the domain owner's TLS certificate, enabling clients to verify its legitimacy. Liang et al. proposed an extended version of DANE [1] that expands its capabilities to allow domain owners to delegate TLS authentication to CDNs by including the CDN's TLS certificate into their TLSA record. However, this solution requires modified versions of both DANE and the certificate validation procedure. Another approach, proposed by Shihan et al., InviCloak [9], establishes an encrypted channel between the browser and the domain owner using the TLSA record's public key, ensuring the privacy of data. While InviCloak aims to minimize performance trade-offs, the domain owner needs to maintain a dedicated server for data protection and may face limitations in utilizing full CDN functionality, such as WAF.

TEE solutions. Trusted Execution Environment (TEE) is a hardware-based security technique that establishes a secure enclave, utilizing a TEE-enabled CPU to protect sensitive data. Phoenix [10], a TEE solution based on Intel SGX, enables domain owners to create enclave containers within CDNs, ensuring protection of sensitive key material, including the TLS certificate's private key, and preventing unauthorized access from the CDN. However, Phoenix incurs noticeable performance overhead, diminishing the performance advantages of using CDNs. Furthermore, SGX vulnerabilities [58–60], such as data leakage, raise concerns about the overall security of the system. Additionally, Phoenix's adoption is constrained by the requirement of an SGX-enabled CPU, which increases deployment costs and reduces incentives for CDNs to adopt the solution. Other proposed TEE solutions [11–13] encounter similar challenges due to their reliance on SGX and limited utilization of network functions, which is critical for use of CDNs.

Crypto solutions Cryptographic solutions [14, 15] are designed to enable processing of encrypted data without decrypting it, thereby eliminating the need to share the TLS certificate's private key. The downside of cryptographic solutions is that their deployment involves introducing an additional encryption scheme, which can result in performance overheads. Furthermore, utilization of encryption techniques may impose limitations on the functionality of CDNs.

TLS extension solutions Researchers also proposed to extend TLS to protect domain owners' contents from CDNs. Several TLS extension-based solutions [16–18] have been proposed to protect the domain owner's content at the TLS encryption layer, but they all require additional deployment, result in performance overhead, or limit the use of the full functionality of CDNs.

10 CONCLUSION

We propose a new design for delegation of TLS authentication called RDC, which offers a method of revoking DCs by utilizing the DNS infrastructure, an essential part of web communication. RDCs leverage the NSEC record, which allows CDNs to determine the revocation status of an RDC by checking for the existence of a subdomain named with its unique RDC_serial, while ensuring the security of DNSSEC and DoH. To evaluate its feasibility, we implemented RDC into the Go tls package and the NSS library in the Firefox Nightly browser. Through our testbed, utilizing this implementation, we successfully demonstrated that a TLS server can operate using an RDC's private key instead of the TLS certificate's private key, and that the RDC can be promptly revoked through the DNS provider. Consequently, the RDC design allows domain owners to revoke their delegation keys in scenarios such as discontinuing use of their CDN or detecting a compromise in their delegation key. Overall, our findings confirm that the RDC design effectively fulfills all the design goals outlined in (§3.1).

11 ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful feedback. This research was supported in part by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korea government [23ZR1300, Research on Intelligent Cyber Security and Trust Infra, 50%] and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [RS-2023-00215700, Trustworthy Metaverse: blockchain-enabled convergence research, 50%].

REFERENCES

[1] Jinjin Liang, Jian Jiang, Haixin Duan, Kang Li, Tao Wan, and Jianping Wu. When https meets cdn: A case of authentication in delegated service. In *2014 IEEE Symposium on Security and Privacy*, pages 67–82. IEEE, 2014.

[2] Frank Cangialosi, Taejoong Chung, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. Measurement and analysis of private key

sharing in the https ecosystem. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 628–640, 2016.
[3] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. An end-to-end measurement of certificate revocation in the web's pki. In *Proceedings of the 2015 Internet Measurement Conference*, pages 183–196, 2015.
[4] Taejoong Chung, Jay Lok, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, John Rula, Nick Sullivan, and Christo Wilson. Is the web ready for ocsp must-staple? In *Proceedings of the Internet Measurement Conference 2018*, pages 105–118, 2018.
[5] How Do Browsers Handle Revoked SSL/TLS Certificates?, 2021. https://www.ssl.com/blogs/how-do-browsers-handle-revoked-ssl-tls-certificates/.
[6] OCSP Stapling in Firefox, 2013. https://blog.mozilla.org/security/2013/07/29/ocsp-stapling-in-firefox/.
[7] Keyless SSL: The Nitty Gritty Technical Details, 2014. https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/.
[8] Richard Barnes, Subodh Iyengar, Nick Sullivan, and Eric Rescorla. Delegated Credentials for (D)TLS draft-ietf-tls-subcerts-15, 2022.
[9] Shihan Lin, Rui Xin, Aayush Goel, and Xiaowei Yang. Invicloak: An end-to-end approach to privacy and performance in web content distribution. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1947–1961, 2022.
[10] Stephen Herwig, Christina Garman, and Dave Levin. Achieving keyless {CDNs} with conclaves. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 735–751, 2020.
[11] Changzheng Wei, Jian Li, Weigang Li, Ping Yu, and Haibing Guan. Styx: a trusted and accelerated hierarchical ssl key management and distribution system for cloud based cdn application. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 201–213, 2017.
[12] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. {SafeBricks}: Shielding network functions in the cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 201–216, 2018.
[13] Rufaida Ahmed, Zirak Zaheer, Richard Li, and Robert Ricci. Harpocrates: Giving out your secrets and keeping them too. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 103–114. IEEE, 2018.
[14] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *Proceedings of the 2015 ACM conference on special interest group on data communication*, pages 213–226, 2015.
[15] Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. Embark: Securely outsourcing middleboxes to the cloud. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 255–273, 2016.
[16] Hyunwoo Lee, Zach Smith, Junghwan Lim, Gyeongjae Choi, Selin Chun, Taejoong Chung, and Ted Taekyoung Kwon. matls: How to make tls middlebox-aware? In *NDSS*, 2019.
[17] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. And then there were more: Secure communication for more than two parties. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 88–100, 2017.
[18] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-context tls (mctls) enabling secure in-network functionality in tls. *ACM SIGCOMM Computer Communication Review*, 45(4):199–212, 2015.
[19] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019, February 2019.
[20] CDN Usage Distribution in the Top 100k Sites, 2023. https://trends.builtwith.com/cdn/traffic/Top-100k.
[21] RFC 2535. Domain Name System Security Extensions, 1999.
[22] charter-ietf-doh-01. DNS Over HTTPS), 2019.
[23] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. Measuring {HTTPS} adoption on the web. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1323–1338, 2017.
[24] Percentage of Web Pages Loaded by Firefox Using HTTPS, 2023. https://letsencrypt.org/stats/#percent-pageloads.
[25] Google. Google Transparency Report: HTTPS encryption on the web. https://transparencyreport.google.com/https/overview?hl=en.
[26] Chrome Root Program Policy, Version 1.4, 2023. https://www.chromium.org/Home/chromium-security/root-ca-policy/.
[27] Mozila Common CA Database (CCADB), 2023. https://ccadb-public.secure.force.com/mozilla/CACertificatesInFirefoxReport.
[28] Apple Root Certificate Program, 2022. https://www.apple.com/certificateauthority/ca_program.html.
[29] RFC 5280. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008.
[30] RFC 8446. The Transport Layer Security (TLS) Protocol Version 1.3, 2018.
[31] Disable Universal SSL certificates. https://developers.cloudflare.com/ssl/edge-certificates/universal-ssl/disable-universal-ssl/.
[32] Usage statistics and market share of Cloudflare. https://w3techs.com/technologies/details/cn-cloudflare.
[33] CDN Industry: Trends, Size, And Market Share. https://blog.intricately.com/cdn-industry-trends-market-share-customer-size.
[34] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by Internet-wide scanning. In *22nd ACM Conference on Computer and Communications Security*, October 2015.
[35] Evaluation Dashboard: OCSP failure rate, 2022. https://telemetry.mozilla.org/new-pipeline/dist.html.
[36] Remote code execution in cdnjs of Cloudflare, 2021. https://blog.ryotak.net/post/cdnjs-remote-code-execution-en/.
[37] Matthew Prince, Daniel Stinson-Diess, Sourov Zaman. The mechanics of a sophisticated phishing scam and how we stopped it, 2022. https://blog.cloudflare.com/2022-07-sms-phishing-attacks/.
[38] Kunal Anand. Security Incident Update, 2019. https://www.imperva.com/blog/ceoblog/.
[39] John Graham-Cumming. Incident report on memory leak caused by Cloudflare parser bug, 2017. https://blog.cloudflare.com/incident-report-on-memory-leak-caused-by-cloudflare-parser-bug/.
[40] Geo Key Manager: How It Works, 2017. https://blog.cloudflare.com/geo-key-manager-how-it-works/.
[41] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.
[42] Go tls package. https://pkg.go.dev/crypto/tls.
[43] RFC 3845. DNS Security (DNSSEC) NextSECure (NSEC) RDATA Format, 2004.
[44] ICANN. M7: DNSSEC Deployment, 2023. https://ithi.research.icann.org/graph-m7.html.
[45] Rishabh Chhabra, Paul Murley, Deepak Kumar, Michael Bailey, and Gang Wang. Measuring dns-over-https performance around the world. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 351–365, 2021.
[46] Austin Hounsel, Kevin Borgolte, Paul Schmitt, Jordan Holland, and Nick Feamster. Comparing the effects of dns, dot, and doh on web performance. In *Proceedings of The Web Conference 2020*, pages 562–572, 2020.
[47] Mozilla Firefox. https://www.mozilla.org/en-US/.
[48] Google Chrome. https://www.google.com/chrome/.
[49] Edge. https://www.microsoft.com/en-us/edge?form=MA13FJ&exp=e00.
[50] RFC 6960. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, 2013.
[51] Network Security Services (NSS). https://firefox-source-docs.mozilla.org/security/nss/index.html.
[52] Mozilla. Network Monitor. https://firefox-source-docs.mozilla.org/devtools-user/network_monitor/.
[53] Mozilla. Network request details. https://firefox-source-docs.mozilla.org/devtools-user/network_monitor/request_details/index.html#Timings.
[54] RFC 2308. Negative Caching of DNS Queries (DNS NCACHE), 1998.
[55] RFC 8198. Aggressive Use of DNSSEC-Validated Cache, 2017.
[56] Richard Barnes, Jacob Hoffman-Andrews, Daniel McCarney, and James Kasten. Automatic certificate management environment (acme). Technical report, 2019.
[57] RFC 6698. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA, 2012.
[58] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.
[59] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
[60] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lippi, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. Lvi: Hijacking transient execution through microarchitectural load value injection. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 54–72. IEEE, 2020.