Proxy Certificates: The Missing Link in the
Web’s Chain of Trust

L. Chuat†, A. Abdou‡, R. Sasse†, C. Sprenger†, D. Basin† and A. Perrig†

†Department of Computer Science, ETH Zürich, Switzerland
‡School of Computer Science, Carleton University, Canada

Abstract—The ability to quickly revoke a compromised key is critical to the security of a public-key infrastructure. Regrettably, most certificate revocation schemes suffer from latency, availability, or privacy issues. The problem is exacerbated by the lack of a native delegation mechanism in TLS, which increasingly leads domain owners to engage in dangerous practices such as sharing their private keys with third parties. We investigate the utility of “proxy certificates” to address long-standing revocation and delegation shortcomings in the web PKI. By issuing proxy certificates, entities holding a regular (non-CA) certificate can grant all or a subset of their privileges to other entities. This fine-grained control on delegating privileges requires no further actions from a CA, yet does not require trust on first use (TOFU). The lifetime of a proxy certificate can be made almost arbitrarily short to curb the consequences of a key compromise. We analyze the benefits of this approach in comparison to alternatives, discussing various use cases and technical implications. We also show that combining short-lived proxy certificates with other schemes constitutes an attractive solution to several pressing problems. Overall, we make the case that the benefits obtained from incorporating proxy certificates into the current PKI substantially outweigh the changes required in practice. Such changes are minimal, and would only be required on the browser end, should a domain owner opt to use proxy certificates.

Index Terms—public-key infrastructure, certificate revocation, delegation, certificate transparency

1. Introduction

Certificate revocation remains a challenge in the HTTPS public-key infrastructure (or web PKI, for short). Certificate revocation lists (CRLs) [1], [2] grow linearly in the number of revocations, making their communication to browsers inefficient. CRLs and OCSP [3] require an extra round-trip communication initiated by the browser to verify the certificate’s validity. This both increases the page-loading delay, and may also reveal the users’ browsing habits, compromising privacy. The extra round trip may also block the connection whenever the browser fails to receive a response [4]. Failing open (i.e., proceeding with the connection anyway) is almost equivalent to not checking if a certificate is revoked, jeopardizing security. As a result, some browser vendors have decided to disable online revocation checks (i.e., OCSP and CRL) and instead rely upon small sets of emergency revocations pushed to clients through software updates [5], [6]. OCSP stapling [7] addresses the extra-round-trip problem but, like CRLs, places an additional burden and reliance on certification authorities (CAs) as they must be frequently contacted over the lifespan of the certificate. Moreover, a stapled certificate status is typically valid for four days, meaning a rogue certificate or a compromised private key remains usable by the adversary for four days in the worst case; the consequences can thus be severe.

Short-lived certificates [8], [9] provide comparable security and efficiency benefits to OCSP stapling. Again, four days is a suggested validity period. Questions remain, however, around the feasibility of reducing this period to a few minutes to limit adversarial capabilities in case of private key compromise. In general, the tradeoff between disseminating revocation information to browsers promptly (requiring an extra round-trip, burdening CAs) and increasing the system’s efficiency (but sacrificing security) is now well established. Unfortunately, browser vendors generally prioritize efficiency at the expense of security when it comes to certificate revocation [10].

Besides revocation, new requirements, with associated problems, have emerged for the web PKI. In particular, the use of content delivery networks (CDNs) has become ubiquitous and begs for a secure delegation system [11]. In practice, rather than explicitly delegating specific rights to CDNs, domain owners often resort to some form of key sharing [12]. The delegation problem is, in fact, intimately related to that of revocation. For example, if we could rely on an efficient and secure revocation system, the negative consequences of key sharing could be less impactful as compromised keys could be invalidated as soon as misbehavior is observed.
To tackle these issues, potential solutions should satisfy the following requirements:

(R1) limit the exposure and scope of private keys;
(R2) enable domain owners to autonomously delegate all or a subset of their privileges to other entities; and
(R3) enable domain owners to autonomously decide on the validity of their own certificates.

Contemplating these requirements leads us to ask a fundamental question: Why is a domain owner required to visit a CA for every issuance/renewal/revocation, for every key update they perform to any of their subdomains? We argue domain owners should be able to issue certificates for their own (sub)domains, even if they only hold a non-CA certificate (i.e., the CA flag in the X.509 certificate is set to false). This allows domain owners to take control of their policies, enabling them even to issue certificates that live only for a few hours (possibly for high-profile transactions), and others for longer periods (for non-critical subdomains). In contrast to self-signed certificates, a CA herein continues to issue certificates to domain owners, but domain owners now have the option to extend the certificate chain themselves, as they see fit, without CA support.

An analogous concept was introduced more than a decade ago in the context of grid computing [13], and these certificates were referred to as “proxy certificates”. Entities holding non-CA certificates are able to delegate some or all of their privileges to other entities through proxy certificates. We thoroughly investigate the challenges and perks of doing so when proxy certificates are issued within current PKI and HTTPS practices.

This shift in the state of affairs begets several security and efficiency advantages. These include a domain owner’s ability to update and use multiple distinct keys (e.g., for multiple subdomains, or for different transactions per subdomain) much easier and faster than current practices (Requirement R2), and issue short-lived certificates for added security (Requirement R3). Perhaps more importantly, the owner’s main private key, namely the one corresponding to the public key in the CA-issued certificate, need not be involved in online cryptographic operations, and could thus be safely stored in a disconnected (air-gapped) device (Requirement R1).

In summary, the main contributions of our work are:

• We analyze the usage of short-lived proxy certificates as a solution for the long-standing delegation and revocation problems that plague today’s web PKI.
• We present and analyze three concrete use cases for such certificates.
• We provide a detailed analysis of proxy certificates and compare it to related models. We also show how the standard certificate chain validation algorithm can be modified to support proxy certificates. In Section 5, we present three concrete use cases for proxy certificates; content delivery networks, private/separated subdomains, and dynamic policies. These example cases highlight the substantial advantages that proxy certificates bring to the web PKI. In Section 6, we analyze the consequences of using short-lived certificates and investigate how TLS should handle these certificates. In Section 7, we compare related schemes and show how proxy certificates fit into the bigger picture as they complement existing schemes. In Section 8, we discuss related work, and in Section 9 we draw conclusions.

2. Background: Dealing with Key Compromise

We start by reviewing different techniques developed over the years to prevent, detect, and remedy key compromise.

2.1. Revocation

Certificate revocation is a notoriously challenging aspect of the web PKI [10]. In recent years, researchers have been concerned with the question of how revocations should be delivered to clients, with objectives such as efficiency, deployability, and privacy in mind. For this reason, we classify revocation schemes based on the delivery process. Specifically, we distinguish four types of revocation schemes: client-driven, where the browser establishes an out-of-band connection with a revocation provider; server-driven, where the TLS server attaches a recent revocation status to the connection; vendor-driven, where browser vendors push revocations to clients through software updates; and middlebox-driven, where networking devices between the client and the server deliver revocations.

Category I: client-driven. A traditional certificate revocation list (CRL) [1] simply contains a set of revocations typically established by a CA. The distribution point of such a list can be specified within each certificate. A client may fetch the entire list (which can grow quite large) from that distribution point when establishing a TLS connection. OCSP [3] improves upon that design by letting clients contact a special responder to determine the status of any specific certificate. Both approaches suffer from latency or privacy drawbacks.

Category II: server-driven. With OCSP stapling [7],
the web server can add a timestamped OCSP response (signed by the corresponding CA) to the TLS handshake. Unfortunately, OCSP stapling is ineffective unless the browser knows when to expect a stapled response, as an attacker could just not include any OCSP status when using a revoked certificate. The must-staple extension [14] addresses this issue but has yet to gain widespread support. PKISN [15] tackles another problem: collateral damage resulting from the revocation of the certificate of a large CA, i.e., the sudden invalidation of all certificates previously signed by this CA, which should be unaffected. Their solution requires all certificates to be timestamped by a verifiable log server, whereby CA certificate revocations can be performed effectively from a specific point in time, not affecting previously issued certificates. Although the authors [15] note that PKISN could also be deployed with a vendor-driven model, we will consider its main deployment model as one where servers deliver the revocation status to browsers with an OCSP-stapling-like mechanism.

Category III: vendor-driven. We identified three main examples of vendor-driven revocation scheme. CRLSets [5] and OneCRL [6] are, respectively, Google’s and Mozilla’s effort to push a verifiable set of critical revocations to their web browsers. CRLite [16], based on the same approach of disseminating revocations through software updates, uses a cascade of Bloom filters to efficiently represent a much larger number of revocations.

Category IV: middlebox-driven. This category of schemes follows the observation that the communication and storage burden incurred by revocations can be carried by a single device for multiple hosts. RevCast [17] propagates revocations over FM radio using the RDS protocol. An RDS-to-LAN bridge then delivers revocations to end hosts. In a similar vein, RITM [18] relies on a network device on the client–server path to deliver revocations to end hosts by analyzing and appending relevant information to TLS handshakes.

2.2. Delegation

As we describe in Section 3, in most deployment models, hosting providers (such as CDNs) need browser-accepted certificates for the domains they serve. This often leads to key sharing, which can take various forms [11], [12]. In its simplest form, key sharing consists of having the domain owner directly upload its private key(s) to the hosting provider (through a web interface, for example). Alternatively, the hosting provider may use a certificate with a subject alternative name (SAN) list containing domain names from numerous distinct customers. Such certificates raise a number of questions. Cangialosi et al. [12] refer to these as “cruise-liner” certificates, and ask:

“Who on a cruise-liner certificate deserves access to the certificate’s corresponding private key, given that whoever has it can impersonate all others on the certificate? Who among them has the right to revoke the certificate, if so doing potentially renders invalid a certificate the others rely on? Cruise-liner certificates are not covered explicitly by X.509, but we can infer that, in all likelihood, only the hosting provider has the private keys and right to revoke.”

Because X.509 certificates do not natively support explicit delegation, researchers and practitioners have proposed different approaches that would allow domain owners to use proxies (CDNs, in particular) without any key sharing. Keyless SSL [19] (developed by Cloudflare) splits the TLS handshake so that most of the connection establishment is handled by edge servers, while operations requiring the domain’s private key are delegated to a key server maintained by the domain owner. Keyless SSL is compatible with both RSA and Diffie-Hellman handshakes. In RSA mode, the key server decrypts the premaster secret (generated and encrypted by the browser using the domain’s public key) and sends it back to the edge server over an encrypted channel. In the Diffie-Hellman case, the edge server sends a hash of parameters and nonces to the key server, which the key server signs and returns. The protocol was analyzed in the cryptographic model [20] where new attacks show that Keyless SSL, as specified for TLS 1.2, does not meet its intended security goals. Additionally, a new design for Keyless SSL for both TLS 1.2 and TLS 1.3 is given, together with a proof of security. In their design, session resumption is forbidden except in special cases.

SSL splitting [21] is an older but similar technique with the additional guarantee that data served by the proxy server is endorsed by the origin server. This is achieved by requiring the origin server to compute message authentication codes: for each record, the origin server sends the MAC and a short unique identifier that the proxy server uses to look up the corresponding payload in its local cache. Unfortunately, this approach limits the benefits of using a CDN as it increases latency (even more so than Keyless SSL, which only affects the initial handshake).

Liang et al. [11] proposed a solution, based on DANE [22], that makes the delegation to a CDN explicit through the name resolution process. The domain owner must add a special TLSA record containing both its own certificate and the certificate of the CDN to its DNSSEC records. This approach requires that a modified version of DANE be deployed, and that a browser extension be installed. It also increases page-load delay, as it requires an extra round trip during the TLS handshake.

2.3. Related Certificate Features

Revocation and delegation, as we will show, are two issues that can be addressed together. Instead of relying on ad-hoc schemes to satisfy the requirements stated in the introduction section, these goals can be met using
existing X.509 features, but not without major drawbacks.

The name constraints extension [1] allows CAs to issue CA certificates with a limited scope. The constraint is specified as a fully qualified domain name and may or may not specify a host. For example, as indicated in RFC 5280, both “host.example.com” and “my.host.example.com” would satisfy the “example.com” constraint, but “example.com” would not. Unfortunately, this mechanism suffers from limited support and implementation issues. Liang et al. [11] report that major browsers do not check name constraints, and even when they do, an improper enforcement of these constraints can allow a dishonest intermediate CA (normally restricted by a name constraint) to issue certificates for arbitrary domain names.

Short-lived certificates [9] reduce the attack window after a key compromise but does not allow domain owners to delegate their rights to a third party. Moreover, they require CA support, thus placing an extra burden on those CAs; the burden increases with shorter expiry windows, limiting how short a certificates’ validity period could be. Self-signed certificates, in contrast, allow domain owners to both delegate and select their attack window, but require trust on first use (TOFU) or an authenticated, out-of-band transmission of the self-signed root certificate.

2.4. Certificate Transparency

The Certificate Transparency (CT) framework [23] was developed by Google in response to several cases of CA compromise that resulted in the issuance of illegitimate certificates for high-profile domains, including *.google.com [24]. The objective of CT is to eventually record all browser-accepted certificates. To do so, CT relies on append-only log servers (many of which are currently maintained by Google) that anyone can consult. With the aim of making the logging process verifiable, certificates are incorporated into a Merkle hash tree, which allows log servers to produce efficient proofs of presence and consistency. When a certificate is submitted to a log server—provided that the certificate is valid and rooted to an accepted trust anchor—the log will reply with a Signed Certificate Timestamp (SCT). This constitutes a promise that the certificate was or will be incorporated into the hash tree within a predefined period. Chrome (since version 68) requires that SCTs be provided with all certificates issued after April 30, 2018 [25].

Certificate Transparency is relevant to our discussion for several reasons: CT allows detecting illegitimate actions from CAs and is thus a precious source of information for taking revocation decisions. As an integral part of the current HTTPS ecosystem, CT must be accounted for when proposing a new scheme, for compatibility and efficiency reasons. Finally, as an extensive source of certificates, CT is particularly helpful for understanding today’s PKI.

3. Background: Content Delivery Networks

A content delivery network, or CDN, is composed of servers and data centers who distribute content to users based on the users’ location. The objective is usually to increase performance and availability, but CDN vendors may also support security-related services.

The caching servers controlled by the CDN, commonly referred to as edge servers, act as intermediaries between clients and the origin server (controlled by the domain owner). This communication model is incompatible with TLS, which was specifically designed to prevent man-in-the-middle (MitM) attacks, since it may force domain owners to share their private key with the CDN operator who then acts as a MitM.

Multiple options exist for redirecting HTTP requests to CDN servers. The most common techniques include:

- **Authoritative**: The CDN’s name servers are defined as authoritative for the domain in question. This lets the CDN take full control over the resolution of the entire domain. When trying to contact example.com, the browser should obtain the IP address of one of the CDN’s edge servers. As the redirection happens through DNS, the browser will attempt to establish a connection based on a valid certificate for example.com; therefore, the edge server must know the corresponding private key.

- **CNAME**: The redirection is done through a DNS Canonical Name (CNAME) record, which allows for a more fine-grained mapping as it supports the redirection of specific subdomains. For example, such a record could specify the following mapping:

  \[ s1.exmpl.com \rightarrow s1.exmpl.com.cdn.net. \]

As in the previous case, if the browser tries to contact s1.exmpl.com, then it will expect to see a valid certificate for it.

- **URL rewriting**: The URLs of specific resources (e.g., images, videos, documents) are modified (either automatically by the web server, or manually by the domain owner) so that they point to CDN servers. This is the most fine-grained approach but it has drawbacks: URL rewriting does not support common security features that the CDN may offer, such as DDoS protection and web application firewalls. Proxy certificates will play a less significant role in this scenario, as the browser can accept the CDN’s own certificate.

4. Proxy Certificates for the Web

The concept of proxy certificates was first proposed in the early 2000s in the context of grid computing [26],
than it is today. Yet exist, and the use of HTTPS was far less common just been created [27], Certificate Transparency did not yet exist, and the use of HTTPS was far less common than it is today.

A proxy certificate is signed using the domain owner’s regular private key. It informs clients that the holder of the proxy certificate may legitimately serve content for the specified domain name. The term “proxy” must be interpreted here as a “delegate” or “agent” (but not necessarily an “intermediary”) to which the domain owner has conferred certain rights. In this paper, proxies are HTTPS servers, under the control of either the domain owner or a third party such as a CDN or hosting provider.

4.1. Trust model and benefits

Figure 1 illustrates the trust model of proxy certificates alongside other related models. In a traditional PKI, the “chain of trust” is typically only three certificates long, and certificates are long-lived (on the order of months, if not years). With name constraints, a link may be added to the chain in the form of a restricted CA certificate. Theoretically, a CA certificate with the right constraints could be issued to a domain owner, but because browsers do not properly support name constraints at the moment, CAs do not offer this option. Also illustrated in Figure 1 are classic short-lived certificates. Previous work suggested that short-lived certificates should be issued by CAs [9]. Instead, the model we propose is that of short-lived proxy certificates, as they offer the best of both worlds: domain owners can impose their own policies and define the maximum attack window. Better still, issuing proxy certificates does not require any form of CA support. As opposed to name constraints, proxy certificates are fail-safe: if the browser does not support proxy certificates, it will reject the connection, whereas ignoring name constraints may lead to accepting a rogue certificate. Proxy certificates allow domain owners to issue their own certificates, as self-signed certificates do, but do not require TOFU as the certificate chain starts from a CA certificate. With these properties, short-lived proxy certificates satisfy our Requirements R2 and R3.

As we describe in detail in Section 5, several use cases exist where proxy certificates are beneficial. One of our overarching principles guiding their use is that the scope and exposure of private keys should be limited (Requirement R1). User-facing web servers, in particular, should only hold private keys with limited capabilities, because they are especially exposed to various threats. This has been demonstrated by recent vulnerabilities, such as Heartbleed [28], which allow attackers to remotely read protected memory on the vulnerable machine and thus extract private keys.

Another important aspect to consider is the economic consequences proxy certificates would have on the current ecosystem. They could reduce the number of requests to CAs (as having a separate certificate for each subdomain would not be necessary anymore, for example) and thus reduce their revenue. Although this is not in their interest, CAs will not be able to hinder the deployment of proxy certificates, as they need not be involved in the issuance process. Reducing CA-incurred costs is beneficial for the deployment of HTTPS, as demonstrated by the success of Let’s Encrypt, for example.

In the remainder of this paper, we use the term domain certificate to refer exclusively to a certificate issued by a CA, as opposed to proxy certificate, which refers to a certificate that extends the chain of trust starting from a domain certificate. We use the term domain key to refer to the private key that corresponds to a domain certificate. The domain key gives its holder authority over the entire domain, while proxy certificates may be restricted.

4.2. Certification Path Validation

In the model we propose, a proxy certificate is similar to any other X.509 certificate with regard to its format; the distinction from a regular certificate comes from the certification path. In a nutshell, a proxy certificate is considered valid for a given name if it extends the chain from a non-CA certificate that is valid for that name and the proxy certificate also contains that name in its (possibly reduced) set of permitted names. However, we also allow for chains of proxy certificates and the restriction thereof through “path length” constraints, which—among other factors—makes the validation of proxy certificates non-trivial. We describe our algorithm below. It was designed to require only minor changes to the current X.509 specification and implementations.

An algorithm for the validation of a regular certification path is given in RFC 5280 [1, Section 6]. Trust anchor information (typically in the form of self-signed certificates) must be provided as input to the algorithm. A prospective certification path of length n (which does not contain the trust anchor) is also provided as input, along with other information such as the current date/time and policies. The algorithm then iterates over the n certificates and verifies that they satisfy a number of conditions. We extend that algorithm as follows:

1) Split the certification path into a regular path (stopping at the first non-CA certificate, i.e., the domain certificate) and a proxy path (consisting of zero or more proxy certificates).
2) Run the algorithm exactly as specified in RFC 5280 on the regular path. If the algorithm indicates a failure, then stop and return the failure indication; otherwise, continue with the next step.
3) If the proxy path is null, then return a success indication; otherwise, run the algorithm of RFC 5280 on the proxy path as follows:

- Provide the domain certificate as the “trust anchor information” input to the algorithm.
- Ignore the “CA” boolean(s) in the “basic constraints” of the proxy certificate(s).
- Consider “subject alternative name” values in both the domain certificate and subsequent proxy certificates as additional name constraints. That is, in each iteration $i$ of the algorithm, update the “permitted subtrees” state variable, which defines a set of names “within which all subject names in subsequent certificates in the certification path MUST fall.” [1], as follows:

$$\text{PST}_i = \text{PST}_{i-1} \cap \text{NC}_i \cap (\text{SAN}_i \cup \text{CN}_i),$$

where $\text{PST}_{i-1}$ is the previous value of the “permitted subtrees” variable, $\text{NC}_i$ is the set of acceptable names defined by the “name constraints” extension, $\text{SAN}_i$ is the set of names indicated in the “subject alternative names” extension, and $\text{CN}_i$ is the “common name” field of the $i$-th certificate. Initially, $\text{PST}_0 = \text{SAN}_0 \cup \text{CN}_0$, where $\text{SAN}_0$ and $\text{CN}_0$ are defined in the domain certificate.

Note that the path length is interpreted as is done in RFC 5280 for the proxy path as well, restarting the count. The above validation algorithm then guarantees that the following properties are satisfied:

- The set of acceptable names cannot be extended by an additional proxy certificate down the certification path. However, anyone holding a proxy certificate and the corresponding private key may sign another proxy certificate with an equal or smaller set of acceptable names. For example, a CDN holding a proxy certificate for www.example.com may issue a valid proxy certificate (one level below in the chain of trust) for www.example.com or cdn1.www.example.com, but not for admin.example.com.
- Since every certificate in the validation path must be valid, the expiration time of a proxy certificate cannot be deferred by extending the validation path with an additional proxy certificate.
- The domain certificate must be issued by a trusted CA. It may be used as is by the domain owner, without any proxy certificate in the chain (i.e., with a null proxy path).

RFC 3820 [13] provides analogous logic. However, we base our algorithm upon the more recent and complete RFC 5280 [1], which does not cover proxy certificates. In contrast to our algorithm, RFC 3820 (a) forbids usage of the “subject alternative name” extension in proxy certificates, (b) does not specify how “name constraints” should be treated, and (c) introduces an additional field for restricting the length of the proxy path, rather than utilizing the existing “path length” parameter as we do here.

### 4.3. Certificate Logging

As indicated in Figure 1, we submit that proxy certificates need not be logged (by CT servers, for example). Our rationale is as follows. First, a framework like CT was created with the objective of uncovering CA misbehavior and compromise, not attacks against individual domain owners. Second, comparing with the current situation, not logging proxy certificates does not reduce security: an attacker who compromises a private key can already impersonate the corresponding domain; issuing bogus proxy certificates for that domain would not give the attacker any more capabilities. Therefore, logging proxy certificates does not help domain owners discover breaches.
In the traditional model, short-lived certificates are issued by CAs and, except for their validity period, cannot be distinguished from regular certificates. This implies that each short-lived certificate must be logged as any other certificate, increasing the pressure on log servers. Although potential solutions have been proposed, there is no consensus on how log servers should deal with short-lived certificates [29]. Fortunately, as proxy certificates need not be logged, they do not suffer from this issue.

5. Use Cases

In this section, we describe three use cases that highlight the benefits that proxy certificates would bring to the web PKI. These use cases are non-exhaustive, and the features we present can be combined in different ways. For example, a domain owner may want to (a) use a CDN, (b) have multiple subdomains, and (c) be able to specify different policies on each subdomain.

Use Case 1: Content Delivery

The primary use case we envision for proxy certificates is content delivery, with a potentially large number of caching servers distributing content fetched from an origin server. The infrastructure needed to best take advantage of proxy certificates in this scenario depends on different factors, including whether the domain owner decides to put a focus on security, deployability, or efficiency. An entire range of possible configurations exists. At one end of the spectrum, the administrator of a static website with no sensitive data could choose to forgo using proxy certificates altogether, as our model does not make them mandatory. At the other extreme, the domain key can be stored on an air-gapped device, kept in a secure location and configured only to sign very-short-lived proxy certificates, which could then be extracted from the signing device using QR codes [30] displayed on an attached screen and read by a networked camera, for example. This might be the solution of choice for an organization with high security requirements, such as a financial institution.

Before an HTTPS connection can be established, the client must obtain the IP address of the edge server through DNS resolution. The edge server could be one of many servers under the control of a CDN, for example, or a machine that the domain owner controls directly. If the edge server is controlled by the domain owner, the DNS resolution may be straightforward (through an A or AAAA record). Redirection to a CDN server, on the other hand, can be realized in different ways, as described in Section 3.

The client then establishes an HTTPS connection with the edge server, which temporarily stores some or all of the resources fetched from an origin server, as illustrated in Figure 2. To establish a TLS connection with the client, the edge server needs a proxy certificate that it can either fetch from the certificate server, or that the certificate server can push at regular intervals. Issuing these proxy certificates requires an initial setup:

1) The edge server (or the entity controlling it) generates a key pair and puts the public key into an unsigned certificate signing request (CSR) with the same format as a standard CSR. We call this file a proxy CSR.

2) The domain owner obtains the proxy CSR through an authentic channel (e.g., the web portal of the CDN).

3) The domain owner configures the certificate server to issue proxy certificates (based on the proxy CSR) at a specified frequency and with a specified validity period. Only the “not before” and “not after” fields [1] are updated. The validity periods of two consecutive proxy certificates must overlap to avoid downtimes.

Thereafter, when needed, the certificate server creates a new proxy certificate and signs it using the domain key. This implies that the certificate server must either have direct access to the domain key or access to a device able to produce the appropriate signatures. For example, the domain key could be stored on a hardware security module (HSM). The proxy certificate is then transmitted from the certificate server to the edge server. As proxy certificates are public information, any communication protocol can be employed. The CDN could even extend the certification path further (with a second layer of proxy certificates) so that different keys are used by different edge servers.

Under normal circumstances, even if the proxy certificates are short-lived, the key pair used by the edge server can remain unchanged over a long time. Indeed, the re-issuance process only serves the purpose of extending a lease to the proxy, but does not imply a key rollover. If needed, key rollover can be achieved simply by having the edge server generate a new key pair, and use the new public key for subsequent key signing requests. The domain owner has the option of terminating that lease at any time, by simply stopping the issuance process, thus effectively revoking delegation privileges.

It is recommended, in such a context, to restrict traffic to the certificate and origin servers, especially from the public Internet. A firewall, for example, may be configured to limit incoming connections to those from the CDN’s IP space, which is generally public information [31].

Since 2016, Let’s Encrypt has been offering free certificates to domain owners, thanks in part to a completely automated domain validation process supported by the Automatic Certificate Management Environment (ACME) protocol [32]. Let’s Encrypt also offers tools for (re-)issuing and revoking certificates. Similar automation could be adopted by domain owners to handle and maintain proxy certificates, which could be implemented using a software framework like that of Let’s Encrypt.
Use Case 2: Private, Separated Subdomains

Another use case for proxy certificates is that of using different private keys on different subdomains. Wildcard certificates (albeit generally more expensive than regular ones) are appreciated by domain owners because they allow them to protect any number of subdomains they want, independently from CAs. However, a wildcard certificate normally implies that the same private key is used on all subdomains. In contrast, by using a wildcard in the domain certificate (such as *.example.com) but a more specific name (such as s1.example.com) in all proxy certificates (see Figure 1), the domain owner can make sure that the consequences of a key compromise or misbehavior from a hosting provider are confined to the corresponding subdomain.

Moreover, as proxy certificates need not be included in certificate logs such as CTs (see Section 4.3 above for motivation), they would not disclose private subdomain names such as secret-project.example.com, which is an inherent and undesirable consequence of Certificate Transparency [33]. Scheitle et al. [34] even showed that CT logs are being actively monitored to find new domain names as targets.

Use Case 3: Dynamic Security Policies

One of the main benefits of proxy certificates is that they lend themselves to short validity periods, but this is only part of the story. The more general advantage of proxy certificates is that they allow domain owners to dynamically define security policies and separately for each subdomain: every time a proxy certificate is created or renewed, the domain owner is free to define a new policy or change an existing one using the appropriate fields in the proxy certificate itself. We do not attempt to exhaustively list all such policies here, but discuss a few salient examples. Table 1 shows five X.509 fields (three that are part of the current standard and two that we propose) through which domain owners can control the validity of proxy certificates for their domain. The notBefore and notAfter fields define the validity period of the certificate, while pathLenConstraint restricts the length of the proxy path.

As we have seen, the assumption that the domain name and servers are controlled by the same entity no longer holds. A consequence of this is that a domain owner may not approve of the server’s configuration. For example, a CDN (or attacker having compromised the domain’s private key) may try to use session resumption to extend a TLS session far beyond the certificate’s expiration date (see Section 6). With a dedicated certificate extension field in the proxy certificate (resumptionAllowed), a security-focused domain owner could indicate to the browser that session resumption should not be attempted. The browser may be configured to strengthen certain domain policies, but should not weaken them.

Another example of a policy relates to how browsers deal with protocol errors. As pointed out in previous research [35], domain owners should be able to influence the decision of the browser to either completely stop a TLS communication in case an anomaly occurs, a hard fail, or give the user an option to proceed, knowing that there is the risk that the channel’s security may be compromised. Indeed, the domain owner is the entity best able to specify the level of security and availability that their application requires and determine whether the browser should hard fail or soft fail. We suggest that domain owners should be able to express their preference for a failure mode through an extension in the proxy certificates they issue (failureMode).

6. TLS with Short-Lived Certificates

TLS was not designed with short-lived certificates in mind. In this section, we explain why it is critical that the certificate lifetime be considered even after a connection is established. We stress that this issue does not only concern our proposal. Our discussion concerns any TLS session that is sufficiently long-lived to extend beyond the certificate’s expiration time. This situation is not as unlikely as it may initially appear when we consider session resumption, but has unfortunately often been neglected in previous work on short-lived certificates [9].

6.1. TLS sessions and resumption

We focus on TLS 1.3 [36] and use its nomenclature, rather than that of older TLS versions. However, similar
arguments apply to older TLS versions, in particular, TLS 1.2.

TLS describes a key-exchange protocol between two parties, resulting in shared keying material used in a subsequent connection transferring data. Such a connection usually times out after not being used between 1 to 5 minutes, but can otherwise stay alive indefinitely. This connection is part of a session that may (and in practice does) provide a pre-shared key (PSK) which is single-use and provisioned for session resumption, which creates a new connection.

Based on a PSK, a client can reconnect to the server while the PSK stays valid, without the certificate being checked again. Note that a PSK stays valid for up to 7 days, at the issuing server’s discretion. Using PSK-based resumption (usually) provisions a new PSK, again valid for 7 days from its issuance, allowing indefinite chaining [36, Section 4.6.1].

6.2. The problem with resumption

At no point after the initial full key-exchange is the certificate, or its lifetime, considered. The TLS 1.3 specification suggests that the validity of connections and sessions should consider certificate validity, but does not mandate it [36, at the end of Section 4.6.1].

Using session resumption can therefore squander the security benefits of short-lived certificates. Current web browsers do not check that a resumption is performed within the certificates’ lifetime. A malicious edge server can issue a session PSK with a long lifetime (maximum of 7 days) to clients and prolong them on every subsequent connection, thereby subverting the benefit of short-lived certificates.

6.3. Possible solutions using proxy certificates

Currently, the only secure solution to this problem is to disallow session resumption completely, thereby forsaking its efficiency benefits. For non-security-sensitive web sites, one can opt for more efficiency and use longer-lived certificates together with session resumption. Unfortunately, current browsers do not allow users to configure the use of session resumption. However, some privacy-aware browsers such as Tor or the JonDoBrowser disable session resumption globally in order to prevent user tracking [37]. We propose a more flexible approach based on dynamic policies (as explained in Use Case 3 above, Section 5) where a domain can determine a (on-off) session resumption policy, which is then enforced by the browser.

A further improvement that combines security and efficiency would require a coordination between the lifetimes of the short-lived certificates and the session PSKs, ensuring that a PSK cannot be used after the expiration of the edge server’s (or subdomain’s) certificate. For the certificate obtained from the server during the first connection of a session, this could be achieved by dropping the connection when the certificate expires. Unfortunately, however, handling the extended expiration time of subsequently issued short-lived certificates appears hard to achieve, since it additionally mandates a timely update of these certificates in the browser. Such an update could be achieved by either (a) establishing a fresh session, which involves a full TLS handshake, or (b) updating the certificate in the web browser by an out-of-band communication with the domain. The former solution means that the use of the session PSK is limited by the lifetime of a single short-lived certificate, which makes resumption useless with minute-range certificate lifetimes. The latter would effectively enable the use of session resumption as long as the browser possesses a valid certificate for the domain. However, implementing this solution would likely involve considerable modifications to current infrastructure.

Using proxy certificates, a domain owner can specify its own session resumption policies, which are then enforced by the browser. This allows a fine-grained split, for example by subdomain, where security-critical subdomains (e.g., a login page) disallows resumption, while other pages allow it. In conclusion, as proxy certificates provide domain owners fine-grained control over their policies, they achieve the efficiency benefits of quick session resumption without losing the security benefits across all subdomains.

7. Analysis

To assess the advantages of proxy certificates for the web PKI, we analyze them along with various revocation and delegation schemes proposed in the literature with respect to a wide range of properties. Our results are summarized in Table 2. We focus on the schemes introduced in Section 2 (table rows), which appear the most relevant and representative to us. However, this list is by no means exhaustive. We formulate the properties in terms of benefits (table columns) that the different schemes may provide and we categorize these benefits into six classes: revocation-related, delegated-related, security, efficiency, deployability, and cross-category benefits. We consider 17 individual benefits in total, four of which cover the requirement we have set out in the introduction section. We classify each scheme as to whether it provides, partially provides, or does not provide each of these benefits. This way we can clearly show which improvements our scheme provides in comparison to the literature.

At the bottom of the table we also list various combinations of schemes that add benefits. Short-lived proxy certificates (our baseline combination) offers already most of the benefits. By combining these certificates with different revocation schemes, we can achieve additional benefits.
**Evaluation Criteria**

A. Revocation-Related Benefits.

Supports CA revocation: A revocation scheme should give authorized entities (such as CAs or software vendors) the ability to invalidate the certificates of root and intermediate CAs. We give partial points to schemes that theoretically support revoking CAs but require contacting those CAs to do so.

Supports damage-free CA revocation: Revoking a CA certificate should not cause collateral damage, i.e., it should not invalidate certificates issued by the revoked CA before it was compromised. Standard revocation invalidates all previously issued certificates by a revoked CA, thus leading to massive collateral damage.

Supports low-profile certificate revocation (R3): All certificates can be revoked. Because the set of all revocations can be large, certain schemes (such as CRLSets and OneCRL) only use a manually selected subset of revocations deemed critical. The benefit is offered by schemes that let domain owners revoke their certificates as they see fit and always provide those revocations to clients. This benefit corresponds to Requirement R3. Short-lived certificates offer this benefit, though strictly speaking they do not support revocation; they rather allow domain owners to rapidly make a compromised key unusable for the attacker, which is equivalent. In contrast, self-signed certificates do not offer this benefit as their revocation requires additional mechanisms.

B. Delegation-Related Benefits.

Delegation without key sharing (R1): We grant this benefit to schemes that let domain owners use third-party hosting/proxying without sharing any private key. This clearly limits the exposure of private keys and therefore covers Requirement R1.

Domain-controlled delegation (R1, R2): Schemes offering this benefit enable the domain owner to establish and revoke delegations at her own discretion, without the help of a third-party such as a CA, and limit their scope (e.g., to a particular subdomain). This benefit covers the control aspect of Requirement R2 and limits the scope of private keys as demanded by Requirement R1.

C. Security Benefits.

Domain-owner-defined policies (R2, R3): This benefit is granted to schemes that let domain owners define policies, e.g., dictating how long an attacker could carry out a MitM attack in case the private key used by the HTTPS server was compromised, or specifying whether session resumption is authorized or not (see Use Case 3 in Section 5). This benefit covers the policy aspects of Requirements R2 and R3.

No trust-on-first-use required: This benefit is granted to schemes that do not require trusting the domain’s certificate the first time it is encountered. We give partial points to self-signed certificates as they can alternatively be obtained through an authentic channel.

Preserves user privacy: Browsers should not have to contact any third party when validating a certificate or delegation as it may compromise user privacy.

D. Efficiency Benefits.

Does not increase page-load delay: We grant this benefit to schemes that do not substantially increase the time it takes for the browser to load pages. Specifically, we ignore small processing delays, but do not grant this benefit to schemes that incur additional network delay.

Low burden on CAs: Experience has shown that if a scheme requires CAs to make considerable operational efforts, but provides limited financial benefits, then the scheme is unlikely to be successful. In particular, CAs should not have to handle high numbers of requests from clients (e.g., to check a revocation status) or be required to re-issue certificates at a high frequency.

Reasonable logging overhead: The scheme should not put excessive pressure on certificate logs. Logging certificates with a short validity period or issued by the domain owners themselves (hence, potentially too many), in particular, would make logs blow up.

E. Deployability Benefits.

Non-proprietary: A scheme is more generally useful if it is not bound to or controlled by a particular software vendor. We grant this benefit to open schemes, where no entity is responsible for deciding which revocations should be considered.

No special hardware required: This holds for schemes without special hardware requirements. We grant partial points to schemes that do not require a special kind of hardware but require that hardware (e.g., middleboxes) be updated to support the scheme.

CA compatible: We grant this benefit to schemes that can work without the participation of CAs. We grant partial points to schemes that need CA participation but are widely supported.

Server compatible: This benefit means that no changes are required on the server side. OCSP stapling partially offers the benefit as it requires actions from the web server but is widely supported. Proxy certificates also partially offer the benefit as they could be used a priori by any web server, but software updates would be necessary to let domain owners fully take advantage of proxy certificates.
Table 2. Comparison of revocation and delegation approaches. The proposal herein and scheme combinations using proxy certificates are highlighted in gray. Columns represent evaluation benefits we devised to distinguish schemes. Each benefit falls under one of six categories (A–F). Benefits corresponding to our requirements are marked with R1, R2, or R3. See Section 2 for our categorization of revocation schemes (Cat. I–IV) and Section 7 for the details of our analysis criteria.

| Scheme | Reference |
|--------|-----------|
| a. Regular CRL (Cat. I) | [1] |
| b. Hard-fail OCSP (Cat. I) | [3] |
| c. OCSP stapling (Cat. II) | [7] |
| d. PKISN (Cat. II) | [15] |
| e. CRLSets (Cat. III) | [5] |
| f. OneCRL (Cat. III) | [6] |
| g. CRLite (Cat. III) | [16] |
| h. RevCast (Cat. IV) | [17] |
| i. RITM (Cat. IV) | [18] |
| j. SSL splitting | [21] |
| k. Keyless SSL | [20] |
| l. DANE-based delegation | [11] |
| m. Self-signed certificates | [2] |
| n. Short-lived certificates | [9] |
| o. Name-constrained cert. | [38] |
| p. Cruise-liner certificates | [12] |
| q. Proxy certificates (herein) | |
| Combinations of Schemes | |
| n + q | |
| n + q + d | |
| n + q (e or f) | |
| n + q + g | |

A = Supports CA revocation; B = Supports damage-free CA revocation; C = Supports low-profile cert. revocation (R3); D = Delegation without key sharing (R1); E = Domain-controlled delegation (R1, R2); F = Domain-owner-defined policies (R2, R3). Benefits corresponding to our requirements are marked with R1, R2, or R3. See Section 2 for our categorization of revocation schemes (Cat. I–IV) and Section 7 for the details of our analysis criteria.

● = offers the benefit; ○ = partially offers the benefit; ◯ = does not offer the benefit.
**Browser compatible**: This indicates that no changes to the browser are required. We grant partial benefits to schemes that are standardized and implemented by major browser vendors but often turned off by default or improperly enforced. Self-signed certificates also received partial points for this benefit as they typically generate error messages on first use.

We purposely did not include “incrementally deployable” in our analysis as we deem it to be a vague and potentially misleading “benefit” in this context. OCSP stapling, for example, can be considered incrementally deployable in its soft-fail variant i.e., the browser does not return an error when the server does not support stapling), but this allows an attacker to simply omit the OCSP status of a revoked certificate and thus provides no added security. Incremental deployment can also be interpreted as “clients that have not been updated to support a scheme can still communicate with servers that support it (although clients will not reap the security benefits of that scheme)”, but this is captured by our “browser compatible” criterion. The flip side of deploying a security scheme that is compatible with previous browser versions is that users with outdated software are unlikely to realize that their browser does not support the latest security standards, and thus does not encourage adoption.

### F. Cross-Category Benefit.

**No out-of-band communication**: Requiring users to communicate with a third party or use a different channel is problematic for several reasons; it can compromise user privacy, increase latency, and captive portals may not allow the connection to be made. We grant this benefit to schemes that require users to neither use a separate channel nor establish a connection with a server that would not be contacted otherwise. We grant partial points to self-signed certificates for this benefit, as an out-of-band communication may be used to circumvent the TOFU problem self-signed certificates inherently have otherwise.

### Combinations of Schemes

In Table 2, we evaluate the benefits of combining proxy certificates with other schemes. In particular, when proxy certificates are short-lived (i.e., when schemes n and q are combined), revocation becomes feasible thus offering the “supports low-profile certificate revocation” benefit. In that case, revocation would be undertaken by the domain owner rather than the CA. The owner issues short-lived certificates (e.g., a few hours) and simply refrains from renewing these on time of expiry when revocation is needed.

The revocation of CA certificates is, by definition, not supported by proxy certificates and has a vastly different scale (in terms of total number of certificates) and different requirements. Therefore, we suggest that schemes that were specially designed for the purpose of revoking CAs (such as PKISN [15]) or schemes that work well with smaller numbers of certificates (such as CRLSets and OneCRL) be used in conjunction with proxy certificates.

As we could not include all possible combinations in our comparison table, we only included schemes that offer complementary benefits to those of proxy certificates with minimal drawbacks, and schemes that are already widely deployed (CRLSets, OneCRL). We see that several combinations fulfill all of our three requirements, in most cases requiring only minor software updates.

### 8. Related Work

In 2013, Clark and van Oorschot [39] surveyed problems of the TLS/HTTPS ecosystem and its trust model. They discuss a wide range of problems including TLS protocol flaws, certification, trust anchoring, delegation, revocation, and user interface issues. They evaluate various enhancements with respect to security properties in three categories (detecting MitM attacks, TLS stripping, and PKI improvements) and their general impact on security and privacy, deployability, and usability. Their remarks highlighted how CA infrastructures are increasingly being seen as a fundamental weakness in the PKI system. Our work here provides concrete steps towards effectively reducing reliance on the CA infrastructure in the HTTPS ecosystem. The authors [39] also shed light on defense-in-depth techniques against fraudulent certificates, including then-evolving certificate pinning. With support for pinning dropped after several years of operation [40], it is becoming clearer that such directions provide no hope to address both revocation and delegation challenges simultaneously.

AKI [41] and its successor ARPKI [42] are more holistic approaches to upgrading the web PKI. One of the main ideas in these proposals is that resilience to compromise can be improved by requiring that multiple CAs sign each domain certificate. Additionally, to guarantee that no other rogue certificate exists for a given domain, all certificates must be logged and log servers are used to efficiently produce both presence and absence proofs. Unfortunately, this also implies that only one certificate per domain can be valid at any given time. ARPKI’s key security properties were also formally verified. PoliCert [35] builds on top of ARPKI to solve that problem by replacing the unique domain certificate by a unique domain policy that specifies which CAs are allowed to issue (potentially multiple) certificates for that domain. However, that approach does not allow domain owners to rapidly change their policies or produce their own certificates (i.e., without contacting several CAs). Therefore, proxy certificates could complement ARPKI.
certificate chains as a lightweight and more dynamic alternative to PoliCert.

A number of previous research papers have addressed the problem of delegation. Kogan et al. [43] argue that a secure delegation system should always make explicit “who will do what to whom”, and present a design for the SSH protocol, called Guardian Agent. MacKenzie et al. [44] address the problem of server delegation in the context of capture-resilient devices (i.e., devices required to confirm password guesses with a designated remote server before private-key operations). STYX [45] is a key management scheme, based on Intel SGX, Intel QuickAssist Technology, and the SIGMA (SIGn-and-MAC) protocol, which can be used to distribute and protect SSL/TLS keys.

Sy et al. [37] recently showed, after analyzing as many as 48 browsers, that session resumption was also problematic for user privacy as it can be used to track the average user for up to eight days with standard settings. With a long session resumption lifetime, a majority of users can even be tracked permanently. Problems have also been discovered in the way CAs perform domain validation: exploiting a BGP vulnerability to hijack traffic, an attacker can obtain a rogue certificate from vulnerable CAs [46]. [47]. Brubaker et al. also found serious vulnerabilities in popular implementations of SSL/TLS using “frankencerts”, i.e., certificates with unusual combinations of extensions and constraints [48].

9. Conclusion

On the one hand, using a single long-lived certificate per domain has several advantages: it is more practical for administrators, it lowers the pressure on transparency logs, it reduces CA-incurred costs, and thus benefits the deployment of HTTPS. On the other hand, using short-lived certificates, not sharing private keys with delegated third parties, and limiting the scope and exposure of private keys is preferable from a security standpoint. We found that proxy certificates solve this conundrum with minor changes to existing standards and little overhead. They give more flexibility to domain owners and accommodate practices that have become ubiquitous on the web, such as delegation to CDNs.

Proxy certificates tackle the efficiency-security trade-offs of TLS session resumption. Since they make it feasible for domain owners to maintain as many certificates as they desire without CA reliance for these certificates, resumption policies can be customized individually for different subdomains. A domain owner could, e.g., ban resumption on security-critical subdomains—a policy that could easily be expressed in the proxy certificate and enforced by browsers.

Compatibility with current certificate logging frameworks, most notably CT, is of great importance for deployability. Despite a potential growth in certificate issuance upon adopting proxy certificates for the web, they do not pose a problem to logging frameworks as proxy certificates need not be logged (Section 4.3).

Finally, in contrast to numerous previously-proposed primitives addressing weaknesses in the HTTPS ecosystem, adopting proxy certificates for the web does not require any new dedicated infrastructures, nor any changes to the current CA infrastructure. It also requires no changes to logging frameworks, and no changes to current revocation practices (proxy certificates can be extremely short-lived, at the discretion of the issuing domain owner, thus revoked by simply not being renewed). In fact, adoption only requires browsers to implement the required changes to the X.509 specification (Section 4.2). Moreover, domain owners are free to decide whether and how they want to use proxy certificates: a website with minimal security requirements may choose not to employ proxy certificates at all, while a large company may choose to use them extensively to protect its most important private keys. Given their benefits, we advocate that proxy certificates be supported by future standards and implementations, and call upon the community to join the discussion towards that goal.

References

[1] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and T. Polk, “Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile,” RFC 5280, May 2008.
[2] P. E. Yee, “Updates to the internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile,” RFC 6818, January 2013.
[3] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, “X.509 internet public key infrastructure online certificate status protocol - OCSP,” RFC 2560, June 1999.
[4] E. Stark, L.-S. Huang, D. Israni, C. Jackson, and D. Boneh, “The case for prefetching and prevalidating TLS server certificates,” in Network and Distributed System Security Symposium (NDSS), 2012.
[5] A. Langley, “Revocation checking and Chrome’s CRLs,” 2012. [Online]. Available: https://www.imperialviolet.org/2012/02/05/crlsets.html
[6] Mozilla, “Mozilla’s revocation plan.” [Online]. Available: https://wiki.mozilla.org/CA:RevocationPlan
[7] Y. N. Pettersen, “The transport layer security (TLS) multiple certificate status request extension,” RFC 6961, June 2013.
[8] R. L. Rivest, “Can we eliminate certificate revocation lists?” in Financial Cryptography, 1998.
[9] E. Topalovic, B. Saeta, L.-S. Huang, C. Jackson, and D. Boneh, “Towards short-lived certificates,” Web 2.0 Security and Privacy, 2012.
[10] Y. Lin, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson, “An end-to-end measurement of certificate revocation in the web’s PKI,” in ACM Internet Measurement Conference (IMC), 2015.
[11] 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 IEEE Symposium on Security and Privacy (S&P), Apr. 2014, pp. 67–82.
[12] F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. Maggs, A. Mislove, and C. Wilson, “measurement and analysis of private key sharing in the HTTPS ecosystem,” in *ACM CCS*, 2016, pp. 628–640.

[13] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson, “Internet X.509 public key infrastructure (PKI) proxy certificate profile,” RFC 3820, June 2004.

[14] P. Hallam-Baker, “X.509v3 extension: OCSP stapling required,” Internet Draft, October 2012. [Online]. Available: https://tools.ietf.org/html/draft-hallambaker-muststaple-00

[15] P. Szalachowski, L. Chuat, and A. Perrig, “PKI safety net (PKISN): Addressing the too-big-to-be-revoked problem of the TLS ecosystem,” in *IEEE European Symposium on Security and Privacy (EuroS&P)*, Mar. 2016.

[16] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, “CRLite: A scalable system for pushing all TLS revocations to all browsers,” in *IEEE Symposium on Security and Privacy (S&P)*, May 2017.

[17] A. Schulman, D. Levin, and N. Spring, “RevCast: Fast, private certificate revocation over FM radio,” in *ACM CCS*, 2014, pp. 799–810.

[18] P. Szalachowski, L. Chuat, T. Lee, and A. Perrig, “RTM: Revocation in the middle,” in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, Jun. 2016.

[19] D. Stebila and N. Sullivan, “An analysis of TLS handshake proxying,” in *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20–22, 2015, Volume 1*. IEEE, 2015, pp. 279–286. [Online]. Available: https://doi.org/10.1109/Trustcom.2015.385

[20] K. Bhargavan, I. Boureanu, P.-A. Fouque, C. Onete, and B. Richard, “Content delivery over TLS: a cryptographic analysis of keyless SSL,” in *IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, Aug. 2017, pp. 1–16.

[21] C. Lesniewski-Laa and M. F. Kaashoek, “SSL splitting: Securely serving data from untrusted caches,” *Computer Networks*, vol. 48, no. 5, pp. 763–779, Aug. 2005.

[22] P. Hoffman and J. Schlyter, “The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA,” RFC 6698, August 2012.

[23] B. Laurie, A. Langley, and E. Kasper, “Certificate transparency,” RFC 6962, August 2012.

[24] J. R. Prins, “DigiNotar certificate authority breach “Operation Black Tulip”,” fox-it, Interim Report, September 2011.

[25] D. O’Brien, “Certificate Transparency Enforcement in Chrome and CT Day in London.” *Online*. Available: https://perma.cc/T3NM-SHZN

[26] V. Welch, I. Foster, C. Kesselman, O. Mulino, L. Pearlman, S. Tuecke, J. Gavor, S. Meder, and F. Siebenlist, “X.509 proxy certificates for dynamic delegation,” in *3rd annual PKI R&D workshop*, vol. 14, 2004.

[27] Akamai, “About akamai: Company history,” https://perma.cc/D3Z7-FFB9.

[28] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey *et al.*, “The matter of HeartBleed,” in *ACM Internet Measurement conference (IMC)*, 2014.

[29] D. Lopez, A. Pastor-Perales, Y. Sheffer, and T. Fossat, “Short-lived certificates and certificate transparency,” IETF 100 Singapore, November 2017.

[30] S. Matsumoto, S. Steffen, and A. Perrig, “CASTLE: CA signing in a touch-less environment,” in *Annual Computer Security Applications Conference (ACSAC)*, Dec. 2016.

[31] CloudFlare, “IP ranges.” [Online]. Available: https://www.cloudflare.com/ips/

[32] R. Barnes, J. Hoffman-Andrews, and J. Kasten, “Automatic certificate management environment (ACME),” IETF Draft, August 2018.

[33] Google Code Archive, “Need options for avoiding logging private subdomains (certificate-transparency issue #20),” October 2013. [Online]. Available: https://perma.cc/27RW-PNWF

[34] Q. Scheitle, O. Gasser, T. Nolte, J. Amann, L. Brent, G. Carle, R. Holz, T. C. Schmidt, and M. Wählisch, “The rise of Certificate Transparency and its implications on the Internet ecosystem,” in *ACM Internet Measurement Conference (IMC)*, 2018.

[35] P. Szalachowski, S. Matsumoto, and A. Perrig, “Policert: Secure and flexible tls certificate management,” in *ACM CCS*, 2014.

[36] E. Rescorla, “The transport layer security (TLS) protocol version 1.3,” RFC 8446, August 2018.

[37] E. Sy, C. Burkert, H. Federrath, and M. Fischer, “Tracking users across the web via TLS session resumption,” in *Annual Computer Security Applications Conference (ACSSAC)*, 2018.

[38] M. Cooper, Y. Dzambasow, P. Hesse, S. Joseph, and R. Nicholas, “Internet X.509 public key infrastructure: Certification path building,” RFC 4158, September 2005.

[39] J. Clark and P. C. van Oorschot, “Sok: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements,” in *IEEE Symposium on Security and Privacy (S&F)*, 2013, pp. 511–525.

[40] G. Groups, “Intent To Deprecate And Remove: Public Key Pinning,” 2017. [Online]. Available: https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/he9tr7p3rZ8

[41] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and V. Gligor, “Accountable key infrastructure (AKI): A proposal for a public-key validation infrastructure,” in *International World Wide Web Conference (WWW)*, May 2013.

[42] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski, “Design, analysis, and implementation of arpi: an attack-resilient public-key infrastructure,” *IEEE Transactions on Dependable and Secure Computing (TDSC)*, Oct. 2016.

[43] D. Kogan, H. Stern, A. Talbert, D. Mazières, and K. Winstein, “The case for secure delegation,” in *ACM Workshop on Hot Topics in Networks (HotNets)*, 2017, pp. 15–21.

[44] P. MacKenzie and M. K. Reiter, “Delegation of cryptographic servers for capture-resilient devices,” *Distributed Computing*, vol. 16, no. 4, pp. 307–327, Dec. 2003.

[45] C. Wei, J. Li, W. Li, P. Yu, and H. Guan, “STYX: a trusted and accelerated hierarchical SSL key management and distribution system for cloud based CDN application,” in *ACM Symposium on Cloud Computing*, 2017, pp. 201–213.

[46] H. Birge-Lee, Y. Sun, A. Edmundson, J. Rexpoed, and P. Mittal, “Bamboozling Certificate Authorities with BGP,” in *USENIX Security Symposium*, 2018.

[47] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, “Domain Validation++ for MitM-Resilient PKI,” in *ACM CCS*, 2018.

[48] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, “Using frankencerts for automated adversarial testing of certificate validation in ssl/tls implementations,” in *IEEE Symposium on Security and Privacy (S&F)*, 2014.