2017 IEEE Symposium on Security and Privacy

CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers

James Larisch* David Choffnes* Dave Levin†
Bruce M. Maggs‡ Alan Mislove* Christo Wilson*
* Northeastern University † University of Maryland ‡ Duke University and Akamai Technologies

***Abstract*—Currently, no major browser fully checks for TLS/SSL certificate revocations. This is largely due to the fact that the deployed mechanisms for disseminating revocations (CRLs, OCSP, OCSP Stapling, CRLSet, and OneCRL) are each either incomplete, insecure, inefficient, slow to update, not private, or some combination thereof. In this paper, we present CRLite, an efficient and easily-deployable system for proactively pushing *all* TLS certificate revocations to browsers. CRLite servers aggregate revocation information for all known, valid TLS certificates on the web, and store them in a space-efficient *filter cascade* data structure. Browsers periodically download and use this data to check for revocations of observed certificates in real-time. CRLite does not require any additional trust beyond the existing PKI, and it allows clients to adopt a fail-closed security posture even in the face of network errors or attacks that make revocation information temporarily unavailable.**

We present a prototype of CRLite that processes TLS certificates gathered by Rapid7, the University of Michigan, and Google's Certificate Transparency on the server-side, with a Firefox extension on the client-side. Comparing CRLite to an idealized browser that performs correct CRL/OCSP checking, we show that CRLite reduces latency and eliminates privacy concerns. Moreover, CRLite has low bandwidth costs: it can represent *all* certificates with an initial download of 10 MB (less than 1 byte per revocation) followed by daily updates of 580 KB on average. Taken together, our results demonstrate that complete TLS/SSL revocation checking is within reach for all clients.

I. Introduction

The TLS protocol, coupled with the web's Public Key Infrastructure (PKI), is the cornerstone of security for billions of users and organizations. TLS[1] relies on certificates issued and cryptographically signed by Certificate Authorities (CAs) to provide integrity, confidentiality, and authentication for web traffic. To date, most web browsing occurs over HTTPS [18].

One critical, but sometimes overlooked, facet of the web's PKI is certificate *revocation*. When a CA erroneously issues a certificate [2], or when a certificate's private key is potentially compromised [76], it is imperative that the affected certificate be revoked. Otherwise—if an erroneous or compromised certificate is not revoked—client software (*e.g.*, web browsers) will continue to believe that the certificate is valid until it expires, which may not occur for years [76]. Attackers could use such certificates to perform effective Man-in-the-Middle (MitM) and phishing attacks against users. Thus, it is crucial that certificate owners revoke erroneous or compromised certificates in a timely manner, and that client software properly checks for revocations. Failing to do so is particularly worrisome in the wake of large-scale vulnerabilities like the Debian PRNG bug [75] and Heartbleed [19], [76], which potentially compromised millions of private keys.

Despite the importance of revocations, many client applications do not properly check for certificate revocations. Liu *et al.* [49] found that recent versions of Chrome only make Certificate Revocation List (CRLs) or Online Certificate Status Protocol (OCSP) requests for Extended Validation (EV) certificates (a very small subset of all certificates); Firefox only supports revocation checks via OCSP; and no major mobile browsers check for revocations *at all*. This unfortunate state of affairs has several root causes, including latency concerns (contacting third-parties to perform revocation checks increases connection latencies), bandwidth considerations (some CRLs are over 70 MB), privacy risks (OCSP checks leak the user's browsing behavior), and ambiguity (CRL and OCSP servers may be temporarily unavailable due to network errors, or an active attack) [42], [41].

Recent efforts aim to address these problems by moving from a pull model (wherein clients download revocation information on-demand) to a push model, such as OCSP Stapling [22], Google's CRLSets [40], and Mozilla's OneCRL [31]. (We review these in detail in § II.) Although these efforts by browser vendors are a step in the right direction, they are far from comprehensive: as of January 30, 2017, CRLSet and OneCRL contained 14,436 and 357 revocations, respectively, while we find that there are over 12.7M revoked (but otherwise valid) certificates that were issued by major CAs (details in § IV). In fact, CRLSet and OneCRL would have significant difficulty scaling to handle millions of certificates, because their data formats use 110 and 1,928 bits per revocation, respectively, meaning they would require between 166 MB and 2.9 GB to store all 12.7M revocations (see § VII-E). Additionally, CRLSet and OneCRL require users to place unconditional trust in Google and Mozilla, since these data structures are not auditable.

In this paper, we present CRLite, a system for proactively pushing *all* certificate revocations to browsers on a regular basis. CRLite is implemented in two parts: a server-side system that aggregates revocation information for all known, valid TLS certificates[2] on the web and places them in a *filter*,

[1] In this paper, we refer to the more modern TLS protocol, although our work also applies to the older SSL protocol.

[2] We validate certificates against the roots in the macOS certificate store.

DOI 10.1109/SP.2017.17

	Revocations Covered	Bandwidth Cost	Update Speed	Push Model?	Private?	Auditable?	Failure Model	Deployable Today?
CRL	**All**	29 KB per CRL†	7 days‡	No	**Yes**	**Yes**	fail-open	**Yes**
OCSP	**All**	1.3 KB per request†	4 days‡	No	No	**Yes**	fail-open	**Yes**
CRLSet [40]	14,436	250 KB per day	**1 day**	**Yes**	**Yes**	No	fail-open	**Yes**
OneCRL [31]	357	34 KB per day	**1 day**	**Yes**	**Yes**	No	fail-open	**Yes**
Rev. Trans. [44]	**All***	—	—	No	No	**Yes**	fail-open	**Yes**
RevCast [65]	**All***	0 B (421.8bps of FM RDS)	**10s of seconds**	**Yes**	**Yes**	**Yes**	**fail-closed**	No
CRLite	**All***	10 MB initially, 580 KB per day†	**1 day**	**Yes**	**Yes**	**Yes**	**fail-closed**	**Yes**

TABLE I: Comparison of CRLite to other existing and proposed revocation dissemination systems. *: these systems aggregate revocations for all known certificates on the web. †: these results are averages based on empirical measurements (see § VII). ‡: these results are the median times that clients may cache responses, based on empirical measurements (see § VII).

and a client-side component that downloads filters and uses them to check for revocations of observed certificates.

CRLite's filters provide browsers with a precise mapping of all certificates to their revocation status. We make use of a *filter cascade* [74], [64], which is a sequence of compact, probabilistic data structures (*e.g.*, Bloom filters [6]) without either false positives or negatives. We rigorously show in § III-C how to minimize the size of filter cascades, and we empirically show they have small incremental updates, as well.

CRLite's design addresses six challenges:

1) **Efficiency.** CRLite compresses the revocation status of all certificates using a filter. The entire data structure requires only 10 MB to represent the status of over 30M certificates, achieving a significant size savings over naïve approaches.
2) **Timeliness.** CRLite uses delta-updates to keep clients up-to-date with recent revocations, meaning clients only need to download the complete filter once. We create deltas on a daily basis, which are typically small ($\sim$580 KB). In contrast, the median CRL has a lifetime of 7 days, while the median OCSP response of an Alexa Top-1M website expires after 4 days, meaning that clients using CRLite have more up-to-date revocation information than clients that cache traditional revocation information.
3) **Failure Model.** Because CRLite contains all revocations and has no false positives, it allows clients to adopt a *fail-closed* security posture. All other deployable revocation dissemination schemes and all modern browsers that we are aware of adopt a fail-open model, *i.e.*, if revocation information about a certificate is unavailable, the client assumes that the certificate is valid.[3]
4) **Privacy.** Unlike OCSP, CRLs, or other on-demand revocation checking schemes, users using CRLite do not reveal their browsing history to third parties.
5) **Deployability.** CRLite is deployable today, requires no support from CAs or changes to TLS, and can be easily integrated into modern browsers.[4]
6) **Auditability.** Although CRLite relies on a centralized system to aggregate and produce filters, clients do not have to blindly trust it. CRLite provides cryptographically-signed logs that allow any interested party to audit each filter, *e.g.*, to check for erroneous insertions or omissions. We also present an additional deployment model for CRLite that eliminates the need for auditing if CAs become part of the filter generation process (see § VIII).

We present a prototype of our system that uses Spark to create the filter on the server-side, and a Firefox extension on the client-side. We rely on certificate data from full IPv4 scans of the Internet conducted by the University of Michigan [71] and Rapid7 [60], as well as Google's Certificate Transparency logs [45]. As of January 2017, our dataset contains 184M total certificates, from which we identify 30M valid, unexpired certificates. From these certificates, we extract 10K unique Certificate Revocation Lists (CRLs) and 743 unique OCSP responders, which enable us to check the certificates' revocation status. We crawl these each day to produce updated filters.

We perform extensive evaluations on our CRLite prototype using real data. On the client-side, we show that our Firefox extension has low CPU overhead (10 milliseconds to check each certificate chain) and nominal memory usage, and that it induces an order of magnitude less delay per HTTPS connection than using CRL and OCSP. Having low delay is crucial, as browser makers acknowledge that it is their primary concern related to online revocation checking [40]. Also note that our implementation would be even faster if it was implemented natively within the browser, rather than a JavaScript extension.

Table I compares CRLite to six other revocation dissemination schemes. Revocation Transparency is a proposed, centralized, Merkle Tree-based scheme [45] that is similar to historical proposals from Micali and others [51], [38], [55], [52]; RevCast [65] is a system that disseminates certificate revocations over FM radio. CRLite incorporates the best parts

[3] Browser vendors choose to "fail-open"—essentially prioritizing availability over security—to avoid having to refuse a connection (and cause perceived unreliability) when the CRL or OCSP server cannot be contacted. [41]

[4] Our prototype targets Firefox because it makes TLS APIs available to extensions. Other browser do not export TLS APIs, and would thus require additional modification.

of existing systems (*e.g.*, user privacy, no centralization of trust, a fail-closed security posture, instant deployability) while using less bandwidth than an average webpage (2.3 MB as of 2016) [27] and covering all revocations. CRLite is also ideally suited for mobile and resource constrained devices, since it uses a push model to deliver data (like CRLSet and OneCRL) while covering all certificates (like CRL and OCSP).

Outline. This paper is organized as follows. In § II, we discuss background related to the web's PKI and certificate revocation. In § III, we detail the design of the filter cascade. We present the server- and client-side design of CRLite in § IV, and evaluate its security properties in § V. We present the implementation of our prototype in § VI, which we use to evaluate CRLite and compare it to other revocation mechanisms in § VII. In § VIII, we present an alternative design of CRLite that shows that auditing can be far less expensive provided modest CA participation. Finally, we conclude in § IX.

II. Background

In this section, we briefly overview the web's PKI, with a focus on certificate revocation. We also discuss how web browsers currently implement revocation checking, and survey recent work on alternate strategies for distributing revocations.

A. The TLS Ecosystem

Authenticity and confidentiality of communication on the web are provided by HTTPS, which uses a combination of the TLS protocol and a hierarchical PKI. In the web's PKI, trusted CAs are vested with the authority to issue X.509 certificates that bind identities (*i.e.*, domain names) to cryptographic keys. These *leaf certificates* are cryptographically signed by a CA, forming a signature chain that may include zero or more *intermediate certificates*, and eventually terminating at a self-signed *root certificate*. Servers present this chain to clients during the TLS handshake, who must then validate it.

Certificate Validation. As mentioned above, clients must validate certificates that are presented to them, *e.g.*, by verifying the signatures, looking at the certificates' expiration dates, *etc.* In this work, we focus on the challenges of checking for certificate revocations, but researchers have identified other issues related to certificate validation, such as bugs in popular TLS libraries [8], in browsers [49], and in non-browser software [29]. This has motivated researchers to develop novel certificate validation schemes that leverage advanced cryptographic techniques [17].

Certificate Transparency. Because any CA can issue a certificate for any domain, there are significant concerns about CAs improperly issuing certificates (*e.g.*, after a private key compromise [2]). The Certificate Transparency (CT) project has created public, auditable, append-only logs of certificates, with the idea that all newly issued certificates will be added to the log by CAs. Google is encouraging adoption of CT by CAs through Chrome policies: in January 2015, Chrome began requiring that new EV certificates include an SCT record that is signed by a well-known CT log [43]. In October 2017, Chrome will require *all* new certificates to contain SCT records [66], and Firefox is planning to adopt these policies as well [54].

Measuring the TLS Ecosystem. As HTTPS has grown in importance, many studies have empirically examined aspects of the HTTPS ecosystem. Several studies have broadly investigated certificates on the web [20], [23], [36] with recent work demonstrating that IPv4 scans and CT logs are sufficient to gain broad visibility of valid certificates [73]. Chung *et al.* instead examine the hundreds of millions of invalid certificates that linger on the web [15]. Other work has examined the root certificates trusted by clients [58], [72] and the costs of HTTPS security [56]. Lastly, two studies have specifically examined the security implications of distributing private keys across Content Delivery Networks (CDNs) [48], [11].

B. TLS Certificate Revocation

Revocation is a crucial component of the web's PKI. At any time, the owner of a certificate may request that their CA revoke the certificate, which produces a public, cryptographically-verifiable attestation that the certificate should no longer be trusted (even if it has not expired). There are many reasons why a certificate may be revoked, such as if it uses a weak key [75], but the most important cases occur when a private key is (potentially) compromised [76], [19], or when a certificate is issued erroneously. In these cases, an attacker can misuse the compromised certificate to conduct MiTM or phishing attacks until it expires. Thus, it is vital that such certificates be revoked, and that clients check to see if certificates they are offered in the TLS handshake are revoked.

There are two primary protocols for distributing revocations:

CRLs. A Certificate Revocation List (CRL) is a list of serial numbers from revoked certificates that is signed by a CA. CAs are responsible for including a URL in each certificate they sign that points to the associated CRL. In turn, clients are responsible for downloading the CRLs associated with observed certificates to check if they are revoked. CRLs are signed by the CA to protect their integrity, and contain a validity period during which they may be cached (up to a maximum of 10 days [10]).

OCSP. The successor to CRL is OCSP. OCSP is a web service protocol that allows clients to query a CA for the revocation status of a single certificate. CAs are responsible for inserting a URL into each certificate they sign that points to the corresponding OCSP responder. Similar to CRLs, OCSP responses are signed by the CA, and contain a validity period during which they may be cached.

Recent measurement studies demonstrate that revocation is prevalent in the web's PKI [49]. More than 99% of valid certificates available on the web contain a reachable CRL URL, while 95% include a reachable OCSP responder. Liu *et al.* observe that 8% of all valid certificates are revoked (6% if we focus just on valid EV certificates) [49], with the bulk of these revocations occurring due to Heartbleed [76], [19].

C. Revocation Checking

Clients are responsible for checking all leaf and intermediate certificates they are offered during a TLS handshake for revocations. This can be done by downloading the certificates' CRLs or contacting their OCSP servers, depending on what information is provided in the certificates. Alternatively, servers may push revocation information to the client by "stapling" an OCSP response to the certificate via the OCSP Stapling TLS extension [22], although as of 2015, only 3–5% of certificates on the web were served by hosts that supported stapling [49].

Revocation Checking in Practice. Unfortunately, work by Liu *et al.* shows that browsers do an extremely poor job of checking for revocations [49]. For example, Firefox no longer supports CRLs, while Chrome only issues online requests for EV certificates. Most alarmingly, no major mobile browsers perform any online revocation checks. As a result, users may end up as victims of MitM or phishing attacks due to compromised certificates.

Browser vendors have chosen not to implement thorough revocation checking for a variety of reasons. First, downloading CRLs and making OCSP requests adds delay to the establishment of HTTPS connections. CRLs in particular can become quite large (Apple has a CRL that is over 76 MB! [49]). Second, using OCSP has privacy implications for users, because OCSP requests enable CAs to passively observe the domains users browse to. Although stapling addresses OCSP's privacy problem, stapling is vulnerable to downgrade attacks: an in-network attacker can strip the staple from a certificate, which forces the client to resort to a traditional CRL or OCSP check.

To address stapling downgrade attacks, RFC 7633 defines the OCSP "must-staple" extension, which allows a certificate to require that the server provide a stapled OCSP response during the TLS handshake [34]. Must-Staple effectively addresses latency, privacy, and fail-open issues, but only for certificates that include the new extension; it does not apply to the millions of certificates already in existence. Moreover, this protocol requires that HTTPS servers and browsers be upgraded to support it.

Fail-open vs. Fail-closed. No currently-deployed approaches to disseminating revocations push all revocations to clients. As a result, clients must sometimes make an external request to obtain the revocation status for a certificate (assuming they bother to check at all). However, clients must decide what to do if this request is not answered (*e.g.*, due to transient network error, server failure at the CA, or a MitM attack). All existing systems that we know of adopt a *fail-open* model, whereby they accept a certificate if revocation information cannot be obtained. Browser vendors argue that choosing this model is necessary, as a *fail-closed* model would cause an unacceptable level of failures [41]. However, the fail-open model provides little additional security, as an attacker who can filter the client's traffic can block the revocation status request and cause a revoked certificate to be accepted.

CRLite aims to sidestep this conundrum by ensuring clients have have up-to-date revocation information available for all certificates. As a result, clients need not be faced with a decision between availability and security.

CRLSet and OneCRL. To mitigate the performance and privacy issues surrounding CRL and OCSP, browser vendors have begun pushing partial lists of certificate revocations directly to users' browsers. Google's CRLSet, introduced in 2013, contains between 14K–25K revocations (depending on the date) [49]; this list is updated daily and pushed to Chrome browsers. Google's CRLSet documentation lays out some of the inclusion/exclusion criteria used to decide which revocations appear in the list [40], but prior work has shown that the inclusion criteria remain opaque [49]. Similarly, Mozilla introduced OneCRL in 2015, and it has grown to include 357 revocations. However, OneCRL only includes revoked intermediate certificates [31].

Adam Langley, from the Chrome security team, has investigated the possibility of using compressed, probabilistic data structures for distributing all revocations to clients [39]. Although Google does use compressed data structures in other applications (*e.g.*, SafeBrowsing [63]), no browser that we are aware of has adopted this approach for distributing revocations.

D. Other Revocation Distribution Schemes

Many classic proposals have been made for alternative certificate revocation mechanisms, such as Micali's Certificate Revocation System [51], multi-certificate revocation [1], revocation trees [38], [55], [52], or combinations of these techniques [28], [25]. Recent work has explored extending certificate transparency initiatives [45], [13] to also incorporate revocations [44], [62]. Others have focused on scaling up revocation infrastructure in the face of byzantine failures [77]. Unfortunately, these systems still require clients to perform online revocation checks, thus adding latency to connections. Additionally, several of these schemes require clients to query a central server for every single certificate, which reveals users' browsing behavior.

The Perspectives project aims to obviate the need for revocation (and indeed any form of centralized trust) by relying on multiple, independent observations to determine the veracity of certificates [4]. The assumption is that legitimate certificates will be seen by many users, while fraudulent or stolen certificates will only be seen by a small subset of users who are under attack. However, the security guarantees offered by this approach are probabilistic, and as of 2016, this initiative has failed to gain traction.

Other approaches have proposed more significant modifications to existing systems to enable better revocation dissemination. Chariton *et al.* propose to distribute revocations through DNS [12], much like how DANE uses DNS to distribute certificates [35]. Szalachowski *et al.* propose to distribute revocations to middleboxes in the network that observe TLS handshakes and insert up-to-date revocation information for observed certificates [68]. However, these systems require significant buy-in before they will be practical (from CAs and clients for the former, and from CAs and CDNs for the latter).

In contrast, CRLite is built so that end users can opt-in via a browser extension today.

Schulman *et al.* design a system to distribute revocations through FM radio broadcasts [65]. Although they show that radio broadcasts are an ideal medium for large-scale dissemination of revocations, all clients would need to install FM radio receivers for the scheme to be deployed. Lastly, proposals like AKI [37], PoliCert [70], ARPKI [3], and PKISN [69] aim to replace the existing PKI with a new hierarchy that avoids centralizing trust, is transparent, and supports seamless revocation. However, as with any clean-slate proposal, future adoption of these techniques is uncertain because they necessitates changes to CAs, clients, and (in some cases) certificates. In contrast, one of our goals is to develop a system that is immediately deployable by not requiring changes to CAs and certificates. Interestingly, the authors of PKISN suggest pushing all revocations to clients, but they do not address the problem of encoding this data such that the size is not prohibitive for clients [69].

III. Filter Cascades

There has been considerable work regarding how best disseminate certificate revocations. This has largely involved exploring many different data structures, including lists in the standard CRL [16], Merkle trees [38], [44], [28], [55], [25], [1], Bloom filters [49], and variants thereof [39]. The broadly accepted conclusion amongst this wide-ranging work has been that the trade-offs between *timeliness* (getting all new revocations to clients as quickly as possible) and *bandwidth* are too great to be realistic—some going so far as to suggest eliminating revocation lists altogether [61], [50]. As a result, today's browsers are restricted to a tiny subset of the web's revocations, often checking no revocations whatsoever [49].

In parallel, there have been impressive efforts to make publicly available the set of *all* live certificates on the web. Notably, the CT project [45] and various Internet-wide scanning efforts [21], [60] have brought us to the point that we can reasonably assume that we have a nearly *complete* view of the entire certificate ecosystem [73]. We demonstrate in this section how to use this new information to compactly represent all outstanding revocations.

What stymied prior efforts to create a compact view of revocations was that, ultimately, there is only so much that an arbitrary set of millions (or more) objects can be compressed. We are not subjected to the same constraints. Our insight is that our data structure need not support queries for any arbitrary data item—rather it needs to successfully support queries only for the finite set of unexpired certificates.

A. Insight

To see why this subtle difference is so powerful, we first recall Bloom filters [6]. A Bloom filter is a probabilistic data structure that permits inserting arbitrary data items $d \in \{0,1\}^\infty$, and testing for membership of arbitrary data. They operate by maintaining a bit vector of size m and k hash functions $h_i : \{0,1\}^\infty \to \mathbf{Z}_m$ for $i = 1, \ldots, k$. To insert a data item d, one sets each $h_i(d)$ bit to one (in a traditional Bloom filter, there is no deletion). Testing the set membership of a data item d' checks bits $h_i(d')$; if any of them are zero, then d' is definitively not in the filter, otherwise it *may* be. That is, although Bloom filters have no false negatives, they have some false positive rate $0 < p < 1$ determined by the filter's size, occupancy, and number of hash functions.

Consider a Bloom filter BF with false positive rate p. How many false positives can BF have? Because Bloom filters support insertion and membership queries for arbitrary bit strings, there are an infinite number of false positives. In other words, the set of false positives for a Bloom filter is always larger than the set stored in the filter itself.

However, suppose we knew that all set membership queries were going to come from a *finite* set U . In that case, if $R \subseteq U$ (*e.g.*, the set of all revoked certificates) were stored in BF, then the expected number of false positives would be $p \cdot |U \setminus R|$. This is a strictly *smaller* set than the one that BF was supposed to support membership queries for (U). Thus, if membership queries come only from a known, finite set of items, then the resulting set of false positives will be even smaller.

The key insight is that this small set of false positives can be stored in *another* Bloom filter, and so on, until the number of false positives is zero. In traditional settings, one may not be able to assume that membership queries will come from a constrained set. However, because we can now know the set of virtually all certificates at any time, and because CAs are increasingly adopting CT, we believe we can at last safely make this assumption.

B. Filter Cascade Design

This notion of using a sequence of Bloom filters to store increasingly smaller sets of false positives was originally introduced by Chazelle *et al.* in what they referred to as *Bloomier filters* [14], and refined by subsequent work to develop *filter cascades* [74], [64]. For completeness, we describe the design of a Bloom filter cascade[5], and show how it achieves zero false positives despite being compact.

Suppose that we wish to store a set $R \subseteq U$ of data items, and that $R \cup S = U$.

Insertion into a filter cascade begins by creating a *first-level* Bloom filter (BF_1) with the optimal size and number of hash functions to achieve a given false positive rate $0 < p < 1$ (we show how to optimize p at each level in § III-C; for ease of exposition, assume for now that there is a fixed p across all levels). Into BF_1, we insert each element of R, as normal. If some data item u is not in BF_1, then it is definitively *not* in R, but not vice versa. Thus, the set of first-level false positives (FP_1) contains the elements of S that also appear in BF_1. In expectation, $|\mathsf{FP}_1| = p \cdot |S|$.

Our next task is to compactly represent this set of false positives. To this end, we construct another, *second-level*

[5] In practice, a filter cascade could be made up of any compact filter with false positives, such as cuckoo [26], quotient [5], or Golomb [59] filters. Exploring their trade-offs in CRLite is an interesting area of future work.

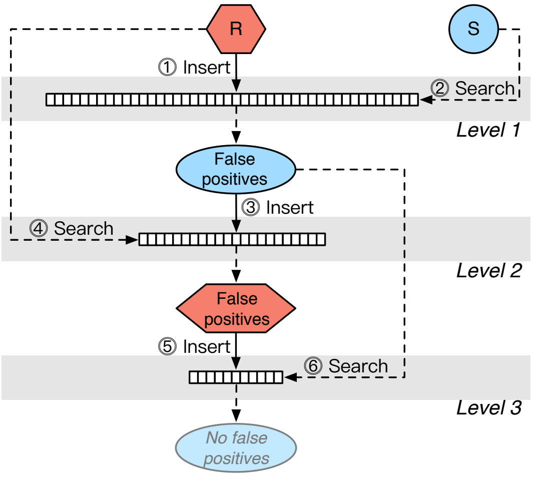


Fig. 1: Inserting into a filter cascade. Dashed arrows represent searches; solid arrows insertions.

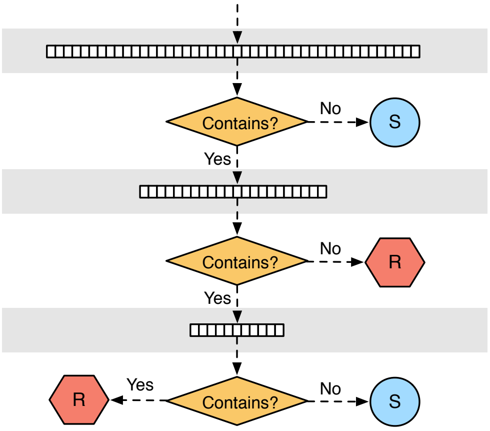


Fig. 2: Lookups in a filter cascade. All lookup queries return definitively.

Bloom filter (BF_2) and insert each element of FP_1. The idea is that BF_2 in essence serves as a "blacklist" to BF_1: it contains the items that should *not* have been in BF_1. Thus, if a data item u is in BF_1 but is *not* in BF_2, then it is definitively *in* R. However, BF_2 can also have false positives. The set of second-level false positives (FP_2) contains the elements of R that appear in BF_2, and is in expectation of size $p \cdot |R|$.

These are our two base cases; we show one more inductive step. If FP_2 is nonempty, then we construct a third-level Bloom filter (BF_3) in which we insert each element of FP_2. If an element is in BF_1 and BF_2 but is not in BF_3, then it is definitively *not* in R. That is, like with BF_1, BF_3 serves as a "whitelist": elements of this filter represent elements of R (and therefore the elements ideally would not have been in BF_2). However, unlike BF_1, the false positives at this level (FP_3) do not come from the entire set S, but rather only the members of S that have not already been ruled out by higher-level filters, *i.e.*, the members of S that are also in BF_1, which is precisely FP_1. In expectation, $|\mathsf{FP}_3| = p \cdot |\mathsf{FP}_1| = p^2 \cdot |S|$.

This process continues: so long as FP_i is nonempty, then we construct a Bloom filter BF_{i+1} and insert into it all elements of FP_{i-1} for $i \geq 2$. Odd-numbered levels represent whitelists (elements that are in R) and even-numbered levels represent blacklists (elements that are not). Figure 1 shows an example with three layers.

Lookup queries in a filter cascade are constrained to U and take the form: "is $u \in U$ in set R?" We emphasize that clients issuing such queries need not know all elements of the set U; it must only be the case that whoever constructed the filter cascade was aware of all possible values $u \in U$ for which it would subsequently be queried.

Lookup queries take a top-down approach similar to insertions. Recall that Bloom filters provide definitive answers only for items *not* stored in the filter, and potentially false positives otherwise. With filter cascades, we can also provide definitive answers in the positive when we know that there are no false positives from the set U.

Putting this together, lookups in a filter cascade begin at level $i = 1$ and continue until the first BF_i is found where $u \notin \mathsf{BF}_i$. At this point, the "is $u \in U$ in set R?" can be answered as follows:

- If i is odd, then u is definitively *not* in R.
- If i is even, then u is definitively *in* R.

If no such BF_i is found (*i.e.*, if u is in all BF_i), then the total number of levels l in the filter cascade determines the answer:

- If l is odd, then u is definitively *in* R.
- If l is even, then u is definitively *not* in R.

Figure 2 shows an example of lookups in a three-layer filter cascade. Note that each level offers definitive answers when a data item is not in that level, and the final level is always definitive.

C. Minimizing Filter Cascade Size

We seek to minimize the size of a filter cascade, so as to consume as little bandwidth as necessary to keep browsers up-to-date with revocation data. To this end, we formally analyze the filter cascade's size and number of levels, and use our findings to develop a strategy for setting false positive rates in a way that minimizes the overall size. To the best of our knowledge, this is the first size-minimizing analysis done on filter cascades, and we believe it to have applications beyond that of certificate revocation.

False Positives and Size. An oft-cited bound on the false positive probability p for a Bloom filter in which r items are inserted into a bit-array of size m using k hash functions is

$$p \leq \left(1 - e^{-rk/m}\right)^k$$

This bound is "proved" by making the slightly untrue assumption that whether one bit is set in the Bloom filter is independent of whether any other bits are set, and approximating $1-\frac{1}{m}$

by $e^{-1/m}$. But for large values of r and m, it is very close to a rigorous bound proved by Goel and Gupta [30], which we will turn to later. The number of hash functions, k, must be integral, as must m and n. Putting aside the issue of integrality for the moment, for a given p, the size of the Bloom filter is minimized by setting $k = \frac{m \ln 2}{r} = \log_2 \frac{1}{p}$ in which case the size is given by

$$m \;=\; \frac{r \ln \frac{1}{p}}{(\ln 2)^2} \;\approx\; 1.44 r \log_2 \frac{1}{p}$$

This formula is accurate provided that m and r are large and (most importantly) $\log_2(1/p)$ is close to integral.

Normally, when using a Bloom filter, one chooses p as a design constraint: it represents a trade-off between the uncertainty the system can accommodate and the cost in space it can afford. However, in filter cascades, there is no uncertainty: there will ultimately be no false positives, and thus p only affects the overall size and number of levels.

What, then, is the correct strategy for setting false positive rates in a filter cascade to minimize the overall size?

Lower Bounds. Before answering this question, we pause to examine a lower bound. Let $r = |R|$, $s = |S|$, and $n = r + s$. The number of bits needed to communicate R is at least $\log_2 \binom{n}{r}$. Applying Sterling's approximation for $n!$ ($\sqrt{2\pi n}\left(\frac{n}{e}\right) \leq n! \leq e\sqrt{n}\left(\frac{n}{e}\right)$), this gives us a lower bound of

$$r \log_2 \frac{n}{r} + (n-r)\log_2 \frac{n}{n-r} + \frac{1}{2}\log_2 \frac{n}{rs} + \log_2 \frac{e}{2\pi} \quad (1)$$

When $r \ll n$, the second term approaches $r/\ln 2 \approx 1.44r$ and the third term approaches $-(1/2)\log_2 r$. The fourth term is about -1.2, so that the dominant terms in Eq. (1) are $r \log_2 \frac{n}{r}$ and $1.44r$. The lower bound can be met if both parties share an ordered list of all n certificates. Unfortunately, when representing real certificates, we cannot assume a globally known ordered list. However, as we show next, with the right choice of false positive rates, filter cascades can be constructed using $1.44r \log \frac{n}{r} + 4.2r$ bits (or perhaps even less).

False Positive Strategy. As the following analysis shows, a strategy of using one false positive probability, p_1, at the first level, and a second false positive probability, p, at all subsequent levels produces a filter cascade whose size is competitive with the lower bound[6]. The simplicity of the strategy makes it straightforward to implement and analyze.

Observe that the size of a Bloom filter at the first level is a function of r and the desired false positive probability, but does not depend on s. Thus, if r is less than s we have some leverage, as we can reduce the number of elements in S using a Bloom filter whose size is based on the smaller value r. We set the false positive probability at the first level to bring the expected number of elements in S down to about r. In particular, we choose $p_1 = r\sqrt{p}/s$, so that the expected number of false positives among the s non-revoked certificates is $r\sqrt{p}$. The $\sqrt{p}$ factor is included in the formula for p_1 so that at all subsequent levels, the ratio between the expected number of elements inserted to the number that are not inserted is always the same, $\sqrt{p}$, which simplifies the analysis.

As a running example, for all levels after the first let $p = 1/2$ and $k = 1$, so that $p_1 = r/\sqrt{2}s$ and the expected number of false positives at the first level is $r/\sqrt{2}$. The size of the first level is $1.44r\log_2(s/r\sqrt{p})$, which, for $p = 1/2$, is at most $1.44r \log_2 \frac{n}{r} + .72r$. The first level is the only one that requires $\Omega(r \log(n/r))$ bits and the leading constant of 1.44 is small. After the first level, the expected number of elements remaining in $R \cup S$ is $O(r)$ (i.e., $(1+1/\sqrt{2})r$ in our example), so only $O(r)$ additional bits will needed.

We now analyze the size of all the levels after the first. The achieved false positive rate at each level is a random variable that depends on the random hash function chosen at that level. These variables are independent, so the expectation of their product is equal to the product of their expectations. Hence, at level $i+1$, the expected number of items to be inserted into the Bloom filter is $r(\sqrt{p})^i$, and the expected number of items not to be inserted but for which false positives might occur is $r(\sqrt{p})^{i-1}$. The total expected size over all levels after the first is then given by

$$\sum_{i=1}^{\infty} 1.44r(\sqrt{p})^i \log_2 \frac{1}{p} = \frac{1.44r\sqrt{p}\log_2 \frac{1}{p}}{1-\sqrt{p}} \qquad (2)$$

For $p = 1/2$, the sum comes to $3.48r$. Hence, in our example the expected total cost over all levels is $1.44r \log_2 \frac{n}{r} + 4.2r$. Although we have not paid particular attention to integrality constraints, by choosing $p = 1/2$ and $k = 1$ in our example we have taken care of the most important such constraint.

Simulations. We close this section by empirically evaluating the size and lookup times of filter cascades using simulation. For a variety of values of r and s, we conduct a systematic search through the possible values of p_1 and p to try to minimize the total side of the filter cascade. In this empirical analysis we insist that m, n, r, s, and the number of hash functions at each level be integral. To calculate expected Bloom filter size, we use the following bound due to Goel and Gupta [30], which holds for $m > 1$ and is derived rigorously, making no independence assumptions or approximations:

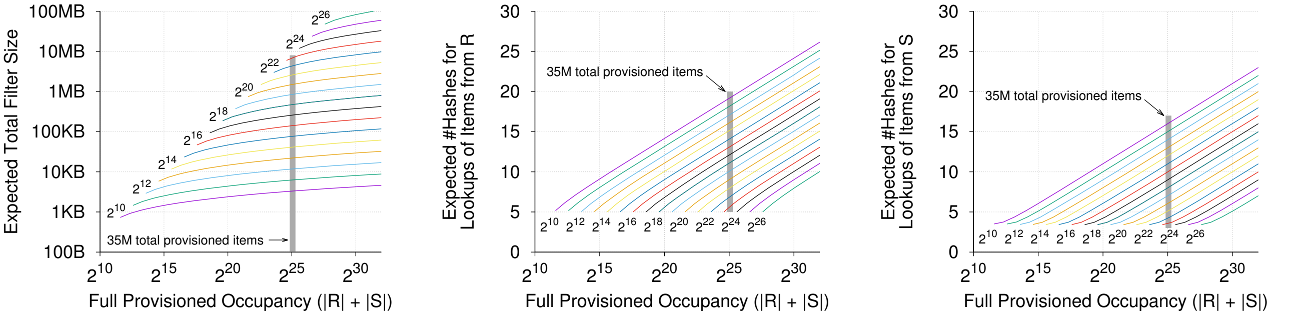
$$p \leq \left(1 - e^{-\left(r+\frac{1}{2}\right)k/(m-1)}\right)^k$$

We find that, in practice, the optimal p tends to be very nearly $1/2$ when $r \ll s$. In estimating the expected number of levels, we apply the bound derived in Appendix A.

Figure 3a shows that the overall size of a filter cascade is determined primarily by the number of elements in R. This is because, as discussed above, the size of the filter cascade is dominated by the size of the first Bloom filter, into which each element of R must be inserted.

Figures 3b and 3c show the expected number of hash functions that need to be computed when looking up an element of R or S, respectively. We make two key observations: First, lookup times increase the smaller $|R|$ is relative to $|S|$; when

[6] Note that prior work on filter cascades assumes a single false positive rate for all levels [74], [64], and thus has slightly worse space utilization.



(a) Expected size of a filter cascade when filled to provisioned occupancy.

(b) Expected number of hashes when looking up an element from R.

(c) Expected number of hashes when looking up an element from S.

Fig. 3: Simulation results of a filter cascade. The size and expected lookup time are functions of the occupancy of both the stored set (R) and its complement (S). Numbers annotating lines represent $|R|$.

there are 35M total provisioned items, it takes ∼19 hashes to look up an element of R when $|R| = 1{,}024$, but only ∼5 hashes when $|R| = |S|$. This is because $p_1 = r\sqrt{p}/s$ and therefore the number of hash functions in the first filter is proportional to $\log s/r$. Second, it takes fewer expected lookups for elements of S than for R: typically 1.5–3.1 fewer hashes in expectation. This is because elements of S can be ruled out in the first level, with a smaller false positive rate (p_1), while every element of R must be checked in the first two levels, at least.

Broadly, these empirical results demonstrate how promising filter cascades are for certificate revocation: Their size is determined predominantly by R, and therefore will not grow considerably with the more prevalent, non-revoked certificates S. Moreover, lookup times for elements of S are faster, which is encouraging because non-revoked certificates are encountered more frequently.

IV. System Design

In this section, we present the design of CRLite. This system applies recent advances in Certificate Transparency [45], along with filter cascades, to achieve complete and universal dissemination of certificate revocation information. At a high level, we aggregate *all* revocation information for *every* known certificate, compactly represent them in a filter cascade, and provide a means by which clients can publicly audit us. We begin this section by describing our goals, assumptions, and threat model. Then, we discuss the CRLite protocol from the perspective of the server and client, respectively. In subsequent sections, we describe the implementation, analysis, and empirical evaluation of CRLite that collectively show that it is effective and practical to deploy *today*.

A. Goals

Our primary objective is to develop a system that quickly and efficiently pushes *all* available revocations to web browsers. To make widespread adoption possible and desirable, our system must:

- Not require changes to CAs, certificates, or websites.
- Be incrementally deployable and incur minimal changes to clients.
- Protect client privacy.
- Offer security guarantees that are no worse (and hopefully much stronger) than existing revocation schemes.
- Not increase (and ideally reduce) page-load times and bandwidth consumption as compared to existing revocation checking schemes.

We are far from the first to propose these goals, but prior efforts have been unable to achieve all of these properties in tandem (see § II). As we will demonstrate, the advent of CT [45] puts them within grasp, at last.

B. Threat Model and Assumptions

Fundamentally, an attack against CRLite seeks at least one of three outcomes: (1) to make a valid certificate appear revoked, (2) to make a revoked certificate to appear valid, or (3) to harm client's user experience by delaying or halting a user's ability to obtain revocation information.

We operate within a set of assumptions that are standard in today's web. We assume an active attacker that is able to manipulate a victim's web traffic, *e.g.*, via a man-in-the-middle (MitM) attack or blocking traffic. However, we make standard cryptographic assumptions: particularly that the attacker is unable to forge signatures without access to a principal's private key. Additionally, we make two assumptions that are standard in the web's PKI: *First*, we assume that clients trust a common set of root certificates [58] to be benign and uncompromised. *Second*, we assume that clients' clocks are loosely synchronized, so that they may check the expiration dates of certificates and of the data we disseminate, to within the order of about a day. Finally, like with the OCSP Must-Staple option [33], we assume that each CA's revocation information is available to our aggregation server at least once every 12 hours to ensure that clients are up-to-date to within a 24-hour period.

Our design uses a logically centralized *aggregator* that is responsible for obtaining all certificates (*e.g.*, from CT logs) and

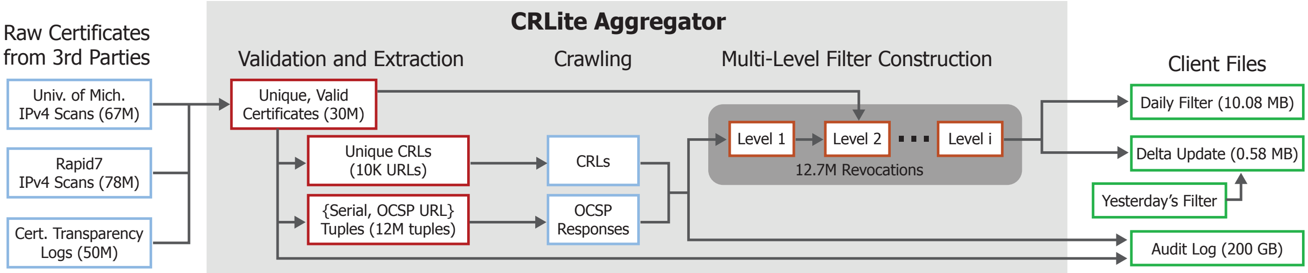

CRLite Aggregator
Raw Certificates from 3rd Parties
Univ. of Mich. IPv4 Scans (67M)
Rapid7 IPv4 Scans (78M)
Cert. Transparency Logs (50M)
Validation and Extraction
Unique, Valid Certificates (30M)
Unique CRLs (10K URLs)
{Serial, OCSP URL} Tuples (12M tuples)
Crawling
CRLs
OCSP Responses
Multi-Level Filter Construction
Level 1
Level 2
Level i
12.7M Revocations
Client Files
Daily Filter (10.08 MB)
Delta Update (0.58 MB)
Yesterday's Filter
Audit Log (200 GB)

Fig. 4: Server-side pipeline of CRLite. Each day, this process constructs a new filter cascade, delta update to the previous day's filter cascade, and an audit log. Example statistics are given as of January 30, 2017.

their corresponding revocation information. We assume CT servers and scanning techniques like ZMap [21] to be trusted, *i.e.*, that they distribute full views of the HTTPS ecosystem to the best of their ability. We assume that aggregators can misbehave by asserting that a valid certificate is revoked or that a revoked certificate is valid. We describe in this section a public audit procedure that involves no participation from an aggregator but results in a proof of misbehavior.

C. Server-Side Operation

The server-side aggregator is responsible for collecting raw data and producing filters and auditable logs for clients. We present an overview of the aggregator's operation in Figure 4.

Obtaining Raw Certificates. To create the filter of revoked certificates, CRLite first needs a list of all valid certificates. VanderSloot *et al.* [73] show that >99% coverage of all TLS certificates known to exist on the web can be obtained by using two sources of certificates: full IPv4 scans on port 443, and Google's CT logs. Thus, we adopt these data sources for CRLite. In the future, if CRLite becomes popular, it would be trivial to extend our certificate dataset by allowing CAs or other interested parties to submit missing certificates, similar to Google's CT logs [45]. Indeed, a logical place to deploy an aggregator would be at a popular CT site.

As shown in Figure 4, CRLite takes as input certificates from University of Michigan's IPv4 scans (covering October 2013 to February 2014) [71], Rapid7's IPv4 scans (covering February 2014 to January 2017) [60], and Google's CT log (from the Pilot server) [45]. Rapid7 conducts new scans on a roughly weekly basis, which CRLite automatically downloads and adds to its database. Similarly, CRLite mirrors the transparency log on a daily basis. In total, these data sources contain 184M unique certificates (though as we will show, most of these are invalid [15]).

Validating Certificates. The next step in our pipeline is cleaning the certificate data. Specifically, CRLite validates all certificates by looking for non-expired, well-formed leaf and intermediate certificates that cryptographically chain to a trusted root from the macOS key store. This process is incremental, *i.e.*, CRLite performs full cryptographic validation on new certificates gathered during the last 24 hours, and clears recently expired certificates from its database. As of January 2017, our database contains 30M valid certificates.

Obtaining All Revocations. To collect revocations, CRLite extracts CRL and OCSP responders from all valid certificates. To obtain revocation information for all certificates as efficiently as possible, we adopt the following approach: if a certificate has a CRL URL, we extract it; otherwise, we extract its OCSP responder and serial number. As of January 2017, 10K unique CRLs cover 18M of the valid certificates, leaving 12M certificates for which CRLite must query an OCSP responder.[7] Note that 99% of the OCSP-only certificates are issued by Let's Encrypt [46].[8]

CRLite downloads all extracted CRLs and queries respective OCSP responders for the revocation status of the OCSP-only certificates. As of January 30, 2017, there are 12.7M revoked certificates in our database. Finally, CRLite constructs the set of non-revoked certificates by subtracting the revoked certificates from the set of valid certificates.

Filter Cascade Construction. The next step is to construct a filter cascade. Recall from § III that this data structure stores some set R in a way that any query from the set $R \cup S$ will result in a definitive, accurate answer (*i.e.*, without any false positives or negatives). In CRLite's scenario, R is the set of revoked certificates and S the set of non-revoked certificates.

In practice, the sets of revoked and valid certificates change over time: new certificates are added, while existing certificates can expire or be revoked. CRLite produces a fresh filter cascade each day that incorporates these changes. However, we also observe that the certificate universe is expanding over time, as HTTPS becomes more widely adopted. To account for this, we provision the filter cascade to hold $\delta_r \cdot |R|$ and $\delta_s \cdot |S|$ certificates, for some $\delta_r, \delta_s > 1$, but only perform insertions for certificates in R and S. We choose δ_r, and δ_s based on the estimated growth rate of R and S over some reasonable time frame, so that the we only need to reparameterize the filter periodically (monthly in our case). This simplifies the

[7] Note that 163 valid certificates exist in our database that contain neither a CRL nor an OCSP responder; these certificates are particularly dangerous, as they can *never* be revoked.

[8] We obtained permission from the Let's Encrypt operators to send such a high volume of OCSP requests to their servers.

production of delta updates, which we describe next. As we show in § VII-A, the size for the full filter is only 10 MB.

Delta Updates. Although the size of our filter cascade is quite reasonable (see § VII-A), it would still be onerous for clients if they had to download fresh copies of the entire filter each day (especially clients subject to data caps). Furthermore, as we show in § VII-B, $|R|$ and $|S|$ only fluctuate by a few percent day-to-day under typical conditions (the exception being events like Heartbleed), meaning that relatively few bits in the filter are changing.

To address this issue and exploit the common-case dynamics of the certificate ecosystem, CRLite produces *delta updates* that allow clients to incrementally update their copy of the filter. As shown in Figure 4, CRLite compares each day's filter to the previous day's filter to produce the delta update; conceptually, we can think of this as a bitwise XOR of the Bloom filters at each level (in practice we use the efficient bsdiff tool, which is essentially diff optimized for binary data). We demonstrate empirically in § VII-B that these deltas tend to be sparse (a small percentage of certificates change on a daily basis). Finally, the server signs the delta update and makes it available to clients. Clients that are d days out-of-date can simply download and apply the most recent d delta updates to their filter (or the latest full filter, whichever is smaller). As we show in § VII-B, the mean size for delta updates is 580 KB.

Audit Log. The last file produced by CRLite is an audit log. The audit log is designed to address the issue of trust, *i.e.*, *how can clients be sure that the CRLite server is constructing filters correctly?* The audit log contains (1) copies of all CRLs and OCSP responses that were used to construct the corresponding filter cascade, and (2) copies of all certificates included in the whitelist. Recall that each CRL, OCSP response, and certificate is signed by its CA, and the audit log as a whole is signed by the CRLite server. Using this data, a third-party can cryptographically verify the integrity of the inputs for a given filter, build a local copy of the filter cascade, and then compare it to the filter provided by the server. If the filters do not match, then the CRLite server incorrectly omitted or inserted a revocation into the filter. Furthermore, the third-party can verify that all (serial number, URL) tuples in the whitelist are not revoked in the corresponding CRLs/OCSP responses. In § VIII, we describe an alternate design that shows with modest CA participation, we can remove the need for audits.

Hosting. CRLite makes filters, delta updates, and audit logs available to clients via a standard web server. In practice, these files could easily be hosted on a cloud-storage service like Amazon S3 or on a Content Delivery Network (CDN), as is common with CRL and OCSP servers today. After several weeks, the system automatically deletes stale files.

We emphasize that none of the server operations described here require participation from CAs, websites, or CT servers. As a result, *anyone* could run a CRLite aggregator today. That said, we believe that those who operate CT servers are a logical place for deployment; they already have (virtually) all certificates [73] and they are already subject to audits.

D. Client-Side Operation

The client-side component of CRLite is responsible for downloading the filter, updating it daily, and performing revocation checks for observed certificates against the filter.

False Positives. CRLite uses filter cascades which *do not suffer from false positives* yet remain compact. This addresses one of the primary concerns that drove the design of CRLSet [40] and the reluctance to incorporate Bloom filters in Chrome [39], [42].

However, there are two corner cases that require more attention. First, certificates created and deployed between updates to the filter cascade may result in a false positive (if they are valid) or a false negative (if they are created and revoked in this small window). This can be easily remedied with the timestamps already present in X.509 certificates: the NotBefore date in a certificate denotes the day and time at which clients should consider the certificate valid. So long as CAs set this correctly, then a client could know not to apply filter cascade, and to instead fall back on traditional methods (or request a delta update). Although falling back to CRL/OCSP may seem to obviate the benefits of CRLite, we note that only 0.005% of certificates in our dataset are created in any given 24-hour window.

Second, enterprise clients are sometimes configured with *private* root certificates that are used to issue internal certificates. CRLite's client-side component only checks a leaf certificate in the filter cascade if the certificate's chain terminates in a root used by CRLite. This behavior is appropriate, as when constructing the filter cascade, CRLite would not have considered an internal certificate to be in U (since it would be invalid from CRLite's perspective).

E. Summary

At this point, CRLite fulfills three of the goals in § IV-A:

- CRLite requires no active participation from CAs or websites, and no modification to certificates.
- CRLite operates as a browser plugin, thereby requiring only minor, incremental modifications to clients.
- In CRLite, almost all revocation checks are local, which preserves users' privacy.

With regard to the final two goals: in § V, we show how CRLite is resilient to attacks that are the root causes for poor revocation checking today [49]. Similarly, in § VII, we evaluate CRLite and show that it offers dramatically lower latency revocation checks than online mechanisms, while consuming bandwidth that is comparable to today's best schemes.

V. Security Analysis

We analyze CRLite's security against various attacks.

MitM. CRLite signs and timestamps all files that are made available to clients, which prevents MitM attackers from serving falsified or stale filters or delta updates to clients. We assume that CRLite's public key is securely distributed to clients through out-of-band mechanisms, such as via browser extension repositories and app stores.

Forcing Fail-open. Consider an alternative form of MitM attack wherein a user joins an adversarial network, such as at a coffee shop with a shared access point. An attacker can perform ARP spoofing or DNS injection to force a client's traditional revocation checks (*e.g.*, OCSP) to fail to connect. In this setting, today's browsers will *fail-open*, *i.e.*, they will assume that certificates are not revoked. In fact, it is because of this perceived necessity to fail-open that has led to very low rates of revocation checking in modern browsers [41], [49].

The primary benefit of CRLite from the client's perspective is that it can obtain and cache the day's revocations while in a safe network, such as at home or work. Later, in an adversarial network, an attacker's attempts to block access to OCSP and CRL servers has no effect, since the revocation information is already resident on the user's machine. In other words, with CRLite, clients can adopt a *fail-closed* security posture.

The sole exception to this scenario occurs if there is a certificate that is issued, compromised, **and** revoked within 24-hours after the daily filter is produced. In this case, CRLite clients will fall-back to traditional online revocation checks, because the certificate is newborn. However, the preconditions for mounting this attack (identifying a newborn cert, compromising it, and mounting a MiTM attack within 24-hours) make it challenging to execute.

DoS. In practice, CRLite filters should be hosted on DoS-resilient infrastructure, like a CDN. This makes it extremely difficult for an attacker to prevent clients from updating their filters by DoS-ing the hosting provider.

However, assume that a powerful attacker could block access to the filter hosting provider by DoSing it. In this case, all clients' filters will slowly become stale. OCSP and CRL servers (and HTTPS websites themselves) are also susceptible to such attacks; CRLite is at no greater risk. In fact, CRLite offers stronger security than traditional revocation mechanisms because of clients' ability to cache all historical revocations. As a result, clients would only miss the delta updates during the time of the attack; because CRLite clients update roughly once per day, the DoS attack would have to be prolonged to have profound impact (by comparison, the recent massive attacks on the root DNS servers lasted less than two hours [53]).

Backdated Certificate. It is possible for a CA to issue a certificate c on day d but not release it publicly until day d', where $d' \gg d$. If a CRLite client observes c on day d', it will erroneously believe that c should be present in the filter, since its `Not Before` date is in the past. However, since c was unknown to the aggregator at the time of filter construction, c could generate a false positive against the filter. Additionally, if c were revoked prior to d', it would generate a false negative. In essence, this scenario violates our assumption that the universe of certificates U known to the aggregator is complete.

In practice, this eventuality is unlikely to happen to today, and will be *impossible* in the future, thanks to CT. As we mention in § II-A, Chrome will require *all* new certificates to contain SCT records [66] signed by a well-known CT log as of October 2017. Crucially, to obtain an SCT for a new certificate, the issuing CA must submit it to a CT log and then wait 24 hours for the log to incorporate the new certificate [45]. In effect, the requirement that certificates be present in CT logs for 24 hours before browsers will validate them precludes the backdated certificate scenario outlined above. VanderSloot *et al.* have shown that 90.5% of all known valid certificates are already present in CT logs [73], and Chrome's validation requirements ensure that this will reach 100% soon.

Rogue Aggregator. In theory, a buggy or malicious CRLite aggregator could produce filters that omit revocations or falsely claim that valid certificates are revoked. To prevent this, CRLite produces signed audit logs that allow distrusting third-parties to recreate the filters from scratch, and compare them to the filters produced by the server. Auditing allows an investigator to detect the following types of rogue behaviors:

- **Omit a CRL or OCSP response.** All known, valid certificates must be covered by a CRL or OCSP response in the audit log.
- **Modify a certificate, CRL, or OCSP response.** All three objects are signed by their respective CAs, which can be independently validated by clients.
- **Equivocate.** Filters and their associated audit logs are timestamped and signed by the aggregator. If two clients receive filters with the same timestamp that are different, then one of the filters must be incorrect (which one can be determined using the audit log).

The only portion of the audit log that requires external information to verify is the list of valid certificates. In theory, a rogue aggregator could pretend that a target revoked certificate does not exist and not include its revocation information in the log.[9] This would lead CRLite clients to believe that the target certificate is not revoked. Fortunately, CRLite uses public information to construct the list of valid certificates, meaning that auditors can also collect this data and independently verify the set of valid certificates present in an audit log. We expect the process of verifying the list of valid certificates to become easier over time as more CAs adopt Certificate Transparency.

In practice, we envision that most CRLite clients will not audit filters since it is an expensive process (the auditor must download the audit log, obtain the universe of valid certificates, and then perform computations that take 10 minutes or more on a 32-core server). Instead, the goal of the audit log is to allow researchers, CAs, and companies (*e.g.*, Google and Mozilla) to act as a check against misbehaving or malicious CRLite servers. CRLite's audit log is a significant improvement over existing schemes like CRLSet and OneCRL that are not auditable.

VI. Implementation

We have developed a prototype implementation of CRLite that produces new filters, delta updates, and audit logs once per day on a 12-node Apache Spark cluster. Raw certificate

[9] Note that this omission attack is only possible if the target certificate is OCSP-only or uses a unique CRL. If other certificates reference the target's CRL, then it will be included in the log.

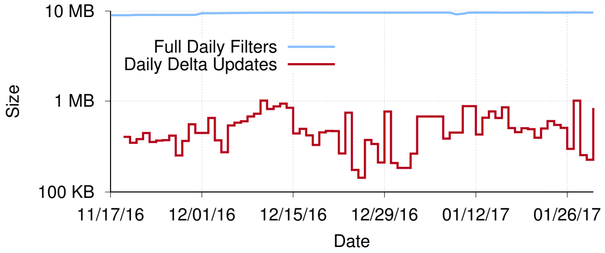


Fig. 5: Sizes of filter cascades and delta updates based on all valid certificates and revocations collected between November 17, 2016 and January 30, 2017.

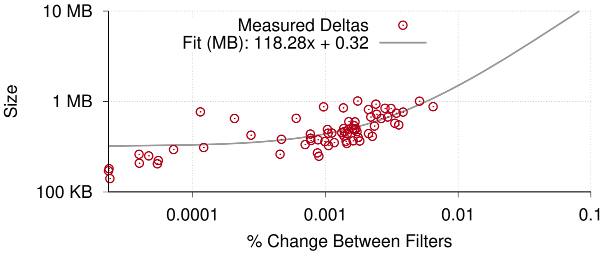


Fig. 6: Delta update sizes as a function of percentage change in the certificate universe (*i.e.*, the sum of changes in R and S), and line of best-fit.

processing, CRL crawling, and filter creation take less than 4 hours to perform. Making ~12M requests to OCSP responders takes ~11 hours. Since performing HTTP requests requires no special infrastructure and our data processing hardware is relatively modest, the barrier of entry to deploying a CRLite aggregator is quite low.

We use the Murmerhash3 function in our implementation because it is designed for speed. We evaluated other hash functions (*e.g.*, the DOM Webcrypto SHA functions) but none were faster in practice.

To demonstrate that CRLite is feasible on the client side, we have implemented a proof-of-concept Firefox extension. After installation, the extension downloads the latest daily filter, and keeps it up-to-date by downloading and applying delta updates each day. Furthermore, the extension inspects each leaf and intermediate certificate presented to the browser, and kills the corresponding TLS connection if the certificate hits the filter cascade. Our prototype does not attempt to alter Firefox's default OCSP checking behavior, although this could be achieved through deeper integration with the browser.

As a performance optimization, our Firefox extension includes a Least Recently Used (LRU) cache of recent lookups in the filter cascade. In practice, users tend to visit popular websites repeatedly, thus subsequent visits to these sites only require an LRU lookup, as opposed to a (relatively slower) filter cascade lookup. The LRU is cleared each time the filter cascade is updated.

We chose to implement our prototype for Firefox because it offers low-level APIs for inspecting TLS certificate chains. Unfortunately, these APIs are deprecated [57]: Chrome, Edge, and Firefox are adopting the WebExtensions API, which does not have TLS APIs. In the future, we envision that CRLite should be integrated directly into browsers and cryptographic libraries, similar to how CRLSet and OneCRL are integrated into Chrome and Firefox. Fortunately, this integration can occur incrementally, *i.e.*, there is no requirement that all clients adopt CRLite for the system to function or provide benefits.

Our implementation demonstrates that it is feasible to deploy CRLite in today's web without requiring buy-in from CAs or websites. In § VIII, we describe a slightly altered design that shows that CA participation could remove the necessity for audits. However, our primary focus is on what can be achieved immediately, so in the following sections, we evaluate the security and the performance of CRLite as described thus far.

VII. Evaluation

In this section, we evaluate our CRLite prototype. *First*, we parameterize our filter cascade by choosing the revoked and non-revoked certificate capacity, as well as the false positive rates. *Second*, we investigate the size of filters and delta updates for CRLite using empirical and simulated data. *Third*, we benchmark our client-side implementation of CRLite for Firefox. *Fourth*, we use data-driven simulations to compare the overhead of CRL, OSCP, and CRLite. Finally, we compare CRLite to CRLSet and OneCRL.

A. Bloom Filter Parameters

We begin by selecting the parameters for CRLite's filter cascade. For this analysis, we leverage data on all valid certificates and all revocations that we gathered between November 17, 2016 and January 30, 2017.

To maintain stable parameters for the filter cascade and minimize disruption for CRLite clients, we choose $|R|$ and $|S|$ at the beginning of each month based on (1) the number of revoked and non-revoked certificates at that time, and (2) the rate of change in the certificate universe. For example, on January 1, 2017 there were 12M revoked and 30M non-revoked certificates. To leave additional room for both sets to grow, we set $r = |R| =$ 13M and $s = |S| =$ 35M. Assuming that the sets of certificates do not outgrow r and s during January, CRLite clients will only need to download delta updates; otherwise the filter cascade must be reparameterized, and clients will need to download a new complete filter.

After choosing the capacities, we must calculate the optimal false positive rates p_1 and p. Given r and s, we empirically locate the value of p (and $p_1 = r\sqrt{p}/s$, as given in § III-C) that minimize the size of the filter cascade. For example, when $r =$ 13M and $s =$ 35M, $p_1 = 0.2652$ and $p = 0.5099$.

Figure 5 shows the sizes of all the daily filters we generate between November 17, 2016 and January 30, 2017. During this time interval, the number of revoked certificates remains very stable ~12M, while the number of valid non-revoked certificates grows from 24M to 32M. Thus, in keeping with the strategy outlined above, we reparameterize the filter cascade on November 17 ($r = 13$M, $s = 30$M), December 1 ($r = 13$M, $s = 35$M), and January 1 ($r = 13$M, $s = 35$M). Despite the growth in overall certificates, the sizes of the daily filters remains very stable, varying between 9 MB and 10 MB.

B. Delta Updates

Next, we examine the size of delta updates to the CRLite filter. Recall that the CRLite server produces a new filter cascade each day, and then computes a delta update between the new filter and the previous day's filter. This delta update contains a compressed bitwise diff between each corresponding level of the filter cascade.

Figure 5 shows the sizes of all the delta updates we generate between November 17, 2016 and January 30, 2017. Their sizes vary between 176 KB and 1 MB, with an average size of 580 KB. These daily updates are quite modest, and are even feasible for mobile users with restrictive data caps (*e.g.*, 2 GB of bandwidth per month). In § VII-D, we compare the sizes of delta updates to the cost of performing traditional revocation checks using CRL and OCSP.

Figure 6 further examines the sizes of delta updates as a function of percentage change in the certificate universe day-to-day. The points show the relationship based on our measured data. However, since we do not observe days with large changes in the universe (*e.g.*, due to an event like Heartbleed), we show a line of best-fit based on linear regression.

As expected, Figure 6 shows that delta update sizes grow as the change rate increases. Under normal circumstances, delta updates are an order of magnitude smaller than a full filter. In the worst case scenario, when the certificate universe changes by 0.1% in 24 hours, delta updates can grow up to 10 MB. In this case, clients are better served by simply downloading a fresh filter. However, events of this size are extremely rare: for example, CloudFlare revoked 19,384 certificates in one day immediately after HeartBleed [76].

C. Microbenchmarks

We now analyze the overhead of our Firefox implementation of CRLite. Using Firefox's built-in profiler, we observe that CRLite uses 11.9 MB of memory for the filter cascade parameters given above. Most of the memory usage comes from the JavaScript ArrayBuffer objects containing the binary-encoded filters, which demonstrates that CRLite's memory requirements almost perfectly mirror the size of the filter.

We also measured the CPU overhead of our CRLite extension. As we describe in the next section, we can simulate a normal user's browsing behavior by browsing websites from the Alexa Top-1M using a Zipf distribution. After visiting 1K sites we observed that, on average, it took CRLite 10 milliseconds to verify a chain of certificates. Note that this includes the time to parse the ASN.1 certificates (since Firefox's API only provides unparsed certificates) and check them against the filters. However, a simple 300-element LRU cache in the extension reduces the average lookup time to 6 milliseconds. In practice, if CRLite was integrated directly into the browser both overheads would be significantly lower, since the certificate would already be parsed, and the code would be native (*i.e.*, not JavaScript). However, even as an extension, the delay induced by CRLite is low in absolute terms, and two orders of magnitude lower than traditional CRL and OCSP checks (which we examine in the next section).

D. Comparison with CRLs and OCSP

In this section, we compare CRLite with CRLs and OCSP. We ask the questions: *how much delay would a typical user experience per day while browsing the web under each revocation checking strategy?* and *how many bytes would a typical user download per day while browsing under each strategy?* We assume that users are using an ideal browser that performs revocation checks on all observed certificates.

Experimental Setup. To answer these questions, we conduct data-driven simulations that mimic a typical user's browsing behavior. We adopt the browsing model from Burklen *et al.* [9], which gives empirically-validated statistical distributions and associated parameters that describe users' browsing behavior. Specifically, our simulated users visit domains from the Alexa Top-1M based on a Zipf distribution, view pages following a Pareto distribution (exponent = 1.3, range = 10–310 seconds), and leave domains following a Pareto distribution (exponent = 3.0). When a simulated user visits a domain that uses HTTPS, their browser checks all certificates for revocations using one of the following three strategies:

- **OCSP and CRL:** This browser checks for revocations using OCSP or CRL, but prioritizes OCSP if it is available. This strategy approximates how modern browsers actually behave [49].
- **CRL Only:** This browser only checks for revocations using CRL. The rationale behind this strategy is that it (a) prioritizes privacy over network performance by eschewing OCSP, and (b) potentially ends up avoiding future CRL requests, as CRLs contain information for a large number of certificates.
- **CRLite:** This browser uses CRLite. We assume the browser already has the filter downloaded, and only pays the cost of delta updates each day.

When our simulated user visits a domain, the simulator actually contacts that domain, performs a TLS handshake, and validates the certificates. In the case of CRLs and OCSP, the simulator contacts the relevant CAs and records the time to complete the requests, as well as the size of the CRL/OCSP responses. Additionally, the simulator caches CRLs and OCSP responses for their validity period, and uses cached information to fulfill future requests whenever possible. In the case of CRLite, the simulated user either incurs a 10 millisecond delay to check the chain in the filter cascade, or a 6 millisecond delay if the certificate is in the LRU cache (see § VII-C).

To make our simulator tractable, it assumes a simplified model of the web where pages do not embed HTTPS content from third parties. Thus, our results should be viewed as a conservative lower bound on the amount of traffic and delay that users will incur as they browse.

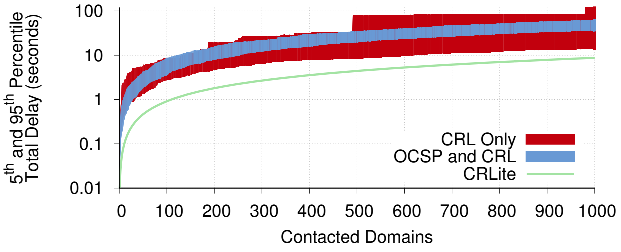


Fig. 7: 5^{th} and 95^{th} percentile cumulative delay experienced by clients using different revocation checking strategies.

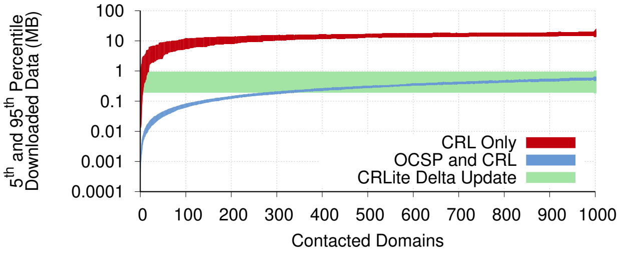


Fig. 8: 5^{th} and 95^{th} percentile cumulative bytes downloaded by clients using different revocation checking strategies.

Results. We conducted 100 runs of our simulator for each revocation checking strategy, during which each simulated user visited 1,000 domains drawn from the Alexa Top-1M.[10]

We first examine total network delay in Figure 7. We observe that the browsers that check CRLs and OCSP accrue an order of magnitude more delay than the CRLite user, which makes sense given that each CRL/OCSP check involves network requests. Although network requests have two orders of magnitude more latency than CRLite (recall that each filter check takes <10 milliseconds), caching eliminates the need for many network requests. Interestingly, we observe that some of the CRL users experience lower delay than the OCSP users; this occurs when a user gets lucky and downloads a CRL that happens to cover many of the domains they browse in the future. However, we also see that the CRL users can get extremely unlucky and experience very high delay if they download several large, slow CRLs. Overall, CRLite is the clear winner, and avoids the high latency penalty that makes browser vendors eschew online revocation checking [40].

Figure 8 presents the 5^{th} and 95^{th} percentile total downloaded bytes for the simulated users as they browse. For the CRLite user, the amount of downloaded bytes is independent from the number of domains visited, since they only download the delta update, and they never encounter newborn certificates that would trigger on online revocation check. Conversely, the OCSP and CRL users accumulate more downloaded bytes as they contact more CAs, although the distributions eventually level-off due to caching effects, *i.e.*, they are highly likely to visit popular sites repeatedly.

Figure 8 demonstrates how much browsing is necessary to amortize the cost of downloading the CRLite delta update as compared to other revocation checking strategies. A browser that prioritizes OCSP downloads 1.0 MB of data after visiting ∼1000 domains. In contrast, the CRL-only browser hits this benchmark after visiting only ∼20 domains. Thus, from the standpoint of minimizing network traffic, OCSP is arguably the most efficient strategy. However, among the two privacy-preserving strategies, CRLite is the clear winner.

E. Comparison with CRLSet and OneCRL

Finally, we briefly compare CRLite to CRLSet and OneCRL. By construction, CRLite provides much greater coverage of revocations: as of January 30, 2017, CRLite contains 12.7M revocations, versus 14,436 and 357 in CRLSet and OneCRL. Similarly, because CRLite uses a filter cascade, it is able to achieve much greater information density. We observe that CRLite uses 6.6 bits per revocation, while CRLSet uses 110 bits per revocation, and OneCRL uses 1,928 bits per revocation (since it uses XML-formatted text). We can extrapolate that CRLSet and OneCRL would require 166 MB and 2.9 GB, respectively, to store 12.7M revocations.

VIII. CRLite without Auditing

We have presented a design of CRLite that decreases the amount of data that clients need to download to stay up-to-date on revocation information, without requiring any buy-in or active participation from CAs. The benefit of not requiring CA participation is that CRLite is deployable *today*. However, the tradeoff is that it centralizes trust in the aggregator, thereby necessitating third-party audits.

In this section we consider alternative CRLite designs that involve active participation from CAs. We show that, with a small amount of CA work, it is possible to achieve nearly equal-sized filters *without centralizing trust*.

Assumptions. We assume that there is a set of N CAs participating in CRLite, and that participating clients know this set and each of the CAs' public keys. We further assume that there is an entity who aggregates the CAs' revocation information; as a result, we assume this aggregator knows the N CAs, but we do not assume that it is trusted either by CAs or clients. (In practice, the CAs themselves could take turns serving this role.) Finally, we assume a well-known global set of CRLite parameters, including the number of levels in the filter cascade, the size of each level, and the set of hash functions. These parameters can be adjusted over time as the certificate ecosystem evolves.

An Interactive Protocol. We present a protocol wherein CAs directly interact with the aggregator. Due to lack of space, we elide details of how to format the messages, which would be critical (though not complicated) in practice.

[10] Note that the sequence of domains for a given simulated user will almost certainly contain duplicate domains.

The aggregator performs most of its tasks as normal: it collects all certificates and revocations (participating CAs could facilitate this, but it is not necessary), and then it creates the filter cascade and delta update for the day. Next, instead of creating an audit log, the aggregator sends the latest filter cascade $\mathsf{F}_{\mathsf{curr}}$, the previous day's $\mathsf{F}_{\mathsf{prev}}$, and the latest delta update Δ to all of the included CAs.

Each CA C_i verifies two things: *First*, that the delta update is correct, *i.e.*, $\mathsf{F}_{\mathsf{curr}} = \mathsf{F}_{\mathsf{prev}} \oplus \Delta$. *Second*, and most important, C_i verifies that its certificates are properly accounted for. If C_i has revoked certificates R_i and non-revoked certificates S_i, then C_i verifies that lookups for any $r \in R_i$ in $\mathsf{F}_{\mathsf{curr}}$ return true, and lookups for any $s \in S_i$ return false. If any of these fail, then C_i aborts. Otherwise, C_i generates signature σ_i of $\langle \mathsf{F}_{\mathsf{curr}}, \mathsf{F}_{\mathsf{prev}}, \Delta \rangle$, and sends it to the aggregator.

The aggregator then distributes to all clients $\mathsf{F}_{\mathsf{curr}}$ or Δ (as needed), as well as all σ_i's that it received. For each CA that provided a valid signature, clients need not perform audits: C_i's signature attests that the filter cascade is complete with respect to C_i. In the event that a CA C_{fail} does not provide valid signatures (*e.g.*, due to failure), clients would still have to perform audits as described in § IV to verify the filter's completeness. However, this does not prevent clients from using the filter for all certificates signed by $C_i \neq C_{\mathsf{fail}}$.

This protocol results in nearly as few bytes sent as the standard CRLite. In particular, it comprises the same-sized filter cascade; what differs is the number of signatures. In a naïve construction, we result in N signatures, one from each CA. However, novel *multi-signature* schemes can be applied to reduce this: such a scheme allows N participants to separately sign a common message M and for a separate party to compute a multi-signature that is the size of a single signature. For instance, Boldyreva demonstrated an elegant multi-signature scheme based on bilinear maps [7]. The common message in this application is $\langle \mathsf{F}_{\mathsf{curr}}, \mathsf{F}_{\mathsf{prev}}, \Delta \rangle$. Thus, the aggregator could combine the signatures to obtain aggregate signature $\sigma^{\star}$, thereby reducing the number of signatures from N to 1. Additionally, the aggregator would need to send a list of CAs that failed to contribute signatures (normally none should fail).

In this scheme, the aggregator need not be trusted to deliver this information. So long as the client knows the set of CAs that are supposed to be included in the CRLite (which we assumed above, or could be explicitly included with the filter), it will be able to detect any modifications to the filter cascade or delta updates. In other words, clients must verify aggregate signatures, but they need not perform costly audits.

Related Constructions. Two recent schemes also propose aggregating signatures. CoSi [67] assigns a witness set to observe and attest to the messages from some "authority" (*e.g.*, a CA), and scalably aggregates their attestations to clients. Our construction differs in that the original data authorities (CAs) do the "witnessing," and rather than detect equivocation [47], each CA verifies its own subset of the filter cascade. As a result, misbehavior in our protocol is detectable immediately. Another recent system, RevCast [65], also proposes using compact multi-signatures to represent certificate revocation. Whereas RevCast simply concatenates their lists of revoked certificates, we perform the space-saving operation of inserting them into a filter cascade. This is what requires us to use an interactive protocol, whereas RevCast is able to operate non-interactively. We speculate that a non-interactive variant of CRLite may be possible with novel cryptographic mechanisms (fully homomorphic signatures [32] appear particularly promising); this is an interesting area of future work.

IX. Conclusion

Software that leverages TLS is currently caught between a rock and a hard place with respect to certificate revocation. On one hand, revocation checking is critical for security: without it, users are vulnerable to MitM and phishing attacks that leverage compromised certificates. On the other hand, existing online mechanisms for revocation checking like CRLs and OCSP are slow, leak browsing information, and fail to offer strong security in practice (since an attacker can block access to the revocation servers, causing existing clients to fail-open). Stopgap solutions like CRLSet and OneCRL increase security, but do not obviate the need for online revocation checks or fundamentally solve the fail-open issue.

Thanks to efforts such as Certificate Transparency [45] and various active scanning projects [21], [60], the possibility of having virtually all HTTPS certificates is a reality. CRLite leverages this bounty by (a) having a centralized aggregator collect revocation status for *every* known certificate and (b) representing them compactly using a filter cascade data structure. Surprisingly, the size of this representation is quite reasonable (10 MB for the entire structure, plus 580 KB per day for updates), making it practical to distribute to all clients. CRLite also includes procedures for publicly auditing aggregators to ensure they do not leave out or extraneously put in any revocations (we analyze various attacks in §V).

We develop a prototype implementation of CRLite, and show that it offers high performance and low overhead. Most importantly, CRLite enables clients to adopt a fail-closed security posture because the filter is guaranteed to not contain false positives or negatives (for any certificates created or revoked >24 hours ago). CRLite requires no work on the part of CAs, no changes to certificates, no modifications to the TLS protocol, and is suitable for resource-constrained clients.

We believe that CRLite shows that it is now worthwhile to reinvestigate the trade-offs of complete and universal delivery of revocation information. As a service to the community, we make our code available at https://www.securepki.org.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Rob Johnson, for their helpful comments. This research was supported by NSF grants CNS-1409249, CNS-1421444, CNS-1563320, and CNS-1564143, and by the NSA as part of a Science of Security lablet.

References

[1] W. Aiello, S. Lodha, and R. Ostrovsky. Fast digital identity revocation. In *International Cryptology Conference*, August 1998.

[2] C. Arthur. Diginotar SSL Certificate Hack Amounts To Cyberwar, Says Expert, September 2011. http://www.theguardian.com/technology/2011/sep/05/diginotar-certificate-hack-cyberwar.

[3] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski. Design, Analysis, and Implementation of ARPKI: an Attack-Resilient Public-Key Infrastructure. *IEEE Transactions on Dependable and Secure Computing*, (99), August 2016.

[4] A. Bates, J. Pletcher, T. Nichols, B. Hollembaek, and K. R. Butler. Forced Perspectives: Evaluating An SSL Trust Enhancement At Scale. In *ACM Internet Measurement Conference*, November 2014.

[5] M. A. Bender, M. Farach-Colton, R. Johnson, R. Kraner, B. C. Kuszmaul, D. Medjedovic, P. Montes, P. Shetty, R. P. Spillane, and E. Zadok. Don't Thrash: How to Cache Your Hash on Flash. In *Conference on Very Large Data Bases*, July 2012.

[6] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, July 1970.

[7] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Helman-Group Signature Scheme. In *Public Key Cryptography (PKC)*, January 2003.

[8] 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*, May 2014.

[9] S. Burklen, P. J. Marron, S. Fritsch, and K. Rothermel. User centric walk: An integrated approach for modeling the browsing behavior of users on the web. In *Annual Symposium on Simulation*, April 2005.

[10] CA/Browser Forum. Baseline Requirements: Certificate Policy for the Issuance and Management of Publicly-Trusted Certificates. Version 1.4.1, September 2016.

[11] F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem. In *ACM Conference on Computer and Communications Security*, October 2016.

[12] A. A. Chariton, E. Degkleri, P. Papadopoulos, P. Ilia, and E. P. Markatos. DCSP: Performant Certificate Revocation a DNS-based approach. In *European Workshop on System Security*, April 2016.

[13] M. Chase and S. Meiklejohn. Transparency Overlays and Applications. In *ACM Conference on Computer and Communications Security*, October 2016.

[14] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables. In *ACM-SIAM Symposium on Discrete Algorithms*, January 2004.

[15] T. Chung, Y. Liu, D. Choffnes, D. Levin, B. Maggs, A. Mislove, and C. Wilson. Measuring and Applying Invalid SSL Certificates: The Silent Majority. In *ACM Internet Measurement Conference*, November 2016.

[16] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate And Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.

[17] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno. Cinderella: Turning Shabby X.509 Certificates into Elegant Anonymous Credentials with the Magic of Verifiable Computation. In *IEEE Symposium on Security and Privacy*, May 2016.

[18] P. Ducklin. Halfway there! Firefox users now visit over 50% of pages via HTTPS, October 2016. https://nakedsecurity.sophos.com/2016/10/18/halfway-there-firefox-users-now-visit-over-50-of-pages-via-https/.

[19] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson. The Matter Of Heartbleed. In *ACM Internet Measurement Conference*, November 2014.

[20] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis Of The HTTPS Certificate Ecosystem. In *ACM Internet Measurement Conference*, October 2013.

[21] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide Scanning and Its Security Applications. In *USENIX Security Symposium*, August 2013.

[22] D. Eastlake. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066, Jan 2011.

[23] EFF SSL Observatory. https://www.eff.org/observatory.

[24] B. Eisenberg. On the expectation of the maximum of IID geometric random variables. *Statistics & Probability Letters*, 78(2):135–143, 2008.

[25] F. F. Elwailly, C. Gentry, and Z. Ramzan. QuasiModo: Efficient Certificate Validation and Revocation. In *International Workshop on Theory and Practice in Public Key Cryptography*, March 2004.

[26] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher. Cuckoo Filter: Practically Better Than Bloom. In *ACM International Conference on Emerging Networking Experiments and Technologies*, December 2014.

[27] K. Finley. The Average Webpage Is Now the Size of the Original Doom, April 2016. https://www.wired.com/2016/04/average-webpage-now-size-original-doom/.

[28] I. Gassko, P. S. Gemmell, and P. MacKenzie. Efficient and Fresh Certification. In *International Workshop on Practice and Theory in Public Key Cryptosystems*, January 2000.

[29] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The Most Dangerous Code In The World: Validating SSL Certificates In Non-browser Software. In *ACM Conference on Computer and Communications Security*, October 2012.

[30] A. Goel and P. Gupta. Small Subset Queries and Bloom Filters Using Ternary Associative Memories, with Applications. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, June 2010.

[31] M. Goodwin. Revoking Intermediate Certificates: Introducing OneCRL. Mozilla Security Blog, March 2015. http://mzl.la/1zLFp7M.

[32] S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled Fully Homomorphic Signatures from Standard Lattices. In *ACM Symposium on Theory of Computing (STOC)*, June 2015.

[33] P. Hallam-Baker. X.509v3 Extension: OCSP Stapling Required, October 2012. https://tools.ietf.org/html/draft-hallambaker-muststaple-00.

[34] P. Hallam-Baker. X.509v3 Transport Layer Security (TLS) Feature Extension. RFC 7633, October 2015.

[35] P. Hoffman and J. Schlyter. The DNS-based Authentication Of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698, August 2012.

[36] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL Landscape – a Thorough Analysis Of The X.509 PKI Using Active And Passive Measurements. In *ACM Internet Measurement Conference*, November 2011.

[37] 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*, May 2013.

[38] P. C. Kocher. On Certificate Revocation and Validation. In *International Conference on Financial Cryptography*, February 1998.

[39] A. Langley. Smaller Than Bloom Filters, April 2011. https://www.imperialviolet.org/2011/04/29/filters.html.

[40] A. Langley. Revocation Checking And Chrome's CRL, February 2012. https://www.imperialviolet.org/2012/02/05/crlsets.html.

[41] A. Langley. No, Don't Enable Revocation Checking, April 2014. https://www.imperialviolet.org/2014/04/19/revchecking.html.

[42] A. Langley. Revocation Still Doesn't Work, April 2014. https://www.imperialviolet.org/2014/04/29/revocationagain.html.

[43] B. Laurie. Improving the Security of EV Certificates, May 2015. https://www.certificate-transparency.org/ev-ct-plan/EVCTPlanMay2015edition.pdf.

[44] B. Laurie and E. Kasper. Revocation Transparency. GitHub, June 2016. https://github.com/google/trillian/blob/master/docs/RevocationTransparency.pdf.

[45] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962, June 2013.

[46] Let's Encrypt. https://letsencrypt.org.

[47] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. TrInc: Small Trusted Hardware for Large Distributed Systems. In *Symposium on Networked System Design and Implementation*, April 2009.

[48] 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*, May 2014.

[49] Y. Liu, 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*, October 2015.

[50] P. D. McDaniel and A. D. Rubin. A Response to "Can We Eliminate Certificate Revocation Lists?". *International Conference on Financial Cryptography*, February 2000.

[51] S. Micali. Efficient Certificate Revocation. In *RSA Data Security Conference*, March 1997.

[52] S. Micali. Scalable Certificate Validation And Simplified PKI Management. In *PKI Research Workshop*, April 2002.

[53] G. C. M. Moura, R. de O. Schmidt, J. Heidemann, W. de Vries, M. Mueller, L. Wei, and C. Hesselman. Anycast vs. DDoS: Evaluating the November 2015 Root DNS Event. In *ACM Internet Measurement Conference*, 2016.

[54] PKI:CT. Mozilla Wiki, December 2014. https://wiki.mozilla.org/PKI:CT.

[55] M. Naor and K. Nissim. Certificate Revocation and Certificate Update. In *USENIX Security Symposium*, January 1998.

[56] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The Cost of the "S" in HTTPS. In *ACM International on Conference on Emerging Networking Experiments and Technologies*, December 2014.

[57] K. Needham. The Future of Developing Firefox Add-ons, August 2015. https://blog.mozilla.org/addons/2015/08/21/the-future-of-developing-firefox-add-ons/.

[58] H. Perl, S. Fahl, and M. Smith. You Won't Be Needing These Any More: On Removing Unused Certificates From Trust Stores. In *Financial Cryptography and Data Security*, March 2014.

[59] F. Putze, P. Sanders, and J. Singler. Cache-, Hash-, and Space-efficient Bloom Filters. *J. Exp. Algorithmics*, 14:4:4.4–4:4.18, Jan. 2010.

[60] Rapid7 SSL Certificate Scans. https://scans.io/study/sonar.ssl.

[61] R. L. Rivest. Can We Eliminate Certificate Revocation Lists? *International Conference on Financial Cryptography*, 1998.

[62] M. D. Ryan. Enhanced Certificate Transparency and End-to-end Encrypted Mail. In *Network and Distributed System Security Symposium*, February 2014.

[63] Safe Browsing. The Chromium Project. https://www.chromium.org/developers/design-documents/safebrowsing.

[64] K. Salikhov, G. Sacomoto, and G. Kucherov. Using Cascading Bloom Filters to Improve the Memory Usage for de Brujin Graphs. In *Workshop on Algorithms in Bioinformatics*, September 2013.

[65] A. Schulman, D. Levin, and N. Spring. RevCast: Fast, Private Certificate Revocation Over FM Radio. In *ACM Conference on Computer and Communications Security*, November 2014.

[66] R. Sleevi. Announcement: Requiring Certificate Transparency in 2017. Certificate Transparency Policy, October 2016. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/78N3SMcqUGw.

[67] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *IEEE Symposium on Security and Privacy*, May 2016.

[68] P. Szalachowski, L. Chuat, T. Lee, and A. Perrig. RITM: Revocation in the Middle. In *IEEE International Conference on Distributed Computing Systems*, June 2016.

[69] 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*, March 2016.

[70] P. Szalachowski, S. Matsumoto, and A. Perrig. Policert: Secure and flexible tls certificate management. In *ACM Conference on Computer and Communications Security*, November 2014.

[71] University Of Michigan HTTPS Ecosystem Scans. https://scans.io/study/umich-https.

[72] N. Vallina-Rodriguez, J. Amann, C. Kreibich, N. Weaver, and V. Paxson. A Tangled Mass: The Android Root Certificate Stores. In *International Conference on Emerging Networking Experiments and Technologies*, December 2014.

[73] B. VanderSloot, J. Amann, M. Bernhard, Z. Durumeric, M. Bailey, and J. A. Halderman. Towards a Complete View of the Certificate Ecosystem. In *ACM Internet Measurement Conference*, November 2016.

[74] Z. Wang. CasAB: Building Precise Bitmap Indices via Cascaded Bloom Filters. In *IEEE Conference on Internet Computing for Science and Engineering*, December 2009.

[75] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When Private Keys Are Public: Results From The 2008 Debian OpenSSL Vulnerability. In *ACM Internet Measurement Conference*, November 2009.

[76] L. Zhang, D. Choffnes, T. Dumitras, D. Levin, A. Mislove, A. Schulman, and C. Wilson. Analysis Of SSL Certificate Reissues And Revocations In The Wake Of Heartbleed. In *ACM Internet Measurement Conference*, November 2014.

[77] L. Zhou, F. B. Schneider, and R. Van Renesse. COCA: A Secure Distributed Online Certification Authority. *ACM Transactions on Computer Systems*, 20(4):329–368, November 2002.

Appendix A
Analysis of Filter Cascade Levels

Here, we extend our analysis from § III-C to rigorously determine the expected number of hashes applied to each certificate and the expected number of levels in the filter cascade. Note that very few certificates survive to the last level of the filter cascade.

We model the removal of certificates from consideration as a coin-flipping process. Recall that we seek to store some set R, that all queries come from some "universe" U, and that $R \cup S = U$. For each set element, whether originally in R or S, there is a coin flipping process that proceeds in rounds. The coin for an element is flipped whenever it is among those that are *not* being inserted into the Bloom filter at a particular level. These are the levels at which a false positive may occur for the element. Hence, for any element, a coin-flipping round occurs every other level. With probability $1-p_1$ or $1-p$, the outcome is "heads" indicating that the element can be removed from consideration and no more flips are necessary. Otherwise the flip is "tails" and the process continues. Assuming that our hash functions choose which bits to set for different elements independently, every coin flip is independent.

Levels per Certificate. We begin by analyzing the expected number of levels at which an individual certificate is inserted or checked for a false positive. Let X_i be a random variable indicating the number of rounds until heads is flipped for the ith coin. For an element in R, it is easy to show that $E[X_i] = 1/(1-p)$. As an example, for $p = 1/2$, $E[X_i] = 2$. For an element in S, the formula is slightly more complicated because the probability of heads in the first round is $1-p_1$, and in later rounds $1-p$. Thus $E[X_i] = (1-p_1) + p_1(1 + 1/(1-p))$. This quantity is smaller than the number of rounds for an element in R, assuming $p_1 < p$. For an element in R, the number of levels, Y_i, at which the certificate is inserted or checked is twice the number of rounds, i.e., $Y_i = 2X_i$, so that $E[Y_i] = 2/(1-p)$. As an example, for $p = 1/2$, $E[Y_i] = 4$. For an element in S, the number of levels is given by $Y_i = 2X_i - 1$, hence $E[Y_i] = 2(1-p_1) + 2p_1(1 + 1/(1-p)) - 1$. Again, for $p_1 < p$, the expected number of levels is smaller for an element of S than for an element of R. Note that the expressions for $E[X_i]$ and $E[Y_i]$ are actually upper bounds, because an element of R need not be considered further if all of the elements of S have been removed from consideration, and vice versa.

Hashes per Certificate. At each level, multiple hash functions may be applied to a certificate. Let k_1 denote the number of hash functions used in the Bloom filter at level

1, and let k denote the number of hash functions used in each Bloom filter at all levels after 1. Let Z_i denote the number of hash functions applied to the ith certificate. For any element, whether in R or S, $Z_i = k_1 + k(Y_i - 1)$. Hence $E[Z_i] = k_1 + k(E[Y_i] - 1)$. As an example, for $p = 1/2$ and $k = 1$, for an element of R we have $E[Z_i] = k_1 + 3$. (For an element of S, $E[Z_i]$ is smaller.)

Levels in Filter Cascade. Finally, we would like to bound the expected number of levels in the entire filter cascade, i.e., the expected number of levels required until every element has flipped heads. Eisenberg [24] states a bound that we can immediately apply to the expected number of even-numbered levels needed before all elements of R are removed from consideration. Suppose that N coins are being flipped in rounds, where in each round, each coin has probability $1-p$ of flipping heads. As before, let X_i denote the number of rounds until the ith coin flips heads. We are interested in the maximum, over all coins, of the number of rounds needed to flip heads, i.e., in the random variable $X = \max(X_1, \ldots, X_N)$. The bound from [24] is

$$\frac{H_N}{\ln \frac{1}{p}} \leq E[X] \leq 1 + \frac{H_N}{\ln \frac{1}{p}}, \tag{3}$$

where H_N is the Nth harmonic number, i.e., $H_N = \sum_{i=1}^{N} 1/i$. For large N, H_N is very close to $\ln N + 0.577$. Let X_R denote the number of even-numbered levels needed. Then applying the bound from Inequality (3) with $N = r = |R|$, yields

$$E[X_R] \leq 1 + (\ln r + 0.577)/\ln(1/p).$$

For the elements of S a slightly messier analysis is required because at the first level these elements suffer false positives with probability p_1, and at all subsequent odd levels with probability p. But for the sake of the analysis, let us begin by assuming that the false positive probability (the probability of flipping tails) in every round is p. Then the bound from Inequality (3) can be applied with $N = s = |S|$. Note that in this coin-flipping process, the probability that an element survives the first i rounds, flipping tails i times, is p^i. Suppose now that we modify the coin-flipping process so that in the first round the probability of tails is p_1, in the next $\lfloor(\ln p_1)/(\ln p)\rfloor - 1$ rounds the probability of tails is 1, and in all subsequent rounds the probability of tails remains p. Here we assume that $p_1 < p$. Observe that after this modification to the process, the expected number of rounds can only decrease because the probability that any element survives i rounds, for any i, can only decrease. Hence, applying Inequality (3) with $N = s$, the expected number of rounds in the modified process is at most $1 + (\ln s + 0.577)/\ln(1/p)$.

Now we modify the process again by removing the rounds in which the false positive probability is 1. Removing these rounds does not affect what happens in any other rounds. The probability that there are no false positives in the first round (and thus that there is only one level) is $(1 - p_1)^s$. Hence, the expected number of rounds in this final process is reduced by $(1 - (1 - p_1)^s)(\lfloor(\ln p_1)/(\ln p)\rfloor - 1)$. Note that this final process is now identical to the process implemented in the odd levels of our filter cascade. Let X_S denote the number of odd-numbered levels needed to remove all elements of S from consideration. Then

$$E[X_S] \leq 1 + \frac{\ln s + 0.577}{\ln(1/p)} - (1 - (1 - p_1)^s)\left(\left\lfloor\frac{\ln p_1}{\ln p}\right\rfloor - 1\right).$$

Building the filter cascade ends when either all of the elements in R have been removed from consideration, or all of the elements in S have been removed from consideration, whichever comes first. Hence, the total number of levels, X_T, is $\min(2X_S - 1, 2X_R)$. Because the expectation of the minimum of two random variables is at most the minimum of the expectations, we have

$$E[X_T] \leq \min(2E[X_S] - 1, 2E[X_R]).$$

High Probability Bounds. It is also straightforward to prove a high probability bound. The probability that a coin is flipped tails i times in a row is p^i. Thus, the probability that there is any coin that has been flipped tails i times in a row is at most Np^i. If we want 99% confidence that at most i rounds are needed, we set $Np^i = 0.01$, and then solve for i.