Let's Revoke: Scalable Global Certificate Revocation

Trevor Smith, Luke Dickinson, Kent Seamons
Brigham Young University
tsmith@isrl.byu.edu, luke@isrl.byu.edu, seamons@cs.byu.edu

***Abstract*—Current revocation strategies have numerous issues that prevent their widespread adoption and use, including scalability, privacy, and new infrastructure requirements. Consequently, revocation is often ignored, leaving clients vulnerable to man-in-the-middle attacks. This paper presents Let's Revoke, a scalable global revocation strategy that addresses the concerns of current revocation checking. Let's Revoke introduces a new unique identifier to each certificate that serves as an index to a dynamically-sized bit vector containing revocation status information. The bit vector approach enables significantly more efficient revocation checking for both clients and certificate authorities. We compare Let's Revoke to existing revocation schemes and show that it requires less storage and network bandwidth than other systems, including those that cover only a fraction of the global certificate space. We further demonstrate through simulations that Let's Revoke scales linearly up to ten billion certificates, even during mass revocation events.**

I. Introduction

In the current web Public Key Infrastructure (PKI), Certificate Authorities (CAs) issue and sign the X.509 certificates that secure TLS connections. It is critical to identify any certificate that was erroneously issued or has had its key pair compromised. Once identified, the owner of the certificate requests the issuing CA to revoke or invalidate their certificate. Notice of this revocation needs disseminating to all clients that rely on the certificate. Otherwise, connections using the compromised certificate are vulnerable to man-in-the-middle (MitM) attacks until the certificate has expired, which can be up to 2 years later [1]. Without accurate and timely revocation checking, attackers can impersonate the server undetected and glean private information such as passwords, emails, financial data, and other personally-identifying information.

Currently deployed revocation strategies have some critical issues that interfere with providing secure and practical revocation checking. Among these concerns are scalability [2]–[4], privacy [5], page loading delays [5], and susceptibility to downgrade attacks [2], [5], [6]. The remaining strategies have other debilitating weaknesses, such as requiring significant infrastructure changes [7], [8] or opening new attack surfaces [9]. Due to these critical issues, certificate revocation is ignored by most clients, leaving many computers and mobile devices vulnerable to attack [10].

Major browser vendors consider the security of their users a high priority. At the same time, browsers emphasize low latency and fast page loads to remain competitive, and mobile browsers must consider network bandwidth usage. Because of these concerns, combined with vulnerability to downgrade attacks, Firefox and Chrome have reduced their support for traditional revocation methods such as Certificate Revocation Lists (CRLs) [2] and Online Certificate Status Protocol (OCSP) [5]. Instead, Firefox and Chrome have both deployed custom revocation systems to protect their users against the compromise of a relatively few critical certificates. Liu et al. [10] additionally found that mobile browsers do not currently evaluate certificate revocation statuses due to the previously described concerns, especially those relating to network bandwidth consumption.

Over the last few years, the certificate space has grown tremendously. From January 2017 to January 2018 alone, the number of live CA-trusted certificates observed in public Internet scans more than doubled, growing from 30 million [11] to over 80 million [12]. Two years later, in January 2020, this number has increased more than fivefold to over 434 million [12]. One reason for this explosive growth is the emergence of a new CA, Let's Encrypt [13], which freely issues certificates through an automated system. As of January 2020, the number of active trusted certificates signed by Let's Encrypt exceeded 231 million [12]. The recent efforts to promote using only encrypted web traffic [14] has also contributed to the increase of trusted certificates. As the number of certificates in use continues to grow, the scalability of revocation strategies becomes increasingly essential.

While revocation strategies should maintain acceptable network bandwidth requirements during normal conditions, it is also imperative that they effectively handle mass revocation events such as the period following the announcement of the Heartbleed vulnerability. In 2014, Liu et al. [10] found that, before the announcement of Heartbleed, the percentage of revoked certificates was approximately 1% of all active certificates (non-expired certificates signed by a trusted CA). Following the announcement, the percentage spiked to nearly 11%. During this time, Cloudflare estimated that they incurred an additional cost of $400,000 per month to publish their enlarged CRL due to increased network bandwidth consumption [15]. After measuring the effects of Heartbleed, Durumeric et al.

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24084
www.ndss-symposium.org

stated: "*The community needs to develop methods for scalable revocation that can gracefully accommodate mass revocation events, as seen in the aftermath of Heartbleed*" [16].

Despite the importance of these performance concerns, perhaps the most troubling issue is that many revocation strategies are subject to downgrade attacks. When a revocation status is unavailable because the authoritative server is inaccessible, modern browsers "soft-fail" by assuming the certificate in question is still valid. This default posture shows that browsers value accessibility over protection against the attack vectors exposed by a compromised certificate. Soft-failing is particularly dangerous since an attacker conducting a MitM attack using a revoked certificate can also trivially force soft-failure by blocking outgoing requests to verify revocation status. On this topic, Langley stated that "*soft-fail revocation checks are like a seat-belt that snaps when you crash*" [17].

These issues motivate the general requirements for an effective revocation strategy, which Larisch et al. [11] formally enumerated as the following six criteria that a universal revocation strategy should fulfill:

1) **Efficiency** - The revocation strategy should require minimal network bandwidth, storage, and computational resources.
2) **Timeliness** - Revocation status updates should be sent frequently to ensure that they never become stale.
3) **Failure Model** - The strategy should enable clients to adopt a hard-failure policy.
4) **Privacy** - The strategy should preserve client traffic privacy.
5) **Deployability** - The revocation strategy should be incrementally deployable and provide incentives for adopters.
6) **Auditability** - Revocation information should be auditable by other parties.

In this paper, we present Let's Revoke, a new revocation strategy designed to overcome the limitations of other revocation strategies and fulfill the six criteria set forth by Larisch et al. First, we provide a survey of current revocation strategies with their strengths and weaknesses. Second, we describe an efficient method for uniquely identifying certificates. Third, we propose a new revocation strategy, Let's Revoke, and describe how it fulfills the six criteria listed above and addresses the concerns surrounding current revocation strategies. Fourth, we compare Let's Revoke to other proposed revocation strategies. Fifth, we provide simulations showing the storage requirements of Let's Revoke for revocation spaces up to 10B certificates and a 10% revocation percentage. While we present Let's Revoke in the context of the web PKI, the scheme applies to any PKI where revocation scalability is an issue.

CRLite, a recent proposal that satisfies the six criteria of a universal revocation strategy, inspired our work for more efficient revocation checking. Let's Revoke offers improvements over CRLite in computational resource efficiency, especially network bandwidth. Our measurements presented in this paper estimate that Let's Revoke clients require only approximately 28% of the network bandwidth of CRLite. Let's Revoke also eliminates the necessity of CRLite to acquire the revocation statuses of all certificates on the Internet to construct the cascading Bloom filter each day, which would currently require over 950 OCSP requests per second to the CA Let's Encrypt alone.

The contributions of the paper are an analysis showing that Let's Revoke requires less storage and network bandwidth than currently deployed revocation methods, even those that cover only a small fraction of the certificate space. Further, we provide simulated revocation data anticipating the growth of the certificate space to both 1B and 10B certificates. These estimates show typical daily revocation download requirements as 612 KB and 7.4 MB, respectively. Storage estimates also remain manageable during mass revocation events that revoke 10% of all valid certificates.

II. Related Work

Many certificate revocation proposals have arisen in the past two decades. These strategies for certificate revocation generally fall into one of three classifications: pull-based, push-based, or network-assisted.[1]

A. Pull-Based Revocation

Pull-based revocation is synonymous with on-demand revocation validation. Pull-based requests for a revocation status occur only at the time a certificate needs to be validated. Numerous revocation strategies of this type have been proposed such as Certificate Revocation Trees [18], [19], Certificate Revocation System [20], and Revocation Transparency [21]. Of the pull-based certificate revocation strategies, CRLs [2] and OCSP [5] are the most commonly used today.

CRLs [2] are lists of all revoked certificates which the issuing CA assembles, signs, and distributes. A client seeking to check a single certificate must download and parse the corresponding CRL to ensure the certificate is not in the list. Scalability is the main criticism against CRLs as they can grow large[2], consuming processor, memory, storage, and bandwidth resources. Due to these issues, Mozilla Firefox and Google Chrome have disabled revocation checks using CRLs.

OCSP [5] responders provide a signed revocation status for individual certificates. OCSP requires clients to make this request for every web session initiated through HTTPS and must wait for the OCSP response to check the revocation status before the page can be fully loaded. In addition to page load delays, OCSP presents a significant privacy concern as each client divulges its browsing history to a third party. While some revocation strategies only share coarse-grained traffic patterns [2], OCSP divulges detailed client traffic patterns to the CA that signs the collection of certificates that the client checks and all the nodes along the path of an unencrypted OCSP request. Despite these concerns, most modern desktop browsers use OCSP.

[1]We employ the pull-based and push-based terminology given by Larisch et al., though clients download the revocation updates themselves [11].

[2]Apple once published a 76 MB CRL [10].

Generally, all pull-based certificate revocation strategies share common features. Depending on the specific implementation, clients typically cache responses to pull-based revocation requests to improve access times and availability. If a request is not cached or has expired, the client request introduces page delays and resorts to soft-failing when the revocation status cannot be ascertained. Thus, any issue in availability, malicious or not, forces the client to soft-fail [22]. Because uncached soft-failing revocation checks allow a MitM attack to go unnoticed, pull-based certificate revocation strategies offer little protection.

B. Push-Based Revocation

Clients using a push-based revocation strategy regularly download revocation information at periodic intervals. In contrast to pull-based strategies, push-based strategies do not reveal client traffic patterns as all clients receive similar payloads. Because the data is collected and cached on the client ahead of time, there is a higher probability that a client can rely on the cached status information at the time of connection. Compared to pull-based strategies, clients collect more certificate revocation status information than necessary.

Some revocation strategies minimize this additional bandwidth consumption by only including a small, hand-picked set of high-priority certificates. Both Google's CRLset and Mozilla's OneCRL fit into this category of selective, push-based revocation strategies.

Google pushes periodic updates to Chrome with a small list of revoked certificates called a CRLset. This list is built internally at Google by filtering Extended Validation (EV) leaf certificates by revocation reason. CRLSets have a maximum size of 250 KB [4], which equates to a capacity of about 40,000 revoked certificates. While CRLSets are useful for protecting against critical certificate compromises, they are not designed to protect most certificates. In 2015, Liu et al. [10] found that only 0.35% of revoked certificates had ever existed in a CRLset.

Mozilla produces a similar revocation list called OneCRL [3]. Instead of filtering through leaf certificates, OneCRL includes only revoked intermediate certificates, which would have a much more significant impact if abused.

Another method to minimize bandwidth consumption is to use a more efficient data structure, such as a Bloom filter. A Bloom filter is an append-only data structure used to test whether a particular element is in a given set. To achieve their efficiency, Bloom filters allow false positives but disallow false negatives. Rabieh et al. [23] showed how using two Bloom filters in tandem, one stating if a certificate is not revoked and one stating if a certificate is revoked, drastically reduced false-positive rates. In rare cases where the filters report a certificate as both revoked and not revoked, an on-demand request is necessary.

Larisch et al. [11] presented CRLite, a revocation strategy that uses a Bloom filter cascade. CRLite allows clients to download the revocation status of all live certificates across the Internet in a compressed, deterministic data structure. The cascade stores any false positive queries into another Bloom filter that tests for the opposite value. The alternating process repeats until a Bloom filter has no false-positive entries. Building the filter cascade requires testing the entire set of both non-revoked and revoked certificates through the data structure, requiring substantial computational and network resources. In January 2017, CRLite required 10 MB of storage with daily updates averaging 580 KB. Using the data we collected (see section 5) in March of 2018, we estimated that the data structure had grown to 18.0 MB.[3]

C. Network-Assisted Revocation

Network-assisted revocation strategies eliminate the need for a client to request a revocation status but instead modify the TLS ecosystem to address revocation.

One approach is through a middlebox. Revocation in the Middle (RITM) [8] is one such strategy that distributes revocation information to middleboxes throughout the Internet via a CDN. As the middlebox intercepts traffic, it checks each certificate's revocation status and appends this status to the connection as part of a TLS extension. Use of a middlebox eliminates the latency of a separate revocation status check since the middlebox is along the route of the connection. However, this strategy requires middleboxes throughout the Internet, potentially costly CDN access to update these systems, and both clients and servers to adopt a new TLS extension.

As an alternative middlebox strategy, Hu et al. [24] proposed Certificate Revocation Guard (CRG), which uses a middlebox to intercept all TLS traffic for an entity such as an organizational gateway. This middlebox performs OCSP requests to check a certificate revocation status. If a revoked certificate is detected, then a malformed certificate is returned to the client, effectively blocking the connection. This strategy does not require clients to make any modifications to participate. However, by nature of using a middlebox, mobile clients such as laptops and smartphones lose protection when they leave the network.

OCSP Stapling [6], proposed in 2001, requires each website administrator, instead of end clients, to download an OCSP response for their certificates. The web server transmits the revocation status to each client during TLS handshakes. This improvement eliminates both the page load delay and the privacy concerns of traditional OCSP. However, OCSP Stapling is still vulnerable to impersonation and man-in-the-middle attacks since an attacker can simply choose not to include the OCSP Staple in their handshake with the client.

OCSP Must-Staple [25] was proposed in 2015 to remedy this issue. An X.509 certificate extension signals the browser to block the connection if the OCSP Staple is missing. The certificate extension corrects the attack vulnerabilities in standard OCSP Stapling but requires server administrators to commit to always giving an updated OCSP Staple. Failure to do so results in their website becoming inaccessible to clients.

[3]We used source code as provided by the authors. CRLite's full Bloom filter cascade size is strongly dependent on the number of revoked certificates (19.1 M) and the number of total certificates (86.2 M). Including revocations only from publicly used certificates (1.1 M) reduces the size is to 1.93 MB.

Full adoption of OCSP Must-Staple also opens the Internet to new attack vectors if proper infrastructure is not in place ahead of time. An OCSP responder hit with an extended Denial of Service (DoS) attack could block access to a large portion of the Internet since server administrators would not be able to acquire the requisite OCSP Staple. Due to these concerns, Google Chrome currently does not support the certificate extension for OCSP Must-Staple [9], [26]. While others [10] have suggested the potential for a DOS attack is not a fundamental problem as a CDN could distribute static revocation information, OCSP Must-Staple has also suffered other problems. These include CA inconsistencies and bugs in server implementations, both of which have slowed adoption [27], [28] shown by the fact that only 0.032% of live certificates use OCSP Must-Staple [29].

Lastly, using short-lived certificates [30] is a strategy that eschews revocation checking. Instead, certificates expire shortly after issuance, generally ranging from a matter of hours [31] to just a few days [32]. This strategy requires the server to renew its certificate regularly. While it was previously not practical to change public keys on renewal [32], the emergence of new technology such as the ACME protocol [33] and the EFF's CertBot[4] enables automatic public key rotation on renewal. If a private key compromise occurs, the server administrator does not renew the certificate. This strategy is similar to OCSP Must-Staple except that the certificate is regularly renewed instead of the OCSP Staple. Removing the need for revocation checks, however, does place additional strain on other elements of the certificate ecosystem, including the issuing systems of CAs and other certificate monitors like certificate transparency logs that must ingest more records as more certificates are issued. More-frequent renewal of certificates also places increased demands on organizations in which any part of the certificate-issuing process is not fully automated, requiring additional human effort to handle.

III. Let's Revoke System Design

As stated above, Let's Revoke is designed to address the concerns raised by previous revocation strategies and fulfill the six criteria of a global revocation strategy. To accomplish these goals, Let's Revoke uses a push-based model and focuses on minimizing the required computational resources, particularly network bandwidth consumption.

Throughout this section, we use the following definitions for the two entities involved in Let's Revoke:

- CA - the entity that issues certificates and tracks revocation statuses, including CAs that issue website certificates, email certificates, code signing certificates, etc.
- Client - the entity that needs to verify the revocation status of a particular certificate.

A. Design Description

Let's Revoke utilizes dynamically-sized bit vectors, known as Certificate Revocation Vectors (CRVs), to accomplish its efficiency goals. CRVs are simple data structures that use a single bit to represent the revocation status of a certificate (0-no, 1-yes). To efficiently map certificates to their corresponding revocation bits, Let's Revoke uses new unique identifiers called Revocation IDs (RIDs). RIDs consist of three parts. The first two parts are fields already present in all certificates, namely the issuing CA and the expiration date. The third part is a new X.509 extension field called a revocation number (e.g., RID = Let's Encrypt : March 1, 2018 : 24561).

1) Revocation Numbers: Revocation numbers (RN) are non-negative integers that represent the index to a revocation bit within a particular CRV. The issuing CA assigns each RN sequentially and includes it in the certificate as a new X.509 extension field (e.g., Revocation Number: 37892). Including the issuing CA and expiration date in an RID means that RNs must be unique only among the certificates issued by the same CA that expire on the same date, thereby reducing the required magnitude of RNs.

To assign RNs, each CA uses a counter, beginning at 0, for each possible expiration date. When issuing a certificate with a given expiration date, the CA assigns an RN with the counter's current value and increments the counter. This process ensures that given n certificates that expire on the same day, each receives a unique RN $[0..n-1]$. Thus, an issuing CA and expiration date uniquely identify a CRV, and a certificate's RN provides the index of its revocation bit within that CRV. In cases where a CA issues relatively few certificates that expire each day, a unique counter can be used for a group of consecutive days to track all certificates issued within that time frame in a single larger CRV.

2) Certificate Revocation Vectors: Let's Revoke maintains a database of CRVs: one for each expiration date per CA. Every CRV is initialized as an empty bit vector. When revoking a certificate associated with a given CRV, the CA updates the CRV as follows:

1) If the newly revoked certificate's RN is larger than or equal to the number of bits in the CRV, then append enough 0-bytes to the CRV so the RN is less than the number of bits in the CRV.
2) Set the bit at the index of the RN to 1.

For example, suppose a CA issues 100,000 certificates that expire on a specific day. The CRV for that day is initially an empty bit vector. When the first certificate with RN = i is revoked, the bit vector expands with enough zero bytes so that bit i can be set to 1. For example, if RN=10 is revoked, then the empty bit vector is replaced with a two-byte vector that has bit 10 set. Suppose the next revoked certificate on the same expiration date for this CA has RN = j. If $i > j$, then bit j is set to 1. For instance, if RN=4 is revoked, then bit 4 in the first byte of the two-byte vector is set to 1. If $j > i$ and bit j is not in the same byte as bit i, then the bit vector is expanded again with enough bytes so that bit j can be set to 1. For example, if RN=30 is revoked, the bit vector is expanded to four bytes and bit 30 is set. The bit vector in each CRV has just enough bytes to mark the highest RN that is revoked.

[4]https://certbot.eff.org/about/

The CA aggregates these individual CRV updates to generate a batch update for end clients using one of the following three methods:

- ADD Method - the CA generates a list of all the newly added RNs and the client iteratively adds each item in the list to the corresponding CRV as described above.
- OR Method - the CA creates a delta CRV that includes only the newly added RNs for the client to bitwise OR with the client's current CRV, yielding a fully updated CRV.
- NEW Method - the CA generates a complete copy of the current CRV that the client will bitwise OR with its previous CRV.

The batch update is compressed, timestamped, and signed by the CA before being sent to clients upon request.

To check the revocation status of a certificate, the client first accesses the correct CRV indicated by the certificate's issuing CA and expiration date. The client then determines the revocation status of the certificate by verifying the value of the bit at index RN.

3) Example: To illustrate the usage of Let's Revoke, we provide the following hypothetical example shown in Figure 1.

The example begins with the Example CA (ECA) that issues 16 certificates on January 1, 2019, that all expire on January 1, 2020. The CA assigns a revocation number to each certificate between 0 to 15 inclusive. On February 2, 2019, the owner of the certificate with the RN of 7 sends a signed request for ECA to revoke that certificate. ECA revokes the certificate and sets the appropriate bit in the ECA CRV corresponding with the expiration date January 1, 2020. ECA then provides clients, upon request, with either the single element list ($\{7\}$) or the current ECA CRV for January 1, 2020 ("0000 0001"). Applying these updates, clients now have version 1 of the January 1, 2020 ECA CRV.

A client, upon receiving a certificate with an RID corresponding to the January 1, 2020 ECA CRV and an RN of 2, checks the corresponding bit in the CRV, determines that it is unset, and proceeds to use the certificate.

A short time later, on February 22, 2019, the two certificates with the RNs of 2 and 4 are submitted for revocation. ECA again revokes the certificates and updates the January 1, 2020 ECA CRV. When a client requests updates to this CRV, indicating it already has version 1, ECA can send the list of newly added elements ($\{2,4\}$), an updating CRV ("0010 1000"), or the current CRV for January 1, 2020 ("0010 1001"). These updates produce version 3 of the CRV. Once updated, if the client again receives the certificate with an RID for January 1, 2020, ECA, and RN 2, it finds the bit in the corresponding CRV is now set and rejects the revoked certificate.

On January 2, 2020, ECA and all clients purge the January 1, 2020 CRV from their data stores since all 16 certificates that expired on January 1, 2020 are no longer valid, irrespective of their revocation status.

B. Design Analysis

1) Revocation IDs and Revocation Numbers: Including the expiration date in a certificate's identification provides two advantages to RIDs compared to the identifiers in other revocation approaches.[5]

First, the RNs in RIDs are efficiently small and unique. Current certificate revocation tracking utilizes large pseudo-random numbers as unique identifiers, including serial numbers ($\approx$128 bits) and SHA256 fingerprints (256 bits). These large numbers are necessary to avoid misidentifying one certificate as another. However, due to their sequential issuance, RNs ensure uniqueness and can use a smaller fixed-length encoding for improved storage and communication efficiency. The design in this paper specifies encoding RNs as 32-bit integers, enabling the representation of over 4 billion certificates per CA per day, while also requiring significantly less space than traditional identifiers.

Second, the expiration date enables the efficient removal of revocation information for already expired certificates. Once a specified expiration date has passed, CRVs labeled with earlier dates are safely removed since all associated certificates have expired. This approach does not waste any storage on obsolete information and requires no additional network connections to complete these removals.

2) Certificate Revocation Vectors: CRVs effectively utilize the efficiency improvements offered by RNs and RIDs to reduce the computational, storage, and network requirements of revocation checking.

Computationally, nearly all operations occur in constant time. To check a revocation status, accessing a specified bit has constant time complexity as the certificate's RN specifies the index.

When updating CRVs, there are two operations to consider: appending additional zero bytes and setting the appropriate bit. Appending additional bytes when necessary is also a constant time operation since the number of required additional bytes is given as the difference between the total number of bytes currently in the CRV and the byte where the new RN is set. Setting a bit within the CRV is likewise done in constant time since the index of the bit is given by the certificate's RN.

Batch updates to CRVs are linear operations according to the number of new RNs since every new RN added causes a constant time update operation.

Storage requirements for CRVs also remain very low, since every certificate is represented using only a single bit As an example, a certificate space of 100M certificates[6] is represented in at most 12.5 MB (100 million bits) without any compression. Since CRVs generally contain long sequences of unset bits, compression algorithms are highly effective at further decreasing storage requirements.

[5]It is worth noting that RNs and RIDs offer efficiency gains to many other revocation strategies independent of CRVs and Let's Revoke.

[6]We chose 100M certificates as a representative number similar to the total number of certificates we found during our Internet scan, which we discuss in section V.

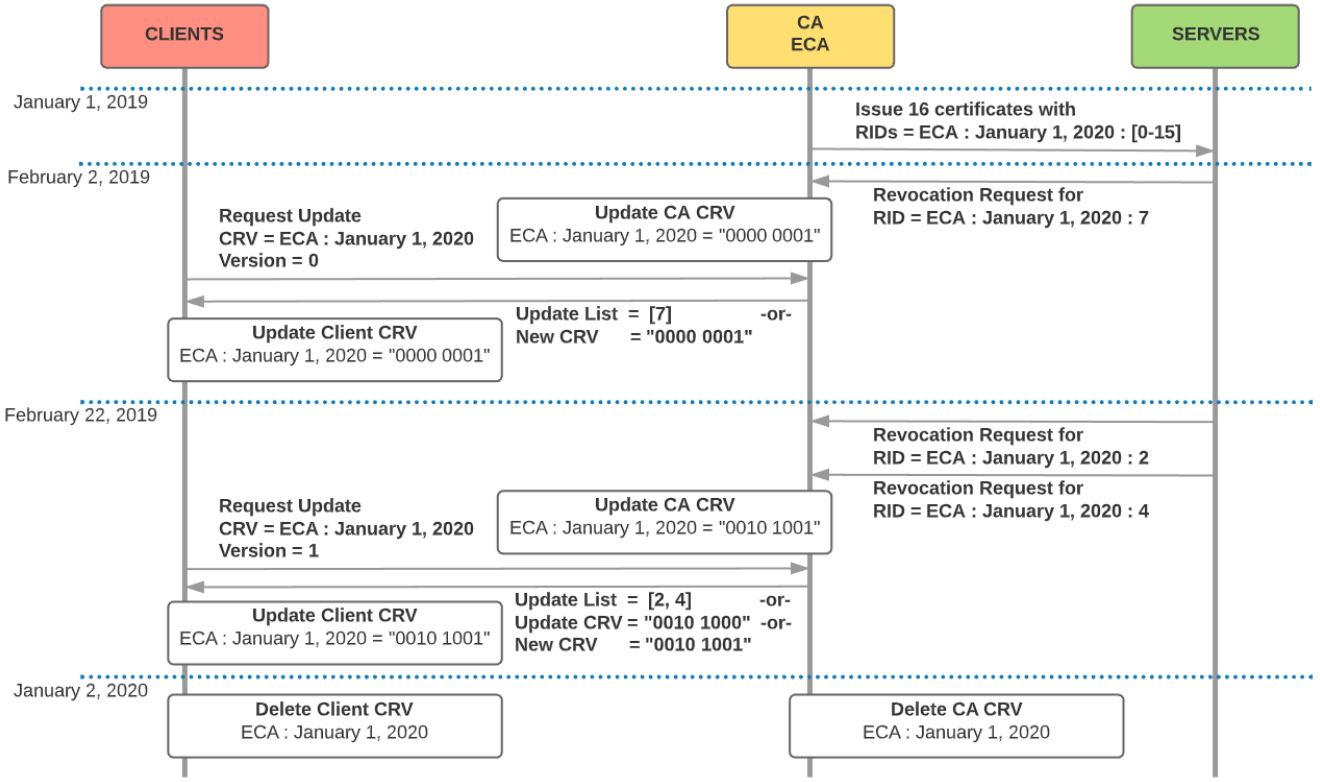


Fig. 1: Sequence Diagram for Let's Revoke showing the processes of certificate issuance to web servers and revocation as well as the update process of CRVs for both the CA and client.

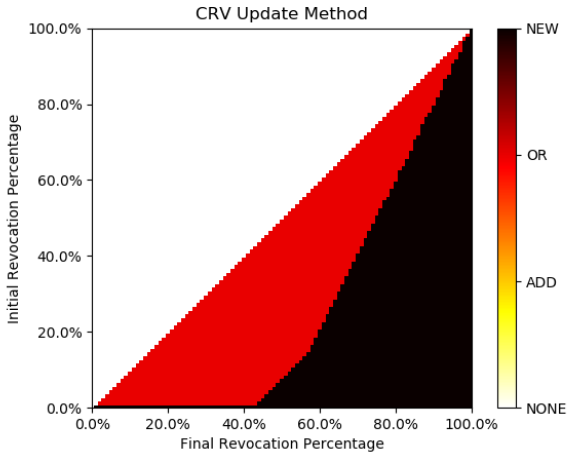
The communication of revocation updates between CAs and clients requires network resources. Let's Revoke supports three methods for communicating batch updates, and it selects the method that minimizes bandwidth costs.

The most efficient update method depends on the total number of certificates represented in a CRV, previously revoked certificates, and newly revoked certificates. The following is a general approximation to determine which update method is most efficient. If fewer than 0.1% of all represented certificates are newly revoked, the CA sends a list of the 32-bit RNs to add to the CRV (ADD method). In the rare case of revoking more than half of the current active certificates since the last update, the CA generates and sends an entirely new CRV (NEW method). For all other updates, it is usually most efficient to create and send a CRV containing only the new additions (OR method). Figure 2 illustrates more exact results for which update method minimizes bandwidth requirements and the level of those requirements at differing revocation percentages for a CRV representing 1M certificates. Most updates are relatively small and therefore use the ADD method for batch updates, typically requiring less than 3 KB for every 1M certificates covered to update daily. The OR method allows for efficient distribution of large updates during mass revocation events or when the client's CRV is sufficiently outdated.

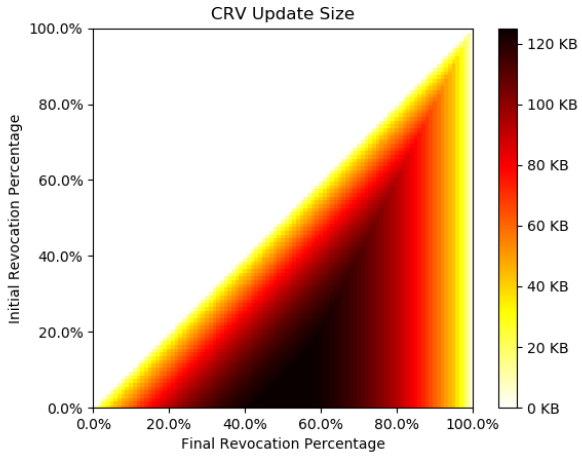
Selecting the minimally-sized update method ensures that the data structure can be communicated using minimal network bandwidth. Assuming a certificate space of 100M certificates and a total revocation rate of 2% distributed as 100 date-separated CRVs (1M certificates issued per day), then each CRV receives an additional 0.04% of new revocations every day. Each update for a single CRV requires 1.14 KB of network bandwidth using the ADD method. In total, a client downloads just 114 KB per day from 100 CRVs to receive complete revocation coverage for 100M certificates.

Since the design calls for individual CAs to generate CRVs, it is important to show that distributing the revocation information across multiple CRVs does not drastically affect the other desired attributes. Since distributed CRVs are smaller than an equivalent monolithic CRV, all computational requirements should decrease (e.g., look-up times, memory requirements). Table I shows a comparison of two different distribution strategies. The first represents a single large CA that issues 30M certificates per month and stores three monthly-generated CRVs. The second shows the corresponding values for 5 smaller CAs that issue 200K certificates per day and store 90 daily-generated CRVs (450 total CRVs). The calculation of the daily update bandwidth assumes a constant daily revocation percentage.

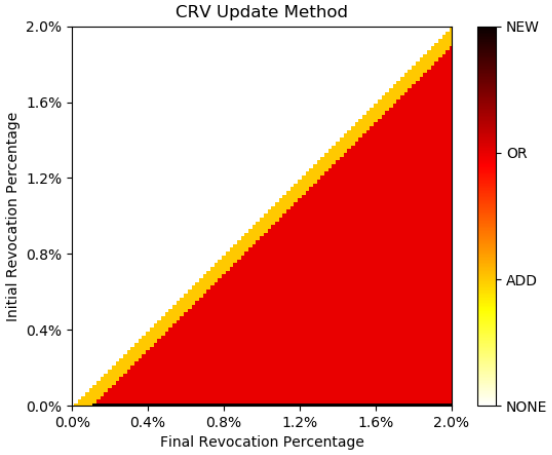
These results show that while there is storage overhead associated with the meta-data required for distributing CRVs, this overhead is minimal compared to the size of the CRVs themselves.



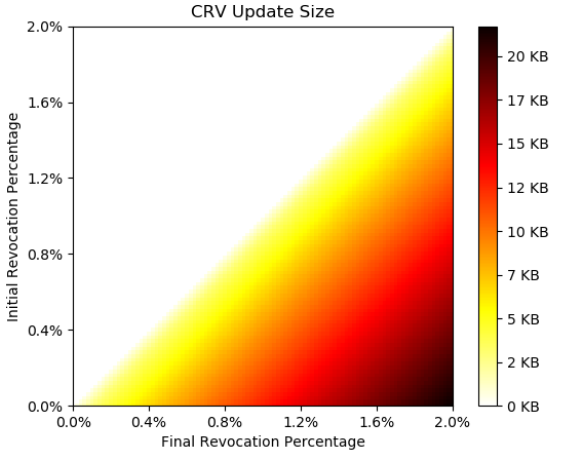
(a) Update Method for All Percentages



(b) Update Size for All Percentages



(c) Update Method for Low Percentages



(d) Update Size for Low Percentages

Fig. 2: Details of update methods and efficiency levels for a CRV representing one million certificates shown as follows:
(a) The optimal update method for the CRV depending on the beginning and ending revocation percentages.
(b) The compressed update size for the optimal update method.
(c) A zoomed in region of (a), showing the update method for normal low revocation percentages.
(d) A similarly zoomed in region of (b), showing the update size for normal low revocation percentages.

C. Distribution Methods

While CRVs are an efficient representation of revocation information, CAs must still distribute this data to potentially every client on the Internet. Currently, both CRLs and OCSP reponses are frequently uploaded to Content Delivery Networks (CDNs) to disseminate revocation information from CAs to clients more efficiently. CAs can similarly utilize CDNs to distribute CRVs by uploading, at regular intervals, the signed latest version of each CRV. For improved bandwidth efficiency, the CA also generates the most efficient CRV update from the methods described earlier. The CA then signs the generated update and uploads it to the CDN and labels it for access by the CRV version number.

Clients update their stored CRVs by periodically requesting, from the CDN, all updates with higher version numbers than their current version. If a significant time has passed since the last update request, clients simply request the latest CRV to replace their outdated version.

While CRVs would enable clients to download all revocation statuses for the entire certificate space, efficiency can be further enhanced by only downloading CRVs for certificates they use. This is important to some clients, such as Internet of Things devices, that require maximum efficiency due to constraints like limited computational, storage, or energy resources. Such clients could use a pull-based discovery mechanism to download the CRV that corresponds to any new

	Compressed Storage	Uncompressed Storage	Daily Update Bandwidth
3 CRVs			
1% Revocation	1.1 MB	11.2 MB	72 KB
10% Revocation	5.5 MB	11.2 MB	407 KB
50% Revocation	11.2 MB	11.2 MB	1.2 MB
450 CRVs			
1% Revocation	1.2 MB	11.2 MB	82 KB
10% Revocation	5.8 MB	11.2 MB	480 KB
50% Revocation	11.2 MB	11.2 MB	1.4 MB

TABLE I: A storage and bandwidth comparison of different distribution levels for CRVs representing 90M Certificates.

certificate they encounter. From that point on, the client would request push-based updates periodically for the CRV until it was no longer necessary. This approach conserves bandwidth for both clients and CAs, but comes at the cost of possible security and privacy trade-offs, which we describe later.

D. Limitations

Since Let's Revoke requires a new certificate field (RN) it can only be used to revoke new certificates that adopt its format. It is not designed to be backward compatible to handle revocation for existing certificates. Furthermore, CAs must accept the responsibilities of issuing these unique RNs, tracking revocations in CRVs, and providing CRV updates to clients.

To achieve its efficiency, Let's Revoke only provides the revocation status of a given certificate. Additional information such as revocation time and reason are not available using CRVs. However, other revocation schemes that include these details, like CRLs, can be supported in tandem with CRVs to provide this additional information to requesting clients.

IV. Comparing Revocation Strategies

This section compares Let's Revoke to a range of revocation strategies based on the six criteria outlined for evaluating a revocation strategy. We compare Let's Revoke to two other centralized revocation strategies (CRLite and a simple "RN Listing" strategy) that allow clients to adopt a hard-failure policy.[7] Additionally, CRLs, OCSP, and CRLsets are compared with Let's Revoke because these revocation strategies are currently used today. Our analysis is summarized in Table II.

A. Efficiency

There are two primary resources for considering the efficiency of a revocation strategy, namely device storage and network bandwidth.

For CRLs and OCSP these requirements are highly variable depending on the usage of certificates by a given client, though average costs can be established for both of these strategies. For CRLs, the average CRL requires 173 KB in both bandwidth and cache storage, as determined by an Internet-wide scan that we describe in Section V. OCSP requests require 1.3 KB for again both bandwidth and storage costs. CRLSets have a fixed maximum size requiring 250 KB of device storage and network bandwidth daily. However, these three strategies do not provide secure coverage for all revocation statuses.

To provide a global revocation system comparable to Let's Revoke and CRLite, the above strategies could be used or extended to generate a list of all the revocation statuses for every non-expired certificate each day. As a model for such a listing strategy, we generated a list of the revoked certificates using 32-bit binary representations of the associated revocation numbers as unique identifiers. This Revocation Number Listing (RN Listing) strategy requires significantly fewer bits per revocation than both traditional CRLs and CRLSets, which use serial numbers and are typically around three to four times the size of a revocation number. CRLs also include the date of revocation and, in some situations, the reason for revocation. Thus, the listing model we used provides a reasonable lower bound for a list of revoked certificates.

To compare the storage requirements of Let's Revoke against both CRLite and RN Listing, we utilized all three strategies to store information for a certificate space containing one million certificates[8] at various revocation percentages. For each of these strategies, we ran 100 simulations within two ranges of revocation percentage, incrementing the revocation percentage evenly through the range. The first ranged from 0-100%, which demonstrates the scalability of the strategy as revocation percentages rise dramatically. The second ranged from 0-4% to show a finer granularity between strategies at more typical revocation percentages.

For each simulation of Let's Revoke, we built and compressed one large CRV representing all one million certificates by randomly selecting bits equal to the number of revoked certificates and setting each of these to 1. We serialized the resulting CRV into a binary file and XZ compressed the file.

Every CRLite simulation created a Bloom filter cascade[9] using the source code provided by the authors of CRLite. Because the layers of the Bloom filter cascade form a relatively patternless binary stream, XZ compression was not effective.

We also compared Let's Revoke to a combinadics representation, which is a lower bound for representing any combination of values. The combinadics, or combinatorial, number system uses a lexicographic ordering to rank a combination instance, so the index alone denotes it. Because revocation numbers provide an obvious ordering, this type of representation is possible. For example, given 1 million certificates with a 1% revocation rate (10,000 revocations), there

[7]While other revocation strategies also allow clients to adopt a hard-fail policy (such as OCSP Must-Staple [25] and RITM [8]), no other previously proposed strategy can do so without adding new entities in the PKI ecosystem or forcing relatively high numbers of servers to change their configurations and key management practices.

[8]One million certificates roughly corresponds to the daily issuance of the CA Let's Encrypt [12].

[9]We used the following parameters in our simulations:'p'= 0.5 as recommended by the authors; for 'r' and 's', we chose values 102.5% times the receptive certificates used, allowing room for growth and daily deltas; 'p1' = $r * \sqrt{p}/s$.

	Efficiency	Timeliness	Failure Model	Privacy Preserving	Deployability	Auditability
CRL	173 KB per CRL†	7 Days	Soft	Yes	Deployed	Yes
OCSP	1.3 KB per request [11]	4 Days	Soft	No	Deployed	Yes
CRLSet	250 KB per day	1 Day	Soft	Yes	Deployed	No
RN Listing	Initially 5.1 MB; 114 KB per day∗	1 Day	Hard	Yes	Incremental	Yes
CRLite	Initially 3.1 MB; 408 KB per day∗	1 Day	Hard	Yes	Incremental	Yes
Let's Revoke	Initially 2.2 MB; 114 KB per day∗	1 Day	Hard	Yes	Incremental	Yes

TABLE II: Comparison of Let's Revoke to other revocation strategies.
†: The average size of CRLs in our dataset.
∗: Simulated values at 2% revocation rate, 1% daily expiration rate, using 100 Million certificates.

are approximately $10^{24,340}$ or $2^{80,856}$ possible combinations of revoked certificates. A combination of certificates given these parameters can be denoted with its index in 80,856 bits (10.2 KB). While the combinadics representation requires even less storage space than Let's Revoke, unranking algorithms that convert a combinatorial index back into the expanded form have expensive space and time costs, rendering them impractical for certificate revocation.

Figure 3 summarizes the storage requirements across all revocation percentages, while Figure 4 shows the comparison at the low revocation percentages likely to be found in daily use. While there are more efficient methods of providing update information (i.e., delta updates), it is important to note that the compressed storage requirements for each strategy represent the maximum required bandwidth. For low revocation percentages (<0.02) all strategies compress to approximately the same size. Both other strategies quickly outperform the RN Listing strategy. Around a revocation percentage of 0.5% RN Listing requires double the storage requirements of the other strategies. CRLite and CRVs remain competitive until about 2.0% revocation, where CRVs begin to outperform CRLite significantly. Even more interesting is the close relationship between the size of the combinadics representation and that of the compressed CRV. While the combinadics representation is smaller, it is only marginally so, indicating that a compressed CRV is close to the theoretically minimal size for representing a set of randomly revoked certificates.

Uncompressed CRVs require significantly more storage space than either RN Listing and CRLite until relatively high revocation percentages. Thus, storing an uncompressed CRV generally requires significantly more storage than either of the other two considered strategies. However, this can be avoided by only decompressing the CRV at the time of use, which allows end clients to choose between storage and computational efficiency. Our testing showed that revocation checks for a single certificate using an uncompressed CRV representing 1M certificates finished in under 1 ms, while the same checks took

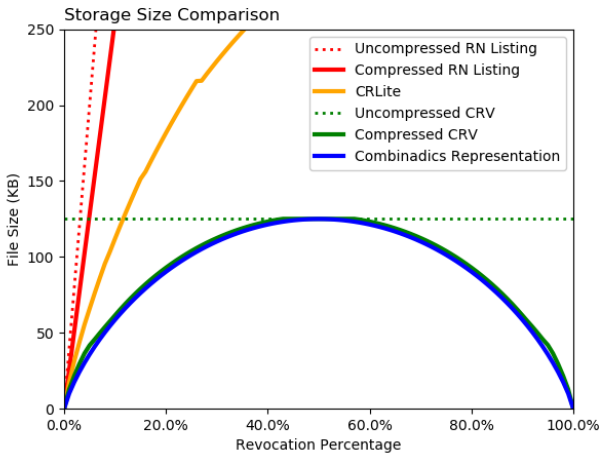


Fig. 3: Storage sizes for different revocation strategies representing one million certificates

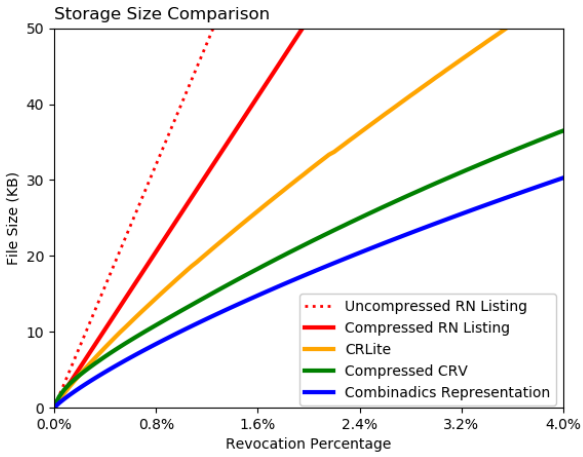


Fig. 4: Storage sizes for different revocation strategies representing one million certificates in typical ranges at higher detail.

approximately 10 ms when starting with a compressed CRV.[10]

Bandwidth requirements were also simulated for our model listing strategy, CRLite, and Let's Revoke. These simulations used the same parameters as those described in Section III.B.4, namely a certificate space of 100 million certificates, a 2% revocation rate, and a 1% daily expiration and issuance rate. These values serve as a representative point since numerous variables affect the final bandwidth requirement. For each strategy, update simulations depend on the initial and final revocation percentages as well as the total number of revocations. CRLite's updates also depend on changes to the size of the entire certificate space and the usage of the total space allotted for future growth. For CRLite, we used a fixed certificate space at 100 million certificates and a 90% storage utilization rate.

Since Let's Revoke and RN Listing both use RNs, and the optimal update method for Let's Revoke at these low revocation percentages is a list of new revocations, their daily update costs are identical at 114 KB per day. The low daily bandwidth is particularly significant since these systems provide full revocation coverage and require less daily bandwidth than even CRLSets. CRLite's daily delta requirements are higher at 408 KB per day, primarily due to the additional instructions required to remove expired certificates from the cascading Bloom filter.

B. Timeliness

Most revocation strategies utilize either revocation status caching or regularly scheduled updates to ensure time and bandwidth efficiency. Pull-based strategies, such as CRLs and OCSP, typically cache revocation status information and assume it is valid for up to 7 days. A 7-day period leaves a large window of time in which an adversary can exploit a compromised certificate, even after correctly revoking that certificate.

Push-based strategies generally use periodic updates. CRLset, OneCRL, and CRLite all push daily updates to clients, which drastically reduces an adversary's window of opportunity when compared to traditional pull-based methods. If deemed necessary, push-based strategies can elect to generate even more frequent updates.

Let's Revoke is intended to be used as a push-based strategy. Regular updates can be pushed out daily as necessary, or as frequently as a CA or client desires. This provides clients with both consistently updated revocation information and the freedom to choose which trade-offs to make between bandwidth costs and more reliable security. Providing revocation information on demand for clients instead of at regular intervals comes at increased costs to a CA that provides this service.

C. Failure Model

As stated previously, pull-based strategies suffer from soft failing revocation checks. This issue directly affects both CRLs and OCSP. CRLSets, though they are a push-based model, also must adopt a soft failing approach by assuming that any certificate not in its limited coverage space is not revoked.

Since Let's Revoke covers the global certificate space and uses a push-based model, clients using Let's Revoke can assume a hard failure model for certificates with an RID. Since all the certificates in the working set have the timestamp of the last successful revocation status check, the client can set a limit on how long they rely on that status check before hard-failing if the check is not refreshed. CRLite is the only other strategy in our comparison that can adopt this hard failure model.

D. Privacy

Since a CRL covers all the certificates issued by a CA and is acquired only upon request, third parties or eavesdroppers learn only extremely coarse access patterns. In contrast, OCSP does not preserve user privacy. OCSP requests correspond to the certificates in use, revealing the users' browsing habits to third parties and eavesdroppers.

Both CRLSets and CRLite provide the same revocation information to every user. Since this coverage does not depend on any user action, it maintains the privacy of its users.

Similar to CRLs, CRVs cover a range of certificates. Downloading a given CRV does not reveal which particular certificate's revocation status a client needs. However, should a client decide to only track the CRVs it uses, the client becomes vulnerable to potential denial of service (DoS) and privacy issues. First, since the client downloads only CRVs for certificates that it encounters, if the CRV is unavailable when needed for any reason, malicious or coincidental, the hard-failure policy results in a DoS until the CRV becomes available. This concern is partially mitigated by the use of CDNs for distributing CRVs. Second, by only tracking selected CRVs, a passive adversary may be able to derive client browsing patterns by correlating the CRVs a particular client downloads. Thus, it remains for the client to choose between privacy concerns and the efficiency gains of selective CRV Tracking.

E. Deployability

CRLs, OCSP, and CRLSets are already deployed. However, modern browsers do not use CRLs and OCSP due to reasons discussed previously.

CRLite can be incrementally deployed today with software updates to clients. It requires an agent to aggregate all available revocation information regularly to provide updated filters. This daily aggregation process imposes significant computational requirements[11] on the CRLite agent and adds similar costs to the CRL and OSCP endpoints. There is little incentive for an agent to incur both the network and computational costs to perform this daily aggregation.

Similarly, Let's Revoke is incrementally deployable now; however, Let's Revoke also incentivizes both CAs and clients by lowering computational and networking resource consumption

[10] We conducted timing measurements on a Dell Laptop with an Intel 8th-gen Core i7 processor and 16GB of RAM.

[11] Aggregating only the revocation statuses of the 231 Million Let's Encrypt certificates every 24 hours would require processing more than 2,500 OCSP requests each second. Further computing resources would be necessary to handle the remaining 203 Million revocation statuses.

while providing the desired security. More precisely, CAs must begin issuing RNs and tracking CRVs. Any CA could begin doing so without negatively impacting their certificates or other revocation processes. Clients that then implement the local CRV store can assume a hard-failure strategy for any certificate with a RN and rely on current revocation strategies for all other certificates.

F. Auditability

Auditability is an important feature of any revocation scheme that desires to prevent malicious or faulty revocation status distribution. Most schemes are auditable; however, due to the nature of CRLSets and OneCRL, these revocation strategies cannot be audited for completeness, modification, or equivocation.

It is easy to audit Let's Revoke for errors of omission and equivocation. To detect omissions, clients apply a bitwise-OR between the most recent CRV with any previous CRV. The resulting CRV matches the most recent CRV if there are no omissions. To detect equivocation, apply a bitwise-XOR to any two clients' CRVs with matching version numbers. The resulting CRV contains all zeros if there is no equivocation.

V. Viability Simulations

To further validate the performance of CRVs, we ran some revocation simulations to show that CRVs work well for current everyday revocation checking and scale both for mass revocation events and the larger certificate spaces of the future.

A. Methodology

1) Data Collection: To ensure the accuracy of our simulations, we acquired the relevant revocation data for the global certificate space as of March 21, 2018. Similar to previous revocation measurement studies [10], we collected all certificates seen in previous scans and then filtered out certificates that have expired or were no longer trusted by any standard root store. We obtained our initial data from Censys.io [34], a search engine created to allow researchers to access data from daily Internet scans. On March 21, 2018, the number of certificates tagged by Censys.io as "Currently Trusted" (non-expired, trusted by Apple's, Microsoft's, or Mozilla NSS's root store) was 88.9M.

First, we filtered this dataset by removing duplicate, expired,[12] private,[13] and invalid certificates.[14] After filtering, 84.1M certificates remained.

Second, we separated the certificates with CRL endpoints (26.8M, 33.6%) from those with only OCSP endpoints (55.3M, 66.3%) [10], [11]. We identified 475 unrevokable certificates that did not have any revocation endpoint[15] and removed them from the dataset as well.

[12]Certificates that expired by March 21st.

[13]Private certificates are those using an LDAP endpoint or are otherwise inaccessible. Most of these certificates returned an unauthorized status code on request.

[14]Each invalid certificate had at least one Zlint error [35].

[15]Of the 475 unrevokable certificates, all but 2 certificates were a root certificate, an intermediate certificate, or an OCSP signing certificate.

	Cleaned Certificates	Good Revocation Status	Revoked Revocation Status	Unknown Revocation Status
From CRL	26,772,989	25,983,705	789,284 (2.90%)	0
(OCSP) Let's Encrypt	53,196,388	52,946,338	250,050 (0.47%)	0
(OCSP) Symantec	2,483,288	2,446,508	36,780 (1.48%)	0
(OCSP) DigiCert	1,157,956	1,149,840	8,116 (0.70%)	0
(OCSP) Other	542,641	541,807	807 (0.15%)	27
Total	84,153,262	83.068,198	1,085,037 (**1.29%**)	27

TABLE III: The reported revocation status of certificates in our data set.

Third, we scanned the remaining certificates to determine their revocation status. Before scanning, we obtained permission from each CA who issued over 1 million OCSP-only certificates for a specific scan rate that would not place an undue burden on their OCSP server. We sent 50 OCSP requests per second to Let's Encrypt's endpoints and 10 requests per second to Symantec's and DigiCert's endpoints. For all other OCSP-only certificate endpoints, we limited our request rate to 10 requests per second. In addition to rate-limiting, our software also implemented exponential back-off for any response errors.

Table III contains a summary of the results of our scan. The revoked certificates comprise about 1.29% (1.08M) of the total certificates, with an average daily revocation rate of 0.007%. The 1.29% rate is very similar to the rate for revoked certificates in 2014 [10] before the discovery of the Heartbleed vulnerability. We believe that the 1-2% rate for revoked certificates is typical in the absence of a mass revocation event.

2) Simulator: We created a simulator with parameters controlling the number of CAs, the number of expiration dates recorded by the CA, the number of certificates issued per day, the percentage of revoked certificates per CA, and the percentage of new revocations on a given day per CA. For each simulation, every CA was assigned RNs spread uniformly across all their possible active CRVs (equal to the number of days until their last certificate expires) corresponding to its number of non-expired certificates. We then provided the revocation percentage for each CA and revoked randomly selected RNs from the CA's CRVs. We generated the resulting CRVs and measured their size. Using the percentage of new revocations, we created an average update (revoking that many more RNs) and built the most efficient update for that day, yielding the update size. Summing the storage and bandwidth requirements for every CRV from every CA gave us the total storage and bandwidth, which we report for various scenarios below.

B. Current Revocation Space

With the data acquired from our scan, we simulated the current requirements for global revocation coverage using revocation numbers and CRVs. For the simulation, we grouped all of the CAs with fewer than 10K total certificates into

	Compressed Storage	Uncompressed Storage	Daily Update Bandwidth
100M Certificates			
1% Revocation	1.3 MB	12.5 MB	62.6 KB
10% Revocation	6.2 MB	12.5 MB	429.2 KB
1B Certificates			
1% Revocation	12.2 MB	125 MB	611.5 KB
10% Revocation	60.1 MB	125 MB	4.1 MB
10B Certificates			
1% Revocation	121.3 MB	1.25 GB	7.4 MB
10% Revocation	605 MB	1.25 GB	41.5 MB

TABLE IV: A storage and bandwidth comparison for CRVs representing different certificate space sizes.

a conglomerate CA. After the grouping, we had 42 CA entities that had issued 84.1M certificates with a revocation percentage of 1.29% and an average daily addition of 0.007%. Our simulation used all of these parameters and yielded results showing that the representation of the entire revocation space compressed to under 5.0 MB with the optimal method for daily updates compressing to less than 25 KB.[16] These small requirements indicate that CRVs are well-suited for the certificate space and revocation conditions we found during our scan.

C. Mass Revocation Event

In contrast to the typically low revocation percentages, specific wide-spread security vulnerabilities have forced periods of mass revocation. Such events include CA compromise (Trustico Revocation Event) and server bugs (Heartbleed Vulnerability). Using data from the Heartbleed Revocation Event, we ran simulations modeling the requirements that would be necessary if a similar event occurred today. To match that event, we raised the reported revocation percentage to 10% by proportionally scaling each CA's revocation percentage and increased the average daily update to 0.06%. This increase brought storage requirements to 10.8 MB and necessary update bandwidth to 150 KB per day. While much higher than the storage and bandwidth requirements for typical revocation percentages, this simulation shows that CRVs can also scale to handle mass revocation events.

D. Growing Revocation Space

To show how CRVs scale into potential future certificate spaces, we ran simulations representing a single large CA responsible for all certificates that divides them into 100 day-separated CRVs. The update bandwidth calculation assumes the associated revocation percentage occurs uniformly across all CRVs each day. Table IV contains a summary for each revocation percentage.

Of particular interest are the results for 1B certificates, which is slightly more than double the size of the current certificate space. The 100 associated CRVs store all the requisite information in less than 125 MB uncompressed. At 1% revocation, these CRVs compressed to 12 MB. Assuming an optimal update methodology of 0.02% new revocations per day, the compressed CRVs only require 612 KB of daily bandwidth. Increasing to the scale of a mass revocation event (10% revocation), the CRVs compressed to 60 MB and required a daily bandwidth of 4.1 MB. These results indicate the ability of CRVs and Let's Revoke to scale gracefully with the ever-growing certificate space.

[16]For reference, fetching the Google home page requires approximately 400 KB of network bandwidth.

VI. Security Analysis

We assume a threat model where an active network attacker can create, modify, and block messages. The attacker has two goals: (1) coerce a client to accept a revoked certificate, and (2) coerce a client to assume a valid certificate is revoked. The threat model does not include a compromised CA or a compromised client.

A. Accept a Revoked Certificate

An attacker can coerce a client to accept a revoked certificate by preventing them from updating their CRV and learning that the certificate is revoked.

1) Update Manipulation: The first method for doing so is to try and provide an update that omits the needed new revocation or remove a previously added revocation. To ensure that a revocation update is valid, the CA must digitally sign each update. It is then the client's responsibility to validate that signature. Further protection prevents the removal of previous revocations. Since the design of CRVs allows only insertions, the only way to remove a previous revocation is for the attacker to send an update indicating a new CRV. However, this attack is easy to prevent by having the client bitwise OR the new (malicious) CRV with the old CRV to ensure that all previous revocations remain even if the malicious CRV has excluded previous revocations.

2) Update Blocking: The second attack vector for an adversary is to prevent updates from reaching the client by blocking traffic from the CA. This attack allows the adversary to conduct a MitM attack on the client using any certificate revoked since the last time the client updated its CRV. Since CRVs are a push-based revocation strategy, the client can detect any interference with the update schedule and warn the user about a potential MitM attack.

B. Revoke a Valid Certificate

An adversary may also coerce a client to believe a valid certificate has been revoked. The net effect is a denial of service attack that prevents a client from using the service associated with the certificate.

1) Unauthorized Revocation: An attacker can attempt to impersonate the owner of the certificate to the CA and revoke the certificate. The revocation process should require the revoking party to prove that they have access to the private key. If an adversary has access to the private key, then the certificate should be revoked.

2) Update Manipulation: The attacker can also attempt to modify updates to the CRV by adding new numbers to any of the update methods. Again, this attack is defeated because the CA digitally signs the updates, and a client verifies those signatures.

VII. Conclusion

This paper presents Let's Revoke, a scalable global revocation strategy that addresses the concerns of current revocation checking. Let's Revoke introduces a new unique identifier to each certificate that serves as an index to a dynamically-sized bit vector containing revocation status information. The bit vector approach enables significantly more efficient revocation checking for both clients and certificate authorities.

We demonstrated how Let's Revoke fulfills six properties [11] of a scalable revocation strategy, namely:

1) **Efficiency** - Let's Revoke offers significant efficiency gains over other push-based revocation strategies.
   a) *Bandwidth* - CRVs minimize bandwidth requirements, not only for end-clients but also for certificate issuers and revocation status responders.
   b) *Storage* - CRVs require fewer storage resources than all other currently implemented and proposed strategies that offer comparable revocation coverage.
   c) *Computational* - CRVs are simply and efficiently constructed and utilized, requiring minimal computational time and resources.
2) **Timeliness** - Updates to CRVs can be acquired daily (or even more frequently) to ensure that they never become stale.
3) **Failure Model** - Since CRVs can efficiently represent the entire revocation space, clients can adopt a hard-failure policy.
4) **Privacy** - Since CRVs provide global coverage they preserve client traffic privacy.
5) **Deployability** - CRVs allow for incremental deployment and provide an incentive to each of the involved entities.
6) **Auditability** - CRVs are auditable by all other parties.

We showed that Let's Revoke requires fewer resources than currently available revocation methods, even those that cover only a fraction of the certificate space. We simulated certificate revocation data anticipating certificate growth to 1B and 10B certificates. Our estimates show the daily revocation download estimates are 612 KB and 7.4 MB, respectively. Storage estimates also remain manageable during mass revocation events that revoke 10% of all valid certificates.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1528022 and CNS-1816929.

References

[1] "Ballot 193 – 825-day Certificate Lifetimes." [Online]. Available: https://cabforum.org/2017/03/17/ballot-193-825-day-certificate-lifetimes/

[2] R. Housley, W. Ford, T. Polk, and D. Solo, " Internet X.509 Public Key Infrastructure Certificate and CRL Profile," RFC Editor, RFC 2560, January 1999.

[3] "CA:RevocationPlan." [Online]. Available: https://wiki.mozilla.org/CA:RevocationPlan#OneCRL

[4] "CRLSets." [Online]. Available: https://dev.chromium.org/Home/chromium-security/crlsets

[5] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," RFC Editor, RFC 2560, June 1999.

[6] D. Eastlake, "Transport Layer Security (TLS) Extensions: Extension Definitions," RFC Editor, RFC 6066, January 2011.

[7] A. Schulman, D. Levin, and N. Spring, "Revcast: Fast, private certificate revocation over fm radio," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[8] P. Szalachowski, C. Amann, T. Lee, and A. Perrig, "RITM: Revocation in the Middle," in *36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016.

[9] "Feature request: OCSP Must Staple (RFC 7633)." [Online]. Available: https://groups.google.com/a/chromium.org/forum/#!topic/security-dev/-pB8IFNu5tw

[10] 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 *Proceedings of the Conference on Internet Measurement Conference (IMC)*. ACM, 2015.

[11] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers," in *2017 Symposium on Security and Privacy (SP)*. IEEE, 2017.

[12] "Censys," 2017. [Online]. Available: https://censys.io/certificates?q=tags.raw%3A+%22trusted%22

[13] J. Aas, R. Barnes, B. Case, Z. Durumeric, P. Eckersley, A. Flores-López, J. A. Halderman, J. Hoffman-Andrews, J. Kasten, E. Rescorla *et al.*, "Let's encrypt: An automated certificate authority to encrypt the entire web," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2473–2487.

[14] "Indexing HTTPS pages by default," 2015. [Online]. Available: https://security.googleblog.com/2015/12/indexing-https-pages-by-default.html/

[15] M. Prince, "The Hidden Costs of Heartbleed," 2017. [Online]. Available: https://blog.cloudflare.com/the-hard-costs-of-heartbleed/

[16] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer *et al.*, "The matter of heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 475–488.

[17] A. Langley, "Revocation checking and Chrome's CRL," 2012. [Online]. Available: https://www.imperialviolet.org/2012/02/05/crlsets.html

[18] P. C. Kocher, "On certificate revocation and validation," in *Financial Cryptography*, R. Hirchfeld, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 172–177.

[19] M. Naor and K. Nissim, "Certificate revocation and certificate update," *IEEE Journal on selected areas in communications*, vol. 18, no. 4, pp. 561–570, 2000.

[20] S. Micali, "Efficient certificate revocation," Cambridge, MA, USA, Tech. Rep., 1996.

[21] B. Laurie and E. Kasper, "Revocation transparency," *Google Research, September*, 2012.

[22] M. Marlinspike, "Defeating OCSP with the Character '3'," *Blackhat 2009*, 2009.

[23] K. Rabieh, M. M. Mahmoud, K. Akkaya, and S. Tonyali, "Scalable certificate revocation schemes for smart grid ami networks using bloom filters," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 4, pp. 420–432, 2017.

[24] Q. Hu, M. R. Asghar, and N. Brownlee, "Certificate Revocation Guard (CRG): An Efficient Mechanism for Checking Certificate Revocation," in *Proceedings of the 41st Conference on Local Computer Networks (LCN)*. IEEE, 2016.

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

[26] A. S. Wazan, R. Laborde, D. W. Chadwick, F. Barrere, and A. Benzekri, "Tls connection validation by web browsers: Why do web browsers still not agree?" in *41st Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, 2017.

[27] H. Bock, "The Problem with OCSP Stapling and Must Staple and why Certificate Revocation is still broken," 2017. [Online]. Available: https://blog.hboeck.de/archives/886-The-Problem-with-OCSP-Stapling-and-Must-Staple-and-why-Certificate-Revocation-is-still-broken.html

[28] T. Chung, J. Lok, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, J. Rula, N. Sullivan, and C. Wilson, "Is the Web Ready for OCSP Must-Staple?" in *Proceedings of the Internet Measurement Conference 2018*. ACM, 2018, pp. 105–118.

[29] "Censys," 2017. [Online]. Available: https://censys.io/certificates?q=%281.3.6.1.5.5.7.1.24%29+AND+tags.raw%3A+%22trusted%22

[30] R. L. Rivest, "Can we eliminate certificate revocation lists?" in *International Conference on Financial Cryptography*. Springer, 1998, pp. 178–183.

[31] Y.-K. Hsu and S. Seymour, "Intranet Security Framework Based on Short-lived Certificates," in *Proceedings of the Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1997.

[32] E. Topalovic, B. Saeta, L.-S. Huang, C. Jackson, and D. Boneh, "Towards Short-Lived Certificates," *Web 2.0 Security and Privacy*, 2012.

[33] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten, "Automatic Certificate Management Environment (ACME) draft-ietf-acme-acme-12," Internet Requests for Comments, Internet-Draft, April 2018. [Online]. Available: https://tools.ietf.org/html/draft-ietf-acme-acme-12

[34] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by Internet-wide scanning," Oct. 2015.

[35] D. Kumar, M. Bailey, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, and J. A. Halderman, "Tracking certificate misissuance in the wild," in *2018 Symposium on Security and Privacy (SP)*. IEEE, 2018.