ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

An Analysis of Bloom Filter Cascades - CRLite

Bachelor Thesis

Karin Holzhauser

February 17, 2020

Advisor: Prof. Dr. Kenneth Paterson
Department of Computer Science, ETH Zürich

Abstract

CRLite, using a cascading Bloom filter structure for compactly encoding all TLS certificates to support fail-closed revocation checking, was published by researchers at the IEEE Symposium on Security and Privacy in 2017. In 2019 Mozilla started to implement CRLite and is actively promoting its use. In analyzing the TLS certificate ecosystem we have found that further growth in certificate numbers is to be expected. Our analysis showed that the ecosystem's move to shorter average lifetime per certificate and its high variability of daily newly issued certificates effect CRLite's overall size negatively. Both with a mathematical model and in experiments we have shown the feasibility of insertion attacks. A potential remedy is oversizing the Bloom filter cascade. This might adversarially affect the needed space efficiency. We recommend further research into an alternative data structure to compactly encode the dynamic nature of TLS certificates for a fail-closed solution.

Acknowledgements

I want to thank my advisor, Prof. Dr. Kenny Paterson, for his guidance and support for this project. Thank you for introducing me to the challenges of research. I also thank you for always making time for my questions and ideas.

My thanks go to Dr. Thyla van der Merwe at Mozilla for sparking my interest in finding a fail-closed solution for revocation checking of TLS certificates. It will keep me busy beyond my Bachelor Thesis. I also want to thank John Aas at Let's Encrypt for helping me better understand the TLS certificate ecosystem.

I thank my sisters, Alex, Corinne, Diya, Lucille, Marion, Oana and Sharon, for having my back and encouraging me to go for my wildest dreams.

To my family, Hakan, Nicolas and Christopher, thank you for all your love and encouragement and your unwavering belief in me. You give my aspirations wings, always. And to my parents, my thanks for laying the foundation for it all.

Contents

Acknowledgements **iii**

Contents **v**

1 Introduction **1**
- 1.1 The Importance of Certification Revocation 1
- 1.2 The Need for a New Revocation Check Protocol 2
- 1.3 The Focus of this Thesis 3
- 1.4 The Structure of the Thesis 3

2 Background: Bloom Filters and Bloom Filter Cascades **5**
- 2.1 Bloom Filters 5
  - 2.1.1 Notations for Bloom Filter 5
  - 2.1.2 Dynamic Data Sets - Dynamic Bloom Filters 7
  - 2.1.3 Optimal Bloom Filters 8
  - 2.1.4 Vulnerabilities of Bloom Filters 10
- 2.2 Bloom Filter Cascades for TLS Certificates 11
  - 2.2.1 Optimal CRLite BFC 11
  - 2.2.2 Notations for CRLite BFC 14
- 2.3 CRLite: CRLite BFC, Delta Updates and CRLite Algorithm . . 15
  - 2.3.1 CRLite BFC and Delta Updates 15
  - 2.3.2 CRLite Algorithm 16
  - 2.3.3 CRLite Principles 16
- 2.4 CRLite BFC Evaluation 18
  - 2.4.1 Mathematical Evaluation of CRLite BFC 18

3 The TLS Certificate Ecosystem **23**
- 3.1 Valid certificates in numbers 23
- 3.2 Consequences for CRLite 26

4 CRLite Vulnerabilities **27**

4.1 Sizing the CRLite BFC 28
4.1.1 Defining the CRLite BFC Parameters 28
4.1.2 Higher than Expected False Positives 30
4.2 CRLite Fail-Closed 32
4.3 Targeted Revocation of Certificates 32
4.3.1 R and S constellations 33
4.3.2 Revocation as a Source of CRLite Vulnerabilities . . . 36
4.3.3 Analysed Scenario 36
4.3.4 Tested Assumptions 36
4.3.5 Evaluation Results 37
4.3.6 Experiments 39
4.3.7 Findings 42
4.4 Further Concerns 45

5 Exploitation of Vulnerabilities **47**
5.1 Threat Model 47
5.1.1 CRLite Model 47
5.1.2 Adversary 48
5.2 Probability of Success and Resulting Risks 49
5.2.1 Operation Destabilize 49
5.2.2 Certificate Masters 50

6 Conclusion **51**

Bibliography **53**

Declaration of Originality **56**

Chapter 1

Introduction

In 2019 Mozilla announced that it would start work on a new TLS certificate revocation status checking approach called CRLite. It actively promotes this approach at conferences [18] and via blogs [10]:

> ″CRLite gives us the ability to deliver all the revocation knowledge needed to replace OCSP, and do so quickly, compactly, and accurately.″

Due to the increasing importance of CRLite this Bachelor Thesis looks into its vulnerabilities under standard operating as well as attack conditions.

1.1 The Importance of Certification Revocation

Browser vendors and websites overwhelmingly use TLS to provide secure interaction on the world wide web. Recent telemetry information collected by Firefox shows that HTTPS, one of the protocols using TLS, accounts now for more than 80% of all browsing activity on the world wide web [15]. TLS works in tandem with the Web public key infrastructure (PKI), which provides digital certificates issued by certificate authorities (CAs). These digital certificates bind for a specified period of time a digital identity to a public key. These certificates are signed and thus given a seal of ″trust″ by a certificate authority. However, due to for example compromised private keys, security incidents at the CA or decommissioned servers, certificates can become untrustworthy. To keep those relying on the certificate's security safe, a certificate can be revoked.

1.2 The Need for a New Revocation Check Protocol

A variety of methods exists to check the revocation status of a certificate [12]:

- *Certificate revocation lists* (CRLs) are being regularly published by CAs. A CRL lists revoked certificates and is issued by a CA for the certificates it signed. Downloading a CRL and searching it adds latency to the revocation checking process. As the URL containing the CRL can be unattainable, the connection can time out, or the search might take too long - in case of doubt, TLS will open the intended communication. An adversary controlling the network connection can abuse this so-called fail-open approach for man-in-the-middle attacks.
- *The Online Certificate Status Protocol* (OCSP) sends a verification request per certificate to the OCSP responder address. This method leaks a user's web activity and has reliability concerns. Latest Firefox telemetry data show it times out in 7% of the cases [11].
- *OCSP staple* staples the OCSP responder response to the certificate to remove privacy concerns. However, the response can be stripped off and revoked certificates might appear non-revoked.
- *OCSP must staple* alerts TLS to the fact that a revocation status update is to be expected. Hosts are slow and few in supporting this protocol, and therefore it cannot be used for all certificate revocation checks.
- *CRLSet* is an approach used by Google. Research done in 2017 shows that only 14,436 of 12 million revoked certificates were contained in CRLSet. Sending all 12 million revoked certificates with 110 bits per revocation would have resulted in a 166 MB file[12].

Larisch et al. concluded in 2017 [12] and so does van der Merwe at Mozilla's security conference in 2019 [18], that the current procedures are either slow, privacy-sensitive and/or unreliable. Most importantly, they use a fall-back option of "fail-open" (but for OCSP must staple, which is not widely usable). "Fail-open" means that when in doubt about the revocation status of a certificate, the intended communication will be opened. Fail-open undercuts the main reason for using TLS, and therefore a new approach offering "fail-closed" is of interest.

CRLite shall encompass all certificates and offer a "fail-closed" approach [12]. It does so by

1. compressing certificate information with cascading Bloom filters into a data structure (we will refer to it as CRLite BFC) that according to the claims made by its proposers in [12] is easy to verify and easy to update,

2. using a specific algorithm (we will refer to it as CRLite algorithm) in combination with CRLite BFC to check the revocation status of all certificates conclusively,
3. making a new monthly CRLite BFC and daily delta updates available for download and thus allow a fail-closed approach.

1.3 The Focus of this Thesis

As described above, a key challenge of all approaches that help check the revocation status of a certificate is reliable and real-time availability of information. As none of these approaches can guarantee it, TLS suffers from fail-open as the fall-back option. Larisch et al. claim that their CRLite, consisting of CRLite BFCs, their regular updates and the use of the CRLite algorithm, will support a fail-closed manner for revocation checks as a compressed Bloom filter cascade with all valid certificates is pushed to the user and thus available in real-time to the relying party [12]. This Bachelor Thesis analysis the vulnerabilities of CRLite. It does so by looking at vulnerabilities of CRLite as such and potential adversarial forces in the certificate ecosystem. It then details different threat models and assesses their probability of success.

1.4 The Structure of the Thesis

- **Chapter 2 - Background: Bloom Filter and Bloom Filter Cascades** establishes the used notation and gives background information. It will introduce the reader to Bloom filters and cascading Bloom filters, with a focus on how they are used in CRLite BFCs in particular. It also contains an explanation of CRLite. This chapter summarizes also the mathematical model built for this thesis to simulate CRlite BFCs and delta updates.
- **Chapter 3 - The TLS Certificate Ecosystem** presents research on the certificate ecosystem. It documents key players, the volume and evolution of certificates and summarizes potential effects on CRLite.
- **Chapter 4 - CRLite Vulnerabilities** describes the aspects of CRLite that make it vulnerable to attacks beyond those addressed in the original paper [12]. We show how we used the CRLite BFC model explained in chapter 2 to simulate different potential scenarios and document its findings. We detail the experiments conducted for a selected number of these scenarios and how we verified agreement between the model and the experiments. We conclude with a summary of our findings.

- **Chapter 5 - Exploitation of Vulnerabilities** contains potential threat-models to assess to which extent a deployed version of CRLite could be subverted.
- **Chapter 6 - Conclusion** summarizes the findings. It concludes the thesis and provides ideas for future work.

Chapter 2

Background: Bloom Filters and Bloom Filter Cascades

Bloom filters are the basis for Bloom filter cascades. In this chapter, we will therefore give first an introduction to Bloom filters and introduce their notation. We will then explain the main mathematical formulas used to optimize Bloom filters. The focus is on those formulas that are needed for the analysis of the CRLite BFC.
We then explain Bloom filter cascades with an immediate focus on CRLite BFC. We introduce the necessary notations and formulas.
Finally, we will present two algorithms explicitly written for this thesis: the first algorithm simulates a CRLite BFC and outputs key performance indicators, such as space requirement. The second algorithm simulates the consequences of certificate revocation and outputs whether an update of CRLite BFC will allow the functioning of the CRLite algorithm. We conclude with findings on those factors that are key for the functioning of an updateable CRLite cascade.

2.1 Bloom Filters

2.1.1 Notations for Bloom Filter

A Bloom filter is a probabilistic data structure with constant lookup times. When defining or analyzing a Bloom filter, the following variables and ratios are used:

- *the filter* itself, b, that is a bit array of individually addressable indices. The variable m is used to denote its length. The filter size m defines the required space and/or needed bandwidth if transmitted. When space or bandwidth for transmission is at a premium, one wants to minimize m.

- The set N contains the elements to be stored in the Bloom filter. The size of N is n.
- Bloom filters work with hash functions. These hash functions are applied to each of the n elements. The output of each hash function is in [0, m - 1] or, when using MurmurHash3, is taken modulo m to be in [0, m - 1]. These results per element are an element's respective indices in the filter. The standard analysis of Bloom filters assumes that the resulting mapping is uniformly at random, that is the probability of one of the m indices being set to 1 is $\frac{1}{m}$. The number of used hash functions is denoted by k.
  The following figure shows how elements are inserted in a Bloom filter. It also explains how a Bloom filter is used to check whether an element is in the filter.

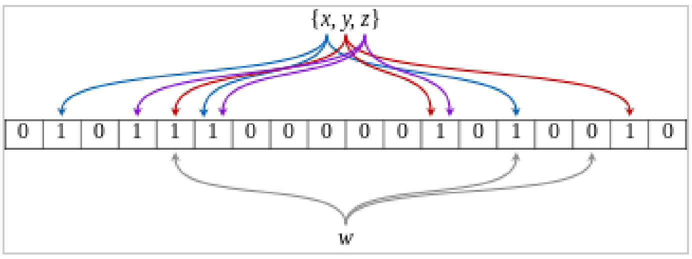


Figure 2.1: This example shows a Bloom filter of size m=18. The elements x, y, and z are inserted. Three hash functions are used to define the indices per element. The three coloured arrows show the results per element. These indices are now set to 1. If one now wants to verify whether an element is in the filter, one has to hash this element three times. If at least one of these indices is 0, one knows for sure that this element cannot be in the filter. This is the case for element w. If all its arrows were to point to a 1, it would be a false positive (figure taken from [16]).

- Bloom filters have no false negatives but can have false positives. The probability of an element that is *not* in N to be considered an element of N is called the false positive rate p. The false positive rate is sometimes also called the error rate. If false positives need to be managed, p is the ratio of those elements that will have to be treated separately.
- Bloom filter performance and vulnerabilities depend also on the hash algorithm used. A non-cryptographic hash-function or truncated hashes have known vulnerabilities [7]. CRLite uses MurmurHash3 [12]. MurmurHash3 is widely used for its speed and good distribution, however, it is a non-cryptographic hash function.
- Bloom filters do have a weight w. w is the fraction of the m indices of a

Bloom filter set to 1 [14]. We use it to analyze delta updates.

When a Bloom filter is created, one defines the relation of m, p and k to n. A Bloom filter can be created for a given m or for a given p. The formulas used for optimal results are given in 2.1.3. Initially, all m indices of the Bloom filter are set to 0. To insert an element into the filter, it has to be hashed k times. Equally, to query if an element is present in the filter one has to hash it also k times. If at least one hash value points to 0, then the element is for sure not one of the inserted elements. However, if all of the hash results point to indices that are 1, then it might be in the filter. The likelihood of this happening is the false positive rate p. If this false positive rate is either negligible or known **and** manageable and if storage and bandwidth consumption are at a premium, Bloom filters are widely chosen as storage solution due to their space efficiency [3]. Mitzenmacher went as far as to stipulate a Bloom filter principle:

> ”Whenever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated” [4]

As we will see when talking about Bloom filter cascades in Section 2.2, chaining together several Bloom filters, each containing the prior false positives, is a way to mitigate the false positives while profiting from Bloom filter space efficiency.

2.1.2 Dynamic Data Sets - Dynamic Bloom Filters

Dynamic data sets bring their own challenges to Bloom filters. While static data sets will, as their name suggests, stay constant in size and composition, dynamic data sets evolve.

- Growing sets result in the insertion of new elements. This leads to either an increasing false positive rate or requires a new, bigger Bloom filter and thus rehashing of all elements. In our context growing sets may arise if for example in a given period the number of newly revoked certificates is larger than the number of expiring certificates.

- Shrinking sets, on the other hand, do not lead to a shrinking false positive rate. Without special provisions, Bloom filters do not support the deletion of elements. Fan et al. are accredited to be the first solving this challenge, conceiving the counting Bloom filter [8]. In the context of CRLite we will not look into counting Bloom filters due to their added space requirements. The question of how to deal with a set with fewer elements than initially stored is relevant for CRLite as certificates do have a limited lifetime.

- The dynamics of a data set can also combine both aspects, with some elements being taken out and others added. As a consequence, a stable set size does not imply a stable Bloom filter, as each element should have a unique hash fingerprint, and thus different indexes will be set to 1.

A Bloom filter with a dynamic data set has to monitor changes and actively maintain its usefulness. As a consequence of Bloom filter changes in size and/or composition one either has to create a new Bloom filter or update it with new insertions. Another option are the proposed delta updates by the authors of [12]. They define it conceptually as creating a bitwise XOR by comparing the filter with the updated content with the prior filter. We will look into the dynamics of certificates in Chapter 3 and its consequences for CRLite in Chapter 4.

2.1.3 Optimal Bloom Filters

As said before, Bloom filters are chosen for their space efficiency and their constant lookup times. Accordingly, when creating a Bloom filter, one wants to optimize the space (m) needed, the times an element has to be hashed (k) and p, the false positive rate. As stated before, p defines the number of elements whose status might be falsely identified as positive and one either ignores or one has to take care of by other means. The following graph shows how the ratio of $m : n$ influences the false positive rate and the required number of hash functions:

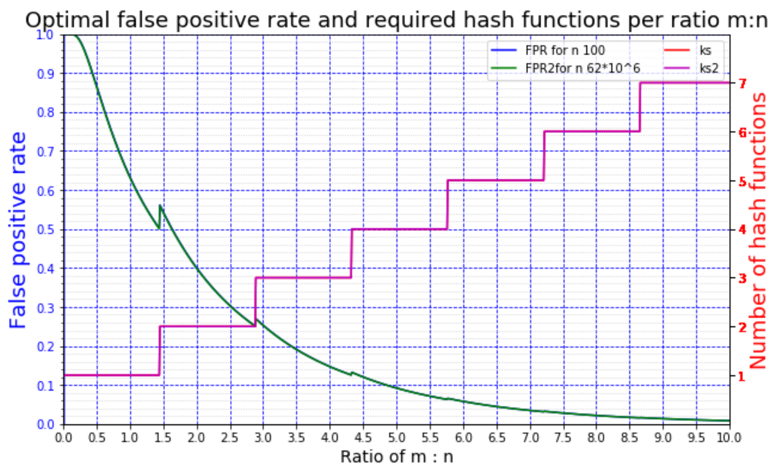


Figure 2.2: The bigger the ratio *m:n* the lower the false positive rate under the condition that k increases. As k has to be an integer, some false positive rates can be obtained with two different *m:n* ratios. This fact can be used when minimizing the size of Bloom filters

The mathematics to arrive at optimum *m, k* and *p* have been widely researched and documented. The following is an excerpt of established findings, adapted from [4]. For the sake of brevity, we omit proofs.

False Positive Rate

The formula most widely used is based on the probability that an index of a Bloom filter is one after k hashes and n insertions:
The probability of changing an index of the filter to 1 equals the probability that a specific bit of the m indices in the bit array is addressed by the first hash of an element: $\frac{1}{m}$. Thus the probability it still being 0 is: $(1-\frac{1}{m})$. After k hashes with independent and perfectly uniformly at random distribution hash functions its probability of being 0 is: $(1-\frac{1}{m})^k$. The probability of a bit still being 0 after all n elements have been inserted is, under the assumption that $k*n \ll m$: $(1-\frac{1}{m})^{k*n} \approx e^{\frac{-k*n}{m}}$. It follows that the expected fraction of m-indexes with value 1 after the insertion of all n elements, also known as w, is [14]:

$$w = 1 - (1 - \frac{1}{m})^{k*n} \tag{2.1}$$

The probability that each of the hash results of an element that is not in the filter points to an index set to 1 is therefore w to the power of k:

$$p = w^k = (1 - (1 - \frac{1}{m})^{k*n})^k \approx (1 - e^{\frac{-k*n}{m}})^k \tag{2.2}$$

These formulas are based on the following assumptions:

1. $k*n \ll m$.
2. the k hash functions are independent, and each hash function has truly uniformly at random results.
3. a potential adversary is not able to influence the stored data nor the used hash-function (that is the non-existence of chosen-insertion attacks)

Optimal Number of Hash Functions

The optimal number of hash functions for a Bloom filter of a given size m and a fixed set N of size n is derived from an optimal false positive rate:

$$\begin{aligned} p &= (1 - e^{\frac{-k*n}{m}})^k \\ \frac{\partial \ln p}{\partial k} &= \ln(1 - e^{\frac{-k*n}{m}}) + \frac{k*n}{m} * \frac{e^{\frac{-k*n}{m}}}{(1 - e^{\frac{-k*n}{m}})} \\ k &= \frac{m}{n} * \ln 2 \end{aligned}$$

As we see, the optimal number of hash functions for an optimal false positive rate depends on the ratio *m:n*. At an optimal *k* the resulting false positive rate *p* is:

$$p = \frac{1}{2}^{\ln 2} * \frac{m}{n}$$

Optimal Size of a Bloom filter

To arrive at a minimal size for a given minimal *p* one can use the above formula for minimal *p*. After applying the natural logarithm to the above function and making some simplifications one arrives at:

$$m = -n * \frac{\ln p}{\ln 2^2} \tag{2.3}$$

Thus *k* can be expressed in relation to *p* using the formula of the optimal *m*:

$$k = -\frac{lnp}{\ln 2} = -\log_2 p \tag{2.4}$$

These results use the approximate formula for *p*. Goe and Gupta are cited in various publications for their proof that the approximate formula for $(1 - \frac{1}{m})^{k*n}$ is only off by at most half an element [16].

2.1.4 Vulnerabilities of Bloom Filters

Research in the usability of Bloom filters in a given security context concludes that the chances for attacks and thus for vulnerabilities stem initially from incorrect design choices potentially due to a lack of adversary models [9], [2].

Misused hash functions, chosen for speed instead of cryptographic performance, for example, increase the likelihood of false positives and thus for attack likelihood [9].

The ability of an adversary to make the false positive rate of a Bloom filter higher than the expected one is another vulnerability [9]. The false positive rate of a Bloom filter can be increased by inserting new elements. Doing so with malicious intent is called **chosen insertion attack**. Chosen insertion attacks range from **pollution attacks**, the adversary can set several bits of the Bloom filter to one, to **saturation attacks**, the adversary can set all bits of a Bloom filter to one [9]. These attacks result in a rising number of false positives and worst case only false positives making the Bloom filter useless.

A Bloom filter with zero false positives due to the addendum of another data structure can be the target of a Query-only-Adversary [9]. Such an adversary floods a Bloom filter to have an increasing number of false positives that have to query the other data structure. The goal is to make such a proposal either expensive or infeasible or both.

With that we conclude the section on Bloom filters, having established the used notation, the importance of distinguishing between static and dynamic sets, the four key formulas, 2.1, 2.2, 2.3 and 2.4 that will be used in the coming analysis, and an overview of Bloom filter vulnerabilities.

2.2 Bloom Filter Cascades for TLS Certificates

Chazelle et al. have shown how several Bloom Filters can be used with the false positives of one Bloom filter acting as a trigger for a recursive Bloom filter approach to achieve a data structure that can establish the set membership of each checked item [6]. This approach profits from the space efficiency of Bloom filters while at the same time mitigating the false positives that they produce. This idea of achieving zero false positives by using the false positives of one Bloom filter to trigger the creation of another Bloom filter is also at the core of CRLite and presented in [12].

From this point forward, we will focus our discussions only on the CRLite Bloom filter cascade (CRLite BFC).

2.2.1 Optimal CRLite BFC

In the following we describe CRLite BFC as defined in [12]:

CRLite addresses the set of all valid certificates, which we will denote as C. Valid certificates are unexpired certificates. Within this set, there are two distinct subsets: the set R of valid-revoked certificates and the set S of valid-non-revoked certificates. The size of these sets is respectively denoted as r and s. Each certificate of C is by the nature of revocation exclusively in either R or S.
CRLite's goal is that there is a definite correct answer for each checked element of C of either valid-revoked or valid-non-revoked. The following shows how the CRLite BFC is constructed to attain this goal:

1. The authors of [12] incorporated into their definition of CRLite BFC the fact that C is a dynamic set and that both R and S are subject to change. To assure that a CRLite BFC can aggregate all valid certificates during a certain period of time without resizing, CRLite BFC is not created for the current sizes of R and S but for r_c and s_c. r_c is defined as: $r_c > maximum(r, r * \delta_r)$ with δ_r the expected growth rate of r. s_c is

defined as: $s_c > maximum(s, s * \delta_s)$ with δ_s the expected growth rate of s.

2. The CRLite BFC consists by the very definition of a Bloom filter cascade of several Bloom filters. In the case of CRLite BFC, there is one Bloom filter at level 0 and at each other level. The false positives of level 0 trigger the creation of another Bloom filter whose respective false positives trigger another Bloom filter. For ease of distinction, the resulting additional Bloom filters are numbered starting with 1 and referred to as belonging to a level with that number. As CRLite BFC has Bloom filters per level, we introduce a new notation to ease of further discussion, b_i denotes the Bloom filter at level i. m_i, n_i, k_i, w_i and p_i are the Bloom filter variables of the respective level.

3. The CRLite BFC stores all valid-revoked certificates at level 0. The Bloom filter at that level is optimized for a given false positive rate, p_0 and size r_c. Using the established formula for optimal Bloom filter, the size of the Bloom filter at level 0 is: $m_0 = -r_c * \ln(p_0) * \ln(2)^{-2}$. The number of optimal hash functions at level 0 k_0 is defined as:
$k_0 = \frac{m_0}{r_c} * \ln(2)$.
The authors of [12] define $p_0 = \frac{r_c}{s_c} * \sqrt{p_i}$. p_i is given both as 0.5 and as empirically verified for a specific data set R and S as more optimal when set to 0.5099.

4. Due to this choice for p_0, as long as $r_c > 0$ there will be false positives, namely $f_0 = s_c * p_0 = r_c * \sqrt{p_i}$ many. As there are more than zero false positives, a new Bloom filter is needed containing these false positives. The CRLite BFC begins to grow by adding level 1. The Bloom filter at level 1 is optimized for p_i and the false positives of level 0, that is f_0. Accordingly, we have the following parameters for that level: $m_1 = -f_0 * \ln(p_i) * \ln(2)^{-2}$. The number of optimal hash functions at level 1 k_1 is defined as $k_1 = \frac{m_1}{f_0} * \ln(2)$.

5. At level 1, one has to check whether elements of level 0 are now erroneously identified as elements of level 1. These false positives are mitigated at the next level. As of level 1, the elements inserted in the prior level are checked if they are false positives at that level. The formulas used to parameterize the Bloom filters at all additionally levels are those of level 1. Instead of f_0, the false positives of the respective prior level are used. If there is no more false positive element to be stored, the CRLite BFC ends.

6. Once CRLite BFC is created for r_c and s_c, R and S are inserted. The following figure explains how this is done:

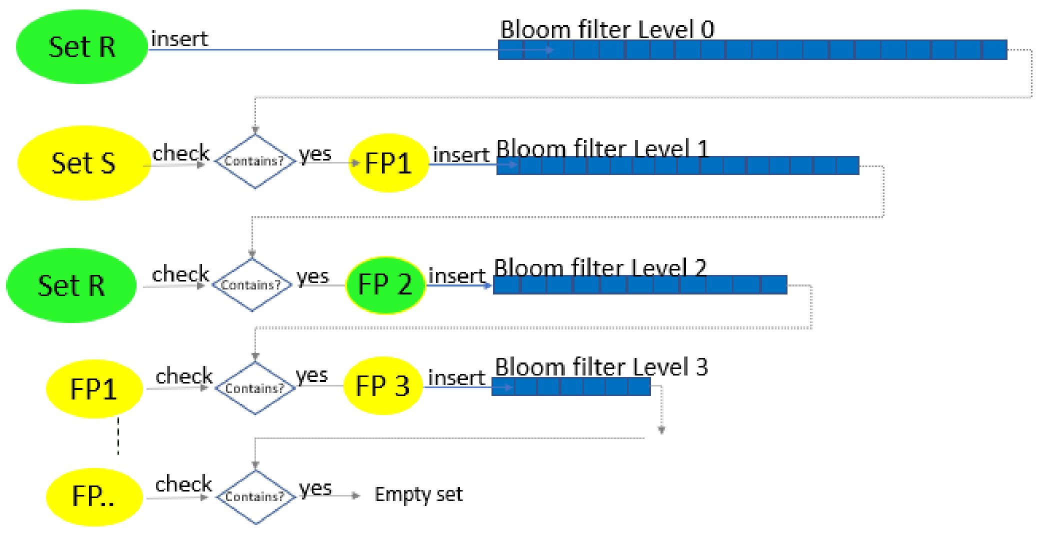


Figure 2.3: CRLite takes the set R of revoked certificates and fills at level 0 the Bloom filter. The valid-non-revoked certificates, set S, are checked against Bloom Filter level 0. Only those that test positive are collected in FP 1 and inserted in the next Bloom Filter. The cascade from now on always check the elements from the prior level. Only the remaining false positives will be stored. This process continues until there are no more false positives.

This concludes the description of how to create a CRLite BFC.

In the following, we summarize all resulting notations and introduce an additional index. Larisch et al. assumed that the release of a new CRLite BFC would happen monthly and daily delta updates were to assure the usefulness between two new release [12]. We therefore introduce t as a second index with $t \in \mathbb{N}$. $t = 0$ indicates a new CRLite BFC that is pushed to web browsers or made available for download. CRLite BFCs with $t > 1$ denote those CRLite BFCs that are built as delta updates to reflect the changes between t and $t + 1$.

2.2.2 Notations for CRLite BFC

t:	Time, $t \in \mathbb{N}$
i:	Level in the cascade Bloom filter, $i \in \{0, 1, ..\}$
$R_{t,i}$	set of revoked, non expired certificates at time t, at level i
$r_{t,i}$	size of $R_{t,i}$
$S_{t,i}$	set of valid certificates at time t, at level i
$s_{t,i}$	size of $S_{t,i}$
r_c	, s_c sizes used to create the Bloom filter
δ_r	, δ_s expected respective growth rates
$N_{t,i}$	set of items that need to be stored at time t, at level i
$n_{t,i}$	size of $N_{t,i}$
$F_{t,i}$	set of false positives that need to be inserted at time t, at level i
$f_{t,i}$	size of $F_{t,i}$
$m_{t,i}$	number of indexes of a Bloom filter at time t, at level i; also called the size of a Bloom filter level at time i.
$w_{t,i}$	number of Bloom filter indexes with value 1 at time t, level i. It is calculated as a function of $m_{t,i}$, $k_{t,i}$ and $n_{t,i}$: $w_{t,i} = (1 - (1 - \frac{1}{m_{t,i}})^{k_{t,i} * n_{t,i}}) * m_{t,i}$
b_i	Bloom filter at level i
p_0	false positive rate for level 0 defined at time t = 0, $p_0 = \frac{r_c}{s_c} * \sqrt{p_i}$
p_i	false positive rate for all level $i \neq 0$, defined at time $t = 0$
$p_{t,i}$	error rate, also called false positive rate, at time t, level i $p_{t,i} = (1 - (1 - \frac{1}{m_{t,i}})^{k_{t,i} * n_{t,i}})^{k_{t,i}}$
$k_{t,i}$	number of hash functions at time t, level i

2.3 CRLite: CRLite BFC, Delta Updates and CRLite Algorithm

In this chapter, we explain how CRLite uses the CRLite BFCs (for details see Section 2.2.1) to achieve a fail-closed solution for certificate revocation checking. At first, we describe the steps that result in up-to-date CRLite BFCs for web browsers and then explain its usage by the CRLite algorithm [12].

2.3.1 CRLite BFC and Delta Updates

CRLite needs to minimize the size of its solution to assure that the CRLite BFC is downloaded and updated in a timely manner. Therefore, Larisch et al. differentiated between a server side and a client side in [12]. The server side generates the CRLite BFC and the delta updates. These delta updates are according to [12] incremental in size of a CRLite BFC and are downloaded by the client side. The alternation between CRLite BFC and delta updates involves the following distinct steps:

- **Collect C**: All valid TLS certificates have to be processed. Among other information, their serial number, CAs and referenced Certificate Revocation List (CRL) or OCSP responder information are collected and the revocation status checked. This information is used to create the distinct sets R and S. Both Larisch et al. and Mozilla have documented this process [12] [11]. The challenges and vulnerabilities of this process are not part of our analysis.
- **Define the parameters**: As stated in Chapter 2, one sizes the CRLite BFC at least as large as one expects the sets R and S to be at the next release date and at least as large as they currently are. As a result, CRLite BFC parameter r_c and s_c are defined.
- **Create the filter**: As described in Section 2.2.1 the CRLite BFC is sized for the above mentioned parameters. Then the valid certificates in R are inserted in the CRLite BFC at level 0. Figure 2.3 illustrates this process for level 0 and further levels. The process of identifying and storing false positives of one level at the next level continues until there are no more false positives [12]. This process takes place both for the creation of CRLite BFC and the delta update.
- **Release a new CRLite BFC or a delta update**: Due to changes in S and R the filter has to be updated at regular intervals. A release of a new CRLite BFC means that a complete CRLite BFC has to be pushed to or downloaded by all users. According to the authors of [12] a new release of the CRLite BFC is made available nonthly and in place of a delta update if at the time of a delta update either $r > r_c$ or $s > s_c$. If neither r nor s trespass their respective upper boundaries,

delta updates are expected to be incremental and sufficient to establish reliable information [1]. Delta updates only change the content of a CRLite BFC but can neither increase the size of the used Bloom filters nor add further Bloom filters nor will a delta update change the number of hash functions used. The timestamp of the CRLite BFC is the date of its creation or that of the latest downloaded delta update.

2.3.2 CRLite Algorithm

CRLite includes an algorithm that checks the revocation status of a looked-up certificate.

1. When looking-up a certificate, the CRLite algorithm only checks valid certificates, expired certificates will be discarded.
2. A certificate with a validity start date before the CRLite BFC timestamp is checked within CRLite BFC.
3. Starting from level 0 the algorithm checks whether the certificate is contained in the Bloom filter of that level. As long as the certificate is contained, the algorithm continues to check the next level.
4. The moment the certificate is not any more contained, the algorithm stops and reads out the number of the level it stopped at. If it stops at an odd level, then the certificate belongs to R and is revoked. If it stops at an even level, then the certificate belongs to S and is not-revoked.
5. Certificates with validity dates after the CRLite timestamp are checked via the method appropriate for that certificate and preferred by the web browser.

2.3.3 CRLite Principles

The following characteristics of CRLite have been extracted from the paper by Larisch et al. [12]. "CRLite principles" summarizes the CRLite design parameters and evaluation based recommendations by the CRLite authors:

- **CRLite Principle 1 - Purity**: CRLite BFC consists purely of Bloom filters.
- **CRLitePrinciple 2 - Priority S:** CRLite BFC is built with the look-up experience in mind. Web browsers need to load those pages fast, that have valid non revoked certificates. All elements contained at level 0 will have to be at least checked in two levels in CRLite BFC, these are the valid-revoked certificates. Certificates in S are mostly resolved at

[1]Larisch at al. recorded in their experiments only rarely situations of large delta updates [12]

level 0, only the false positives need more than 2 checks.
This decision is also very favourable to the overall size of CRLite BFC. By nature $r \ll s$. Therefore it is more space efficient to obtain a small false positive rate by oversizing m for r than it would be for s.

- **CRLite Principle 3 - Zero False Positives:** A Bloom filter at level i+1 contains the false positives of level i: $n_{t,i+1} = f_{t,i}$. There are as many Bloom filters as needed to resolve all expected false positives for a defined size of r_c and s_c.

- **CRLite Principle 4 - Minimize Space:** CRLite BFC is built using two false positive rates in deviation from the approach used by Chazelle et al. [6] to achieve a minimal size CRLite BFC. There is one false positive rate for all levels beyond the first one: p_i. The false positive rate for the first level, level 0, is p_0. p_0 is defined as: $p_0 = \sqrt{p_i} * \frac{r_c}{s_c}$ Best results are obtained by initializing p_i with a value in the proximity of 0.5. The optimized p_i according to experiments for a given r and s is 0.5099.

- **CRLite Principle 5 - Periodicity:** CRLite BFC shall be released in defined intervals. In between these release dates "delta updates" shall bring the currently downloaded CRLite BFC version, in reflection also of principle 1, to become a CRLite BFC version of the same size with the new reality of R and S.

- **CRLite Principle 6 - Version Proof:** In 2017 as is the case now in 2020, the number of valid certificates was constantly growing. Therefore, the sizes used to construct CRLite BFC are: $s_c > max(s, \delta_s * s)$ and $r_c > max(r, \delta_r * r)$

- **CRLite Principle 7 - Release Imperative:** If at any time between two release times either the set R or the set S reaches sizes that are bigger than the respective r_c and s_c of the current version, a new CRLite BFC has to be created and pushed to web browsers.

- **CRlite Principle 8 - Use Proven Formulas:** The following formulas for optimal Bloom filters with $n_{t,i}$ the size of the to be inserted set N at time t and at level i of the Bloom filter are used:
  $m_{t,0} = -r_c * \ln p_0 * \frac{1}{\ln 2^2}$
  $m_{t,i} = -n_{t,i} * \ln p_i * \frac{1}{\ln 2^2}$ for $i \neq 0$
  $k_{t,i} = \frac{m_{t,i}}{n_{t,i}} * \ln 2$

- **CRLite Principle 9 - Fail-Closed:** With the use of CRLite BFC and the CRLite algorithm 2.3.2 the status of any certificate can be determined at any time.

2.4 CRLite BFC Evaluation

As we want to analyze the vulnerability of CRLite, we need to be able to assess CRLite BFC, which is at the heart of CRLite. Therefore we created two mathematical simulation models, one for CRLite BFCs and the other for delta updates. Each model consists of an algorithm programmed in Python using anaconda jupyter notebooks. In the following, we will explain each model.

2.4.1 Mathematical Evaluation of CRLite BFC

We created an algorithm that simulates CRLite BFC at full capacity with the formulas given in Section 2.2.1. The algorithm takes r_c, s_c and p_i as input values and creates a CRLite BFC. It outputs per level of the resulting CRLite BFC the size it is constructed for, the size of the Bloom filter, the number of hash functions and the false positive rate as a result of rounding m and n to integer values.

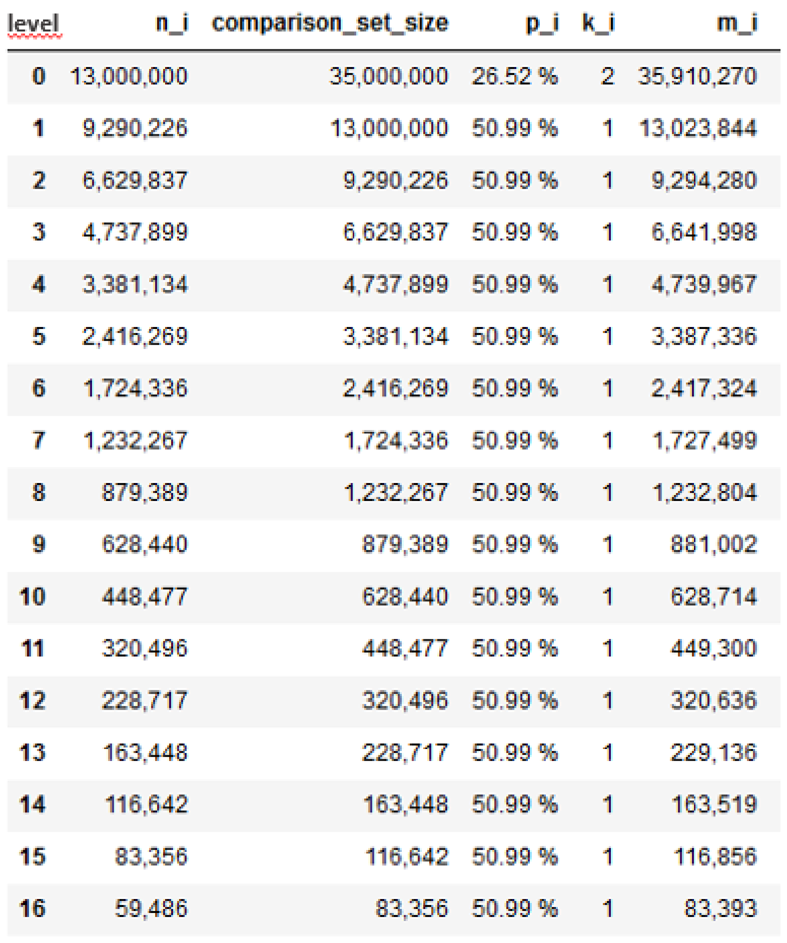
In the following we will first show the pseudocode of this algorithm and then illustrate it with an excerpt from one of the evaluations run.

We realized that Bloom filter sizes got ridiculously small the higher the level in the CRLite BFC, but the authors of [12] had explicitly stated that they used an identical rate for all levels and that was the result.

Algorithm 1 Mathematical Model of a CRLite BFC

```
 1: procedure BFCascade($r_c, s_c, p_0, p_i, M$)
 2:     $level = 0$                          ▷ Start with the Bloom filter containing R
 3:     $m_0 \longleftarrow computeM(r_c, p_0)$
 4:     $k_0 \longleftarrow computeK(m_0, r_c)$
 5:     $p_{0c} \longleftarrow computeP(m_0, k_0, r_c)$                    ▷ $p$ in expectation
 6:     $M[0] \longleftarrow (r_c, s_c, p_0, k_0, m_0)$          ▷ Collect information per level
 7:     $f_0 \longleftarrow computeF(s_c, p_{0c})$   ▷ Number of false positives in expectation
 8:
 9:     $level = level + 1$
10:     $n_i \longleftarrow f_0$
11:     $m_1 \longleftarrow computeM(n_i, p_i)$
12:     $k_i \longleftarrow computeK(m_1, n_i)$
13:     $p_{1c} \longleftarrow computeP(m_1, k_i, n_i)$
14:     $M[1] \longleftarrow (n_i, r_c, p_i, k_i, m_1)$
15:     $f_i \longleftarrow computeF(r_c, p_{1c})$
16:
17:     $level = level + 1$
18:     while $f_i > 0$ do
19:         $n_i \longleftarrow f_i$
20:         $m_i \longleftarrow computeM(n_i, p_i)$
21:         $k_i \longleftarrow computeK(m_i, n_i)$
22:         $p_{1c} \longleftarrow computeP(m_i, k_i, n_i)$
23:         $M[i] \longleftarrow (n_i, r_c, p_i, k_i, m_i)$
24:         $potFP \longleftarrow M[level - 1][n_i]$              ▷ Potential false positives
25:         $f_i \longleftarrow computeF(potFP, p_{ic})$
26:         $level = level + 1$
27:     end while                      ▷ Algorithm stops when zero false positives
28:     return $M$         ▷ M contains the Bloom filter parameters for all levels
29: end procedure
```

Here an example for $r_c = 13$ Million, $s_c = 35$ Million and $p_i = 0.5099$.



level	n_i	comparison_set_size	p_i	k_i	m_i
0	13,000,000	35,000,000	26.52 %	2	35,910,270
1	9,290,226	13,000,000	50.99 %	1	13,023,844
2	6,629,837	9,290,226	50.99 %	1	9,294,280
3	4,737,899	6,629,837	50.99 %	1	6,641,998
4	3,381,134	4,737,899	50.99 %	1	4,739,967
5	2,416,269	3,381,134	50.99 %	1	3,387,336
6	1,724,336	2,416,269	50.99 %	1	2,417,324
7	1,232,267	1,724,336	50.99 %	1	1,727,499
8	879,389	1,232,267	50.99 %	1	1,232,804
9	628,440	879,389	50.99 %	1	881,002
10	448,477	628,440	50.99 %	1	628,714
11	320,496	448,477	50.99 %	1	449,300
12	228,717	320,496	50.99 %	1	320,636
13	163,448	228,717	50.99 %	1	229,136
14	116,642	163,448	50.99 %	1	163,519
15	83,356	116,642	50.99 %	1	116,856
16	59,486	83,356	50.99 %	1	83,393

Figure 2.4: This table shows an excerpt of our model's CRLite BFC for $r_c = 13$ Million, $s_c = 35$ million, $p_i = 0.5099$. The chosen data was taken from [12]. The output lists per level the number of expected elements "n i", the potential false positives "comparison set size", the false positive rate "p i", the number of hash functions used "k i" and the size of the Bloom filter "m i".

Mathematical Evaluation of CRLite BFC Delta Updates

The following algorithm was created by us to simulate a specific CRLite BFC delta update situation which is part of our vulnerability analysis in Chapter 4: We analyze a scenario with a CRLite BFC at full capacity. A delta update has to be created due to new revocations. To create a delta update, the CRLite BFC is filled with the changed R and S sets, following the procedures outlined in [12]. Our goal is to evaluate whether the CRLite BFC containing the new data were still to have zero false positives and if not, how many in expectation.

In the following we describe how we programmed this algorithm:

- This algorithm takes as input r_c, s_c, p_i and a value for the number of newly revoked certificates, xx. As this algorithm is simulating a delta update, the size of the Bloom filter and the number of hash-functions per level are those of the prior CRLite BFC. This algorithm therefore uses the above described algorithm to obtain per level the size of the Bloom filter and the number of hash-functions: $m_{1,i} = m_{0,i}$ and $k_{1,i} = k_{0,i}$.
- At level 0 the following variables are initiated: $r_{0,0} = r_c$, $r_{1,0} = r_{0,0} + xx$ and accordingly, $s_{0,0} = s_c$ and $s_{1,0} = s_{0,0} - xx$.
  It is then verified that $r_{1,0} > r_c$ and that $s_{1,0} < s_c$. Furthermore, it must hold that $r_{1,0} + s_{1,0} = r_{0,0} + s_{0,0}$.
- As stated earlier, the size of the Bloom filter and number of hash functions cannot be adapted for a delta update. The fixed size, fixed number of hash functions and the new number of inserted elements, results in a new false positive rate: $p_{1,0} = (1 - (1 - \frac{1}{m_{0,0}})^{k_{0,0} * r_{1,0}})^{k_{0,0}}$.
- For level 1, the new number of false positives of the prior level, level 0, is calculated: $f_{1,0} = p_{1,0} * s_{1,0}$. These false positives are the number of elements that we expect at level 1: $n_{1,1} = f_{1,0}$. If $n_{0,1} \neq n_{1,1}$ the false positive rate at level 1 changes, too:
  $p_{1,1} = (1 - (1 - \frac{1}{m_{0,1}})^{k_{0,1} * n_{1,1}})^{k_{0,1}}$
- The steps of level 1 are repeated for all further levels until at least one of the following is true:
  1. $n_{1,i} = 0$
  2. $m_{0,i} = 0$: as a delta update cannot add another Bloom filter, the algorithm has to stop here. It returns the number of elements currently false positives and those that are potential false positives.
  3. $w_{1,i} = 100\%$ and $n_{1,i} = n_{1,i-2}$: the Bloom filter at that level was not capable to further reduce the number of false positives. As higher levels have Bloom filters of smaller size, the situation cannot improve. Therefore, the algorithm returns the number of false positives at that level and potential false positives.

This mathematical model can be run for any size of R, S and number of revocations. We will use it in Section 4.3.5.

Chapter 3

The TLS Certificate Ecosystem

CRLite, if implemented as proposed in [12], will aggregate all valid certificates in its CRLite BFC and update it continuously to assure its usability. Therefore, this chapter provides a summary of the TLS certificate ecosystem, focusing on its size and significant forces, and will asses the consequences for CRLite.

3.1 Valid certificates in numbers

In 2017 Larisch et al. reported 30 million valid-non-revoked and 12.7 million valid-revoked certificates, a total of 42.7 million valid certificates [12]. In January 2020, there are more than 210 million valid certificates [5]. This growth is due to an increase in valid-non-revoked certificates as the number of valid-revoked certificates has shrunk to less than 1% of the total number.

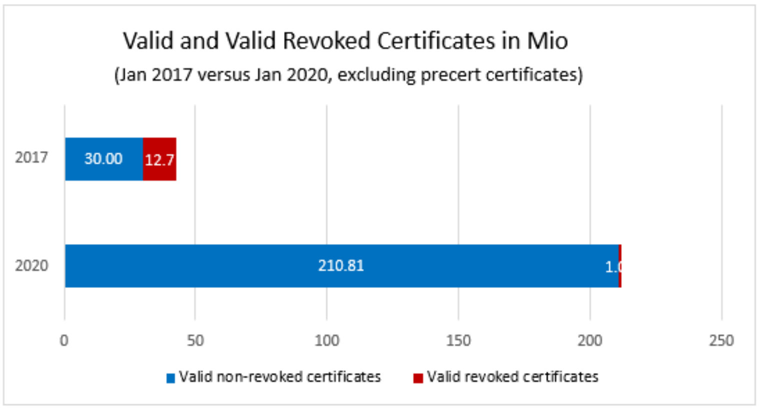


Figure 3.1: In the period 2017 - 2020, the set of valid-non-revoked certificates has grown to seven times its original size. In the same period the set of valid-revoked certificates has shrunk to less than one tenth of its original size [12], [1], [11].

New certificates are being issued as the uptake of HTTPS increases. Let's Encrypt documents the importance of HTTPS for transmitting web traffic and reports the share of pageloads by Firefox using HTTPS [15]. The following Figure 3.2 shows hat HTTPS will remain one of the drivers for expected future growth in certificates.

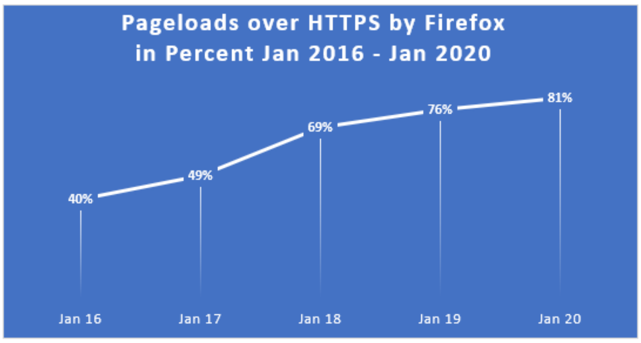


Figure 3.2: In the span of four years the share of pageloads over HTTPS by Firefox has more than doubled [15].

One of the other forces driving certificate number growth is Let's Encrypt. Let's Encrypt changed the certificate ecosystem by offering its services as a CA free of charge. It accounts currently for more than 50% of all valid certificates. The following graph shows the other four major players who between them cover further 35% of all valid certificates.

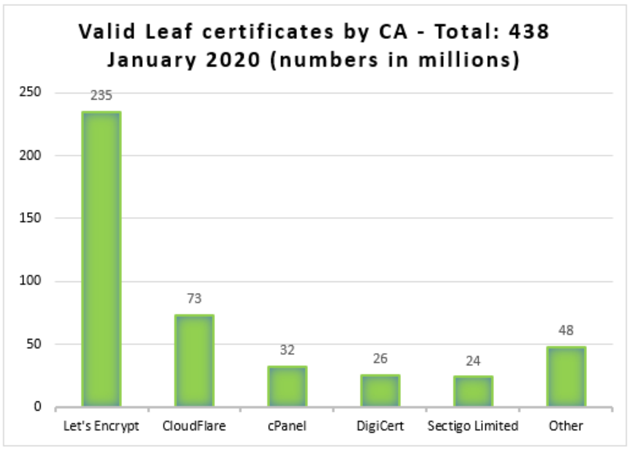


Figure 3.3: Five players hold approximately 90% of the market. Estimates for the total number of CAs range between 200 and 300 accounting for 9% of the market [5].

Let's Encrypt's influence on CRLite BFC will be considerable, should an issuer of CRLite BFCs opt to include it. Its certificates have a maximum lifetime of 90 days and a target to reduce it to 60 [1]. The resulting downside for CRLite BFCs is that the composition of the set of valid-non-revoked changes considerably in a single day. On average more than 1.1 million certificates could expire in a single day. The daily rate of issued certificates varies considerably too, with a 35% spike possible.

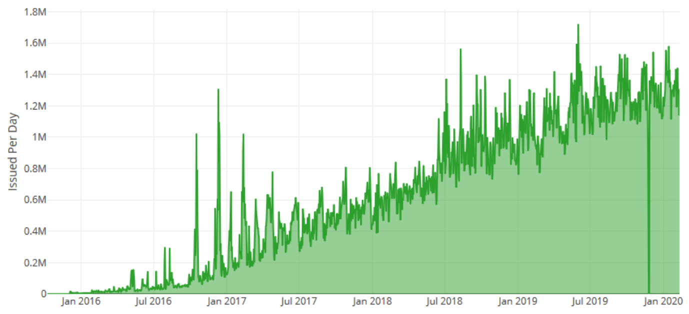


Figure 3.4: This chart shows the variability in the number of daily newly issued certificates and that Let's Encrypt can handle extreme variances in newly issued certificates [15]

An excerpt taken from this data shows the challenge to correctly size CRLite BFC for a given period of time, see Figure 3.5:

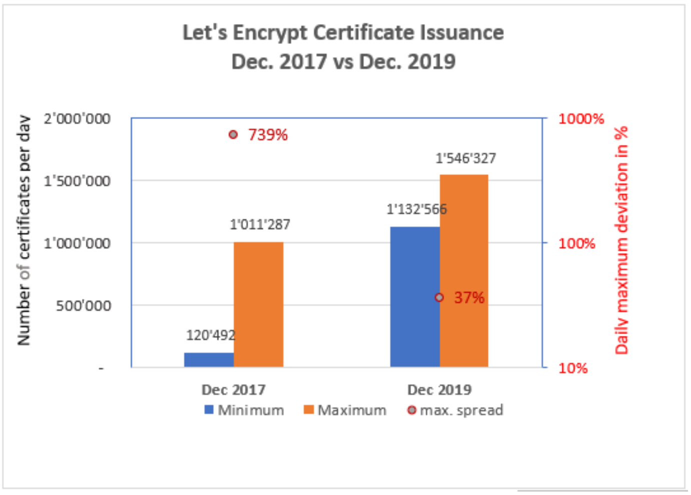


Figure 3.5: In December 2017, the maximum and minimum number of daily newly issued certificates deviated by 739%. Three years later, deviations of 37% are observed. Daily rates of more than 1 million newly issued certificates are not an exception anymore [15].

3.2 Consequences for CRLite

As was shown in the prior Section the TLS certificate ecosystem of 2020 is considerably different from 2017 in size and composition. The relevant learning for the vulnerability analysis of CRLite is:

1. R is currently at approximately 0,5% of the volume of valid certificates and less than one tenth the reported numbers of 2017. The size of R predominantly influences the size of CRLite BFC.
2. S has exploded in the last 3 years. Further growth is to be expected and leads to a growing CRLite BFC.
3. Validity periods will shrink leading to a larger number of expired certificates per day and thus more changes in one CRLite BFC delta update.
4. The daily number of newly issued certificates varies, sometimes dramatically, and puts in question whether sizing CRLite BFC for the expected market growth is optimal.
5. Five companies control nearly 90% of all certificates. Their business models have consequences for CRLite. We note that as of January 2020 Mozilla has excluded all CAs that do not publish certificate revocation lists, including Let's Encrypt [11] which plans to issue CRLs as early as potentially summer 2020 [1]. In Section 5.1.2, we will discuss one potential scenario and its consequences for CRlite.

Chapter 4

CRLite Vulnerabilities

When thinking about potential CRLite vulnerabilities, one has to consider Bloom filter vulnerabilities, as CRLite BFC is a Bloom filter cascade. However, sometimes vulnerabilities of an integrated element can be mitigated by the system using it. Therefore, we will focus in our analysis on CRLite first and will highlight unmitigated CRLite BFC vulnerabilities as we go deeper.

Due to CRLite's positioning as established in Chapter 1, given our understanding of CRLite as detailed in Chapter 2 and the influences from the TLS ecosystem as summarized in Chapter 3, we highlight specific areas of concern:

A **Collecting and aggregating the information** on all valid certificates is the cornerstone of CRLite. If data is not collected in a timely and correct manner, CRLite can suffer from outdated or missing certificates, a low rate of resolution or even worse, wrong answers. The process of collecting data and assuring its completeness and/or making provisions for not accounted for data is therefore key. As with all solutions offering insights based on aggregated data: "garbage in, garbage out". Larisch et al. have documented their approach, including independent aggregation auditing [12]. Mozilla is currently actively and publicly working on this area [10]. Up-to-date information can be found at: `https://github.com/mozilla/crlite/`.

B **The CRLite BFC** is sized to be as minimal as possible and at the same time large enough so that daily delta updates can keep an already downloaded CRLite BFC valid for a month. Delta updates instead of daily CRLite BFCs make CRLite due to their incremental size attractive and feasible. We will show that sizing CRLite BFC for expected market growth results in a higher than anticipated occurrence of CRLite BFC downloads in the place of delta updates. This vulnerability by design can be mended with the recommendations made in 4.1

C **The choice of hash-functions** and input information has to keep in mind the needs of CRLite BFC (random distribution) and the needs of the user, who when loading a web page will not react favourably to lengthy background calculations. The risks incurred when using non-cryptographic or truncated hash functions have already been the focus of research [13]. The specific risks for CRLite using certificate data, public keys and/or serial numbers as the choice of input in combination with MurmurHash3 is interesting but out of our current scope.

D **CRLite is positioned as a fail-closed solution** and Mozilla hopes it can replace OCSP with it [10]. Therefore we will analyse under which assumptions and restrictions CRLite is a failed-closed solution. We consider CRLite's fail-closed solution compromised if at least for one certificate that is included in CRLite BFC the revocation status cannot be resolved correctly by the CRLite Algorithm but could be if using CRL or OCSP.

E **CRLite is aggregating a dynamic data set.** The size of R and S depend on revocations and newly issued certificates for growth. Nearly any domain owner can get a certificate and certificate owners can unilaterally demand the revocation of a certificate. Therefore we want to see, whether chosen insertions or revocations can influence CRLite unfavourably.

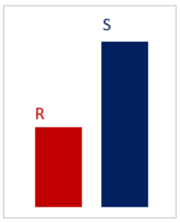
4.1 Sizing the CRLite BFC

We will show two faults in the design of CRLite that can result in a new CRLite BFC instead of delta updates.

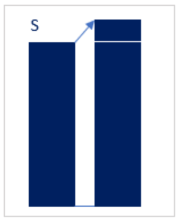
4.1.1 Defining the CRLite BFC Parameters

We recapitulate from Section 2.2.1, that a CRLite BFC is sized for r_c and s_c and only then filled with the certificate data. By slightly oversizing the CRLite BFC Larisch et al. assumed that a CRLite BFC would be large enough for one month, the time before the next CRlite BFC [12]. We will analyze whether this assumption holds.

In the period between two CRLite BFCs newly issued certificates grow the set S. We showed in Chapter 3, that the demand for certificates continues and that considerable differences in the daily numbers of newly issued certificates can occur.



(a) Time 0



(b) The effect of newly issued certificates - set S increases

Figure 4.1: Status at release time and after newly issued certificates

The set R can only grow if valid-non-revoked certificates get revoked. Therefore, if a certificate is revoked, set S shrinks and set R grows. Currently, approximately 0.3% of Let's Encrypt's valid certificates and approximately 0.6% of those certificates aggregated by Mozilla in CRLite are revoked [1] [10]. R is currently much smaller than S.

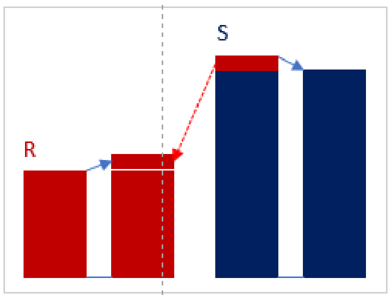


Figure 4.2: Revocation

Certificates have validity start and end dates. As we have seen in Chapter 3, more than 50% of today's certificates are only valid for 90 days. Therefore, it is possible that more than 1 million certificates expire on a single day, even more in the future. Expiring certificates is the only way that set R can shrink.

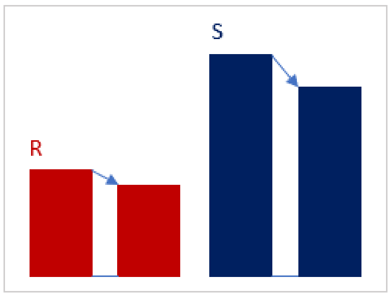


Figure 4.3: Expiration

With the following scenarios, we want to show that it is not sufficient to know, how many certificates expire and how many newly issued and newly revoked certificates one expects. By totalling these numbers and setting s_c or r_c slightly above the expected size, one would assume evolution in certificate numbers as shown in Scenario 1. However, we have seen in Chapter 3 that for example, newly issued certificate numbers within a month can hugely differ. A scenario 2 can happen.

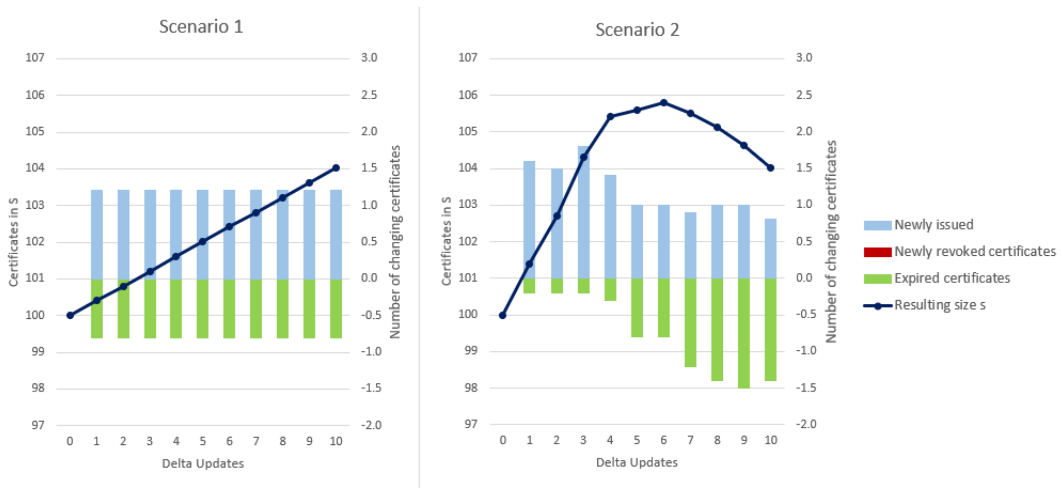


Figure 4.4: If one were to set s_c at 105 million valid-non-revoked certificates, scenario 1 shows one way how the certificate market could evolve without ever surpassing this number. In scenario 2, due to the independent fluctuations in expiring and newly issued certificates, s_c is surpassed.

Conclusion concerning s_c and r_c This example shows, that by following CRLite Principle 6, CRLite BFCs are by design potentially too small. Too much data in a Bloom filter leads to higher false positive rates and more false positives. In the case of CRLite BFC this leads potentially to more than zero false positives, as we will show in Section 4.3. More than zero false positives mean the CRLite Algorithm cannot resolve the certification status of all certificates and therefore instead of a delta update a new CRLite BFC has to be created.
We recommend strongly to remedy this vulnerability. CRLite Principle 6 should be changed to reflect the uppermost size during the validity time as a minimum size requirement besides the actual set size.

4.1.2 Higher than Expected False Positives

In this section, we address another fault in the design. As noted in Section 2.1.3, the formulas used to size the filter assume a uniform at random distribution of insertions, but MurmurHash3 is not known to be 100% uniform at random [17]. As a consequence, the number of indices set to 1 in a Bloom filter at any level of a CRLite BFC might be a bit higher or lower than expected. In Bloom filter terms, the observed weight of the Bloom filter is lower or higher than expected. This results in a higher false positive rate and thus more false positives that need to be inserted at the next level, potentially leading to more than zero false positives. In the following, we explain the mechanism that can lead to such a situation:

1. We want to show a deviation from the uniform at random expected placement of 1s in a Bloom filter. Let therefore $w_{observed}$ be the observed weight of the Bloom filter at level 0 at time 0.

2. At time 0 CRLite is constructed with p_i, r_c, s_c according the CRLite principles. This results in $p_0 = \frac{r_c}{s_c} * \sqrt{p_i}$.

3. We introduce δ_w: it captures the deviation from the observed to the expected weight: $\delta_w = \frac{w_{observed}}{w_0}$. Thus we can write:
   $w_{observed} = \delta_w * w_0$

4. The false positive rate is the weight of the Bloom filter to the power of the number of hash functions used. Therefore: $p_0 = (w_0)^{k_0}$.
   This can be written as: $w_0 = \sqrt[k_0]{p_0}$

5. Applying the formula for false positive rates we obtain the observed false positive rate: $p_{observed} = (w_{observed})^{k_0}$ which can be written as:
   $p_{observed} = (\delta_w * w_0)^{k_0}$

6. We conclude with the resulting deviation of the false positive rate δ_p:
   $\delta_p = \frac{p_{observed}}{p_0}$ and insert $p_{observed}$
   $\delta_p = {\delta_w}^{k_0}$

The number of additional false positives that need to be stored in expectation at level 1 is therefore $s * (\delta_w)^{k_0}$. k_0 depends in CRLite BFCs on the ratio of r:s. Therefore, the smaller this ratio, the bigger k and the bigger the effects a small deviation at level 0 can have on all other levels.

Conclusion When filling a CRLite BFC for the first time and / or during a delta update one has to pay attention that the weight per level the CRLite BFC was built for is not surpassed. Otherwise, even though s and r are smaller than respective s_c and r_c, the number of false positives per level may rise so much that the CRLite BFC has more than zero false positives and and will potentially lead to unexpected behaviour of the CRLite Algorithm.

4.2 CRLite Fail-Closed

CRLite is considered a fail-closed solution and for Mozilla, as quoted in Chapter 1, is an option to replace OCSP. In the following, we want to question how "fail-closed" CRLite can actually be in comparison to OCSP. To be fail-closed CRLite has to be able to resolve all look-up queries in a timely and correct manner. As CRLite BFC is at the user side in combination with the CRLite Algorithm, CRLite is according to [12] faster than CRL and OCSP and more reliable.

We assume that CRLite is implemented as described in [12]. We consider a scenario in which a certificate that is in the CRLite BFC gets revoked. This information is available as CRL and OCSP. The CRL is only published each seventh day [1] but is also available via OCSP [5]. In the strictest sense of fail-closed CRLite has to be as informative as OCSP, that is once the revocation information is available via OCSP it also has to be contained in the CRLite BFC. As a result, a delta update or a new CRLite BFC would have to happen whenever a revocation occurs. If not, the CRLite algorithm uses the "old" and thus wrong CRLite BFC and answers wrongly with valid-non-revoked as valid certificates queries are resolved with the CRLite BFC.

The necessity for timely delta updates or a new CRLite BFC in the face of revocations makes CRLite potentially vulnerable to "Update"-adversaries. Such adversaries want to force the server side to keep sending delta-updates to maintain the quality of the CRLite BFC by revoking valid certificates.

4.3 Targeted Revocation of Certificates

In this section, we analyze whether a CRLite BFC is vulnerable if certificates are deliberately released and / or revoked, as these are the two ways the data set of valid certificates can be influenced by a third party. This would be the case if such actions were to result in a CRLite BFC that has more than zero false positives. Such a situation would necessitate a new CRLite BFC and its download instead of a delta update. If this were not to happen in a timely manner, CRLite fail-closed would lead to wrong answers or an error by the CRLite algorithm as we will show.

We recapitulate from former sections, that delta updates can neither change the size nor add further Bloom filters to the existing CRLite BFC. They are necessary to maintain a CRLite BFC usefulness and are used instead of new CRLite BFCs due to their incremental size. Repeated use of CRLite BFCs instead of delta updates will negatively influence the size of CRLite and can make it less useable.

[1]Digicert's CRL is published weekly as seen the 23rd of February

To assess whether such a vulnerability exists, we will first establish how R and S can change between two CRLite BFC downloads and identify the most vulnerable constellation. We will then analyze this constellation with our evaluation model and verify our mathematical findings in experiments.

4.3.1 R and S constellations

Both R and S can either shrink, stagnate or grow. These three distinct states for both sets result in 9 potential constellations.

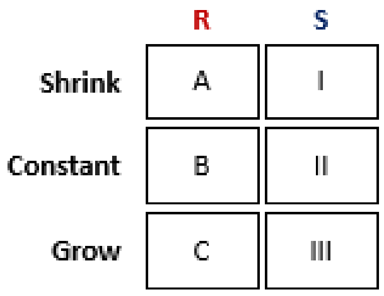


Figure 4.5: R and S constellations

Column R summarizes the following potential evolution of R:

- The number of newly revoked certificates being smaller than the number of expired is state C, R shrinks (A).
- If the number of expired certificates equals the number of newly revoked, R stays constant, state B.
- Set R grows if the number of newly revoked certificates is larger than the number of expired (state C).

Column S stands for the following change scenarios of set S:

- Set S shrinks if the number of newly issued certificates is smaller than the sum of newly revoked and expired certificates, state I.
- A constant size is the result of the sum of newly revoked and expired certificates equaling the number of of newly issued certificates, state II.
- Set S grows if the number of newly issued certificates is larger than the sum of newly revoked and expired certificates, state III.

We now analyze these combinations from the perspective of whether their occurrence could result in a CRLite BFC that has more than zero false positives. This assessment will then be tested.

Combinations A I, A II, B I and B II result in both r and s being smaller than their respective r_c and s_c used to create the CRLite BFC. Shrinking datasets will lead to lower false positive rates and therefore cannot lead to an increase of false positives. Constant sets will in expectation have constant false positive rates and no change in number of false positives.

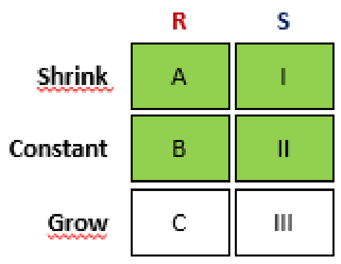


Figure 4.6: CRLite update can proceed

Combinations A III, B III will potentially lead to growing false positive rates throughout the CRLite BFC triggered by a growth in level 1. This constellation can lead to more than zero false positives.

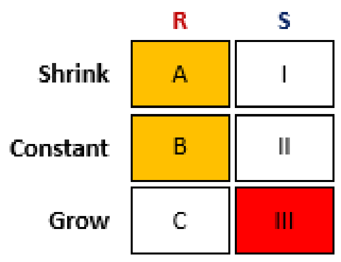


Figure 4.7: Growing S combined with shrinking and stagnating R

Combinations C I, C II occur if valid-non-revoked certificates in S are being revoked by their owners and at the same time the set of valid-non-revoked certificates is not growing. This constellation results in an increase of the false positive rate at level 0 and can lead to more than zero false positives.

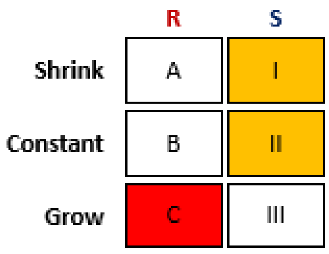


Figure 4.8: Growing R combined with shrinking or stagnating S

Combination C III means that even though the set S shrinks as valid-non-revoked certificates are being revoked, the number of newly issued certificates is higher than the sum of expired and revoked certificates. This situation can lead to more than zero false positives.

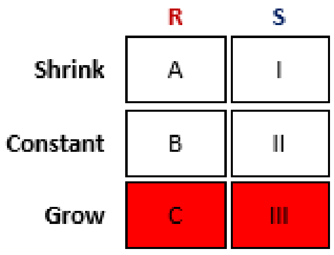


Figure 4.9: Growing R combined with growing S

Assessment of Constellations

Five constellations can potentially lead to more than zero false positives: C I, C II, C III, A III and B III.
A growing S bigger than s_c can be the result of spikes or wrong predictions but also the result of pollution attacks, a problem intrinsic to dynamic Bloom filters whose input cannot be regulated. As mentioned before, a growing S is only possible if new certificates are being issued. Certificates issued after a release are dealt with outside CRLite. Potentially, if there are no new revocations, sitting out an update might be an option as the currently downloaded CRLite BFC contains all relevant information.

A growing R, however, is only possible as the result of new revocations. If these revocations concern a certificate whose validity start date is before the last release, than delta updates have to happen. If one doesn't change the CRLite BFC by inserting the newly revoked certificate at level 0 and changing subsequent levels, than the CRLite Algorithm cannot correctly resolve the revocation status of this certificate and will give a wrong answer. So, using

CRLite one must update CRLite BFC in delta updates to reflect the correct revocation status.

We will therefore analyze C I. If C I with a shrinking set S can lead to more than zero false positives, than C II and CIII will definitely, too.

4.3.2 Revocation as a Source of CRLite Vulnerabilities

As established in 4.3.1, if revocations of certificates with validity dates covered by CRLite BFC occur, then the CRLite BFC has to be updated. Such certificates have to be removed from S and added to the set of R. We will now evaluate how a CRLite BFC at full capacity is affected by such a situation.

In the following sections, the used parameters and assumptions of this evaluation are summarized. We then show the predictions of the mathematical model described in 2.4. We verified these predictions with experiments, whose set-up and result we will explain in a dedicated section. We finalize with our assessment on CRLite vulnerabilities due to revocations in 4.3.7.

4.3.3 Analysed Scenario

We assume that at $t = 0$ the CRLite BFC is at full capacity, $r = r_c$ and $s = s_c$. At $t = 1$ is the next delta update. In the period between $t = 0$ and $t = 1$ only revocations happen. We assume that there are no expiring valid-revoked nor valid-non-revoked certificates. Also, there are no newly issued valid-non-revoked certificates. Of the existing s_c certificates $\boldsymbol{x}$ **are being revoked**.
We further assume that the CRLite BFC is as stated in Section 2.3.3: the CRLite BFC contains all valid-revoked and all valid-non-revoked certificates at $t = 0$. All certificates, whose validity date start before $t = 1$, are by design resolved by the CRLite algorithm by checking their revocation status with the CRLite BFC. As stated earlier in the thesis, it does so by querying the CRLite BFC level after level until it reaches a level in which the looked up certificate is not any more contained. It returns the number of the "not-contained-anymore" level.
We assume that CRLite uses MurmurHash3 as this was the choice by Larisch et al [12].

4.3.4 Tested Assumptions

We assume that by trespassing r_c, one of the sizes the CRLite BFC was constructed to hold, the CRLite BFC will have more than zero false positives.

If this occurs than the CRLite BFC cannot be updated with a delta update as the CRLite Algorithm is built on the assumption the CRLite BFC do have zero false positives. The CRLite Algorithm cannot handle the situation where it arrives during a query at the last level and the queried certificate is in this filter. The CRLite Algorithm returns as described in 2.3.2 only if it arrives at a level where the queried certificate is not contained. Therefore, if a query leads to a situation where the CRLite Algorithm cannot return such a number, its behaviour is undefined.

4.3.5 Evaluation Results

These are the results we obtained when we tested how resilient the CRLite BFC would be in expectation to revocations. We ran the evaluation for $r = 10.000$ and a range of values for s.

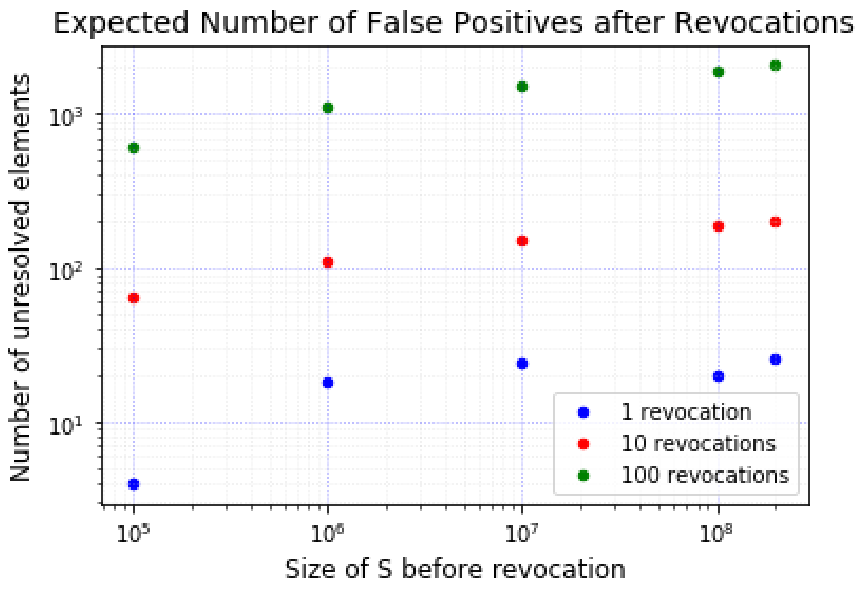
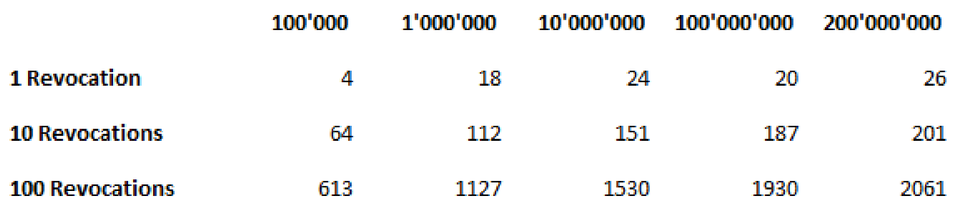


Figure 4.10: False Positives in a CRL BFC per number of revocations shown in relation to size of S. For $r = 10.000$ all tested combinations resulted in more than zero false positives.

Our model predicts that already one revocation results in a delta update with more than zero false positives. The number of expected false positives increases the bigger the size of S in relation to the size of R is and the more elements are revoked.



	100'000	1'000'000	10'000'000	100'000'000	200'000'000
1 Revocation	4	18	24	20	26
10 Revocations	64	112	151	187	201
100 Revocations	613	1127	1530	1930	2061

Figure 4.11: The table gives the numbers used for the above graph. It show the results per size of S and per revocation for r = 10.000.

We verified these and other results in experiments, see Section 4.3.6

4.3.6 Experiments

We wrote a program to verify our mathematical results for the scenario in 4.3.3. It consists of a module creating a CRLite BFC and a module to create a delta update. Each module takes as input any set of data stored in csv format. The number of experiments can be freely chosen. This algorithm supports any size of R and S. One can choose how many elements of the set S one wants to revoke, this number has to be of course smaller than the size of S.

The program's output is a list of all run experiments sorted by the size of simulated revocations and contains the following information:

- result: Results are either 1, 2, 3 or 4.
  - Result 1: CRLite at $t = 1$ has zero false positives.
  - Result 2: CRlite at $t = 1$ has zero false positives and needs less levels than at t = 0
  - Result 3: CRLite at $t = 1$ has one or more false positives at the last level
  - Result 4: CRLite at $t = 1$ has one or more false positives at the last level. It is a special case of Result 3. Result 4 signals that already at levels before the last level the CRLite BFC stopped working. In a CRLite BFC this is the case if the number of false positives is not reduced by adding more levels.

- number of not resolvable certificates: a query of these certificates in the updated CRLite BFC will not arrive at a level at which these certificates are not anymore contained. This means the CRLite Algorithm will not return for these certificates. This number is the sum of valid-revoked and valid-not-revoked false positive certificates at the end of the CRLite BFC due to the delta update.
- level: this indicates the level at which $w_{1,i} > 99\%$. Once the weight is so high, our experiments showed that further reduction in false positives occur rarely. We added this output to assess the degree of vulnerability of the CRLite BFC per number of revoked certificates.

CRLite BFC Program for Experiments

At first, we will show the pseudocode for the module that created a CRLite BFC for given sets R and S and $p_i = 0.5099$ and its main methods. Then we explain the setup of the second module. B is the structure that will contain the resulting CRLite BFC.

Algorithm 2 Create a CRLite BFC

```
procedure BFCascade($R, S, p_0, p_i, B$)                ▷ CRLite works with two p rates
    $r \leftarrow length(R)$
    $s \leftarrow length(S)$
    $level = 0$                                         ▷ Start with the Bloom filter containing R
    $m_0 \longleftarrow computeM(r, p_0)$
    $k_0 \longleftarrow computeK(m_0, r)$
    $b_0 \longleftarrow initializeb(m_0)$
    $b_0 \longleftarrow hashandinsert(R, b_0, k_0)$
    $B \longleftarrow append(b_0)$
    $F_0 \longleftarrow hashandcollect(S, b_0, k_0)$    ▷ Collect false positives of S
    $N_i \longleftarrow F_0$
    $n_i \longleftarrow length(N_i)$
    $level = level + 1$                                 ▷ At level 1: the false positives of S are inserted
    $m_i \longleftarrow computeM(n_i, p_i)$
    $k_1 \longleftarrow computeK(n_i, n_i)$
    $b_1 \longleftarrow initializeb(m_i)$
    $b_1 \longleftarrow hashandinsert(N_i, b_1, k_1)$
    $B \longleftarrow append(b_1)$
    $F_1 \longleftarrow hashandcollect(R, b_1, k_1)$    ▷ Collect false positives of R
    $F_{i-1} \longleftarrow N_i$
    $N_i \longleftarrow F_1$
    $n_i \longleftarrow length(N_i)$
    $level = level + 1$                                 ▷ at level i, elements in i-1 are pot. false positives
    while $n_i > 0$ do
        $m_i \longleftarrow computeM(n_i, p_i)$
        $k_i \longleftarrow computeK(m_i, n_i)$
        $b_i \longleftarrow initializeb(m_i)$
        $b_i \longleftarrow hashandinsert(N_i, b_i, k_i)$
        $B \longleftarrow append(b_i)$
        $F_i \longleftarrow hashandcollect(F_{i-1}, b_i, k_i)$
        $F_{i-1} \longleftarrow N_i$
        $level = level + 1$
        $N_i \longleftarrow F_i$
        $n_i \longleftarrow length(N_i)$
    end while                                           ▷ Algorithm stops when zero false positives
    return $B$                                          ▷ B contains the CRLite BFC
end procedure
```

The method *computeM* uses the following formula, here illustrated for level 0: $m_0 = -r * ln(p_0) * ln(2)^{-2}$. The method *computeK* uses the following formula, here illustrated for level 0: $k_0 = \frac{m_0}{n_0} * ln(2)$

For completeness and ease of understanding, we add the pseudo-code for the two methods needed to fill the Bloom filter at each level.
Method 3 *hashandinsert* takes a set of elements N and inserts them in the bit-array *b* after hashing them *k* times. It returns the filled Bloom filter.

Algorithm 3 Inserting elements in Bloom filter

```
function HASH AND INSERT(N, b, k)            ▷ insert N with k hashes into b
    $m \longleftarrow length(b)$
    for $n_i \in N$ do
        for $j \in range(k)$ do
            $index \longleftarrow (hash_j(n_i)) mod m)$
            $b[index] \leftarrow 1$
        end for
    end for
    return $b$                               ▷ filter b represents now the set N
end function
```

Method 4 *hashandcollect* takes as input a set of potential false positives N and checks if they test positive in the Bloom filter *b*. It returns a set F containing all identified false positives.

Algorithm 4 Identify false positives of one Bloom filter and collect them

```
function HASH AND COLLECT(N, b, k, F)        ▷ identify false positives of N
    $m \longleftarrow length(b)$
    for $n_i \in N$ do
        $positive = True$                    ▷ will only stay true if all hashes point to a 1
        $j = 0$
        while $positive$ and $j < k$ do
            $index \longleftarrow hash_j(n_i) mod m$
            if $b[index] == 1$ then
                $j = j + 1$
            else                             ▷ definitely not a false positive
                $positive = False$
            end if
            if positive then                 ▷ false positives are collected
                $F \leftarrow n_i$
            end if
        end while
    end for
    return $F$                               ▷ F contains now all false positives of N
end function
```

This module results in a CRLite BFC at full capacity. In the second module, we refill a copy of this CRLite BFC with the changed sets R and S: at level 0 we insert the same set of valid-revoked certificates plus the newly revoked certificates, that we choose randomly from the set of valid-non-revoked certificates. The new set S is the old set S minus those certificates now added to set R and its false positives as a result of the changed level 0 are inserted in level 1 and so on and so forth. We then checked for the same results as detailed in 4.3.6.

4.3.7 Findings

We ran more than 1000 experiments on different ratios of *r:s* with 1, 10 or 100 revocations respectively. These experiments lead all to the same conclusion - CRLite BFC is vulnerable to pollution attacks due to their Bloom filter cascade structure. We will explain this finding for a set of 10.000 valid revoked certificates and 1.000.000 valid non-revoked certificates.

The experiments highlight that **CRLite BFC is the victim of the vulnerabilities inherent to Bloom filters**. By inserting more elements in a Bloom filter than it was originally designed for one can change its behaviour, that is its false positive rate. What our evaluations predict in expectation and what our experiments confirm is, that already a minor change in the original false positive rate at level 0 can lead to a chain reaction across the Bloom filter cascade. The results of the three revocation scenarios show a more than 60% chance of this chain reaction resulting in having more than one false positive at the last level of the CRLite BFC.
The following table shows the percentage of experiments that resulted in more than one false positive at the last level per number of revoked elements of S:

number of revoked S	**Percentage of S**	**CRLite BFCs in % with more than Zero False Positives**
1	0,0001 %	64 %
10	0,001 %	83 %
100	0,01 %	100 %

Table 4.1: Percentage of CRLite BFCs with more than zero false positives after a delta update listed per number of newly revoked certificates.

These experiments support the findings of our mathematical evaluation of CRLite BFCs: CRLite BFC is vulnerable due to their dynamic data sets. Even though the size of set S shrinks, the increase in elements at level 0 changes the false positive rate in the delta updated CRLite BFC so much, that also level 1 has more inserted elements. As one increases the magnitude of these changes, the likelihood of a CRLite BFC with more than zero false positives increases. The following graph shows the distribution of the total number of certificates whose revocation status query would result in unexpected behaviour of the CRLite Algorithm if a delta update were sent. These certificates are those who are at the end of the CRLite BFC still false positives.

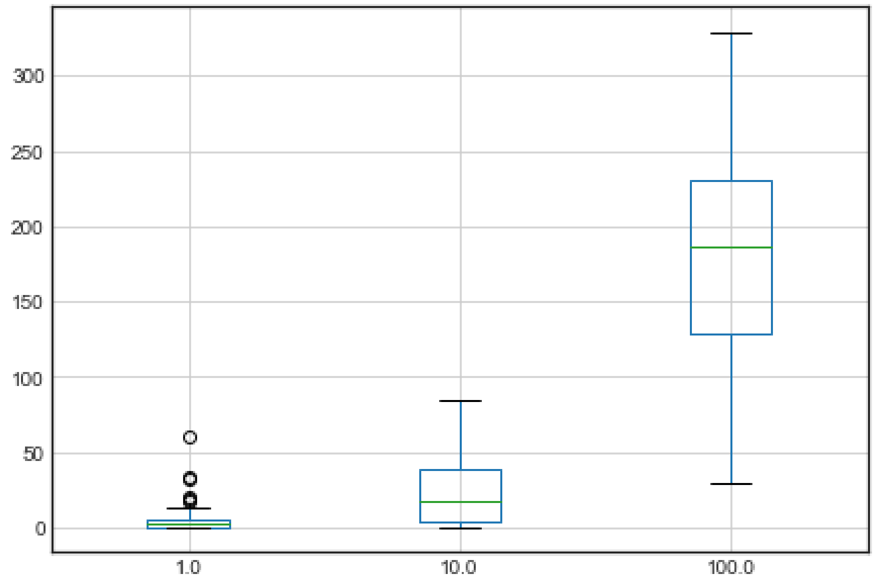


Figure 4.12: Number of False Positives of the CRLite BFC per number of revoked certificates. These are the results for $r_c = 10.000$ and $s_c = 1.000.000$ for 100 experiments.

The numbers indicate that with small effort, e.g. only revoking one element, one can achieve that a delta update of a CRLite BFC results in a multiple of certificates whose look-up query will lead to undefined CRLite Algorithm behaviour if instead of a new CRLite BFC a delta update is made available with this data. This observation is true across the three revocation sizes tested.

number revoked S	**Average Number of False Positives in CRLite BFC**
1	5
10	25
100	183

Table 4.2: Number of False Positives in CRLite BFC after Delta update per number of newly revoked certificates

For further illustration of the problem, the following graph shows the frequency of false positives per valid-revoked and valid-non-revoked certificates. It illustrates that certificates both of R and of S will not be resolvable if a delta update were sent instead of a new CRLite BFC.

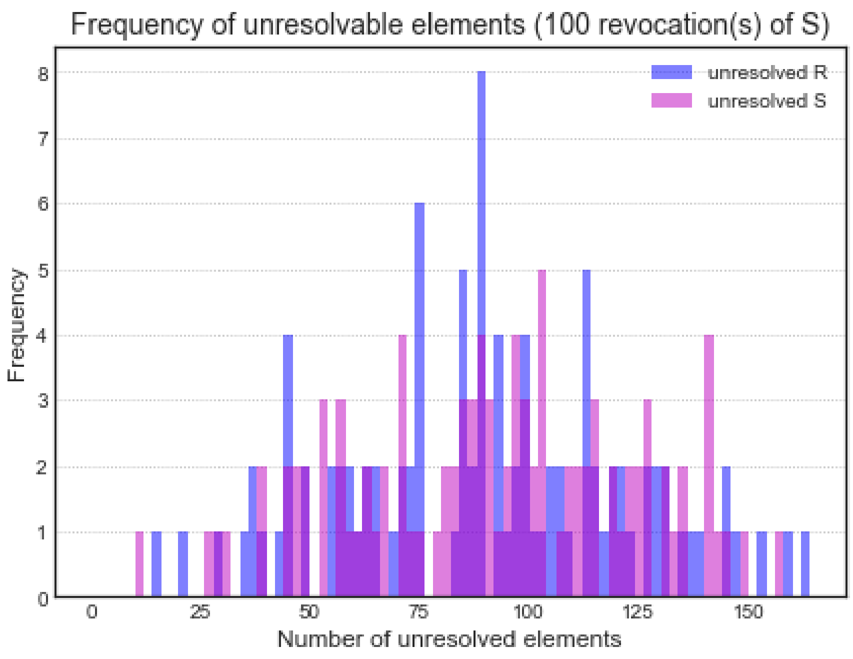


Figure 4.13: Frequency of Number of False Positives of the CRLite BFC differentiated by their membership in R or S. These are the results for $r_c = 10.000$ and $s_c = 1.000.000$ for 100 experiments.

We assume that such a delta update will never be made available and instead a new CRLite BFC would be created and pushed to or downloaded by the client side, the relying party.
Our analysis shows further that the growing number of false positives is due to the fact that the reduction of the size of S is more than compensated by an increase of the false positive rate at level 0. As a consequence, both level 0's and level 1's false positive rate increase, resulting in more false positives to be stored at the next level. Subsequent Bloom filters start to fill up. We observed filters with weights of 100% for delta updated CRLite BFC. In Bloom filters an attack with such a result is called a saturation attack as discussed in Section 2.1.4.

Conclusion and recommendations The experiments we ran confirm the expected behaviour of CRLite BFCs: CRLite BFC are susceptible to potential insertion attack, especially as the cascade structure leads to negative chain effects. As we have shown at full capacity already 1 revocation leads with high probability to a CRLite BFC with more than zero false positives. A delta update is therefore not anymore possible. This vulnerability can be exploited to force unplanned CRLite BFC instead of delta update downloads.

Further analysis of our experiment results showed that the number of false positives is on average higher than the number of newly revoked certificates. The following graph shows how many of those false positives are elements of R. By updating the CRLite BFC the number of resulting false positives from R is on average larger than the number of newly revoked certificates. It might be worthwhile do reconsider the way CRLite BFC is updated.

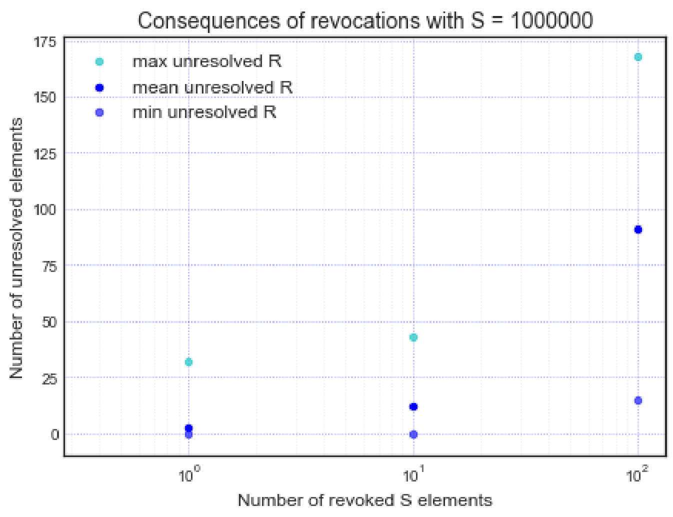


Figure 4.14: Experiment Results: Unresolved R

4.4 Further Concerns

By design, CRLite will resolve most certificates in the top levels. However, those certificates for which the CRLite Algorithm has to traverse the CRLite BFC till the lowest level do have a disadvantage. If CRLite were to become a de facto standard, a top spot in the CRLite BFC could become a concern for web-domains or, alternatively, pushing somebody down the cascade an interest. Targeted pollution attacks might be a potential vulnerability of CRLite and could be explored in the future.

Chapter 5

Exploitation of Vulnerabilities

5.1 Threat Model

In this chapter, we describe the scenario of a potential attack and the adversary's capabilities taking advantage of the detected vulnerabilities.

5.1.1 CRLite Model

We assume that an open source web browser such as Firefox is using CRLite for revocation checking. The parameters used for CRLite are, due to the open source nature of the web browser, publicly available. Therefore the following information is known to an adversary [1]:

- the overall size of the Bloom filter and the number of its levels
- the size of the Bloom filter at each level
- the current weight of the Bloom filter at each level
- the information used to calculate the hashes
- the number and kind of used hash functions at each level
- the name and number of included CAs
- the code used to create CRLite

We assume that CRLite is implemented as suggested in the CRLite paper. There is "cascade inserted information only" both for CRLite BFC downloads and CRLite delta updates. From this follows, that we assume r and s are sized according to expected market growth or at least as big as the current market size. Furthermore, updates are being sent in regular intervals to assure the relevance of CRLite BFC contained information. Updates, given

[1]inspired by [9]

the purity-approach, contain change requests to the existing levels but can neither increase the size m of Bloom filters at selected levels nor add extra levels.

All certificates whose validity date starts between two update times will be verified using CRLite.

Updates are prepared centrally and made available for download at specified intervals. We assume a timespan of one month as cited in the CRlite paper for releases of new versions and daily updates.
We further assume that CTlogs are accessible and valid, CRLs are published at usual intervals and OCSP responders are functioning. In general, we assume that there are no problems registering certificate information nor problems collecting certificate information for sets R and S. We also assume that the aggregator producing CRLite will treat all web sites as equal.

On the user side, we assume that all downloads are successful and the browser software is up-to-date. We assume CRLite to be the default revocation tool used by the browser.

5.1.2 Adversary

We do not consider malicious intent by CAs, CTlog providers, the web browser's employees nor the network used to distribute the updates. Thus we assume all parties are functioning honestly and as expected.

Operation Destabilize

We consider an active adversary that controls an arbitrary number of sites across different regions for which he has the certificate and/or private key. He has the knowledge to understand the workings of CRlite BFC.

Given publicly available information on CRLite he creates a model mirroring the parameters of the CRLite BFC he plans to attack. From the current weight and the known number of hash functions, he calculates the current false positive rate. From the publicly available information on CRLite BFC or via the number of used hash functions and the size of the Bloom filter at the first level he assess the maximum number of valid-non-revoked and valid-revoked certificates CRLite BFC can process without compromise. This is the benchmark he plans to surpass.

From public logs he can establish those certificates that will not expire and using the publicly available CRLite BFC model he can simulate different evolution scenarios. Given these parameters he calculates the number of newly revoked certificates needed to flood one Bloom filter level so that the last level will have false positives. He does all these pre-calculations off-line.

To prevent detection of his malicious intent the adversary revokes the number of needed certificates via a variety of CAs that are part of those CAs covered by CRLite. He does so to avoid potential early warning flags by CAs or refusal of certificate revocation.

This adversary's goal is to undetectedly and repeatedly achieve that the CRLite BFC has more than zero false positives. He wants to disturb CRlite operations.

Certificate Masters

An adversary not necessarily has to have malicious intent. The actions of one party that are disadvantageous and potentially detrimental to another party define such a party as adversary.

Certificate authorities optimizing their business model in the interest of their customers and their market positioning can be an adversary. In this threat model, the Certificate Masters issue certificates with short validity periods for the sake of the security of their customers. They incur a security breach for a portion of their customers. This breach is quickly fixed with revocations and compensations for damages. The portion of newly revoked certificates is 5%. Certificates issued by them are among the Alexa Top 1000. As customers trust these CAs due to the swift and satisfactory response to the problem, none of them leaves this CA.

5.2 Probability of Success and Resulting Risks

5.2.1 Operation Destabilize

Operation Destabilize is both feasible and has a high probability of success. The feasibility assessment is based on:

1. Certificates across a variety of CAs can be acquired without huge resources given Let's Encrypt's business model.
2. All needed resources can be obtained without breaking any laws
3. According to our evaluation tool, with r = 1 Million and s = 200 Millions, at full capacity 10 revocations will in expectation result in 126 false positives instead of zero. We therefore estimate that the number of certificates needed to force a delta update is small.

Operation Destabilize has a great chance of success as CRLite is forced to maintain the quality of its aggregated data. Such an attack will, if carried out as described, force a CRLite BFC instead of a delta-update release.

The resulting risk ranges from a one-off event of CRLite to unpredictable cycles of CRLite BFCs and delta updates. This results in higher complexity for those running CRLite. If such an attack is used repeatedly, the usability of CRLite might be questioned. If the issuers of CRLite were to respond with preemptively oversizing CRLite this might lead to an arms race. Also, oversizing will make the whole CRLite approach bigger in size and thus less use-able.

Operation Destabilize is due to its feasibility and the seriousness of its consequences a potential important risk.

5.2.2 Certificate Masters

Certificate Masters unintentionally effect with their business decisions CR-Lite, already now. Their chance of success is a given.
Any change that shortens certificate lifetime or causes an increase in revocation has a direct negative impact on CRLite as it increases the size of delta updates, as we have discussed. Changes in business model, as also stated in chapter 3, are to be expected and will influence CRLite potentially adversarially.

Chapter 6

Conclusion

The interest in a fail-closed solution for revocation checking is increasing as HTTPS is more widely used in the web. The number of certificates grows in size and patterns of high volatility in daily rates of new certificates become transparent. Few CAs dominate the market and changes in their business practices, such as length of validity and pricing, have a substantial impact on the composition and volatility of valid certificates.

CRLite is positioned as a space-efficient fail-closed solution. It envisions to aggregate all certificate information in Bloom filter cascades that will in regular cycles be distributed as a new CRLite BFC or an incremental delta update. We have distilled the model's formulas and recommendations as CRLite principles. We analyzed those and the steps of the CRLite approach. This analysis was done both with over 1.000 experiments and the use of a mathematical model, programmed for our analysis.

Our analysis finds that some of the principles distilled from the paper lead to vulnerabilities by design, such as sizing for expected market growth or the use of non-random hash-functions. This results in the recommendation to size CRLite for expected maximum outliers during a release cycle plus a mark-up for not-uniformly at random performance of used hash-functions and the variance even ideal hash-functions have. Furthermore, we recommend to study the vulnerability incurred by using MurmurHash3. It is a non-cryptographic hash-function and if a open-source web-browser were to use CRLite the mechanics and parameters of applied hashing would be known to an adversary. A targeted-insertion attack might be feasible.

Lastly, in our experiments and analysis, we reconfirmed that CRLite is only fail-closed if up-to-date revocation data is contained in CRLite BFC, forcing it to represent such information in its regular delta updates or "emergency"

CRLite BFCs. This must-publish makes CRLite vulnerable to saturation attacks abusing, in fact, the inherent vulnerabilities coming with the use of Bloom filters. These attacks can force bigger than expected download sizes and have the potential to influence the functioning of CRLite and of those web browsers using it by default.

We further learned during the experiments that implementing CRLite as summarized in the principles leads to a very long filter cascade. In an information exchange with Mozilla we learned that they cap the length of their CRLite BFCs by setting a minimum size of a Bloom filter at any level of 10.000.

Further research in the following areas is needed:

1. Potential vulnerabilities due to targeted attacks based on MurmurHash3 vulnerabilities
2. Alternative options for CRlite delta updates in between CRLite BFC releases
3. Alternative data structures for the aggregation of the dynamic certificate data.

Bibliography

[1] John Aas. Let's encrypt. Emails to hkarin@student.ethz.ch, 2020 (Email exchange January 2020).

[2] Markku Antikainen, Tuomas Aura, and Mikko Särelä. Denial-of-service attacks in bloom-filter-based forwarding. *IEEE/ACM Trans. Netw.*, 22(5):1463–1476, 2014.

[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[4] Andrei Z. Broder and Michael Mitzenmacher. Survey: Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2003.

[5] censys.io. Certificate data. `https://censys.io`, 2020 (as seen January 2020).

[6] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The bloomier filter: an efficient data structure for static support lookup tables. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 30–39. SIAM, 2004.

[7] Scott A. Crosby and Dan S. Wallach. Denial of service via algorithmic complexity attacks. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003.

[8] Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.

[9] Thomas Gerbet, Amrit Kumar, and Cédric Lauradoux. The power of evil choices in bloom filters. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22-25, 2015*, pages 101–112. IEEE Computer Society, 2015.

[10] J.C.Jones. Crlite end-to-end-design. https://blog.mozilla.org/security/2020/01/09/crlite-part-2-end-to-end-design, 2020 (seen 11th of February 2020).

[11] J.C.Jones. Crlite end-to-end-design. https://blog.mozilla.org/security/2020/01/09/crlite-part-1-all-web-pki-revocations-compressed/, 2020 (seen 11th of February 2020).

[12] James Larisch, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. Crlite: A scalable system for pushing all TLS revocations to all browsers. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 539–556. IEEE Computer Society, 2017.

[13] Jianyuan Lu, Tong Yang, Yi Wang, Huichen Dai, Xi Chen, Linxiao Jin, Haoyu Song, and Bin Liu. Low computational cost bloom filters. *IEEE/ACM Trans. Netw.*, 26(5):2254–2267, 2018.

[14] Michael Mitzenmacher. Compressed bloom filters. In Ajay D. Kshemkalyani and Nir Shavit, editors, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001, Newport, Rhode Island, USA, August 26-29, 2001*, pages 144–150. ACM, 2001.

[15] Firefox telemetrie. Let's encrypt. https://letsencrypt.org/stats/, 2020 (as seen January 2020).

[16] unknown. Bloom filter. https://en.wikipedia.org/wiki/Bloom_filter, 2020 (as seen February 2020).

[17] unknown. Murmurhash. https://en.wikipedia.org/wiki/MurmurHash, 2020 (as seen February 2020).

[18] Dr. Thyla van der Merwe. Bringing academia and industry together. https://www.youtube.com/watch?v=Ult8JPc3rPY, 2019 (seen January 2020).