Design and Implementation of a Compressed Certificate Status Protocol

MICHALIS PACHILAKIS, FORTH-ICS/University of Crete, Greece
ANTONIOS A. CHARITON, University of Crete, Greece
PANAGIOTIS PAPADOPOULOS, Brave Software
PANAGIOTIS ILIA, University of Illinois at Chicago, USA
EIRINI DEGKLERI, University of Crete, Greece
EVANGELOS P. MARKATOS, FORTH-ICS/University of Crete, Greece

Trust in SSL-based communications is traditionally provided by Certificate (or Certification) Authorities (CAs) in the form of signed certificates. Checking the validity of a certificate involves three steps: (*i*) checking its expiration date, (*ii*) verifying its signature, and (*iii*) ensuring that it is not revoked. Currently, such certificate revocation checks (i.e., step (*iii*) above) are done either via Certificate Revocation Lists (CRLs), or Online Certificate Status Protocol (OCSP) servers. Unfortunately, despite the existence of these revocation checks, sophisticated cyber-attackers can still trick web browsers to trust a *revoked* certificate, believing that it is still valid.

Although frequently *updated*, *nonced*, and *timestamped* certificates can reduce the frequency and impact of such cyber-attacks, they add a huge burden to the CAs and OCSP servers. Indeed, CAs and/or OCSP servers need to *timestamp* and *sign* on a regular basis all the responses, for every certificate they have issued, resulting in a very high overhead. To mitigate this and provide a solution to the described cyber-attacks, we present CCSP: a new approach to provide timely information regarding the status of certificates, which capitalizes on a newly introduced notion called *Signed Collections*. In this paper, we present in detail the notion of *Signed Collections* and the complete design, implementation, and evaluation of our approach. Performance evaluation shows that CCSP (i) reduces space requirements by more than an order of magnitude, (ii) lowers the number of signatures required by six orders of magnitude compared to OCSP-based methods, and (iii) adds only a few milliseconds of overhead in the overall user latency.

1 INTRODUCTION

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are the most popular standards for secure Internet communications nowadays. An increasing number of web services are moving away from the traditional plaintext HTTP protocol to the more secure HTTPS [34]. One of the main reasons behind this increased adoption of HTTPS, is the decision of popular browsers (e.g., Google Chrome) to start showing a "Not Secure" warning for all websites that are being served over plain HTTP [14]. Indeed, as of November 2019 [28] more than 88% of the web browsing connections in the USA and more than 80% of the connections in the world currently being implemented over HTTPS.

Although the primary goal of TLS is to provide confidentiality and integrity for the vast majority of today's online communications, the provided security of TLS connections against potential network attackers depends vitally on the correct authentication and validation of the endpoints' public-key digital certificates presented during each connection establishment [39]. Responsible for *issuing*, *validating*, and *revoking* each digital certificate is a *Certificate (or Certification) Authority* (CA): a third party entity trusted by both communicating endpoints. Thus, when a web client connects to a website and receives its certificate, it can trust that this particular public key (contained in the certificate and signed by the CA) indeed belongs to that website. As a result, Certificate Authorities create a web of trust that enables complete strangers (i.e., a web client and a web server which have never previously communicated) to communicate with each other in a secure and trusted way.

Apart from *issuing* certificates, CAs may also need to *revoke* certificates as well. For example, when a private key of a website is stolen, the issued certificate needs to be *revoked*, and users need to be updated as soon as possible in order to not trust any more the web servers that use this certificate. There are two main ways for the web browsers to know when a certificate has been revoked. The simplest and more traditional one is to download from the CA the list of all revoked certificates (the Certificate Revocation List - CRL) and locally look up if the certificate in question is included in the CRL. In that case, the certificate has been revoked and should not be trusted any more. However, the increasing size of these CRLs [13] with median size of 51 KB, reaching as large as 76 MB in some cases [30, 35], has forced the clients to download CRLs rather sporadically. Unfortunately, such sporadic updates of CRLs leave clients with a certain *window of vulnerability*: between two successive downloads of the CRL clients may wrongly consider a *revoked* certificate as *valid*.

To close this window of vulnerability, the Online Certificate Status Protocol (OCSP) came to the rescue: CAs maintain available servers, namely OCSP responders, which are able to respond in real-time to queries about the revocation status of a *single* certificate. More specifically, OCSP works as follows: when a web client connects to a website and receives its certificate, the client queries an OCSP responder about the revocation status of that certificate. OCSP responders consult their local up-to-date database and are usually able to respond back to the browser, in most cases, in less than one second [36]. To avoid replay attacks, web clients usually provide OCSP responders with a *cryptographic nonce* and require OCSP responders to digitally *sign* their reply, including the nonce. To improve performance even further, some OCSP responders solicit the help of widely deployed Content Delivery Networks (CDNs) [2, 46], such as Akamai [37], managing to reduce their response time to less than a tenth of a second.

In the same spirit, DCSP [4], a newly introduced protocol, solicits the help of DNS resolvers and proposes to store revocation information in the publicly accessible DNS infrastructure. With the help of DNS, and its associated lightweight UDP protocol, DCSP is able to reduce end-user latency to just a few tens of milliseconds.

Although OCSP and DCSP provide good performance and timely information, they share a common characteristic: it is the *receiver* of the certificate (i.e., the web browser) who has to retrieve the required revocation information and verify whether the certificate is still valid or not. Thus, the *receiver (i)* has to download the certificate from a web site and *(ii)* verify that the retrieved certificate is indeed valid by contacting a third party directory such as the OCSP responders, the CDNs, or the DNS system. This usually incurs an extra TCP or UDP connection to the web client, increasing overhead accordingly.

OCSP Stapling and Must-Staple [8, 12, 19], on the other hand, provide a different type of solution: They advocate that it is *not the receiver* but the *supplier* of the certificate who has the responsibility to provide enough evidence to convince the client that the certificate is valid. In this aspect, the web server provides the web clients with two pieces of information:

(1) The certificate itself (much like previously).
(2) Revocation information about the certificate, *signed* by the issuer CA. This revocation information would be practically the same as the reply of the OCSP responder.[1]

To put it simply, under OCSP stapling the web server provides the web client *(i)* with the certificate, and *(ii)* with a signed confirmation that the certificate has not been revoked. OCSP Stapling is fast,

[1] Certificate Transparency[26], explained in depth in Section 2, provides a third piece of information as well: a signed proof that the certificate has been included in a publicly-accessible Log.

does not force the client to contact any third-party services, and has a low overhead. To keep its overhead as low as possible, web servers typically update the signed confirmations sporadically - once every few days or once a week. Unfortunately, during this period (i.e. between two updates) OCSP stapling is susceptible to *man-in-the-middle attacks* (MITM[42]).[2] Indeed, an attacker who has managed to steal the private key of a web server may provide to a victim browser an old (now revoked) public key and an old *signed* confirmation that the certificate has not been revoked. Since both the certificate and the confirmation are signed, the victim browser has no other option but to accept the old (now revoked) public key. Then, it will start communicating with the attacker believing that it communicates with the legitimate web server.

One way to mitigate this attack is to *timestamp* the revocation information before signing it, and pushing it to web servers to be served via OCSP Stapling. In this way, when a client receives OCSP Stapling information, it will first check the timestamp of the information. It will accept the revocation information only if the timestamp is recent: old timestamps are probably a sign of man-in-the-middle attacks. Although fine-grain timestamps (every few seconds or so) could solve man-in-the-middle replay attacks by minimizing the window of vulnerability to no more than a few seconds, the frequent timestamping and signing would impose a tremendous load on the OCSP responders.

To address the overheads imposed by frequent timestamps and signatures, in this paper, we design and implement CCSP[3] (Compressed Certificate Status Protocol): a new approach for timestamping and signing Certificate Revocation responses. CCSP is based on *Signed Collection* which provide revocation information *not for a single certificate* (like what OCSP and OCSP Stapling do), but *for a collection of certificates*. Since Signed Collections require just *one* signature for an *entire* collection of certificates, they have the potential to reduce the number of signatures needed and the associated overhead. Our approach saves more than six orders of magnitude signatures compared to state-of-the-art OCSP Stapling, and more than two orders of magnitude storage space compared to traditional CRLs.

To summarize, we make the following contributions:

- We introduce a new abstraction: the abstraction of *Signed Collections* that can be used to communicate revocation information in a very compact form.
- We present the detailed design of CCSP: an extension to OCSP and OCSP Stapling, which by using *Signed Collections* manages to improve performance by several orders of magnitude.
- To explore the trade-offs and applicability of our approach in the real world, we implement a fully functional prototype inside the popular GnuTLS library. In addition, we conduct an experimental evaluation to asses its imposed overheads. Our results show that it adds negligible overhead to the user experience compared to the state-of-the-art certificate revocation checking approach.
- We further evaluate CCSP using both simulation-based and experimental evaluation. Our results show that: *(i)* CCSP is able to pack revocation information for more than one million certificates in less than 10 KB of space, *(ii)* CCSP is able to reduce the number of required signatures by more than 6 orders of magnitude, and *(iii)* CCSP requires an average traffic rate of only a few bytes per second.

Paper Organization: Section 2 covers in full detail the advantages and disadvantages of the different existing approaches, and relates CCSP to other works in the field. Section 3 describes

[2] The shorter the validity period the larger the number of cryptographic signatures required, and the higher the overhead will be to transmit them.

[3] First proposal of this work appeared in [5]

the threat model of the paper and portrays the capabilities of the adversary, as well as a possible attack scenario. Section 4 presents, in full detail, the design of CCSP and introduces the notion of *Signed Collections*. Section 5 and Section 6 present the analysis and simulation-based evaluation regarding compression and deltas. Section 7 describes the implementation of the CCSP prototype, while Section 8 presents an experimental evaluation of this prototype. Finally, Section 9 discusses various aspects of our approach and Section 10 concludes the paper.

2 RELATED WORK ON CERTIFICATE REVOCATION ALTERNATIVES

In this section, we present the different approaches proposed in the literature as alternatives to the current state-of-the-art of OCSP- and OCSP Stapling- based certificate revocation. Then, we put our work into the context.

2.1 Revoke the trust from Certificate Authorities

Although most CAs are considered trusted, there exists a significant body of literature which assumes that CAs should not be trusted and should be replaced by some other mechanism. Perspectives [44], for example, is a project that later inspired the Convergence [16, 31] strategy for replacing CAs; it employs a crowdsourcing network of "notary servers", which build a global database of certificates used by each site, by regularly monitoring websites. These notary servers can be maintained by anyone e.g., organizations, institutions, private companies, the EFF, Google, Universities, or even a group of friends. By allowing several entities to maintain information about certificate status, the users are free to pick the entity of their trust, and query the validity of a certificate. Unfortunately, this operation imposes a significant amount of latency to the users' browsing TLS sessions. To make matters worse, to ensure the validity of the response, the user may need to query more than one notary server, and consider the response of the majority, an operation that significantly increases the certificate verification latency.

Over the past few years, Certificate Transparency[26] (or simply CT) has become widely popular. Aiming to address the issue of rogue, or compromised, Certificate Authorities, CT advocates that all valid certificates should be publicly and widely known. To support this publicity, CT maintains several independent certificate *Logs*: append-only repositories of all known certificates. When a CA issues a certificate, it adds the certificate to a Log and is given back a *receipt* (a signed certificate timestamp (SCT)). When a client receives a certificate from a web site, it also demands the signed certificate timestamp (SCT): the proof that this certificate is included in some publicly available Log. If the client does not receive the receipt, it does not trust the certificate. Certificate Transparency is an excellent way to deal with rogue or compromised CAs. Indeed, if a rogue CA includes fake certificates in some Logs, they will be easily spotted by the website owners who periodically scan all Logs for fake certificates. On the other hand, if the rogue CA does not include its certificates in any Log, it will not receive the SCT, which will result to its web clients not accepting these certificates. In both cases, *rogue CAs will not be able to continue their nefarious activities without being noticed.*

Although Certificate Transparency deal with rogue CAs, it does *not* explicitly deal with revocation. Indeed, as noted in the FAQ of CT's official website [27]: *"Certificates are revoked in the usual way and Certificate Transparency does not change that. It provides a mechanism by which you can know that a certificate needs to be revoked, but does not itself handle revocation."* Having said that, there exist some approaches aiming to integrate revocation in CT. For example, Revocation Transparency (RT) [25], provides a way to supply fresh revocation information. Unfortunately, careful studies of RT [38] suggest that the original RT proposal has overhead linear to the number of revocations. As the number of revocations increases with time, such an overhead is probably prohibitive for most practical applications. Although recent work has reduced the overhead to logarithmic [38], it may still be high compared to other Certificate Revocation approaches that respond in (average) constant time.

Chen et al. in [6] propose a blockchain-based certificate audit scheme for TLS connections named CertChain. Their approach includes four entities: the client, the CA, the domain (website) and the bookkeepers, which record the certificate operations into the blockchain. In their solution when clients want to verify a certificate they need to contact a bookkeeper, which looks into the blockchain in an efficient way, and receive back a response with the certificate status. Contrary to our approach, CertChain responses contain information only for the certificate in audit, which in the majority of the cases, poses a higher latency than CCSP because of the extra communication.

Larisch et al. in [24] propose CRLite, which aggregates revocation information for all known valid TLS certificates. By utilizing the filter cascade data structure, CRLite can store efficiently all the revocation information available without false positives. Similar to our approach, CRLite aims to provide the latest revocation information in the most space efficient way. However, contrary to our solution that only needs a few KB for a full collection and even less for the delta, CRLite requires several MB for the initial list and about 580KB for each delta.

Schulma et al. in [40] introduce RevCast, a system which uses low bit-rate radio broadcast (in their case FM) to deliver revocation information in a timely, scalable and privacy preserving manner. Although they prove the viability of their approach through real world experiments, there are some hard to overcome obstacles. Specifically, their solution would require several changes to the infrastructure responsible to deliver the revocation information as well as the adoption of FM receivers from all the end-host devices.

2.2 Leveraging the DNS infrastructure

DNS-based Authentication of Named Entities (DANE) [21] is another approach towards the direction of replacing Certificate Authorities. DANE leverages the DNS infrastructure to distribute the public key of the website. To achieve that, DANE introduces a new type of DNS record, named TLSA, in which it stores the *whole* certificate of a domain, and uses DNSSEC to validate its integrity. Unfortunately, it is absolutely dependent on the deployment and validation of DNSSEC, the latter of which has remained stagnant at around 20% worldwide [3]. To make matters worse, a study [47] has shown the current DANE deployments have security inconsistencies. In addition, given that it stores the entire certificate in TLSAs, 33% of the responses analyzed were larger than 1500 Bytes, thus resulting to IP fragmentation, imposing this way additional latency to the query procedure and leaving the protocol vulnerable to fragmentation attacks [20].

DNS-based Certificate Status Protocol (DCSP) [4] leverages the existing DNS infrastructure to distribute certificate status information without abolishing the role of CAs. On the contrary, DCSP assigns to the CAs the responsibility of maintaining and signing the DNS records, guaranteeing in this way their validity and freshness. it uses DNS as a fast cache, and capitalizes on multiple DNS TXT type records to allow CAs to publish the revocation information of certificates. In that way, DCSP *(i)* achieves better performance than traditional OCSP, and *(ii)* preserves the privacy of the user's browsing history. However, utilizing the existing DNS infrastructure requires specific changes to the way CAs handle certificate revocation. In CCSP, we extend the current revocation mechanism, thus allowing our approach to be more easily applicable. Finally, by further improving the grouping abstraction of DCSP, we are able to significantly reduce the number of signatures required by each CA.

2.3 The Google Approach

Chromium browser and thus Google Chrome and Brave use *CRLsets* [22]: a compressed list of a small set of revoked certificates (similarly Mozilla Firefox uses OneCRL [17]). In this way, web browsers may have handy revocation information about a (small) set of revoked certificates, so that most of the time they will be able to check the revocation status locally, and quickly. CRLsets have great performance when they encounter a *hit*, but need to resort to OCSP (or similar) when the certificate they are looking for is not included in it. Given that CRLsets cover less than 1% of revocations [30], there may be room for further improvements, that will deal with the remaining 99% of revocations.

2.4 Adoption in the web

In a more recent work [43] the authors investigate how different browsers implement the validation process of TLS. They report inconsistent behavior among the browsers with potential dangers for the users. They suggest that the reasons behind those inconsinstencies are the complexity of the standards and the absence of coordination between the browser developers. Interestingly enough they also report that current revocation mechanisms suffer from performance overheads or other limitations which can affect the user experience. In [9] the authors check whether the web supports correctly the OCSP Must-Staple protocol. They collect a large corpus of certificates to understand the adoption of OCSP Must-Staple. They discover that just 0.02% of the web actually adopts this protocol and even in that cases they found several errors across all the infrastructure, starting from misconfigured and unavailable OCSP responders, to wrongly developed web servers. They also note that all but one of the major browsers do not actually check correctly the OCSP responses.

2.5 Where does CCSP fit in the spectrum?

CCSP is not competing to, but is complementing most of the previous approaches. By introducing the abstraction of *Signed Collections*, it is able to encode revocation information *not only for one certificate, but for a large set of certificates* into a very small space. This encoding can be used in several existing revocation approaches that would like to pack as much revocation information into as little space as possible. For instance, OCSP enriches each OCSP response with a nonce and signs each reply including the nonce in the signature. The introduction of timestamped *Signed Collections* can substitute the use of nonces. This is because the timestamp is a unique number (much like nonce) which is up-to-date and recognizable by all clients (assuming they have relatively sycnhronized clocks. Note, however, that Signed Collections need to be signed *once* for each timestamp no matter how many clients are going to receive the signed collection. The utilization of *Signed Collections* can increase user privacy, improve the overall performance and significantly reduce the number of signatures (more details can be found in section 9).

3 THE THREAT MODEL

In this paper, we assume that the attacker is able to launch a man-in-the-middle (MITM) attack against a victim. This attack can happen in a variety of settings, including, for example, the attacker *(i)* controlling a public, free Wi-Fi, *(ii)* controlling a VPN the victim uses, *(iii)* and installing a rogue Wi-Fi router. In addition, we assume that the attacker managed to get access, possibly through hacking, to the private key of a web site (that will be revoked). We finally assume, that after realizing this hacking event, the web site, as expected, revoked its certificate.

By deploying such an attack, the adversary is able to assume the identity of a legitimate owner of the certificate, and by falsifying responses from DNS and/or HTTP endpoints (via the MITM attack), the adversary manages to successfully impersonate the owner of the certificate. As a consequence, the attacker can deceive users, making them establish wrongly trusted TLS connections with it, and exchange secrets as if it was the actual legitimate party. The goal of our approach is to make the clients aware of the impersonation attack, in order for them to abort the connection with the adversary soon enough, to prevent any possible leakage of private information.

3.1 Possible Attack Scenario

Based on the assumptions above, a possible attack scenario may be the following: When a victim tries to connect to the web site (whose certificate has been revoked), the attacker (who managed to launch a MITM attack) presents the victim with the *old* (revoked) certificate, which *was* valid sometime in the recent past. To convince the victim that the certificate has not been revoked, the attacker will *replay* to the victim an old but *signed* OCSP Stapling response, claiming that the certificate is *still valid.* Once presented with a signed OCSP Stapling response, the victim's web browser will assume that the certificate is valid (even though it has been revoked), and will start communicating with the attacker thinking that it is communicating with the legitimate web site[4].

Similarly to the case of OCSP Stapling, a man-in-the-middle attack can be also launched in the case where OCSP responses are served by a Content Delivery Network (CDN). The attacker, who has managed to launch a man-in-the-middle attack, is able to impersonate both the web site and the CDN-based OCSP responder. When a victim tries to connect to the web site, the attacker presents the victim with the *old* certificate. Furthermore, the attacker, who is able to impersonate the CDN-based OCSP responder, presents to the victim the old but *signed* OCSP response, tricking the victim to believe that the certificate is *still valid.* Then, the victim's web browser will start communicating with the attacker, thinking that it is communicating with the legitimate website.

From that point onwards, the victim is in a downward spiral: they will probably supply their real password to the attacker, disclose personal information, and, depending on the web site's expected functionality, may suffer identity theft, may be tricked to install malware, and may even suffer financial losses.

3.2 The Need for Timely Revocation

Although such attacks can be considered rare and not easy to launch, the damage they can perform is severe, including monetary loss, loss of privacy, security, or in critical systems (e.g., medical devices) even loss of life. Should an attack like the above be carried out, the victim and their software have no way to distinguish if the website is indeed the correct one, or one presented by the attacker. One might think that the green browser "padlock" and the "Secure" indicator of browsers might provide a warning, but they can still be tricked and will wrongly re-assure the user that he is communicating with the correct web site[5]. Studies have measured the revocation rate to be around 8% [18, 30], or 180,000 in absolute numbers (as measured by the popular Certificate Authority Let's Encrypt[29]), which is quite high considering a window of vulnerability of up to seven days, and even more in some cases. Note, that incidents like Heartbleed may skyrocket the rate of revocations: e.g., from 29 certificate revocations per day before Heartbleed, it reached to an average of 1,414 revocations per day right after Heartbleed was publicly disclosed [45].

3.3 The Solution Framework

Defending against the above kind of attack is not easy. By managing to issue a man-in-the-middle attack, the attacker has enclosed the victim in a fake virtual world and may provide fake information at will. One way for the victim to break out of this fake world is to ask for *timely* information: i.e., ask for information that is *timestamped* with the *current* time and *signed* by a trusted third party (such as a CA). In the context of OCSP Stapling, this would imply that CAs should frequently *timestamp* and *sign* all OCSP Stapling responses. Thus, when the browser is presented with a response that

[4] Several recent security incidents have exposed the problem of rogue CAs: that is, CAs which provide bogus certificates or bogus information. The recently proposed "Certificate Transparency" manages to uncover such rogue CAs as explained in the related work section. However, the detection and mitigation of such Rogue CAs is outside the scope of this work.

[5] Google removed green padlock icon in Chrome since only HTTPS supporting websites will be accessible[1].

has an *old timestamp*, it will just reject the response and its associated revocation information as stale. Unfortunately, timestamping and signing all OCSP Stapling responses very frequently (say every few seconds or so) may place a tremendous burden on CAs and OCSP responders who may have issued tens of millions of certificates and are now required to *timestamp, sign, and distribute millions of responses per second*.

To reduce this overhead, in this paper we introduce *Signed Collections*: an abstraction that packs revocation information *not* for a single certificate, but for a *collection* of certificates, in a single response, and thus, reduces the associated number of required signatures by several orders of magnitude. For example, 1,000,000 revocation bits can be packed together requiring only one signature operation, instead of one signature operation for each and every certificate. Hence, CCSP needs about 6 orders of magnitude less signature operations than the above case of frequently timestamped OCSP Stapling responses[6].

4 DESIGN

4.1 High-level Design

CCSP introduces the notion of *Signed Collections*: an abstraction that enables us to pack revocation information about *several* certificates in a *single* OCSP response. This response, i.e., the Signed Collection, is actually a *bitmap*. Each bit of the bitmap corresponds to the revocation status of a single certificate: if the bit is “1”, the certificate that corresponds to this bit is revoked; if the bit is “0”, the certificate is still valid. We call these bits *Revocation Bits*, because they provide information about the revocation status of certificates.

When a certificate C is issued, the issuer CA assigns it to a Signed Collection SC. The name of this Signed Collection, as well as the certificate’s index in the collection, which corresponds to the revocation status of this certificate, is included in the certificate itself. So, when a client connects to a web site, it receives the certificate C, which contains the name of the Signed Collection (i.e., SC) and the index i of the certificate within the collection. Then, if the client is provided with an OCSP Stapling response, it will contain SC. If the web server does not support OCSP Stapling, the browser will fetch SC from an OCSP Responder, possibly hosted in a CDN near the user. Finally, the client will check the certificate’s validity bit $SC[i]$. If this $SC[i]$ value is “1”, the certificate has been revoked. If this value is “0”, the certificate is still valid.

4.1.1 Compression.

One might think that in a Signed Collection (which is essentially a bitmap) of size S ($SC[1..S]$), we can fit revocation information for about S certificates and no more than that. Fortunately, in our design we show that it is possible in S bits to fit revocation information for *more* than S certificates. Although this may sound counter-intuitive, we can easily achieve it using *compression*. Actually, we compress along two dimensions: *(i) space*, and *(ii) time*.

(1) **Space:** If we take a careful look at a Signed Collection SC, we will realize that most of the bits in the bitmap $SC[]$ are “0”. This should be expected since most of the time, most of the certificates are not revoked: that is why most of the bits are “0”. Experimental results suggest that for most CAs less than 1% of the certificates are revoked [30], reaching as low as 0.2% in some cases [29]. This implies that roughly more than 99% of the bits in the $SC[]$ bitmap will be “0”. Thus, simply compressing such a bitmap may reduce its size significantly. For the purposes of this work, we support two compression algorithms: *(i)* the DEFLATE

[6] One may suggest a selective approach where only some of the OCSP responses get frequently timestamped (see Section 9.2).

compression algorithm which is a variation of LZ77 [48], and *(ii)* the Golomb compression algorithm [15].

(2) **Time:** Recall, that in CCSP we timestamp and sign each Signed Collection periodically - once in every, what we will call from now on, *epoch*. Taking a closer look at these periodic releases of a given Signed Collection, we see that they are very similar to each other. That is, each periodic release of a Signed Collection has little, if any, changes compared to the release of the same Signed Collection of the previous epoch. Indeed, in the time period of an *epoch*, which is in the range of seconds or at most minutes, we expect only a very small number of certificates, if any at all, to be revoked. Thus, if instead of releasing each Signed Collection from scratch at the beginning of each epoch, we release the Signed Collection's *changes* (its *delta* compared to a previous version), then we will be able to reduce the size of the released Signed Collections significantly.
To put the compression algorithm in focus and explain the two forms of redundancy, we divide the time into *epochs* and *eons*. For the purposes of this work, an eon is a time interval in the range of hours and an epoch is a time interval in the range of minutes or seconds. Using this terminology, each Signed Collection has to be downloaded *once per eon*, while at each epoch we need to download only the Signed Collection's *delta* from the start of the current eon. The latest SC delta in the current eon contains all the revocation information of the previous epochs in that eon, making sufficient to download just the latest delta to retrieve the most recent revocation information.

4.2 Detailed Design

4.2.1 Certificates.

CCSP extends the definition of the Certificate with two fields:

- "SignedCollection": this is the OCSP URL of the Signed Collection *SC* in which this certificate belongs. This field contains the OCSP Responder and path to obtain the needed *SC*. It always starts with "https://".
- "SignedCollectionIndex": this is the index in the Signed Collection bitmap. The bit pointed to by this index contains the revocation information of the certificate.

4.2.2 OCSP Responders.

In order to implement CCSP, some changes are required in the OCSP Responders. To successfully serve the Signed Collections, each OCSP Responder must accept HTTP "GET" requests to the path "/ccsp/SC-UID/sc", where SC-UID is the unique identifier of each Signed Collection. This path, up to and including SC-UID is contained in the "SignedCollection" field of the certificate. In order to serve the deltas within an eon, the OCSP Responder must accept HTTP "GET" requests to the path "/ccsp/SC-UID/delta", where SC-UID is, of course, the unique identifier of the Signed Collection. Note that the users are able to receive only the latest delta and not the previous ones since they are not needed in order to retrieve the current state. The reason behind the selection of these paths, is that there is no special software modification required and a simple web server can be used to serve everything as static content. This is especially useful for CAs that employ CDNs, because they only need to push all the files once and have the CDN cache them globally, instead of requiring an origin pull in case of a cache miss. This ensures 100% cache hit rates and also does not require any additional code that may introduce complexity and/or bugs.

4.2.3 **OCSP Stapling Web Servers.**

Web Servers can support CCSP and accelerate the user experience by serving the Signed Collections during the TLS handshake. This eliminates the need for a connection to an OCSP Responder. In order to support this feature, web servers need to be able to send two stapled responses: *(i)* the Signed Collection for this eon, and *(ii)* the Signed Collection's delta for this epoch. Although sending both responses will not add significant size to the response, the client can specifically request only the delta if it already possesses the latest Signed Collection (i.e., received from a previous connection). These responses can be served by the existing OCSP Stapling mechanism of web servers, with only slight modifications.

4.2.4 **Signed Collections.**

A Signed Collection is a response provided by an OCSP Responder and is updated every *eon*. It contains the following fields:

- VERSION: the version of CCSP used, as an 8-bit unsigned integer, to accomodate future changes
- HEADER: a 16-bit field with only the first bit currently used as a "delta bit": if set to "1", this response is a delta and not a full Signed Collection
- SIZE: the size of the compressed bitmap, in bytes, as a 32-bit unsigned integer
- BM: the compressed bitmap, in raw bytes
- SC-UID: the unique identifier of the Signed Collection, in ASCII, null-terminated, and exactly as it appears in the "Signed Collection" part of the certificate
- EON: the *eon* identifier, i.e., the date and time, as a 64-bit unsigned integer
- COMPRESSION: the ID of the compression type used in the bitmap[7], as a 16-bit unsigned integer
- SIGNATURE: the cryptographic signature by the CA which can be calculated by signing a cryptographic hash of the concatenation of the fields above, in raw bytes

4.2.5 **Signed Collection Deltas.**

For every *epoch* within an *eon*, a new delta needs to be produced. This delta consists of the following fields, with the same size and type as the ones above:

- VERSION: the version of CCSP used
- HEADER: a 16-bit field, just like above
- SIZE: the size of the included data
- DATA: the data of the delta, in raw bytes
- SC-UID: the unique identifier of the Signed Collection of this delta, exactly as it appears in the "Signed Collection" part of the certificate,
- EPOCH: the *epoch* identifier of the current delta
- COMPRESSION: the ID of the compression type used for the data, just like previously
- SIGNATURE: the cryptographic signature by the CA, calculated similarly to the signature of *Signed Collections*

A delta can be computed in two ways. The first way is by calculating the output of the bitwise "XOR" of the Signed Collection bitmap in the current *eon* with the latest state of the bitmap in the current *epoch*. This will produce a new bitmap in which all bits will have a value of "0" unless their

[7] We currently support Golomb and DEFLATE compression. More compression algorithms can be easily added. The current IDs are "0" for no compression, "1" for DEFLATE, "2" for Golomb, and "3" for Index Based Compression.

Notation	Explanation
S	size of compressed bitmap
x	number of revocations per *eon*
r	size in delta needed for each revocation
k	average time (in *epochs*) between two revocations of any certificate in a signed collection

Table 1. Summary of Notation

corresponding certificates have been revoked within the current *eon*. This bitmap is then compressed with an algorithm and added in the "DATA" section of the delta. The client can decompress and reconstruct the latest bitmap by performing bitwise "XOR" between the latest Signed Collection and the latest delta.

The second way involves Index-Based Compression. This method does not create a bitmap but a list of the indices of the positions that turned into "1". The CA calculates a bitmap with the method used above and then determines the positions of "1"s in this bitmap and appends them in the "DATA" area of the response, as 32-bit unsigned integers. The client then switches all bits in the Signed Collection to "1" if their index in the Signed Collection is included in this list.

5 ANALYSIS

For the purposes of system description we have assumed that an *eon* is a time interval in the range of several minutes or hours. One might wonder, however, how frequently we should start a new *eon*. Actually, there is a very interesting trade-off here: a very short *eon* will force CCSP to transfer the entire revocation bitmap (even if compressed) frequently. On the other hand, a very long *eon* will allow the deltas to slowly increase in size until they are not space-efficient any more. In this section, we will try to find an optimal value for the length of the *eon*.

Assumptions: In the rest of this analysis, we make the following assumptions, which are also summarized in Table 1:

- the length of the *eon* (in certificate revocations) is x.
- the size of the entire bitmap in bytes (compressed with Golomb compression) at the start of an *eon* is S.
- we expect to have one certificate revocation in the bitmap every k *epochs*.
- each revocation will increase the delta by size r.

Analysis: Based on those assumptions, it seems that after the 1st revocation, the deltas will have size r, after the 2nd revocation: $2r$, after the 3rd revocation: $3r$, ..., and after x revocations the deltas will have size xr. This implies that after x revocations (i.e. at the end of the *eon*), we will have sent kx deltas (recall that we assume that we have 1 revocation every k epochs) with a total size of:

$$\sum_{i=1}^{x} kir = \frac{krx(x-1)}{2}$$

So, in the duration of x revocations the total information transferred (initial bitmap plus deltas) will be:

$$S + \frac{krx(x-1)}{2}$$

So, the average size of deltas will be:

$$I(x) = \frac{S}{kx} + \frac{r(x-1)}{2} \tag{1}$$

To find the optimal value of x, we will just need to take the derivative of the above size of deltas and solve for x. So:

$$\frac{dI}{dx} = -S/(kx^2) + r/2,$$

which when solved for x gives the optimal value of x to be:

$$x = \sqrt{2S/(kr)} \tag{2}$$

The actual values of S, k, and r may vary from system to system and thus the optimal value of x may slightly change. For the purposes of illustration, let us assume some reasonable values for S, k, and r and try to calculate the bandwidth needed to support CCSP. Our experiments in the next section will show that we need less than 10 KB to store revocation information for one million certificates and thus a reasonable value for S would be: 10 KB. The choice for r is straightforward: 4 bytes - the size of a 32-bit long word. Finally, for k we assume a rather frequent certificate revocation process: one revocation per each epoch, so k is 1. Plugging these numbers in equation 2 gives us an x close to 71. This implies that every 71 epochs or so, we should start a new *eon*. The average information that will be transferred per epoch (from equation 1) would be:

$$I(x) = \frac{S}{kx} + \frac{r(x-1)}{2}$$

or

$$I(x) = \frac{S}{k\sqrt{2S/(kr)}} + \frac{r(\sqrt{2S/(kr)} - 1)}{2}$$

or

$$I(x) = \sqrt{\frac{S^2}{k^2(2S/(kr))}} + \sqrt{\frac{2r^2S}{4kr}} - r/2$$

or

$$I(x) = \sqrt{\frac{Sr}{2k}} + \sqrt{\frac{rS}{2k}} - r/2$$

or

$$I(x) = \sqrt{\frac{2rS}{k}} - r/2$$

which implies that at each *epoch* we need to transfer the following amount of bytes:

$$I(x) = \sqrt{\frac{2rS}{k}} - r/2 \tag{3}$$

For the values of S, k, and r, which we have assumed, (i.e. $S = 10$ Kbytes, $k = 1$, $r = 4$ bytes) the above equation, as can be also seen in Figure 1, implies that we need to transfer an average of 284 bytes per *epoch*. Given that each *epoch* is around 60 seconds, we need to transfer about *5 Bytes per second* which is a trivial overhead to pay by today's standards.

Finally, in Figure 1, we simulate the average size of deltas for 3 different cases of certificate revocation rates: (i) a low rate of 0.1 revocations per epoch, (ii) the rate of 1 revocation per epoch that was described above and (iii) an extreme scenario of the high rate of 10 revocations in each epoch. As can be seen, even in the scenario of a high certificate revocation rate, the average overhead imposed to the user is only about 15 Bytes per second.

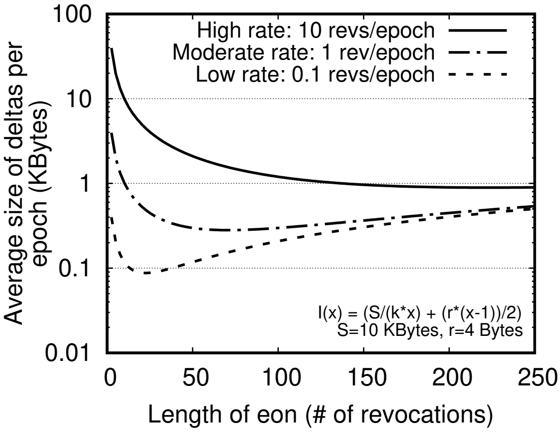

Fig. 1. Average size of deltas per epoch. In the case of a moderate certification revocation rate, the average size of deltas per epoch is 284 bytes, which accounts to an overhead of less than 5 Bytes per second to the user. Even in the scenario of a very high revocation rate, the average overhead imposed to the user is about 15 Bytes per second.

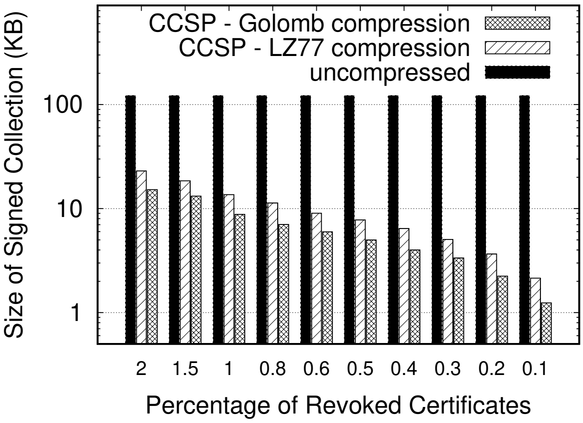

Fig. 2. Size of a Signed Collection of 1M Revocation Bits. Without compression it would require 122 KBytes to be stored. Both compression algorithms of CCSP can reduce the size needed by one to two orders of magnitude.

6 SIMULATIONS

6.1 The effect of Spatial Redundancy: how much space do you need to store one million bits?

At first, we set out to explore whether we can effectively compress revocation information. For the purpose of this study we assume that we have a Signed Collection of one million revocation bits and we would like to find how much space it needs to be stored. Apparently, in the absence of compression we would normally need 1,000,000 (certificates) / 8 (bits per byte) or roughly 122 KBytes to store 1 million revocation bits. However, CCSP employs spatial compression using LZ77 or Golomb which implies that we may be able to "squeeze" 1 million bits in less than 122 KBytes.

Obviously, the effectiveness of compression depends on the type of data we want to compress. Indeed, if we want to compress random data (e.g. a sequence of "1" and "0" generated by a random number generator with equal probability each), then compression will have little effect: the data will be "too random" to be compressed efficiently. On the other hand, if the data consist only (or mostly) of "0", then compression will be very effective. In our case, the effectiveness of compression depends on the number of revocation bits set to "1", which is basically the number of revoked certificates. Although the percentage of revoked certificates may vary form one CA to another, it is usually in the range of 1%. Actually "Let's Encrypt", one of the largest and most popular CAs, reports less than 0.2% revoked certificates [29].

Figure 2 shows how much space is needed to store 1 million Revocation Bits using the two compression approaches we employ: LZ77 (CCSP-LZ77) and Golomb (CCSP-Golomb) as a function of the percentage of revoked certificates. The first thing we notice is that both versions of CCSP (i.e., both CCSP-LZ77 and CCSP-Golomb) perform well. Indeed, for revocation rate close to 1% CCSP-Golomb requires roughly 8.8 KBytes. For revocation rate close to 0.1% the space requirements for CCSP-Golomb drops to just a bit higher than 1 KByte (i.e., 1.24 KBytes): two orders of magnitude less space than the "uncompressed" case. CCSP-LZ77 is a bit higher but stays in the same range. That means that preferring LZ77 due to its easier implementation, wide adoption by browsers, speed, and lower memory footprint will come at minimal bandwidth cost, compared to Golomb. What we learn from this Figure is that *it is possible to hold revocation information for millions of certificates in just a few KBytes of space*. We believe that these results debunk any concerns about the space needed to hold revocation information and open the road for CCSP and similar solutions to certificate revocation. Given that we have such small space requirements, one might wonder if we still need *both* eons and epochs, or if *eons* would be enough. This is a reasonable concern. However, having both eons and epochs allows Signed Collections to contain a larger number of certificates - several million certificates - or even tens of millions of certificates. Given that the size of deltas in epochs is no more than a few Kbytes, epochs can be as small as a few seconds, even when Signed Collections contain tens of millions of certificates. At the same time, the computation costs for signatures and verifications are rather small or even unoticeable: a modern-day computer can verify close to 60,000 RSA-2048 signatures per second (per thread).

> **Finding**: One to Two KBytes of space is all you need to store revocation information for one million certificates.

6.2 Would Bloom Filters achieve better compression?

CCSP proposes an *exact* way to representing Revocation Bits: it uses one bit for each certificate; the value of the bit represents whether the certificate is revoked or not. On the other hand, it has been proposed that Bloom Filters may possibly reduce space requirements [30] compared to *exact* methods. Although Bloom filters are very efficient at representing sets of objects using only a very small amount of memory, they do suffer from false positives. In our case this means that it is possible that a *non*-revoked certificate can be reported by Bloom Filters as revoked (i.e., false positive). To mitigate false positives, Bloom Filters may fall back to search the entire bitmap (or the entire Certificate Revocation List) when they have a false positive. Actually, since Bloom Filters can not distinguish false positives from true positives, they need to consult the entire list whenever they encounter a positive (False or True). Despite their false positives, Bloom Filters have small space requirements and we would like to explore what are their space needs compared to CCSP.

For the purposes of this evaluation, we assume that we have 1 million certificates, a small percentage of which are revoked. This information can be stored as a Signed Collection (much like CCSP does) or as a Bloom Filter (much like it was proposed in [23]). Note that the actual space

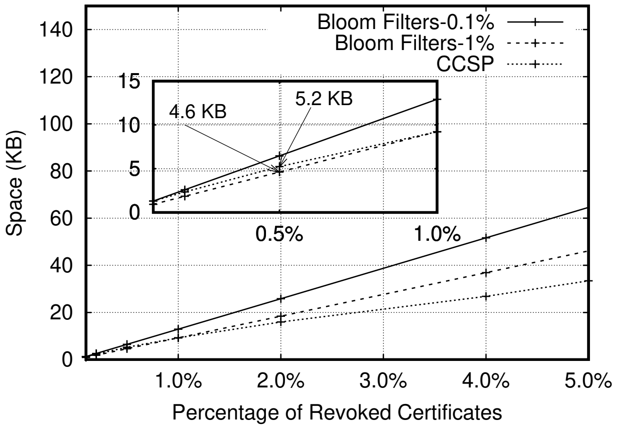

Fig. 3. Space required to store revocation information for one million certificates, using bloom filters and our approach, both compressed with the Golomb compression algorithm.

requirement for Bloom Filters is influenced by the false positive rate they are willing to tolerate. Indeed, the lower the false positive rate, the larger the number of "0" they will have and thus, the larger the size needed by the Bloom Filters. Typical false positive rates in the literature range between 0.1% and 1%. Higher false positive rates may nullify the speed benefits of Bloom Filters and lower false positive rates may result in extremely high space requirements.

In this experiment, we find the space requirements of Bloom Filters for false positive rates between 0.1% and 1%. For both CCSP and Bloom Filters we use Golomb compression to reduce their size to (almost) the minimum possible. Figure 3 presents the size of the Signed Collection of CCSP-Golomb and the size of the Bloom Filter as a function of the percentage of the revoked certificates. We see that for a revocation rate of about 1% the size required by CCSP is a bit less than 10 KBytes - as expected. This size increases with the higher percentage of revoked certificates and reaches a bit more than 30 KBytes for 5% revoked certificates. In general, Bloom Filters (both for 1% and for 0.1% false positives) seem to require more space than CCSP-Golomb. If we focus to percentages (or revoked certificates) smaller than 1% (inset plot), the size required for Bloom Filters (1%) is barely smaller than the size required for CCSP. For example, for a percentage of revocation certificates equal to 0.5%, CCSP-Golomb requires 5.2 KBytes of space, while "Bloom Filters-1%" require 4.6 KBytes, and "Bloom Filters-0.1%" require 6.4 KBytes. It is true that for 0.5% revocation rate "Bloom Filters-1%" seem to need about 10% less space than CCSP-Golomb but they have a hidden cost: false positives. Indeed, in 1% of the cases they will require the clients to consult the entire CRL, which implies an extra TCP connection to the CA. Since, at the time of this writing, the benefits of Bloom Filters (as a compression function) are not really clear compared to other methods (e.g., Golomb compression), we have not included them among the compression functions in the "Signed Collections" record. Since, however, the record allows the use any kind of compression functions, we do not preclude their possible inclusion in the future.

Finding: Bloom Filters may decrease space requirements but only marginally. Golomb compression is already very effective at reducing space needs.

6.3 How will CCSP perform if massive revocation happens?

Although most of the time revocations would come at a slow pace, there exist rare cases where massive revocations will be initiated. Take for example the Heartbleed bug which forced a large number of certificates to be revoked [11]. What would the performance of CCSP be? Would it collapse? or would it be able to tolerate the blow? Would web servers (that use OCSP stapling) and CDN networks (that serve CDN-based OCSP) create an unreasonable amount of traffic?

To drive our answers, we use real statistics from Heartbleed[11]. According to them, the largest revocation burst following Heartbleed was from GlobaSign which revoked 56,353 certificates over 2 days, which amounts to just below 20 revocations per minute, or about 20 revocations per *epoch*, since an epoch is about one-minute long. Let us now assume that each *eon* contains around 71 epochs (as estimated in Section 5) and that each *epoch* contains around 60 seconds; these revocations will create 71 deltas with a total size of: $\sum_{i=1}^{71} 20 \times i \times 4 = 194\ KBytes$. Adding to this the traffic needed to transfer one bitmap per *eon* ($=1,000,000/8/1024 = 122$ KBytes). Thus, the total traffic that will be created is $194 + 122 = 316$ KBytes over 71 epochs or $316 \times 1024/71/60 = 0.074$ Kbytes per second. Thus, the amount of traffic due to revocation that needs to be transferred between a CA and a CDN is in the range of 0.07 KBytes per second. However, for the ease of calculations, in the above we considered a non-compressed bitmap (and deltas), which represents the worst case scenario i.e., when about half of the certificates in the bitmap are revoked. Given the high capacity networks that CDNs employ, it seems that this 0.07 KB per second, even in the worst case scenario, is a tiny percentage they should not worry about. When considering only the transfer of deltas (since the bitmap will be typically compressed to a significantly smaller size), only 0.04 KB per second needs to be transferred by CCSP for handling certificate revocation under such a Heartbleed-like scenario.

> **Finding**: Even during a new, Heartbleed bug the traffic generated due to CCSP-based revocation information will be only around 0.04 KBytes per second.

7 IMPLEMENTATION

To assess the feasibility and effectiveness of CCSP, we implemented a preliminary prototype library of our approach. In this CCSP prototype, we modified the existing TLS implementation of the GnuTLS [32] library (`handshake_server()`) to enable a web server to distribute the most updated CCSP's *Signed Collections* and deltas. In addition to that, we created our own pair of private and public keys and signed our own CCSP supporting certificate: a certificate which (by following the design presented Section 4) contains information about the Signed Collection it belongs to, and its index within that Signed Collection.

Then, we deployed an Apache web server configured with the `mod_gnutls` extension to load our modified GnuTLS server side library. This server is responsible of running CCSP, thus providing users with both (i) the certificate and (ii) the Signed Collection and deltas. By doing so our prototype works as follows: initially, during the connection handshake, the server sends to the client the *eon* and *epoch* identifiers of the current (latest) Signed Collection and delta. After that, the client checks if it already posses any of them from previous connections (e.g., visited another website with certificate in that Signed Collection) and responds to the server with what it needs. The possible cases are: (i) the client needs both the latest Signed Collection and delta, (ii) it needs none, because of a visit to a website belonging in this Signed Collection very recently, during the current epoch, and (iii) it needs only the latest delta, because of a website visit some time in the recent past, during the latest eon but not in the current epoch. After receiving the client's response, the server responds by sending the latest Signed Collection and/or delta.

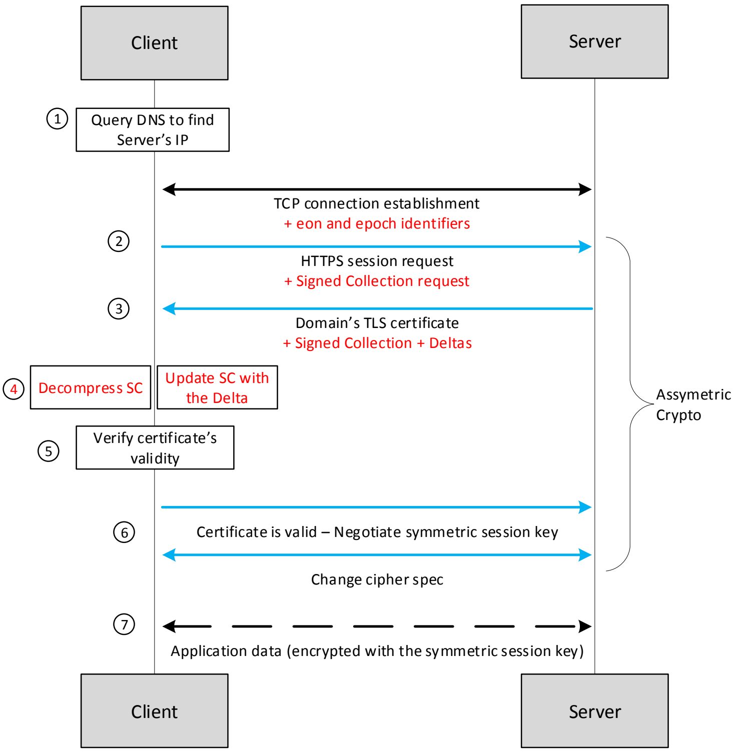
Client
Server
① Query DNS to find Server's IP
TCP connection establishment
+ eon and epoch identifiers
② HTTPS session request
+ Signed Collection request
③ Domain's TLS certificate
+ Signed Collection + Deltas
④ Decompress SC
Update SC with the Delta
⑤ Verify certificate's validity
⑥ Certificate is valid – Negotiate symmetric session key
Change cipher spec
⑦ Application data (encrypted with the symmetric session key)
Assymetric Crypto
Client
Server

Fig. 4. High level overview of a TLS Handshake using our CCSP implementation. Marked with red are the additional steps we integrated in GnuTLS, where the user retrieves the needed Signed Collection and delta.

On the client side, we also modified the GnuTLS library (the functions: `_gnutls_x509_cert_verify_peers()` and `handshake_client()`), to receive CCSP's supporting certificates and CCSP revocation information (Signed Collection and delta) and check the validity of the certificate based on that instead of the traditional OCSP responses. So, first the typical certificate checks are performed (i.e., checking its expiration date, validity of signature, etc.), and then the received Signed Collection and delta are passed to `_gnutls_x509_cert_verify_peers()`. In this function, we perform all the necessary processing for CCSP certificate validation, which includes XORing the Signed Collection and the latests delta, and checking the specific revocation bit of the bitmap that corresponds to the given certificate. If the revocation bit is "0" the client proceeds with the connection, otherwise it terminates the connection as the certificate had been revoked.

Figure 4 presents a high level overview of a TLS Handshake in the above CCSP stapling implementation. As can be seen, the TLS handshake begins after the establishment of a TCP connection (step ①), with the client requesting an HTTPS session from the server (step ②). In this very first request, the client piggybacks a header to declare to the server his interest in receiving the latest Signed Collection and/or delta (step ②). After that, (step ③) the server will respond back with the domain's certificate, the Signed Collection (if the client requested for it) and the deltas since the beginning of the *eon*. If the client receives a new Signed Collection, it will decompress and update it with any received deltas (i.e. XOR SC with delta) before storing it locally (step ④). Finally, the

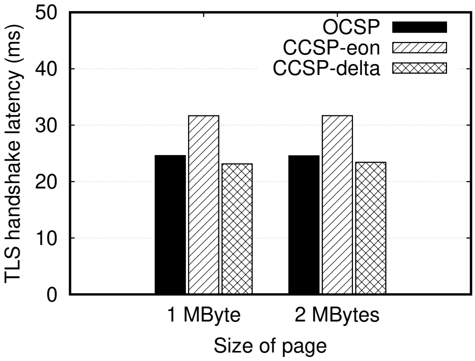

Fig. 5. Time it takes for OCSP and CCSP to complete a TLS handshake. CCSP-eon is 6 ms slower than the OCSP Stapling and retrieves the entire signed collection once every eon. On the contrary, the more frequently appeared CCSP-delta completes a TLS handshake 1 ms faster than OCSP Stapling.

client is able to validate the received certificate (step ⑤) and either proceed with the symmetric session key negotiation or terminate the TLS handshake (step ⑥).

Our prototype can be used by any browser capable of using the GnuTLS library, but for the purpose of simplicity, in this work, we decided to load our library in the simple browser of GNU `wget` [41] as a proof-of-concept. We decided to use GNU `wget` in order to avoid dealing with the complexity and any other issues those browsers may possibly have, and any latencies and overhead that is non related with our approach. For implementing and deploying our CCSP prototype we only made a few modifications in the GnuTLS library, without the need to modify the browser software or any other libraries, or any specialized infrastructure. Both the client and the server components of our prototype are written in C, and our modifications took around a thousand lines of code.

8 EVALUATION

In this section, we measure the performance of our prototype. We are particularly interested in measuring the latency cost of our approach compared to the traditional state-of-the-art approach of OCSP.

8.1 Experimental Environment

Our experimental infrastructure includes two different linux virtual machines: one in a datacenter located in Frankfurt, Germany, and another in a datacenter located in Amsterdam, the Netherlands. The hypervisors are running KVM, equipped with two Intel Xeon E5-2620 v3 CPUs, DDR4 1866 MHz ECC memory, and two 40 GbE network interfaces. The virtual machines have a single logical core, 508 MB of RAM, and 20 GB of RAID-10 SSD storage. The two datacenters are connected with a dedicated private fiber connection implemented using MPLS. There are seven IP routers between the two VMs.

8.2 Performance Evaluation

To evaluate our prototype in terms of latency, we created two webpages of different sizes: (a) one webpage of size 1 MByte and (b) one of size 2 MBytes. In order to compare our approach with the state-of-the-art approaches, we fetch each webpage using both (i) OCSP stapling and (ii) CCSP.

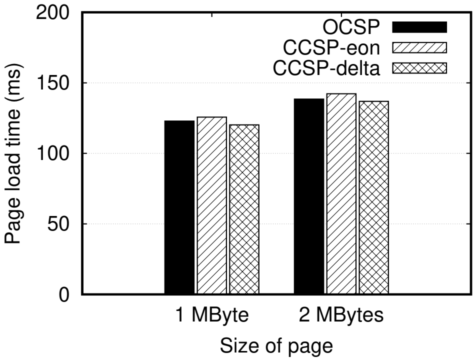

Fig. 6. TLS handshake latency constitutes a quite small part of the overall page load time.In case of CCSP-delta (epoch) the total CCSP latency is slightly lower than the OCSP which have minimal impact to the user experience. Even in case of CCSP-eon where the client downloads revocation information for as many as 1M certificates, the imposed latency is utterly insignificant compared to the overall page load time and is unable to affect the user experience.

To achieve that, we use the vanilla GnuTLS-loaded `wget` tool for the OCSP stapling case and `wget` loaded with our modified GnuTLS version for the CCSP case.

In our first measurement, we measure the time it takes for each of the two approaches to perform a TLS handshake. In CCSP we have two cases: (i) the beginning of each eon (CCSP-eon), where the client retrieves the entire Signed Collection and (ii) the delta case (CCSP-delta), where the client has only to download the deltas. In case of CCSP-delta the client needs to download only the most current delta (current epoch) since it contains all the new revocation information since this time inside the eon. We run each experiment 1000 times and in Figure 5, we report the median values. As expected, TLS handshake does not depend on the webpage size. In addition, we see that CCSP-eon completes a TLS handshake 6 ms slower than OCSP Stapling (solid black bar). That is expected since the client retrieves the entire Signed Collection (1M certificates), which is much larger in size than the stapled OCSP response of OCSP Stapling (4 KBytes vs 0.5 to 1 KByte). However, CCSP-eon appears only once per eon; on the contrary, we see that the more frequently appeared CCSP-delta (once per epoch) completes a TLS handshake 1 ms faster than OCSP Stapling.

In our next experiment we would like to see if the three different algorithms result in significant changes in page load time. Similarly to the experiment before, we run each experiment 1000 times and we report in Figure 6 the median values. We immediately notice that all three approaches have almost the same performance which hovers around 130 to 140 ms. It seems TLS handshake time is too small compared to the overall page download time, and thus, any differences in TLS handshake latency between the three algorithms is hardly visible in the overall page load time.

8.3 Summary

To summarize and put our findings in perspective, Table 2 compares CCSP and the most widespread previous approaches: CRL, OCSP, OCSP-CDN, and OCSP Stapling along four important dimensions: *(i)* privacy, *(ii)* number of signatures required, *(iii)* latency, and *(iv)* freshness of information. We see that CCSP (i) along with OCSP Stapling and CRLs protect user's privacy, (ii) requires much less signatures than all versions of OCSP, (iii) achieves low latency, and (iv) provides fresh (timely) information. We see that along all dimensions (i.e., privacy, signatures, latency and freshness), CCSP is comparable to or exceeds the best of the rest of the approaches. Indeed, no approach of the mentioned above comes close to CCSP along all the dimensions studied.

Method	Privacy	Low Number of Signatures	Low Latency	Freshness of Information
CRLs [10]	✓	✓	≈	≈
OCSP-CDN [2]	✗	✗	✓	≈
OCSP [33]	✗	✗	✗	✓
OCSP-Stapling	✓	≈	✓	≈
CCSP	✓	✓✓	✓	✓

Table 2. Comparison of certificate revocation approaches along four dimensions: (i) privacy, (ii) low number of signatures, (iii) low latency, and (iv) freshness of information. We compare five different approaches CRLs, OCSP-CDN, OCSP, OCSP-stapling, and CCSP. When an approach scores very well in a particular dimsension we give it a ✓. When it scores badly, we give it a ✗. When it scores so and so, we give it a ≈. For example, we see that CRLs have good privacy and a low number of signatures, but are "so and so" in latency and freshness of information. On the other hand, OCSP is very bad in privacy, latency, and number of signatures, but it provides fresh information. CCSP seems to score very well in all dimensions: good privacy, low latency, very small number of signatures, and very fresh information.

9 DISCUSSION

In this section, we discuss various aspects of our approach. We perform a head-to-head comparison with the traditional OCSP in terms of performance, preservation of privacy, overhead for CAs, etc. Among others, we discuss how CCSP handles (i) expired and (ii) corrupted certificates, and how easy it is for CCSP to be adopted by contemporary web clients.

9.1 CCSP Vs. OCSP: Head-to-head

To avoid man-in-the-middle replay attacks, OCSP enriches each and every OCSP request with a nonce and *signs each and every reply* including the nonce in the signature. However, this use of nonces by OCSP has two major disadvantages:

- **Loss of privacy:** If web clients check each and every certificate with the OCSP server, then OCSP servers can learn the browsing history of web clients.

- **Low Performance:** recent OCSP requests are served by CDNs (Content Delivery Networks) in just a few tens of milliseconds (compared to hundreds of milliseconds that OCSP servers need to reply). Unfortunately, requests that contain a nonce are treated by CDNs as *cache misses.* This implies that they are *not* served by the fast CDNs but they are served by the slower OCSP servers.

We believe that the introduction of timestamped Signed Collections can substitute the use of nonces. That is, instead of requesting signed nonces, web clients may request timestamped Signed Collections. Alternatively, one may see this as defining the nonce to be the current *epoch* number. The introduction of timestamped Signed Collections in OCSP has several advantages:

- **Preservation of Privacy:** OCSP servers will know that web clients are interested in one of the web sites of a Signed Collection but they will not know *which* one. If there is a large number of web sites in a collection (one million or more) then the OCSP server can get practically no information on what the users are interested in. [8]

[8] Note that it is the existence of the Signed Collections that preserves privacy here. The larger the collection, the better the privacy is preserved.

- **Fast Response:** OCSP responses with timestamped Signed Collections can be served by CDNs. Since they are *already signed*, they do not need to be forwarded to the OCSP server and they can be treated as a cache hit by nearby CDNs.
- **Lower number of signatures:** Timestamped Signed Collections require one signature per one million sites per epoch (assuming that each Signed Collection contains revocation information for one million certificates). This can be one signature per minute (assuming that each epoch lasts for one minute). On the contrary, nonces require one signature per web site per client request - several orders of magnitude more signatures. In the case of OCSP Stapling, that's one signature for every website, for every epoch, which at the current epoch of one week is tens of millions of signatures. Reducing the one week time frame to one minute, like CCSP, will cause tens of millions of signatures per minute, making this technically impossible for a CA to perform.

Therefore, the use of timestamped Signed Collections instead of nonces, can be used by OCSP to *(i)* alleviate worries about the user's privacy, *(ii)* improve performance via using CDNs, and *(iii)* significantly reduce the number of signatures required.

9.2 Selective frequently timestamped OCSP responses

In Section 3.3, we discuss why the alternative approach of timestamping OCSP responses as frequent as CCSP would impose a massive overhead on CAs. At this point, one might suggest to follow a more selective approach and frequently timestamp responses for only a subset of web sites – the ones with "strong security requirements". However, again in order for this to have a computation overhead (in terms of signature operations) comparable to that of CCSP, it would need to restrict its application to about one millionth of all web sites and would leave the rest 99,9999% vulnerable to attacks. And although one might argue that not all web sites might be a target for this attack and thus, their users are safe from the attackers, the recent Mirai botnet and the Wannacry malware attacks proved that attackers would compromise anything that can be compromised. Thus, no vulnerable target can be considered "safe". And to make matters worse, in case of a compromised web hosting service, then hundreds of thousands of web sites would be at risk as their private keys could be leaked.

9.3 CCSP and Certificate Revocation Lists

We believe that CCSP can significantly improve the performance of more traditional approaches, such as Certificate Revocation Lists (CRLs). CCSP has demonstrated that revocation information for as many as 1M web sites can fit in as little as 10 KBytes of space. This can mean a breakthrough for CAs that use CRLs. Indeed, Signed Collections imply that CAs now may send information about *all* their certificates (not just the *revoked* ones) in just a few KBytes of space. If combined with deltas the space requirements will be reduced to less than 1 KByte. Thus, instead of needing tens or hundreds of KBytes, CRLs (with appropriate compression) may now be transferred in less than 1 KByte - one to two orders of magnitude space improvement. Another benefit of CCSP that comes with its size reduction is freshness. If users have to download smaller files, it means that these files can be downloaded more frequently, and if they also contain information about non-revoked certificates, it means that they have to be downloaded less times. This is one of the reasons that CCSP focuses on freshness with epochs and eons.

9.4 Handling expired certificates

It could be argued that a Signed Collection may eventually contain non-useful/redundant information since all the issued certificates expire after a specific period of time. Indeed, this would be the case if the Certificate Authorities do not take into account the certificates' expiration date during the construction of Signed Collections and assign certificates to Signed Collections at random (i.e., include in the same Signed Collection certificates with very large variance in expiration dates). When considering that the certificate validity check should fail when a certificate under validation is found to be expired, without even the need to further check for its revocation status, these expired certificates could consume some space in the Signed Collection.

However, as the expiration dates of certificates are not specified arbitrarily (a certificate can only be valid for the exact period of three months, one or two years), CAs construct Signed Collections that contain certificates expiring approximately at the same period. Furthermore, even if the Signed Collections are not constructed properly, the waste of space is minimal after considering the compression of Signed Collections. Even better, *since we do not need revocation information for these expired certificates, the revocation bits that correspond to these certificates can be set to the values that maximize the effect of compression!*. For example, if the revocation bit of an expired certificate is neighbouring to a sequence of bits that correspond to already revoked certificates, the compression will be more effective if this bit is also assigned to 1, which is permitted since the certificate is expired and this operation will not interfere with the validity check.

9.5 How about corrupted CAs?

Recent incidents [7] suggest that there may exist corrupted CAs. These CAs may issue fake new certificates and/or may "un-revoke" several revoked certificates. Unfortunately, this is true: rogue CAs may significantly compromise any communication that uses their "signed" certificates. Although an important problem, dealing with corrupted CAs is outside the scope of this paper. Fortunately, there have been several approaches (as described in section 2) that deal with corrupted CAs.

9.6 Why should the client care for 1 million certificates?

In this paper, we present an approach on how to deal with millions of certificates. However, the vast majority of the clients may access only a few dozen of websites and thus care about the freshness of those certificates only. As a result, one might feel that the proposed approach may add unnecessary overhead. Fortunately, CCSP has the same overhead independent of whether the client accesses one hundred or one million sites. This is because with the use of (i) compression and (ii) deltas we managed to reduce the transferred information to just a few KBytes. Consequently, any further reductions may provide no really visible improvements.

9.7 Adoptability of the proposed approach

It is apparent that similar to any other certificate verification checking approach, CCSP, also requires modifications in the existing infrastructure. More specifically:

- Traditional OCSP requires modifications on the browser and the CA: the browser needs a module to perform OCSP queries and CAs need to maintain stand-alone OCSP responders to provide timely revocation information.

- Traditional CRL lists also require modifications on the browser and the CA: the browser needs a module to fetch, store and search large CRL lists and CAs need to maintain servers where CRLs are publicly accessible.

- OCSP Stapling requires modifications on the browser and lots of modifications on the website side: the web server has to periodically poll OCSP server for revocation status of its own certificate(s), and send OCSP response along with the certificate to the browser during TLS handshake.

- CRLSet (or OneCRL) requires heavy modifications on the browser: the vendor needs to regularly crawl CRLs from the major CAs around the world and compose its own comprehensive list. Contrary to traditional CRL, this list does not include of end-server leaf certificates but only high value intermediate CA certificates.

Similar to CRLs, in CCSP, the browser needs to fetch, store, and maintain a small list of bits (the *Signed Collection*), when CAs need to compose and distribute these lists of bits. Considering the minimal amount of changes required compared to the CRL approach, and the increased necessity for a reliable certificate verification checking method [39], we believe that browsers and CAs will be highly motivated to adopt the principles of CCSP. CCSP has been designed by always keeping in mind that changes to existing software should be kept to a minimum. That said, it only requires some changes in the user agents, such as the browsers, and not in the server-side, as a simple static file web server can work just fine.

10 CONCLUSION

In this paper we introduced a new approach for certificate revocation checking: CCSP. Based on the newly-introduced abstraction of *signed collections*, CCSP is able to resist man-in-the-middle attacks with much better performance than previously-proposed approaches. Indeed, by using bitmaps and aggressive time- and space-based compression, CCSP is able to pack revocation information about one million certificates in less than 10 KBytes. CCSP significantly reduces the number of signature operations needed by OCSP servers and CAs - by as much as six orders of magnitude in some cases.

We implemented our approach with around a thousand lines of code inside the popular GnuTLS library and performance results suggest that CCSP adds only a few milliseconds of overhead in the overall user latency compared to the state-of-the-art alternative, OCSP stapling. We believe that with the increasing percentage of encrypted Internet Traffic, and the widespread awareness about certificate validity, the benefits of our approach are bound to increase in the future.

11 ACKNOWLEDGMENTS

This work was supported by the project GCC, funded by the Prevention of and Fight against Crime Programme of the European Commission – Directorate-General Home Affairs under Grant Agreement HOME/2011/ISEC/AG/INT/4000002166, the FP7 project iSocial ITN, funded by the European Commission under Grant Agreement No. 316808 and the project SHARCS, under Grant Agreement No. 644571. In addition, this project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 690972. The paper reflects only the authors' view and the Agency is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] Alfred Ng. 2018. Google Chrome says goodbye to green 'Secure' lock on HTTPS sites. https://www.cnet.com/news/say-good-bye-to-that-green-secure-lock-on-google-chrome/.

[2] Richard F Andrews and Quentin Liu. 2013. Accelerating ocsp responses via content delivery network collaboration. http://www.google.com/patents/US20150100778 US Patent App. 14/050,245.

[3] APNIC Labs. 2019. Use of DNSSEC Validation for World (XA). https://stats.labs.apnic.net/dnssec/XA?c=XA&x=1&g=1&r=1&w=7&g=0.

[4] Antonios A Chariton, Eirini Degkleri, Panagiotis Papadopoulos, Panagiotis Ilia, and Evangelos P Markatos. 2016. DCSP: Performant Certificate Revocation a DNS-based approach. In *Proceedings of the 9th European Workshop on System Security*. ACM, 1.

[5] Antonios A Chariton, Eirini Degkleri, Panagiotis Papadopoulos, Panagiotis Ilia, and Evangelos P Markatos. 2017. CCSP: A compressed certificate status protocol. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.

[6] Jing Chen, Shixiong Yao, Quan Yuan, Kun He, Shouling Ji, and Ruiying Du. 2018. CertChain: Public and efficient certificate audit based on blockchain for TLS connections. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2060–2068.

[7] Richard Chirgwin. 2016. Google publishes list of Certificate Authorities it doesn't trust. https://www.theregister.co.uk/2016/03/23/google_now_publishing_a_list_of_cas_it_doesnt_trust/.

[8] Taejoong Chung, Jay Lok, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, John Rula, Nick Sullivan, and Christo Wilson. 2018. Is the Web Ready for OCSP Must-Staple?. In *Proceedings of the Internet Measurement Conference 2018* (Boston, MA, USA) *(IMC '18)*. ACM, New York, NY, USA, 105–118. https://doi.org/10.1145/3278532.3278543

[9] Taejoong Chung, Jay Lok, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, John Rula, Nick Sullivan, and Christo Wilson. 2018. Is the Web Ready for OCSP Must-Staple?. In *Proceedings of the Internet Measurement Conference 2018*. ACM.

[10] Cooper Dave, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. 2008. Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile. https://tools.ietf.org/html/rfc3280. (2008).

[11] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. [n.d.]. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (Vancouver, BC, Canada). 14. https://doi.org/10.1145/2663716.2663755

[12] Donald Eastlake et al. 2011. Transport layer security (TLS) extensions: Extension definitions. https://tools.ietf.org/html/rfc6066.

[13] C. Ellison and B. Schneier. 2000. Ten Risks of PKI: What You're Not Being Told About Public-Key Infrastructure. *Computer Security Journal* 16, 1 (2000), 1–7. https://www.schneier.com/academic/paperfiles/paper-pki.pdf

[14] Emily Schechter. 2018. Evolving Chrome's security indicators. https://blog.chromium.org/2018/05/evolving-chromes-security-indicators.html.

[15] S. Golomb. 1966. Run-length encodings. *IEEE Transactions on Information theory* 12, 3 (1966), 399–401.

[16] Dan Goodin. [n.d.]. Qualys endorses alternative to crappy SSL system. http://www.theregister.co.uk/2011/09/30/qualys_endorses_convergence/.

[17] Mark Goodwin. 2015. Revoking Intermediate Certificates: Introducing OneCRL. https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/.

[18] Roger A. Grimes. [n.d.]. The sorry state of certificate revocation. https://www.csoonline.com/article/3000574/security/the-sorry-state-of-certificate-revocation.html.

[19] Phillip Hallam-Baker. 2015. X. 509v3 Transport Layer Security (TLS) Feature Extension. https://tools.ietf.org/html/rfc7633.

[20] A. Herzberg and H. Shulman. 2013. Fragmentation Considered Poisonous, or: One-domain-to-rule-them-all.org. In *IEEE Conference on Communications and Network Security (CNS)*.

[21] Paul Hoffman and Jakob Schlyter. 2012. The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA. https://tools.ietf.org/html/rfc6698.

[22] Adam Langley. [n.d.]. Revocation checking and Chrome's CRL. https://www.imperialviolet.org/2012/02/05/crlsets.html.

[23] Adam Langley. [n.d.]. Smaller than Bloom filters. https://www.imperialviolet.org/2011/04/29/filters.html.

[24] James Larisch, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. 2017. CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers.

[25] Ben Laurie and Emilia Kasper. 2016. Revocation Transparency. http://www.links.org/files/RevocationTransparency.pdf.

[26] B. Laurie, A. Langley, and E. Kasper. [n.d.]. Certificate Transparency. https://tools.ietf.org/html/rfc6962.

[27] Ben Laurie, Adam Langley, and Stephen McHenry. [n.d.]. Certificate Transparency. https://www.certificate-transparency.org/faq.

[28] Let's Encrypt. 2018. Percentage of Web Pages Loaded by Firefox Using HTTPS. https://letsencrypt.org/stats/.

[29] LetâĂŹs Encrypt. [n.d.]. Let's Encrypt Stats. https://letsencrypt.org/stats/.

[30] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. [n.d.]. An End-to-End Measurement of Certificate Revocation in the Web's PKI. In *Proceedings of the 2015 ACM Internet Measurement Conference* (Tokyo, Japan). 14. https://doi.org/10.1145/2815675.2815685

[31] Moxie Marlinspike. [n.d.]. Convergence. http://www.convergence.io/details.html.

[32] Nikos Mavrogiannopoulos and Simon Josefsson. [n.d.]. The GnuTLS Transport Layer Security Library. http://www.gnutls.org/.

[33] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. 1999. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. https://tools.ietf.org/html/rfc6960.

[34] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. 2014. The cost of the S in HTTPS. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 133–140.

[35] Netcraft. [n.d.]. CRL sites ordered by average body size. http://uptime.netcraft.com/perf/reports/performance/CRL.

[36] Netcraft. [n.d.]. Total http time of OCSP sites. http://uptime.netcraft.com/perf/reports/performance/OCSP.

[37] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. 2010. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review* 44, 3 (2010), 2–19.

[38] Mark Dermot Ryan. 2014. Enhanced Certificate Transparency and End-to-End Encrypted Mail. In *21st Annual Network and Distributed System Security Symposium, NDSS*. http://www.internetsociety.org/doc/enhanced-certificate-transparency-and-end-end-encrypted-mail

[39] Alexey Samoshkin. [n.d.]. SSL certificate revocation and how it is broken in practice. https://medium.com/@alexeysamoshkin/how-ssl-certificate-revocation-is-broken-in-practice-af3b63b9cb3.

[40] Aaron Schulman, Dave Levin, and Neil Spring. 2014. RevCast: Fast, Private Certificate Revocation over FM Radio. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*.

[41] Giuseppe Scrivano and Hrvoje Niksic. [n.d.]. GNU Wget 1.18 Manual. https://www.gnu.org/software/wget/.

[42] Nick Sullivan. [n.d.]. High-reliability OCSP stapling and why it matters. https://blog.cloudflare.com/high-reliability-ocsp-stapling/.

[43] Ahmad Samer Wazan, Romain Laborde, David W Chadwick, François Barrère, and Abdelmalek Benzekri. 2017. TLS Connection Validation by Web Browsers: Why do Web Browsers still not agree?. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. IEEE.

[44] Dan Wendlandt, David G. Andersen, and Adrian Perrig. 2008. Perspectives: Improving SSH-style Host Authentication with Multi-path Probing. In *USENIX 2008 Annual Technical Conference* (Boston, Massachusetts) *(ATC'08)*. USENIX Association, Berkeley, CA, USA, 321–334. http://dl.acm.org/citation.cfm?id=1404014.1404041

[45] Liang Zhang, David Choffnes, Dave Levin, Tudor Dumitras, Alan Mislove, Aaron Schulman, and Christo Wilson. 2014. Analysis of SSL certificate reissues and revocations in the wake of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 489–502.

[46] Liang Zhu, Johanna Amann, and John Heidemann. 2016. Measuring the latency and pervasiveness of TLS certificate revocation. In *International Conference on Passive and Active Network Measurement*. Springer, 16–29.

[47] Liang Zhu, Duane Wessels, Allison Mankin, and John Heidemann. 2015. Measuring dane tlsa deployment. In *International Workshop on Traffic Monitoring and Analysis*. Springer, 219–232.

[48] J. Ziv and A. Lempel. 2006. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Inf. Theor.* 23, 3 (Sept. 2006), 337–343. https://doi.org/10.1109/TIT.1977.1055714