CCSP: a Compressed Certificate Status Protocol

Antonios A. Chariton*, Eirini Degkleri*†, Panagiotis Papadopoulos*†, Panagiotis Ilia*†, and Evangelos P. Markatos*†
*University of Crete, Greece, csd3235@csd.uoc.gr
†FORTH-ICS, Greece, {degleri, panpap, pilia, markatos}@ics.forth.gr

***Abstract*—Trust in SSL-based communications is provided by Certificate 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 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, may trick web browsers to trust a *revoked* certificate, believing that it is still valid. Consequently, the web browser will communicate (over TLS) with web servers controlled by cyber-attackers.**

Although frequently *updated*, *nonced*, and *timestamped* certificates may reduce the frequency and impact of such cyber-attacks, they impose a very large overhead to the CAs and OCSP servers, which now 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 overhead 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 the design, preliminary implementation, and evaluation of CCSP in general, and *signed collections* in particular. Our preliminary results suggest that CCSP (i) reduces space requirements by more than an order of magnitude, (ii) lowers the number of signatures required by 6 orders of magnitude compared to OCSP-based methods, and (iii) adds only a few milliseconds of overhead in the overall user latency.

I. Introduction

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), is the most popular standard for secure Internet communications nowadays. More and more web services are moving away from the traditional plaintext HTTP protocol to the more secure HTTPS. Indeed, recent results suggest that around 50% of the HTTP connections are currently being implemented over HTTPS [19]. Although the primary goal of TLS is to provide confidentiality and integrity for the vast majority of the 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. Responsible for *issuing*, *validating*, and *revoking* these digital certificates is a *Certificate Authority* (CA): a third party 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 the website. As a result, Certificate Authorities create a web of trust that enables complete strangers (i.e., a web client and a web server) 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 no more trust web servers using 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 by downloading the list of all revoked certificates, the Certificate Revocation List (CRL), from the CA and locally lookup if the certificate is included in the CRL. If included, it means that the certificate has been revoked and should not be trusted anymore. However, the increasing size of these CRLs [6] with median size of 51 KB, reaching as large as 76 MB in some cases [16], [20], 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 remedy this issue, i.e., to avoid downloading large CRLs every once in a while, 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 the 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 [21]. 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], [26], such as Akamai [22], managing to reduce their response time to less than a tenth of a second. In the same spirit, DCSP [3], 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 verify that the certificate is valid. That is, the *receiver (i)* has to download the certificate from a web site and *(ii)* has to verify that the certificate is still 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 provides a different type of solution: It

advocates 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 make it simple, 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, does not force the client to contact any third-party services, and reduces the overhead to the minimum possible.

Unfortunately, OCSP Stapling is susceptible to *man-in-the-middle attacks*. Indeed, an attacker who has managed to steal the private key of a web server may provide a victim browser with the 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 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 may solve man-in-the-middle (MITM) attack replays by minimizing the window of vulnerability, they also impose a tremendous load on the OCSP responders, which now need to timestamp and sign revocation information for each and every issued certificate every few seconds or so.

To address the overheads imposed by frequent timestamps and signatures, in this paper, we propose CCSP (Compressed Certificate Status Protocol): a new approach for timestamping and signing Certificate Revocation responses. CCSP is based on *signed collections* which provide revocation information *not for a single certificate* (like 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 2 orders of magnitude storage space compared to traditional CRLs, and more than 6 orders of magnitude signatures compared to traditional OCSP Stapling.

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.
- We present a preliminary implementation study of CCSP to allow us to explore the trade-offs of our approach.
- We present an evaluation of CCSP based on simulation and analysis. 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) it is able to reduce the number of required signatures by more than 6 orders of magnitude, and (iii) it requires an average traffic rate of only a few bytes per second.

[1] Certificate Transparency [14], explained in depth in Section VIII, provides a third piece of information as well: a signed proof that the certificate has been included in a publicly-accessible Log.

II. Related Work

A. Revoke the trust from Certificate Authorities

Although today 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 [4], for example, is a project which later inspired the Convergence [10], [17] strategy for replacing CAs; it employs a **crowd-sourcing 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 entity and consider the response of the majority, an operation that may sky-rocket the certificate verification latency.

Over the past few years, Certificate Transparency [14] (or simply CT) has been widely popular. Aimed 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 its fake certificates in some Logs it 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 and thus, its web clients will not accept its certificates. In both cases, *rogue CAs will not be able to continue their nefarious activities without being noticed.*

Although CT is an excellent way to deal with rogue CAs, it does *not* explicitly deal with revocation. Indeed, as noted in the FAQ of the CT's official website [15]: *"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 to integrate revocation in CT. For example, Revocation Transparency (RT) [13], provides a way to supply fresh revocation information. Unfortunately, careful studies of RT [23] 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 [23], it may still be high compared to other Certificate Revocation approaches that may respond in (average) constant time.

B. Call DNS to the rescue

DNS-based Authentication of Named Entities (DANE) [11] is another approach towards the direction of replacing the Certificate Authorities. DANE leverages 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 stores the *whole* certificate of a domain and uses DNSSEC to validate its integrity. Unfortunately, DANE is subject to MITM attacks. To mitigate such attacks, DANE may use timestamped DNS records which, in turn, place a heavy overhead in the signer of the certificate that have to timestamp and sign each and every record frequently. This burden is also transferred to the DNS infrastructure itself which now needs to deal with orders of magnitude higher load than previously.

DCSP [3], 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 this way their validity and freshness. DCSP 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 this way, *(i)* DCSP achieves better performance than traditional OCSP, and *(ii)* preserves the privacy of the user's browsing history. Its easy to anticipate though, that 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.

C. The Google Approach

Google Chrome and Chromium browser use *CRLsets*: a compressed list of a small set of revoked certificates [1]. 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 in the CRLset. Given that CRLsets cover less than 1% of revocations [16], there may be room for further improvements.

D. Where does CCSP fit in the spectrum?

We view that CCSP is not competing to, but it is complementing most of the previous approaches. By introducing the abstraction of *signed collections*, CCSP 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.

III. The Threat Model

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

By deploying such an attack, the adversary is able to assume the identity of a legitimate party, and by falsifying responses from DNS and/or HTTP endpoints, manage to successfully impersonate this party. As a consequence, an adversary can deceive users, making them to 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 client aware of the impersonation attack in order for her to seize the connection with the adversary soon enough to prevent any possible leakage of private information.

A. 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[2].

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 web site.

From that point onwards, the victim is in a downward spiral: they will probably reveal their password, disclose personal

[2]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.

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.

B. The Solution Framework

This is not a trivial problem to solve. 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* by 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 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 millions of certificates and are now required to *timestamp, sign, and distribute millions of certificates per second*. To reduce this overhead CCSP introduces *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.

IV. Design

A. 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 (which 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 created, the CA assigns it to a signed collection SC. The name of this signed collection, as well as the 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 at 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.

Compression. It seems that in a Signed Collection (a bitmap) of size 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*.

i. Space: If we take a careful look at a signed collection SC, we realize that most of the bits in the bitmap $SC[]$ are "0". This should be expected: 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 [16], reaching as low as 0.2% in some cases [7]. This implies that roughly more than 99% of the bits in the $SC[]$ bitmap are "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 compression algorithm which is a variation of LZ77, and *(ii)* the Golomb algorithm [9].

ii. 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 current eon.

B. Detailed Design

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 "http://" or "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.

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 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.

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.

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, as a 64-bit unsigned integer
- COMPRESSION: the ID of the compression type used in the bitmap, [3] 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

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*

[3]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 I. Summary of Notation

A delta can be computed in two ways. It depends on the CA and can change between epochs. 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 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.

V. Implementation

To assess the feasibility and effectiveness of CCSP, we implemented a preliminary prototype of our approach. Our implementation is based on *rapid prototyping* by putting together *off-the-self components* including: (i) the `wget` open source tool [24] to retrieve web pages and (ii) the GnuTLS [18] library, to allow us to perform TLS handshakes, validate the received certificates and check their status (either by contacting the CAs' OCSP server or by using the stapled OCSP responses.).

At first, we modified the server-side component of GnuTLS to provide custom certificate status responses during a TLS connection, and its associated client-side component to properly handle and parse these custom responses. After that, we set up a standalone server, configured to serve the CCSP responses. For the client side, we compiled `wget` using the modified version of the GnuTLS library. An example usage includes a client, which connects to the web server via `wget` and starts a TLS handshake. The server provides the Signed Collection (and delta) to `wget`, which in turn, passes it to the GnuTLS library (client-side) for validation.

Our preliminary implementation suggests that the necessary changes are confined to no more than a few hundred lines of code (excluding the libraries for compression and signing which are already available).

VI. Analysis

For the purposes of system description so far, 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 anymore. In this section, we will try to find an optimal value for the size of the *eon*.

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

- 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, we will have sent kx deltas (recall that we assume that we have 1 revocation every k epochs) with a total size of:

$$\sum_{i=0}^{x} kir = \frac{krx(x-1)}{2}$$

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 per epoch 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 transfered per epoch (from equation 1) would be:

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

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

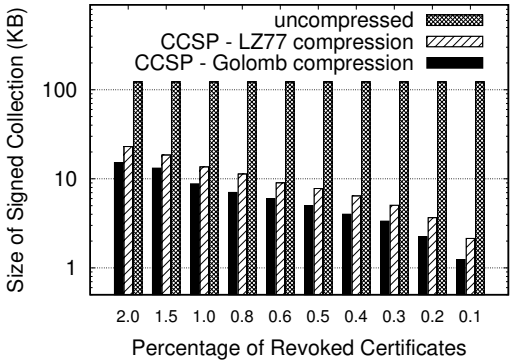

Fig. 1: Size of a Signed Collection of 1M Revocation Bits. Without compression it would require 122 KB to be stored. Both compression algorithms of CCSP can reduce the size needed by one to two orders of magnitude.

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, the above equation 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.

VII. Simulations

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

At first, we set out to explore how effective compression is. 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) = 122.07 KB 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 KB.

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 random sequence of "1" and "0" generated by a perfect 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 which will be "one", 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 [7].

Figure 1 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

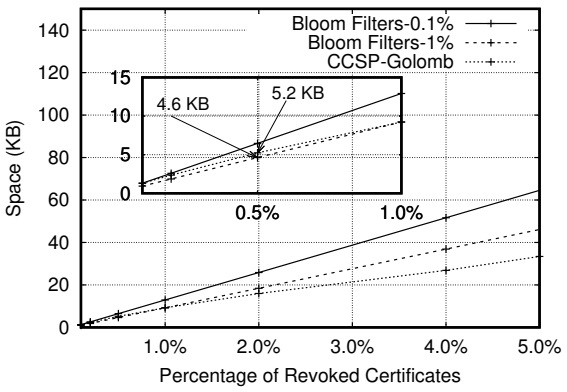

Fig. 2: Space required to store revocation information for one million certificates.

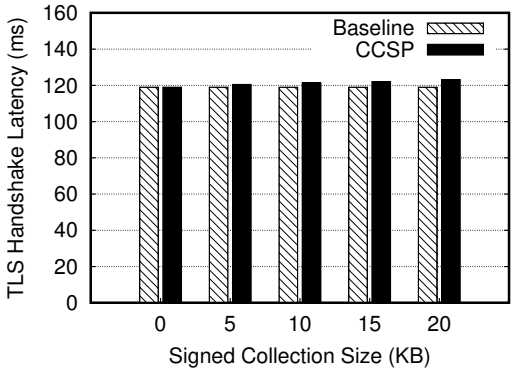

Fig. 3: TLS Handshake Latency. CCSP adds very little latency: 2.5 ms for signed collection size of 5 KB, while it adds less than 5 ms for very large signed collections of 20 KB.

thing we notice it 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 KB. For revocation rate close to 0.2% the space requirements for CCSP-Golomb drops to just a bit higher than 1 KB: two orders of magnitude less space than the "no compression" case. CCSP-LZ77 is a bit higher but stays in the same range. That means 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 KB 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.

> **Finding**: 1 KB of space is all you need to store revocation information for one million certificates.

B. 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 [16] 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 [12]). Note that the actual space 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 2 plots 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 KB - as expected. This size increases with the higher percentage of revoked certificates and reaches a bit more than 30 KB 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 KB of space, while "Bloom Filters-1%" require 4.6 KB, and "Bloom Filters-0.1%" require 6.4 KB. 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.

C. What if Heartbleed happens again?

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 recent Heartbleed bug which forced a large number of certificates to be revoked [5]. 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 [5]. 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 about 20 revocations per minute, or about 20 revocations per *epoch*, since an epoch is about one-minute long. These revocations will create 71 deltas with a total size of: $\sum_{i=1}^{60} 20 \times i \times 4 = 138\,KB$, which will create a traffic of about 0.032 KB per second. Adding to this the traffic needed to transfer one bitmap per *eon* ($=1,000,000/8/72/60/2014 = 0.028$ KB per second). Thus, the total traffic that will be created is $0.032 + 0.028 = 0.06\,KB$ 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 0.06 KB per second. Given the high capacity networks that CDNs employ, it seems that this 0.06 KB per second is a tiny percentage they should not worry about. Given that each *eon* contains around 71 epochs, and that each *epoch* contains around 60 seconds, these 138 KB would amount for $138/71/60 = 0.032$ KB per second.

> **Finding**: Even during a new, Heartbleed bug the traffic generated due to CCSP-based revocation information will be only around 0.03 KB per second.

D. Latency

In this section we emulate the latency added by the *signed collections*. We use OpenSSL to connect to a web server and measure the latency incurred for various sizes of *signed collections*. We see (Figure 3) that when we use OCSP Stapling without signed collections, the TLS handshake latency is about 120 ms. When we add signed collections, the latency increases only by a handful of milliseconds. We must note, however, that Figure 3 reports only the TLS handshake latency. If we factor in the data transfer latency as well (i.e., the time it takes to transfer an entire web page) the overhead of *signed collections* will be much smaller in relative terms. In terms of size, the average TLS handshake is about 4 KB, out of which 0.5 KB is the Stapled OCSP Response. Therefore, a 5 KB Signed Collection would make the TLS handshake 8.5 KB, which is insignificant compared to the average size of a website, which is about 2 MB [8] [25].

> **Finding**: CCSP adds only 2-3 milliseconds in the overall end user latency compared to OCSP stapling.

E. Summary

To summarize and put our findings in perspective, Table II 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 above mentioned ones comes close to CCSP along all the dimensions studied.

Method	Privacy	Low Number of Signatures	Low Latency	Freshness of Information
OCSP-CDN	✗	✗	✓	≈
OCSP	✗	✗	✗	✓
OCSP Stapling	✓	≈	✓	≈
CRLs	✓	✓	≈	≈
CCSP	✓	✓✓	✓	✓

TABLE II. Summarizing table of the comparison results

VIII. Discussion

A. OCSP

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. This use of nonces and 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 (say 1 million) then the OCSP server can get practically no information on what the users are interested in.
- **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. On the contrary, nonces require one signature per web site per client request - several orders of magnitude more signatures.

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.

B. CRLs

CCSP has demonstrated that revocation information for as many as 1M sites can fit in as little as 10 KB 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 KB of space. If combined with deltas the space requirements will be reduced to less than 1 KB. Thus, instead of needing tens or hundreds of KB, CRLs (with appropriate compression) may now be transferred in less than 1 KB - 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.

C. How about corrupted CAs?

Recent incidents 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 II) that deal with corrupted CAs.

D. 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.

IX. Conclusion

In this paper we introduced a new approach for certificate revocation checking: CCSP. Based on the newly-introduced notion 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 KB. 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 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.

X. 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/40000 02166, 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] Adam Langley. Revocation checking and Chrome's CRL. https://www.imperialviolet.org/2012/02/05/crlsets.html.
[2] R. F. Andrews and Q. Liu. Accelerating ocsp responses via content delivery network collaboration, Oct. 9 2013. US Patent App. 14/050,245.
[3] A. A. Chariton, E. Degkleri, P. Papadopoulos, P. Ilia, and E. P. Markatos. DCSP: Performant certificate revocation a dns-based approach. In *Proceedings of the 9th European Workshop on System Security*, EuroSec '16, 2016.
[4] A. P. Dan Wendlandt, David G. Andersen. Perspectives: Improving ssh-style host authentication with multi-path probing. In *USENIX 2008 ATC*.
[5] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson. The matter of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*.
[6] C. Ellison and B. Schneier. Ten risks of PKI: What you're not being told about public-key infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.
[7] L. Encrypt. Let's encrypt stats. https://letsencrypt.org/stats/.
[8] Gigaom. The overweight web: Average web page size is up 15% in 2014. https://gigaom.com/2014/12/29/the-overweight-web-average-web-page-size-is-up-15-in-2014/.
[9] S. Golomb. Run-length encodings. *IEEE Transactions on Information theory*, 12(3):399–401, 1966.
[10] D. Goodin. Qualys endorses alternative to crappy ssl system. http://www.theregister.co.uk/2011/09/30/qualys_endorses_convergence/.
[11] P. Hoffman and J. Schlyter. The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA. https://tools.ietf.org/html/rfc6698, 2012.
[12] A. Langley. Smaller than bloom filters. https://www.imperialviolet.org/2011/04/29/filters.html.
[13] B. Laurie and E. Kasper. Revocation transparency. http://www.links.org/files/RevocationTransparency.pdf.
[14] B. Laurie, A. Langley, and E. Kasper. Certificate transparency. https://tools.ietf.org/html/rfc6962.
[15] B. Laurie, A. Langley, and S. McHenry. Certificate transparency. https://www.certificate-transparency.org/faq.
[16] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson. An End-to-End Measurement of Certificate Revocation in the Web's PKI. In *Proceedings of the 2015 ACM Conference IMC*.
[17] M. Marlinspike. Convergence. http://www.convergence.io/details.html.
[18] N. Mavrogiannopoulos and S. Josefsson. The gnutls transport layer security library. http://www.gnutls.org/.
[19] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The cost of the "s" in https. In *Proceedings of the 10th ACM International CoNEXT '14*.
[20] Netcraft. Crl sites ordered by average body size. http://uptime.netcraft.com/perf/reports/performance/CRL.
[21] Netcraft. Total http time of ocsp sites. http://uptime.netcraft.com/perf/reports/performance/OCSP.
[22] E. Nygren, R. K. Sitaraman, and J. Sun. The akamai network: A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*
[23] M. D. Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In *21st Annual Network and Distributed System Security Symposium, NDSS*, 2014.
[24] G. Scrivano and H. Niksic. Gnu wget 1.18 manual. https://www.gnu.org/software/wget/.
[25] SOASTA. Page bloat update: The average web page is more than 2 mb in size. https://www.soasta.com/blog/page-bloat-average-web-page-2-mb/.
[26] L. Zhu, J. Amann, and J. S. Heidemann. Measuring the latency and pervasiveness of TLS certificate revocation. In *Proceedings of the 17th International Conference, PAM'16*.