

Privacy-Preserving Revocation Checking*

M. Narasimha, J. Solis and G. Tsudik[†]
Computer Science Department
University of California, Irvine
{mnarasim,jsolis,gts}@ics.uci.edu

Abstract

Digital certificates signed by trusted certification authorities (CAs) are used for multiple purposes, most commonly for secure binding of public keys to names and other attributes of their owners. Although a certificate usually includes an expiration time, it is not uncommon that a certificate needs to be revoked prematurely. For this reason, whenever a client (user or program) needs to assert the validity of another party's certificate, it performs a certificate revocation check. There are several revocation techniques varying in both the operational model and underlying data structures. One common feature is that a client typically contacts some third party (whether trusted, untrusted or semi-trusted) and obtains some evidence of either revocation or validity (non-revocation) for the certificate in question.

While useful, revocation checking can leak sensitive information. In particular, third parties of dubious trustworthiness can discover the identity of the party performing the revocation check, as well as the target of the check. The former can be easily remedied with techniques such as onion routing or anonymous web browsing. Whereas, hiding the target of the query is not obvious. This paper focuses on the privacy in revocation checking, explores the loss of privacy in current revocation checking techniques and proposes simple and efficient privacy-preserving techniques for two well-known revocation methods.

Keywords: *Privacy-Preserving Revocation Checking, Anonymity and Privacy, Revocation, Certificate Revocation Lists, Certificate Revocation Trees.*

1. Introduction and Motivation

Public key cryptography allows entities to establish secure communication channels without pre-established shared secrets. While entities can be assured that communication is confidential, there is no

guarantee of authenticity. Authenticity is obtained by binding a public key to some claimed identity or name which is later verified via digital signatures in conjunction with public key certificates (PKCs). A public key certificate, signed by a recognized certification authority (CA), is used to verify the validity, authenticity and ownership of a public key. As long as the issuing CA is trusted, anyone can verify the CA's certificate signature and bind the included name/identity to the public key. Public key certificates work best in large interconnected open systems, where it is generally infeasible to directly authenticate the owners of all public keys. X.509 [34] is one well-known certificate format widely used in several Internet-related contexts. The peer-based PGP/GPG [2, 11] format represents another popular approach.

Since a certificate is a form of a capability, one of the biggest problems associated with large-scale use of certificates is *revocation*. There are many reasons that can lead to a certificate being revoked prematurely. They include [34]: loss or compromise of a private key, change of affiliation or job function, algorithm compromise, or change in security policy. To cope with revocation, it must be possible to check the status of any certificate at any time.

Revocation techniques can be roughly partitioned into implicit and explicit classes. In the former, each certificate owner possesses a timely proof of non-revocation which it supplies on demand to anyone. Lack of such a proof implicitly signifies revocation. An example of implicit revocation is Micali's Certificate Revocation System (CRS) [25]. Most revocation methods are explicit, i.e., they involve generation, maintenance and distribution of various secure data structures that contain revocation information for a given CA or a given range of certificates.

Certificate Revocation Lists (CRLs) represent the most widely used means of explicit revocation checking. Each certificate issuer periodically generates a signed list of revoked certificates and publishes it at (usually untrusted) public directories or servers. Inclusion of a certificate in the list signifies explicit re-

*Portions of this paper appeared in [32] and [30].

[†]Corresponding Author.

vocation. Verifiers retrieve and cache the latest CRL and use it during certificate validation. Typically, a revoked certificate is included in a CRL from the time it is revoked until its validity period expires. Since certificate lifetime is typically measured in years, even modest revocation rates can result in very long accumulated CRLs. In bandwidth-constrained environments, transferring such CRLs can be expensive. Furthermore, since CRLs are published periodically, another potential concern is that many verifiers may request them around the time of publication. The burst of requests immediately following CRL publication may result in very high network traffic and can cause congestion. Thus, one of the biggest disadvantages of CRLs is the high cost associated with updating and querying the lists and this raises serious scalability concerns.

Other well-known explicit revocation methods include Certificate Revocation Trees (CRTs) [18] and Skip-Lists [12]. Another prominent technique is the On-line Certificate Status Protocol (OCSP) [27] which involves a multitude of validation agents (VAs) which respond to client queries with signed replies indicating current status of a target certificate. However, these explicit revocation methods have an unpleasant side-effect: they divulge too much information. Specifically, a third party (agent, server, responder or distribution point) of dubious trustworthiness knows: (1) the entity requesting the revocation check (source), and (2) the entity whose status is being checked (target). An even more important **loss of privacy** results from the third party tying the source of the revocation checking query to that query's target. This is significant, because revocation status check typically serves as a prelude to actual communication between the two parties. (We assume that communication between verifiers and on-line revocation agents (third parties) is private, i.e., conducted over secure channels protected by tools such as IPsec [15] or SSL/TLS [13, 9].)

Given the continual assault on privacy by governments, spammers and just plain hackers, privacy leakage in certificate revocation checking is an important issue worth considering. Consider, for example, a certain country with a less-than-stellar human rights record where mere intent to communicate (indicated by revocation checking) with an *unsanctioned* or *dissident* web-site may be grounds for arrest or worse. In the same vein, sharp increase in popularity (deduced from being a frequent target of revocation checking) of a web-site may lead authorities to conclude that something *subversive* is going on. Clearly, the problem can also manifest itself in other less sinister settings. For example, many internet service providers already keep detailed statistics and build elaborate profiles based on their clients' communication patterns. Current revoca-

tion checking methods – by revealing sources and targets or revocation queries – represent yet another source of easily exploitable (and misused) personal information.

Contributions: The primary motivation for this work is lack of privacy in current certificate revocation checking. The contribution of this paper is two-fold: first, it explores the loss of privacy inherent in current certificate revocation checking, and, second, it constructs simple, efficient and flexible privacy-preserving add-on components for two well-known revocation methods. We believe that, due to their simplicity, our techniques have a good chance of eventually being adopted by the Internet *masses* most of whom unfortunately ignore revocation checking at present.

Organization: Section 2 elaborates on the problem at hand, followed by the discussion of relevant prior work in Section 3. Section 5 describes our privacy-preserving technique based on CRTs and Section 6 presents its CRL-based counterpart. Next, the two methods are compared in Section ??, followed by future research directions and conclusions in Section 9. Appendix A overviews our implementation and Appendix B describes how to adapt our CRT-based technique for use with skip-lists.

2. Focus

Hiding the source of a revocation check can be easily achieved with modern anonymization techniques, such as onion routing [10], anonymous web browsing [26] or remailers [8]. While this would protect the source, the target of the check is still known to the third party or parties. Furthermore, although anonymization techniques are well-known in the research community, their overall penetration remains fairly low. Also, in order to take advantage of an existing anonymization infrastructure, a user either needs to place some trust in unfamiliar existing entities (e.g., remailers, re-webbers or onion routers) or make the effort to create and configure some of these entities.

In this paper we focus on the second problem – hiding the targets of revocation queries. We start by examining current revocation techniques and settle on two that are most amenable to supporting efficient privacy-preserving querying: Certificate Revocation Trees (CRTs) and Certificate Revocation Lists (CRLs).

We observe that the privacy problem of the type described above is not unique to revocation checking. A very similar problem arises in the context of a name

service, e.g., the Internet Domain Name System (DNS) [17]. In DNS, at least one (and potentially many) name servers become privy to both the source and target of a name-to-address resolution query. For the same reasons as revocation checking, information culled from DNS queries can be used for sinister, or at least unintended, purposes. In fact, the privacy problem in DNS is much more rampant and thus more important than that in revocation checking. This is because revocation checking is still a fringe activity among Internet users, in contrast to DNS which is used by nearly all.

3. Related Work

There is very little in terms of closely related work. However, this paper is not the first to consider privacy in revocation checking. The problem was initially brought up by Kikuchi [16]. This work identified the main issue (privacy loss) and proposed a fairly heavy-weight (inefficient) cryptographic technique specific to CRLs. The solution relies on so-called cryptographic accumulators [3] which are quite expensive.

Another relevant research topic is Private Information Retrieval (PIR) [5, 19]. PIR refers to a set of cryptographic techniques and protocols that – in a client-server setting – aim to obscure the actual target(s) of database queries from potentially malicious servers. Although PIR techniques could be applicable in our context, they tend to be relatively inefficient owing to either (or both) many communication rounds/messages or expensive cryptographic operations. As will be seen in subsequent sections, PIR techniques would amount to overkill in the context of privacy-preserving revocation checking.

4. Overview of Certificate Revocation Techniques

We now briefly overview some popular certificate revocation techniques and associated data structures. In the following, we refer to the entity validating certificates (answering certificate status queries) as a Validation Authority (VA). A distinct entity – Revocation Authority (RA) – is assumed responsible for actually revoking certificates, i.e., generating signed data structures, such as CRLs.

Strictly speaking, a certificate or a public key is never actually revoked. What is revoked is the binding between an identity string and a certificate serial number (which may contain a public key but does not have to, e.g. in case of attribute certificates).

Certificate Revocation Lists (CRLs): CRLs are a common means of checking the revocation status of public key certificates. A CRL is a signed list of certificates that have been revoked before their scheduled expiration date. Each entry in the CRL indicates the serial number of the revoked certificate. The CRL entry may also contain other relevant information, such as the time, and reason for the revocation. CRLs are usually distributed in one of two ways: In the “pull” model, RA distributes the CRLs via a public directory. The clients/queriers download the CRL from these public databases as needed. In the “push” model, the RA directly sends the CRL to the clients, either at regular intervals or at times indicated in prior CRLs [20]. If CRLs are distributed using a pull model, they should be issued at regular intervals even if there are no changes, to prevent new CRLs being maliciously replaced by old CRLs. Alternatively, if the inter-CRL interval is not fixed each CRL needs to include the specific time for the issuance of the next CRL. Since a CRL can get quite long, a RA may instead post a signed Δ -CRL which contains only the new entries consisting of the list of certificates revoked since the last CRL was issued. This requires end-users maintain (and update) secure local images of the CRL. [23] lists some of the other proposed ways to improve the operational efficiency of the CRLs such as Segmented CRLs and CRL distribution points.

Online Certificate Status Protocol (OCSP): this protocol [27] avoids the generation and distribution of long CRLs and can provide more timely revocation information. To validate a certificate in OCSP, a client sends a certificate status request to a VA. The latter sends back a signed response indicating the status (revoked, valid or unknown) of the specified certificate. Note that, in this case, the VA is:

- the CA who issued the certificate in question, or
- a Trusted Responder whose public key is trusted by the client, or
- a CA Designated Responder (Authorized Responder) who is authorized to issue OCSP responses for that CA.

In other words, the VA is an on-line authority trusted by both the client and the CA. A VA can also serve multiple CAs. In practice, in order to reduce VA load, pre-signed responses are often used: a VA signs (once) an oft-requested status a given certificate and uses it in to reply to many requests. This helps performance but sacrifices the timeliness of VA’s replies.

Certificate Revocation Trees (CRTs): this technique was proposed by Kocher [18] as an improvement for OCSP [18]. Since the VA is a global service, it must be sufficiently replicated in order to handle the load of all validation queries. This means the VA’s signature key must be replicated across many servers which is either insecure or expensive. (VA servers typically use tamper-resistance to protect their signing keys). Kocher’s idea is a single highly secure RA which periodically posts a signed CRL-like data structure to many insecure VA servers. Users then query these insecure VA servers. The data structure proposed by Kocher is basically a Merkle Hash Tree (MHT) [24] where the leaves represent currently revoked certificate ranges sorted by serial number (lowest serial number is the left-most leaf and the highest serial number is the right-most leaf). The root of the hash tree is signed by the RA. A client queries to the nearest VA server which produces a short proof that the target certificate is (or is not) on the CRT. If n certificates are revoked, the length of the proof is $O(\log n)$ (In contrast, the proof size in plain OCSP is $O(1)$).

Skip-lists and 2-3 trees: One problem with CRTs is that, each time a certificate is revoked, the whole tree must be recomputed and distributed in its entirety to all VA servers. A data structure allowing for dynamic updates would solve the problem since a secure RA would only need to send small updates to the data structure along with a signature on the new root of the structure. Both 2-3 trees proposed by Naor and Nissim [29] and skip-lists proposed by Goodrich, et al. [12] are natural and efficient for this purpose. Additional data structures were proposed in [1]. When a total of n certificates are already revoked and k new certificates must be revoked during the current time period, the size of the update message to the VA servers is $O(k \log n)$ (as opposed to $O(n)$ with CRT’s). The proof of certificate’s validity is of size $O(\log n)$, same as with a CRT.

5. Privacy-Preserving Revocation Checking with CRTs

Looking at the approaches over-viewed above, it seems that retrofitting privacy into CRLs or Δ -CRLs is not easy. This observation is supported by Kikuchi in [16]. As mentioned in Section 3, the cryptographic accumulator approach is inefficient in terms of both bandwidth and computation. There is, of course, a trivial solution that would entail, for each revocation check, requesting the entire CRL (or Δ -CRL). Although effective – the target of the revocation check remains unknown – this approach is grossly inefficient in terms of band-

width and client storage. We re-consider CRLs in Section 6 below.

Making plain OCSP privacy-preserving is also difficult because the type of a revocation/non-revocation *proof* it employs is basically an on-demand public key signature by the VA. It does not rely, at least as far as clients are concerned, on any specific data structure for representing revoked certificates.

This leaves us with CRTs and related structures, such as 2-3 trees and skip-lists. We start with CRTs (skip-lists are discussed in the Appendix) since they turn out to be quite amenable to supporting privacy and inherently guarantee *completeness* of query replies. (*Completeness* means that a lazy or malicious server can not omit leaf nodes in response to a query without causing verification of the root hash to fail.) Admittedly, our approach is simple (even trivial) and relies on two basic tools:

(1) Range Queries: Because the number of revoked certificates typically constitutes only a small fraction of issued certificates, we suggest, instead of posing revocation queries by a specific target, to query a range or certificates. The size of the range is determined by the combination of two basic parameters: (1) the degree of privacy desired by the querier, and (2) the density/number of revoked certificates. The latter directly influences bandwidth and client storage overhead.

(2) Permuted Ordering: As designed, CRT involves ordering of revoked certificates by (typically) certificate serial numbers. Since most CAs assign consecutive serial numbers to consecutively issued certificates (which makes perfect sense) groups of related certificates, e.g., issued to the same company, would have consecutive serial numbers. Thus, we need to avoid situations where querying for a range of certificates betrays some information about somehow related consecutive blocks of serial numbers contained in the range.

5.1. CRT Details

We now describe the CRT/OCSP scheme in more detail and, in the process, introduce the notation used in the rest of this paper.

Consider a CRT corresponding to a specific CA and/or a block of certificates. Let lo and hi be the lowest- and highest-numbered certificates, respectively and $n = (hi - lo + 1)$ be the total number of certificates. A certificate with the serial number i is denoted C_i . To simplify the description, we assume that the total number of revoked certificates $2 < m \leq n$ (leaf nodes) is a

power of 2.¹ Let L_1, \dots, L_m represent the leaf nodes of the CRT. Each leaf contains the serial number of the corresponding revoked certificate and possibly other information, such as the certificate hash, data/time of, and reason for, revocation. Finally, the notation $N(L_p)$ means the serial number of the certificate referred to by L_p for $1 \leq p \leq m$, and, for all L_p , $C(L_p) = i$ where $lo \leq i \leq hi$. Conversely, $L(C_i)$ is the leaf index of a revoked certificate, i.e., for each revoked C_i , there exists a unique p , such that: $1 \leq p \leq m$ and $L(C_i) = p$.

Consider two revoked certificates C_j and C_k such that $j < k$ and, for each i , $j < i < k$, C_i is **not** revoked. (In other words, all certificates with serial numbers between j and k are valid.) Then, it is easy to see that there exists p such that $C(L_p) = j$ and $C(L_{p+1}) = k$. In most cases (with over 75% probability) any two adjacent leaf nodes are either siblings or cousins.

Another requirement for building a CRT is a cryptographically suitable (efficient, one-way and second pre-image collision-resistant) hash function $H()$ such as SHA-256 [31]. As in any Merkle Hash Tree [24], each non-leaf node is recursively computed bottom-up by hashing the concatenation of its left and right children. Once the root node is computed, its hash, along with additional information such as issuance and expiration date, is signed by the CA. Finally, the signed CRT is distributed to all VAs (responders or distribution centers).

For any node in the tree, we use the term *co-path* to mean a sequence of nodes representing siblings of all direct ancestors of that node.

To check revocation status, a client sends a request containing the certificate serial number, say i , to its closest VA. If C_i is not revoked, the response consists of:

1. Two adjacent leaf nodes L_p, L_{p+1} such that $N(L_p) < i < N(L_{p+1})$
2. Three co-paths: one from L_p and one from L_{p+1} , to their LCA, and a third co-path from the LCA to the root.
3. The signed root node.

If C_i is revoked, the response includes:

1. Two adjacent sibling leaf nodes L_p, L_{p+1} such that either $N(L_p) = i$ or $N(L_{p+1}) = i$.
2. A co-path to the root starting with the sibling of their parent.
3. The signed root node.

¹In practice, a CRT does not need to be perfectly balanced.

In each case, using the data in the response, the client recomputes the root of the CRT and compares it to the signed root. It then (in case it has not done so yet for some previous query) verifies the CA's signature on the root. This forms a proof of the target certificate's status.

The CRT/OCSP scheme is computation-efficient since it obviates the need to sign each reply. Moreover, it removes most of the trust from VAs which are no longer required to maintain on-line keys, as in plain OCSP. Also, the bandwidth overhead is modest, logarithmic in terms of m – the number of revoked certificates. However, bandwidth overhead is higher than in plain OCSP which has constant-sized query replies. Figure 1 illustrates an example CRT with a query to

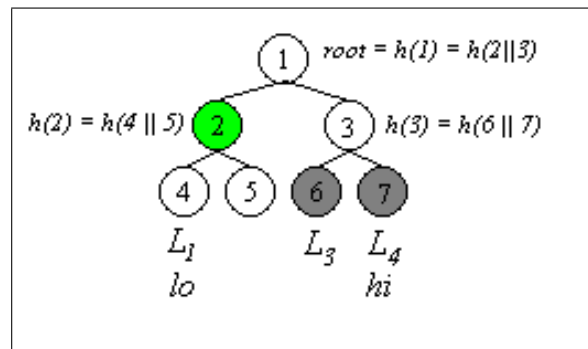


Figure 1. An example CRT query for node L_3 .

node L_3 . The co-path returned to the client is denoted by the green nodes.

5.2. Range Queries

The basic idea behind range queries is very simple. Instead of querying by a specific certificate serial number i , the client queries a range of serial numbers (j, k) with $j \leq i \leq k$. This allows us to effectively hide the certificate of interest. The only information divulged to the VA (third party) is that the target certificate lies in the interval $[j, k]$ which translates into the probability of correctly guessing i : $P_i = \frac{1}{k-j+1}$. Each number in the range is equally likely to be the serial number of interest and the third party has no means, other than guessing, of determining the target certificate.

Furthermore, the third party has no way of telling whether the target is a revoked or a non-revoked certificate. Assuming uniform distribution of revoked certificate serial numbers over the entire serial number range, $\frac{m}{n}$ is the fraction of revoked certificates. The very same fraction of certificates would then be revoked in any (j, k) range and hence $(k - j + 1) * \frac{m}{n}$ adjacent leaf nodes would be contained in the query reply.

We stress that using range queries in conjunction with CRTs does not involve any modifications to the

basic CRT data structure.

5.3. Range Size

As is typical with simple solutions, the challenge is in the details. Clearly, there is no perfect privacy attainable with range queries. The highest possible privacy is $\frac{1}{n}$ which corresponds to querying the full certificate serial number range, i.e., $[j = lo, k = hi]$, and entails receiving the entire set of CRT leaf nodes.² The lowest privacy level corresponds to querying – as currently done – by a specific serial number, i.e., setting $j = k = i$.

The optimal query range is determined by the source of the query, i.e., the client. Several factors must be taken into account: (1) desired level of privacy, e.g., the probability of guessing equal to 0.001 which, equivalently, the desired level of privacy equal to $k - j + 1 = 1000$, (2) additional bandwidth and storage overhead stemming from a set of adjacent leaf nodes in the reply. It is important to note that additional bandwidth overhead does not depend on the height of the CRT. This is because, in the plain CRT scheme, any query reply always includes a co-path. The same holds for our modification. The only “new” overhead is incurred due to the number of adjacent leaf nodes returned. As described in Section 5.1, at most two leaf nodes are returned if a certificate-specific query ($j = k = i$) is posed. In contrast, a range query of size r entails returning $\lceil \frac{r * m}{n} \rceil$ contiguous leaf nodes.

Once the range size (r) is decided, the client proceeds to set the actual range boundaries: j and k . To do so, it first generates a b -bit random number X where $b = \log(r)$ or the bit-length of r . X determines the position, within the range, of the actual target certificate serial number. This step is necessary to randomize/vary the placement of the target. Next, the boundaries are set as: $j = i - X$ and $k = j + r - 1$. (Special care must be taken if $i < X$ or $i - X < lo$. More on this below.)

We observe that, if a client poses repeated queries against the same target certificate, varying the query range and its boundaries is not advisable, for obvious reasons. In this case, narrowing the overlap of all queries’ ranges gradually erodes privacy and might eventually yield a single target certificate. We now discuss one possible solution to eliminate such inference attacks. To avoid this situation, our prototype implementation – described in Appendix A – keeps a cache of previously queried certificates along with corresponding ranges.

A related privacy-enhancing measure is to reuse previously queried ranges. If a certificate of interest is contained within a previously queried range, then re-using an old query range that contains the (new) certificate of

interest leaks no additional information. Instead of generating a random number (offset) to determine the position of the target certificate within the range, the client generates a b -bit number using a *keyed hash function* h' which takes as input the target certificate serial number i and a secret key sk chosen by the client. $h'(i, sk) = X$ where $|X| = b$. Now, the client sets the range boundaries as described earlier, i.e., $j = i - X$ and $k = j + r - 1$. The client then queries the VA with the range $[j, k]$. The use of the secret key sk to compute range boundaries ensures that the same range is consistently used for repeated queries against a specific certificate.

5.4. Range Size Analysis

The intuition behind our claim that a range query provides privacy is fairly straightforward. It is impossible for the distribution center, and indeed anyone intercepting traffic, to determine with any significant advantage the targeted certificate in the returned range. In other words, we claim that:

Given a client query range (j, k) and corresponding results from the server, no adversary can distinguish with probability negligibly over 50% among two certificates $a, b \in (j, k)$ where a is the certificate of interest and b is not.

The only information that the adversary learns about the target of the query is the range. Since we require the range to be randomly determined (as long as the certificate of interest is within the range) and the client performing repeated queries against the same certificate uses the same range (j, k) , the attacker gains no additional information about the actual certificate of interest. Each certificate in the range is equally likely to be the certificate of interest with probability $\frac{1}{k-j}$.

We note that it does not matter if the client is interested in a certificate which is actually revoked or is still valid. The adversary does not know if the client is asking about a revoked or a non-revoked certificate. If the range size is $X = k - j$ and revocation density is r ($r \ll 1$), the expected number of revoked certificates in that range is $(X * r)$. However, even in the unlikely event that there are no revoked certificates in the entire range, the adversary still gains no knowledge about which certificate ($1 \text{ out of } X$) is of interest to the client. The only information that the adversary gains is that the client’s target certificate is **NOT** revoked. Since there are X such certificates, the adversary can only guess (with probability $1/X$) the query target.

However, we concede that, if revocation status of a particular certificate is being queried by *many* clients – and each client picks its own random range – the target certificate will be contained within the intersection of all such queries’ ranges.

²This is equivalent to obtaining an entire CRL.

5.5. Revocation Density

In order to achieve a tailored trade off between privacy and (mostly bandwidth) overhead, the client has to be aware of the revocation density, i.e., the ratio of revoked-to-unrevoked certificates, denoted by $\frac{m}{n}$. We suggest two simple ways of obtaining this value.

The simpler method requires no modifications whatsoever to the CRT data structure. A client merely poses a dummy revocation query with a randomly generated certificate serial number (no range query). The purpose is to elicit a reply in the form of the proof containing a CRT co-path. Verifying the reply securely convinces the client of the CRT's height. Given the height, the number of leaf nodes is easily computed, assuming again that the tree is balanced. The revocation density immediately follows. (Note that the dummy query is needed only once per CRT, supposing that the CRT update interval is globally known.)

If dummy queries are undesirable or keeping the CRT balanced is not practical, a minor modification solves the problem. Recall that the root of the CRT is always signed by the issuing CA or its trusted off-line agent. One obvious modification is to include the number of leaf nodes (or the actual ratio) in the computation of the root node's signature. A client initially obtains the signed root and obtains the associated tree revocation density as a consequence of successfully verifying the root signature.

5.6. Query Response

Upon receipt of a range query (j, k) , the VA first determines the contiguous sequence of leaf nodes corresponding to all revoked certificates within the range. It then adds to this sequence two sentinel leaf nodes: one just beyond k and one immediately preceding j (unless either j or k correspond to the leftmost or rightmost leaves in the CRT, respectively). This is needed to prove completeness of the query reply. Completeness in this context refers to expectation that a client will receive *all* nodes within the range, i.e., a server can not omit leaf nodes without causing root hash verification to fail.

All of these leaf nodes have the lowest common ancestor denoted by LCA. The reply must include the sequence of leaf nodes and a co-path from the LCA up to the root. In addition, the reply needs to include two partial co-paths to enable the client to recompute the LCA. This differs from the plain CRT scheme where a single co-path to the root is sufficient. Of course, the additional (over plain CRT) overhead is mainly due to returning *multiple* leaf nodes as part of the verification object. As long as the revocation density – which is used to determine the query range – is uniform, on the

average $\lceil r * (\frac{m}{n}) \rceil$ leaf nodes are returned. Also, of the two co-paths leading up to the LCA, one represents additional overhead imposed by our method.

The respective bandwidth costs (ignoring constants) of plain CRT and the range query extension can be compared as follows:

- Plain CRT: $\log(m)$ – two leaf nodes and a co-path from their parent (or grandparent) to the root.
- Range Query: $\log(m) + \log(\frac{r*m}{n}) + \lceil \frac{r*m}{n} \rceil$ – a set of $\lceil \frac{r*m}{n} \rceil$ contiguous leaf nodes, a co-path from their LCA to the root and two co-paths from sentinel leafs to the LCA.

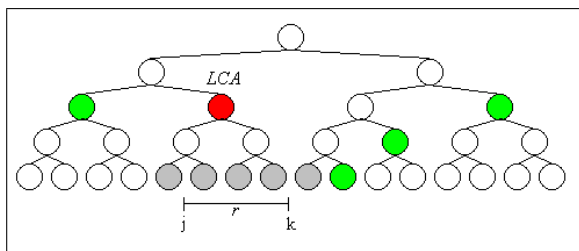


Figure 2. LCA and co-path nodes for a given (j, k) range.

Figure 2 illustrates an example with two co-paths necessary to compute the root hash. The first co-path includes all nodes on the left side of the subtree and the second includes all nodes on the right. These nodes, along with the leaf nodes in the (j, k) range, are used to compute the root of the CRT. The figure also illustrates how computing the LCA for nodes returned in the (j, k) range results in shorter co-paths, i.e., by computing the LCA, the co-path can begin from the sibling node of the LCA instead of the leaf nodes.

5.7. Forcing Uniform Distribution

In a worst-case scenario, all certificates in the desired range are revoked and corresponding r leaf nodes must be returned to the client. The simplest solution to this is to force all revoked certificate serial numbers to be uniformly distributed over the entire serial number range. However, this is unrealistic, since, in practice, certificate issuers assign serial numbers to certificates consecutively over well-defined subranges.³ Each subrange can be used to indicate a different product or class of products, e.g., VeriSign supports the following classes: Standard, Commerce and Premium [6].

³This is true even in light of certain new attacks [22]. Such attacks allow the adversary to construct a pair of valid X.509 certificates when the template for the certificate is known or easily guessed, i.e., with high probability two sequential certificates will have near identical header templates.

Requiring uniform non-sequential certificate distribution would create a maintenance nightmare for both issuers and certificate-holders. Furthermore, gathering and analysis of statistical data would become problematic.

We propose a simple extension to the range query technique that addresses the problem while guaranteeing uniformity among the CRT leaf nodes. Instead of sorting according to serial numbers, we sort leaf nodes along *permuted* serial numbers. One obvious choice of suitable permutation that ensures uniformity is a block cipher, e.g., DES, with a known and fixed key. Note that the brute-force resistance of the block cipher is not important here. The only issue of concern is the block cipher’s quality – for a fixed key – as a pseudo-random permutation (PRP). Ideally, the space of all possible serial numbers would match the set of all possible block cipher outputs. For example, DES-ECB mode outputs 64-bit blocks which matches the size of certificate serial numbers for many X.509v3 CAs. However, the underlying permutation can be resolved with any good block cipher, such as Blowfish or AES. We further observe that a cryptographic hash function is not a good choice for the kind of a PRP we require. Unlike a PRP, a hash function “reduces” its input and collisions are expected, however difficult they might be to compute. Whereas, a PRP resolved with a block cipher such as DES-ECB with a fixed key, guarantees no collisions.

The primary advantage of this extension is that certificate issuers can continue issuing sequentially-numbered certificates over well-defined subranges. As long as an appropriate PRP is used, we can assure uniform distribution of the CRT leaf nodes. An unfortunate drawback of this technique is that revoking a whole block of consecutive certificate serial numbers becomes inefficient. This is because permuted serial numbers are scattered throughout the total range of serial numbers, which complicates the corresponding CRT.

6. Privacy-Preserving CRL-based Revocation Checking

We now turn our attention to the CRLs – the most common data structure used for certificate revocation. We start by noting that a CRL trivially meets the privacy requirement, since in a typical usage scenario, a verifier simply downloads and stores the entire CRL. This allows it to locally perform any number of revocation checks against any certificate issued by a particular CA. However, downloading and caching a large CRL in its entirety is neither bandwidth nor storage efficient for the client. Although delta-CRLs tend to be smaller and more bandwidth-efficient, they still impose high storage overhead on the client. In addition, a client is required

to periodically download the base-CRL as well as all subsequent delta-CRLs in order to keep the CRL cache recent and accurate. In this section, we propose a new CRL model and outline a mechanism which overcomes the aforementioned drawbacks, while allowing efficient and private revocation checking.

In our model, a trusted RA periodically generates CRLs and publishes them at untrusted online VA-s. A client interested in validating a certificate queries a VA to obtain authentic revocation information corresponding to that certificate. Although this model bears similarities to the OCSP method, unlike OCSP, the VA-s in our model can be third party responders who are not trusted by either clients or RA-s.

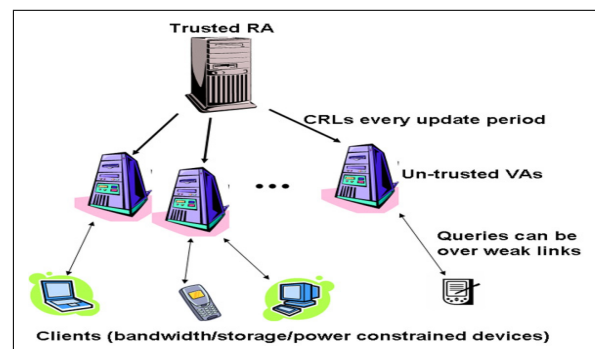


Figure 3. System Overview

Since VA-s are untrusted, it is essential to ensure the integrity and authenticity of the revocation information of a certificate with respect to the RA responsible for that certificate. To achieve this, the RA individually signs each CRL entry. In other words, information regarding each revoked certificate is separately signed by the RA and placed onto the CRL. This differs from the usual CRL structure where all revoked certificates are included in the CRL which is then collectively signed once by the RA. Our modification clearly introduces some additional burden for the RA who is now required to sign each entry. However, we claim that this modification makes certificate revocation checking very efficient for clients; at the same time, it is not prohibitively expensive for a typical RA which is a computationally-powerful machine. (We analyze actual overheads in section 7.5.) This relatively minor modification obviates the need for the client to download the entire CRL.

6.1. Normal CRL Revocation Checking

When a VA receives a revocation status query from a client, it checks to see if there is a CRL entry corresponding to that certificate. If it exists, the VA sends back a **Revoked** reply along with the CRL entry signed

by the RA, as the proof. However, if the certificate in question is not revoked, then simply sending a **Not Revoked** reply does not provide integrity and authenticity guarantees with respect to the RA⁴. To achieve authenticity of **Not Revoked** replies, we suggest chaining of CRL entries, as described below.

Signature Chaining: To provide authenticity of query replies, the RA securely links the signatures of the revoked certificates to form a so-called *signature chain*. To construct a signature chain, the RA generates the individual signature for each revoked certificate as follows:

$$\text{Sign}(i) = h(h(i)||h(\text{IPC}(i)))_{SK}$$

where i is the (permuted) serial number of a revoked certificate, $h()$ is a suitable cryptographic hash function, $||$ denotes concatenation, IPC denotes the immediate predecessor certificate and SK is the private signing key of the RA.

The immediate predecessor is a revoked certificate with the highest serial number which is less than the serial number of the current revoked certificate. In other words, the RA sorts all revoked certificates in ascending order by serial number and computes the signature of each revoked certificate by including the hash of the immediate predecessor, thereby explicitly chaining (linking) all signatures.

With signatures chained in the above fashion, when a client queries the status of an un-revoked certificate, the VA composes an **Not Revoked** reply by returning the two boundary certificates $CERT_a$ and $CERT_b$ that subsume the non-existent serial number along with the signature of $CERT_b$.

6.2. Privacy-Preserving Checking

Our CRL modification allows a client to query the VA for revocation status of a certificate and obtain a concise and authentic proof of either revocation or validity. However, this introduces privacy concerns since clients now query the VA by a specific certificate serial number, thus revealing the identity of the target entity. To address this problem we employ the same technique as in CRT case. Instead of querying by a specific certificate serial number i , we propose querying by a randomly selected range of serial numbers (j, k) with $j \leq i \leq k$. This effectively hides the certificate of interest. The only information divulged to the VA is that the target certificate lies in the interval $[j, k]$. This

⁴Recall that VAs are untrusted in our model, and a malicious or lazy VA might not properly execute the query and send incorrect reply to the client.

translates into the probability of correctly guessing i as: $P_i = \frac{1}{k-j+1}$. Each number in the range is equally likely to be the serial number of interest and the VA has no means, other than guessing, of determining the target certificate. Furthermore, the VA has no way of telling whether the actual query target is a revoked or a valid certificate.

Let n be the total number of certificates and m be the number of revoked certificates. Then, assuming uniform distribution of revoked certificate serial numbers over the entire serial number range, m/n is the fraction of revoked certificates and the very same fraction would be revoked within any $[j, k]$ range.

Clearly, perfect privacy is not attainable with range queries. The highest possible privacy is $1/n$ which corresponds to querying the full certificate serial number range.⁵ The lowest privacy level corresponds to querying by a specific serial number, i.e., setting $j = k = i$. The optimal query range is determined by the source of the query, i.e., the client. Several factors must be taken into account: (1) desired level of privacy e.g., if $k - j + 1 = 1000$, the probability of correctly guessing i is $P_i = 0.001$, (2) additional bandwidth and storage overhead stemming from returning a set of CRL entries as a reply.

Once the range size (r) is determined, the client proceeds to set the actual range boundaries: j and k . To do so, it first generates a b -bit random number X where $b = \log(r)$ or the bit-length of r . X determines the position, within the range, of the actual target certificate serial number. This step is necessary to randomize/vary the placement of the target. Next, the boundaries are set as: $j = i - X$ and $k = j + r - 1$. The client poses a query to the VA asking for the revocation status of all certificates within $[j, k]$. Since each entry corresponding to a revoked certificate in the $[j, k]$ range is linked only to its IPC (as noted above), it seems that the VA would need to send all signatures to the verifier. This is clearly inefficient for the verifier. However, we can modify the signature generation process as follows:

- Sort all revoked certificates in ascending order of permuted serial numbers to obtain $\{CERT_0, CERT_1, CERT_2, \dots, CERT_m, CERT_{m+1}\}$. The two “dummy” boundary values: $CERT_0$ and $CERT_{m+1}$ represent $-\infty$ and $+\infty$, respectively, i.e., they denote values outside the allowed range.
- Now, compute the signature of each revoked certificate by signing the **running hash** of all certificates in the chain from the first entry to the current certificate:

$$\text{Sign}(CERT_i) = h(CERT_i || h(CERT_{i-1}) || \dots || h(-\infty))_{SK}$$

⁵This is equivalent to obtaining an entire CRL.

where $h()$ is a suitable cryptographic hash function, $||$ represents concatenation and SK is the private (signing) key of the RA.

- The resulting CRL is stored at the VA-s as before.

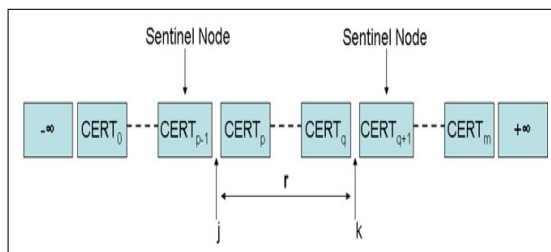


Figure 4. Revocation Query Reply

- To assert revocation status of all the certificates in the range $[j, k]$, a VA releases the revoked certificates $[CERT_p, \dots, CERT_q]$ in the actual range $[j, k]$, two sentinel nodes $CERT_{p-1}$ and $CERT_{q+1}$ just outside the range to prove completeness of the reply, running hash for $CERT_{p-2}$, and a **single** signature – $Sign(CERT_{q+1})$. Since all signatures are computed on running hashes, it can be easily seen that $Sign(CERT_{q+1})$ provides a proof of authenticity and completeness for the entire range.

7. Practical Considerations

We now discuss some practical implications of the proposed modifications.

7.1. CRL Generation and Size

With the technique described above, the RA signs revoked certificates separately. This increases the overall size of the CRL as well as the computation load on the RA. However, we claim that this overhead is acceptable, for the following reasons:

Although it might seem that computational overhead of signing each revoked certificate might be significant for the RA, we show that this is not the case in practice. Experiments show that it takes 6.82ms to generate one RSA signature on a “weakling” P3-977MHZ Linux PC. Assuming that the update interval for the CRL is one week and the CRL contains 100,000 certificates, this translates to 11.3 minutes of compute time for every CRL update period. Clearly, the same task would take much less (one or two orders of magnitude) time on more powerful platforms readily available today. For example, the Sun Fire T2000 server can compute 12,850 1024-bit RSA signatures and 18,720

1024-bit DSA signatures in one second with 1 UltraSPARC T1 processor and 32 GB memory running Solaris 10[33]. Going back to our example scenario, if the RA were to be deployed on a T2000 server, it could recompute 100,000 signatures in just under 10 seconds.

The proposed technique requires storing a separate signature per CRL entry. This increases the size of the CRL and, consequently, increases the cost of RA-VA communication. However, it is reasonable to assume that this communication is conducted over fast communication links, in contrast to VA-Client communication which can take place over low-bandwidth channels. Furthermore, RA-VA communication occurs relatively infrequently (once per update period), whereas the frequency of clients-VA communication is significantly higher. We present sample metrics for communication costs in the next section.

7.2. Freshness Considerations

In order to provide freshness and to prevent malicious VA-s from replaying old CRL-s, the RA must re-sign revoked certificates every update period, even if there are no changes to the CRL. We can reduce the time to re-sign the revoked signatures in the event of no changes to the CRL by using the Condensed RSA signature scheme proposed in [28].

Condensed-RSA Signature Scheme: Given t different messages $\{m_1, \dots, m_t\}$ and their corresponding signatures $\{\sigma_1, \dots, \sigma_t\}$ generated by the same signer, a Condensed-RSA signature is computed as the product of all t individual signatures:

$$\sigma_{1,t} = \prod_{i=1}^t \sigma_i \pmod{n} \quad (1)$$

The resulting aggregated (or condensed) signature $\sigma_{1,t}$ is of the same size as a single standard RSA signature. Verifying an aggregated signature requires the verifier to multiply the hashes of all t messages and checking that:

$$(\sigma_{1,t})^e \equiv \prod_{i=1}^t h(m_i) \pmod{n} \quad (2)$$

Condensed-RSA for Re-signing: It is possible to use condensed-RSA for re-signing if the RA signs the revoked certificate data and the time-stamp separately and then aggregates the two by multiplying the resultant signatures as explained above. If there are no changes to the CRL, the RA can re-sign all the revoked certificates by: (1) re-using the signatures for the revoked certificate data; (2) creating a single new signature on the new time-stamp; and (3) aggregating the signature of

the new time-stamp with the signatures of all the revoked certificates to re-sign all the revoked certificates.

If we assume, once again, that the CRL contains 100,000 entries. As mentioned above, it would take 11.3 minutes to re-sign all these certificates with the new timestamp on a P3-977MhZ Linux machine. However, if we use condensed-RSA, re-signing of the same 100,000 entries with the new time-stamp, using our method can be done in just 4.4 seconds on the same hardware. However, this optimization is less useful if there are changes to the CRL. Because signatures are chained explicitly, any change to the CRL (insertion or deletion of an entry from the chain) causes all the signatures from that point until the end of the chain to be re-computed.

7.3. Forcing Uniform Distribution

We use the same PRP-based technique as in the case of CRTs, as described in Section 5.7.

7.4. Query Range

The mechanism for generating random range boundaries is the same as in the case of CRTs, as described in Section 5.3.

7.5. Overhead Assessment

We now briefly analyze the overhead factors introduced by the above technique.

Client Overhead: With traditional CRLs, clients are required to store entire lists containing all revoked certificates locally. As such lists grow, this can result in significant storage overhead. On the other hand, with our technique, clients do not incur any storage costs. A client queries the on-line VA for the revocation status of a certificate and does not need to store/cache anything locally.

For traditional CRL, the client is required to periodically contact a CRL distribution point (or directory server) to obtain the most recent CRL. If Δ -CRLs are used, the client needs to periodically download updates in order to keep her copy of the CRL recent and complete. Thus, communication overhead using traditional CRLs and delta-CRLs depends upon the frequency of CRL updates. On the other hand, with our proposed technique, a client incurs one round of communication with the VA for each revocation status check of a certificate.

RA Overhead: In our method, the RA is required to separately sign each entry in the list, whereas, in a tra-

ditional CRL, the entire CRL is signed once in its entirety, for each update. This increases both the size of the CRL as well as the computation costs for the RA. However, as discussed in the previous section, we assume that the frequency of querying is much greater than the frequency of CRL updates; thus, we focus our scheme on minimizing costs for the interaction between the VA and its clients.

Communication Costs: We now compare communication costs of the traditional CRL with our modified version. Table 1 summarizes the notation and parameter values assumed for the evaluation. **Update cost** measures RA-to-VA communication and **query cost** measures VA-to-Client communication.

n	# of Certificates $n = 2 * 10^6$
p	Fraction of the certificates revoked $p = 0.1$
t	update period $t = \text{weekly}$
c	# of clients $c = 10^5$
q'	# of queries posed by each client $q' = 100$
q	Total # of queries in an update period $q = c * q' = 10^7$
r	Size of the range query
l_{sig}	Length of a signature $l_{sig} = 1024$ bits
l_{entry}	Length of a CRL entry $l_{entry} = 160$ bits
l_{hash}	Length of a hash digest $l_{hash} = 160$ bits

Table 1. Notation

Traditional CRLs: The CRL update cost is $n * p * (l_{entry} + l_{sig})$ for each update period since the RA sends the entire CRL to the VAs. We are assuming that there are c clients and each client poses q' certificate validation checks on the cached CRL in a update period. The CRL weekly query cost is: $c * (p * n * l_{entry} + l_{sig})$ since for every query the VA sends the whole CRL to the client.

Modified CRLs: To update the CRL, the RA sends $n * p * (l_{entry} + l_{sig})$ bits to the VA. To answer a user's query, the VA returns $l_{entry} + l_{sig}$ for a **Revoked** reply and $2l_{entry} + l_{sig}$ for a **Not Revoked** reply if queried by a specific serial number (i.e., for non-privacy preserving querying). Therefore, the VA sends up to $q * (2l_{entry} + l_{sig})$ bits each update period to answer q queries. If we employ range queries to enable privacy-preserving querying, then the communication cost depends on the number of revoked certificates in the range. In general, if we assume uniform distribution of revoked certificates, then the communication cost for user queries for a range of certificates is given by $q * ((r * p) + 2) * l_{entry} + l_{sig}$ bits for each update

period. Note that $(r * p) + 2$ denotes the total number of revoked certificates in the specified range plus two sentinel nodes.

The following table shows the estimated communication costs (in bits) according to traditional as well as modified CRL schemes. As shown in the table, the modified CRL query costs are orders of magnitude lower than traditional CRL costs. Although the cost of updating a CRL is higher in our scheme, the advantage of our scheme is that it allows the clients to query and obtain portions of the CRL securely thus making our scheme flexible, efficient (for the clients) and privacy preserving.

	Traditional CRL	Modified CRL no privacy	Modified CRL r=100
Update Cost (RA-to-VA)	$3.2 * 10^7$	$2.36 * 10^8$	$2.36 * 10^8$
Query Cost (VA-to-Client)	$3.2 * 10^{12}$	$1.18 * 10^{10}$	$2.94 * 10^{10}$

Table 2. Communication Cost comparison

8. Comparison

In this section we compare the two proposed methods and clarify their respective advantages and disadvantages. Using the notation from Table 1, we compare the following factors:

Achievable Privacy: Both methods allow range queries and thus achieve the same level of privacy. Specifically, the probability of guessing the target certificate i is: $P_i = \frac{1}{k-j+1}$. The highest possible privacy is $\frac{1}{n}$ (obtaining entire CRL or CRT).

RA Overhead: In the CRT-based method, the RA has to compute a hash tree and sign its root. The total cost for n certificates is: $2n - 1$ hash operations and one signature. In the modified CRL method, the RA separately signs each entry in the list. The total cost for n certificates is n signatures.

Update Cost: To update a CRT, the RA sends $(n * p * l_{entry}) + l_{sig}$ bits. Whereas, to update a modified CRL, the RA sends $n * p * (l_{entry} + l_{sig})$ bits. For a set of revoked certificates, the modified CRL method results in $l_{sig} * ((n * p) - 1)$ more bits than its CRT-based counterpart.

VA Overhead: Since VAs only respond to client queries, there is no additional computation overhead associated with either method. The overhead incurred at the VA amounts to space to store revoked certificates and signature information. Storage overhead is equivalent to the update cost.

Query Cost: The exact communication cost depends on the number of revoked certificates in the query range. Assuming uniform distribution of revoked certificates, the communication cost of the CRT method is:

$$q * ((\log(n * p) + \log(r * p)) * l_{hash} + ((r * p) + 2) * l_{entry} + l_{sig})$$

The same cost in the modified CRL method is: $q * (((r * p) + 2) * l_{entry} + l_{sig})$. Note that $(r * p) + 2$ denotes the total number of revoked certificates in the specified range as well as two sentinel nodes. For a given query, the CRT method sends $(\log(n * p) + \log(r * p)) * l_{hash}$ more bits (associated with co-path information) than the modified CRL method.

Client Overhead: The advantage of both proposed techniques is that clients are not required to store CRLs. Large CRLs can result in significant storage and bandwidth overhead and are burdensome to clients. Allowing clients to query an on-line VA saves local storage space on clients.

The cost of verifying a VA response is similar in both methods. In the CRT method, a client computes the root of the hash tree and verifies its signature. In the modified CRL method, a client computes a hash chain and verifies a signature. For a given query, a client must compute $\log(n * p) + \log(r * p)$ hashes in the former and $(r * p) + 2$ hashes in the latter. With reasonably efficient hash functions (such as SHA-2), the difference between the two methods is minimal.

9. Real-World Scenarios

Public Key Infrastructures (PKIs) are already well-established in commercial, educational and government venues. For example, VeriSign, one of the leading certificate issuers has more than 450,000 public key certificates in many different countries throughout the world [35]. The majority of e-commerce sites utilize VeriSign certificates. Additionally, the United States Army has instituted a program that issues public keys (contained on a personal smartcard) to all military personnel, selected reservists, civilian employees, and on-site contractors in the Department of the Army [21]. This initiative is quite remarkable because of its huge scale.

The Department of the Army is expecting to issue a total of around 1.4 million smartcards. Researchers have already started pointing out potential problems with the planned implementation of the PKI infrastructure [14, 4].

Both VeriSign and the Department of the Army use CRLs as the primary means of distributing information about invalid certificates. VeriSign hosts a public website with all CRLs [7]. Each CRLs issued includes the certificate serial numbers along with a hash of the certificate. The CRLs combined together represent over 115,000 revoked certificates and take up 3.6 MBytes of space. The situation is worse for the Department of the Army. In a study by the National Institute of Standards and Technology (NIST), Berkovits, et al. [4] predict certificate revocation frequency as high as 10%. This is based on the relatively fast re-issue rate (every three years) and the high fluidity of the user base. Supporting CRLs with upwards of 140,000 certificates translates into a bandwidth nightmare requiring each of the 1.4 million smart card owners to periodically download the CRLs.

With such high bandwidth requirements for traditional CRLs, alternate solutions providing low bandwidth costs need to be explored. Our approaches, as described above, offer client-tunable bandwidth/privacy trade-off.

10. Future Directions and Conclusions

An outstanding issue is assessing loss of privacy in the presence of repeated queries. If we assume that multiple clients, at about the same time, are all interested in a particular target certificate (e.g., because of a breaking news article) and the adversary (third party or VA) is aware of the potential target, correlating multiple range queries does not seem difficult since all the range queries in question would have at least one certificate in common. As part of our future work, we want to study this problem in greater detail to make such inferences more difficult.

Additionally, the usability factor remains largely unexplored. Many interesting security- and privacy-enhancing techniques have been proposed and lauded by the research community only to quietly fade into obscurity due to usability issues. As mentioned earlier in the paper, revocation checking is unfortunately all but ignored by the majority of Internet users. For this reason, finding simple and unobtrusive ways of making average users aware of both the need for revocation checking and the need to protect their privacy (as part of revocation checking) is a major challenge.

In conclusion, we described two simple (yet novel) approaches for privacy-preserving revocation checking

using CRTs and CRLs, respectively. We proposed minor modifications to both data structures. These modifications enable efficient (for the clients) and privacy-preserving revocation checking. We provided a mechanism for verifiers to query untrusted online entities regarding the revocation status of a certificate without having to retrieve the entire CRT or CRL. Each client, depending on the desired level of privacy, can determine a revocation query range that best suits its needs. This results in a basic trade-off between privacy and bandwidth overhead. Furthermore, experience from real-world environments (based on revocation statistics from government, commercial, and military sources) suggests that the proposed solutions would work well since revoked certificates represent a small fraction of the total numbers of issued certificates.

Acknowledgments

We thank EUROPKI'07 and PET'06 anonymous reviewers for their help in improving earlier version of this paper's components. We are also grateful to Einar Mykletun and Marina Blanton for their helpful comments.

References

- [1] W. Aiello, S. Lodha, and R. Ostrovsky. Fast digital identity revocation. In Hugo Krawczyk, editor, *Proceedings of Crypto'98*, number 1462 in LNCS. IACR, Springer Verlag, 1998.
- [2] The OpenPGP Alliance. Openpgp: Open pretty good privacy, <http://www.openpgp.org/>.
- [3] N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Proceedings of Eurocrypt'97*, pages 480–494, 1997.
- [4] S. Berkovits, S. Chokhani, J. Furlong, J. Geiter, and J. Guild. Public key infrastructure study: Final report, April 1994. Produced by the MITRE Corporation for NIST.
- [5] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylog communication. In *Proceedings of Eurocrypt'99*, LNCS. IACR, Springer Verlag, 1999.
- [6] Verisign Corporation. Compare all ssl certificates from verisign, inc. <http://www.verisign.com/products-services/security-services/ssl/buy-ssl-certificates/compare/index.html>.
- [7] Verisign Corporation. Public online crl repository. <http://crl.verisign.com/>.
- [8] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of 2003 IEEE Symposium on Security and Privacy*, May 2003.
- [9] T. Dierks and E. Rescorla. The transport layer secu-

- urity (tls) protocol, version 1.1. Internet Request for Comments: RFC 4346, April 2006. Network Working Group.
- [10] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of 13th USENIX Security Symposium*, August 2004.
- [11] Inc. Free Software Foundation. Gnu privacy guard, <http://www.gnupg.org/>.
- [12] M. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proceedings of DARPA DIS-CEX II*, 2001.
- [13] OpenSSL User Group. The openssl project web page, <http://www.openssl.org>.
- [14] J. Hackerson. Rethinking department of defense public key infrastructure. In *Proceedings of 23rd National Information Systems Security Conference*, October 2000.
- [15] S. Kent and K. Seo. Security architecture for the internet protocol. Internet Request for Comments: RFC 4301, December 2005. Network Working Group.
- [16] H. Kikuchi. Privacy-preserving revocation check in pki. In *2nd US-Japan Workshop on Critical Information Infrastructure Protection*, pages 480–494, July 2005.
- [17] J. Klensin. Role of the domain name system (dns). Internet Request for Comments: RFC 3467, February 2003. Network Working Group.
- [18] P. Kocher. On certificate revocation and validation. In *Proceedings of Financial Cryptography 1998*, pages 172–177, 1998.
- [19] E. Kushilevitz and R. Ostrovsky. Computationally private information retrieval with polylog communication. In *Proceedings of IEEE Symposium on Foundation of Computer Science*, pages 364–373, 1997.
- [20] RSA Laboratories. Crypto faq: Chapter 4.1.3.16. what are certificate revocation lists (crls)?, <http://www.rsa.com/rsalabs/node.asp?id=2283>.
- [21] US Army Research Laboratory. Using the cac with pki - faqs. http://www.usaar1.army.mil/CBT/EndUser/chapter_06b/chapter06b.html.
- [22] A. Lenstra, X. Wang, and B. de Weger. Colliding x.509 certificates. Cryptology ePrint Archive, Report 2005/067, 2005. <http://eprint.iacr.org/>.
- [23] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1997. ISBN 0-8493-8523-7.
- [24] R. Merkle. *Secrecy, Authentication, and Public-Key Systems*. PhD thesis, Stanford University, 1979. PH.D Dissertation, Department of Electrical Engineering.
- [25] S. Micali. Certificate revocation system. United States Patent 5666416, September 1997.
- [26] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol — Version 2. IETF Internet Draft, July 2003.
- [27] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. Internet public key infrastructure online certificate status protocol - OCSP. Internet Request for Comments: RFC 2560, 1999. Network Working Group.
- [28] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *Symposium on Network and Distributed Systems Security (NDSS'04)*, February 2004.
- [29] M. Naor and K. Nissim. Certificate revocation and certificate update. *IEEE Journal on Selected Areas in Communications (JSAC)*, 18(4):561–570, April 2000.
- [30] M. Narasimha and G. Tsudik. Privacy-preserving revocation checking with modified crls. In *Proceedings of EuroPKI'07*, June 2007.
- [31] National Institute of Standards and Technology. Federal information processing standards (fips), publication 180-2, secure hash standard (shs), February 2004.
- [32] John Solis and Gene Tsudik. Simple and flexible revocation checking with privacy. In *Workshop on Privacy-Enhanced Technologies (PET'06)*, July 2006.
- [33] Sun Microsystems. Sun Fire T1000 and T2000 Servers Benchmarks. <http://www.sun.com/servers/coolthreads/t1000/benchmarks.jsp>.
- [34] International Telecommunication Union. Recommendation x.509 (1997 e): Information technology open systems interconnection - the directory: Authentication framework, 6-1997, 1997. Also published as ISO/IEC International Standard 9594-8.
- [35] Verisign Corporation. Corporate Overview: Fact Sheet from VeriSign, Inc. <http://www.verisign.com/verisign-inc/corporate-overview/fact-sheet/index.html>.

Appendix A: Prototype Implementation

The range query approach described above has been implemented as a stand-alone proof-of-concept prototype available for both Linux and Win32 platforms. The tools and the source code are available for download at <http://sconce.ics.uci.edu/ppr>. The toy prototype consists of the client and server components and utilizes the popular OpenSSL crypto library [13]. There is also a separate CA component which issues certificates and CRTs.

The prototype components are configured with the following parameters:

- Pseudo-Random Permutation Function: $PRP(\cdot)$
- CA Public/Private Key-Pair: (PK, SK)
- The CRT Root Hash and its RSA signature

In this implementation, the permutation function can be one of the following block ciphers supported by OpenSSL: Blowfish, DES, RC4.

The server component takes as input the path to an ASCII configuration file, or loads from a default file if one is not supplied. Currently, there is no interactive way of configuring the server. The configuration file allows for selecting a (PRP) block cipher (or none, if so desired), the keys to be used, as well as the information about each revoked certificate in the CRT. The information required for each revoked certificate includes: certificate certificate number, reason for revocation, and path to a (file) copy of the revoked certificate. The certificates are assumed to be in the X.509v3 format, generated by the OpenSSL CA tool, in the default *.pem* output. A more complete description of the configuration file is included in the *default.conf* file distributed with the tool.

Once all settings are loaded from the configuration file, the server generates the corresponding CRT based on the permuted (via $PRP(\cdot)$) serial numbers and waits for clients to initiate a connection. When a client initially connects, the server responds with the global parameters for the system and waits for an actual query. When a query is received, the server returns all appropriate leaf nodes in the range requested by the client as well as the interior nodes corresponding to the co-paths, as described in Section 5.6 above.

The client component takes as input the server's IP address, the desired privacy level $p = \frac{1}{r}$ (where r is the query range size) and the serial number (C_i) of the target certificate. It then computes $PRP(K, C_i)$, and performs two queries on the server. The first query refers to a specific but random certificate. As described in Section 5.5, this is needed to establish revocation density. The client verifies the first reply, and, using, the length of the returned co-path in the reply, computes the number of leaf nodes. Then, it generates the random range boundaries necessary for the desired privacy level. The formulation of the second query, its processing by the server and reply verification by the client follow the protocol as described above.

The current prototype is a mere proof-of-concept of little practical use. Work is currently underway to construct a privacy-preserving CRT plug-in for the Mozilla Thunderbird and Eudora e-mail clients. These plug-ins will have the functionality roughly equivalent to the stand-alone prototype and will allow user-transparent certificate status checking for the intended email destination (in case of sending) and for the email source (as part of processing received email).

Appendix B: Range Queries with Skip-Lists

We now show how to construct privacy-preserving revocation checking with skip-lists proposed by Goodrich, et al. [12] The authenticated dictionary approach based on skip-lists and commutative hashing [12] can be used as a certificate revocation structure. The resultant data structure is a traditional skip-list amended with commutative hashing. A hash function is said to be commutative if $h(x,y) = h(y,x)$ for all x and y . A candidate construction for such a hash function is:

$$h(x,y) = f(\min\{x,y\}, \max\{x,y\})$$

Here, $h()$ is a hash function that takes as input two integer arguments, x and y of equal bit-size and maps them into a k -bit integer $h(x,y)$. Additionally, sequences of integers (x_1, x_2, \dots, x_n) can be hashed together by using the resultant hash as the input for the next iteration of the hash function: $h(x_1, h(x_2, \dots, h(x_{n-2}, h(x_{n-1}, h_n)) \dots))$

This notion of commutative hashing allows for the creation of authenticated dictionary based on skip-lists. Each node in the skip-list contains the hash of its neighbor to the right causing a hash chain up to the root. The root node represents the combined hash of all nodes in the skip-list. Further details of the hashing process (for both tower and plateau nodes) can be found in [12].

When used as certificate revocation structure, a skip-list with commutative hashing can also provide a short proof.

When a query for a target node is posed, the nodes along the search path are returned to the client who, by repeated commutative hashing, can verify the hash of the root. If the hash value matches the signed root then

Figure 9 shows the query path for value 75 in the skip-list. The colored nodes represent the search path taken to locate the node. These colored nodes become the hash values returned to the user to verify the root the hash. We can now easily

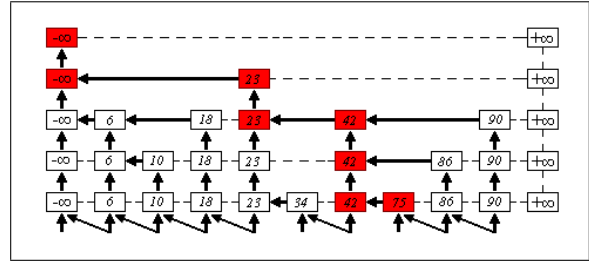


Figure 5. Query for 75 on a skip-list with commutative hashing. Colored nodes represent search path and arrows represent direction of hash flow to root.

extend this data structure to preserve privacy. The technique is similar to the original CRT solution. Instead of querying for a single node, we query for a single node and for a range of nodes to return. The result from the server is the search path for the smallest node in the query and all nodes in the query range.

Since each node contains the hash value of the node immediately to its right, the client takes each returned node and computes the hash for the smallest node. If the computed value correctly matches the value returned by the server the client can be assured that no nodes have been omitted from the search results, and that the results are *complete*.

The second step of the verification process involves using the search path to the smallest node and the smallest node itself to compute the hash value of the root node. If the computed value matches the signed root hash value then the client can be assured that all nodes returned are revoked and that none have been omitted. An example of this process can be seen in Figure 6. In this example, a client makes a query on node 10 with a range of 60, making the complete search range from 10 to 70. Node 75 must be included in the results to prove to the client that no nodes have been omitted from the search results.

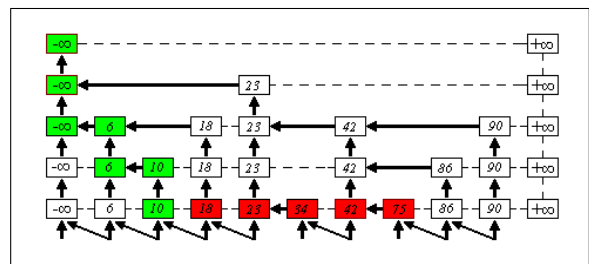


Figure 6. Range query (10,60) using skip-lists and commutative hashing.

The green nodes represent the search path to node 10, while the red nodes represent all nodes in the range returned to the client. A client can then verify the validity of the results using the process described above. In this example, nodes are hashed from 73 down to 10 and then verified that this value is the hash value returned by the server. If this verifies then the root hash is computed by using the search path (green) nodes.