Simple and Flexible Revocation Checking with Privacy

John Solis and Gene Tsudik

Computer Science Department
University of California, Irvine
{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 revocation checking. There are many revocation techniques varying in both the operational model and underlying data structures. One common feature is that a client typically contacts an on-line third party (trusted, untrusted or semi-trusted), identifies the certificate of interest and obtains some form of a proof of either revocation or validity (non-revocation) for the certificate in question.

While useful, revocation checking can leak potentially sensitive information. In particular, third parties of dubious trustworthiness discover two things: (1) the identity of the party posing the query, as well as (2) the target of the query. 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 as obvious. Arguably, a more important **loss of privacy** results from the third party's ability to tie the source of the revocation check with the query's target. (Since, most likely, the two are about to communicate.) This paper is concerned with the problem of privacy in revocation checking and its contribution is two-fold: it identifies and explores the loss of privacy inherent in current revocation checking, and, it constructs a simple, efficient and flexible privacy-preserving component for one well-known revocation method.

1 Introduction and Motivation

As is well-known, public key cryptography allows users to communicate privately without having pre-established shared secrets. While parties can be assured that communication is private, there is no guarantee of authenticity. Authenticity is obtained by binding a public key to some identity or name which is later verified via digital signatures in conjunction with public key certificates. A public key certificate, signed by a recognized certification authority (CA), can be 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

G. Danezis and P. Golle (Eds.): PET 2006, LNCS 4258, pp. 351–367, 2006.

the included name/identity to the public key. Digital certificates work best in large interconnected open systems, where it is generally infeasible to directly authenticate the owners of all public keys. X.509 [24] is one well-known certificate format widely used in several Internet-related contexts. The peer-based PGP/GPG [2,8] format represents another popular approach.

Since a certificate is essentially 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 [24]: private key loss or compromise, 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) [20]. 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.

Well-known explicit revocation methods (data structures) include Certification Revocation Lists (CRLs) and variations such as Δ-CRLs, CRL Distribution Points (CRL-DPs), Certificate Revocation Trees (CRTs) [15] and Skip-Lists [9]. Another prominent technique is the On-line Certificate Status Protocol (OCSP) [21] which involves a multitude of "somewhat-trusted" validation agents (VAs) which respond to client queries with on-demand signed replies indicating current status of a target certificate.

Regardless of their particulars, all current 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 checking revocation status (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 the revocation status check typically serves as a prelude to actual communication between the two parties.[1]

In the society preoccupied with gradual erosion of (electronic) privacy, loss of privacy in current revocation checking is an important issue worth considering. Consider, for example, certain countries with less-than-stellar human rights records where mere intent to communicate (indicated by revocation checking) with a "unsanctioned" 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. The problem can also manifest itself in less sinister

[1] We assume that communication between clients and on-line revocation agents (third parties) is private, i.e., conducted over secure channels protected by tools such as IPSec [12] or SSL/TLS [10].

settings. Many internet service providers keep detailed statistics and build elaborate profiles based on their clients' communication patterns. Current revocation checking methods – by divulging both sources and targets or revocation queries – represent yet another source of personal information that can be exploited by potentially unscrupulous providers.

The primary motivation for this paper is current lack of privacy in certificate revocation checking. The intended contribution of this paper is two-fold: first, it explores the loss of privacy inherent in current certificate revocation checking, and, second, it constructs a simple, efficient and flexible privacy-preserving add-on component for one well-known revocation method. We believe that the simplicity of our approach has a good chance of enabling its eventual adoption by the Internet *masses* most of whom at present (unfortunately) ignore revocation checking.

1.1 Focus

The first problem mentioned above (hiding the source of a revocation query) can be easily remedied with modern anonymization techniques, such as onion routing, anonymous web browsing or remailers. While this might protect the source of a revocation query, the target of the query remains known to the third party. Furthermore, although anonymization techniques are well-known in the research community, their penetration remains, overall, fairly low. Also, in order to take advantage of an existing anonymization infrastructure, one either needs to place some trust in unfamiliar existing entities (e.g., remailers, re-webbers or onion routers) or make the effort to create/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 the one that is most amenable to supporting efficient privacy-preserving querying.

Note 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) [14]. 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 niche' activity among Internet users, in contrast to DNS which is used by nearly all.

1.2 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 honor belongs to the recent work of Kikuchi [13]. This work identified the problem and proposed a fairly heavy-weight (inefficient) cryptographic technique specific to CRLs. The solution relies on so-called cryptographic accumulators which are, unfortunately, quite expensive.

A related research topic is Private Information Retrieval (PIR) [4,16]. 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.

2 Certificate Revocation Techniques

In this section, we briefly overview 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 secure data structures such as CRLs. Unlike a CA, which is always off-line, an RA may be partially on-line to facilitate fast distribution of revocation information.

CRLs and Δ-CRLs: These are the most common ways to handle certificate revocation. The Validation Authority (VA) periodically posts a signed list (or a similar structure) containing all revoked certificates. Such lists are placed on designated servers, called CRL Distribution Points. Since a list can get quite long, a VA may post a signed Δ-CRL which only contains the list of certificates revoked since the last CRL was issued. In the context of encrypted email, at the time email is sent, the sender checks if the receiver's certificate is included in the latest CRL. To verify a signature on a signed email message, the verifier first checks if (at present time) the signer's certificate is included in the latest CRL.

OCSP: The Online Certificate Status Protocol (OCSP) [21] avoids the generation and distribution of long CRLs and provides 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, unknown) of the specified certificate.

Certificate Revocation Trees: In 1998, Kocher suggested an improvement for OCSP [15]. Since the VA is a global service, it must be sufficiently replicated in order to handle the load of all the 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 the VA's signing key). Kocher's idea is a single highly secure VA 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 a hash tree[2] where the leaves represent currently revoked certificates sorted by serial number (lowest serial number is the left-most leaf and

[2] More, accurately, a Merkle Hash Tree (MHT) [19].

the highest serial number is the right-most leaf). The root of the hash tree is signed by the VA. This data structure is called a Certificate Revocation Tree (CRT). When a client wishes to validate a certificate CERT, she issues a query to the nearest VA server. Any insecure VA can produce a proof that CERT is (or is not) on the CRT. If n certificates are revoked, the length of the proof is $O(\log n)$. In contrast, the length of the validity proof 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 VA 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 [22] and skip-lists proposed by Goodrich, et al. [9] 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.

3 Zooming In

Looking at the approaches reviewed above, it seems that retrofitting privacy into CRLs or Δ-CRLs is not easy. This observation is supported by the recent attempt by Kikuchi in [13]. As mentioned in Section 1.2, 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 bandwidth and client storage.

Similarly, making plain OCSP privacy-preserving is 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:

– **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.

– **and**

– **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.

In the rest of this paper we describe CRTs in more detail (Section 42), present our modifications to support privacy (Section 5, describe our prototype implementation (Section 6) and conclude with examples (Section 7) and future directions (Section 8). Our approach in the context of skip-lists is presented in the Appendix.

4 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. [3] Let $L_1, ..., L_m$ 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 [23]. As in any Merkle Hash Tree [19], each non-leaf node is recursively computed bottom-up by hashing the concatenation of its left and right

[3] In practice, a CRT does not need to be perfectly balanced.

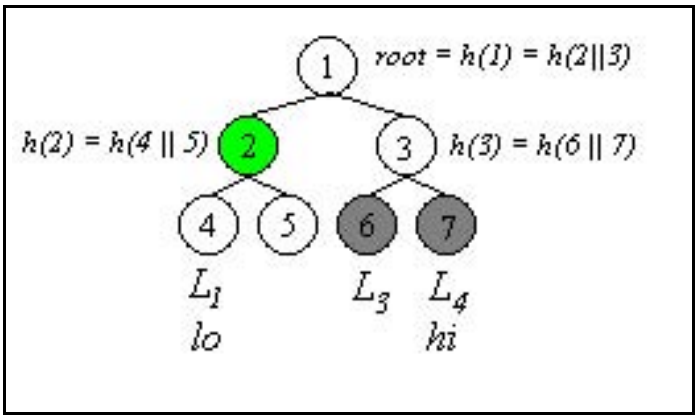

Fig. 1. An example CRT query for node L_3

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.

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

5 Range Queries

The basic idea behind range queries is very simple, even trivial. 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.

5.1 Range Size

As with many simple solutions, the challenge lies 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.[4] 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 4, 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:

[4] This is equivalent to obtaining an entire CRL.

$j = i - X$ and $k = j + r - 1$. (Special care must be taken if $i < X$ or $i - X < lo$. More on this below.)

Incidentally, we observe that, if a client poses **repeated queries** against the same target certificate, varying the query range and boundaries is not advisable. This is because, otherwise, the adversary can gradually narrow down the set of possible targets by repeatedly computing the intersection of multiple query ranges. To avoid this situation, our prototype implementation – described in Section 6 – 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.

5.2 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. Put another way, we claim that:

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

The only information learned by an adversary about the potential 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 a 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 $\frac{1}{k-j}$. 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 queires' ranges.

5.3 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 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.

5.4 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 of 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 includes 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 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.

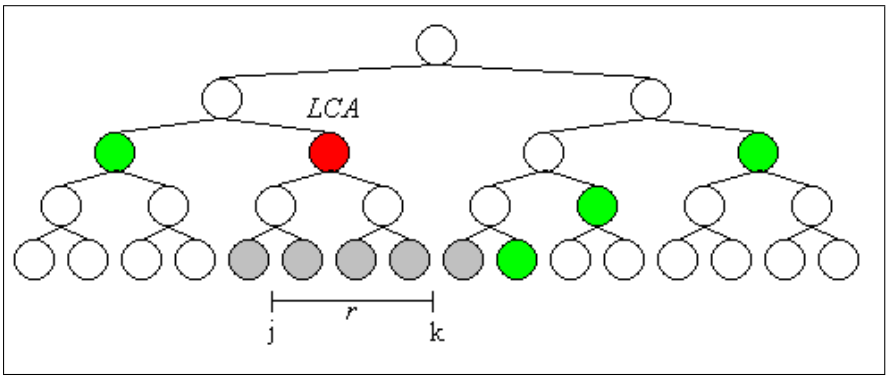

Fig. 2. The LCA and co-path nodes for a given (j, k) range

5.5 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. [5] 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 [5]. 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

[5] This is true even in light of certain new attacks [18]. Such attacks allow an 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.

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 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 [10]. 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 5.4 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 5.3, 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).

7 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 [6]. 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 [17]. 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 [11,3].

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. [3] 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 approach offers client-selectable bandwidth/privacy trade-off.

8 Future Directions

The proposed range query approach takes advantage of the CRT structure to offer low bandwidth overhead and obtain client-specified level of privacy. The CRT structure has the additional benefit of providing efficient (in terms of computation) cryptographic proofs for target certificates. However, our approach represents only the initial simple step in this line of research and much more remains to be done.

One outstanding issue is the analysis of privacy loss 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, co-relating multiple range queries does not seem difficult since all the range queries in question would have at least one certificate in common. A similar situation occurs if a single client, over time, repeatedly queries the status of the same target certificate – in this case, narrowing the overlap of all queries' ranges gradually erodes privacy and might eventually yield a single target certificate.

Finally, the usability factor remains largely unexplored. Many wonderful 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.

9 Conclusions

The work described in this paper represents a very simple yet novel approach for addressing privacy concerns in revocation checking. Each client, depending on the desired level of privacy, can determine a query range that best suits its needs. This results in a fundamental trade-off between privacy and bandwidth overhead. In the worst case, the overhead can be significant if the desired privacy level is high and as is the number of revoked certificates. However, if only a small fraction of all certificates are revoked, our approach results is reasonably efficient. Furthermore, experience from real-world environments (based on revocation statistics from government, commercial, and military sources) suggests that the proposed solution would work well since revoked certificates represent a tiny fraction of the total numbers of issued certificates.

Acknowledgments

We thank Einar Mykletun, Maithili Narasimha and Marina Blanton for their comments on the draft of this paper.

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. 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.
4. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylog communication. In *Proceedings of Eurocrypt'99*, LNCS. IACR, Springer Verlag, 1999.
5. Verisign Corporation. Compare all ssl certificates from verisign, inc. `http://www.verisign.com/products-services/security-services/ssl/buy-ssl-certificates/compare/index.html`.
6. Verisign Corporation. Corporate overview: Fact sheet from verisign, inc. `http://www.verisign.com/verisign-inc/corporate-overview/fact-sheet/index.html`.
7. Verisign Corporation. Public online crl repository. `http://crl.verisign.com/`.
8. Inc. Free Software Foundation. Gnu privacy guard, `http://www.gnupg.org/`.
9. M. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proceedings of DARPA DISCEX II*, 2001.
10. OpenSSL User Group. The openssl project web page, `http://www.openssl.org`.
11. J. Hackerson. Rethinking department of defense public key infrastructure. In *Proceedings of 23rd National Information Systems Security Conference*, October 2000.
12. S. Kent and R. Atkinson. Security architecture for the internet protocol. Internet Request for Comments: RFC 2401, November 1998. Network Working Group.
13. H. Kikuchi. Privacy-preserving revocation check in pki. In *2nd US-Japan Workshop on Critical Information Infrastructure Protection*, pages 480–494, July 2005.
14. J. Klensin. Role of the domain name system (dns). Internet Request for Comments: RFC 3467, February 2003. Network Working Group.
15. P. Kocher. On certificate revocation and validation. In *Proceedings of Financial Cryptography 1998*, pages 172–177, 1998.
16. 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.
17. US Army Research Laboratory. Using the cac with pki - faqs. `http://www.usaarl.army.mil/CBT/EndUser/chapter_06b/chapter06b.html`.
18. Arjen Lenstra, Xiaoyun Wang, and Benne de Weger. Colliding x.509 certificates. Cryptology ePrint Archive, Report 2005/067, 2005. `http://eprint.iacr.org/`.
19. R. Merkle. *Secrecy, Authentication, and Public-Key Systems*. PhD thesis, Stanford University, 1979. PH.D Dissertation, Department of Electrical Engineering.
20. S. Micali. Certificate revocation system. United States Patent 5666416, September 1997.
21. 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.

22. M. Naor and K. Nissim. Certificate revocation and certificate update. *IEEE Journal on Selected Areas in Communications (JSAC)*, 18(4):561–570, April 2000.
23. National Institute of Standards and Technology. Federal information processing standards (fips), publication 180-2, secure hash standard (shs), February 2004.
24. 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.

Appendix A: Range Queries in Skip-Lists

The ranqed query technique can also be applied to other revocation structures mentioned in Section 2. We now discuss providing privacy in the context of skip-lists which were proposed for revocation purposes by Goodrich, et al. [9] The authenticated dictionary approach based on skip-lists and commutative hashing [9] can be used as a certificate revocation structure. The resultining 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 two integer arguments, x and y of equal bit-size and maps them to a k-bit integer $h(x, y)$. Additionally, sequences of integers $(x_1, x_2, ..., x_n)$ can be hashed together by using the resultant hash as the input for the next iteration of the hash function: $h(x_1, h(x_2, ...h(x_{n-2}, h(x_{n-1}, h_n)) \cdots))$

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 [9].

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 3 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 extend this data structure to preserve privacy. The technique is similiar 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.

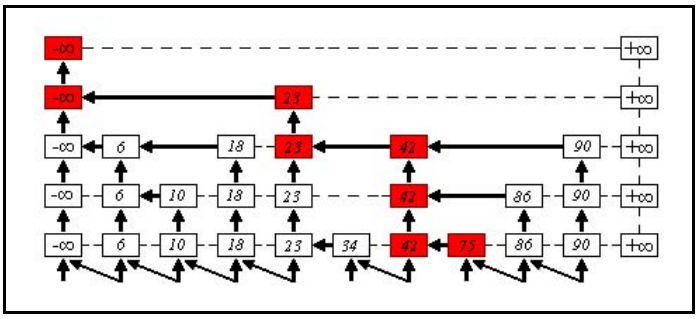

Fig. 3. Query for 75 on a skip-list with commutative hashing. Colored nodes represent search path and arrows represent direction of hash flow to root

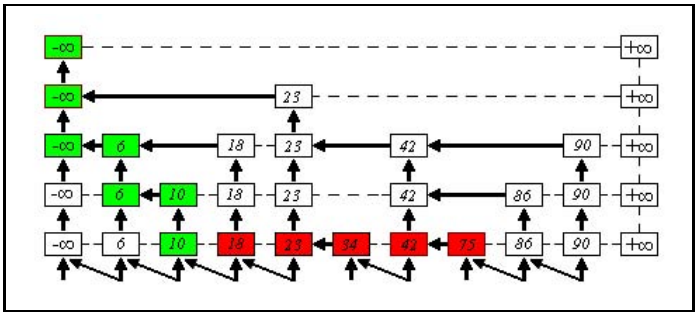

Fig. 4. Range query (10,60) using skip-lists and commutative hashing

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 4. 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.

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.