Certificate Revocation and Certificate Update

Moni Naor* Kobbi Nissim
Dept. of Applied Mathematics and Computer Science
Weizmann Institute of Science
Rehovot 76100, Israel
{naor, kobbi}@wisdom.weizmann.ac.il

Abstract

A new solution is suggested for the problem of certificate revocation. This solution represents Certificate Revocation Lists by an authenticated search data structure*. The process of verifying whether a certificate is in the list or not, as well as updating the list, is made very efficient. The suggested solution gains in scalability, communication costs, robustness to parameter changes and update rate. Comparisons to the following solutions are included: 'traditional' CRLs (Certificate Revocation Lists), Micali's Certificate Revocation System (CRS) and Kocher's Certificate Revocation Trees (CRT).*
Finally, a scenario in which certificates are not revoked, but frequently issued for short-term periods is considered. Based on the authenticated search data structure scheme, a certificate update scheme is presented in which all certificates are updated by a common message.
The suggested solutions for certificate revocation and certificate update problems is better than current solutions with respect to communication costs, update rate, and robustness to changes in parameters and is compatible e.g. with X.500 certificates.

1 Introduction

The wide use of public key cryptography requires the ability to verify the authenticity of public keys. This is achieved through the use of certificates (that serve as a mean for transferring trust) in a *Public Key Infrastructure* (PKI). A certificate is a message signed by a publicly trusted authority (the *certification authority,* whose public key authenticity may be provided by other means) which includes a public key and additional data, such as expiration date, serial number and information regarding the key and the subject entity.

*Research supported by BSF grant no. 94-00032.

When a certificate is issued, its validity is limited by an expiration date. However, there are circumstances (such as when a private key is revealed, or when a key holder changes affiliation or position) where a certificate must be revoked prior to its expiration date. Thus, the existence of a certificate is a necessary but not sufficient evidence for its validity, and a mechanism for determining whether a certificate was revoked is needed.
A typical application is a credit card system where the credit company may revoke a credit card, temporarily or permanently, prior to its expiration, e.g. when a card is reported stolen or according to its user's bank account balance.
This work focuses on the problem of creating and maintaining efficient authenticated data structures holding information about the validity of certificates. I.e. how to store, update and retrieve authenticated information concerning certificates.
There are three main types of parties involved with certificates:

1. **Certification authority (CA)**: A trusted party, already having a certified public key, responsible for establishing and vouching for the authenticity of public keys, including the binding of public keys to users through certificates and certificate revocation.

   A CA does not provide on-line certificate information services to users. Instead, It updates a directory on a periodic basis.

   A CA issues certificates for users by signing a message containing the certificate serial number, relevant data and an expiration date. The certificate is sent to a directory and/or given to the user himself. The CA may revoke a certificate prior to its expiration date.

2. **Directory**: One or more non-trusted parties that get updated certificate revocation information from the CA and serve as a certificate database accessible by the users.

3. **User**: A non-trusted party that receives its certificate from the CA, and issues queries for certificate information. A user may either query the validity of other users' certificates (we denote users that query other users' certificates as *merchants*) or, get a proof of the validity of his certificate in order to present it with his certificate (for the latter, the proof must be transferable).

The rest of this paper is organized as follows: In Section 2 we briefly review the solutions we are aware of (CRL, CRS and CRT), memory checkers and incremental cryptographic schemes. In Section 3 we give some basic definitions and the theoretical background and restate the problem in terms of finding efficient *authenticated directories* and, in particular, *authenticated search data structures.* The proposed scheme is described in detail in Section 4 and compared with the other schemes in Section 5. Finally, in Section 6, we consider a model in which a directory is not used for extracting certificates, and certificates are updated periodically. We show how a simple modification of our revocation scheme works in this model.

2 Related work and background

In this section we review the solutions we are aware of, namely *Certificate Revocation List* (CRL [20]), *Certificate Revocation System* (CRS [18]) and *Certificate Revocation Trees* (CRT [16]). We continue by reviewing memory checkers, and incremental cryptographic schemes, relating these problems to certificate revocation, these two sections are included as theoretical background, and are not necessary for understanding the rest of the paper.

2.1 Certificate Revocation List (CRL)

A CRL is a signed list issued by the CA identifying all revoked certificates by their serial numbers. The list is concatenated with a time stamp (as an indication of its freshness) and signed by the CA that originally issued the certificates. The CRLs are sent to the directory on a periodic basis, even if there are no changes, to prevent the malicious replay of old CRLs instead of new CRLs.
As an answer to a query, the directory supplies the most updated CRL (the complete CRL is sent to the merchant).

- The main advantage of the scheme is its simplicity.
- The main disadvantage of the scheme is its high directory-to-user communication costs (since CRLs may get very long). Another disadvantage is that a user may not hold a succinct proof for the validity of his certificate.

A reasonable validity expiration period should be chosen for certificates. If the expiration period is short, resources are wasted reissuing certificates. If the expiration period is long, the CRL may get long, causing high communication costs and difficulties in CRL management. Kaufman *et al.* [15, Section 7.7.3] suggested reissuing all certificates whenever the CRL grows beyond some limit. In their proposal, certificates are marked by a serial number instead of an expiration date. (Serial numbers are incremented for each issued certificate. Serial numbers are not reused even when all certificates are reissued.) The CRL contains a field indicating the *first valid certificate.* When all certificates are reissued, the CRL first valid certificate field is updated to contain the serial number of the first reissued certificate.

2.2 Certificate Revocation System

Micali [18] suggested the Certificate Revocation system (CRS) in order to improve the CRL communication costs. The underlying idea is to sign a message for every certificate stating whether it was revoked or not, and to use an off-line/on-line signature scheme [11] to reduce the cost of periodically updating these signatures.
To create a certificate, the CA associates with each certificate two numbers (Y_{365} and N) that are signed along with the 'traditional' certificate data. For each certificate, the CA chooses (pseudo)randomly two numbers N_0, Y_0 and computes (using a one-way function f) $Y_{365} = f^{365}(Y_0)$ and $N = f(N_0)$. (Actually, a stronger assumption on f is required, e.g. that f is one-way on its iterates, i.e. that given $y = f^i(x)$ it is infeasible to find x' such that $y = f(x')$. This is automatically guaranteed if f is a one-way permutation.)
The directory is updated daily by the CA sending it a number C for each certificate as follows:

1. For a non-revoked certificate the CA reveals one application of f, i.e. $C = Y_{365-i} = f^{365-i}(Y_0)$, where i is a daily incremented counter, $i = 0$ on the date of issue.

2. For a revoked certificate, $C = N_0$.

Thus the most updated value for C serves as a short proof (that certificate x was or was not revoked) that the directory may present in reply to a user query x.

- The advantage of CRS over CRL is in its query communication costs. Based on Federal PKI (Public Key Infrastructure) estimates, Micali [18] showed that although the daily update of the CRS is more expensive than a CRL update, the cost of CRS querying is much lower. He estimated the resulting in 900 fold improvement in total communication costs over CRLs. The exact parameters appear in Section 5.

  Another advantage of CRS is that each user may hold a succinct transferable proof of the validity of his certificate. Directory accesses are saved when users hold such proofs and presents them along with their certificates.

- The main disadvantage of this system is the increase in the CA-to-directory communication (it is of the same magnitude as directory-to-users communication, where the existence of a directory is supposed to decrease the CA's communication). Moreover, since the CA's communication costs are proportional to the directory update rate, CA-to-directory communication costs limit the directory update rate.

  The complexity of verifying that a certificate was not revoked is also proportional to the update rate. For example, for an update once an hour, a user may have to apply the function, f, $365 \times 24 = 8760$ times in order to verify that a certificate was not revoked, making it the dominant factor in verification.

The complexity of the Micali's method of verifying a certificate may be improved as follows. Let h be a collision intractable hash function. To issue a certificate, the CA creates a binary hash tree as follows: The tree leaves are assigned (pseudo)random values. Each internal node v is assigned the value $h(x_1, x_2)$ where x_1, x_2 are the values assigned to v's children. The CA signs the root value and gives it as a part of the certificate, the other tree values (and specifically the (pseudo)random values assigned to its leaves are not revealed). To refresh a certificate the ith time, the CA reveals values of the nodes on the path from the root to leaf $2i$ and their children (Thus, the verifier can check that the nodes are assigned a correct. Note that it is not necessary to explicitly give the values of the nodes on the path since these values may be easily computed given the other values). The path serves as a proof for the certificate validity. Amortizing the number of tree nodes the CA has to send the directory, we get that four nodes are sent to update a user's certificate. We make further use of the hash tree scheme in the following sections.

2.3 Certificate Revocation Trees

Kocher [16] suggested the use of Certificate Revocation Trees (CRT) in order to enable the verifier of a certificate to get a short proof that the certificate was not revoked. A CRT is a hash tree with leaves corresponding to a set of statements about certificate serial number X issued by a CA, CA_x.
The set of statements is produced from the set of revoked certificates of every CA. It provides the information whether a certificate X is revoked or not (or whether its status is unknown to the CRT issuer).
There are two types of statements: specifying ranges of unknown CAs, and, specifying certificates range of which only the lower certificate is revoked. For instance, if CA_1 revoked two certificates, $X_1 < X_2$, than one of the statements is:

$$\textit{if } CA_x = CA_1 \textit{ and } X_1 \leq X < X_2 \textit{ then } X \textit{ is revoked iff } X = v$$

To produce the CRT, the CRT issuer builds a binary hash tree [17] with leaves corresponding to the above statements.
A proof for a certificate status is a path in the hash tree, from the root to the appropriate leaf (statement) specifying for each node on the path the values of its children.

- The main advantages of CRT over CRL are that the entire CRL is not needed for verifying a specific certificate and that a user may hold a succinct proof of the validity of his certificate.

- The main disadvantage of CRT is in the computational work needed to update the CRT. Any change in the set of revoked certificates may result in re-computation of the *entire* CRT.

2.4 Checking the correctness of memories

Blum *et al.* [3] extended the notion of program checking to programs which store and retrieve data from unreliable memory. In their model, a data structure resides in a large memory controlled by an adversary. A program interacts with the data structure via a *checker*. The checker may use only a small reliable memory and is responsible for detecting errors in the data structure behavior while performing the requested operations. An error occurs when a value returned by the data structure does not agree with the corresponding value entered into the data structure. Blum *et al.* showed how to construct an online checker for RAMs using a variant of Merkle's hash-tree authentication scheme for digital signatures [17]. They used universal one way hash functions [19]. [1]

Certificate revocation may be regarded as a variant of memory checking. As in memory checking, an unreliable memory is used for storing and retrieving data. The difference is that in memory checking the same program writes into and reads from memory via the checker, whereas in certificate revocation there exist two distinct non-communicating programs. One program (the CA) writes into an unreliable memory (the directory), the other (a user) reads from the unreliable memory. The fact that the two programs are disconnected raises the need for a mechanism to prevent an adversary from *replaying* old data.

Returning to memory checking, our solution may be regarded as a checker for *dictionaries*.

2.5 Incremental cryptographic schemes

The high CA-to-directory communication in CRS is due to the fact that the CA has to update values not only for certificates whose status was changed since the last update, but for all certificates. Since the fraction of certificates that change status in every update is expected to be small, it would be preferable to use a scheme with communication (and computational work) depending mostly on the number of certificates that change status. Such issues are addressed by *incremental cryptography*.

Incremental cryptography was introduced by Bellare, Goldreich and Goldwasser [4, 5]. The goal of incremental schemes is to quickly update the value of a cryptographic primitive (e.g. hash function, MAC etc.) when the underlying data is modified. Informally, for a given set of modification operators (e.g. insert, delete, replace), in an *ideal* incremental scheme, the computational work needed for updating a value depends only on the number of data modifications.

An ideal incremental authentication scheme based on a 2-3 search tree was suggested in [5]. The scheme is a modification of a standard tree authentication scheme [17] in order to allow efficient insert/delete block operations along with replace block operations. This scheme cannot be used directly for our problem, we modify it for our purposes in Section 4.

[1] Informally, U is a family of universal one way hash functions if $\forall x$, for f chosen at random from U, it is infeasible to find y such that $f(x) = f(y)$.

3 Authenticated dictionaries

In this section we consider a more abstract version of the problem and translate it to the problem of finding efficient authenticated data structures. The less theoretically inclined readers may skip directly to Section 4 that presents a self-contained description of such a data structure.

Put in an abstract framework, the problem we deal with is the following: find a protocol between a non-trusted prover, P, and a verifier, V for (non)membership in a set S, where S is some finite set defined by a trusted party (the CA) but not known to V. Given an input x, P should prove either $x \in S$ or $x \notin S$, The trusted party may change S dynamically, but it is assumed to be fixed while P and V interact.

The prover has access to a data structure representing S along with some approved public information about S, created by the trusted party prior to setting x. The non-trusted prover should have an efficient procedure for providing on-line a short proof (e.g. of low order polynomial in $|x|, \log |S|$) for the appropriate claim for x.

3.1 Definitions

Let U be a universe and S be a set such that $S \subseteq U$. Let D_S be a data structure representing S.

- A *membership query* is of the form $\langle e \rangle$. The response to the query is a string $\langle a \rangle$ where $a \in$ {YES,NO}, corresponding to $e \in S$, $e \notin S$.
- An *authenticated membership query* is of the form $\langle e \rangle$. The response to the query is a string $\langle a, p \rangle$ where $a \in$ {YES,NO} and p is a proof for a authenticated by a CA.
- *Update* operations are of the form
  1. $\langle Insert, e \rangle$ where e is an element in $U \setminus S$. The resulting data structure $D_{S'}$ represents the set $S' = S \cup \{e\}$.
  2. $\langle Remove, e \rangle$ where e is an element in S. The resulting data structure $D_{S'}$ represents the set $S' = S \setminus \{e\}$.

Definition 3.1 *A* dictionary *is a data structure D_S representing S supporting membership queries and update operations.*

Definition 3.2 *An* authenticated dictionary *is a data structure D_S^{au} representing S supporting authenticated membership queries and update operations.*

In our model, the set S is known both to the CA and the prover, P, but not to the verifier, V. The CA controls S and supplies P with the information needed to create an authenticated dictionary representing S.

Since an authenticated dictionary is dynamic, a mechanism for proving that an authenticated proof is updated is needed. Otherwise, a dishonest directory may replay old proofs. In our model we may assume either that CA updates occur at predetermined times, or, that the user issuing a query knows when the most recent update occurred. In any case, the verifier should be able to check the freshness of a proof p.

The parameters we are interested in regarding authenticated dictionaries are:

- Computational complexity:
  1. The time and space needed to authenticate the dictionary, i.e. creating and updating it.
  2. The time needed to perform an authenticated membership query.
  3. The time needed to verify the answer to an authenticated membership query.
- Communication complexity:
  1. The amount of communication (CA to prover) needed to update the dictionary.
  2. The length of a proof p for an authenticated membership query.

3.2 Implementing authenticated dictionaries

For a small universe U, one can afford computational work proportional to $|U|$. There are two trivial extremes with respect to the number of signed messages, the computation needed in authentication and verification, and the prover to verifier communication complexity:

- For every $e \in U$ the CA signs the appropriate message $e \in U$ or $e \notin U$. To update D_s, $|U|$ signatures are supplied, regardless the number of changed elements in D_S. An example of this solution is the certificate revocation system reviewed in Section 2.2.
- The CA signs a message $M = \sigma_1, \sigma_2, \ldots, \sigma_{|U|}$ where for every $u_i \in U$, σ_i indicates whether $u_i \in S$.

If S is expected to be small, two simple solutions are:

- The CA signs intervals of elements not in S. Such an interval is a pair (s_1, s_2) satisfying $s \notin S$ for all $s_1 \leq s \leq s_2$.
- The CA signs a message M containing a list of every $s \in S$. An example is the certificate revocation lists reviewed in Section 2.1.

In all the solutions above, the messages are signed by the CA and include the time of update.

In the following we describe a generic method for creating an authenticated dictionary D_S^{au} from a dictionary D_S. The overhead in membership queries and update operations is roughly a factor of $\log |D_S|$. In this construction we use a *collision intractable hash function.*

Definition 3.3 *A collision intractable hash function is a function $h()$ such that it is computationally infeasible to find $y \neq x$ satisfying $h(x) = h(y)$.*

Let D_S be a dictionary of size $|D_S|$ representing a set S. Let T_q, T_u be the worst case time needed to a compute membership query or an update operation respectively.

Let h be a collision intractable hash function, and T_h be the time needed to compute h on instances of U. Consider the representation $D_S = (\delta_1, \delta_2, \ldots)$ of S. This may be e.g. a list of all the variables' values composing D_S, or, the way D_S is represented in random access memory. The authenticated dictionary D_S^{au} contains D_S plus a hash tree [17] whose nodes correspond to $\delta_1, \delta_2, \ldots$, and a message signed by the CA containing the tree root value and the time of update.

The hash tree is constructed as follows: A balanced binary tree is created whose leaves are assigned the values $\delta_1, \delta_2, \ldots$. Each internal node v is then assigned a value $h(x_1, x_2)$ where x_1, x_2 are the values assigned to v's children.

- A membership query is translated to an authenticated membership query by supplying a proof for every item δ_i of D_S accessed in the computation. The proof consists of the values of all the nodes on the path from the root to position i and their children. The complexity of an authenticated membership query is thus $O(T_q \cdot T_h \cdot \log |D_S|)$.

- After an update, the portion of the hash tree corresponding to elements δ_i that were changed is re-computed (i.e. all paths from a changed element δ_i to the root). The complexity of an update operation is thus $O(T_u \cdot T_h \cdot \log |D_S|)$.

3.3 Authenticated search data structures

The general method of Section 3.2 for creating dictionaries has a logarithmic (in $|D_s|$) multiplicative factor overhead. The reason is that the internal structure of D_S was not exploited in the authentication/verification processes.

Our goal is to create authenticated dictionaries based on efficient search data structures that save the logarithmic factor overhead. We denote these as *authenticated search data structures.*

CRTs reviewed in Section 2.3 save this logarithmic factor in membership query complexity (but not in update where the amount of computational work is not a function of the number of changes but of the size of the revocation list). In Section 4.1 we show how to create authenticated search data structures based on search trees. An interesting open question is how to construct more efficient authenticated search data structures, e.g. based on hash tables, where membership query is processed in roughly $O(1)$.

4 The proposed scheme

The proposed scheme is closer in spirit to CRL and CRT than to CRS, since it maintains a list of only the revoked certificates. It reduces the CA's communication and actually makes it feasible to update the directory periodically many times a day, achieving a very fine update granularity. The revoked certificates list is maintained in an authenticated search data structure. The benefits of this construction are:

1. It is easy to check and prove whether a certain certificate serial number is in the list or not, without sending the complete list.
2. List update communication costs are low.

The underlying idea is to imitate a search in a data structure constructing a proof for the result during the search. For that, we combine a hash tree scheme (as in [17]) with a sort tree, such that tree leaves correspond to revoked certificates, sorted according to their serial numbers. Both proving that a certificate is revoked and that a certificate is not revoked reduce to proving the existence of certain leaves in the tree:

- Proving that a certificate was revoked is equivalent to proving the existence of a leaf corresponding to it.
- Proving that a certificate was not revoked is equivalent to proving the existence of two certificates corresponding to two neighboring leaves in the tree. One of these certificates has a lower serial number than the queried certificate, and the other has a higher serial number. (We modify this to a proof of the existence of a single leaf at the end of this section.)

4.1 An authenticated search data structure

We maintain a 2-3 tree with leaves corresponding to the revoked certificates' serial numbers in increasing order. (In a 2-3 tree every interior node has two or three children and the paths from root to leaves have the same length. Testing membership, inserting and deleting a single element are done in logarithmic time, where the inserting and deleting of an element affect only the nodes on the insertion/deletion path. For a detailed presentation of 2-3 trees see [1, pp. 169-180].) The property of 2-3 trees that we use is that membership queries, insertions and deletions involve only changes to nodes on a search path. I.e. every change is local and the number of affected paths is small.

The tree may be created either by inserting the serial numbers of the revoked certificates one by one into an initially empty 2-3 tree, or, by sorting the list of serial numbers and building a degree 2 tree with leaves corresponding to the serial numbers in the sorted list (because the communication complexity is minimal when the tree is of degree 2).

Every tree node is assigned a value according to the following procedure:

- Each leaf stores a revoked certificate serial number as its value.
- The value of an internal node is computed by applying a *collision intractable* hash function $h()$ to the values of its children.

The tree update procedure is as follows:

1. Delete each expired certificate serial number from the 2-3 tree, updating the values of the nodes on the deletion path.

2. Insert each newly revoked certificate serial number into the tree, updating the values of the nodes on the insertion path.

During tree update some new nodes may be created or some nodes may be deleted due to the balancing of the 2-3 tree. These nodes occur only on the search path for an inserted/deleted node.
The certification authority vouches for the authenticity of the data structure by signing a message containing the tree root value and the tree height. A proof that there exists a leaf in the tree with a certain value consists of all node values on a path (of length equal to the tree height) from the root to a leaf, plus the values of these nodes children. The proof is verified by checking the values of the tree nodes values on the given path and its length. Finding a fallacious proof for the existence of a leaf in the tree amounts to breaking the collision intractability of h.

Remark 4.1 *Possible choices for h include the more efficient MD4 [22], MD5 [23] or SHA [21] (collisions for MD4 and for the compress function of MD5 were found by Dobbertin [9, 10]) and functions based on a computational hardness assumption such as the hardness of discrete log [8, 7, 4]* [2] *and subset-sum [14, 12] (these are much less efficient).*

Remark 4.2 *Note that an explicit serial number is not needed. Instead, any string that is easily computed from the certificate (e.g. hash of the certificate) may be used.*

Remark 4.3 *It is possible to use a family of universal one-way hash functions, U, instead of collision intractable hash functions by letting every internal node, v, hold also an index of a function $h \in U$. The function h is randomly chosen whenever v lies in a deletion or insertion path. The value of a node is computed by applying h to the values of its children concatenated with their hash function indices. A motivation for using universal one-way hash functions instead of collision intractable hash functions is the successful attacks on MD4 and MD5 [9, 10]. Since universal one-way hash functions are not susceptible to birthday attacks, their application may result in a smaller increase in communication and storage costs with respect to collision intractable functions. Bellare and Rogaway [6] discuss methods for creating practical universal one-way hash functions.*

[2] The function is $h(x_1, x_2, x_3) = g_1^{x_1} g_2^{x_2} g_3^{x_3} \pmod p$. Let g be a generator in $\mathbb{Z}_p$, the CA may generate $g_i = g^{a_i}$ and compute h using a single exponentiation by $h = g^{\sum a_i x_i \pmod{p-1}} \pmod p$.

Remark 4.4 *The scheme may be used also for on-line revocation checking of certificates (where the latency between certificate revocation and CRL update is reduced). As the result of a query, the on-line service is supposed to return the current certificate status.*
In general, on-line revocation checking requires the certificate validator to be trusted (where in off-line checking, the directory could be a non-trusted party). In practice, it is enough that the certificate validator honestly informs a user about the last time it was updated by the CA (and may be dishonest with respect to other information). then it is not needed for the CA to update it only on predetermined times, and the CA may update the directory whenever the status of a certificate is changed. Even if such an assumption is not plausible, the CA may use the authenticated search data structure to reduce the number of signatures it has to compute, since a signature has to be computed only when a certificate status is changed and not for every query.

4.1.1 Other data structures

For a simpler implementation of the authenticated data structure, random treaps [2] may be used instead of 2-3 trees. *Treaps* are binary trees whose nodes are associated with (key, priority) pairs. The tree is a binary search tree with respect to node keys (i.e. for every node the keys in its left (resp. right) subtrees are small (resp. greater) than its key), and a heap with respect to node priorities (i.e. for every node its priority is higher than its descendents' priorities). Every finite set of (key, priority) pairs has a unique representation as a treap. In *random treaps*, priorities are drawn at random from a large enough ordered set (thus, they are assumed to be distinct).
Seidel and Aragon [2] present simple algorithms for membership queries, insert and delete operations with expected time complexity logarithmic in the size of the set S stored in the treap. Random treaps may be easily converted into authenticated search data structures similarly to 2-3 trees. The communication costs of these schemes is similar since the expected depth of a random treap is similar to its 2-3 tree counterpart.

- The main advantage of random treaps is that their implementation is much more simple than the implementation of 2-3 trees.

- A drawback of using random treaps is that their performance is not guaranteed in worst case.

E.g. some users may (with low probability) get long authentication paths.

- Another drawback is that a stronger assumption is needed with respect to the directory. The analysis of random treaps is based on the fact that the adversary does not know the exact representation of a treap. A dishonest directory with ability to change the status of certificates may increase the computational work and communication costs of the system.

4.2 The scheme

We now give details for the operations of the three parties in the system.

CA operations:

- **Creating certificates:** The CA produces a certificate by signing a message containing certificate data (e.g. user name and public key), certificate serial number and expiration date.
- **Initialization:** The CA creates the 2-3 tree, as above, for the set of initially revoked certificates. It computes and stores the values of all the tree nodes and sends to the directory the (sorted) list of revoked certificates serial numbers along with a signed message containing the tree root value, the tree height and a time stamp.
- **Updating:** The CA updates the tree by inserting/deleting certificates from it. After each insertion/deletion, all affected tree node values are re-computed. To update the directory, the CA sends a difference list (stating which certificates are to be added/deleted from the previous list) to the directory plus a signature on the new root value, tree height and time stamp.

Directory operations:

- **Initialization:** Upon receiving the initial revoked certificates list, the directory computes by itself the whole 2-3 tree, checks the root value, tree height and time stamp, and verifies the CA's signature on these values.
- **Response to CA's update:** The directory updates the tree according to the difference list received from the CA. It re-computes the values for all the affected nodes and checks the root value, tree height and time stamp.
- **Response to users' queries:** To answer a user query the directory supplies the user with the signed root value, tree height and time stamp.
  1. If the queried certificate is revoked, for each node in the path from the root to the leaf corresponding to the queried certificate, the directory supplies the user its value and its children values.
  2. If the queried certificate is not revoked (not in the list), the directory supplies the user the paths to two neighboring leaves ℓ_1, ℓ_2 such that the value of ℓ_1 (resp. ℓ_2) is smaller (resp. larger) than the queried serial number.

  Note that to reduce the communication costs, the directory need not send the node values on the path from root, but only the values of the siblings of these nodes, since the user may compute them by itself.

User operations:

The user first verifies the CA's signature on the certificate and checks the certificate expiration date. Then, the user issues a query by sending the directory the certificate serial number s. Upon receiving the directory's answer to a query, the user verifies the CA's signature on the root value, tree height and time stamp.

1. If the directory claims the queried certificate is revoked, the user checks the leaf to root path supplied by the directory by applying the hash function h.
2. If the directory claims the queried certificate is not revoked, the user checks the two paths supplied by the directory and checks that they lead to two adjacent leaves in the 2-3 tree, with values ℓ_1, ℓ_2 The user checks that $\ell_1 < s < \ell_2$.

In the above scheme, the communication costs of verifying that a certificate was not revoked may be twice the communication costs of verifying that a certificate is in the list. To overcome this, the tree may be built such that every node corresponds to two consecutive serial numbers – thus having to send only one path in either case. Since the number of bits needed for holding the value of a tree node, i.e. the hash function security parameter (ℓ_{hash} in the notation below) is more than twice the bits needed for holding a certificate serial number, this does not influence the tree size.

5 Evaluation

The CA-to-directory communication costs of our scheme are optimal (proportional to the number of changes in the revocation list), enabling high update rates. The proof supplied by the directory is of length logarithmic in the number of revoked certificates. This allows the user to hold a short transferable proof of the validity of his certificate and present it with his certificate (This proof may be efficiently updated, we will make use of this feature in the certificate update scheme of Section 6).

In the following, we compare the communication costs of CRL, CRS and our system (the communication costs of CRT are similar to ours). Basing on this analysis, we show that the proposed system is more robust to changes in parameters, and allows higher update rates than the other.

Other advantages of the proposed scheme are:

- The CA has to keep a smaller secret than in CRS.
- Since CA-to-directory communication is low, the CA may communicate with the directory using a slow communication line secured against breaking into the CA's computer (the system security is based on the ability to protect the CA's secrets).
- Since we base our scheme on a 2-3 tree, there is never a need to re-compute the entire tree to update it. This allows higher update rates than CRT.
- Another consequence of the low CA-to-directory communication is that a CA may update many directories, avoiding bottlenecks in the communication network.

5.1 Communication costs

The parameters we consider are:

- n - Estimated total number of certificates ($n = 3,000,000$).
- k - Estimated average number of certificates handled by a CA ($k = 30,000$).
- p - Estimated fraction of certificates that will be revoked prior to their expiration ($p = 0.1$). (We assume that certificates are issued for one year, thus, the number of certificates revoked daily is $\frac{n \cdot p}{365}$.)
- q - Estimated number of certificate status queries issued per day ($q = 3,000,000$).
- T - Number of updates per day ($T = 1$).
- ℓ_{sn} - Number of bits needed to hold a certificate serial number ($\ell_{sn} = 20$).
- ℓ_{stat} - Number of bits needed to hold the certificate revocation status numbers Y_{365-i} and N_0 ($\ell_{stat} = 100$).
- ℓ_{sig} - Length of signature ($\ell_{sig} = 1,000$).
- ℓ_{hash} - Security parameter for the hash function ($\ell_{hash} = 128$).

Values for $n, k, p, q, T, \ell_{sn}, \ell_{stat}$ are taken from Micali [18], ℓ_{sig} and ℓ_{hash} are specific to our scheme.

CRL costs

- The CRL daily update cost is $T \cdot n \cdot p \cdot \ell_{sn}$ since each CA sends the whole CRL to the directory in each update. An alternative update procedure where the CA sends to the directory only a difference list (which serial numbers to add/remove from the previous CRL) costs $\frac{n \cdot p \cdot \ell_{sn}}{365}$.
- The CRL daily query cost is $q \cdot p \cdot k \cdot \ell_{sn}$ since for every query the directory sends the whole CRL to the querying user.

CRS costs:

- The CRS daily update cost is $T \cdot n \cdot (\ell_{sn} + \ell_{stat})$ since for every certificate the CA sends ℓ_{stat} bits of certificate revocation status.
- The CRS daily query cost is $\ell_{stat} \cdot q$.

The proposed scheme:

- To update the directory, the CA sends difference lists of total daily length of $\frac{n \cdot p \cdot \ell_{sn}}{365} + T \cdot \ell_{sig}$.
- To answer a user's query, the directory sends up to $2 \cdot \log_2(p \cdot k)$ numbers, each ℓ_{hash} bits long, totaling $2 \cdot q \cdot \ell_{hash} \cdot \log_2(p \cdot k)$ bits.

The following table shows the estimated daily communication costs (in bits) according to the three schemes.

	CRL costs	CRS costs	Proposed scheme
Daily update (CA-directory)	$6 \cdot 10^6$	$3.6 \cdot 10^8$	$1.7 \cdot 10^4$
Daily queries (Directory-users)	$1.8 \cdot 10^{11}$	$3 \cdot 10^8$	$7 \cdot 10^9$

As shown in the table, the proposed scheme costs are lower than CRL costs both in CA-to-directory and in directory-to-users communication. The CA-to-directory costs are much lower than the corresponding CRS costs but, the directory-to-user (and thus the over all) communication costs are increased. Note that in practice, due to communication overheads, the difference between CRS and the proposed method in Directory-to-users communication may be insignificant.

5.2 Robustness to changes

Our scheme is more robust to changes in parameters than CRL and CRS. Since these are bound to change in time or due to the specific needs of different implementations, it is important to have a system that is robust to such changes.

Changes will occur mainly in the total number of certificates (n) and the update rate (T). In the proposed method, changes in n are moderated by a factor of p. Changes in T are moderated by the fact that the update communication costs are not proportional to nT but to T. Figure 1 shows how the CA-to-directory update communication costs of the three methods depend on the update rate (all other parameters are held constant). The update communication costs limit CRS to about one update a day (Another factor that limits the update rate is the amount of computation needed by a user in order to verify that a certificate was not revoked). The proposed method is much more robust, even allowing once per hour updates.

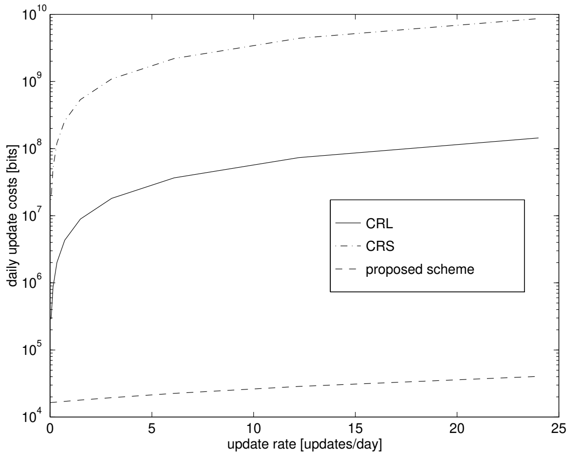

Figure 1: Daily CA-to-directory update costs vs. update rate.

6 A certificate update scheme

Some protocols avoid the need for a revocation system by using short-term certificates. (e.g. micropayments protocols when a certificate owner may cause a limited damage [13]). These certificates are issued daily and expire at the end of the day of issue. Actually, even shorter periods are desired and the main limit is due to the increase in the certification authority computation (certificates for all users have to be computed daily) and communication (certificates should be sent to their owners) short-term certificates cause.

An on-line/off-line digital signature scheme (like CRS) will reduce the computation the CA has to perform, but, it will not reduce significantly the communication costs, since the CA has to send *different* messages to *different* users, making the CA a communication bottleneck. This calls for a solution where the CA performs a simple computation (say, concerning only new users and users whose certificates are not renewed) and sends a *common* update message to *all* users. Using this message, exactly all users with non-revoked certificates should be able to prove the validity of their certificates.

We suggest a simple modification of our certificate revocation scheme that yields an efficient certificate update scheme in which the CA sends the same update message to all users. In this solution we do not assume the existence of a directory with information about all certificates, but of local directories that may hold the latest messages that were sent by the directory.

6.1 The scheme

As before, the scheme is based on a tree of revoked certificates created by the certification authority, presented in Section 4.1. Since there is no way to extract certificates from a directory, every user gets an initial certificate that may be updated using the CA's messages. Specifically, the CA augments every issued certificate with the path proving its validity, this is the only part of the certificate that is updated periodically.

To update all certificates simultaneously, the CA updates its copy of the tree, and publishes the tree paths that where changed since the previous update. Every user holding a non-revoked certificate locates the lowest node, v, on a path that coincides with his path, and updates his path by copying the new node values from v up to the root. All users holding a revoked certificate can not update their path, unless a collision is found for the hash function h.

The information sent by the CA is optimal (up to a factor of ℓ_{hash}). For r insertions/deletions since the previous update, the CA has to publish a message of length $r\ell_{hash} \log n$ bits.
Since the CA communication is reduced, one may use this update scheme for, say, updating certificates once every hour. This may cause some users to lag in updating their certificates, and the local directories should save several latest update messages, and some aggregate updates (combining update messages of a day) enabling uses that lag several days to update their certificates.

Acknowledgments

We thank Omer Reingold for many helpful discussions and for his diligent reading of the paper. We thank the anonymous referees for their helpful comments.

References

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman. "Data Structures and Algorithms". Addison-Wesley, 1983.

[2] R.G. Seidel., C.R. Aragon "Randomized Search Trees". Proc. 30th Annual IEEE Symp. on Foundations of Computer Science, pp. 540-545, 1989.

[3] M. Blum, W. Evans, P. Gemmell, S. Kannan, M. Naor. "Checking the Correctness of Memories". Algorithmica Vol.12 pp. 225-244, Springer-Verlag, 1994.

[4] M. Bellare, O. Goldreich, S. Goldwasser. "Incremental Cryptography: The Case of Hashing and Signing". Advances in Cryptology - Crypto 94. Ed. Y. Desmedt. Lecture Notes in Computer Science 839, Springer-Verlag, 1994.

[5] M. Bellare, O. Goldreich, S. Goldwasser. "Incremental Cryptography and Application to Virus Protection". Proc. 27th ACS Symp. on Theory of Computing, 1995.

[6] M. Bellare, P. Rogaway. "Collision-Resistant Hashing: Towards Making UOWHFs Practical". Advances in Cryptology - CRYPTO '97, Lecture Notes in Computer Science, Springer-Verlag, 1997.

[7] S. Brands. "An efficient off-line electronic cash system based on the representation problem". CWI Technical Report, CS-R9323, 1993.

[8] D. Chaum, E. van Heijst and B. Pfitzmann. "Cryptographically strong undeniable signatures, unconditionally secure for the signer". Advances in Cryptology - CRYPTO '91, Lecture Notes in Computer Science 576, Springer-Verlag, 1992, pp. 470-484.

[9] H. Dobbertin. "Cryptanalysis of MD4". D. Gollmannn, Ed. Fast Software Encryption, 3rd international workshop. Lecture Notes in Computer Science 1039, Springer-Verlag, pp. 53-69, 1996.

[10] H. Dobbertin. "Cryptanalysis of MD5". Rump session, Eurocrypt 1996. `http://www.iacr.org/conferences/ec96/rump/index.html`

[11] S. Even, O. Goldreich, S. Micali. "On-Line/Off-Line Digital Signatures". Journal of Cryptology, Springer-Verlag, Vol. 9 pp. 35-67, 1996.

[12] O. Goldreich, S. Goldwasser, and S. Halevi. "Collision-Free Hashing from Lattice Problems". *ECCC*, TR96-042, 1996. `http://www.eccc.uni-trier.de/eccc/`

[13] A. Herzberg, H. Yochai. "Mini-Pay: Charging per Click on the Web". Proc. 6th International World Wide Web Conference, 1997. `http://www6.nttlabs.com/`

[14] R. Impagliazzo, M. Naor. "Efficient Cryptographic Schemes Provably as Secure as Subset Sum". Journal of Cryptology, Springer-Verlag, Vol. 9 pp. 199-216, 1996.

[15] C. Kaufman, R. Perlman, M. Speciner. "Network Security. Private Communication in a Public World". Prentice Hall series in networking and distributed systems, 1995.

[16] P. Kocher. "A Quick Introduction to Certificate Revocation Trees (CRTs)". `http://www.valicert.com/company/crt.html`

[17] R. C. Merkle. "A Certified Digital Signature". Proc. Crypto '89, Lecture Notes in Computer Science 435, pp. 234-246, Springer-Verlag, 1989.

[18] S. Micali. "Efficient Certificate revocation". Technical Memo MIT/LCS/TM-542b, 1996.

[19] M. Naor, M. Yung. "Universal one-way hash functions and their cryptographic applications". Proc. 21st ACM Symp. on Theory of Computing, pp. 33-43, 1989.

[20] U.S. National Institute of Standards and Technology. "A Public Key Infrastructure for U.S. Government unclassified but Sensitive Applications". September 1995.

[21] U.S. National Institute of Standards and Technology. "Secure Hash Standard". Federal Information Processing Standards Publication 180, 1993.

[22] R. Rivest. "The MD4 message-digest algorithm". Internet RFC 1320, 1992.

[23] R. Rivest "The MD5 message-digest algorithm". Internet RFC 1321, 1992.