QuasiModo: Efficient Certificate Validation and Revocation⋆

Farid F. Elwailly[1], Craig Gentry[2], and Zulfikar Ramzan[2]

[1] Lockheed Martin
farid.f.elwailly@lmco.com
[2] DoCoMo Communications Laboratories USA, Inc.
{cgentry, ramzan}@docomolabs-usa.com

Abstract. We present two new schemes for efficient certificate revocation. Our first scheme is a direct improvement on a well-known tree-based variant of the NOVOMODO system of Micali [11]. Our second scheme is a direct improvement on a tree-based variant of a multi-certificate revocation system by Aiello, Lodha, and Ostrovsky [1]. At the core of our schemes is a novel construct termed a QuasiModo tree, which is like a Merkle tree but contains a length-2 chain at the leaves and also directly utilizes interior nodes. This concept is of independent interest, and we believe such trees will have numerous other applications. The idea, while simple, immediately provides a strict improvement in the relevant time and communication complexities over previously published schemes.

1 Introduction

As we move to an increasingly online world, public-key cryptography will be prevalent. Underlying such use we must have a public-key infrastructure (PKI) that constitutes the policy, procedures, personnel, components, and facilities for binding public keys to identities or authorizations for the purposes of offering desired security services. Typically, a PKI includes a certificate authority (CA) that not only issues binding certificates but also manages them. When issuing a certificate, the CA obviously must check that a user's credentials are accurate, but even a legitimately issued certificate may need to be revoked. Handling revocation is one of the most challenging components of certificate management.

The Certificate Revocation Problem. While a certificate's validity may be limited by an expiration date, we may sometimes wish to revoke a certificate prior to this time. For example, a key holder may change his affiliation or position, or his private key may have been compromised. This problem is both fundamental and critical – the lack of an efficient solution will hinder the widespread use of PKI. Accordingly, we need an efficient mechanism for revoking a certificate.

One common approach is a *certificate revocation list* (CRL), which is a signed and time-stamped list issued by the CA specifying which certificates have been

⋆ A very preliminary portion of this work was conducted when F. Elwailly and Z. Ramzan were at IP Dynamics, Inc.

revoked according to some identifier like a serial number. These CRLs must be distributed periodically, even if there are no changes, to prevent illegitimate reuse of stale certificates. CRLs are appealing because of their simplicity. However, their management may be unwieldy with respect to communication, search, and verification costs. An alternative approach, proposed by Kocher [7], is a Certificate Revocation Tree (CRT), which is a Merkle tree that associates each leaf with a revoked certificate. We describe Merkle trees in greater detail below.

Rather than posting full-fledged lists of revoked certificates, the CA may instead answer online queries about specific certificates. This approach is used in OCSP [13], but it has limitations. In particular, the CA must sign each response, which may be computationally infeasible given that it may have to handle numerous requests. A centralized CA creates a major scalability issue because all requests are routed through it. On the other hand, a decentralized CA may lower security since the precious signing key will be replicated on multiple servers, thereby creating multiple attack points.

The NOVOMODO Approach. Micali [10, 11, 12] addressed these problems in an elegant scheme now called NOVOMODO. His scheme works with any standard certificate format such as X.509 and allows a CA to provide validity status of a certificate at any pre-specified time interval such as a day, an hour, etc. NOVOMODO uses a hash chain together with a single digital signature. The advantage is that the cost of the single signature is amortized over many validity proofs. Unfortunately, NOVOMODO requires verification time proportional to the number of periods that have passed between two queries, assuming that the verifier caches information from previous sessions. If, however, the verifier does not cache such information, verification time is proportional to the number of intervals that have passed since the certificate's creation. Even though hash functions require much less time to compute than traditional signatures, hash chain traversal costs may be prohibitively expensive for long chains. For example, benchmark tests conducted using the Crypto++ library showed that SHA-1 is about 5000-6000 times faster than RSA-1024 signing and about 200 times faster than verification. On the other hand, SHA-1 is only 500-600 times faster than ESIGN-1023 signing and about 200 times faster than verification. See [3] for further details. This data suggests that while cryptographic hash functions are faster than signatures, long hash chains are very undesirable, especially for some of the faster signature schemes like ESIGN [16]. Therefore, a natural extension to NOVOMODO that uses Merkle trees was pointed out by Gasko et al. [4] as well as by Naor and Nissim [14]. This variant has the nice property that validity proof size is logarithmic in the total number of update periods.

Multi-Certificate Revocation. Aiello, Lodha, and Ostrovsky [1] discovered a clever extension to the NOVOMODO approach which allows the CA to provide validity status for a group of certificate owners with a single proof. The idea is to form a cover set $\mathcal{F}$ consisting of various subsets of the set of certificate owners, and construct a Merkle tree or a hash chain for each element of the cover. The cover is constructed so that for any arbitrary subset of revoked users, there are elements in the cover whose union exactly constitutes the set of non-revoked users. Then, at a given interval, instead of providing validity information for each individual certificate owner, the CA instead finds elements from $\mathcal{F}$ whose union is the set of non-revoked users. The validity proof, which consists of various Merkle tree or hash chain values, is published just for these elements.

Our contribution: The QuasiModo Approach. We propose an alternative to Merkle trees which we term QuasiModo trees. QuasiModo trees have two differences. First, their leaves are augmented with hash chains of length 2. Second, rather than starting validity proofs at the leaves, as is typically done in Merkle trees, QuasiModo trees are carefully numbered to allow proofs to start with alternate internal nodes of the tree. The idea, while simple, does not seem to have appeared previously. Yet, the result is a direct improvement in both the overall verification complexity, as well as the communication complexity, over previous tree-based schemes. Moreover, validity proofs are small enough to fit within a single packet – so the extra communication (compared to hash chains) required in practice is negligible. Table 1 summarizes the results of using QuasiModo trees as compared to Merkle trees. QuasiModo trees are of independent interest and may be used to improve other schemes involving Merkle trees. For example, they have recently been applied to the problem of secure billing in networks [5].

Organization. The next section states various preliminaries. Section 3 describes the NOVOMODO scheme and section 4 explains the QuasiModo improvement to NOVOMODO. Section 5 discusses the multi-certificate revocation extension to NOVOMODO proposed by [1] and describes how to improve it using QuasiModo trees. Finally, section 6 analyzes the performance of QuasiModo trees as compared to Merkle trees, and provides a security proof for our schemes.

2 Preliminaries

Model and Notation. We have a certificate authority $\mathcal{C}$ who issues public-key certificates, and two participants Alice $\mathcal{A}$ and Bob $\mathcal{B}$. $\mathcal{B}$ has a public key that $\mathcal{A}$ wishes to verify. We assume the existence of an open or closed PKI where both $\mathcal{C}$ and $\mathcal{B}$ have public-private key pairs. Let $(\mathsf{Sk}, \mathsf{Pk})$ denote a key pair where Sk is the private signing key for computing the signature on a message, and Pk is the public verification key corresponding to Sk. Subscripts denote which keys belong to specific individuals. So, the key pair for $\mathcal{C}$ is $(\mathsf{Pk}_{\mathcal{C}}, \mathsf{Sk}_{\mathcal{C}})$ and the key pair for $\mathcal{B}$ is $(\mathsf{Pk}_{\mathcal{B}}, \mathsf{Sk}_{\mathcal{B}})$. Let $\mathcal{DS} = (\mathsf{KG}, \mathsf{Sign}, \mathsf{Vf})$ denote a digital signature scheme that is secure against existential forgery under adaptive chosen message attack [6]. Here KG denotes the key generation algorithm, $\mathsf{Sign}(\mathsf{Sk}, M)$ denotes the signing algorithm which outputs a signature σ on message M under signing key Sk (the signing algorithm may be randomized), and $\mathsf{Vf}(\mathsf{Pk}, M, \sigma) \in \{0, 1\}$ denotes the verification algorithm which evaluates to 1 if the signature σ on message M is correct with respect to the public key Pk. We remark that KG implicitly takes as input a security parameter specifying the lengths of the keys it should generate.

Let $\{0,1\}^*$ denote the set of all bit strings. Let H denote a *cryptographic compression function* that takes as input a b-bit payload and produces a v-bit output. In our constructions $b = 2v$ which can be achieved by all well-known compression function constructions through padding. H also utilizes a v-bit initialization vector or IV which we assume is fixed and publicly known. For simplicity, we do not view the IV as an actual hash function argument, so we may not always explicitly list it as an input. A practical example of such a cryptographic compression function is SHA-1 [15] whose output and IV size is 20-bytes, and whose payload size is 64-bytes. In any practical instantiation of our schemes we will not need to operate on data larger than the compression function payload size; however there are numerous standard techniques such as iterated hashing or Merkle-trees [9] for doing so. For convenience, we use the term hash function instead of compression function, where it is understood that a hash function can take arbitrary length strings $\{0,1\}^*$ and produce a fixed length output in $\{0,1\}^v$. The symbol $\mathcal{H}$ denotes such a function. We assume cryptographic compression functions and the hash functions built on top of them are *one way and collision resistant* (i.e., finding two distinct inputs $m_1 \neq m_2$ such that $\mathcal{H}(\mathsf{IV}, m_1) = \mathcal{H}(\mathsf{IV}, m_2)$ is difficult).

For a length-preserving function $f : \{0,1\}^n \rightarrow \{0,1\}^n$ and an integer $i \geq 1$, let f^i denote its i-fold composition: $f^i(x) = f(x)$ for $i = 1$ and $f^i(x) = f(f^{i-1}(x))$ for $i > 1$. We say f is a one-way function if, given $f(x)$, where x is randomly chosen, it is hard to find a z such that $f(z) = f(x)$, except with negligible probability. We say f is one way on its iterates if for any i, given $f^i(x)$, it is hard to find a z such that $f(z) = f^i(x)$, except with negligible probability. In practice, one often constructs a candidate function that is one way on its iterates by starting with a hash function $\mathcal{H}$ and padding part of the payload to make it length preserving. Finally, for a real number r, let $\lceil r \rceil$ denote the smallest integer greater than or equal to r. Similarly, $\lfloor r \rfloor$ denotes the largest integer less than or equal to r.

Merkle Trees. We now describe *Merkle trees* [9]. Suppose that we have m values $x_1, \ldots, x_m$, each of which is in $\{0,1\}^n$. For simplicity, assume that m is a power of 2. Let $\mathcal{H} : \{0,1\}^{2n} \rightarrow \{0,1\}^n$ be a cryptographic hash function. The Merkle tree associated with $x_1, \ldots, x_m$ under hash function $\mathcal{H}$ is a balanced binary tree in which each node is associated with a specific value $\mathsf{Value}(v)$. There are m leaves, and for each leaf ℓ_i, $\mathsf{Value}(\ell_i) = x_i$, $1 \leq i \leq m$. For an interior vertex v, let $\mathsf{C}_0(v)$ and $\mathsf{C}_1(v)$ denote its left and right children. Let $\circ$ denote the concatenation operation. Then, $\mathsf{Value}(v) = \mathcal{H}(\mathsf{IV}, \mathsf{Value}(\mathsf{C}_0(v)) \circ \mathsf{Value}(\mathsf{C}_1(v)))$. Merkle trees may be used to digest data in digital signatures, where the signed digest corresponds to the value associated with the root. If the underlying compression function is collision resistant, then it is hard to find two different messages whose Merkle root value is identical [2, 8]. We will also make use of the notion of the *co-nodes* for a given vertex in a Merkle tree. For a vertex v, $\mathsf{CoNodes}(v)$ is the set of siblings of the vertices on the path from v to the root. More formally, if we let $\mathsf{Sib}(v)$ and $\mathsf{Parent}(v)$ denote v's sibling and parent respectively, then:

$$\mathsf{CoNodes}(v) = \begin{cases} \emptyset & \text{if } v \text{ is the root} \\ \{\mathsf{Sib}(v)\} \bigcup \mathsf{CoNodes}(\mathsf{Parent}(v)) & \text{otherwise.} \end{cases} \tag{1}$$

Finally, for a set of co-nodes, we abuse notation by letting $\mathsf{Value}(\mathsf{CoNodes}(v))$ denote the values associated with the co-nodes of a vertex v. The analogous notion of co-nodes exists for any arbitrary tree. Given the values of a vertex and its co-nodes, we can calculate the root value of the tree. In particular, let the value associated with a vertex be v and let the values of its co-nodes be $v_1, \ldots, v_\ell$. Then, the root value is h_ℓ where $h_1 = \mathcal{H}(v \circ v_1)$ and $h_i = \mathcal{H}([h_{i-1}, v_i])$, $2 \leq i \leq \ell$, where $[h_i, v_i]$ equals $v_i \circ h_i$ if v_i is a left child or $h_i \circ v_i$ if v_i is a right child.

3 NOVOMODO

We now describe the NOVOMODO scheme of Micali [10, 11, 12]. The scheme can be broken up into three phases: a set up phase in which the CA $\mathcal{C}$ issues a certificate to a user Bob $\mathcal{B}$, an update phase in which $\mathcal{C}$ provides an efficient proof of revocation or validity, and a verification phase where a user Alice $\mathcal{A}$ determines the status of $\mathcal{B}$'s certificate.

Set Up. Let f be a function that is one way on its iterates. Let $\mathcal{D}$ denote traditional certificate data (e.g., $\mathcal{B}$'s public key, a serial number, a string that serves as $\mathcal{B}$'s identity, an issue date, and an expiration date). Let p denote the number of periods in the certificate scheme. The CA $\mathcal{C}$ associates with the certificate data $\mathcal{D}$ two numbers y_p and N computed as follows. $\mathcal{C}$ picks values y_0 and N_0 at random from $\{0,1\}^n$. He sets $y_p = f^p(y_0)$ and $N_1 = f(N_0)$. We refer to y_p as the validity target and N_1 as the revocation target for reasons that will shortly become clear. The certificate consists of $(\langle \mathcal{D}, y_p, N_1 \rangle, \mathsf{Sign}(\mathsf{Sk}_{\mathcal{C}}, \langle \mathcal{D}, y_p, N_1 \rangle))$.

Periodic Certificate Updates. The directory is updated each period (for example, if $p = 365$, then the update interval might be daily for certificates that are valid for one year). At period i, if the certificate is valid, then $\mathcal{C}$ sends out $y_{p-i} = f^{p-i}(y_0)$. If the certificate has been revoked, $\mathcal{C}$ sends out N_0.

Verifying Certificate Status. Suppose $\mathcal{A}$ wants to verify the status of a certificate at period i. We assume $\mathcal{A}$ performs the standard checks; e.g., the certificate has not expired and $\mathcal{C}$'s signature on the certificate is valid. Now, if $\mathcal{C}$ claims the certificate has been revoked, then $\mathcal{A}$ takes the value N_0 sent by $\mathcal{C}$ and checks if $N_1 = f(N_0)$. Note that she knows N_1 since it is in the certificate. Similarly, if $\mathcal{C}$ claims the certificate has not been revoked, then $\mathcal{A}$ takes the value y_{p-i} sent by $\mathcal{C}$ and checks if $f^i(y_{p-i}) = y_p$. Again, note that $\mathcal{A}$ knows y_p.

NOVOMODO with Merkle Trees. One undesirable property of NOVOMODO is that the verification time is linear in the size of the interval between consecutive validity checks made by $\mathcal{A}$ assuming $\mathcal{A}$ always caches responses from previous queries. For example, if the update period is every 3 hours and certificates are valid for a year, then $\mathcal{A}$ may have to make up to several thousand hash function calls when verifying a certificate. To address this concern, the following use of Merkle trees in NOVOMODO has been suggested [1, 4, 14]. The CA $\mathcal{C}$ creates a Merkle tree with $2p$ leaves $\ell_1, \ldots, \ell_{2p}$, each of which is assigned a secret pseudorandom value, and signs the root.[3] The leaves are numbered left to right from 1 to $2p$, and at time period i, if the certificate is valid, $\mathcal{C}$ sends out $\mathsf{Value}(\ell_{2i})$ and $\mathsf{Value}(\mathsf{CoNodes}(\ell_{2i}))$.

4 QuasiModo Trees for Single Certificate Revocation

Having described NOVOMODO, we describe our QuasiModo approach. At a high level, QuasiModo replaces the NOVOMODO Merkle trees with QuasiModo trees. These trees yield a performance improvement over using Merkle trees.

QuasiModo Trees. QuasiModo trees bear some similarity to the Merkle trees used in NOVOMODO, except that we first append length-2 hash chains to the bottom of the tree, and we next carefully number every other interior vertex so they can be efficiently used directly in validation proofs. The power of using such trees is that a subset of the *internal* nodes can be directly utilized in the certificate revocation scheme and we do not always have to use the leaves as is done in the normal Merkle case. The upshot is a sizeable improvement in both the verification complexity and communication complexity.

We start with m randomly chosen values, $x_1, \ldots, x_m$; note that these values can be pseudorandomly generated from a single sufficiently large random seed. For simplicity, suppose that $m = 2^k$ for some integer $k > 0$. We set up a tree as follows. The bottom layer has m vertices which are *only-children* (i.e., they have no siblings). Next, we construct a balanced binary tree of depth $k + 1$ which resides on top of the bottom-level m vertices. We assign values to each of the vertices as follows. The bottom-level m vertices take on the values $x_1, \ldots, x_m$ respectively. For the layer that is directly on top of the bottom layer, we assign the n-bit value $f(x_i)$ to the i^{th} such vertex, where $f : \{0,1\}^n \to \{0,1\}^n$ is a one-way function. That is, if ℓ'_i is such a vertex, then $\mathsf{Value}(\ell'_i) = f(x_i)$, for $1 \le i \le m$. For any interior node v that is above the bottom two layers $\mathsf{Value}(v) = \mathcal{H}(\mathsf{IV}, \mathsf{Value}(\mathsf{C}_0(v)) \circ \mathsf{Value}(\mathsf{C}_1(v)))$. In practice, we would typically construct f by appropriately padding $\mathcal{H}$; so, from now on, we only refer to $\mathcal{H}$.

Another way to precisely characterize the same tree is as follows. There are $3m - 1$ vertices. These are respectively: $\ell_1, \ldots, \ell_m$, $\ell'_1, \ldots, \ell'_m$, and $v_1, \ldots, v_{m-1}$. The values are assigned as follows. $\mathsf{Value}(\ell_i) = x_i$ and $\mathsf{Value}(\ell'_i) = f(x_i)$, for $1 \le i \le m$. Next, let $\lambda(i) = 2(i - m/2) + 1$ and let $\rho(i) = 2(i - m/2) + 2$. For $i \in \{m/2, m/2+1, \ldots, m-1\}$, we have $\mathsf{Value}(v_i) = \mathcal{H}(\mathsf{Value}(\ell'_{\lambda(i)}) \circ \mathsf{Value}(\ell'_{\rho(i)}))$. Finally, for $i \in \{1, \ldots, m/2-1\}$, we have $\mathsf{Value}(v_i) = \mathcal{H}(\mathsf{Value}(v_{2i}) \circ \mathsf{Value}(v_{2i+1}))$. This constitutes the assignment of values to the vertices. Now, we describe the

[3] Though it does not seem to have been observed previously in [1, 4, 14], the values ℓ_{2i}, $1 \le i \le p$, can be made public without compromising security of the scheme.

directed edges. There is a directed edge from ℓ_i to ℓ'_i for $1 \leq i \leq m$. For $i \in \{m/2, m/2+1, \ldots, m-1\}$, we have a directed edge from $\ell'_{\lambda(i)}$ to v_i and a directed edge from $\ell'_{\rho(i)}$ to v_i. Finally, for $i \in \{1, \ldots, m/2-1\}$, we have a directed edge from v_{2i} to v_i, and a directed edge from v_{2i+1} to v_i. At a high level, we put a directed edge from a vertex u to a vertex w if $\mathsf{Value}(u)$ was explicitly used to calculate $\mathsf{Value}(w)$.

Next, we apply the following two-coloring to the nodes in the tree. If a vertex is a left child or has no siblings (as in the case of the ℓ_i vertices), we color it grey. All other vertices, including the root, are colored white. Finally, the grey nodes are numbered breadth first (but where the edge directions are ignored). That is, we start at the top of the tree, and work our way down to each consecutive level, numbering each grey node sequentially from left to right. At first this idea of numbering the grey vertices may seem somewhat unnatural, but it turns out to be convenient since the i^{th} grey vertex value is involved in the validation proof at period i. We refer to the i^{th} grey vertex by $\mathsf{gv}(i)$. Figure 1 illustrates a QuasiModo tree that can accommodate a revocation scheme with 7 periods and a Merkle tree that accommodates 8 periods.

In general, a QuasiModo tree accommodating $p = 2^k - 1$ periods requires $\frac{3p+1}{2}$ vertices. A Merkle tree accommodating $p = 2^k$ periods requires $4p - 1$ vertices. A QuasiModo tree is thus approximately $\frac{8}{3} - \frac{14}{9p+3}$ times smaller than the corresponding Merkle tree. Note that we may naturally extend the notion of a QuasiModo tree to an ℓ-chained QuasiModo tree in which each internal vertex is replaced with a hash chain of length ℓ. This extension provides a middle ground between the tradeoffs achieved from QuasiModo trees and regular hash chains.

Set Up. As in NOVOMODO, let $\mathcal{D}$ denote traditional certificate data. The CA $\mathcal{C}$ associates with the certificate data $\mathcal{D}$ two numbers y_r and N_1 computed as follows. $\mathcal{C}$ constructs a QuasiModo tree and sets y_r to be value assigned to the root of that tree. He sets $N_1 = f(N_0)$ like he did for NOVOMODO. The certificate consists of $(\langle \mathcal{D}, y_r, N_1 \rangle, \mathsf{Sign}(\mathsf{Sk}_{\mathcal{C}}, \langle \mathcal{D}, y_r, N_1 \rangle))$.

Periodic Certificate Updates. The directory is updated each period. At period i, if the certificate is valid, $\mathcal{C}$ sends out $\langle \mathsf{Value}(\mathsf{gv}(i)), \mathsf{Value}(\mathsf{CoNodes}(\mathsf{gv}(i))) \rangle$. If the certificate has been revoked, he sends out N_0. Note that if $\mathcal{A}$ received conode values from previous validity checks, it is not necessary for $\mathcal{C}$ to send every value in $\mathsf{Value}(\mathsf{CoNodes}(\mathsf{gv}(i)))$.

Verifying Certificate Status. Suppose that $\mathcal{A}$ wants to verify the status of a certificate at period i. We assume she first performs all the standard checks; e.g., the certificate has not expired and $\mathcal{C}$'s signature is correct. Now, if $\mathcal{C}$ claims the certificate has been revoked, then $\mathcal{A}$ takes the value N_0 sent by $\mathcal{C}$ and checks if indeed $N_1 = f(N_0)$. If $\mathcal{C}$ claims the certificate has not been revoked, then $\mathcal{A}$ takes the values $\mathsf{Value}(\mathsf{gv}(i))$ and $\mathsf{Value}(\mathsf{CoNodes}(\mathsf{gv}(i)))$ uses them to compute the QuasiModo tree root. Note that this step requires at most $\lfloor \log_2 i \rfloor + 1$ hash computations for QuasiModo trees as opposed to $\lceil \log_2 p \rceil + 1$ for Merkle trees. If the computed root matches the value y_r, then the certificate is valid. Alternatively, if $\mathcal{A}$ has already verified a certificate for a previous period j (and has

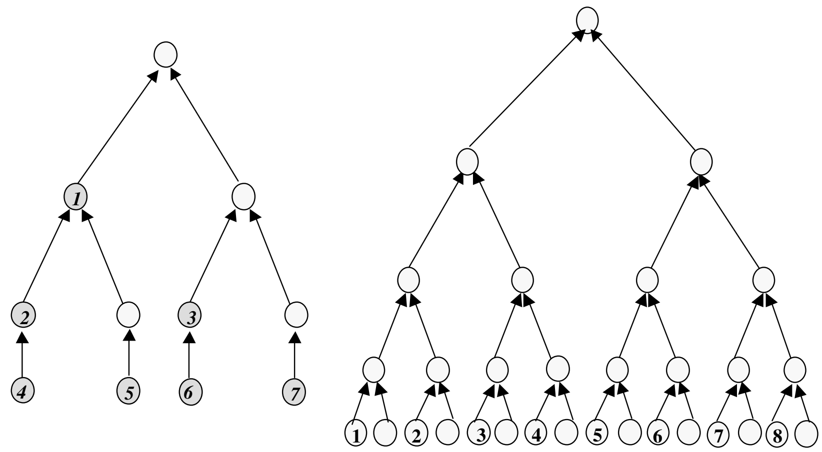

Fig. 1. On the left we have an 11-vertex QuasiModo tree, which can be used for 7 periods; the value of each interior node is the hash of the concatenation of the values of its children. Every grey vertex is numbered sequentially top-down left-to-right. On the right, we have a 31-vertex Merkle tree, which can be used for 8 periods. By using interior nodes and a hash chain at the end, we can get a more compact tree – resulting in shorter proofs, shorter verification time and lower communication complexity.

stored the proof), and some of the vertex values associated with period i are in a subtree rooted at a vertex associated with the certificate for period j, then $\mathcal{A}$ only needs to use the co-nodes to compute up to that subtree root.

5 QuasiModo Trees for Multi-Certificate Revocation

We now propose the use of QuasiModo trees to improve a scheme of Aiello, Lodha, and Ostrovsky (ALO) [1]. We first describe the generalized scheme, and then give examples of how to instantiate it. To describe the scheme, we must consider the notion of a *complement cover family*. Let U denote the universe; in our setting, it will be the set of all certificate holders (regardless of whether the certificate has been prematurely revoked). Let $R \subseteq U$; in our setting, R will denote the set of certificate holders whose certificates have been revoked prior to expiration. Let $\bar{R} = U - R$. That is, $\bar{R}$ will be the set of certificate holders whose certificates are currently not revoked. Now, let $\mathcal{S}$ be a set whose elements are subsets of U. We say that $\mathcal{S}$ is a complement cover of R if $\bigcup_{W \in \mathcal{S}} = \bar{R}$. We can extend this notion to the universe as follows. Let $\mathcal{F}$ be a set whose elements are subsets of U. We say that $\mathcal{F}$ is a complement cover family of U if and only if, for every subset R of U, $\mathcal{F}$ contains a complement cover of R. That is, for every subset R of U, there is a subset $\mathcal{S}$ of $\mathcal{F}$ such that $\mathcal{S}$ is a complement cover of R. The set of all singletons is a simple example of a complement cover family. That is, $\mathcal{F} = \{\{u_1\}, \ldots, \{u_N\}\}$ where $U = \{u_1, \ldots, u_N\}$. Indeed, it is very easy to see that the singleton cover must be contained in any complement cover family for the universe U. At another extreme, the power set, or set of all subsets of a set, is also trivially seen to be a complement cover family.

At a high level in the ALO [1] scheme, the CA first constructs a complement cover family for the universe of certificate holders. Next, he assigns a Merkle tree to each element of the complement cover family. For a given certificate owner $\mathcal{B}$, let $\mathcal{F}(\mathcal{B})$ denote the set of elements of $\mathcal{F}$ to which the user belongs. The validation targets the CA incorporates, in its user certificate, are the roots of the Merkle trees corresponding to the elements of $\mathcal{F}(\mathcal{B})$. Now, to provide a validation proof at period i for a group of users, the CA first determines the set of revoked users R. Then, he computes the complement cover of R contained in $\mathcal{F}$ – call it $\mathcal{S}$. Note that such a complement cover $\mathcal{S}$ exists since $\mathcal{F}$ is a complement cover family for the universe U. The CA produces the i^{th} leaf and its co-nodes in the associated Merkle tree for each element of $\mathcal{S}$. To check the validity of $\mathcal{B}$'s certificate in period i, a verifier $\mathcal{A}$ checks that the CA has revealed the i^{th} leaf for at least one element of $\mathcal{S}$ in $\mathcal{F}(\mathcal{B})$. We can replace these Merkle trees with QuasiModo trees, and we now describe how to do so.

Set Up. Let U denote the universe of all certificate holders. Then the CA $\mathcal{C}$ constructs a complement cover family $\mathcal{F}$. Let p denote the number of periods. For each element of $\mathcal{F}$, the CA $\mathcal{C}$ constructs an independent QuasiModo tree that allows for p periods. We let $\mathcal{D}$ denote traditional certificate data. The CA $\mathcal{C}$ associates with the certificate data $\mathcal{D}$ a set of validation targets and a single revocation target as follows. $\mathcal{C}$ picks a value N_0 at random from $\{0,1\}^n$. He sets $N_1 = f(N_0)$ – where N_1 represents the revocation target. $\mathcal{C}$ constructs a set of validity targets for the certificate owner $\mathcal{B}$ as follows. He computes $\mathcal{F}(\mathcal{B})$, which is the set consisting of elements of $\mathcal{F}$ for which $\mathcal{B}$ is a member. Suppose that there are κ elements of $\mathcal{F}(\mathcal{B})$ – call them $\mathcal{F}_1, \ldots, \mathcal{F}_\kappa$. Let $r_1, \ldots, r_\kappa$ denote the values of the roots of the QuasiModo trees associated with $\mathcal{F}_1, \ldots, \mathcal{F}_\kappa$. The certificate consists of $(\langle \mathcal{D}, r_1, \ldots, r_\kappa, N_1\rangle,\ \mathsf{Sign}(\mathsf{Sk}_\mathcal{C}, \langle \mathcal{D}, r_1, \ldots, r_\kappa, N_1\rangle))$. We remark that for specific complement cover constructions, one can reduce the number of root values r_i that are included in the augmented certificate data.

Periodic Certificate Updates. The directory is updated each period. At period i, if a given certificate is revoked, then $\mathcal{C}$ sends out the pre-image of the revocation target (i.e., the value N_0 value associated with each certificate); if the certificate is valid, then $\mathcal{C}$ does the following. It first determines the set R of revoked holders. It computes the element $\mathcal{S} \in \mathcal{F}$ such that $\mathcal{S}$ is a complement cover for R. For each element of $\mathcal{S}$, $\mathcal{C}$ sends out the value $\mathsf{Value}(\mathsf{gv}(i))$ associated with the tree corresponding to that element, together with $\mathsf{Value}(\mathsf{CoNodes}(\mathsf{gv}(i)))$.

Verifying Certificate Status. Suppose that $\mathcal{A}$ wants to check the status of $\mathcal{B}$'s certificate at period i. She first checks the expiration date and that the signature by the CA $\mathcal{C}$ is valid. If $\mathcal{C}$ claims the certificate has been revoked, then $\mathcal{A}$ takes the value N_0 sent by $\mathcal{C}$ and checks if indeed $N_1 = f(N_0)$. If $\mathcal{C}$ claims the certificate has not been revoked, then $\mathcal{A}$ takes the values $\mathsf{Value}(\mathsf{gv}(i))$ and $\mathsf{Value}(\mathsf{CoNodes}(\mathsf{gv}(i)))$ associated with the element of the complement cover that is in $\mathcal{F}(\mathcal{B})$. $\mathcal{A}$ computes the QuasiModo tree root value. If the computed root value matches one contained in the certificate, then the certificate is valid. Alternatively, if $\mathcal{A}$ has already verified a certificate for a previous period j (and has stored the relevant verification information), and a vertex associated with the proof in period i is in a subtree rooted at a vertex associated with the certificate for period j, then $\mathcal{A}$ only needs to use the co-nodes to compute up to that subtree root.

Binary Tree Hierarchy. For completeness, we review a specific complement cover family construction known as the binary tree hierarchy. Assume, for simplicity, that the number of certificate holders is 2^k for some integer $k \geq 0$. We create a binary tree with 2^k leaves and assign to every vertex a subset of the universe of certificate holders. At each leaf, we assign the singleton set corresponding to a single certificate holder. At each internal node, we assign the subset corresponding to the union of the subsets of the nodes of its children. The complement cover family $\mathcal{F}$ consists of the sets assigned to all the vertices. It is clear that $\mathcal{F}$ forms a complement cover family; the following steps yield a minimal-size complement cover of any subset $R \subseteq U$:

1. "Mark" every leaf vertex corresponding to an element of $\bar{R}$;
2. "Mark" every interior vertex on the path from the marked leaf to the root;
3. Determine the non-marked vertices whose parents are marked;
4. Consider the subsets associated with these vertices.

6 Performance and Security Analysis

Our QuasiModo single-certificate and multi-certificate revocation systems are quite efficient in terms of both computation and communication. We compare the performance to their Merkle tree analogues. Our analysis applies to both single-certificate revocation as in NOVOMODO and multi-certificate revocation as in ALO [1]. Table 1 summarizes the results.

Complexity Without Caching. Suppose we have p periods where $p = 2^k - 1$ for some integer $k > 0$. To refresh a certificate at period p_t, $\mathcal{C}$ sends $\mathsf{Value}(\mathsf{gv}(p_t))$ and $\mathsf{Value}(\mathsf{CoNodes}(\mathsf{gv}(p_t)))$. The number of co-nodes to be sent is equal to the depth of this vertex which is $\lfloor \log_2(p_t) \rfloor + 1$. Therefore, the total proof size is $\lfloor \log_2(p_t) \rfloor + 2$ since we need to send the value at vertex p_t itself as part of the proof. To verify, the receiver computes at most $\log_2(p_t) + 1$ hashes, assuming he has not cached any previous values; if he has saved some information from a previous period, then the number of hashes is smaller. In particular, if the verifier caches the value of a vertex at level L of the QuasiModo tree on the path from grey vertex p_t to the root, then he need only compute $\lfloor \log p_t \rfloor - L$ hashes.

For the tree-based version of NovoModo suggested by [4, 14], there are $2p$ leaves, and hence a binary tree of depth $\log_2 p + 1$. However, since this scheme only uses the leaves, the proof size at period p_t is *always* $\lceil \log_2 p \rceil + 2$ and the number of hashes to verify the proof is always $\lceil \log_2 p \rceil + 1$. However, since $p_t \leq p$, we have that $\lfloor \log_2 p_t \rfloor \leq \lceil \log_2 p \rceil$. So, the QuasiModo scheme provides a strict improvement. Not only are fewer hash function computations required, but also fewer cache look-ups are required to retrieve proof vertex values.

Complexity With Caching. We compare the bandwidth consumption of QuasiModo tree schemes with Merkle tree schemes assuming that the verifier checks the certificate status at each update period and caches all received results.[4] For $p = 2^k - 1$ periods the corresponding QuasiModo tree has $\frac{3p+1}{2}$ vertices; so the total number of proof node values transmitted is $\frac{3p-1}{2}$ since the root is not counted. For p transactions, the amortized proof size is $\frac{3}{2} - \frac{1}{2p}$ hash values per transaction, and assuming caching $\mathcal{C}$ always sends exactly 2 values for non-leaf vertices and 1 value for leaf vertices. For a Merkle-tree with $p = 2^k$ periods, there are $4p - 1$ vertices ($2p$ leaves and $2p - 1$ internal nodes). Again, ignoring the root value, the total number of proof node values transmitted is $4p - 2$. Thus, the amortized proof size of p transactions is $4 - \frac{2}{p}$. Therefore, the improvement factor is $\frac{8}{3} - \frac{4}{9p-3}$ which approaches $2\frac{2}{3}$ as p gets large. In practice, however, the effects may be more pronounced since the proof sizes in the Merkle setting will vary with each iteration – going up to $\lceil \log_2 p \rceil + 1$ hash values – whereas for QuasiModo trees the size will always be one or two hash values. This variance exhibited by Merkle trees may create performance issues.

We now compare the time complexity of verifying QuasiModo proofs versus Merkle-tree proofs. For p periods, the amortized proof size in a QuasiModo tree is $\frac{3}{2} - \frac{1}{2p}$, and we only require p total calls to a cryptographic compression function for verification at each step assuming that these values fit in the compression function payload, which is the case for practical examples such as SHA-1 [15]. For a Merkle tree the total number of compression-function calls during proof verification is equal to the number of internal (non-leaf) vertices since each internal vertex results from a single compression function call applied to the concatenation of the values associated with its children. Therefore, the number of total compression function calls is $2p - 1$. Consequently, the improvement factor from using QuasiModo trees is $2 - \frac{1}{p}$ which approaches 2 as p gets large.

A potential drawback of the QuasiModo approach is that achieving constant-time verification requires the verifier to cache many of the values it receives. In the worst case, for a QuasiModo tree with p periods, the verifier may have to cache up to $\frac{p+1}{2}$ vertex values (corresponding to the values of the vertices one level from the bottom). This might not be a problem for reasonable parameter values. For example, suppose that a given verifier deals with 100 certificates concurrently, each of which permits 1023 periods (approximately a six-month certificate with update periods every four hours). Then, in the worse case, he needs to keep track of $(100 \cdot \frac{1023+1}{2})$ hash values, which requires under a megabyte of storage assuming we use the SHA-1 hash function with a full 20-byte tag.

[4] In practice there are likely to be many gaps in certificate status checks, but we examine this always-check always-cache case since it lends itself to a cleaner analysis. This portion of the analysis does not apply to our multi-certificate revocation scheme because there may always be gaps. Note, however, that our tree-based constructions are especially advantageous when there are gaps between checks.

Hash Chains Versus Hash Trees. In a chain-based approach the computation cost may be high since it is linear in the gap size between two verification steps. Trees reduce this to a logarithmic cost. Of course, we make the very reasonable assumption that roughly $\mathcal{O}(\log p)$ processor cache look-ups require less time than $\mathcal{O}(p)$ cryptographic hash function computations. Alternatively, QuasiModo proofs may potentially be short enough to be loaded directly into data registers when reading the incoming proof packet from the CA $\mathcal{C}$. However, one ostensible reason to prefer hash chains is that the proof size is smaller – involving the transmission of just a single hash function value. While the communication requirements of chains, in theory, are smaller, this may not translate into an actual performance improvement in practice since transmission time is typically proportional to the number of packets sent (assuming that they are reasonably sized) rather than the number of bits sent. The average TCP packet, for example, holds a payload on the order of 536 bytes (after removing 20-bytes each for the TCP and IP packet headers) and TCP packet sizes up to approximately 1500 bytes (the maximum ethernet packet size) are reasonable – especially if we perform path maximum transmission unit detection to prevent fragmentation. With packet sizes that are much larger than 20 bytes, we may find room for a few extra hash values without requiring the transmission of any extra packets. In particular we can fit 26 hash values (resp. 70+ hash values) in an *average sized* (resp. *larger sized*) TCP packet with room to spare. These values would permit over 16 *million* (resp. 256 *quintillion* $= 256 \times 10^{18}$) intervals – far more than we may ever require in any practical application. So, in all practical instances, QuasiModo proofs, like NovoModo proofs, would fit into a single packet. Yet, QuasiModo proofs take far less time to verify.

Metric	**QuasiModo trees**	**Merkle trees**
Tree Size	$\frac{3p+1}{2}$	$4p-1$
Proof Size (NC)	$\lfloor \log_2(p-r) \rfloor + 2$	$\lceil \log_2 p \rceil + 2$
Verification Time (NC)	$\lfloor \log_2(p-r) \rfloor + 1$	$\lceil \log_2 p \rceil + 1$
Amortized Proof Size (C)	$\frac{3}{2} - \frac{1}{2p}$	$4 - \frac{2}{p}$
Amortized Verification Time (C)	1	$2 - \frac{1}{p}$
Max. Proof Size (C)	2	$\lceil \log_2 p \rceil + 2$
Max. Proof Verification Time (C)	1	$\lceil \log_2 p \rceil + 1$
Min. Proof Size (C)	1	2
Min. Proof Verification Time (C)	1	1

Table 1. Comparing QuasiModo trees to Merkle trees for p periods. Here r denotes the number of periods remaining. Sizes are measured with respect to hash function output size (e.g., 20-bytes). Running times are measured in terms of the number of hash computations. Here *(C)* denotes that the verifier performs validation checks at each interval and caches all values it receives from the CA. We use *(NC)* when the verifier does not cache at all, but does check at each interval.

Security Analysis. Since a QuasiModo tree is essentially a type of hash tree, it is very straightforward to see the security of our scheme. For completeness, however, we sketch the proof of the following security theorem.

Theorem 1. *Assuming that $\mathcal{H}$ is a one-way collision-resistant hash function and that $\mathcal{DS}$ is a secure signature scheme, neither a proof of revocation nor a proof of validity can be forged.*

Proof. (Sketch) We first consider the slightly more involved case of the validity proof. First observe that assuming the security of $\mathcal{DS}$, no adversary can forge the certificate, except with negligible probability. Therefore, an adversary must use an existing certificate and come up with proof of validity that hashes to at least one validity target. Suppose that t update periods have already passed, and an adversary is trying to forge a validity proof for update period $t+\Delta$. Denote the adversary's spurious validity proof by $\mathsf{Value}'(\mathsf{gv}(t+\Delta)), \mathsf{Value}'(\mathsf{CoNodes}(\mathsf{gv}(t+\Delta)))$, where $\mathsf{Value}'(\mathsf{gv}(t+\Delta))$, and $\mathsf{Value}'(\mathsf{CoNodes}(\mathsf{gv}(t+\Delta)))$ denote spurious values for the co-nodes in the CA's QuasiModo tree. Let r denote the root of the tree, which is already known to a verifier since it is part of the certificate. For a verifier to accept the proof, the spurious values must hash to r. For notational simplicity, let $v = \mathsf{Value}'(\mathsf{gv}(t+\Delta))$ and let $r'_1, \ldots r'_\ell$ denote the values of the co-nodes ordered along the siblings of the vertices on the path from the vertex to the root. First note that if ℓ is greater than the depth of the original tree, then the expiration period would be reached (it would also imply that the adversary inverted $\mathcal{H}$ at a random point, which we assume to be infeasible, except with negligible probability). So, let us suppose ℓ is bounded by the depth of the original tree. Now, let $r_1, \ldots, r_\ell$ denote the *actual values* corresponding to what the CA generated in the actual QuasiModo tree. If for all $i \in \{1, \ldots, \ell\}$, it holds that $r_i = r'_i$, then it follows that the adversary correctly computed a pre-image of $\mathcal{H}$ since the CA never revealed all the r_i. This event only happens with negligible probability since $\mathcal{H}$ is a one-way collision-resistant cryptographic hash function.

So, suppose that the r_i and r'_i are not all equal; we show how to construct a hash function collision. Because the r'_i verifiably hash to the root, it follows that the root value can be calculated as h'_ℓ where $h'_1 = \mathcal{H}(\mathsf{Value}'(\mathsf{gv}(t+\Delta))) \circ r'_1)$, and $h'_i = \mathcal{H}([h'_{i-1}, r'_i])$ for $i \in \{1, \ldots, \ell\}$. Likewise, the same root value can be calculated as h_ℓ where $h_1 = \mathcal{H}(\mathsf{Value}(\mathsf{gv}(t+\Delta))) \circ r_1)$ and $h_i = \mathcal{H}([h_{i-1}, r_i])$. Because both calculations yield the same committed root value, it follows that $h_\ell = \mathsf{Value}(r) = h'_\ell$. Now since the r_i and r'_i are distinct, but $h_\ell = h'_\ell$, there must be some index $j \in \{1, \ldots, \ell\}$ for which $h'_j = h_j$, but $(h_{j-1}, r_j) \neq (h'_{j-1}, r'_j)$. In that case, $h'_j = \mathcal{H}([h'_{j-1}, r'_j]) = \mathcal{H}([h_{j-1}, r_j]) = h_j$, which is a collision since the inputs to $\mathcal{H}$ are distinct. We have therefore violated the collision-resistance property of $\mathcal{H}$, which can only happen with negligible probability.

We now consider the revocation target. A forgery yields a pre-image of the revocation target. Since the CA constructed the revocation target by applying $\mathcal{H}$ to a random value, that means the adversary can invert $\mathcal{H}$ at a random point, which happens with negligible probability by the one-wayness of $\mathcal{H}$.

References

[1] W. Aiello, S. Lodha, and R. Ostrovsky. Fast Digital Identity Revocation. In *Proc. of CRYPTO '98*.

[2] I. Damgård. A Design Principle for Hash Functions. In *Proc. of CRYPTO '89*.

[3] W. Dei. Crypto++ library v5.1.

[4] I. Gassko, P. S. Gemmell, and P. MacKenzie. Efficient and Fresh Certification. In *Proc. of PKC 2000*.

[5] C. Gentry and Z. Ramzan. Microcredits for Verifiable Foreign Service Provider Metering. In *Proc. of Financial Cryptography 2004 (to Appear)*.

[6] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[7] P. Kocher. On Certificate Revocation and Validation. In *Proc. of Financial Cryptography '98*.

[8] R. Merkle. One-way Hash Functions and DES. In *Proc. of CRYPTO '89*.

[9] R. Merkle. Protocols for Public-Key Cryptography. In *Proc. of IEEE Symposium on Security and Privacy '80*.

[10] S. Micali. Efficient Certificate Revocation. In *Proc. of RSA Data Security Conference '97*.

[11] S. Micali. NOVOMODO: Scalable Certificate Validation and Simplified PKI Management. In *Proc. of PKI Research Workshop '02*.

[12] S. Micali. Efficient Certificate Revocation. LCS/TM 542b, Massachusetts Institute of Technology, 1996.

[13] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. In *Internet RFC 2560*, June.

[14] M. Naor and K. Nissim. Certificate Revocation and Certificate Update. In *Proc. of USENIX Security '98*.

[15] National Institute of Standards. FIPS 180-1: Secure Hash Standard. 1995.

[16] T. Okamoto, E. Fujisaki, and H. Morita. TSH-ESIGN: Efficient Digital Signature Scheme Using Trisection Size Hash. *Contribution to IEEE P1363 '98*.

A Acknowledgements

We thank Alejandro Hevia, Ravi Jain and Toshiro Kawahara for helpful discussions and feedback on earlier manuscript drafts.