MTR 98B0000093

MITRE TECHNICAL REPORT

A Comparison of Certificate Validation Methods for Use in a Web Environment

November, 1998

Shimshon Berkovits
Jonathan C. Herzog

MITRE
Center for Integrated Intelligence Systems
Bedford, Massachusetts

A Comparison of Certificate Validation Methods for Use in a Web Environment

Shimshon Berkovits, G022
Jonathan C. Herzog, G021

Abstract

This paper reports on an investigation into several differing certificate validation techniques. It compares their mechanisms and, more importantly, their performance. An analysis is presented showing the largest number of validator clients that each validation method can support. This is done in each of five differing scenarios, scenarios that are encountered in a web environment. The affect of caching validation information is also examined. Finally, a new, hybrid method for disseminating certificate validity information is presented and recommendations are made for which method use in varying circumstances.

Table of Contents

Table of Figures

Page

Table of Tables

Page

Introduction

1.1 Trust in Public Key Certificates

From its inception, public key (asymmetric) cryptography has faced a serious and fundamental problem. How does one insure that a specific public key is trustworthy? This question breaks down into several component concerns. How does one know that the person who is supposed to have control of the corresponding private key actually does have control? How can one be sure that no one else has discovered or stolen that private key? If the possession of a particular private key implies that the subject has certain affiliations or has some specific authority, how can we know that the affiliation was still valid or that the authority still stood at the time the key was used to create a signature?

The generally accepted approach to the trustworthiness of public keys has been the creation of a public key infrastructure (PKI). The central element of a PKI is a set of certification authorities (CAs). A CA certifies that a given public key is associated with a certain subject who possesses the corresponding private key. The CA may also certify that the subject has some particular affiliation or some specific authority. All this information is tied together in a certificate that the CA signs and makes available to the public. Now the question of trusting a public key becomes a question of trusting the CA that certified the binding of the public key to a subject and, perhaps, to an affiliation or to some authority.

There are several possible reasons for trusting a specific CA and its signature on a certificate.

- The CA is personally known to and trusted by the person seeking to trust a particular public key.
- Failing that, the CA is well known and its public key is well published.
- Another CA, whose key in turn is certified by yet another CA, certifies the CA's public key. The chain of CAs leads back to a CA that is personally known and trusted or that is well known with a well-published public key.

Thus, there may be a chain of CAs from the CA that certified the public key in question to a CA that the person wishing to establish trust in a key already knows and trusts. It is assumed that there is no reason that trust not be transitive, i.e., we assume that any CA trusted by a CA that is already trusted will itself be trusted. If, for any reason, this assumption is incorrect, it is unwise to rely on the public key in question.

1.2 Certificate Revocation and Validation

Trusting the CA that issued a certificate is, however, insufficient. At the time the certificate is issued, the CA carefully establishes the identity of the key holder and verifies that the key holder also posses the corresponding private key, the claimed affiliation, and the specific authority. It is not obvious that all or even any of these conditions, that held at the time the CA certified the public key, still hold at the time the key in the certificate or the corresponding private key is being used. Perhaps the corresponding private key has been lost, stolen or otherwise compromised. Maybe the subject of the certificate has left the organization that vouched for her identity and on whose behalf the CA issued the certificate. It is also possible that the authority that the subject had at the time of

certification has since been changed or rescinded completely. If any of these circumstances apply to a specific certificate, that certificate must be revoked, preferably by the CA that issued it.

Subjects wishing to rely on the information contained in a certificate must be able to establish the continuing validity of that certificate. The simplest way to accomplish this is to query the CA each time a client wishes to trust the certificate. However, CAs are assumed to be busy machines and unable to handle the additional load of responding to many validation queries. Hence, it is common for CAs to delegate the responsibility of responding to validation queries to other machines, which we call Repositories.[1] Periodically, the CA sends to each Repository some data structure that contains the revocation status of each unexpired certificate.[2] The Repository then uses this information to construct revocation status responses. Each query response may be the same data structure as the updates themselves or it may be constructed from them. Figure 1 diagrams this flow of information.

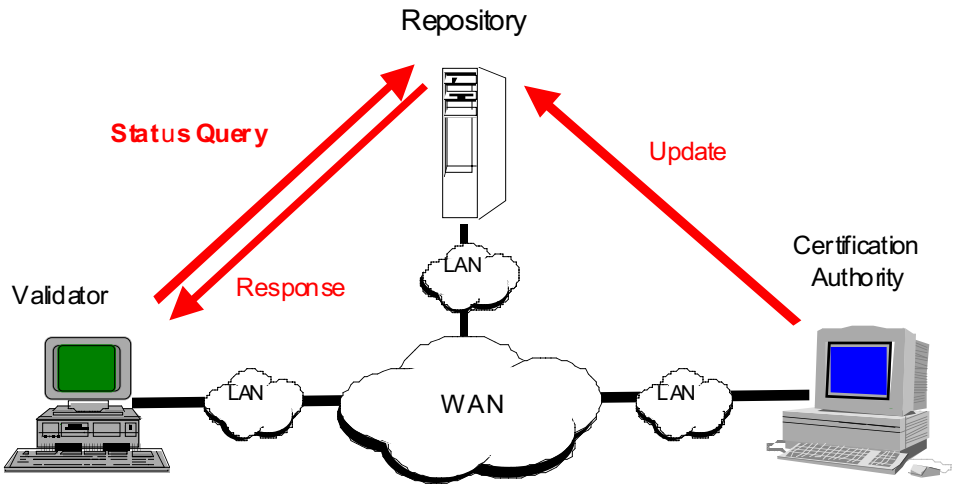

Figure 1: Validation Update, Query, and Response

In the most widely used update method, the CA maintains a list of all unexpired but revoked certificates. Such a list is known as a certificate revocation list (CRL). The CA makes the

---

[1] For historical reasons, these machines are also called "Directories." In some cases, these machines are more active than the term "Directory" would imply. Thus, our choice of the name "Repository."

[2] Certificates contain a great deal of information in addition to the binding between public key and key owner. They also contain the name of the signer, a serial number unique to that certificate, the date of certificate creation, the lifetime of the certificate, and other information. Certificates are to be trusted only during their lifetime. Expired certificates will be rejected as untrusted. Hence, the only revoked certificates of interest for the purposes of revocation are unexpired ones. From this point on, we will limit our discussion to unexpired certificates.

information on its CRL available to any subject that wishes to check the validity of a given certificate. There are several different ways that a CA can make the revocation status of its certificates available. Early techniques include the periodic publication of CA-signed copies of the latest CRL on some directory server [X509], which sent the entire CRL in response to a query. As an alternative, the CA keeps the CRL itself and sends signed responses to queries about specific certificates [BERK]. Both these approaches appear to make rather extensive use of one resource or another.

More recently, other methods for publishing CRL information have been suggested. CRLs can be published less frequently with additions to each CRL (called Delta CRLs or δ-CRLs) appearing more frequently. The heavy load on the communication system caused by many users downloading complete CRLs, thus, occurs only infrequently. More common is the downloading of the shorter delta CRLs. Another approach is to divide the CRLs according to reason codes. Thus, a user who cares only about certificates that have been revoked because the corresponding private key has been compromised need download only that smaller, sub-CRL. Other schemes propose placing certificate serial numbers in data structures designed for efficient searching. This will allow users to ascertain quickly whether the certificates in which they are interested are revoked or not. Finally, there are designs that have each CA regularly publish the status of each certificate it has issued. The status reports can be distributed at any frequency desired, even as often as daily or hourly.

1.3 Goals of the MOIE

The objective of this research effort is to compare and contrast the various approaches to certificate validation. It is intended that each of the contrasting methods of publicizing the information on the CRLs be analyzed in several differing scenarios, especially scenarios that fit naturally in a web environment. For each scenario, a comparison will be made of the communication load and the computational requirements. The ultimate consideration for comparing the various methods for dispersing validation status information will be an estimate of the maximum number of validators that can be supported.

The final goal is a set of recommendations advocating the use of one of the certificate validation approaches examined for the various web scenarios. It is possible that a new approach will suggest itself during this process. Should that happen, it is intended that the new method will be analyzed in the same way the others were so that its benefits can be underscored.

1.4 Structure of the Report

The remainder of this report is organized as follows. In the next section, the certificate validation methods to be examined are introduced. Section 3 contains descriptions of the various web scenarios that we shall consider. The load assumptions and the results of evaluating each validation method in each scenario are presented in the next section. Finally, recommendations make up the concluding section.

2 Methods of Certificate Validation

The purpose of certificate validation is to supply sufficient, trustworthy information to allow any subject to construct a proof that the certificate in question is not now revoked or was not revoked at some specific, earlier time. It is the issuing CA, generally, that generates the needed information. However, there are circumstances in which another (perhaps higher level) CA may assume this responsibility. This can happen should the issuing CA be down for repairs or be completely defunct. The entity that wishes to rely on the certificate can either create the necessary proof from the information provided or rely on a trusted validation server to produce the proof. In the latter case, the communication between the proof requester and the proof validator is minimal. Similarly, the proof requester's computational load is little more than the verification of the validation server's signature on the response message. (The requestor, of course, must hold a trusted copy of the validator's public key.) Hence forth, we shall examine the communication and computation loads generated by any entity obtaining and verifying the requisite proof information.

We look at different types of validation proofs. Techniques for making the validation proof information available to the public come in four distinct styles. Each fundamental design may have several variations associated to it. This section contains brief descriptions of each of the basic designs and of those variants that are included in this comparative study.

The four fundamental designs for certificate status proofs are:

- OLV On-line Validation
- CRL Certificate Revocation List
- CRS Certificate Revocation System
- CRT Certificate Revocation Trees

Table 2 (at the end of the section) summarizes the advantages and disadvantages of these methods.

2.1 On-line CA Validation

The simplest method for determining if a specific certificate is still valid is to ask the entity that creates or holds the proof information for the result of the proof. The entity may be the CA that issued the certificate. Alternatively, it may be a trusted responder that the issuing CA has named in the certificate. The CA or the trusted responder replies "yes" or "no" in a short, signed message. The response the serial number of the certificate in question along with a time interval during which the response is to be accepted and after which it should be ignored. That way, the inquirer knows with which certificate to associate the response and also is assured that the response is not a replay of a reply to an earlier request for information about the same certificate. The On-line Certificate Status Protocol (OCSP) [OCSP] is a commonly used standard protocol for on-line validation.

The questioner must have a trusted copy of the public key of the responder, be it the CA or the trusted Repository of proof information. This key is needed to verify the signature on the response. If it is a CA and the CA is the requester's own CA, this is hardly a problem. If the certificate is issued by some other CA then, presumably, the certificate is part of a certification chain

leading from a CA for which a trusted copy of the public key is already in hand. As each certificate in the chain is processed, its signature is verified using the trusted public key of the CA that issued it. The same trusted public key is used to verify that CA's signature on the validity response message. Finally, if the "yes" or "no" message is signed by some other entity, the inquirer will need a trusted copy of that entity's public key.

If the "yes" or "no" message comes from a trusted Repository, the picture is not so simple. The public key of a local trusted Repository may well be known. The public key of a distant Repository will have to be obtained via a certification path that is probably the same as the path for the target certificate – but it need not be. For the purposes of the current analysis, we are assuming certification paths of length one. We also assume that the requester knows the public key of the trusted Repository.

One obvious advantage of this system is that the information about the certificate's validity is as fresh as it can possibly be. This, of course, is true only if the issuing CA or the CA responsible for the revocation immediately informs the trusted Repository of each revocation. Additionally, the validation request and response messages are both relatively short. However, this approach does place a heavy burden on an entity that undertakes to answer all status queries. Such an entity must be available on-line to respond to status requests. Furthermore, the entity spends a substantial part of its resources in signing the responses. Each request carries a time interval to assure that the reply is fresh. That interval can be quite short or can be as long as a day or more. The responder must decide whether it is most important to have the freshest certificate validity information. In that case, it will use short validity periods and sign every response before it sends it out. Otherwise, the responder can use a longer period, sign a single response as a query about each certificate arrives. This response can be cached and reissued whenever another query about the same certificate is received.

2.2 Certificate Revocation Lists

The original X.509 [X509] concept for publishing information about which certificates have been revoked is a certificate revocation list (CRL). Periodically, each CA publishes a singed list of the serial numbers of its revoked, unexpired certificates. How frequently a CA publishes a CRL is a matter of CA policy and varies with the community the CA serves. Each CRL includes the date of its issue and a date when the next CRL will be issued. Anyone wishing to check whether a specific certificate was valid at a certain time obtains the appropriate CA's CRL issued immediately before that time. If the certificate appears on that CRL, then it cannot be trusted. If it is not on that specific CRL and there is a subsequent CRL available, it may useful to check that next CRL to ascertain whether the certificate was still valid after the time of interest. Should there be no subsequent CRL, the requester cannot be completely sure the certificate is/was valid at the time of interest. There is an unavoidable period of uncertainty between CRL publications. This interval of doubt can be reduced only by issuing CRLs more frequently.

In [BERK], we showed that CRLs could be expected to consume significant amounts of communication resources and, at the prevailing communication rates, significant sums of money. In fact, CRLs comprised the most expensive part of running a PKI. Each CRL can become quite lengthy. A single subject verifying the validity of several certificates in a certificate chain may need to download several different CRLs in order to decide whether to trust a single end certificate. The

costs mount quickly. Although infrastructure implementations are only recently beginning to reach the size envisioned in that report, concern for CRL size has existed for some considerable time. Several suggestions on how to reduce the CRL strain on resources have been put forward.

2.2.1 Delta CRLs

One way to reduce the resource drain caused by CRLs is to issue the lists less frequently. Then there is no reason that the CRLs from frequently referenced CAs cannot be downloaded when issued and cached until the next one is issued. This saves communication resources but results in possibly stale validation information. To disseminate more timely revocation data, each CA additionally issues shorter signed lists containing only changes to the last published CRL. These are called "delta CRLs." Each delta CR has a date of issue and a date for the anticipated issue of the next one. A subject checking the validity of a particular certificate looks first on the CRL (as before). If the certificate serial number is not listed there, the subject next examines the delta CRLs to see if the certificate has been revoked since the main CRL was issued. The interval of vulnerability is now reduced from the longer period between CRL publications to the shorter time between the delta CRLs.

This approach reduces the communication load because the delta CRLs are considerably shorter than the main CRLs. Thus, the more frequent download is of the shorter delta CRL. However, this method of proving certificate status requires caching of the full CRLs (perhaps on CD-ROM). There is little extra processing because, presumably, the signature on the cached CRL and the validity of the associated CA certificate were both verified when the CRL was downloaded and cached. Actually looking for a certificate serial number on the full CRL is relatively inexpensive. The only real cost is downloading the delta CRL, verifying its signature, and seeking the certificate serial number. For complete safety, the validity of CA certificate may again have to be checked at the time of verifying the signature on the delta CRL.

2.2.2 Segmented CRLs

A US Patent held by Oorschot, Ford, Hillier and Otway [OOR] presents another method for overcoming the problem of CRLs growing to an unmanageable size. Their methodology has been incorporated in the 1997 version of the X.509 Recommendation [X509]. The authors propose segmenting each CRL into smaller, more easily managed pieces. CRLs can be divided merely on the basis of size. The segments of the CRLs are to be stored in various locations that are to act as distribution points. Each certificate carries the location of the distribution point or points where it will be listed should it ever be revoked.

Subjects validating a certificate may be interested only in ascertaining if the certificate was revoked for some specific reason. For example, an individual making a personal purchase may sign the purchase order with a public key certified through the CA at her place of employment. To the seller, it is important to know whether the certificate was revoked at the time of the order and whether the revocation was the result of the loss or compromise of the corresponding private key. The seller does not care if the certificate was revoked because the individual is no longer associated with that particular employer. On the other hand, if the purchase is made on behalf of the company for which she works, the seller wants to know if her authority to place such an order on behalf of her employer was revoked.

This interest in certificates that are revoked for specified reasons leads to a second way of segmenting a CRL. Each CA subdivides its large CRL are into smaller lists, each consisting of all certificates revoked for a specific reason. There is a key compromise CRL, an affiliation termination CRL, a subject name change CRL, and perhaps a few others. Anyone concerned only about certificates revoked for one reason need download only that specific revocation list. This list is far shorter than the full CRL and its transmission in place of the full CRL is a more economical use of network resources. The effort required in verifying the signature on this shorter list and validating the public key used in signing it is no more than was required for the full CRL. On the other hand, in a certain fraction of cases, it is necessary to consult several of the smaller CRLs issued by one CA. The signatures on each must be verified although only one validation of the CA's certificate is required. Ultimately, if it is important to know whether a particular certificate has been revoked for any reason, it becomes necessary to examine and validate every smaller CRL or to obtain the full CRL.

2.3 Certificate Revocation System

In a Certificate Revocation System, each CA updates the status of each certificate periodically by preparing a certificate revocation status (CRS) for each certificate it has issued. The frequency of update can be daily or even more frequently than that. It is designed to enable positive proof of certificate status without relying on any trusted intermediaries. This Certificate Revocation System was invented and patented by Silvio Micali [MIC].

Associated with each certificate are three quantities, a base "Yes" value Y_0, a final "Yes" value Y_k and a current "Yes" value Y. There are three similar "No" values N_0, N_k, and N. Preferably, each of these quantities is at least one hundred bits long and cannot be guessed or forged by an attacker trying to pass off an invalid certificate as valid. The base numbers Y_0 and N_0 are known only to or can be reconstructed only by the certificate issuing CA. The final values Y_k and N_k (where *k* is the number of status update intervals in the total validity period planned for the certificate) are stored in the certificate. Y and N are the current "Yes" and "No" values in the status proof information. $Y = Y_{k-i}$ encodes the number *i* of status update intervals (since the certificate was issued) during which the certificate has been valid. The value $N = N_{k-j}$ encodes the number of status update intervals *j* that the certificate has been invalid. This last number *j* is, hopefully, zero. The fact that the sum of the two numbers, *i* and *j*, must equal the current status update interval number can be used as a check on the accuracy of the status proof information.

Note that at each update only one of Y or N is changed. If the certificate is still valid, Y goes from $Y_{k-(i-1)}$ to Y_{k-i} and N remains equal to N_k. If this is the j^{th} update when certificate is invalid (either because it was already invalid at the last status update or it has just become invalid, i. e. $j = 1$) Y is unchanged at Y_{k-i} and N goes from $N_{k-(j-1)}$ to N_{k-j}. The complete status proof information update consists of a CA-signed file containing the current status update interval index, all certificates issued since the last proof information update, and a CRS for each and every older, unexpired certificate. Each CRS includes the certificate serial number and the updated value V that is either a new Y or a new N. The proof Repository can distinguish whether each V is an update to the current Y value or to the current N value. In either case, it replaces that value with V and leaves the other number as before. From the currently stored values of Y and N, a requestor can establish for how many update intervals the certificate has been valid and for how many thereafter, if any, it has been invalid.

Micali suggests that Y_{k-i} and N_{k-j} should be computed as $F^{k-i}(Y_0)$ and $F^{k-j}(N_0)$ respectively, where F is some one-way function, i. e., a function whose value is easy to compute but whose inverse is very difficult to evaluate. For example, F can be a secure hash function.

The following diagrams demonstrate how this certificate validity proof system works. In the example, the validity period for each certificate is a week and CRSs are issued daily. Thus, $k = 7$. The CA computes Y_7 and Y_0, N_7 and N_0 and sends these values to the Repository. The relationships between Y_0 and Y_7, between N_0 and N_7 are shown in Figure 2. The values of Y_7 and N_7 are also encoded into the certificate itself so that they are available to anyone who wishes to validate that certificate.

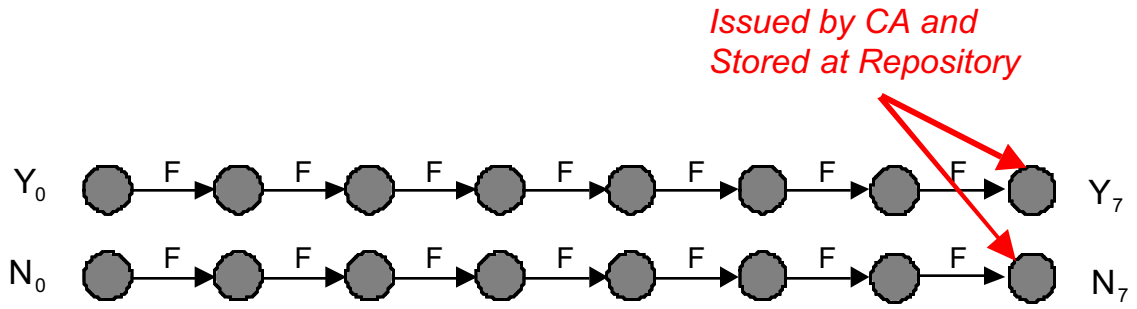

Figure 2: The relationship between initial and final Micali "Yes" and "No" numbers

After a full day in which the certificate has not been revoked, the CA computes the value Y_6 from the Y_0, which it knows. The CA sends this number to the Repository. By applying the function F to the received value, the Repository can determine that the number it received is Y_6 and not N_6. The Repository replaces Y_7 with Y_6 as the current "Yes" value. The current "No" value is still N_7. These "Yes" and "No" values are displayed in Figure 3.

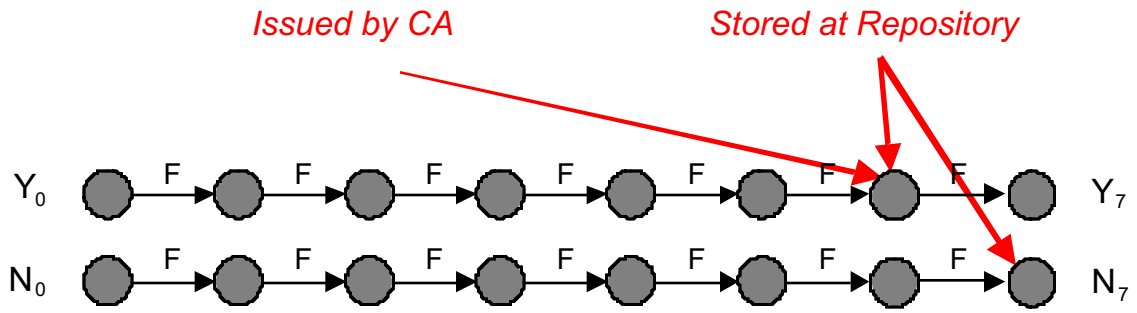

Figure 3: Current "Yes" and "No" values after one day – certificate still valid

Should a client wish to validate the certificate before the next CRS is issued, the Repository sends these two values. The client counts that one application of F to the current Y value is required to reach the Y_7 stored in the certificate and the current N value equals the certificate's N_7. Since

only one daily update of the CRS has occurred since the certificate was issued, the client deduces that the Y and N values are legitimate and that the certificate is valid.

If after the second day the certificate is still valid, the CA issues a CRS for this certificate with the value Y_5. The CA must calculate this anew from Y_0 or may have saved all the Y values during the computation of Y_7. In the latter case, the CA simply looks up the value of Y_5. Again, by one application of F, the Repository recognizes the value it receives from the CA as a new Y value. It now stores Y_5 and N_7 as the current values. This situation is shown in Figure 4.

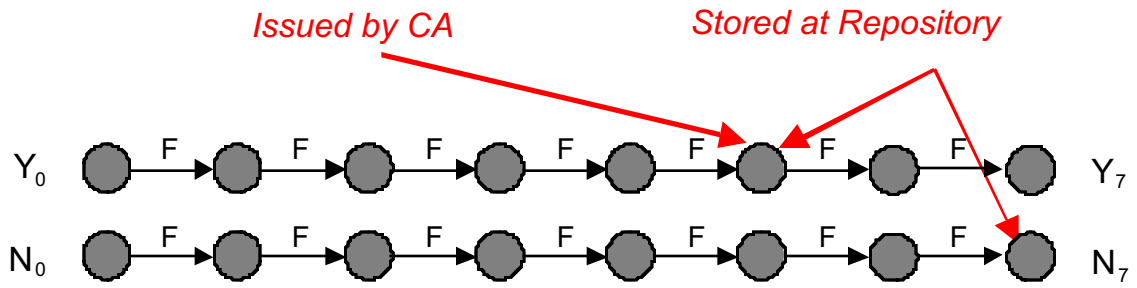

Figure 4: Current "Yes" and "No" values after two days – certificate still valid

When a client requests validation information at this point, the Repository responds with the current values, Y_5 and N_7. Two applications of F show that the certificate has been valid through two CRS updates. The fact that the current N value equals N_7 and two updates are exactly the number that should have occurred since the certificate was issued assures the client that these values can be trusted.

Suppose, now, that the certificate is revoked on the third day. At the subsequent CRS issue, the CA computes N_6 from N_0 or retrieves it from memory if it has saved all intermediate values. The CA sends this value to the Repository. One application of F does not yield the current Y value, Y_5. However, the result does match the current N value, N_7. The Repository stores it as such. Thus, the Repository now holds Y_5 and N_6 as the current values (see Figure 5).

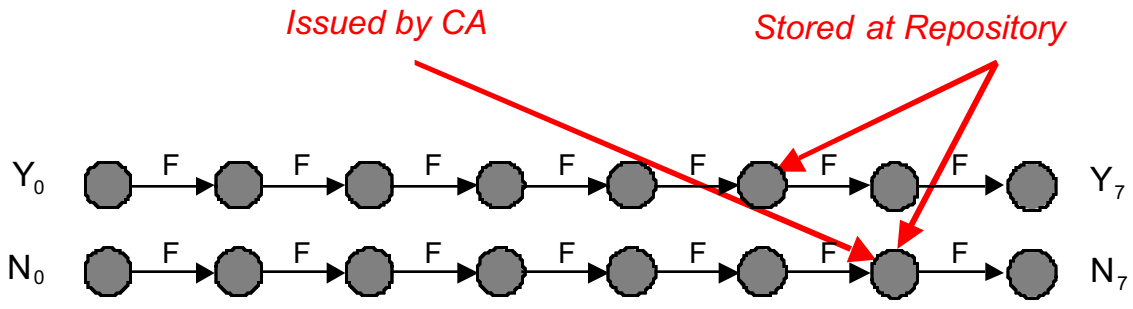

Figure 5: Current "Yes" and "No" values after three days – certificate now invalid

Now, when a client inquires about the certificate, it receives Y_5 and N_6 in response. The client must apply F three times to ascertain the certificate's validity history. Two applications of F move Y_5 and Y_7 and one moves N_6 and N_7. Thus, on the three days that have passed since the certificate was issued, the certificate was valid at the end of the first two but invalid by the end of the third. It is apparent that the certificate was revoked some time on the third day.

This method has three advantages and (at least) one disadvantage. One advantage is that the Repository does not need to be trusted. As opposed to the On-Line method (for example), the Repository performs no trusted computation. It computes the hash of what it receives and decides whether the value is a Y value or an N value. It then replaces that value with the one sent to it by the CA. If the Repository does not do this as required, the Validator will discover that fact when checking the response to a query. Also, the size of a query response small—two hash values. In addition, the Micali scheme has the advantage that all the values that are needed to update the validations status of a certificate are calculated at certificate creation. As Figure 2 above shows, to calculate the Y and N values that will be included in the certificate, the CA will also need to calculate every other value in the hash chain. Hence, if the CA stores those values for future reference, it need not recalculate the hash chain for each update. However, the Micali method also has a significant downside. Under this method, when issuing an update the CA must include a value for *each* certificate, while under other methods the CA need only include a value for revoked certificates. Hence, the updates may be much larger under this method than they would under other methods.

2.4 Certificate Revocation Trees

Certificate Revocation Trees (CRTs) attempt to strike a balance between CRLs and on-line validation methods. Whereas on-line methods require a signature for every verification request, CRTs leverage one signature by the CA into providing cryptographic security for every validation reply. On the other hand, while CRLs provide information about every revoked certificate, CRT validation replies contain only information pertinent to the certificate mentioned in the validation request.

CRTs were developed by Paul Kocher [KOCH] of Valicert and were later improved by Naor and Nissim [NOAR] of the Weizmann Institute[3]. Construction and use of a CRT begins by noting that the set of all certificates issued by one CA can be sorted by serial number and broken into continuous ranges of valid certificates.

For example, suppose that certificates 21, 36, and 54 had been revoked. Then the following four statements are true:

- If a certificate has a serial number in the range $-\infty$ to 21, then the certificate is valid unless it has serial number 21
- If a certificate has a serial number in the range 22 to 36, then the certificate is valid unless it has serial number 36

---

[3] While Kocher's CRTs are apparently in use at Valicert, the Naor-Nissim improvement does not appear to have been implemented yet.

- If the certificate has a serial number in the range 37 to 54, then the certificate is valid unless it has serial number 54
- If the certificate has a serial number in the range 55 to ∞, then the certificate is valid unless it has serial number ∞.

These four statements may seem somewhat strange. Note, however, that they all have the same format and that, for any verification query, exactly one of the above statements is relevant.

Given a reasonable encoding of the above statements, a Certificate Revocation Tree is a binary tree[4] constructed in the following way:

- The leaves of the tree are the structures representing statements like those above.
- The internal nodes hold hashes of their children (See Figure 6).

To use a CRT, the CA creates one of these trees internally and signs the root node. The CA then sends the signed root node to the Repositories, along with an unsigned list of revoked certificates – an unsigned CRL. The Repository, upon receiving the signed root node and revoked certificate list, constructs its own copy of the tree and verifies that the signed root node matches the root node of the tree it just generated.

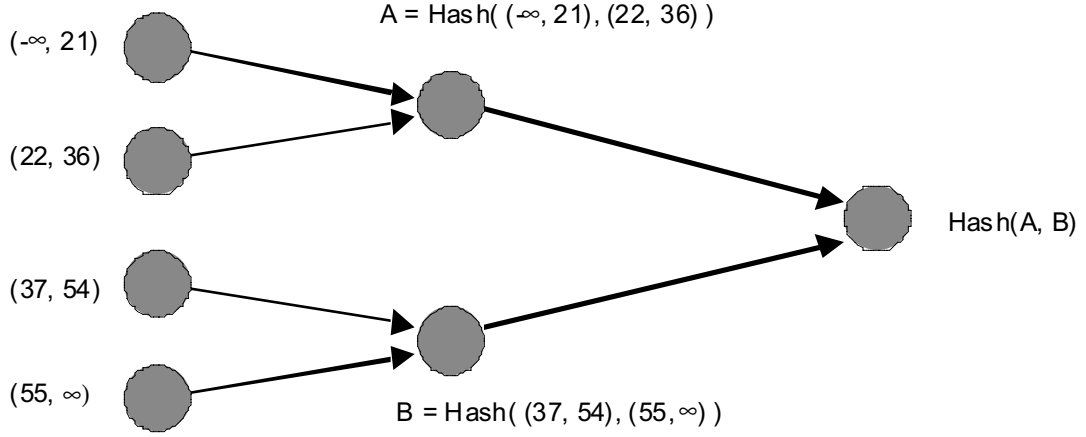

Figure 6: Example Certificate Revocation Tree

[4] Meaning that every internal node has exactly two children.

When responding to a verification query, the Repository first determines the relevant leaf, then returns:

- The CA's signature on the root node,
- The relevant leaf
- The relevant leaf's sibling,
- The sibling of each of the relevant leaf's ancestors.

Figure 7, for example, has circled the nodes returned in response to a verification request for certificate number 40.

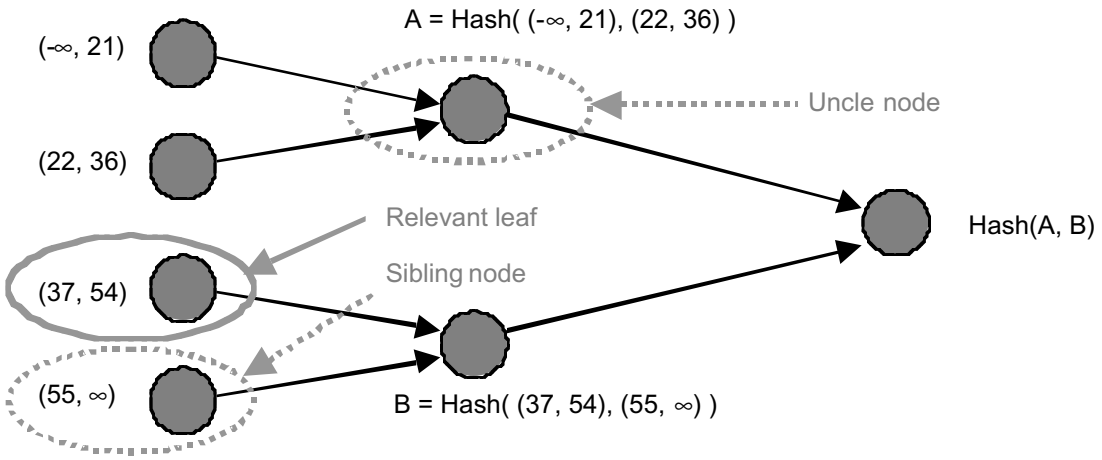

Figure 7: Nodes returned in response to query regarding certificate 40

The validator hashes the relevant leaf with its sibling to get its parent node, hashes that with the uncle node to get the grandparent node, hashes that with the great-uncle to get the great-grandparent node, and so on. This process continues until the validator has what it believes to be the root node of the tree. The validator then checks this root node against the root node signed by the CA. If the two root nodes match, then the validator can be cryptographically sure that the leaf in question was in fact a leaf in the CA's tree.

Forging a response to a validation query requires either the forging of a signature or the inversion of a hash. If the signature of the CA can be forged, then a malicious adversary can create any hash tree it wishes and can forge the signature of the CA on the root node of that tree. On the other hand, since the signature probably cannot be forged, an adversary is forced to fabricate a chain of false internal tree hash nodes. These nodes produce same value for the root node as that of the signed root node—in other words, he must invert a hash. Under the very reasonable assumption that both inverting a hash and forging a signature are infeasible, the signature of the CA on the root node supports trust in every query response.

In this respect, CRTs share the advantages of CRLs. One signature by the CA secures all validation responses, thus reducing the load upon Repositories. However, the size of CRT

validation responses, sent from Repository to validator, are much smaller than the corresponding messages under CRLs. Whereas the size of a CRL grows linearly with the number of revoked certificates, the size of a validation response using CRTs grows as the *log* of that number. Therefore, while doubling the number of revoked certificates will double the size of a CRL, it will add only a (small) constant amount to the size of a CRT validation response. It should be noted, however, that the size of this response is still much larger than would be used in an on-line method, where the size of a validation response is constant.

On the other hand, while the Repository does no work when constructing a validation response, it may have to do a considerable amount of work when receiving an update from the CA. Using Kocher's construction, the Repository is forced to re-construct the entire tree to add or delete a single leaf. Naor and Nissim address this concern by suggesting that the revocation tree use a slightly more complicated structure called a 2-3 tree.[5] The main advantage to this structure is that additions and deletions of nodes can be done quickly. The time required is proportional to the height of the tree (length of a path from root to leaf), which is proportional to the *log* of the number of nodes in the tree.

2.5 δ-CRL/Micali Hybrid

While the δ-CRL method is efficient for distributing updates, it is less efficient as a query response mechanism. On the other hand, Micali's Certificate Revocation Status is extremely efficient for query responses but enormously costly for updates. In an attempt to combine the advantages of both, we also tested an hybrid of the two methods. In the δ-CRL/Micali hybrid, the CA chooses values for Y_0 and N_0 and computes the associated Y and N values when it creates a certificate. As it would in Micali's original method, the CA includes the values of Y and N in the certificate. It sends the certificate and the secret Y_0 and N_0 values, encrypted, to the Repositories. As updates, the CA issues δ-CRLs. The Repositories are responsible for updating the current values of Y_i and N_j based on the δ-CRL. This they can do because they have the Y_0 and N_0 values. (They can also take advantage of the pre-computation optimization available to the CA in Micali's original scheme.)

2.6 Comparison of Validation Methods

Table 1 presents a comparison of the message lengths required by each of the above validation methods. It examines messages both between the CA and the Repository and between the Repository and the Validator. It also lists the computational load on the Repository in responding to a validation query.

The second table presents a synopsis of the strengths and the weaknesses of each validation method. Both tables include entries for Black Lists [Perl], a technique in which all certificates are valid for a relatively short time. Consequently, CRLs (called Black Lists) are relatively short. However, the short validity period forces frequent issuing of new certificates for all users. This seems to be an impractical approach and we did not include Black Lists in our analysis. They appear in the tables for the sake of completeness.

---

[5] In a 2-3 tree, every internal node has either two or three children, and every path from the root to a leaf has the same length.

Table 1: Work and Message Lengths, by Validation Method

	Size of CA-Validator messages	Repository work per validation request (time)	Size of Repository-Validator messages
CRLs	Proportional to number revoked but unexpired certificates $O(mR_r)$	None	Proportional to number of revoked but unexpired certificates $O(mR_r)$
Δ-CRLs	Proportional to number of certificates revoked per day[6] $O(mR_d)$	None	Proportional to number of certificates revoked per day $O(MR_d)$
Segmented CRLs	Proportional to number of unexpired certificates revoked per reason of revocation $O(mR_r/N_r)$	None	Proportional to number of unexpired certificates revoked per reason of revocation $O(mR_r/N_r)$
Black Lists	Proportional to number of certificates revoked since last cut-off date	None	Proportional to number of certificates revoked since last cut-off date
On-line Verification	Proportional to number of revoked but unexpired certificates $O(mR_r)$	1 Signature	Constant
CRSs	Proportional to number of unexpired certificates $O(mR_r)$	None	Constant
CRTs	Proportional to number of revoked but unexpired certificates $O(mR_r)$	None	Proportional to the *log* of the number of revoked but unexpired certificates $O(\log(mR_r))$

- m- number of unexpired certificates
- n_u- number of updates per certificate lifetime
- R_d- proportion of certificates revoked per day
- R_r- proportion of certificates revoked at any given time (= _ R_d (n_u+1))
- N_r- number of reasons for revocation

[6] This assumes that Delta CRLs are being issued daily.

Table 2: Advantages and Disadvantages of Validation Methods

	Advantages	**Disadvantages**
CRLs	• Requires no work on part of Repository	• Message lengths proportional to number of certificates • Messages, including the many query responses, can grow quite long
δ-CRLs	• Shorter query response message lengths • Requires no work on part of Repository	• Message lengths still grow proportional to number of certificates
Segmented CRLs	• Shorter query response message lengths • Requires no work on part of Repository	• Message lengths still grow proportional to number of certificates
Black Lists	• None	• Message lengths grow proportional to number of certificates • Requires re-issuing all certificates frequently
On-line Verification	• Small message lengths • Timeliness	• Requires trust in Repository • Requires work on part of Repository
CRSs	• Short message lengths • Requires no work on part of Repository	• Very large CA-Validator communication
CRTs	• Requires no work on part of Repository	• Moderately large message lengths

3 Web Architectures

The objective of this effort is to compare and contrast the various approaches to certificate validation and to analyze each of these methods in different scenarios, particularly those scenarios that reflect the web environment. In a web environment, the chief uses of certificates are either the authentication in SSL connections or the enforcement of access controls. The validator can be a web server authenticating a client or a client authenticating a server. Although the verification rates for client and server certificates probably are different, the network topology is same.

In each of these scenarios, there are three different players: the Certificate Authority (CA), the Repository, and the Validator. Once per update period, the Certificate Authority sends information concerning all unexpired certificates to each appropriate Repository. The Repositories use this information to respond to verification requests from the Validators. The exact nature of the messages involved—for both CA-Repository and Repository-Validator communication—depend on the verification scheme in use. There are, of course, multiple Validators and there may be multiple Repositories. Each player in a particular scenario is connected to a shared WAN by a LAN; the LAN itself may be shared by more than one player. Because we are examining the web environment, it we do not consider Validators that are on the same LAN as the CA.

We have identified, for use in our analysis, five different scenarios, each described in its own section below.

3.1 Scenario 1: Third-party Repository

In this scenario, the CA communicates with a number of Repositories scattered across the WAN. This is a situation much like the Domain Name System (DNS), in which a number of servers are responsible for serving a given zone.[7] DNS specifications require that some of these servers be located in different LANs across the Internet. This scenario will also arise if CAs wished to store information in Repositories on other networks for redundancy reasons. The purpose is to minimize the effects of network outages.

In this scenario, the number of Repositories is small—typically on the order of 10—and it does not grow with the number of Validators. Figure 8 depicts this scenario.

---

[7] In DNS, a *zone* is a set of domain names under a common administration.

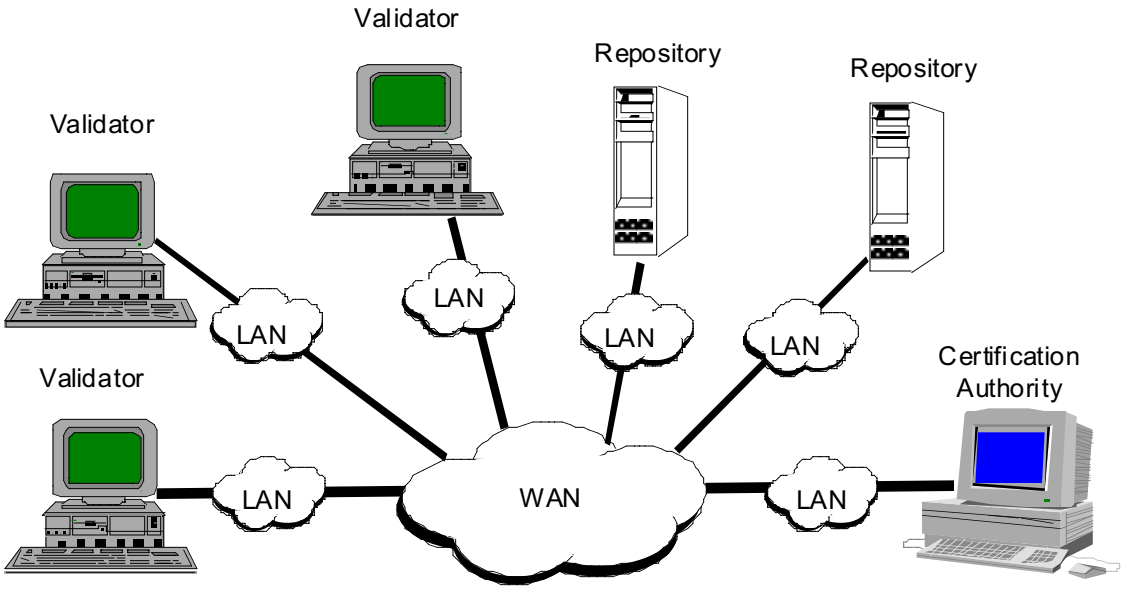

Figure 8: Third-Party Validator

3.2 Scenario 2: Associated Repository

In this scenario, the CA communicates only with a local Repository. One would expect to find this scenario in any PKI presently deployed. Typically, agencies/organizations that administer PKIs maintain a CA and a Repository on the same LAN.

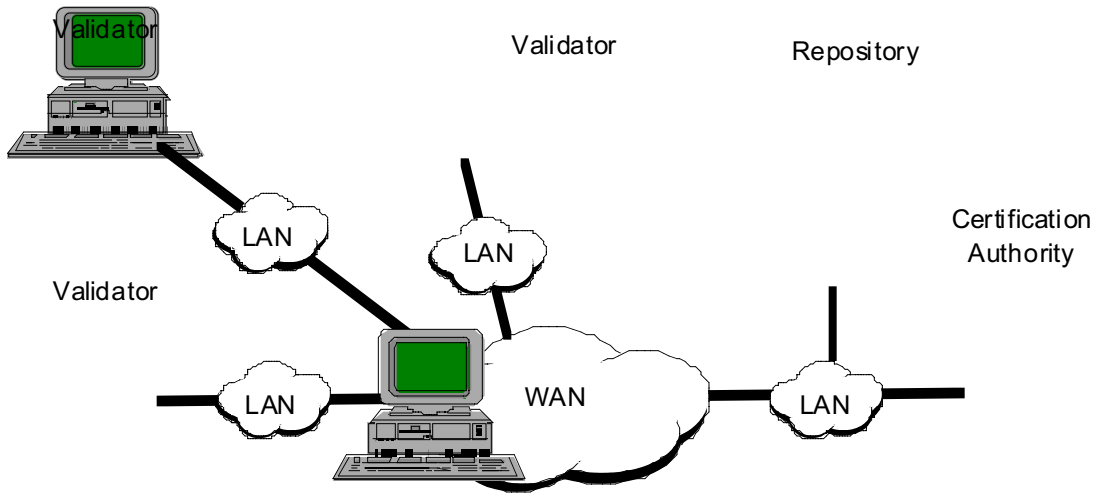

Figure 9: Associated Repository

3.3 Scenario 3: Local Repositories

Here the assumption is that the Validators are grouped on LANs, each of which has its own local Repository. This scenario applies when the PKI is used to provide security for an enterprise-wide intranet, such as the MITRE Information Infrastructure (MII). Validators (users) are segregated into departments, each with its own network and local servers.

The number of Validators per LAN is fixed, in this scenario, at some large number (100 to 1000). Consequently, the number of Repositories grows with the number of Validators. Figure 10 depicts this scenario.

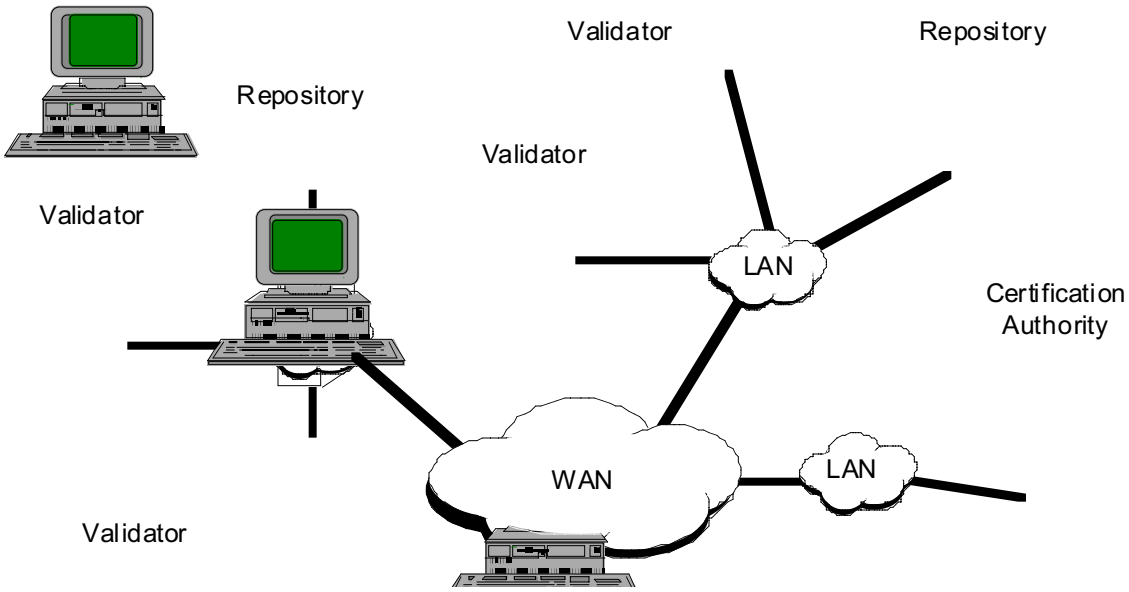

Figure 10: Local Repositories

3.4 Scenario 4: No Repository (CA Proxy)

In this scenario, there are no independent Repositories. The CA, which saves all its revocation information anyway, responds directly to verification queries. In essence, there is exactly one Repository, which happens to be located in the same host as the CA. This scenario is shown in Figure 11.

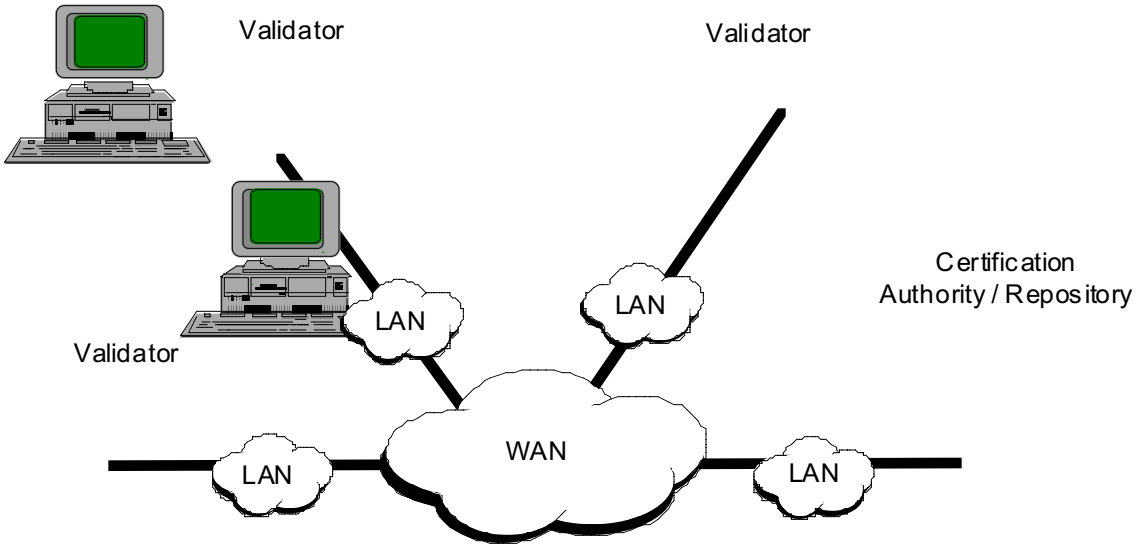

Figure 11: No Repository (CA Proxy)

3.5 Scenario 5: No Repository (Validator Proxy)

As in Scenario 4, this scenario contains no Repository. However, here the role of the Repository is assumed by the Validators. Once per update, the CA sends to each Validator the information it would have sent to the Repository. In other words, the CA pushes to each Validator all the proof material the latter might possibly need for all possible certificates. This situation arises with Compromised Key Lists (CKLs), which are CRLs pushed from CA to Validators. This alternative is depicted in Figure 12.

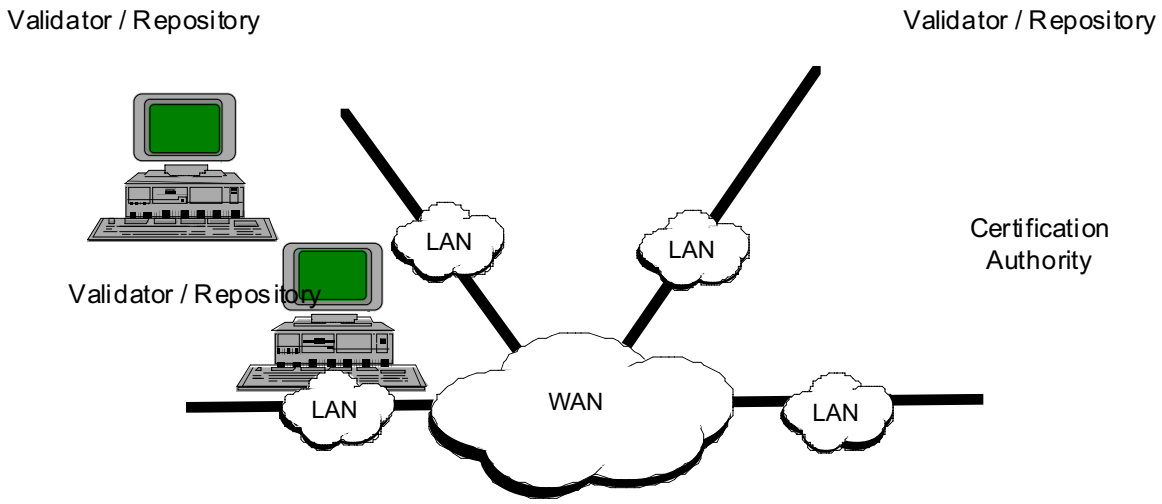

Figure 12: No Repository (Validator Proxy)

4 Method Analysis

In this section, we describe the model, its assumptions, and its constituent components. We then outline the results produced by the model. We analyze and explain the significance of those results.

4.1 Resource Consumption Model

The model used in this analysis is a resource-consumption model. This means that the model measures the consumption of critical computation and communication *resources* in response to various independent *driver* variables. The drivers indirectly consume those resources by producing *events*. In this model, there are two drivers, three resources, and two events:

4.1.1 Drivers

- **Validators** are the agents that wish to validate certificates. Each validator produces validation requests at some frequency, which depends largely on two considerations.
    - *Type of validator:* Because we are examining validation methods in the web environment, it is expected that the validators will be either web clients or web servers. It is assumed that web clients connect to only a few secure web servers each day and so will need to validate only a small number of certificates. On the other hand, web servers handle large numbers of connections daily and are expected to validate thousands-- or even millions-- of certificates.
    - *Caching:* A client validates a certificate once and does not need to validate that certificate again during the same update period. (If it were to attempt a second validation in one update period, it would receive the same information as it did the first time.) On the other hand, after validating one certificate, the validator may also gain the information it needs to validate other certificates. For example, once a validator retrieves a CRL, it need never make another validation request during the same update period.
- **Updaters** are the agents that issue periodic certificate status updates. Because the number of updaters is independent of the validation method, we assume for simplicity's sake that the CA is the only updater in the system.

4.1.2 Resources

- **Certificate Authority**: Aside from certifying key pairs and perhaps even generating them, the CA is responsible for updating the revocation information in the Repositories. Depending upon the verification method in use, this update may take one of several forms. Under the CRL method, for example, the CA hashes the list of revoked certificates and signs the resulting digest. When using CRTs, however, the CA builds the hash tree and signs the root node. Each verification method requires some work on the part of the CA. This work consumes processor time.

Since the resource provided by the CA is processor time, the CA can provide at most one second per second. In practice, a CA will probably devote less than full capacity to updates because it has other functions it must perform.

- **Repository**: Some verification methods require an active Repository. For example, the On-line Verification method requires the Repository sign each response, a costly operation.

  Like the CA, the resource provided by the Repository is processor time. Each Repository can provide up to one second per second. The actual time devoted to responding to status inquiries may be less since the Repository is also responsible for distributing certificates to clients that request them.

- **LAN**: The CA, each Repository, and each Verifier have a LAN connection to a common WAN. Although two or more parties share a LAN in some of the architectures (see Section 4), they do not share a common LAN on all architectures. The bandwidth of each LAN bounds the size and number of messages being sent over it.

4.1.3 Events:

Each event will consume some resources. The degree to which a given event will consume a given resource will depend on two things, the architecture and the method used. The events are:

- **Validation Queries and Query Replies**: As stated above, each Validator will produce validation requests with some frequency and each request will require a reply. We assume that the size and cost of the request itself does not change from method to method. What does change, however, is the size and cost of the query reply. Hence, in this model we will ignore the actual request, and consider only the resulting reply.

- **Updates** are produced by Updaters periodically. As with validation requests and responses, we ignore the constant, overhead that would be associated with all forms of updates, and consider only the variable costs affected by the particular method is use.

4.1.4 Model use:

The drivers produce events, which in turn consume resources. Figure 13 shows the relations between the drivers, the events, and the resources. Because the resources have finite capacities, the number of drivers that can be supported before some resource becomes exhausted is bounded. We assume that the number of updaters is fixed at 1, and that the frequency with which it produces updates is also fixed, at once per day. This means that the system can support only a finite number of validators before some resource becomes overloaded. That number, however, will depend on several things:

1. The type of validator (client or server),
2. Whether or not validators cache previous query responses,
3. The size of the PKI (i.e., the number of certificates in circulation)
4. The architecture (see Section 3), and
5. The validation method in use.

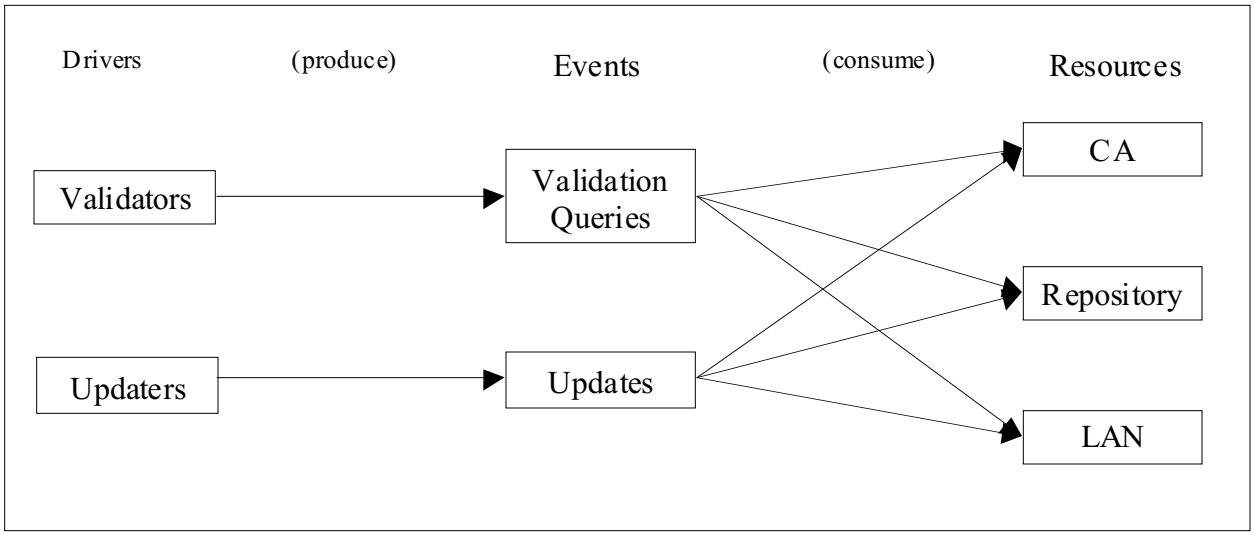

Figure 13: The Relationship between Drivers, Events, and Resources

Because the focus of this investigation is to compare the scalability of the various validation methods (i.e., the number of validators the method can support) we will use the model in the following way:

- Enumerate all possible combinations of conditions 1—5 above.[8]
- For one particular combination, find the largest number of Validators the resources can support.
- For each combination of conditions 1—4, find the validation model that can support the largest number of Validators.

A method that is found this way is the one that scales the best, under the given, particular set of conditions.[9]

- For the rest of this section, we will develop this model in more detail.

4.2 Resource Consumption Model

- Drivers produce *events* that are either *updates* or *validation requests*.
- Driver *i* produces event *j* at frequency f_{ij}. With one exception, these frequencies are fixed. The frequency with which validators produce validation requests varies. It depends on the type of validator, whether or not the validator caches, and, if it does, the type of validation method being used.

---

[8] In the case of condition 3, choose discrete PKI sizes representing the range likely to be encountered in actual use.

[9] Although it is tempting to believe that the model can produce some predictive value, and that the number of validators the method can support in the model is the number of validators the method can handle in real life, it is unfortunately not so. The model was designed for comparison purposes, and hence ignores large aspect of a PKI that are irrelevant to validation. However, it is entirely possible that a model of this kind could be constructed for an entire PKI if a predictive model is desired.

- CRLs and δ-CRLs contain information about all certificates. Hence, once a validator obtains either type of CRL in response to a query, that validator need never make another validation request until the next update period. Of course, the next update time occurs sooner for a δ-CRL than for a full CRL.
- On-line methods, Micali, and the δ-CRL/Micali hybrid, on the other hand, provide only enough information to determine the status of one query. Hence, a validator will have to make a separate query for each certificate it wishes to validate.
- Each leaf in a CRT contains information about some range of certificates. Consequently, once a validator obtains one leaf of a CRT, it has information about several certificates. Because CRTs have one leaf per revoked certificate, caching validators using CRTs need never make more validation requests than there are revoked certificates. If it ever possesses every leaf of the CRT, the validator has complete information about all certificates. It need then never make another validation request during that update period.

Values for the frequencies f_{ij} are shown in Table 3.

Table3: Frequencies for Driver Induced Events

$f_{updater,\ update}$ = 1 per day per updater

$f_{updater,\ val\ req}$ = Never

$f_{validator,\ update}$ = Never

$f_{validator,\ val\ req}$ (per day, per validator)	CRL	δ-CRL	On-line	CRT	Micali	δ-CRL/ Micali
Non-Caching						
Server	1,000	1,000	1,000	1,000	1,000	1,000
Client	1	1	1	1	1	1
Caching						
Server	1	1	1,000	Once per revoked certificate, at most	1,000	1,000
Client	1	1	1	1	1	1

$$\frac{1}{R_k} \sum_{j \in Events} \sum_{i \in Drivers} f_{ij} D_i a_{jk} m_{jk}$$

- If there are D_i drivers of type *I* (e.g., there are $D_{validators}$ validators), then the total number of events of type *j* is:

$$\sum_{i \in Drivers} f_{ij} D_i$$

- If there are R_k resources of type *k* (e.g., there are $R_{Repository}$ Repositories), then the total number of events of type *j* per unit or resource is:

$$\frac{1}{R_k} \sum_{i \in Drivers} f_{ij} D_i$$

  R_k depends on the architecture and is given in Table 4[10]

Table 4: Frequencies for Driver Induced Events

Architecture	R_{LAN}	R_{CA}	R_{resp}
1	1	1	10
2	1	1	1
3	1	1	1 per 1,000 validators
4	1	0	0
5	1	1	1 per validator

- Hence, the total consumption of resource *k* is:

Where

- a_{jk} is an indicator variable which is
  - 1 if resource *k* is consumed by event *j* in the architecture,
  - 0 otherwise, and

---

10 Technically, there is more than one LAN in each architecture. However, in each architecture, one LAN is the busiest. The system will be at maximum capacity when the busiest LAN is saturated. In architecture 1, that LAN is the one on which a repository lies. In all the other architectures, it is the LAN on which the CA lies.

- m_{jk} is the degree to which resource k would be consumed by event j, if it is consumed in the architecture. It is this variable that most shows the differences between the various validation methods.

The values of a_{jk} and m_{jk} are given in Table 5 and in Tables 6a and 6b respectively.

Table 5: Indicator Variables for Resource Consumption, by Event and Architecture

a_{jk}	LAN	CA	Rep
Update			
Architecture 1	1	1	1
Architecture 2	1	1	1
Architecture 3	1	1	1
Architecture 4	0	1	
Architecture 5	1	1	1
Validation Request			
Architecture 1	1	0	1
Architecture 2	1	0	1
Architecture 3	0	0	1
Architecture 4	1	1	
Architecture 5	0	0	1

Table 6a: Resource Consumption, by Event and Method: Variable Definitions

s -- time required for a signature	.173 seconds[11]
$\|s\|$ -- length of a signature	1024 bits
v – time required to verify a signature	.004 seconds[10]
n_c – number of certificates	Varies
r_r – revocation rate (% of certificates revoked at any time)	5%
h – length of hash output	160 bits[12]
l – length of certificate serial number	160 bits
n_u -- number of updates per certificate lifetime	365
r_h – hashing rate (bits / second)	1,565,576 bytes / second[13]

[11] JSAFE 1.1 Benchmarks, assuming Pentium computer (166MHz). RSA Data Security Engineering Report (1998)

[12] Output length of SHA-1

[13] JSAFE benchmarks

Table 6b: Resource Consumption, by Event and Method: Values

m_{jk}	LAN	CA	Repository
Update			
CRL	$n_c\ r_r\ l + \|s\|$	$(n_c\ r_r\ l) / r_h + s$	$(n_c\ r_r\ l) / r_h + v$
δ-CRL	$(n_c\ r_r\ l) / n_u + \|s\|$	$(n_c\ r_r\ l) / (n_u\ r_h) + s$	$(n_c\ r_r\ l) / (n_u\ r_h) + v$
Micali	$n_c\ h$	$(n_c / n_u) * (2\ n_u\ l/r_h)$	0
CRT	$n_c\ r_r\ l + \|s\|$	$(2\ n_c\ r_r\ l + n_c\ r_r\ h) / r_h + s$	$(2\ n_c\ r_r\ l + n_c\ r_r\ h) / r_h + s$
On-line	$(n_c\ r_r\ l) / n_u + \|s\|$	$(n_c\ r_r\ l) / (n_u\ r_h) + s$	$(n_c\ r_r\ l)/(n_u\ r_h) + v$
δ-CRL / Micali	$(n_c\ r_r\ l) / n_u + \|s\|$	$(n_c\ r_r\ l) / (n_u\ r_h) + s$	$(n_c\ r_r\ l) / (n_u\ r_h) + v + (n_c / 365) * (2\ n_u\ l / r_h)$
Validation response			
CRL	$(n_c\ r_r\ l) + \|s\|$	0	0
δ-CRL	$(n_c\ r_r\ l)/ n_u + \|s\|$	0	0
Micali	$2h$	0	0
CRT	$2l + log2(n_c\ r_r)\ h$	0	0
On-line	$\|s\| + 300$	0	S
δ-CRL / Micali	$2h$	0	0

Lastly, if resource k has capacity C_k (per unit of resource), then the normalized consumption of resource k is:

$$\frac{1}{R_k\ C_k} \sum_{j \in Events} \sum_{i \in Drivers} f_{ij} D_i a_{jk} m_{jk}$$

Values for C_k are given in Table 7

Table 7: Resource Capacities

C_k	
LAN	100,000 bits/second[14]
CA	1 second / second
Repository	1 second / second

The variables shown in Table 6b have been implement in a spreadsheet. The spreadsheet results produced by the model are used to find the best validation method. The best method is the one that supports the largest number of validators under a given set of conditions. The results are dependent on several considerations.

1. Validators can be either clients or servers. The type of validator will affect the frequency with which each validator issues validation requests. (See Section4.1.)
2. Validators can cache or not cache, which also affects the frequency with which they issue validation requests. (Again, see Section 4.1)
3. The PKI can vary in size, which will affect the message lengths of some methods. We used the five sample sizes given in Table 8

Table 8: PKI Sizes

10,000
100,000
1,000,000
10,000,000
100,000,000

4. The architecture can vary among the five listed in Section Section 4.1, and
5. The validation method is one of the six given in Section 4.1.

4.3 Model Results & Discussion

Before discussing the results of the model, it is helpful to make an initial observation. The architectures can be broken into two broad categories, based on their growth behavior:

[14] 1% of a 10Mbit/sec network

- In architectures 1 (Third-Party Repository), 2 (Associated Repository) and 4 (CA-Repository), the number of Repositories is constant. Hence, as the number of validators increases, the number of validators each repository is required to handle increases as well. Ignoring possible effects of caching, one would expect the most common event in this type of architecture to be the validation request. Therefore, the most efficient method for these architectures will most likely be the one that handles validation responses the best, i.e. Micali's Hash Chain method and the δ-CRL/Micali hybrid.

- On the other hand, in architectures 3 (Local Repository) and 5 (Validator-Repository), the number of Validators per Repository remains constant. Hence, as the number of validators increases, the number of Repositories must do so as well. Thus, in these architectures, the most common communication will be updates. We expect that he most efficient methods are those that best handle updates, i.e. those that use δ-CRLs for updates.

In fact, this is exactly the behavior that occurs (although caching does have an effect under certain circumstances). Figure 14 below shows the relative capacities of each method for non-caching clients in Architecture 1. One would see similar results for servers in Architecture 1, and for both clients and servers in architectures 2 or 4.

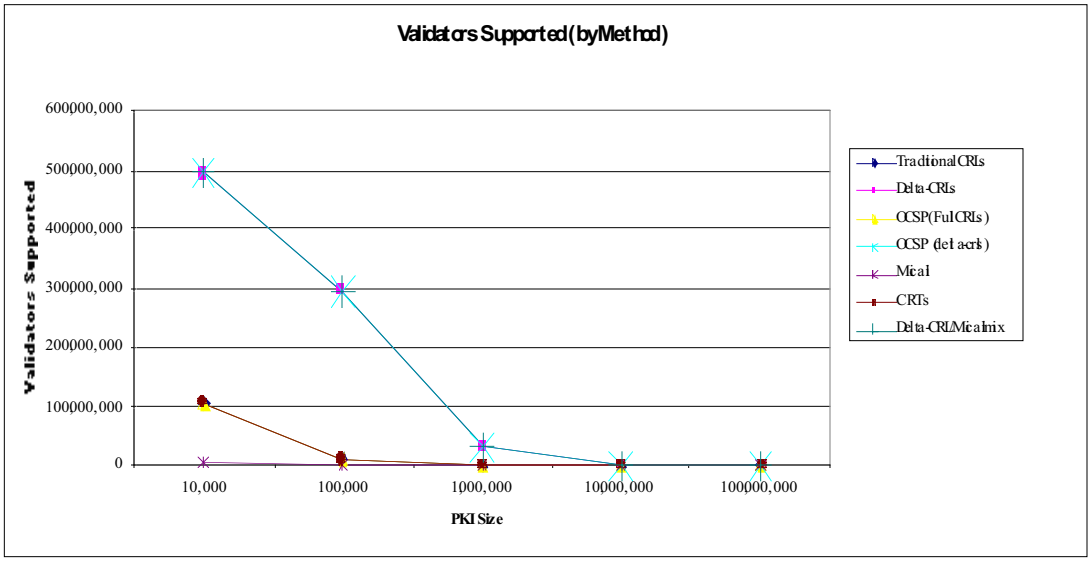

Figure 13: Maximum Number of Non-Caching Servers Supportable – Architecture 1

As can be seen, the methods based on Micali-like query responses show a marked advantage over all other methods—up to a point. Using δ-CRLs for the update can extend this point somewhat, but all methods have their limits.

For the second type of architecture, Figure 15 shows the relative capacities of the various methods in Architecture 3. Again, there are similar results for both clients and servers. The graphs for both clients and servers in Architectures 5 are also similar. The three methods that use δ-CRLs for updates (δ-CRLs, OCSP with δ-CRLs, and the δ-CRL/Micali hybrid) show the highest capacity and produce almost identical results. This demonstrates that, in these architectures, the capacity of each method depends on the method's update scheme.

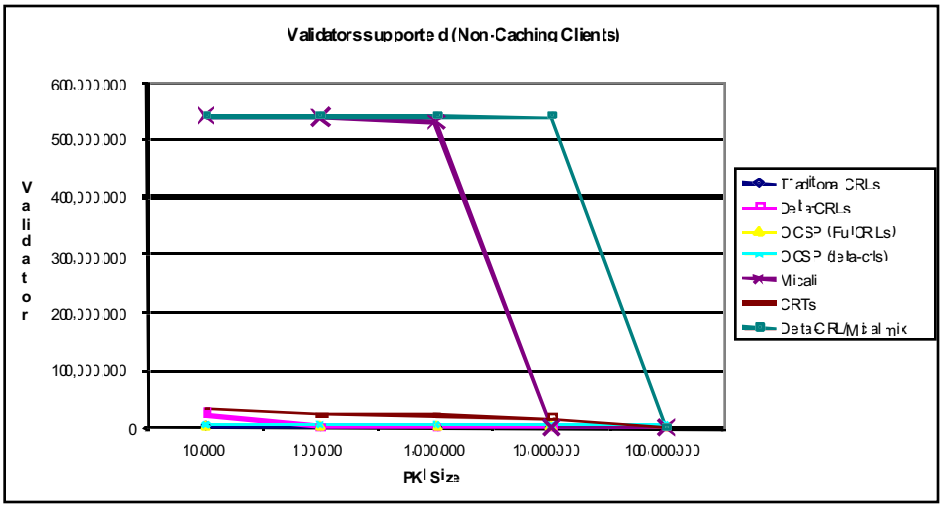

Figure 14: Maximum Number of Non-Caching Servers Supportable – Architecture 3

The results for non-caching validators are summarized in Table 9.

Table 9: Non-Caching Validators

Architecture	Most clients supported by:	Most servers supported by:
Third-party (1), Associated (2), CA-repository (4)	• Micali • δ-CRL/Micali Hybrid	
Local Repository (3), Validator-Repository (5)	• δ-CRL/Micali Hybrid	• δ-CRL • δ-CRL / On-line • δ-CRL / Micali hybrid

The results for validators that cache are much the same, showing that caching only has a tangible effect under fairly narrow conditions:

- Caching improves efficiency by eliminating the need for many validation queries. The only methods that allow the validator to take advantage of caching are those that provide information about more that one certificate at a time—namely CRLs, δ-CRLs, and to some extent, CRTs. However, each CRT response provides information about a small fraction of the total number of certificates. As a result, CRTs are not as effective as CRLs or δ-CRLs.
- Making caching possible for a reasonably sized client or server means that the CRLs or δ-CRLs must be of a manageable size. This requirement implies that the PKI must be small.
- For a validator to gain the benefits of caching, it must need to validate many certificates per update period. Hence, clients—which as we defined in this report to be those agents that need to validate only a few certificates per update period—gain very little by caching.
- Lastly, for caching to improve efficiency, it must improve the performance of that process that actually exhausts the resources. Because caching improves the performance of the validation process (as opposed to the update process), it is this process that is important. In other words, caching only improves efficiency in Architectures 1, 2 or 4.

As a result, caching improves efficiency for servers using CRLs or δ-CRLs in small PKIs and only in Architectures 1, 2, or 4. Figure 16 shows the capacities of caching client validators:

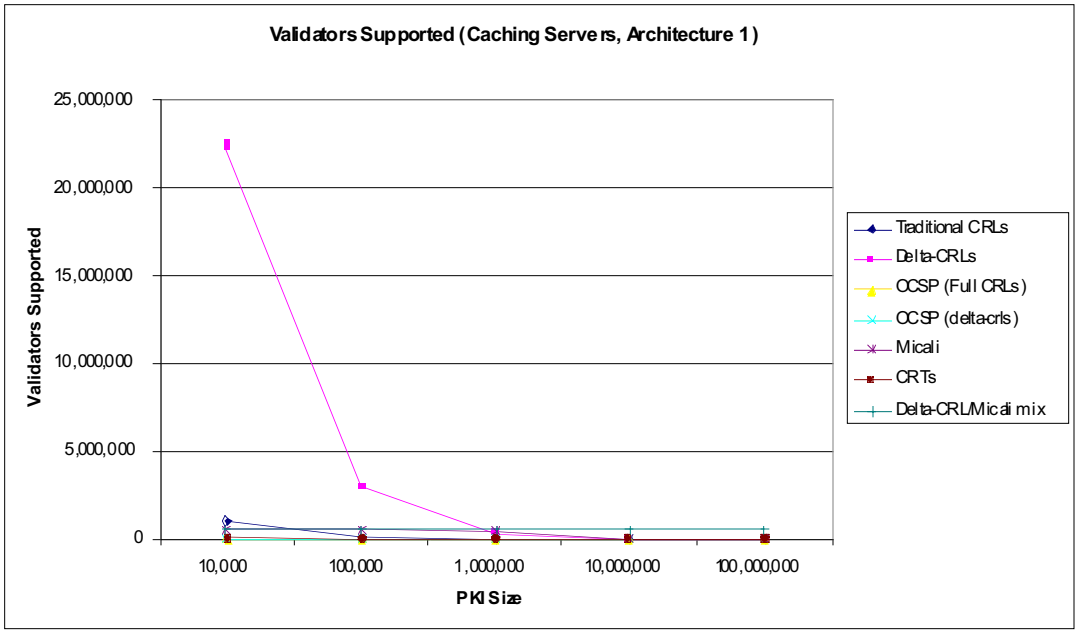

Figure 16: Maximum Number of Caching Servers Supportable – Architecture 1

It is apparent that caching gives δ-CRLs a marked advantage over all other methods, especially at small PKI sizes. On the other hand, although one would expect similar effects for traditional CRLs, their larger size negates this possible advantage. Table 10 summarizes the results for caching validators.

Table 10: Caching Validators

Architecture	Most clients supported by:	Most servers supported by:
Third-party (1), Associated (2), CA-repository (4)	• Micali • δ-CRL/Micali Hybrid	**Small PKI**: δ-CRLs **Large PKI**: • Micali, • δ-CRL / Micali hybrid
Local Repository (3), Validator-Repository (5)	• δ-CRL/Micali Hybrid	• δ-CRL • δ-CRL / On-line • δ-CRL / Micali hybrid

5 Recommendations

As can be seen from Table 9 and Table 10 above, the δ-CRL/Micali hybrid is consistently among the most efficient validation methods. The only exception to this observation occurs in a Type 1 architecture (one in which the number of Repositories remains constant). There more servers can be supported with straight δ-CRLs—provided they cache validation replies.

Hence, we recommend that:

- Certificate Authorities place final Y and N values in each certificate issued. CAs use δ-CRLs to send updates to repositories. They also send to Repositories (in some secure way) the initial secret Y_0 and N_0 Micali values for each newly issued certificate.
- Repositories update the current Y_i and N_j values for each certificate as the δ-CRLs are received. They respond to validation queries with the current Micali hash values. Furthermore, Repositories in a Type 1 architecture with small δ-CRLs[15], should also make the original δ-CRLs available.
- If the choice of either Micali hash values or δ-CRL is made available, then each validator can choose the method most appropriate to its individual circumstances. Thus, caching servers can choose the δ-CRL while all others should can the Micali hash pair.

It should be noted that these recommendations assume that efficiency is the sole basis for choosing a validation method. Other considerations include the following.

- The above recommendations assume that the Repositories are trusted. We consider the case of the untrusted Repository in Appendix A.
- As mentioned in Section 2, the Micali hash chain method has the nice feature that all values possibly needed for validation can be calculated ahead of time. In our recommendations above, we assume that for optimum efficiency the Repositories will calculate and store these valued for each new certificate. However, this might consume a large amount of storage. If a certificate has a lifetime of one year and its validation status is updated daily, at generation it will require the storage of 730 hash values—dwindling to 365 hash values just before expiration. Assuming the use of SHA-1, hashes are 160 bits long. Each certificate requires, on average, the storage of 548 hash values. A PKI with 1,000,000 certificates would require each Repository to devote 11GB to storage of these hash values—before overhead.[16]

---

[15] The threshold value for δ-CRL size will depend on many factors, and will probably have to be adjusted periodically for optimum performance.

[16] Some minor optimization can be done to reduce this number. Under the assumption that most certificates are never revoked, the repository can store only the Y_k values. If a certificate is revoked, it can then delete the unused Y_ks, and replace them with the appropriate N_j values. In this manner, some re-calculation will have to be re-done for a small percentage of certificates, but storage demands are reduced to 365 hashes at certificate generation and 0 hash values at

5.1 Conclusion

We have examined several different certificate validation methods in several different web architectures. We have considered the use of caching of validation information wherever it is appropriate. A spreadsheet analysis of the maximum number of Validators that can be supported in each case has produced some interesting results.

For updating the validation information from the CAs to the Repositories, either δ-CRLs or Micali Certificate Revocation Status appears to offer maximum capacity. For responses to validation queries, Micali appears to be the route to take. Our own suggestion of a combination of CRLs and Micali Certificate Revocation Status yields the best performance in those architectures where it is reasonable.

---

the point of expiration. Thus the average certificate will require the storage of 183 hash values, requiring 3.65GB of storage at the Repository (again, before overhead).

6 Appendix

Non-Trusted Repositories

The above discussion assumes that trust in the Repository (as required by the on-line method and the δ-CRLs / Micali hybrid) is not an issue. Should external considerations dictate that the Repository not be trusted, the picture changes somewhat.

In this case, Micali is still best at handling validation responses, and δ-CRLs are still best for updating validation information. However, both of these methods become prohibitively expensive as the size of the PKI increases. Of the two remaining methods, On-line methods and CRTs, it appears that CRTs perform better. See Tables 11 and 12.

Table 3: Non-caching Validators, untrusted directory

Architecture	Small PKI	Large PKI
Third-party (1), Associated (2), CA-repository (4)	• Micali	• CRTs
Local Repository (3), Validator-Repository (5)	• δ-CRLs	

Caching, of course, will help, but only under the same circumstances as before.

Table 4 – Caching Validators, untrusted directory

Architecture	Small PKI	Large PKI
Third-party (1), Associated (2), CA-repository (4)	• Client: Micali • Server: δ-CRLs	• CRTs
Local Repository (3), Validator-Repository (5)	• δ-CRLs	

References

[BERK] Berkovits,Shimshon, Santosh Chokhani, Judith A. Furlong, Jisoo A. Geiter, Jonathan C. Gould, *Public Key Infrastructure Final Report*, The MITRE Corp., Oct., 1994. Available at http://csrc.nist.gov/pki/documents/mitre.ps

[KOCH] Kocher, Paul, *A quick introduction to Certificate Revocation Trees*, 1998, Available at http://www.valicert.com/resources/

[MIC] Micali; Silvio, *Certificate revocation system*, United States Patent No. 5,666,416, Sept. 9, 1997

[NOAR] Naor, Moni and Kobbi Nissim, , Certificate Revocation and Certificate Update. 7th USENIX Security Symposium, 1998. Available at http://www.wisdom.weizmann.ac.il/~naor/onpub.html

[OCSP] Myers, Michael, Rich Ankney, Ambarish Malpani, Slava Galperin, Carlisle Adams, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP P*, Sep., 1998, <draft-ietf-pkix-ocsp-07.txt>, Available at http://www.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-pkix-ocsp-07.txt

[OOR] Van Oorschot; Paul C., Warwick S. Ford, Stephen W. Hillier, Josanne Otway, *Method for efficient management of certificate revocation lists and update information*, United States Patent No. 5,699,431, Dec. 16, 1997.

[PERL] Perlman; Radia J.and Charles W. Kaufman, *Method of issuance and revocation of certificates of authenticity used in public key networks and other systems*, United States Patent No. 5,261,002, Nov. 9, 1993.

[X509] Housley, R., W. Ford, W. Polk, D. Solo, *Internet X.509 Public Key InfrastructureCertificate and CRL Profile*. <draft-ietf-pkix-ipki-part1-11.txt>, Available at http://www.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-pkix-ipki-part1-11.txt

****

The Entrust paper on which our model was built:

T. Moses, *Scalability of Public Key Infrastructures*, May, 1997.