Windowed Key Revocation in Public Key Infrastructures

Patrick McDaniel Sugih Jamin
Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48109-2122
{*pdmcdan,jamin*}*@eecs.umich.edu*

October 12, 1998

Abstract

A fundamental problem inhibiting the wide acceptance of a Public Key Infrastructure (PKI) in the Internet is the lack of a mechanism that provides scalable certificate revocation. In this paper, we propose a novel mechanism called *Windowed Revocation*. In windowed revocation, certificate revocation is announced for short periods in periodic Certificate Revocation Lists (CRLs). Due to the assurances provided by the protocol over which certificates are retrieved, we bound the amount of time that any certificate is cached by users. Thus, we can limit the announcement of revocation only to the time in which the certificate may be cached; not until its expiration. Because the time in which certificate are announced is short, CRLs are similarly small. By limiting the size of CRLs, we are able to integrate other mechanisms that increase the scalability of the PKI. One such mechanism is the use of "pushed" CRLs using multicast. We include a proof of the correctness of our approach.

1 Introduction

Over the past several years, the use of distributed applications has grown immensely. These applications allow geographically distant users to communicate, leading to social, educational, and commercial interactions that were previously impossible. Unfortunately, because of the openness of the Internet, the form and content of these interactions is vulnerable to attack. Limiting these vulnerabilities is essential to the future success of these applications.

A popular approach to securing communication over large networks is to use public keys. Researchers and standards bodies have argued at great length over possible architectures for providing an authentication service under which public key certificates can be securely distributed. A central point of contention in these discussions is the mechanisms over which public keys are revoked.

A *certificate* is a data structure that defines an association between an entity (the *principal*) and a public key. A trusted authority, called a *Certificate Authority* (CA), states its belief in the validity of the association by digitally signing the certificate. Certificate revocation is the mechanism under which a CA can revoke the association before its documented expiration. The CA may wish to revoke a certificate because of the loss or compromise of the associated private key, in response to a change in the owner's access rights, or strictly as a precaution against cryptanalysis. As stated by the CA, the *revocation state* of a certificate indicates the validity or cancellation of its association. A *verifier* determines the revocation state through the *verification* of the certificate.

In this paper we investigate *windowed revocation*, a novel approach to certificate revocation within a global certificate distribution service, called a *Public Key Infrastructure* (PKI). The central design objectives of windowed revocation are:

1. **Correctness** - All entities within the PKI must be able to correctly determine the revocation state of a certificate within well-known (time) bounds.
2. **Scalability** - The costs associated with the management, retrieval, and verification of certificates should increase at a slower rate than the size of the serviced community.
3. **Flexibility** - Windowed revocation must be able to support mechanisms consistent with existing security policies and requirements.

As with many security solutions, certification revocation mechanisms are subject to the fundamental tradeoff between security and scalability. Solutions with strict security objectives require more resources than systems with more relaxed security objectives. Thus, security requirements have a direct influence on scalability. Our proposed architecture provides a flexible framework for managing this tradeoff by incorporating the following design principles into the key revocation mechanism:

1. *Revocation window:* By bounding the time over which the revocation of a certificate is announced, we limit the size of such announcements.

2. *Push delivery:* With limited revocation announcement size, we can contemplate the active delivery of this information to verifiers. This reduces the load on the CAs by curtailing the number of verifier initiated retrievals.

3. *Certificate caching:* A cached certificate may be used until it expires, is revoked, or the issuer specified TTL is reached. The expiration of a time-to-live indicates that the associated entity's policy requires the certificate to be re-validated.

4. *Scheduled Announcement:* By stipulating that CAs generate revocation announcements at a documented schedule, we allow verifiers to detect lost announcements.

5. *Multicast delivery:* Given verifiers' ability to detect missing revocation announcements, we can use unreliable transport protocol without sacrificing the security of certificate revocation. This allows us to use IP multicasting, where available, to further reduce the bandwidth requirements of the revocation mechanism.

6. *Lazy verification:* Verification of a cached certificate's revocation state may be postponed until the certificate is to be used.

7. *Revocation aggregation:* Revocation announcements from multiple sources are aggregated by higher level authorities.

In the next section, we discuss the design tradeoffs of revocation mechanisms in general and outline the advantages of our windowed revocation mechanism over other approaches proposed in the literature. In Section 3 we describe the working of windowed revocation and provide a formal proof of the correctness of the mechanism. Section 4 discusses protocol issues and presents windowed revocation as a X.509 v3 [HFPS98] extension. Section 5 gives a brief overview of related work. We conclude this paper in Section 6.

2 Design Tradeoffs

We recognize two fundamental approaches used to distribute revocation state: explicit and implicit. Systems using explicit revocation require all parties to verify the state each time a certificate is used. In X.500 based systems, such as Privacy Enhanced Mail (PEM) [Ken93], each CA periodically generates a list of certificates that have been revoked, but have not yet expired. The presence of the certificate in the list,[1] called a *Certificate Revocation List* (CRL), explicitly states revocation.

Verifiers retrieve and cache the latest CRL during the certificate verification process. Thus, the frequency with which the CA generates CRLs bounds the time in which a revoked certificate can be used. A revoked certificate is included in a CRL from the time it is revoked until it expires. Because the length of time a certificate may be valid is commonly measured in years, CRLs can become large. In an effort to reduce the costs of CRL processing, some systems present revocation information in authenticated dictionaries [NN98, Koc98, Mic96]. Using authenticated dictionaries, verifiers interactively construct a proof of the presence or absence of the certificate in the CRL. They need not retrieve the entire CRL, but request only enough information to validate the certificate. However, these approaches are not without cost; they often involve heavyweight cryptographic operations, long interactive protocols, and/or significant CA resources.

In PKI architectures that employ implicit revocation, the revocation state is implicitly stated in a verifier's ability to retrieve the certificate. Any certificate retrieved from the issuing CA is guaranteed to be valid at the time of retrieval. Associated with each certificate is the TTL which represents the maximum time the certificate may be cached. This bounds the time that a revoked certificate may be used without detection. The Secure DNS (DNSSec) [Gal96, EK99] architecture uses implicit key revocation.

A central parameter to PKIs employing implicit revocation is the length of the certificate TTL. PKI administrators must trade-off security (as stated by the bound on revoked certificate use) with the frequency of retrieval. A long TTL may expose the verifier to a revoked certificate. A short TTL requires the verifier to retrieve the certificate frequently, thus limiting the scalability of the PKI. In extant systems, each retrieval requires heavyweight operations by the verifier, the CA, or both.

Windowed revocation uses a hybrid of both explicit and implicit revocation. Similar to explicit approaches, windowed revocation uses CRLs to announce revocation. CRLs are generated at a documented rate, and revocation is indicated by the presence of the certificate's associated serial number. Similar to implicit approaches, windowed revocation requires the successful retrieval of a certificate to implicitly state the validity and freshness of the certificate. Also similar to implicit approaches, windowed revocation allows verifiers to re-acquire certificates at frequencies commensurate with their security requirements.

[1] The entire certificate is generally not present in the list, but is referenced by some unique identifier. This identifier is commonly known as a serial number.

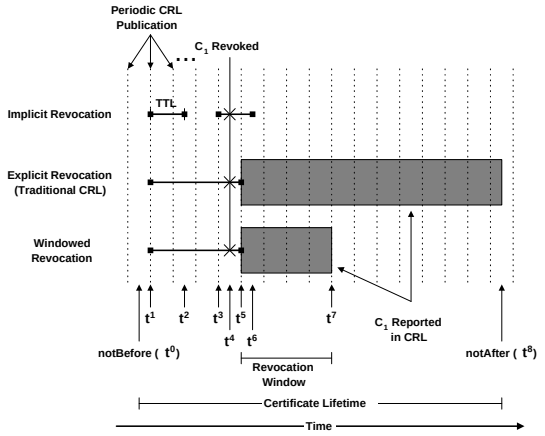

Figure 1: Implicit, explicit, and windowed revocation in PKI architectures.

Different from implicit approaches, windowed revocation does not require re-acquisition of certificates at fixed intervals. Instead, windowed revocation allows for the freshness of a certificate to be re-asserted with each statement of its validity via CRL. Different from explicit approaches, windowed revocation limits the period over which a certificate's revocation is announced. In windowed revocation, the size of a certificate's revocation window is assigned by the issuing authority and is documented within the certificate. By bounding the time that each revoked certificate must be included in the periodic CRLs, we reduce the size of each individual CRL. Because of the small CRL size, we can actively deliver CRLs to verifiers.

We illustrate implicit, explicit, and windowed revocation in Figure 1. In the figure we show the lifetime of a certificate C_1, which has a documented validity period from `notBefore` (t^0) to `notAfter` (t^8). At time t^4, C_1 is revoked. Assume C_1 is verified at times t^1 and t^3 in each example.

In implicit revocation, the user securely retrieves and caches C_1 at time t^1. No further verification is performed between t^1 and t^2. After the freshness TTL expires at time t^2, the certificate is dropped. The certificate need not be re-acquired until it is needed again at time t^3. Because verification is performed only during retrieval, the revocation of C_1 will not be discovered until it is dropped at time t^6 and re-acquired afterward. We call the bound on the longest time a revoked certificate may be used the *window of vulnerability*. For implicit revocation, the window of vulnerability is exactly the freshness TLL ($t^6 - t3$).

In explicit revocation, the certificate and last generated CRL is retrieved at time t^1. Each subsequent use (t^3) of the certificate requires that the most recent CRL be checked for a revocation announcement. Because a cached certificate is only authenticated as required by use, there is no bound on the time in which a CRL will be retrieved by the user. Therefore, the CA must announce the revocation from the CRL immediately following the revocation until the certificate expires (t^5 to t^8). Because CRLs are the only medium from which revocation state can be obtained, the window of vulnerability in explicit revocation is equal to the periodicity of CRL publication (see Section 3.4 for a correctness proof).

Windowed revocation bounds the time at which a certificate may be cached through the *revocation window*. When the certificate is retrieved (t^1) it is guaranteed to be fresh and unrevoked. After revocation (t^4), the CA need only include the certificate in the CRL for the revocation window (t^5 to t^7). The CA knows that one of the following cases occurred for every host caching the certificate: 1) a CRL was received within the revocation window, and C_1 was dropped, or 2) the revocation window has expired, and C_1 was dropped. In either case, windowed revocation stipulates that the certificate will no longer be cached by any host at the end of the revocation window, hence the CA can discontinue announcing the revocation. After the revocation window has been reached, the CA may remove the revoked certificate from its internal lists. No master list of revoked certificates is required. Similar to explicit revocation, the window of vulnerability in windowed revocation is equal to the periodicity of CRL publication.

For reasons of policy or inter-operability, a CA may wish to provide exclusively implicit or explicit revocation. These requirements can be met by the proper manipulation of the revocation window. By setting the revocation window equal to or greater than the validity period of a certificate, explicit revocation can be achieved. A converse manipulation of the window yields strictly implicit revocation. We detail the operation and implications of revocation window configuration in Section 3.5.2.

3 Architecture

In this section we describe the design and operation of our key revocation mechanism. For investigative and illustrative purposes, we define a simple Public Key Infrastructure architecture called Key Distribution Hierarchy (KDH). While we study the operation of windowed revocation within KDH, windowed revocation is not dependent on KDH.

3.1 Key Distribution Hierarchy

The hierarchy of KDH is similar to the ICE-TEL [CY97] PKI, but avoids many of the complexities of its construction. We provide a more thorough comparison of KDH and ICE-TEL, as well as a thorough description of

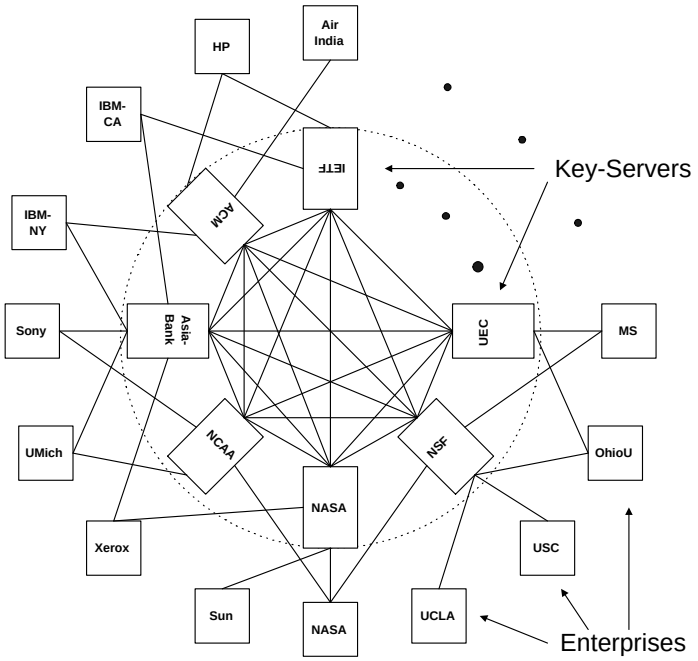

Figure 2: Internet Level Architecture

the architecture, the certificate retrieval protocol, and related policy issues in [MJ98].

KDH introduces a two level hierarchy consisting of the *keyserver* level and the *enterprise* level. The keyserver level contains a set of servers from which enterprise and keyserver certificates can be retrieved. The enterprise level contains independent hierarchies of end users. In ICE-TEL parlance, each keyserver corresponds to a PCA, and each enterprise corresponds to a security domain. Figure 2 describes an Internet-centric view of one possible configuration of the architecture. In the figure, a link between two entities represents an exchange of digital signatures, where each end-point signs and permanently caches the other's certificate. The exchange of certificates and signatures is called *registration*.

KDH stipulates that keyservers form a fully-connected graph of peers, where all keyservers have exchanged certificates with all others. By mandating a fully-connected graph, we limit the length of certification path used in the retrieval and verification of a certificate. An authenticated certificate of any keyserver can be retrieved from any other keyserver.[2]

[2]The requirement that all keyservers exchange signatures is used to bound the transitivity of trust during certificate authentication. The effect of relaxing this requirement would be the introduction of additional intermediate keyservers into the authentication process (certification path), which may lower confidence in the process. In the degenerate case, the retrieval process would become similar to authentication in the PGP [Zim94] system.

Enterprises register with keyservers using an out-of-band channel. It is from these keyservers that the enterprise later retrieves authenticating certificates. In essence, the exchange of signatures between a keyserver and an enterprise states that the enterprise trusts the keyserver to advertise correct certificates. However, this trust need not be absolute. Later, during authentication, multiple keyservers may be consulted.

Keyservers are intended to be administered independently by regional, national, or global organizations. In terms of hardware and administrative practices, these servers should have many of the same characteristics as those defined for the PCA services in RFC 1422 [Ken93]. These practices define procedures used for mutual authentication before enterprise registration. An enterprise provides its certificate to each keyserver with which it wishes to register. After appropriate mutual validation of credentials, the keyserver signs and caches the enterprise certificate and the enterprise root signs and caches the keyserver certificate. A thorough description of the use of digital signatures can be found in [DH76].

Each enterprise encompasses some organization of end users. The enterprise is intended to represent a group of geographically close local area networks under control of a single administrative authority. A distinct host, called the enterprise root, is logically the single point of contact for requests for certificates of the enterprise. The enterprise root corresponds to the organizational certification authority (CA) in the ICE-TEL systems. We stipulate that

each enterprise contains only one enterprise root. In larger enterprises, it may be necessary to replicate this service.

As determined by need, users and hosts may belong to multiple enterprises. For example, users may belong to different enterprises in which they perform professional and personal related activities. All certificates for entities within an enterprise are permanently stored at the enterprise root. When a local host registers its public key with the enterprise, they mutually authenticate and sign each other's certificates. When an external entity requests a certificate for one of these hosts, the enterprise root will respond with the stored certificate. If the root is properly placed (e.g. at a network border), very little traffic should be generated by external requests on the enterprise network.

Hosts internal to the enterprise directly contact the local service (enterprise root) to make requests for internal or external certificates. Retrieved certificates are cached at the enterprise root and each end user host. Detection of the revocation of cached certificates is described in Section 3.3.

While in the preceding architectural overview we have described each CA as a single entity, in practice it consists of two components: a *CA*[3] and a *directory* service [BAN90]. The CA performs the mission critical duties of certificate signing and CRL generation, communicating only with the directory service. The directory service acts as the distribution point for certificates and CRLs. When retrieving certificates, verifiers assume complete trust in the CA, and a limited form of trust in the directory service. The directory is trusted to correctly advertise certificates and CRLs, and the CA is trusted to comply with procedures outlined in its policy statement. We see policy compliance failures [Dav96] as orthogonal to our investigation. For ease of exposition and without loss of correctness, we continue to treat the CA and directory as a single logical entity in the remainder of this paper.

3.2 Certificate Retrieval Protocol

As is the case with most PKIs, certificate retrieval in KDH is accomplished by the collection and authentication of signed certificates. The verifier logically traverses a graph representing signature exchanges between the enterprises and keyservers, collecting certificates at each hop. Each certificate's signatures is verified and the appropriate CRLs are consulted. If all certificates are authentic and unrevoked, the user is free to use them. We now present a step by step description of this process.

[3]In KDH, both keyservers and enterprise roots perform CA duties, but the type of certificates managed and the generation of CRLs differ. Throughout this paper, we use the term CA only when the context applies to both keyserver and enterprise root.

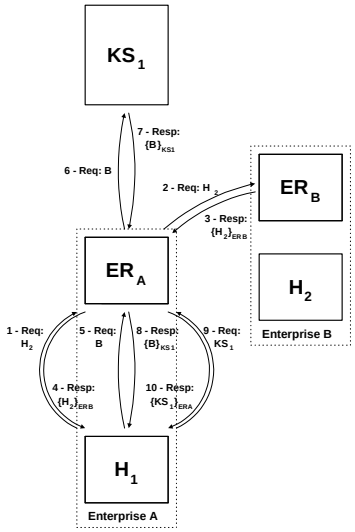

Figure 3: The certificate request process.

Each enterprise root node begins operation with permanent entries for the certificates of entities within the enterprise, the enterprise certificate, and the certificates of each keyserver with which it has registered.

When an enterprise root node receives an external request for a certificate belonging to an entity within the enterprise, it returns the certificate and a list of keyservers with which it has exchanged signatures. The list of keyservers associated with the enterprise is always cached with the certificate.

When a verifier request cannot be serviced by the local host cache, the request is forwarded to the enterprise root node. If the request is for a certificate external to the enterprise, it is forwarded by the enterprise root to the external enterprise. The response is cached and returned to the requesting host. A similar process is used for keyserver certificates, with the originating verifier specifying from which keyserver it wishes to retrieve the certificate.[4]

It is worth noting that we do not specify a mechanism for locating the enterprise root node of an external enterprise. There are several existing designs for scalable network directory services, such as DNS [Moc87a, Moc87b]. These services are readily available within today's Internet infrastructure, and as such are beyond the scope of this paper.

We illustrate the retrieval process through an example in Figure 3. Assume all nodes initially have empty caches, save the permanent entries. We state that the enterprise root nodes ER_A and ER_B have exchanged signatures with keyserver KS_1. In Figure 3, we show the request process used by enterprise host H_1 in enterprise A to ob-

[4]We note the possibility of reducing the number of round-trips during the retrieval/verification process by consolidating requests. For clarity, the operational descriptions below will treat each request independently.

tain and authenticate the certificate of a host H_2 in enterprise B. H_1 begins by requesting from ER_A the certificate of H_2 (step 1 in Figure 3).

ER_A forwards the request to ER_B, returning the results to H_1 (steps 2-4). H_1 then determines that the certificate of ER_B is needed, and repeats the request process, specifying that the certificate be retrieved from the keyserver KS_1 (steps 5-8). Based on the keyserver information returned in the H_2 request, H_1 notes that both enterprises shared the keyserver. As stated in the local host policy, H_1 determines that this is an acceptable relationship because they share a common keyserver, which it trusts. Finally, H_1 requests and receives the certificate for keyserver KS_1 (steps 9 and 10 in Figure 3). Having assembled all the certificates, H_1 recursively authenticates the digital signatures. Based on the results of the authentication, H_1 may initiate some secure action using the certificate.

In [MJ98], we discuss the cases when the enterprise of a verifier host and the enterprise of the requested certificate do not share a common keyserver (in terms of registration) and when more than one certificate for a single target is received with valid signatures. For brevity, we do not include the discussion of these cases here.

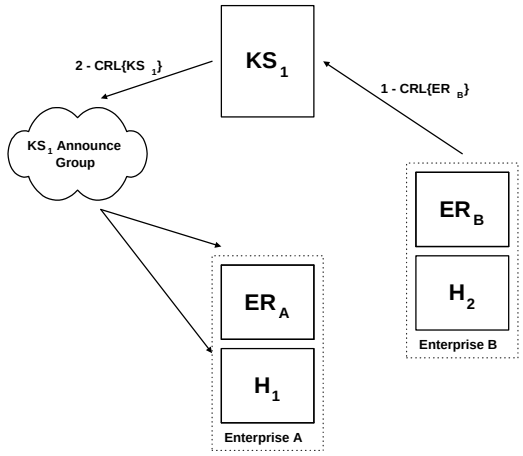

Figure 4: Certificate revocation delivery. After its revocation, certificate C_{H_2} is included in subsequent CRLs generated for the local enterprise (B). Each CRL is reliably unicast by the enterprise root (ER_B) to all keyservers with which the enterprise has registered (KS_1). The enterprise CRL is summarized (with CRLs from other enterprises) and included in the keyserver CRL. The resulting keyserver CRL is multicast to all interested parties.

3.3 Certificate Revocation Protocol

In windowed revocation, we use explicit notification as the primary revocation mechanism. CRLs are generated per the schedule documented in the associated certificate. These CRLs are then delivered on keyservers' announcement groups. We require each entity holding a cached certificate to listen for revocation announcement from the corresponding keyserver. We explore two other CRL distribution mechanisms and evaluate their potential scalability problems in Section 3.5.1.

The generation and delivery of CRLs from source enterprise to verifier host is demonstrated through the following example. The key distribution hierarchy used in the previous example is depicted in Figure 4 along with the keyserver's announcement group. The hierarchy consists of a keyserver KS_1, two enterprises (ER_A, ER_B), and two hosts (H_1 of enterprise A and H_2 of B).

Continuing with the example in the previous section, at some point after host H_1 acquired certificate C_{H_2}, C_{H_2} is revoked. Subsequent to the revocation of C_{H_2}, requests for H_2's certificate will return either a newly generated C'_{H_2} (with a unique serial number), or an error if no new certificate for H_2 has been created. Whether a new certificate for H_2 is generated or not, the next scheduled CRL from ER_B will include the revocation of the old C_{H_2}.

Each CRL generated by ER_B is reliably unicast to all keyservers with which it has registered, which in this example is only KS_1 (step 1). The keyserver KS_1 stores the CRL from enterprise B in preparation for the publication of the next keyserver CRL (see Section 3.3.1).

When the next keyserver CRL is generated, the CRL from enterprise B containing the revocation of C_{H_2} is included. The keyserver then multicasts the CRL over the keyserver *announcement address* (step 2). The scalability of traditional PKIs is limited by the requirement that verifiers actively retrieve CRLs. We use push delivery in windowed revocation to enable passive verification. If a pushed CRL is lost in transit and it is required by a verifier, the verifier may retrieve it from the CA (or refresh the certificate by re-acquiring it). Hence CRL delivery may use unreliable transport protocol, such as IP multicasting. Note that the use of unreliable transport protocol does not affect the security of CRL delivery (see Sections 3.5.1 and 4.1).

Revoked certificates are included in the scheduled CRLs for a period equal to its *revocation window*. The revocation window of each certificate is documented in the certificate. The revocation window limits the length of time a certificate may be cached without the holder of the cached certificate receiving an associated CRL. Because revocation is explicitly stated in the CRL only for this period, the verifier will have no means of determining the correct revocation state afterwards. Therefore, if a verifier does not receive an associated CRL during the revocation window, it must drop the certificate from its cache.

When the CRL associated with a certificate cannot be obtained, the certificate must be re-acquired. As CAs are prohibited from advertising revoked certificates, and the retrieval process is freshness protected (see Section 4.1), all retrieved certificates are guaranteed to be both fresh and unrevoked. Therefore, if a recent CRL cannot be obtained, the revocation state can be determined by the direct acquisition of the certificate.

By providing low cost delivery of CRLs in the average case (multicast keyserver CRL delivery), we avoid the vast amount of active CRL retrievals normally associated with traditional PKI architectures. In the aberrant case, where the most recent CRL has not been received, we provide a means of recovery through direct retrieval.

The *CRL publication period* and *revocation window* are documented as additional fields in all certificates within the PKI. The CRL publication period is the length of time, in minutes, between each new CRL publication. The revocation window is the number of CRL publications in which a revocation is included. Additionally, keyserver certificates contain a *CRL announcement address*. The CRL announcement address is the identity of the group over which CRLs are delivered (see Section 4.2).

In the following sections, we outline the Windowed Revocation protocol and supporting features. The next two sub-sections describe CRL generation and distribution within KDH. We conclude this sub-section by outlining the cache management policy.

3.3.1 Keyserver CRL Generation

Traditional CRL revocation requires hosts wishing to validate certificates from potentially many CAs to retrieve and validate as many CRLs as the number of CAs involved. In attempting to address this and other limitations, the IETF Public Key Infrastructure Working Group (PKIX) provides the Indirect CRL extension [HFPS98]. Using Indirect CRLs, a CA may delegate CRL generation to other entities. We extend this approach by stipulating *a priori* indirect CRLs. Keyservers aggregate CRLs by collecting all the CRLs of enterprises that have registered with them. After the authenticity of each enterprise CRL has been verified, the enterprise revocation information is incorporated into the keyserver CRL. By allowing the keyserver to authenticate enterprise revocation information, verifiers need not collect or verify each enterprise CRL.[5]

Each keyserver generates CRLs at the documented CRL publication period. The keyserver CRL contains revocation state of certificates belonging to enterprise roots that have been registered with the keyserver, summary information of CRLs from registered enterprises, and a digital signature calculated over the previous fields. The keyserver delivers its CRLs to all interested parties over its announcement address.

To reduce the window of vulnerability in which a certificate holder may not have learned of a certificate's revocation and thus continues to use the revoked certificate, we expect a keyserver's CRL publication period to be significantly smaller than the CRL publication periods of the ERs registered with it.

If the keyserver does not have the most recent enterprise CRL (whose announcement schedule is documented in the enterprise certificate), this fact is noted in the keyserver CRL. The only scenario in which the keyserver will not have the most recent CRL is when the enterprise root experiences a process or communication failure.

We note the possibility of keyserver supported *Freshness CRLs*. CAs supporting Freshness CRLs [AZ98] generate CRLs at differing frequencies. Users retrieve the CRL with a publication rate commensurate with their needs. In extending this approach, a keyserver may support several announcement groups with different CRL publication rates.

Finally, we consider the special case of keyserver certificates revocation. Each keyserver is the root of a portion of the PKI hierarchy, and as such has no higher authority to announce its revocation. This makes dealing with a compromised keyserver private key difficult. One popular solution is to have a single highly protected root CA. We believe that locating a single source of trust for all users in the Internet is problematic, if not impossible. In our architecture, we assume an out-of-band method for contacting registered enterprises after keyserver certificate is compromised. In addition to out-of-band revocation, all keyservers self-revoke their own certificates. That is, each keyserver wishing to revoke its own key will include it in subsequent CRLs. This may aid the quick distribution of the revocation notification.

3.3.2 Certificate Cache Management

The operation of the cache at either enterprise root or end-user hosts is dependent on the ability of the host to retrieve CRLs. Hosts which consistently retrieve or receive CRLs may cache and use certificates as needed. When these CRLs cannot be reliably obtained, the host must actively authenticate each certificate.

We present the following algorithm used by the verifier to determine the revocation state of a cached certificate. In the following text, a distinction is made between the *last published CRL* and the *last received CRL*. The last published CRL is the last CRL generated by the CA previous to the verification of the certificate. The last received CRL is the last CRL received by the verifier.

[5] As policy dictates, the verifier may wish to verify domain CRLs directly. In the absence of this, the verifier must trust the keyserver to correctly perform this task.

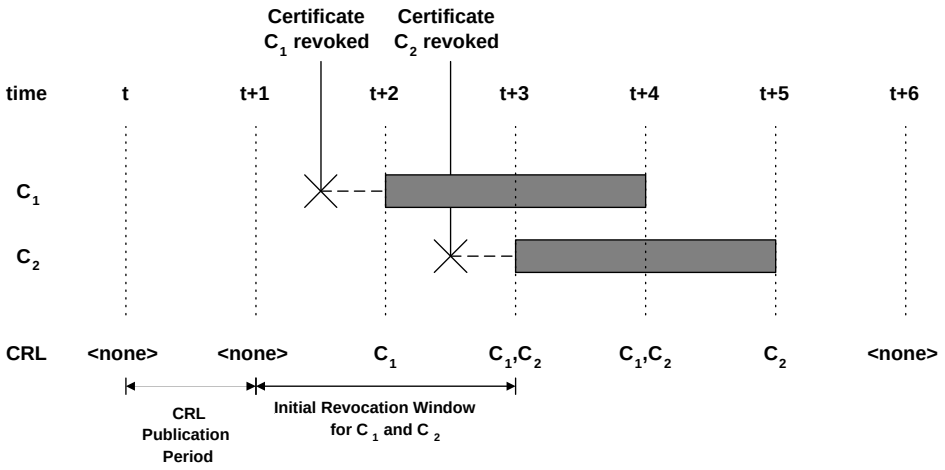

Figure 5: Example CRL generation - In this example, we show the revocation of certificates C_1 and C_2 and their inclusion in subsequent CRLs.

1. If the last published CRL has been received from the CA and the certificate has not been revoked, it can continue to be used.

2. If the last published CRL has not been received:

   (a) If the difference between the current time and the last received CRL is less than the revocation window, the last published CRL is retrieved. Once retrieved, the CRL is used to determine the revocation state of the certificate.

   (b) If the difference between the current time and the last received CRL is greater than the revocation window, the certificate is dropped and must be re-acquired. The expiration of a certificate window indicates that revocation announcements for the associated certificate may have been lost.

   (c) If the last published CRL cannot be retrieved, the certificate is dropped from the cache, and must be re-acquired from the CA.

At the time of retrieval, two timers are associated with each cached certificate. For host and enterprise certificates, the *clean timer* is set to the CRL publication period of the enterprise (p) plus the publication period of the keyserver ($p\prime$). This ensures that all hosts listening to the keyserver announcement address receive keyserver CRLs before the clean timers expire. The *revocation window timer* is set to the revocation window (w) multiplied by the enterprise CRL publication period. The time of the enterprise CRL publication is denoted t^{CRL}. As CRLs arrive, the clean timer associated with each un-revoked certificate are reset to $t^{CRL} + p + p\prime$. After receiving a CRL, revocation window timer is reset to $t^{CRL} + wp$. Revoked certificates are removed from the cache.

As clean timers expire, the associated entries are marked "dirty". In the normal case, keyserver CRLs are received regularly, and cached certificates will never be marked dirty. Certificates not marked dirty were not revoked at the time the last CRL was generated, and may continue to be used.

When a dirty certificate is requested by a verifier and the certificate's revocation window timer has not expired, the host attempts to validate the certificate by retrieving the most recent CRL. If the CRL is successfully retrieved, all relevant cache entries are updated, and the certificate is returned to the end-user. If the CRL cannot be retrieved, the entry is dropped from the cache, and must be re-acquired using the certificate retrieval protocol described in Section 3.2).

If the revocation window timer of a certificate has expired, hosts can not determine the revocation state of this certificate using the latest CRL. In this case, the certificate is dropped from the cache, and must be re-acquired.

We now illustrate the certificate cache management process with an example. In Figure 5, we describe a series of events involving a certificate caching host. In this example, the CRL publication period for the CA associated with certificates C_1 and C_2 is equal to 1 (where a CRL is generated at $t, t+1, t+2, \ldots$).

The revocation window documented in each certificates C_1 and C_2 is 2 (periods). Between $t+1$ and $t+2$, certificate C_1 is revoked. Between $t+2$ and $t+3$, certificate C_2 is revoked. The CRLs published by the CA at time $t+2$ and $t+3$ will contain the revocation of certificate C_1, while the revocation state of certificate C_2 will be in-

cluded in the CRLs published at time $t+3$ and $t+4$. The CRL published at time $t+4$ will no longer contain the revocation state of certificate C_1. Should a host try to retrieve a CRL from the CA between time $t+3$ and $t+4$, the CRL returned will be the one published at time $t+3$, which included the revocation of certificate C_1. This period of inclusion of a certificate revocation state is represented in Figure 5 as grey boxes.

Consider an end-user host whose cache contains both certificates C_1 and C_2. Assume that the host received the CRL published at time $t+1$. Thus at time $t+1$, the host set the revocation window timer for both C_1 and C_2 to $t+3$. We now describe three possible scenarios relating to this example.

If all CRLs are successfully received, C_1 will be removed in response to the CRL at time $t+2$, and C_2 will be removed from the cache in response to the CRL at time $t+3$.

If the CRL at time $t+2$ is not received and certificate C_1 is accessed by an end-user between $t+2$ and $t+3$, an attempt to retrieve the CRL directly from the keyserver or enterprise root will occur. If this process fails, the host will drop and re-acquire the certificate. Section 3.2.

In the case when both CRLs at time $t+2$ and $t+3$ are lost and cannot be retrieved, the host is unable to determine the revocation state of either C_1 or C_2. The revocation window timer for both certificates expires at time $t+3$, and the certificates are removed from the cache.

Now consider a second host who retrieves certificate C_2 at time $t+2$. It knows at the time of retrieval that C_2 is fresh and unrevoked, so it sets the clean timer to expire at $t+3$ and the revocation window timer to expire at time $t+4$. The certificate is handled as in the previous case, with the exception of the different timer expirations.

Note that while the size of the revocation window is the same in all hosts for a given certificate, the start time of the revocation window timer itself is not. In each host, the revocation window is reset each time the validity of a certificate is asserted.

We address the latencies incurred by the delivery of CRLs by stipulating that clean timers are set to the publication period plus a propagation delay value. The propagation delay is a short period (measured in milliseconds) that estimates the maximum time needed for the generation and delivery of the CRL. This value is site dependent, and must be set by the local network administrator.

3.4 Proof of Correctness

In this section, we formally prove the bound on the use of revoked certificates. In Figure 6, we describe the lifetime of certificate C. C is valid from time t^1 until its expiration at time t^n. CRLs are generated by the CA at the publication period p. In the proof we assume that the keyserver publication period ($p\prime$) is strictly less than the enterprise publication period (p) (see Section 3.3.1). We denote the time of an arbitrary CRL publication as t^c. At time t^i, C is retrieved and cached by an end user. At some time t^r, C is revoked. Each certificate defines a revocation window w, which states the length of time its revocation will be recorded in periodic CRLs. Before presenting the proof, we formally define two central properties of windowed revocation.

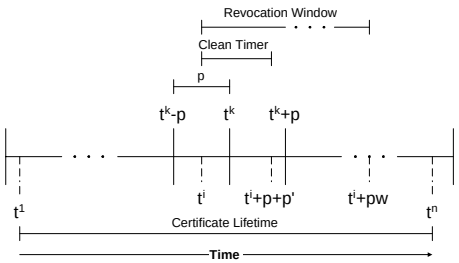

Figure 6: We show the lifetime of certificate host C, which is valid from t^1 to t^n. At time t^i, an end user retrieves the certificate. In response, the dirty and revocation window timers are set to $t^i+p+p\prime$ and t^i+wp, where p is the publication period of the enterprise, $p\prime$ is the publication period of the keyserver from which keyserver CRLs will be received, and w is the revocation window. The CA publishes CRLs at times $\ldots, t^k-p, t^k, t^k+p, \ldots$

Property 1 - *Fresh Certificate Retrieval* - This property ensures that all certificates are fresh and unrevoked at the time of retrieval. More formally, $t^r > t^i$ holds for the retrieval and revocation of any certificate C.

Property 2 -*Windowed Revocation* - This property ensures that all revoked certificates are included in the CRLs published within the documented revocation window. Formally,

$C \in CRL^j$ for all CRLs published at $t^c + mp$, where $\min(t^c)|t^c > t^r, 0 \leq m \leq w$.

Intuitively, t^c is the CRL publication time immediately following the revocation, i.e. the publication time of the first CRL that contains the revocation.

Theorem: *The length of time any revoked certificate may be used is bounded by the length of the clean timer ($p + p\prime$).*[6]

Proof: After retrieval, the initial clean timer for C is set to $t^i + p + p\prime$, and the revocation window timer is set to $t^i + wp$. It is sufficient to show the theorem holds for verifications (and use) of C at time t^δ, for all $t^\delta > t^i$.

[6]Note that the bound on the use of revoked keys is actually the clean timer length plus the propagation delay value. For simplicity and without loss of correctness, we omit mention of the propagation delay value.

- Case 1: $t^\delta \leq t^i + p + p\prime$: The certificate is verified before the initial clean timer expires.

  $t^\delta > t^i$, (*by definition*)
  $\Rightarrow t^i + p + p\prime - t^\delta \leq p + p\prime$,

  so the theorem holds.

- Case 2: $t^i + p + p\prime < t^\delta < t^i + wp$: The certificate is verified after the initial clean timer expires, but before the revocation window expires.

  a) If C is not marked dirty, then there exists some CRL^j published at time $t^j < t^r$ that was received by the host. At t^j, we know C has not been revoked. The clean timer has not expired, so $t^\delta - t^j < p + p\prime$.

  Therefore,

  $t^\delta - t^j < p + p\prime$, ($C$ *is not marked dirty*)
  $t^j < t^r$, ($C \notin CRL^j$)
  $\Rightarrow t^\delta - t^r < p + p\prime$.

  Intuitively, a certificate having an unexpired clean timer means that the certificate has not been revoked within $p + p\prime$ since the last CRL publication time, thus the theorem holds.

  b) If C is marked dirty and the most recent CRL^j published at time t^j is retrieved. If $t^r > t^j$, $C \notin CRL^j$, the clean timer is reset to $t^j + p + p\prime$, and this case reduces to case 2(a).

  If $t^r \leq t^j$, then it suffices to prove $C \in CRL^j$. By property 2, $C \in CRL^j$ if and only if

  $t^c \leq t^j \leq t^c + wp$,

  where t^c is $\min(t^c)|t^c > t^r$, the CRL publication on or immediately following t^r. From this, we can conclude that:

  $\Rightarrow t^c \leq t^j$,
  $t^i < t^r$, (*property 1*)
  $t^r \leq t^c$, (*property 2*)
  $\Rightarrow t^i < t^c$,
  $\Rightarrow t^i + wp < t^c + wp$,
  $t^j < t^i + wp$, (*from case definition*)
  $\Rightarrow t^j < t^c + wp$.

  Hence:

  $\Rightarrow t^c \leq t^j < t^c + wp$,

  and

  $\Rightarrow C \in CRL^j$.

  So the theorem holds. A similar argument holds for certificates whose revocation window is reset in response to a received CRL.

- Case 3: $t^\delta \geq t^i + wp$: The revocation window has expired, so the certificate is dropped. Thus, the theorem holds. □

3.5 Design Evaluation

In this section we evaluate our approach to key revocation in terms of our two other stated design goals of scalability and flexibility.

3.5.1 Scalability

Windowed revocation is scalable both in its bandwidth requirements and the size of the supported community. As indicated in Section 1 and throughout the paper, the scalability of windowed revocation is based on its use of the revocation window and CRL push delivery. By limiting the size of CRLs through the use of the revocation window, we reduce the costs associated their distribution.

Through certificate caching, we attempt to maximize the total number of supportable verifiers. Moreover, we use the CRL publication as a form of cache invalidation protocol. Given our reduced CRL size, we can push deliver CRLs to verifiers. This allows verifiers to passively maintain the validity of their cached certificates without having to independently request information from the CAs. We avoid unnecessary validation by allowing verifiers to postpone the verification of a cached certificate's revocation state until the certificate is to be used. In this, CRLs are reliably retrieved only when CRLs are lost and a certificate verification is needed. While a push mechanism for CRL delivery is mentioned in [Pro94], we are not aware of any existing design that uses the push mechanism with provable correctness.

Our use of IP multicasting in CRL push delivery minimizes network bandwidth usage by not duplicating data transmission to multiple destinations where their paths overlap. For scalability reasons, IP multicasting uses the unreliable transport protocol, UDP, for data delivery [DC90]. Our ability to use unreliable transport protocol for push delivery of CRLs rests fundamentally on the use of documented scheduled intervals. A verifier with a cached certificate knows the periodicity at which CRL is expected. If the CRL is not received within the expected period, the verifier uses a reliable transport protocol for validation.

An important distinction to note is that the use of unreliable transport protocol in no way affect the security of received CRLs. The security of received CRLs is based on digital signature, and as such are as secure as the signers' CRL generation process (see Section 4.1).

By stipulating that certificate revocations be aggregated at and distributed by keyservers, we reduce total costs of CRL distribution. Thus, the number of enterprise and

end-user certificates scales well with the number of keyservers.

We have considered other approaches to CRL delivery. In one approach, the keyserver create and publish a new CRL every time it receives one from an ER, instead of postponing generation until the next KS CRL publication. Alternatively, each ER can multicast its CRLs directly to certificate holders either on the KS's announcement address or its own multicast group. Both alternatives have the advantage of reducing the window of vulnerability from $wp + p\prime$ to wp. Compared to our proposed protocol, however, these alternatives require the network to carry more messages and certificate holders to be interrupted more frequently, check more digital signatures, and keep a larger number of timers. In addition to the performance trade-offs along these same parameters, requiring a KS to generate a new CRL everytime it receives one from an ER means the KS must execute more digital signatures; constructing and maintaining a multicast tree for each ER may also overtax the networking infrastructure. Nevertheless, we plan to compare the performance of these alternatives against the protocol proposed here in a future study.

3.5.2 Flexibility

We bound the time in which a revoked certificate can be used by its associated CRL publication period. Any certificate which is cached longer than the clean timer is subject to verification explicitly through a fresh CRL, or implicitly by re-acquisition from the CA. The revocation window allows the CA to control the resources required to process CRLs. Smaller revocation windows reduce the size of CRLs, but require hosts to validate or re-acquire certificates more frequently.

An advantage of this approach is that a CA using windowed revocation can mimic traditional key revocation mechanisms. By setting the revocation window equal to the maximum lifetime of any certificate, the CRLs generated will be functionally equivalent to those found in explicit revocation systems. In this way, no cached certificate will ever have its revocation window timer expire before the certificate expiration date. To mimic implicit revocation, CAs running windowed revocation simply set the CRL publication period to 0 and never publish CRLs. This forces all certificates to be re-acquired after their clean timers expire.

Windowed revocation supports verifiers who wish to retrieve revocation state at rates faster than the CRL publication period by setting the clean timer to any period less than the CRL publication period, and the revocation window timer to 0.

Name	Type	Status
Certificate Extensions		
windowedCRLIndicator	BOOLEAN	critical
crlPublicationPeriod	INTEGER	critical
revocationWindow	INTEGER	critical
crlAnnouncementAddress	Name	non-critical
Certificate Revocation List Extensions		
windowedCRLIndicator	BOOLEAN	critical
crlPublicationPeriod	INTEGER	critical
revocationWindow	INTEGER	critical

Table 1: Extensions to the X.509 v3 standard.

4 Issues

4.1 Secure Certificate Retrieval

A central requirement of our revocation mechanism is for freshness assurances in the certificate retrieval process. Without such protection, the retrieval process would be subject to *replay attacks*. By replaying an old response, an adversary may deceive a user into using a revoked key. There are several approaches for achieving freshness described in [NS78] and [Sch96].

In windowed revocation, we avoid the inherent costs of providing freshness on a per request basis by only guaranteeing freshness within a short interval. To achieve this, the directory service generates a certificate packet for each certificate once per configurable period. Included in this packet is the certificate, a timestamp, and a digital signature computed over the previous fields. This packet is returned in response to each request. Based on the signature, the requester can determine that the request is fresh within the bounds of the configured period. In this context, we use the timestamp as a *nonce* value. A verifier is assured of the freshness of the response because the nonce uniquely identifies the packet being generated within the short period.

As the freshness guarantees rely on the quality of the nonce value, this mechanism requires loosely synchronized clocks. This is not an exceptional need, as other secure systems such as Kerberos [SNS88, NT94] require it. There are several widely deployed systems for achieving loosely synchronized clocks in [Mil92].

4.2 Certificate Format

The IETF Public Key Infrastructure Working Group (PKIX) has developed a set of standards for integrating a PKI into the Internet. One standard, the X.509 v3 [HFPS98] draft, provides a flexible interface for specifying certificate distribution and revocation. Through certificate and CRL *extensions*, the issuing authority identi-

fies the location and mechanism used to retrieve revocation state.

In the interests of inter-operability, we propose to implement our revocation mechanism as the set of X.509 v3 extensions listed in Table 1. The *windowedCRLIndicator* field included in the CRL and certificate indicates the use of our mechanism. The *crlPublicationPeriod* field indicates the CRL generation rate, in minutes, of CRLs by the issuing authority. The *revocationWindow* describes the number of periodic CRLs that a revocation announcement will be included. Optionally included in the certificate of each keyserver is the *crlAnnouncementAddress*, which designates the multicast address over which CRLs are delivered.

Note that the *windowedCRLIndicator* and parameter fields are marked as critical. Within X.509 v3 specification, implementations are prohibited from accepting certificates with unsupported critical extensions. In the absence of this, an application may misinterpret a windowed CRL as a traditional CRL, potentially resulting in the use of a revoked certificate.

The delivery of CRLs over multicast is independent of the windowed approach to key revocation. This channel may be used to improve performance in the validation process, but is not necessary for the correct operation of windowed revocation. Consistent with the X.509 v3 philosophy, our mechanism may be used in conjunction with other extensions (see Section 5).

5 Related Work

The Privacy Enhanced Mail [Ken93] architecture (PEM) stipulates that all revoked certificates in each domain be included in periodic CRLs. Due to the long lifetimes of certificates, the size of these lists made CRL distribution difficult. Several approaches to reducing the size the CRLs have been proposed [AZ98, HFPS98], many of which have been included in the IETF Public Key Infrastructure Working Group (PKIX) draft standards. The X.509 v3 certificate format standard [HFPS98] provides extensions in which new mechanisms can be incorporated. Primarily, the existing extensions attempt to reduce CRL associated costs by partitioning the revocation information or by delegating the responsibilities of CRL generation and distribution. Two approaches related to windowed revocation are the *delta CRL* and *freshness CRL* extensions.

CAs supporting delta CRLs [HFPS98] periodically publish a traditional CRL, called a base CRL. Verifiers retrieve and cache the base CRL and more frequently published delta CRLs. Delta CRLs only contain revocation information generated since the last base CRL. In this way, the CA can shorten the publication period without requiring that verifiers obtain the entire CRL each period. A CRL in windowed revocation is similar to the delta CRL in that it presents revocation information within some bounded period. However, unlike CRLs in windowed revocation, delta CRLs continually increase in size between base CRLs. Furthermore, PKIs using delta CRLs are required to acquire, validate, and cache the potentially large base CRLs.

In systems that use freshness CRLs [AZ98], delta CRLs are generated at multiple rates. Verifiers retrieve the CRLs at a rate commensurate with their security requirements. The frequency of freshness CRLs is determined by the CA, and thus limits the verifier to a set of predetermined guarantees. In windowed revocation, each verifier may acquire revocation state at any rate by dropping and reacquiring certificates as needed. Using this mechanism, the verifier can obtain a tight bound on the delivery of revocation state.

The Pretty Good Privacy (PGP) [Zim94] system provides a suite of tools for generating, managing, and revoking certificates within a local environment. PGP does not specify certificate distribution or revocation protocols, but relies on users to define mechanisms commensurate with their needs. In response to this lack of specification, users construct *ad-hoc* relationships between themselves called *webs of trust.* Revocation is explicitly stated by the generation and distribution of a *revocation certificate*.

There is a direct parallel between global certificate and name-space management. In recognition of this fact, the authors of DNSSec [Gal96, EK99] designed an architecture for certificate distribution and revocation using the existing DNS service. As with DNS, certificates are retrieved from the source domain and held for a short time. Later validation is performed by re-acquisition of the certificate. Thus, no explicit revocation notification mechanism is necessary. A limitation of this system is in the inherent cost of retrieval. Dissimilar from existing DNS records, certificates must be retrieved with freshness guarantees. As DNSSec requires each request to be digitally signed by the CA, it is unclear how well it will scale in large networks.

Another architecture using a form of implicit revocation is the Simple Distributed Security Infrastructure (SDSI) [RL96]. SDSI defines a language and toolkit under which user and group certificates can be created, distributed, and revoked. SDSI requires certificate owners to document a *reconfirmation* TTL. When this TTL expires, the validity of the certificate is required to be confirmed by some authority. This is functionally equivalent to the implicit revocation mechanism found in DNSSec.

6 Conclusions and Future Work

In this paper, we have presented a novel approach to key revocation in Public Key Infrastructures. *Windowed revocation* attempts to limit the size of CRLs by announcing revocation only as long as necessary. The time a certificate can be held by a host is bounded by the announcement period, called the *revocation window*. Thus, all certificates will be verified: (1) explicitly by CRL or, (2) implicitly by retrieval. Through the manipulation of revocation window, the CA may influence the CRL size and the frequency with which certificates are retrieved.

We provide an end-to-end push mechanism for CRL delivery using multicast. Using this mechanism, the costs and latencies associated with verifier initiated CRL retrieval are alleviated.

In our design, we provide *a priori* indirect CRLs [HFPS98]. CRLs from potentially many security domains are aggregated and authenticated by a centralized authority. Using aggregated CRLs may increase the performance of the CRL retrieval and validation process.

Within this work, there are performance issues that must be resolved: the observable reduction of CRL size, the frequency with which certificates are retrieved, the costs and benefits of pushing CRLs via multicast, and many others. While an analysis using existing usage characteristics will provide significant insight into the validity of our solution, we feel the best measurement will be the effectiveness of an implementation within the Internet.

We are in the initial stages of an implementation of the KDH PKI. This software will be deployed within our local environment and used as a test-bed to study the usage, performance, and validity of our approach. Further, we plan to integrate the KDH services with SSLeay [HY98], a widely-used session layer providing secure point to point communication. Once our evaluation and implementation is complete, we intend to integrate windowed revocation into systems currently supporting the PKIX working group standards.

References

[AZ98] C. Adams and R. Zuccherato. A general, flexible approach to certificate revocation. http://www.entrust.com/resources/whitepapers.htm, June 1998.

[BAN90] M. Burrows, M. Abadi, and R.M. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8, February 1990.

[CY97] D. Chadwick and A. Young. Mergin and Extending the PGP and PEM Trust Models - The ICE-TEL Trust Model. *IEEE Network*, May/June 1997.

[Dav96] D. Davis. Compliance Defects in Public-Key Cryptography. In *Proceedings of the 6th USENIX Security Symposium*, pages 171–178, July 1996.

[DC90] S.E. Deering and D.R. Cheriton. "Multicast Routing in Internetworks and Extended LANs". *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[DH76] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[EK99] D. Eastlake and C. Kaufman. RFC 2065, Domain Name System Security Extensions. *RFC 2065, Internet Network Working Group*, January 1999.

[Gal96] J. Galvin. Pubilc key distribution with secure dns. pages 161–170, July 1996.

[HFPS98] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure, Certificate and CRL Profile ¡draft-ietf-pkix-ipki-part1-08.txt¿. *IETF X.509 PKI (PKIX) Working Group (Draft)*, June 1998.

[HY98] T. Hudson and E. Young. SSLeay and SSLapps FAQ. http://psych.psy.uq.oz.au/ ftp/Crypto/, September 1998.

[Ken93] S. Kent. RFC 1422, Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management. *RFC 1422, Internet Network Working Group*, February 1993.

[Koc98] P. Kocher. A quick introduction to certificate revocation trees (crts). http://www.valicert.com/resources/body.html, 1998.

[KPS95] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network Security, Private Communication in a Public World*. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[Mic96] S. Micali. Efficient certificate revocation. Technical Report Technical Memo MIT/LCS/TM-542b, Massachusetts Institute of Technology, 1996.

[Mil92] D. L. Mills. Network Time Protocol (Version 3): Specification, Implementation, and Analysis. *RFC 1305, Internet Network Working Group*, March 1992.

[MJ98] P. McDaniel and S. Jamin. Key distribution hierarchy. Technical Report CSE-TR-366-98, EECS , University of Michigan, Ann Arbor, 1998.

[Moc87a] P. Mockapetris. Domain Names - Concepts and Facilities. *RFC 1034, Internet Network Working Group*, November 1987.

[Moc87b] P. Mockapetris. Domain Names - Implementation and Specification. *RFC 1035, Internet Network Working Group*, November 1987.

[NN98] M. Noar and K. Nassim. Certificate Revocation and Certificate Update. In *Proceedings of the 7th USENIX Security Symposium*, pages 217–228, January 1998.

[NS78] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[NT94] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, pages 33–38, September 1994.

[Pro94] Produced by the MITRE Corperation for NIST. Public Key Infrastructure Study, Final Report. , April 1994.

[RL96] R. Rivest and B. Lampson. SDSI A Simple Distributed Security Infrastructure. http://theory.lcs.mit.edu/ rivest/sdsi11.html, October 1996.

[Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., New York, Chichester, Brisbane, Toronto, Singapore, second edition, 1996.

[SNS88] J. G. Steiner, B. C. Neuman, and J. J. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the Usenix Conference*, pages 191–202, 1988.

[Zim94] P. Zimmermann. PGP user's guide. Distributed by the Massachusetts Institute of Technology, May 1994.