A Method for Fast Revocation of Public Key Certificates and Security Capabilities*

Dan Boneh† dabo@cs.stanford.edu
Xuhua Ding‡ xhding@isi.edu
Gene Tsudik‡ gts@ics.uci.edu
Chi Ming Wong† bc@cs.stanford.edu

Abstract

We present a new approach to fast certificate revocation centered around the concept of an on-line semi-trusted mediator (SEM). The use of a SEM in conjunction with a simple threshold variant of the RSA cryptosystem (mediated RSA) offers a number of practical advantages over current revocation techniques. Our approach simplifies validation of digital signatures and enables certificate revocation within legacy systems. It also provides immediate revocation of all security capabilities. This paper discusses both the architecture and implementation of our approach as well as performance and compatibility with the existing infrastructure. Our results show that threshold cryptography is practical for certificate revocation.

1 Introduction

We begin this paper with an example to illustrate the premise for this work. Consider an organization – industrial, government or military – where all employees (referred to as *users*) have certain authorities and authorizations. We assume that a modern Public Key Infrastructure (PKI) is available and all users have digital signature, as well as encryption, capabilities. In the course of performing routine everyday tasks users take advantage of secure applications such as email, file transfer, remote log-in and web browsing.

Now suppose that a trusted user (Alice) does something that warrants immediate revocation of her security privileges. For example, Alice might be fired, or she may suspect that her private key has been compromised. Ideally, immediately following revocation, Alice should be unable to perform any security operations and use any secure applications. Specifically, this means:

- Alice cannot read secure (private) email. This includes encrypted email that is already residing on Alice's email server. Although encrypted email may be basically delivered (to Alice's email server), she cannot decrypt it.
- Alice cannot generate valid digital signatures on any further messages. (However, signatures generated by Alice prior to revocation may need to remain valid.)
- Alice cannot authenticate herself to corporate servers.

In Section 7, we discuss current revocation techniques and demonstrate that the above requirements are impossible to satisfy with these techniques. Most importantly, current techniques do not provide immediate revocation.

1.1 The SEM architecture.

Our approach to immediate revocation of security capabilities is called the SEM architecture. It is easy to use and its presence is transparent to peer users (those that encrypt messages and verify signatures). The basic idea is as follows:

We introduce a new entity, referred to as a SEM *(SEcurity Mediator). A* SEM *is an online semi-trusted server. To sign or decrypt a message, Alice must first obtain a message-specific token from the* SEM*. Without this token Alice cannot use her private key.*[1] *To revoke Alice's ability to sign or de-*

*This work is supported by the Defense Advanced Project Agency (DARPA) under contract F30602-99-1-0530.

†Computer Science Department, Stanford University.
‡Department of Information and Computer Science, University of California, Irvine.

[1] The exact description of the token is in Section 2.

crypt, the security administrator instructs the SEM *to stop issuing tokens for Alice's public key. At that instant, Alice's signature and/or decryption capabilities are revoked. For scalability reasons, a* SEM *serves many users.*

We emphasize that the SEM architecture is transparent to peer users: with SEM's help, Alice can generate a standard RSA signature, and decrypt standard messages encrypted with her RSA public key. Without SEM's help, she cannot perform either of these operations. The SEM architecture is implemented using threshold RSA [3] as described in section 2.

To experiment with this architecture we implemented it using OpenSSL [12]. SEM is implemented as a daemon process running on a server. We describe our implementation, the protocols used to communicate with the SEM, and give performance results in Sections 5 and 6.

We also built a plug-in for the Eudora client enabling users to send signed email. All signatures are generated with SEM's help (see [15]). Consequently, signing capabilities can be easily revoked.

1.2 Decryption and signing in the SEM architecture

We now describe in more detail how decryption and signing is done in the SEM architecture:

– Decryption: suppose Alice wishes to decrypt an email message using her private key. Recall that encrypted email is composed of two parts: (1) a short header containing a message-key encrypted using Alice's public key, and (2) the body contains the email message encrypted using the message-key. To decrypt, Alice first sends the short header to her SEM. SEM responds with a short token. This token enables Alice to read her email. However, it contains no useful information to anyone but Alice. Hence, communication with the SEM does not have to be protected or authenticated. We note that interaction with the SEM is fully managed by Alice's email reader and does not require any intervention on Alice's part. This interaction does not use Alice's private key. If Alice wants to read her email offline, the interaction with the SEM takes places at the time Alice's email client downloads Alice's email from the email server.

– Signatures: suppose Alice wishes to sign a message using her private key. She sends a hash of the message to the SEM which, in turn, responds with a short token enabling Alice to generate the signature. As with decryption, this token contains no useful information to anyone but Alice; therefore, the interaction with the SEM is not encrypted or authenticated.

Note that all interaction with the SEM involves very short messages.

1.3 Other benefits of using a SEM

Our initial motivation for introducing a SEM is to enable immediate revocation of Alice's key. We point out that the SEM architecture provides two additional benefits over standard revocation techniques: (1) simplified signature validation, and (2) enabling revocation in legacy systems. These benefits apply when the following semantics for validating digital signatures are used:

Binding signature semantics: a digital signature is considered valid if the certificate associated with the signature was valid at the time the signature was issued.

A consequence of binding signature semantics is that all signatures issued prior to certificate revocation are valid. Binding semantics are natural in business contracts. For example, suppose Alice and Bob enter into a contract. They both sign the contract at time T. Bob begins to fulfill the contract and incurs certain costs in the process. Now, suppose at time $T' > T$, Alice revokes her own certificate. Is the contract valid at time T'? Using binding semantics, Alice is still bound to the contract since it was signed at time T when her certificate was still valid. In other words, Alice cannot nullify the contract by causing her own certificate to be revoked.

(We note that binding semantics are inappropriate in some scenarios. For example, if a certificate is obtained from a CA under false pretense, e.g., Alice masquerading as Bob, the CA should be allowed to declare at any time that **all** signatures ever issued under that certificate are invalid.)

Implementing binding signature semantics with existing revocation techniques is complicated, as discussed in Section 7. Whenever Bob verifies a signature generated by Alice, Bob must also verify that Alice's certificate was valid at the time the signature was issued. In fact, every verifier of Alice's signature must perform this certificate validation step. However, unless a trusted timestamping service is involved in generating all of Alice's signatures, Bob cannot trust the timestamp provided by Alice in her signatures.

Implementing binding semantics with the SEM architecture is trivial. To validate Alice's signature, a verifier need only verify the signature itself. There is no need to check the status of Alice's certificate.[2] Indeed, once Alice's certificate is revoked she can no longer generate valid signatures. Therefore, the mere existence of the signature implies that Alices's certificate was valid at the time the signature was issued.

The above discussion brings out two additional benefits of a SEM over existing revocation techniques, assuming binding semantics are sufficient.

– Simplified signature validation. Verifiers need not validate the signer's certificate. The existence of a (verifiable) signature is, in itself, a proof of signature's validity.

– Enabling revocation in legacy systems. Consider legacy systems doing signature verification. Often, such systems have no certificate validation capabilities. For example, old browsers (e.g., Netscape 3.0) verify server certificates without any means for checking certificate revocation status. In SEM architecture, certificate revocation is provided without any change to the verification process in these legacy systems. (The only aspect that needs changing is the signature generation process. However, we note that, often, only a few entities generate signatures, e.g., CAs and servers.)

2 Mediated RSA

We now describe in detail how the SEM interacts with users to generate tokens. The proposed SEM architecture is based on a variant of RSA which we call Mediated RSA (mRSA). The main idea in mRSA is to split each RSA private key into two parts using threshold RSA [3]. One part is given to a user while the other is given to a SEM. If the user and the SEM cooperate, they employ their respective half-keys in a way that is functionally equivalent to (and indistinguishable from) standard RSA. The fact that the private key is not held in its entirety by any one party is transparent to the outside world, i.e., to the those who use the corresponding public key. Also, knowledge of a half-key cannot be used to derive the entire private key. Therefore, neither the user nor the SEM can decrypt or sign a message without mutual consent. (A single SEM serves a multitude of users.)

[2] We are assuming here that revocation of Alice's key is equavalent to revocation of Alice's certificate. In general, however, Alice's certificate may encode many rights, not just the right to use her key(s). It is then possible to revoke only some of these rights while not revoking the entire certificate.

2.1 mRSA in detail

Public Key. As in RSA, each user (U_i) has a public key $EK_i = (n_i, e_i)$ where the modulus n_i is product of two large primes p_i and q_i and e_i is an integer relatively prime to $\phi(n_i)$.

Secret Key. As in RSA, there exists a corresponding secret key $DK_i = (n_i, d_i)$ where $d_i * e_i = 1 \pmod{\phi(n_i)}$. However, as mentioned above, no one has possession of d_i. Instead, d_i is effectively split into two parts d_i^u and d_i^{sem} which are held by the user U_i and a SEM, respectively. The relationship among them is:

$$d_i \ = \ d_i^{sem} \ + \ d_i^u \mod \phi(n)$$

mRSA Key Setup. Recall that, in RSA, each user generates its own modulus n_i and a public/secret key-pair. In mRSA, a trusted party (most likely, a CA) takes care of all key setup. In particular, it generates a distinct set: $\{p_i,\ q_i,\ e_i,$ and $d_i, d_i^{sem}\}$ for each user. The first four are generated in the same manner as in standard RSA. The fifth value, d_i^{sem}, is a random integer in the interval $[1, n_i]$. The last value is set as: $d_i^u = d_i - d_i^{sem}$.

After CA computes the above values, d_i^{sem} is securely communicated to a SEM and d_i^u – to the user U_i. The details of this step are elaborated in Section 5.

mRSA Signatures. A user generates a signature on a message m as follows:

1. The user U_i first sends a hash of the message m to the appropriate SEM.
   - SEM checks that U_i is not revoked and, if so, computes a partial signature $PS_{sem} = m^{d_i^{sem}} (\bmod n_i)$ and replies with it to the user. This PS_{sem} is the token enabling signature generation.
   - concurrently, U_i computes $PS_u = m^{d_i^u} \pmod{n_i}$
2. U_i receives PS_{sem} and computes $m' = (PS_{sem} * PS_u)^{e_i} (\bmod n_i)$. If $m' = m$, the signature is set to: $(PS_{sem} * PS_u) = m^{d_i} \pmod{n_i}$.

Note that in Step 2 the user U_i validates the response from the SEM. Signature verification is identical to that in standard RSA.

mRSA Encryption. The encryption process is identical to that in standard RSA. (In other words, ciphertext is computed as $c = m^{e_i} (\bmod n_i)$ where m is an appropriately padded plaintext, e.g., using OAEP.) Decryption, on the other hand, is very similar to signature generation above.

1. upon obtaining an encrypted message c, user U_i sends it to the appropriate SEM.
   - SEM checks that U_i is not revoked and, if so, computes a partial cleartext $PC_{sem} = c^{d_i^{sem}} (\bmod n_i)$ and replies to the user.
   - concurrently, U_i computes $PC_u = c^{d_i^u} \pmod{n_i}$.
2. U_i receives PC_{sem} and computes $c' = (PC_{sem} * PC_u)^{e_i} (\bmod n_i)$. If $c' = c$, the cleartext message is: $(PC_{sem} * PC_u) = c^{d_i^u}$.

2.2 Notable Features

As mentioned earlier, mRSA is only a slight modification of the RSA cryptosystem. However, at a higher, more systems level, mRSA affords some interesting features.

CA-based Key Generation. Recall that, in RSA, a private/public key-pair is typically generated by its intended owner. In mRSA the key-pair is typically generated by a CA, implying that the CA knows the private keys belonging to all users. In the global Internet this is clearly undesirable. However, in a medium-sized organization this "feature" provides key escrow. For example, if Alice is fired, the organization can still access her work-related files by obtaining her private key from the CA.

If key escrow is undesirable, it is easy to extend the system in a way that no entity ever knows Alice's private key (not even Alice or the CA). To do so, we can use a technique due to Boneh and Franklin [2] to generate an RSA key-pair so that the private key is shared by a number of parties since its creation (see also [4]). This technique has been implemented in [8]. It can be used to generate a shared RSA key between Alice and the SEM so that no one knows the full private key. Our initial implementation does not use this method. Instead, the CA does the full key setup.

Immediate Revocation. The notoriously difficult revocation problem is greatly simplified in mRSA. In order to revoke a user's public key, it suffices to notify that user's SEM. Each SEM merely maintains a list of revoked users which is consulted upon every service request. Our implementation uses standard X.509 Certificate Revocation Lists (CRL's) for this purpose.

Transparency. mRSA is completely transparent to those who encrypt data for mRSA users and those who verify signatures produced by mRSA users. To them, mRSA appears indistinguishable from standard RSA. Furthermore, mRSA certificates are identical to standard RSA certificates.

Coexistence. mRSA's built-in revocation approach can co-exist with the traditional, explicit revocation approaches. For example, a CRL- or a CRT-based scheme can be used in conjunction with mRSA in order to accommodate existing implementations that require verifiers (and encryptors) to perform certificate revocation checks.

CA Communication. CA remains an off-line entity. mRSA certificates, along with private half-keys are distributed to the user and SEM-s in an off-line manner. This follows the common certificate issuance and distribution paradigm. In fact, in our implementation (Section 5) there is no need for the CA and the SEM to ever communicate directly.

No Authentication. mRSA does not require any explicit authentication between a SEM and a user. Instead, a user implicitly authenticates a SEM by verifying its own signature (or encryption) as described in Section 2.1. In fact, signature and encryption verification steps assure the user of the integrity of the communication with the SEM.

3 Architecture

The overall architecture is made up of three components: CA, SEM, and user.

A single CA governs a (small) number of SEMs. Each SEM, in turn, serves many users. The assignment of users to SEMs is assumed to be handled off-line by a security administrator. A user may be served by multiple SEM's.

Our CA component is a simple add-on to the existing CA and is thus considered an off-line entity. For each user, the CA component takes care of generating an RSA public key, a corresponding RSA public key certificate and a pair of half-keys (one for the user and one for the SEM) which, when combined, form the RSA private key. The respective half-keys are then delivered, out-of-band, to the interested parties.

The user component consists of the client library that provides the mRSA sign and mRSA decrypt operations. (As mentioned earlier, the verify and encrypt operations are identical to standard RSA.) It also handles the installation of the user's credentials at the local host.

The SEM component is the critical part of the architecture. Since a single SEM serves many users, performance, fault-tolerance and physical security are of paramount concern. The SEM is basically a daemon process that processes requests from its constituent users. For each request, SEM consults its revocation list and refuses to help sign (or decrypt) for any revoked users. A SEM can be configured to operate in a stateful or stateless model. The former involves storing per user state (half-key and certificate) while, in the latter, no per user state is kept, however, some extra processing is incurred for each user request. The tradeoff is fairly clear: per user state and fast request handling versus no state and somewhat slower request handling.

We now describe the SEM architecture in more detail. A user's request is initially handled by the SEM controller where the packet format is checked. Next, the request is passed on to the client manager which performs a revocation check. If the requesting user is not revoked, the request is handled depending on the SEM state model. If the SEM is stateless, it expects to find the so-called SEM *bundle* in the request. This bundle, as discussed in more detail later, contains the mRSA half-key, d_i^{SEM}, encrypted (for the SEM, using its public key) and signed (by the CA). The bundle also contains the RSA public key certificate for the requesting user. Once the bundle is verified, the request is handled by either the $\mathsf{mRSA_{sign}}$ or $\mathsf{mRSA_{decrypt}}$ component. In case of the appropriate signature request, the optional timestamping service is invoked. If the SEM maintains user state, the bundle is expected only in the initial request. The same process as above is followed, however, the SEM's half-key and the user's certificate are stored locally. In subsequent user requests, the bundle (if present) is ignored and local state is used instead.

The administrator communicates with the SEM via the admin interface. The interface enables the administrator to manipulate the revocation list.

4 Security of the SEM architecture

We now briefly summarize the security features of mRSA and the SEM architecture.

First, consider an attacker trying to subvert a user (Alice). The attacker's goal is to decrypt a message sent to Alice or to forge Alice's signature on a certain message. Recall that the token sent back to Alice is $t = x^{d^{sem}} \bmod N$ for some value of x. The attacker sees both x and the token t. In fact, since there is no authentication of the user's request to the SEM, the attacker can obtain this t for any x of its choice. We claim that this information is of no use to an attacker. After all, d^{sem} is just a random number in $[1, n]$ independent of the rest of the attacker's view. More precisely, we argue that any attack possible with the SEM architecture is also possible when the user uses standard RSA. This statement can be proven using a simulation argument. In attacking standard RSA one can simulate the SEM (by picking a random integer d^{sem} in $[1, n]$) and thus use the attack on the SEM to mount an attack on standard

RSA. Furthermore, the attacker cannot masquerade as the SEM since Alice checks all responses from the SEM as described in Section 2.1.

Suppose the attacker is able to compromise the SEM and expose the secret key d^{sem}. This enables the attacker to "unrevoke" revoked, or block possible future revocation of currently valid, certificates. However, knowledge of d^{sem} does not enable the attacker to decrypt messages or sign messages on behalf of users. Nevertheless, it is desirable to protect the SEM's key. A standard approach is to distribute the key among a number of SEM servers using secret sharing. Furthermore, the key should never be reconstructed at a single location. To extract the SEM's key an attacker would need to break into multiple SEM servers. When using mRSA, it is possible to distribute the SEM's secret in this way using standard techniques from threshold cryptography [3].

Once Alice's key is revoked, she cannot decrypt or sign messages using her private key. To show this, we argue that, if Alice could sign or decrypt messages using only her share of private key, then RSA is insecure.

Finally, note that each user is given her own random RSA modulus n_i. This means that if a number of users are compromised (or a number of users collude) there is no danger to other users. The private keys of the compromised users will be exposed, but private keys of all other users will remain unaffected.

5 Implementation

We implemented the entire SEM architecture for the purposes of experimentation and validation. The reference implementation is publicly available. Following the architecture described earlier, the implementation is composed of three parts:

1. CA and Admin Utilities:
   includes certificate issuance and revocation interface.

2. SEM daemon:
   SEM architecture as described in Section 3

3. Client libraries:
   mRSA user functions accessible via an API.

Our reference implementation uses the popular OpenSSL [12] library as the low-level cryptographic platform. OpenSSL incorporates a multitude of cryptographic functions and large-number arithmetic primitives. In addition to being efficient and available on many common hardware and software platforms, OpenSSL adheres to the common PKCS standards and is in the public domain.

The SEM daemon and the CA/Admin utilities are implemented on Linux and Solaris while the client libraries are available on both Linux and Windows98 platforms.

In the initialization phase, CA utilities are used to set up the RSA public key-pair for each client (user). The set up process follows the description in Section 2. Once the mRSA parameters are generated, two structures are exported: 1) SEM *bundle*, which includes the SEM's half-key d_i^{SEM}, and 2) user *bundle*, which includes d_i^u and the entire server bundle. The format of both SEM and user bundles conforms to the PKCS#7 standard.

The server bundle is basically an RSA envelope signed by the CA and encrypted with the SEM's public key. The client bundle is a shared-key envelope also signed by the CA and encrypted with the user-supplied key which can be a password or a passphrase. (A user cannot be assumed to have a pre-existing public key.)

After issuance, the user bundle is distributed in an out-of-band manner to the appropriate user. Before attempting any mRSA transactions, the user must first decrypt and verify the bundle. A separate utility program is provided for this purpose. With it, the bundle is decrypted with the user-supplied key, the CA's signature is verified, and, finally, the user's new certificate and half-key are extracted and stored locally.

To sign or decrypt a message, the user starts with sending an mRSA request with the SEM bundle piggybacked. The SEM processes the request and the bundle contained therein as described in Section 3. (Recall that the SEM bundle is processed based on the state model of the particular SEM.) All mRSA packets have a common packet header; the payload format depends on the packet type. The packet header is defined in Figure 1.

```
0              8                16              24               32
+--------------+----------------+---------------+---------------+
| PROTOCOL     |  PACKET_TYPE   |      DATA_LENGTH              |
+--------------+----------------+---------------+---------------+

PROTOCOL   :  protocol identifier. Set to MRSA(=1) in current code
PACKET_TYPE:  one of the following:
         1) REG_REQ_T : register request
         2) REG_RLY_T : register reply
         3) SIG_REQ_T : signature request
         4) SIG_RLY_T : signature reply
         5) DEC_REQ_T : decrypt request
         6) DEC_RLY_T : decrypt reply
```

Figure 1: mRSA Packet Header

5.1 Email client plug-in

To demonstrate the ease of using the SEM architecture we implemented a plug-in for the Eudora email reader [15]. When sending signed email the plug-in reads the user bundle described in the previous section. It obtains the SEM address from the bundle and then communicates with the SEM to sign the email. The resulting signed email can be verified using any S/MIME capable email client such as Microsoft Outlook. In other words, the email recipient is oblivious to the fact that a SEM is used to control the sender's signing capabilities.

Figure 2 shows a screen snap shot of trying to send signed email using a revoked key. In this example, the plug-in contacts the SEM and is told that the SEM will not supply the token for a revoked key. Consequently, the plug-in displays a message informing the user that the email cannot be signed.

6 Experimental Results

We conducted a number of experiments in order to evaluate the practicality of the proposed architecture and our implementation.

We ran the SEM daemon on a Linux PC equipped with an 800 Mhz Pentium III processor. Two different clients were used. The fast client was on another Linux PC with a 930 MHz Pentium III. Both SEM and fast client PC-s had 256M of RAM. The slow client was on a Linux PC with 466 MHz Pentium II and 128M of RAM. Although an 800 Mhz processor is not exactly state-of-the-art, we opted to err on the side of safety and assume a relatively conservative (i.e., slow) SEM platform. In practice, a SEM might reside on much faster hardware and is likely to be assisted by an RSA hardware acceleration card.

Each experiment involved one thousand iterations. All reported timings are in milliseconds (rounded to the nearest 0.1 ms). The SEM and client PCs were located in different sites interconnected by a high-speed regional network. All protocol messages are transmitted over UDP.

Client RSA key (modulus) sizes were varied among 512, 1024 and 2048 bits. (Though it is clear that 512 is not a realistic RSA key size any longer.) The timings are only for the mRSA sign operation since mRSA decrypt is operationally almost identical.

6.1 Communication Overhead

In order to gain precise understanding of our results, we first provide separate measurements for communication latency in mRSA. Recall that both mRSA operations involve a request from a client followed by a reply from a SEM. As mentioned above, the test PCs were connected by a high-speed regional network. We measured communication latency by varying the key size which directly influences message sizes. The results are shown in Table 1 (message sizes are in bytes). Latency is calculated as the round-trip delay between the client and the SEM. The numbers are identical for both client types.

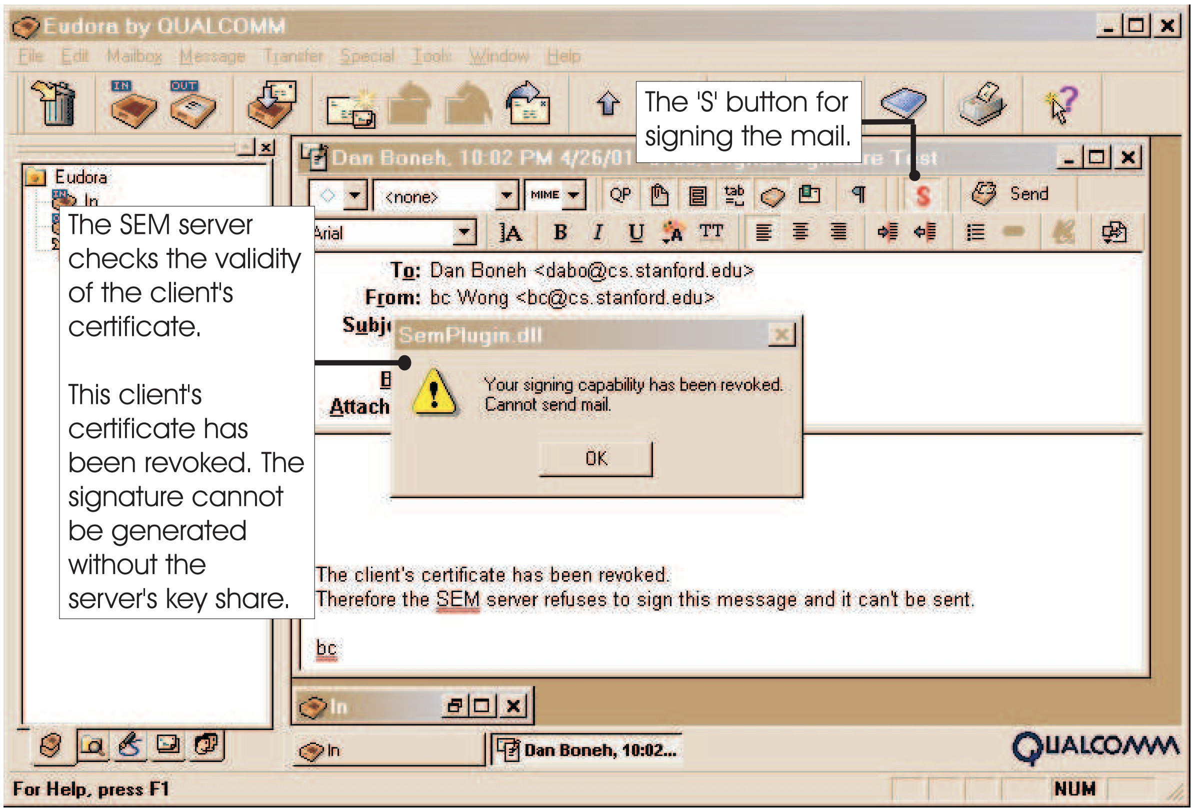

Figure 2: Screen snapshot of SEM email plug-in

Keysize (bits)	Message Size (bytes)	Comm. latency (ms)
512	102	4.0
1024	167	4.5
2048	296	5.5

Table 1: Communication latency

6.2 Standard RSA

As a point of comparison, we initially timed the standard RSA sign operation in OpenSSL (Version 0.9.6) with three different key sizes on each of our three test PCs. The results are shown in Tables 2 and 3. Each timing includes a message hash computation followed by an exponentiation. Table 2 reflects optimized RSA computation where the *Chinese Remainder Theorem (CRT)* is used to speed up exponentiation (essentially exponentiation is done modulo the prime factors rather than modulo N). Table 3 reflects unoptimized RSA computation without the benefit of the CRT. Taking advantage of the CRT requires knowledge of the factors (p and q) of the modulus n. Recall that, in mRSA, neither the SEM nor the user know the factorization of the modulus, hence, with regard to its computation cost, mRSA is more akin to unoptimized RSA.

As evident from the two tables, the optimized RSA performs a factor of 3-3.5 faster for the 1024- and 2048-bit moduli than the unoptimized version. For 512-bit keys, the difference is slightly less marked.

6.3 mRSA Measurements

The mRSA results are obtained by measuring the time starting with the message hash computation by the user (client) and ending with the verification of the signature by the user. The measurements are

Keysize (bits)	466 Mhz PII (slow client)	800 Mhz PIII (SEM)	930 Mhz PIII (fast client)
512	2.9	1.4	1.4
1024	14.3	7.7	7.2
2048	85.7	49.4	42.8

Table 2: RSA results with CRT (in milliseconds).

Keysize (bits)	466 Mhz PII (slow client)	800 Mhz PIII (SEM)	930 Mhz PIII (fast client)
512	6.9	4.0	3.4
1024	43.1	24.8	21.2
2048	297.7	169.2	144.7

Table 3: Standard RSA results without CRT (in milliseconds).

illustrated in Table 4.

It comes as no surprise that the numbers for the slow client in Table 4 are very close to the unoptimized RSA measurements in Table 3. This is because the time for an mRSA operation is determined solely by the client for 1024- and 2048- bit keys. With a 512-bit key, the slow client is fast enough to compute its PS_u in $6.9ms$. This is still under $8.0ms$ (the sum of $4ms$ round-trip delay and $4ms$ RSA operation at the SEM).

The situation is very different with a fast client. Here, for all key sizes, the timing is determined by the sum of the round-trip client-SEM packet delay and the service time at the SEM. For instance, $178.3ms$ (clocked for 2048-bit keys) is very close to $174.7ms$ which is the sum of $5.5ms$ communication delay and $169.2ms$ unoptimized RSA operation at the SEM.

All of the above measurements were taken with the SEM operating in a stateful mode. In a stateless mode, SEM incurs further overhead due to the processing of the SEM bundle for each incoming request. This includes decryption of the bundle and verification of the CA's signature found inside. To get an idea of the mRSA overhead with a stateless SEM, we conclude the experiments with Table 5 showing the bundle processing overhead. Only 1024- and 2048-bit SEM key size was considered. (512-bit keys are certainly inappropriate for a SEM.) The CA key size was constant at 1024 bits.

7 Comparison of SEM with existing certificate revocation techniques

Certificate revocation is a well recognized problem with the existing Public Key Infrastructure (PKI). Several proposals address this problem. We briefly review these proposals and compare them to the SEM architecture. For each proposal we describe how it applies to signatures and to encryption. For simplicity we use signed and encrypted Email as an example application. We refer to the entity validating and revoking certificates as the Validation Authority (VA). Typically, the VA is the same entity as the Certificate Authority (CA). However, in some cases these are separate organizations.

A note on timestamping. Binding signature semantics (Section 1.3) for signature verification states that a signature is considered valid if the key used to generate the signature was valid at the time signature generation. Consequently, a verifier must establish exactly when a signature was generated. Hence, when signing a message, the signer must interact with a trusted timestamping service to obtain a trusted timestamp and a signature over the user's (signed) message. This proves to any verifier that a signature was generated at a specific time. All the techniques discussed below require a signature to contain a timestamp indicating when a signature was issued. We implicitly assume this service. As we will see, there is no need for a trusted time service to implement binding signature semantics with the SEM architecture.

Keysize (bits)	466 Mhz PII (slow client)	930 Mhz PIII (fast client)
512	8.0	9.9
1024	45.6	31.2
2048	335.6	178.3

Table 4: Timings for mRSA (in milliseconds).

SEM key size	Bundle overhead
1024	8.1
2048	50.3

Table 5: Bundle overhead in mRSA with a SEM in a stateless mode (in milliseconds).

7.1 Review of existing revocation techniques

CRLs and Δ-CRLs: Certificate Revocation Lists are the most common way to handle certificate revocation. The Validation Authority (VA) periodically posts a signed list of all revoked certificates. These lists are placed on designated servers called CRL distribution points. Since these lists can get quite long, the VA may alternatively post a signed Δ-CRL which only contains the list of revoked certificates since the last CRL was issued. For completeness, we briefly explain how CRLs are used in the context of signatures and encryption:

- Encryption: at the time email is sent, the sender checks that the receiver's certificate is not on the current CRL. The sender then sends encrypted email to the receiver.
- Signatures: when verifying a signature on a message, the verifier checks that, at the time that the signature was issued, the signer's certificate was not on the CRL.

OCSP: The Online Certificate Status Protocol (OCSP) [11] improves on CRLs by avoiding the transmission of long CRLs to every user and by providing more timely revocation information. To validate a specific certificate in OCSP, the user sends a certificate status request to the VA. The VA sends back a signed response indicating whether the specified certificate is currently revoked. OCSP is used as follows for Encryption and signatures:

- Signatures: When verifying a signature, the verifier sends an OCSP query to the VA to check if the corresponding certificate is currently valid. Note that the current OCSP protocol prevents one from implementing binding semantics: it is not possible to ask an OCSP responder whether a certificate was valid at some time in the past. Hopefully this will be corrected in future versions of the protocol.

  One could potentially abuse the OCSP protocol and provide binding semantics as follows. To sign a message, the signer generates the signature, and also sends an OCSP query to the VA. The VA responds with a signed message saying that the certificate is currently valid. The signer appends both the signature and the response from the VA to the message. To verify the signature, the verifier checks the VA's signature on the validation response. The response from the VA provides a proof that the signer's certificate is currently valid. This method reduces the load on the VA: it is not necessary to contact the VA every time a signature is verified. Unfortunately, there is currently no infrastructure to support this mechanism.
- Encryption: Every time the sender sends an encrypted message to the receiver she sends an OCSP query to the VA to ensure that the receiver's certificate is still valid.

Certificate Revocation Trees: Kocher suggested an improvement over OCSP [7]. Since the VA is a global service it must be sufficiently replicated in order to handle the load of all the validation queries. This means the VA's signing key must be replicated across many servers which is either insecure or expensive (VA servers typically use tamper-resistance to protect the VA's signing key). Kocher's idea is to have a single highly secure VA periodically post a signed CRL-like data structure to many insecure VA servers. Users then query these insecure VA servers. The data structure proposed by Kocher is a hash tree where the leaves are the currently revoked certificates sorted by serial number (lowest serial number is the left most leaf and the highest serial number is the right most leaf). The root of the hash tree is signed by the VA. This hash tree data structure is called a Certificate Revocation Tree (CRT).

When a user wishes to validate a certificate CERT she issues a query to the closest VA server. Any insecure VA can produce a convincing proof that CERT is (or is not) on the CRT. If n certificates are currently revoked, the length of the proof is $O(\log n)$. In contrast, the length of the validity proof in OCSP is $O(1)$.

Skip-lists and 2-3 trees: One problem with CRT's is that, every time a certificate is revoked, the entire CRT must be recomputed and distributed in its entirety to the various VA servers. A data structure allowing for dynamic updates would solve this problem since the secure VA would only need to send small updates to the data structure along with a signature on the new root of the structure. Both 2-3 trees proposed by Naor and Nissim [10] and skip-lists proposed by Goodrich [5] are natural data structures for this purpose. Additional data structures were proposed in [1]. When a total of n certificates are already revoked and k new certificates must be revoked during the current time period, the size of the update message to the VA servers is $O(k \log n)$ (as opposed to $O(n)$ with CRT's). The proof of certificate's validity is $O(\log n)$, same as with CRTs.

7.2 Comparison with SEM architecture

CRLs and OCSP are the most commonly deployed certificate revocation techniques. Some positive experiments with skip-lists are reported in [5]. We compare the SEM architecture with CRLs and OCSP. Since CRT's and skip-lists are used in the same way as OCSP (i.e., query a VA to obtain a proof of validity) most everything in our OCSP discussion applies to these methods as well.

Immediate revocation: Suppose we use CRLs for revocation. Then, Bob verifies a signature or encrypts a message he must first download a long CRL and verify that the Alice's certificate is not on the CRL. Note that Bob is uninterested in all but one certificate on the CRL. Nevertheless, he must download the entire CRL since, otherwise, the VA's signature on the CRL cannot be verified. Since CRLs and Δ-CRLs tend to get long, they are downloaded infrequently, e.g., once a week or month. As a result, certificate revocation might only take effect a month after the revocation occurs. The SEM architecture solves this problem altogether.

Suppose now that OCSP is usd for revocation. Whenever Bob sends email to Alice he first issues an OCSP query to verify validity of Alice's certificate. He then sends email encrypted with Alice's public key. The encrypted email could sit on Alice's email server for a few hours or days. If, during this time, Alice's key is revoked (e.g., because Alice is fired or looses her private key) there is nothing preventing the holder of Alice's private key from decrypting the email after revocation. The SEM solves this problem by disabling the private key immediately after revocation.

Implicit timestamping: Both OCSP and CRLs require the signer to contact a trusted time service at signature generation time to obtain a secure timestamp for the signature. Otherwise, a verifier cannot determine with certainty when the signature was issued. If binding semantics are sufficient, the time service is unnecessary when using the SEM architecture. Once a certificate is revoked, the corresponding private key can no longer be used to issue signatures. Therefore, a verifier holding a signature is explicitly assured that the signer's certificate was valid at the time the signature was generated.

Shifted validation burden: With current PKIs, the burden of validating certificates is placed on: (1) senders of encrypted messages and (2) verifiers of signed messages. In the SEM architecture, the burden of certificate validation is reversed: (1) receivers of encrypted messages and (2) signers (generators) of signed messages.

SEM **Replication (A disadvantage):** Since many users need to use the SEM for decryption and signing, it is natural to replicate it. However, replicating the SEM across organizations is not recommended for the same reason that replicating the VA in OCSP is not recommended. Essentially, the SEM generates tokens using a private key known only to the SEM. The result of exposing this key is that an attacker could unrevoke certificates. Replicating the SEM might make it easier to expose the SEM's key. Hence, the SEM architecture is mainly applicable in the same environments where OCSP is used, i.e., mainly medium-sized organizations. The SEM architecture is not geared towards the global Internet.

8 Conclusions

We described a new approach to certificate revocation. Rather than revoking the user's certificate our approach revokes the user's ability to perform cryptographic operations such as signature generation and decryption. This approach has several advantages over traditional certificate revocation techniques: (1) revocation is instantaneous – the instant the user's certificate is revoked the user can no longer decrypt or sign messages, (2) when using binding signature semantics there is no need to validate the signer's certificate during signature verification, and (3) using mRSA this revocation technique is transparent to the peer – the system generates standard RSA signatures and decrypts standards RSA encrypted messages.

We implemented the SEM architecture for experimentation purposes. Our measurements of the implementation show that signature and decryption times are essentially unchanged from the user's perspective. Therefore, we believe this architecture is appropriate for a medium-size organization where tight control of security capabilities is desired. The SEM architecture is not designed for the global Internet.

9 Acknowledgments

The authors gratefully acknowledge Carl Ellison for helping us to get this paper into proper shape and bringing up a number of interesting points. We also thank Paolo Montini and Ignacio Solis for their help with the initial SEM and mRSA prototyping.

References

[1] W. Aiello, S. Lodha, R. Ostrovsky, "Fast digital identity revocation", In proceedings of CRYPTO '98.

[2] D. Boneh, M Franklin, "Efficient generation of shared RSA keys", In Proceedings of Crypto' 97, Lecture Notes in Computer Science, Vol. 1233, Springer-Verlag, pp. 425–439, 1997.

[3] P. Gemmel, "An introduction to threshold cryptography", in CryptoBytes, a technical newsletter of RSA Laboratories, Vol. 2, No. 7, 1997.

[4] N. Gilboa, "Two Party RSA Key Generation", in Proceedings of Crypto '99.

[5] M. Goodrich, R. Tamassia, and A. Schwerin, "Implementation of an Authenticated Dictionary with Skip Lists and Commutative Hashing", In Proceedings of DARPA DISCEX II, June 2001.

[6] S. Haber, W.S. Stornetta, "How to timestamp a digital document", J. of Cryptology, Vol. 3, pp. 99–111, 1991.

[7] P. Kocher, "On Certificate Revocation and Validation", Financial Cryptography – FC '98, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1465, 1998, pp. 172-177.

[8] M. Malkin, T. Wu, and D. Boneh, "Experimenting with Shared Generation of RSA keys", In proceedings of the Internet Society's 1999 Symposium on Network and Distributed System Security (SNDSS), pp. 43–56.

[9] S. Micali, "Enhanced certificate revocation system", Technical memo, MIT/LCS/TM-542b, March 1996.

[10] M. Naor, K. Nissim, "Certificate revocation and certificate update", In proceedings of USENIX Security '98.

[11] M. Myers, R. Ankney, A. Malpani, S. Galperin and C. Adams, "X.509 Internet PKI Online Certificate Status Protocol - OCSP". IETF RFC 2560, June 1999.

[12] OpenSSL, `http://www.openssl.org`

[13] R. Rivest, "Can we eliminate Certificate Revocation Lists", Financial Cryptography – FC '98, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1465, 1998, pp. 178-183.

[14] R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", CACM, Vol. 21, No. 2, February 1978.

[15] SEM Eudora plug-in.
`http://crypto.stanford.edu/semmail/`