Instantaneous Revocation of Security Capabilities

Dan Boneh
dabo@cs.stanford.edu

Xuhua Ding
xhding@isi.edu

Gene Tsudik
gts@cs.uci.edu

Abstract

Current certificate revocation methods are unable to handle immediate revocation of security privileges and credentials. This paper describes a new approach to credentials revocation centered around the concept of an on-line semi-trusted helper (SEM). The use of a SEM in conjunction with a variant of the RSA cryptosystem (mediated RSA) offers a number of practical advantages over and above immediate revocation. Both the architecture and an implementation of this new approach are discussed as well as the performance, compatibility and usability aspects.

1 Introduction

We begin 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 lose her private key, or she might be fired. 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 currently in Alice's mailbox. Although the encrypted mail is in Alice's mailbox, she cannot decrypt the mail.
- Alice cannot generate valid digital signatures. We want that revocation will prevent Alice from signing any further messages.
- Alice cannot authenticate herself to corporate servers.

However, at the same time, signatures generated by Alice prior to revocation should remain valid. This is standard semantics for signature verification.

In Section 2, we review current revocation techniques and demonstrate that the above requirements are impossible to satisfy with current techniques. Specifically, current certificate revocation techniques do not provide immediate revocation.

The SEM architecture: To provide for immediate revocation of security capabilities we propose an architecture, called the SEM architecture, that is relatively easy to use and is transparent to peer users (the ones encrypting and verifying signatures). The basic idea is as follows: we introduce a new entity, called a SEM (SEcurity Mediator), which is an online semi-trusted service. 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. The exact description of the token is presented in Section 3. To revoke Alice's ability to sign or decrypt the administrator instructs the SEM to stop issuing tokens for Alice's public key. At that instant Alice's capabilities are revoked. A single SEM serves many users. We emphasize that this SEM architecture is transparent to peer users: using the token Alice generates a standard RSA signature, and can decrypt standard Email encrypted using her RSA public key. Without the token she cannot do either of these operations.

To experiment with this architecture we implemented it using OpenSSL [10]. The SEM is implemented as a daemon process running on some server. We describe our implementation, the protocols used to communicate with the SEM, and give performance numbers in Sections 6 and 7.

Next, we briefly describe in more detail how decryption and signing is done using 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 the Email Alice first sends the short header to the SEM. The SEM responds with a short token. Alice can then read her mail. This token contains no useful information (to anyone but Alice). Hence communication with the SEM does not have to be protected or authenticated. We note that the interaction with the SEM is completely managed by Alice's mail reader and does not require any intervention on Alice's part. This interaction does not use Alice's private key. To enable Alice to read her mail offline, the interaction with the SEM takes places at the time Alice's mail client downloads Alice's mail from the mail server. We are currently integrating our prototype implementation with the Eudora mail reader to experiment with this system.

– Signatures: Suppose Alice wishes to sign a message using her private key. She sends a hash of the message to the SEM. The SEM responds with a short token enabling Alice to generate the signature. As above, this token contains no useful information (to anyone but Alice). Hence communication with the SEM does not have to be protected or authenticated.

Note that either way, communication with the SEM involves very short messages. All traffic is in the clear.

Our initial motivation for introducing a SEM is to enable immediate revocation of Alice's key. As we will see, the SEM is very useful for simplifying the semantics of signature verification. First, note that under the current PKI model whenever Bob verifies a signature issued by Alice, Bob must also verify that Alice's certificate was valid at the time the signature was issued. Otherwise the signature is meaningless. Every entity verifying Alice's signature must repeat this certificate validation step. When using a SEM there is no need to ever validate Alice's certificate: if Alice's certificate was revoked Alice could not have generated the signature. Hence, the existence of the signature implies that Alice's certificate was valid at the time the signature was issued. This shows that the SEM significantly simplifies the semantics of signature verification.

2 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 in charge of validating and revoking certificates as a 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. The common semantics for signature verification is as follows: a signature is considered valid if the key used to issue the signature was valid at the time the signature was issued. This implies that the verifier must be able to determine when the signature was issued. Hence, when signing a message, the signer must interact with a trusted time server to obtain a timestamp proving when the signature was issued. This timestamp is included in the signature. All the techniques below require that the signature contain a timestamp indicating when the signature was issued. We implicitly assume this is so. As we will see, there is no need for a trusted time service when using the SEM architecture.

2.1 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 the Email is sent the sender verifies that the receiver's certificate is not on the current CRL. The sender then sends the encrypted mail to the receiver.
- Signatures: when verifying a signature on a given message the verifier checks that at the time that the signature was issued the signer's certificate was not posted on the CRL.

OCSP: The Online Certificate Status Protocol (OCSP) improves on CRLs by avoiding transmitting long CRL lists to every user. To validate a specific certificate using 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 valid, or was valid on the given date. OCSP is used as follows for Encryption and signatures:

- Signatures: there are two ways of using OCSP.

  Method 1: To sign a message the signer generates a signature and appends it to the message. When verifying the signature the verifier sends an OCSP query to the VA to check if the certificate was valid at the time the signature was issued.

  Method 2: 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 point is that the response from the VA provides a proof that the signer's certificate is currently valid. The signer might only contact the VA once an hour to obtain the proof of validity for that hour.

  Method 2 is clearly preferred since it reduces the load on the VA: it is not necessary to contact the VA every time a signature is verified. Also note that when using Method 2 the VA could, in principal, also function as the timestamping service.
- 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 [6]. Since the VA is a global service it must be sufficiently replicated if it is going 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 (the VA servers typically use tamper resistant 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. If CERT is revoked the VA responds with all hashes along the path from the CERT's leaf to the root and the VA's signature on the root. If the CERT is not revoked then the VA responds with the paths to the root of two adjacent certificates in the CRT such that the serial number of the first is smaller than the serial number of CERT and the serial of the second is larger then CERT. Since these two certificates are adjacent in the CRT this proves that CERT is not in the CRT. Hence, either way the user obtains a proof for the current status of the certificate. This proof can then be used for signing and encrypting as in OCSP. If currently n certificates are revoked then the length of the proof is $O(\log n)$. In OCSP the length of the validity proof 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 tree proposed by Naor and Nissim [9] and skip-lists proposed by Goodrich [4] are natural data structures for this purpose. Additional data structures were proposed in [1]. When there is a total of n revoked certificates and k new certificates are revoked during the current time period then the size of the update message to the VA servers is $O(k \log n)$ (as opposed to $O(n)$ when using CRT's). Furthermore, the proof of validity for a certificate is $O(\log n)$ as when using CRTs.

2.2 Comparison with SEM architecture

CRLs and OCSP are the most commonly deployed certificate revocation techniques. Some positive experiments with skip-lists are reported in [4]. We compare the SEM architecture with CRLs and with OCSP. Since CRT's and skip-lists are used in the same way as OCSP (query a VA and obtain a proof of validity) everything we say about OCSP applies to these methods as well.

Immediate revocation: Suppose we use CRLs for revocation. Then the user (verifying a signature or encrypting a message) must download a long CRL and verify that the peer's certificate is not on the CRL. Note that the user does not care about most certificates on the CRL. Nevertheless the user must download the entire CRL since otherwise she cannot verify the VA's signature on the CRL. Since CRLs and Δ-CRLs tend to get long they are downloaded infrequently, e.g. once a week or once a month. As a result, certificate revocation might only take affect a month after the revocation occurs. The SEM solves this problem altogether.

Suppose we use OCSP for revocation. Whenever Bob sends mail to Alice he first issues an OCSP query to verify validity of Alice's certificate. He then sends the mail to Alice encrypted using Alice's public key. The encrypted mail could sit in Alice's mailbox for a few hours or days. If during this time Alice's key is revoked (say Alice is fired or looses her private key) there is nothing preventing the holder of Alice's private key from decrypting the mail after revocation. The SEM solves this problem by disabling the private key altogether after revocation.

No need for timestamping when using a SEM: Both OCSP and CRLs require that during signature generation the signer contact a trusted time service to obtain a timestamp for the signature. Otherwise, the verifier cannot determine when the signature was issued with any confidence. The time service is unnecessary when using a SEM. Once a certificate is revoked the corresponding private key can no longer be used to issue signatures. Therefore, when the verifier sees a signature he is explicitly assured that the signer's certificate was valid at the time the signature was issued.

Signature length and format: When using OCSP for generating signatures using Method 2 the signature also contains a proof of validity for the certificate. Hence, the signature is somewhat longer than a standard RSA signature. This is even more pronounced when using CRTs since the length of the proof of validity is $O(\log n)$ where n is the total number of revoked certificates. When using a SEM signature validation is easier than when using Method 2, and in addition signature length is the same as a standard RSA signature.

Shifts validation burden: With the current PKI the burden of validating the certificate is on the peer user: (1) for privacy (encryption) the sender must validate the certificate, (2) for non-repudiation the verifier must validate the certificate. When using a SEM the burden of validating is the reverse: (1) for privacy the receiver validates the certificate, (2) for non-repudiation the signer validates it.

Replication: Since many users need to use the SEM for decryption and signing one might wish to replicate the SEM. However, replicating the SEM across many data centers 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. Hence, the SEM architecture is mainly applicable in environments where OCSP applies. These are mainly medium sized organizations. Most likely the SEM architecture is not applicable to the global Internet.

3 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. 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.)

3.1 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 n_i.

Secret Key. As in RSA, there exists a corresponding secret key $DK_i = (n_i, d_i)$ where $d_i * e_i = 1(\bmod \phi(n))$. 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:

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

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: $\{p_i,\ q_i,\ e_i,\ \text{and}\ d_i, d_i^{sem}\}$ for each user. The first four are generated in the same manner as in standard RSA. The fifth value, d_i^{sem}, is relatively prime with $\phi(n)$ and selected at random from the interval $[1, \phi(n)]$. The last value is set as: $d_i^u = d_i - d_i^{sem} \pmod{n_i}$.

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 6.

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 $m' = (PS_{sem} * PS_u)^{e_i} \pmod{n_i}$. If $m' = m$, the signature is set to: $(PS_{sem} * PS_u) = m^{d_i}$.

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 $c = m^{e_i} \pmod{n_i}$ 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 $PC_{sem} = c^{d_i^{sem}} \pmod{n_i}$ 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 $c' = (PC_{sem} * PC_u)^{e_i} \pmod{n_i}$. If $c' = c$, the cleartext message is: $(PC_{sem} * PC_u) = c^{d_i^u}$.

3.2 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 undesirable. However, in a medium sized organization this "feature" provides key escrow. Indeed, in case Alice is fired the organization can still access her work related files by obtaining her private key from the CA.

If this key-escrow feature is undesirable then it is not difficult to extend the system so that no entity ever known Alice's private key (not even Alice or the CA). To do so, we 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 [3]). This technique has been implemented in [7]. 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 implementation does not use this method. Instead, our implementation allows the CA to do the full key setup.

Immediate Revocation. The notoriously difficult revocation problem is greatly simplified using 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.

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 in mRSA. mRSA certificates, along with private half-keys are distributed to the user and SEM-s in an off-line manner. This is in line with the common certificate issuance and distribution paradigm. In fact, in our implementation (Section 6) there is no need for the CA and the SEM to ever communicate.

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 3.1. In fact, signature and encryption verification steps assure the user of the integrity of the communication with the SEM.

4 Architecture

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

The CA component is a simple add-on to the existing CA. Consequently, it is considered to be 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 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 describe the SEM architecture in more detail. A user request is initially handled by the SEM controller where packet format is checked. Next, the request is passed on to the client manager which first 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 greater 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.

5 Security of the SEM architecture

We briefly review some of the security features of mRSA and the SEM architecture. We give more details in the full version of the paper.

First, recall that the token sent back to the user is simply $t = x^{d^{sem}} \bmod N$ for soe value of x. An attacker sees both x and the token t. In fact, since there is no authentication of the user to the SEM an attacker could obtain this t for any x of its choice. We claim that this information is of no use to an attacker for mounting any type of attack. To see this we argue that any attack that is possible when using the SEM architecture is also possible when the user uses standard RSA. One shows that this type of statement is true using a simulation argument. An attacker attacking standard RSA can simulate the SEM (by picking a random d^{sem}) and thus use the attack on the SEM to mount an attack on standard RSA. We give more details in the full version of the paper.

Suppose an attacker is able to compromise the SEM and expose the SEM's secret key d^{sem}. This enables the attacker to unrevoke certificates. Note, however, that the SEM's key 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 way of doing so 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 must 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.

Finally, note that each user is given her own random RSA modulus N. 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 be unaffected.

6 Implementation

We implemented the entire SEMarchitecture 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 4
3. Client libraries:
   mRSA user functions accessible via an API.

Our reference implementation uses the popular OpenSSL [10] 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 3. 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 pubic 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 piggy-backed. The SEM processes the request and the bundle contained therein as described in Section 4. (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               |
+---------------+-----------------+---------------+----------------+

         Figure 4.1. mRSA Packet Header

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

7 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 assissted 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.

7.1 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 7.1 (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

Keysize	Message Size	Comm. latency
512	102	4.0
1024	167	4.5
2048	296	5.5

Keysize	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 1: RSA results with CRT.

types.

7.2 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 7.2 and 7.2. Each timing includes a message hash computation followed by an exponentiation. Table 7.2 reflects optimized RSA computation where the *Chinese Remainder Theorem (CRT)* is used to speed up exponentiation. Table 7.2 reflects standard 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. In mRSA, neither the SEM nor the user know the factorization of the modulus, hence, it is more appropriate to compare the efficiency of mRSA with the 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.

7.3 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 illustrated in Table 7.3.

It comes as no surprise that the numbers for the slow client in Table 7.3 are very close to the unoptimized RSA measurements in Table 7.2. 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

Keysize	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 2: Standard RSA results without CRT.

Keysize	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

SEM key size	Bundle overhead
1024	8.1
2048	50.3

$4ms$ 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 7.3 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.

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) 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 this revocation system to experiment with it. Our implementation shows 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 desirable.

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 Crypto' 97, Lecture Notes in Computer Science, Vol. 1233, Springer-Verlag, pp. 425–439, 1997.

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

[4] M. Goodrich, R. Tamassia, "Efficient authenticated dictionaries with skip lists and commutative hashing", manuscript.

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

[6] P. Kocher, "A quick introduction to Certificate Revocation Trees (CRTs)" `http://www.valicert.com/company/crt.html`.

[7] 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.

[8] S. Micali, "Enhanced certificate revocation system", Technical memo, MIT/LCS/TM-542, Nov. 1995. `ftp://ftp-pubs.lcs.mit.edu/pub/lcs-pubs/tm.outbox`

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

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

[11] R. Rivest, "Can we eliminate Certificate Revocation Lists", in proceedings of Financial Cryptography'98, pp. 178–183.

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