Fine-Grained Control of Security Capabilities

DAN BONEH
Stanford University
XUHUA DING
Singapore Management University
and
GENE TSUDIK
University of California, Irvine

We present a new approach for fine-grained control over users' security privileges (fast revocation of credentials) 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. The benefits include simplified validation of digital signatures, efficient certificate revocation for legacy systems and fast revocation of signature and decryption capabilities. This paper discusses both the architecture and the implementation of our approach as well as its performance and compatibility with the existing infrastructure. Experimental results demonstrate its practical aspects.

Categories and Subject Descriptors: E.3 [**Data Encryption**]—*Public key cryptosystems*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Algorithms, Security

Additional Key Words and Phrases: Certificate Revocation, Digital Signatures, Public Key Infrastructure

1. INTRODUCTION

We begin this article with an example to illustrate the premise of this work. Consider an organization—industrial, government, or military—where all employees (referred to as *users*) have certain authorizations. We assume that a Public Key Infrastructure (PKI) is available and all users have digital signature,

This work was supported by the Defense Advanced Project Agency (DARPA) under contract F30602-99-1-0530. An earlier version of this paper was presented, in part, at the 2001 Usenix Security Symposium.

Authors' addresses: D. Boneh, Stanford University, Computer Science Dept., Gates 475, Stanford, CA 94305-9045; email: dabo@cs.stanford.edu; X. Ding, School of Information Systems, Singapore Management University, Singapore 25976; email: xuhuad@smu.edu.sg; G. Tsudik, School of ICS, 458 CS Building, Irvine CA 92697-3425; email: gts@ics.usi.edu.

as well as en/de-cryption, 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, the key holder, either Alice herself or an attacker, should be unable to perform any security operations or use any secure applications. Specifically, this might mean:

—The key holder cannot read any secure email. This includes encrypted email that already resides on Alice's email server (or local host) and possible future email erroneously encrypted for Alice. Although encrypted email may be delivered to Alice's email server, the key holder should be unable to decrypt it.

—The key holder cannot generate valid digital signatures on any further messages. However, signatures generated by Alice prior to revocation may need to remain valid.

—The key holder cannot authenticate itself to corporate servers (and other users) as a legitimate user.

Throughout the paper, we use email as an example application. While it is a popular mechanism for general-purpose communication, our rationale also applies to other secure means of information exchange.

To provide immediate revocation it is natural to first consider traditional revocation techniques. Many revocation methods have been proposed; they can be roughly classified into two prominent types: 1) explicit revocation structures such as *Certificate Revocation Lists (CRLs)* and variations on the theme, and 2) real time revocation checking such as the *Online Certificate Status Protocol (OCSP)* [Myers et al. 1999] and its variants. In both cases, some trusted entities are ultimately in charge of validating user certificates. However, the above requirements for immediate revocation are impossible to satisfy with existing techniques. This is primarily because they do not provide fine-grained enough control over users' security capabilities. Supporting immediate revocation with existing revocation techniques would result in heavy performance cost and very poor scalability, as discussed in Section 8.

As pointed out in McDaniel and Rubin [2000], since each revocation technique exhibits a unique set of pros and cons, the criteria for choosing the best technique should be based on the specifics of the target application environment. Fast revocation and fine-grained control over users' security capabilities are the motivating factors for our work. However, the need for these features is clearly not universal since many computing environments (e.g., a typical university campus) are relatively "relaxed" and do not warrant employing fast revocation techniques. However, there are plenty of government, corporate and military settings where fast revocation and fine-grained control are very important.

Organization. This paper is organized as follows. The next section provides an overview of our work. The technical details of the architecture are presented in Section 3 and Section 4, respectively. Then, Section 5 shows four extensions. Sections 6 and 7 describe the implementation and performance results, respectively. A comparison with current revocation techniques is presented Section 8, followed by the overview of related work in Section 8.2 and a summary in Section 9.

2. OVERVIEW

We refer to our approach as the SEM architecture. The basic idea is as follows: We introduce a new entity, referred to as a SEM (SEcurity Mediator): an online semi-trusted server. To sign or decrypt a message, a client must first obtain a message-specific token from its SEM. Without this token, the user cannot accomplish the intended task. To revoke the user's ability to sign or decrypt, the security administrator instructs the SEM to stop issuing tokens for that user's future request. At that instant, the user's signature and/or decryption capabilities are revoked. For scalability reasons, a single SEM serves many users.

We stress that the SEM architecture is transparent to non-SEM users—a SEM is not involved in encryption or signature verification operations. With SEM's help, a SEM client (Alice) can generate standard RSA signatures, and decrypt standard ciphertext messages encrypted with her RSA public key. Without SEM's help, she cannot perform either of these operations. This backwards compatibility is one of our main design principles.

Another notable feature is that a SEM is not a *fully* trusted entity. It keeps no client secrets and all SEM computations are checkable by its clients. However, a SEM is *partially trusted* since each signature verifier implicitly trusts it to have checked the signer's (SEM's client's) certificate status at signature generation time. Similarly, each encryptor trusts a SEM to check the decryptor's (SEM's client's) certificate status at message decryption time. We consider this level of trust reasonable, especially since an SEM serves a multitude of clients and thus represents an organization (or a group).

In order to experiment and gain practical experience, we prototyped the SEM architecture using the popular OpenSSL library. SEM is implemented as a daemon process running on a secure server. On the client side, we built plug-ins for the Eudora and Outlook email clients for signing outgoing, and decrypting incoming, emails. Both of these tasks are performed with the SEM's help. Consequently, signing and decryption capabilities can be easily revoked.

It is natural to ask whether the same functionality can be obtained with more traditional security approaches to fine-grained control and fast credential revocation, such as Kerberos. Kerberos [Neuman and Ts'o 1994], after all, has been in existence since the mid-80s and tends to work very well in corporate-style settings. However, Kerberos is awkward in heterogeneous networks such as the Internet; its inter-realm extensions are difficult to use and require a certain amount of manual setup. Furthermore, Kerberos does not inter-operate with modern PKIs and does not provide the universal origin authentication offered by public key signatures. On the other hand, the SEM architecture is fully compatible with existing PKI systems. In addition, the SEM is only responsible for revocation. Unlike a Kerberos server, the SEM cannot forge user signatures or decrypt messages intended for users. As we discuss later in the paper, our approach is not mutually exclusive with Kerberos-like intra-domain security architectures. We assert that the SEM architecture can be viewed as a set of complementary security services.

2.1 Decryption and Signing in the SEM Architecture

We now describe in more detail how decryption and digital signature generation are performed in the SEM architecture:

- Decryption: suppose that Alice wants to decrypt an email message using her private key. Recall that public key-encrypted email is usually composed of two parts: (1) a short preamble containing a per-message key encrypted with Alice's public key, and (2) the body containing the actual email message encrypted using the per-message key. To decrypt, Alice first sends the preamble to her SEM. SEM responds with a token that enables Alice to complete the decryption of the per-message key and, ultimately, to read her email. However, this token contains no information useful to anyone other than Alice. Hence, communication with the SEM does not need to be secret or authenticated. Also, interaction with the SEM is fully managed by Alice's email reader and does not require any intervention on Alice's part. If Alice wants to read her email offline, the interaction with the SEM takes places at the time Alice's email client downloads her email from the mail server.
- Signatures: suppose that Alice wishes to sign a message using her private key. She sends a (randomized) hash of the message to her SEM, which, in turn, responds with a token (also referred to as a half-signature) enabling Alice to generate the signature. As with decryption, this token contains no useful information to anyone other than Alice.

2.2 Other Features

Our initial motivation for introducing a SEM is to enable immediate revocation of Alice's public key. As a byproduct, the SEM architecture provides additional benefits. In particular, validation of signatures generated with the help of a SEM does not require the verifier to consult a CRL or a revocation authority: the existence of a verifiable signature implies that the signer was not revoked at the time the signature was generated. Consequently, signature validation is greatly simplified.

More importantly, the SEM architecture enables revocation in legacy systems that do not support certificate revocation. Consider a legacy system performing digital signature verification. Often, such systems have no certificate status checking capabilities. For example, old browsers (e.g., Netscape 3.0) verify server certificates without any means for checking certificate revocation status. Similarly, Microsoft's Authenticode system in Windows NT (used for verifying signatures on executable code) does not support certificate revocation. In the SEM architecture, certificate revocation is provided without requiring any change to the verification process in such legacy systems. The only aspect that needs changing is signature generation. Fortunately, in many settings (such as code signing) the number of entities generating signatures is significantly smaller than that of entities verifying them.

Semantics. The SEM architecture naturally provides the following semantics for digital signatures:

> **Binding Signature Semantics:** *a digital signature is considered valid if the public key certificate associated with the private key used to generate the signature was valid at the time the signature was issued.*

According to this definition, all verifiable signatures—by virtue of their existence—are generated prior to revocation and, hence, are considered valid. Binding signature semantics are natural in many settings, such as 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 (e.g., by "losing" her private key). Is the contract valid at time T'? With 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 pretenses, for example, Alice masquerading as Bob, the CA should be allowed to declare at any time that **all** signatures generated with that certificate are invalid.

Implementing binding signature semantics with existing revocation techniques is non-trivial, as discussed in Section 8. Whenever Bob verifies a signature generated by Alice, Bob must also check that Alice's certificate was valid at the time the signature was generated. In fact, every verifier of Alice's signature must perform this certificate validation step. Note that, unless a trusted time-stamping service is involved in generating all of Alice's signatures, Bob cannot trust the timestamp included by Alice in her signatures.

Not surprisingly, 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. (We are assuming here that revocation of Alice's key is equivalent 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.) Once Alice's certificate is revoked, she can no longer generate valid signatures. Therefore, the mere existence of a valid signature implies that Alice's certificate was valid at the time the signature was issued.

3. MEDIATED RSA

We now describe in detail how a SEM interacts with clients to generate tokens. The SEM architecture is based on a variant of RSA that we call Mediated RSA (mRSA). The main idea is to split each RSA private key into two parts using simple 2-out-of-2 threshold RSA [Gemmel 1997; Boyd 1989]. One part is given to a client and the other is given to a SEM. If the client and its 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—to 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 client nor the SEM can decrypt or sign a message without mutual consent. (Recall that a single SEM serves many clients.)

The mRSA method is composed of three algorithms: mRSA key generation, mRSA signatures, and mRSA decryption. We present them in the next section.

3.1 mRSA Key Generation

Similar to RSA, each client U_i has a unique public key and private key. The public key PK_i includes n_i and e_i, where the former is a product of two large distinct primes (p_i, q_i) and e_i is an integer relatively prime to $\phi(n_i) = (p_i - 1)(q_i - 1)$.

Logically, there is also a corresponding RSA private key $SK_i = (n_i, d_i)$ where $d_i * e_i = 1 \bmod \phi(n_i)$. However, as mentioned above, no one party has possession of d_i. Instead, d_i is effectively split into two parts: d_i^u and d_i^{sem} that are secretly held by the client U_i and a SEM, respectively. The relationship between them is:

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

Unlike plain RSA, an individual client U_i cannot generate its own mRSA keys. Instead, a trusted party (most likely, a CA) initializes and distributes the mRSA keys to clients. The policy for authenticating and authorizing clients' key generation requests is not discussed in this paper. Once a client's request is received and approved, a CA executes the mRSA key generation algorithm described below.

mRSA Key Setup. CA generates a distinct set: $\{p_i, q_i, e_i, d_i, d_i^{sem}, d_i^u\}$ for U_i. The first four values are generated as in standard RSA. The fifth value, d_i^{sem}, is a random integer in the interval $[1, n_i]$, where $n_i = p_i q_i$. The last value is set as: $d_i^u = d_i - d_i^{sem} \bmod \phi(n_i)$. We show the protocol in Figure 1.

After CA computes the above values, d_i^{sem} is securely communicated to the SEM and d_i^u is communicated to U_i. The details of this step are elaborated upon in Section 6.

3.2 mRSA Signatures

According to PKCS1 v2.1 [RSA Labs 2002], RSA signature generation is composed of two steps: message encoding and cryptographic primitive computation. The first step is preserved in mRSA without any changes. However, the second step requires SEM's involvement since, in mRSA, a client does not possess its entire private key.

We denote by $EC()$ and $DC()$ the encoding and decoding functions, respectively. Both encodings include hashing the input message m using a collision resistant hash function. For now, we assume the message encoding function

```
Algorithm: mRSA.key (executed by CA)
Let $k$ (even) be a security parameter
(1) Generate random $k/2$-bit primes: $p_i, q_i$
(2) $n_i \leftarrow p_i q_i$
(3) $e_i \overset{r}{\leftarrow} Z^*_{\phi(n_i)}$
(4) $d_i \leftarrow 1/e_i \bmod \phi(n_i)$
(5) $d_i^{sem} \overset{r}{\leftarrow} \{1, \ldots, n_i - 1\}$
(6) $d_i^u \leftarrow (d_i - d_i^{sem}) \bmod \phi(n_i)$
(7) $SK_i \leftarrow (n_i, d_i^u)$
(8) $PK_i \leftarrow (n_i, e_i)$
```

Fig. 1. mRSA key generation algorithm.

```
Algorithm mRSA.sign (executed by User and SEM)
(1) USER: Send $m$ to SEM.
(2) In parallel:
    2.1. SEM:
    (a) If USER revoked return (ERROR);
    (b) $PS_{sem} \leftarrow EC(m)^{d_i^{sem}} \bmod n_i$
        where EC() is the EMSA-PKCS1-v1_5 encoding function, recommended in
        RSA Labs [2002].
    (c) send $PS_{sem}$ to USER
    2.2. USER:
    (a) $PS_u \leftarrow EC(m)^{d_i^u} \bmod n_i$
(3) USER: $S \leftarrow PS_{sem} * PS_u \bmod n_i$
(4) USER: Verify that $S$ is a valid signature on $m$ under the public key $(N, e_i)$. If
    not then return (ERROR)
(5) USER: return (m,S)
```

Fig. 2. mRSA signature algorithm.

$EC()$ is **deterministic**. A user (U_i) generates a signature on a message m as follows:

1. Preprocessing: U_i sends the message m to the SEM.
2. Computation:
   - —SEM checks that U_i is not revoked and, if so, computes a partial signature $PS_{sem} = EC(m)^{d_i^{sem}} \bmod n_i$ and replies with it to the client. This PS_{sem} is the token enabling signature generation on message m.
   - —Concurrently, U_i computes $PS_u = EC(m)^{d_i^u} \bmod n_i$
3. Output: U_i receives PS_{sem} and computes $S_i(m) = (PS_{sem} * PS_u) \bmod n_i$. It then verifies $S_i(m)$ as a standard RSA signature. (This step also verifies the SEM's computation.) If the signature is valid, U_i outputs it.

The algorithm is shown in Figure 2.

We observe that the resulting mRSA and RSA signatures are indistinguishable since: $w^{d_i^u} * w^{d_i^{sem}} = w^{d_i^u + d_i^{sem}} = w^{d_i} \bmod n$. Consequently, the mRSA signature generation process is transparent to eventual signature verifiers, since both the verification process and the signature format are identical to those in standard RSA.

Security. We briefly discuss the security of the signature scheme of Figure 2. At a high level, we require two properties: (1) the user cannot generate signatures after being revoked, and (2) the SEM cannot generate signatures on behalf of the user. For both properties we require existential unforgeability under a chosen message attack. Precise security models for this scheme (used in a slightly different context of multicative version of mRSA) can be found in Bellare and Sandhu [2001] where a proof of security is given.

Randomized encodings. Note that, we assumed above that the encoding procedure $EC()$ is deterministic, as in EMSA-PKCS1-v1_5 [RSA Labs 2002] encoding and Full Domain Hash (FDH) encoding [Bellare and Rogaway 1996]. If $EC()$ is a randomized encoding, such as EMSA-PSS [RSA Labs 2002], we have to make sure both the user and SEM use the same randomness so that the resulting signature is valid. At the same time, to prevent the user from generating signatures without its help, the SEM has to ensure that the random bits used for the encoding are chosen independently at random. Therefore, we cannot simply let the user choose the randomness for the encoding. Instead, the user and the SEM must engage in a two-party coin flipping protocol to generate the required shared randomness. Neither party can bias the resulting random bits. Consequently, these bits can be used as the randomness needed for the encoding function. However, when using deterministic encoding, such as EMSA-PKCS1-v1_5, there is no need for this step.

We note that in the above protocol the user sends the entire message m to the SEM in step (1). For privacy reasons, one might instead consider sending the digest $EC(m)$ the SEM. This would eliminate the difficulty with randomized encodings mentioned above. Unfortunately, the resulting system cannot be shown to be as secure as the underlying RSA signature scheme. Specifically, when only sending $EC(m)$ to SEM, we are unable to prove that the user cannot generate signatures after being revoked. The problem is that, while the user is not revoked, the SEM functions as an unrestricted RSA inversion oracle for the user. For example, the user can use the attack of Desmedt and Odlyzko [1985] to generate signatures after being revoked. A proof of security is still possible, using a strong assumption on RSA: a variant of the "One-more RSA Assumption" [Bellare and Sandhu 2001]. Nevertheless, when using EMSA-PKCS1-v1_5 [RSA Labs 2002] encoding, which is only heuristically secure, it might be fine to send $EC(m)$ to the SEM.

3.3 mRSA Decryption

Recall that PKCS1 [RSA Labs 2002] stipulates that an input message m must be OAEP-encoded before carrying out the actual RSA encryption. We use $EC_{oaep}()$ and $DC_{oaep}()$ to denote OAEP encoding and decoding functions, respectively. The encryption process is identical to standard RSA, where $c = EC_{oaep}(m)^{e_i} \bmod n_i$ for each client U_i. Decryption, on the other hand, is very similar to mRSA signature generation as described above.

```
Algorithm: mRSA.decryption (executed by User and SEM)
(1) USER: $c \leftarrow$ encrypted message
(2) USER: send $c$ to SEM
(3) In parallel:
    3.1. SEM:
    (a) If USER revoked return (ERROR)
    (b) $PC_{sem} \leftarrow c^{d_i^{sem}}$ mod $n_i$
    (c) Send $PC_{sem}$ to USER $U_i$
    3.2. USER
    (a) $PC_i^u \leftarrow c^{d_i^u}$ mod $n_i$
(4) USER: $w \leftarrow (PC_{sem} * PC_u)$ mod $n_i$
(5) USER: OAEP decode $w$. If success, output $m = DC_{oaep}(w)$.
```

Fig. 3. mRSA decryption algorithm.

1. U_i obtains encrypted message c and forwards it to its SEM.
   —SEM checks that U_i is not revoked and, if so, computes a partial clear-text $PC_{sem} = c^{d_i^{sem}} \mod n_i$ and replies to the client.
   —concurrently, U_i computes $PC_u = c^{d_i^u} \mod n_i$.
2. U_i receives PC_{sem} and computes $c' = PC_{sem} * PC_u \mod n_i$. If OAEP decoding of c' succeeds, U_i outputs the clear-text $m = DC_{oaep}(c')$.

Security. We now briefly discuss the security of the mRSA decryption scheme shown in Figure 3. At a high level, we require two properties: (1) the user cannot decrypt ciphertexts encrypted with the user's public key after being revoked, and (2) the SEM cannot decrypt messages encrypted using the user's public key. For both properties we require semantic security under a chosen ciphertext attack. Unfortunately, we cannot quite claim that the above scheme satisfies both properties.

OAEP and its variants are designed to provide chosen ciphertext security for RSA encryption in the random oracle model. The above protocol provides chosen ciphertext security against an attacker who is neither the SEM nor the user. However, OAEP does not necessarily satisfy properties (1) and (2) above. The problem is that the user can employ the SEM as an RSA inversion oracle until the user is revoked. There is no way for the SEM to check whether a partial decryption it generates corresponds to a well-formed plaintext. However, as in the previous section, security can be proven in a weaker model under a strong assumption on RSA. (A detailed proof will be available in the extended version of this paper.)

We note that one way to make sure that the user cannot decrypt messages without the help of the SEM would be to use a Chosen Ciphertext Secure threshold cryptosystem [Shoup and Gennaro 1998; Canetti and Goldwasser 1999]. However, this would render the resulting scheme incompatible with currently deployed encryption systems (based on PKCS1).

3.4 Notable Features

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

CA-based Key Generation. Recall that, in a normal PKI setting with RSA, a private/public key-pair is always generated by its intended owner. In mRSA, a clients' key-pair is instead generated by a CA or some other trusted entity. Nonetheless, a CA only generates client's keys and does not need to keep them. In fact, a CA must erase them to assure that any successful future attack on the CA does not result in clients' keys being compromised. In spite of that, having a trusted entity that generates private keys for a multitude of clients can be viewed as a liability. If CA-based key generation is undesirable then one can use a protocol of Boneh and Franklin [2001] to distributively generate an RSA key between the SEM and the user. The downside is that this requires more work than letting the CA generate the key and give shares to the user and SEM. We note that CA-based key generation enables key escrow (provided that clients' keys are not erased after their out-of-band distribution). For example, if Alice is fired, her organization can still access Alice's encrypted work-related data by obtaining her private key from the CA.

Fast Revocation. The main point of mRSA is that the revocation problem is greatly simplified. In order to revoke a client's public key, it suffices to notify that client's SEM. Each SEM merely maintains a list of revoked clients that is consulted upon every service request. Our implementation uses standard X.509 Certificate Revocation Lists (CRLs) for this purpose.

Transparency. mRSA is completely transparent to entities encrypting data for mRSA clients and those verifying signatures produced by mRSA clients. To them, mRSA appears indistinguishable from standard RSA. Furthermore, mRSA certificates are identical to standard RSA certificates. Thus, the SEM architecture is completely backwards compatible for the signature verifier and message encryptor.

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. In mRSA, a CA remains an off-line entity. mRSA certificates, along with private half-keys are distributed to the client and SEMs in an off-line manner. This follows 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 directly.

SEM Communication. mRSA does not require explicit authentication between a SEM and its clients. A client implicitly authenticates a SEM by verifying its own signature (or decryption) as described in Sections 3.2 and 3.3. These signature and encryption verification steps assure the client of the validity of SEM's replies. Although authentication of a client to a SEM is not required for the security of mRSA itself, it is needed for protection against denial-of-service attacks on a SEM. This can be easily accomplished with the authentication protocol described in Section 5.

Semi-trusted SEM. The SEM cannot issue messages on behalf of unrevoked users nor can it decrypt messages intended for unrevoked users. The worst-case damage caused by a compromise at the SEM is that users who were previously revoked can become unrevoked. This is similar to a compromise at a standard Revocation Authority, which would enable the attacker to unrevoke revoked users.

4. ARCHITECTURE

The overall architecture is made up of three components: CA, SEM, and clients. A typical organizational setup involves one CA, a small set of SEMs and a multitude of clients. A CA governs a small number of SEMs. Each SEM, in turn, serves many clients. (In Section 5 we show how a single client can be supported by multiple SEMs.) The assignment of clients to SEMs is assumed to be handled off-line by a security administrator.

The CA component is a simple add-on to the existing CA and is thus still considered an off-line entity. For each client, the CA component takes care of generating an RSA public key, the corresponding certificate, and a pair of half-keys (one for the client 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 client component consists of the client library, which provides the mRSA signature and mRSA decryption operations. It also handles the installation of the client's credentials at the local host.

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

4.1 Details

We now describe the SEM interaction in more detail. A client's request is initially handled by the SEM controller, which checks the format of the request packet. Next, the request is passed on to the client manager, which performs a revocation check. If the requesting client 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 client. Once the bundle is verified, the request is handled by either the $\mathsf{mRSA}_{\mathsf{sign}}$ or $\mathsf{mRSA}_{\mathsf{decrypt}}$ component. If the SEM maintains client 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 client's certificate are stored locally. In subsequent client requests, the bundle (if present) is ignored and local state is used instead.

The security administrator communicates with a SEM via the administrative interface. This interface allows the administrator to manipulate the revocation list, which, in our implementation is a regular X.509 CRL. (The X.509 format is not a requirement; a CRL can be represented in any signed format as long as it contains a list of revoked clients' certificate serial numbers.)

4.2 Implications of SEM Compromise

Suppose that an attacker compromises a SEM and learns d_i^{sem}. This knowlege can be used to "unrevoke" already revoked clients or block (ignore) future revocations. In the worst case, an attacker could neutralize SEM's mandatory involvement and thus cause the system to degrade to the reliance on normal revocation techniques, such as CRLs. However, we stress that knowledge of d_i^{sem} alone does not enable an attacker to decrypt or sign messages on behalf of a client.

Another interesting side-effect is the observation that there is no need to revoke all clients public keys whose key shares are exposed due to a compromised SEM. As long as a given client is not malicious (or compromised) its public key can remain the same. Specifically, in case of SEM compromise, a CA can simply generate a new pair of mRSA private half-keys for a given client using the same RSA (e_i, d_i, n_i) setting, but with a different SEM. This is possible because there are many ways to randomly "split" a given private exponent d into two parts. More generally, since each SEM client has a distinct RSA setting, even if a number of malicious clients collude, there is no danger to other (non-malicious) clients.

5. EXTENSIONS

We now briefly discuss several simple extensions of mRSA: multi-SEM support, mRSA blind signatures, identity-based mRSA, and authentication of mRSA requests.

5.1 Multi-SEM Support

Since each SEM serves many clients, a SEM failure—whether due to malicious or accidental causes—prevents all of its clients from decrypting data and generating signatures. To avoid having a single point of failure, mRSA can be modified to allow a single client to use multiple SEMs.

The easiest approach is to physically replicate a SEM. While this helps with assuring service availability with respect to accidental (non-malicious) failures, replication does not protect against hostile attacks.

Another trivial solution is to allow a client to be served by multiple SEMs, each with a different mRSA setting. This would require the CA to run the mRSA

```
Algorithm: mRSA.multi-key (executed by CA)
Choose a collision-resistant hash function H : ${0,1}^* \to [1, \ldots, L]$ where
$L \geq 1024$. Let $k$ (even) be the security parameter. Assume client $U_i$ is
authorized to obtain service from $\{\mathsf{SEM}_0, \mathsf{SEM}_1, \ldots, \mathsf{SEM}_m\}$.
(1) Generate random $k/2$-bit primes: $p, q$
(2) $n_i \leftarrow p_i q_i$
(3) $e_i \overset{r}{\leftarrow} Z^*_{\phi(n_i)}$
(4) $d_i \leftarrow 1/e \bmod \phi(n_i)$
(5) $x \overset{r}{\leftarrow} Z_{n_i} - \{0\}$
(6) For each $j \in [0, \ldots, m]$, construct a server bundle for $SEM_j$:
        $d_i^{sem} \leftarrow d_i - \mathtt{H}(x, SEM_j) \bmod \phi(n)$
(7) $SK_i \leftarrow (n_i, x)$
(8) $PK_i \leftarrow (n_i, e_i)$
```

Fig. 4. mRSA key generation for multiple SEMs.

key generation algorithm t times (if t is the number of SEMs) for each client. In addition to the increased computational load for the CA, this approach would entail each client having t distinct certificates or a single certificate with t public keys. The main disadvantage would be for other users (be they SEM clients or not) who would have to be aware of, and maintain, t public keys for a given SEM client.

Our approach allows a SEM client to have a single public key and a single certificate while offering the flexibility of obtaining service from any of a set of SEMs. At the same time, each SEM maintains a different mRSA half-key for a given client. Thus, if any number of SEMs (who support a given client) collude, they are unable to impersonate that client, that is, unable to compute the client's half-key. Multi-SEM support involves making a slight change to the mRSA key generation algorithm, as shown in Figure 4.

To co-operate with SEM_j, the client simply computes $\mathtt{H}(x, SEM_j)$ as the corresponding mRSA half-key for the decryption or signatures.

5.2 Blind Signatures with mRSA

The concept of blind signatures was introduced by Chaum [1983, 1985]. They were originally intended for use in e-cash schemes where a bank (the actual signer) helps a client generate a signature on a message m, without knowledge of either m or the final signature.

mRSA can be easily extended to produce blind signatures. Suppose U_i wants to generate a blind signature on a message m. U_i first masks m by choosing $r \in_{\mathcal{R}} \mathbb{Z}^*_{n_i}$ and setting $m' = r^{e_i} EC(h(m)) \bmod n_i$. Then U_i sends a signature request on m' to SEM and, in the meantime, computes $PS_u = m'^{d_i^u} \bmod n_i$. SEM operates in the same way as in normal mRSA. When U_i receives PS_{sem}, it simply obtains $PS = r^{-1} * PS_u * PS_{sem} = EC(h(m))^{d_i} \bmod n_i$.

5.3 Identity-Based mRSA

The concept of identity-based cryptosystems was introduced by Shamir [1985]. In an identity-based cryptosystem, all clients in an organization share a common cryptographic setting and their public keys are efficiently derived from their identities using a public algorithm. Therefore, personal public key certificates are not needed, which greatly simplifies certificate management and reduces reliance on PKIs. Several identity-based signature systems have been developed in the past, for example, Guillou and Quisquater [1988]. The first practical identity-based encryption system was recently proposed by Boneh and Franklin [2003]. However, no RSA-compatible identity-based cryptosystem has been developed thus far.

It turns out that mRSA can be modified to obtain an identity-based RSA variant (for both encryption and signatures) where clients share a common RSA modulus n and a client's public key e_i is derived from its identity. We briefly outline it here, however, a more detailed description can be found in Ding and Tsudik [2003].

In this variant, only the CA knows the factorization of the common modulus n. For each client U_i, CA computes a unique public key e_i from the U_i's identity (e.g., its email addresses) using a collision-resistant hash function. Then, a CA computes the corresponding $d_i = e_i^{-1} \bmod \phi(n)$. The private key splitting as well as the signature and decryption are all the same as in mRSA, except that a CA does not issue an individual public key certificate to each client. Instead, a CA issues a site-wide (or organization-wide) attribute certificate, which includes, among other things, the common modulus n.

It is well-known that sharing a common modulus among multiple clients in plain RSA is utterly insecure, since knowledge of a single RSA public/private key-pair can be used to factor the modulus and obtain others' private keys. However, this is not an issue in identity-based mRSA since no client possesses an entire private key. However, collusion of a SEM and a single malicious client will result in a compromise of all clients of that SEM. Thus, a SEM in identity-based mRSA must be a fully trusted entity.

5.4 Authenticated mRSA

As discussed earlier, authentication of mRSA client requests can provide protection against denial-of-service (DoS) attacks on a SEM. To address DoS attacks, we can modify both mRSA signature and decryption protocols to allow the SEM to authenticate incoming requests. For example, a client U_i can use its half-key d_i^u to sign each SEM -bound request message. (This can be done, for example, by having the client generate a partial signature on each request that is then verified by the SEM, much in the same manner that a client verifies SEM's reply in Step 5 of Figure 2.)

Although this method is simple and requires no additional setup costs, it does not really prevent DoS attacks, since a SEM would need to perform two modular exponentiations to authenticate each request. A simpler, more cost-effective approach is to use a MAC or keyed hash, for example, HMAC [Bellare et al. 1997], to authenticate client requests. Of course, this would require a shared secret key between a SEM and each client. A CA could help in the generation and distribution of such shared secrets at the time of mRSA key generation. Yet another alternative is to rely on more general encapsulation techniques, such as SSL, to provide a secure channel for communication between SEMs and clients.

6. IMPLEMENTATION

We implemented the entire SEM architecture for the purposes of experimentation and validation. The reference implementation is publicly available at `http://sconce.ics.uci.edu/sucses`. Following the SEM 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 client functions accessible via an API.

The reference implementation uses the popular OpenSSL 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 Unix while the client libraries are available on both Linux and Windows platforms.

In the initialization phase, CA utilities are used to setup the RSA public key-pair for each client (client). The setup process follows the description in Section 3. Once the mRSA parameters are generated, two structures are exported: 1) server or SEM *bundle*, which includes the SEM's half-key d_i^{SEM}, and 2) client *bundle*, which includes d_i^u, the new certificate, and the entire server bundle if SEM is a stateless server.

A SEM bundle is a PKCS7 envelope. It contains d_i^{SEM} encrypted with the SEM's public key and signed by the CA. The client bundle is in PKCS12 format integrating the password privacy and public key integrity modes: it is signed by the CA and encrypted with the client-supplied key which can be derived from a password or a passphrase. (Note that a client cannot be assumed to have a pre-existing public key.)

After issuance, the client bundle is distributed out-of-band to the appropriate client. Before attempting any mRSA transactions, the client must first decrypt the bundle with its key and verify the CA's signature. Finally, the client's new certificate, the SEM bundle and half-key are extracted and stored locally.

To sign or decrypt a message, the client starts with sending a mRSA request with the SEM bundle piggybacked. 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.) If SEM can successfully open its bundle, it checks whether the client's certificate is on the revocation list. If not, SEM follows the protocol and returns a corresponding reply to the client.

6.1 Email Client Plug-In

To further demonstrate the ease of use and practicality of the SEM architecture, we implemented plug-ins for both Eudora email reader and Outlook 2000 email client. When sending signed email, the plug-in reads the client 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. When reading an encrypted email, the plug-in automatically loads the client bundle and decrypts the message by cooperating with SEM. For the sender, all S/MIME capable email composers can encrypt email for mRSA clients without any changes.

6.2 mRSA Email Proxy

An alternative way to use mRSA is through an mRSA-enabled email proxy. A proxy resides on the client's local host, runs in the background (as a daemon on Unix or a TSR program on Windows) and relays email messages between the local host and a remote SMTP server. An outbound email message, if requested, can be processed by the mRSA proxy using the same mRSA protocol as in the plug-in. For inbound email, the proxy can decrypt or verify signatures, if necessary. The main benefit of using a proxy is that it provides a single unified interface to the end-client and all email applications. This obviates the need to customize or modify email clients and offers a greater degree of transparency as well as ease of installation and configuration.

7. EXPERIMENTAL RESULTS

We conducted a number of experiments in order to evaluate the efficiency of the SEM 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.

Table I. Communication Latency

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

Table II. Standard RSA (with CRT) Signature Timings in ms

Key-size (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 III. Standard RSA (without CRT) Signature Timings in ms

Key-size (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

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 I (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.

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 II and III. Each timing includes a message hash computation followed by a modular exponentiation. Table II reflects optimized RSA computation where the *Chinese Remainder Theorem (CRT)* is used to speed up exponentiation (essentially exponentiations are done modulo the prime factors rather than modulo N). Table III 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 client 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 to 3.5 faster for the 1024- and 2048-bit moduli than the unoptimized version. For 512-bit keys, the difference is slightly less marked.

Table IV. mRSA Signature Timings in ms

Key-size (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 V. Bundle Overhead in mRSA with a SEM in a Stateless Mode (in milliseconds)

SEM key size	Bundle overhead
1024	8.1
2048	50.3

7.3 mRSA Measurements

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

It comes as no surprise that the numbers for the slow client in Table IV are very close to the unoptimized RSA measurements in Table III. 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 V 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. RELATED WORK

Our system is constructed on top of a 2-out-of-2 threshold RSA algorithm, for the purpose of instant certificate revocation. In the following, we compare it with others' work with related functionality as well as those with similar cryptographic setting.

8.1 Current Revocation Techniques

Certificate revocation is a well-recognized problem in all current PKIs. Several proposals attempt to address this problem. We briefly review these proposals and compare them to the SEM architecture. For each, we describe how it applies to signatures and encryption. We refer to the entity validating and revoking certificates as the Validation Authority (VA). Typically, a VA and a CA are one and the same. However, in some cases (such as OCSP) these are separate entities.

CRLs and Δ-CRLs: these are the most common ways to handle certificate revocation. The Validation Authority (VA) periodically posts a signed list (or another data structure) containing all revoked certificates. Such lists are placed on designated servers, called CRL Distribution Points. Since a list can get quite long, a VA may post a signed Δ-CRL that only contains the list of certificates revoked since the last CRL was issued. In the context of encrypted email, at the time email is sent, the sender checks if the receiver's certificate is included in the latest CRL. To verify a signature on a signed email message, the verifier first checks if (at present time) the signer's certificate is included in the latest CRL.

OCSP: the Online Certificate Status Protocol (OCSP) [Myers et al. 1999] avoids the generation and distribution of potentially long CRLs and provides more timely revocation information. To validate a certificate in OCSP, the client sends a certificate status request to the VA. The VA sends back a signed response indicating the status (revoked, valid, unknown) of the specified certificate.

We remark that the current OCSP protocol prevents one from implementing binding signature semantics; it is impossible 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 OCSP. 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.

Certificate Revocation Trees: Kocher [1998] suggested an improvement over OCSP. 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 signature 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 client 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 CRTs is that, each time a certificate is revoked, the whole CRT must be recomputed and distributed in its entirety to all VA servers. A data structure allowing for dynamic updates would solve this problem since a 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 [2000] and skip-lists proposed by Goodrich [Goodrich et al. 2001] are natural and efficient for this purpose. Additional data structures were proposed in [Aiello et al. 1998]. 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 the certificate's validity is $O(\log n)$, the same as with CRTs.

A note on timestamping. Binding signature semantics (Section 2.2) for signature verification states that a signature is considered valid if the key used to generate the signature was valid at the time of 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 client's (signed) message. This proves to any verifier that a signature was generated at a specific time. All the techniques discussed above require a signature to contain a trusted timestamp indicating when a signature was issued. There is no need for a trusted time service to implement binding signature semantics with the SEM architecture. This is because a SEM can be used to provide a secure time-stamping service as part of its mandatory involvement in each client's signature.

8.2 Two-party RSA

Several other research results developed schemes similar to the SEM architecture although in different security domains. Among them, Yaksha [Ganesan 1996] and S-RSA [MacKenzie and Reiter 2001a,b] are the schemes conceptually closest to ours. Both Yaksha and S-RSA involve 2-party RSA function sharing where clients do not possess complete RSA private keys and rely on an on-line server to perform certain private key operations.

The Yaksha system is a reusable security infrastructure that includes key exchange and key escrow. It allows a legitimate authority to recover clients' short-term session keys without knowing their long-term private keys. The client and the Yaksha server separately hold two shares such that their product forms a complete RSA private key for the client. When a Yaksha server receives a request for generating a session key, it chooses the key at random, encrypts it with the client's public key and decrypts it partially with the corresponding key share so that the result can be decrypted by the client using the other share.

Compared with our scheme, Yaksha is more expensive. A Yaksha client is unable to perform its local computation before it receives the server's result, whereas, the client's and SEM's computations in our scheme are executed concurrently. Also, a Yaksha server is a fully trusted entity and its compromise completely breaks the system security. In contrast, a SEM is only partially trusted; its compromise can only impair the intended service. Furthermore, a Yaksha server is a single point of failure and not scalable in that it serves for all users in the system. Our scheme allows multiple SEMs, each serving (possibly overlapping) subsets of clients.

Another related result, S-RSA, is due to MacKenzie and Reiter [2001a,b]. It aims to safeguard password-protected private keys on a captured networked device from offline dictionary attacks. In this scheme, the client's share is derived from its password; the server's share is contained in a token encrypted with the server's public key and stored in the device. The sum of the two shares forms the client's private RSA key. When needed, the encrypted token is sent to the server, which extracts the key share and helps the client to issue a signature. The client is also able to notify the server to disable its key share by revealing certain secret information. Although the underlying cryptographic algorithms are similar to ours, the goals are fundamentally different: we focus on fine-grained control and fast revocation while S-RSA aims to protect networked devices.

Many other two-party schemes have been proposed in the literature. For example, Boneh and Franklin [2001] showed how to share the RSA key generation function between two parties. Nicolosi et al. [2003] designed a proactive two-party Schnorr signature scheme. MacKenzie and Reiter [2001c] developed a provable secure two-party DSA signature scheme. However, none of these schemes are used in the content of revocation of security privileges.

9. SUMMARY

We described a new approach to certificate revocation and fine-grained control over security capabilities. Rather than revoking the client's certificate our approach revokes the client'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 fast—when its certificate is revoked, the client can no longer decrypt or sign messages, (2) with binding signature semantics, there is no need to validate the signer's certificate as part of signature verification, and (3) our revocation technique is transparent to the peers since it uses standard RSA signature and encryption formats.

We implemented the SEM architecture for experimentation purposes. Our measurements show that signature and decryption times are not significantly higher from the client's perspective. Therefore, we believe the SEM architecture is appropriate for small- to medium-sized organizations where tight control of security capabilities is desired. The SEM architecture is clearly not appropriate for the global Internet or for educational campus-like environments.

REFERENCES

AIELLO, W., LODHA, S., AND OSTROVSKY, R. 1998. Fast digital identity revocation. In *Advances in Cryptology—CRYPTO '98*, H. Krawczyk, Ed. Number 1462 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

BELLARE, M., CANETTI, R., AND KRAWCZYK, H. 1997. HMAC: Keyed-hashing for message authentication. Internet Request for Comment RFC 2104, Internet Engineering Task Force. Feb.

BELLARE, M. AND ROGAWAY, P. 1996. The exact security of digital signatures: How to sign with rsa and rabin. In *Advances in Cryptology—EUROCRYPT '96*, U. Maurer, Ed. Number 1070 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

BELLARE, M. AND SANDHU, R. 2001. The security of practical two-party rsa signature schemes, http://www.cs.ucsd.edu/users/mihir/papers/splitkey.html.

BONEH, D. AND FRANKLIN, M. 2001. Efficient generation of shared RSA keys. *J. ACM (JACM) 48*, 4, 702–722.

BONEH, D. AND FRANKLIN, M. 2003. Identity-based encryption from the Weil Pairing. *SIAM J. Comput. 32*, 3, 586–615. Extended abstract in Crypto '01.

BOYD, C. 1989. Digital multisignatures. *Cryptography and Coding*, 241–246.

CANETTI, R. AND GOLDWASSER, S. 1999. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *Advances in Cryptology—EUROCRYPT '99*, J. Stern, Ed. Number 1592 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

CHAUM, D. 1983. Blind signatures for untraceable payments. In *Advances in Cryptology—CRYPTO '82*, R. L. Rivest, A. Sherman, and D. Chaum, Eds. Plenum Press, New York, 199–203.

CHAUM, D. L. 1985. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM 28*, 10 (Oct.), 1030–1044.

DESMEDT, Y. AND ODLYZKO, A. 1985. A chosen text attack on the rsa cryptosystem and some discrete logarithm schemes. In *Advances in Cryptology—CRYPTO '85*.

DING, X. AND TSUDIK, G. 2003. Simple identity-based encryption with mediated RSA. In *Progress in Cryptology—CT-RSA 2003*. LNCS 2612. Springer-Verlag, Berlin Germany.

GANESAN, R. 1996. The yaksha security system. *Commun. ACM 39*, 3 (Mar.), 55–60.

GEMMEL, P. 1997. An introduction to threshold cryptography. *RSA CryptoBytes 2*, 7.

GOODRICH, M., TAMASSIA, R., AND SCHWERIN, A. 2001. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proceedings of DARPA DISCEX II*.

GUILLOU, L. AND QUISQUATER, J. 1988. Efficient digital public-key signature with shadow. In *Advances in Cryptology—CRYPTO '87*, C. Pomerance, Ed. Number 293 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Santa Barbara, CA, USA, 223–223. Lecture Notes in Computer Science No. 293.

KOCHER, P. 1998. On certificate revocation and validation. In *Financial Cryptography—FC '98, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1465*. 172–177.

RSA LABS. 2002. PKCS #1v2.1: RSA cryptography standard. Tech. rep., RSA Laboratories. June.

MACKENZIE, P. AND REITER, M. 2001a. Delegation of cryptographic servers for capture-resilient devices. In *8th ACM Conference on Computer and Communications Security*, P. Samarati, Ed. ACM Press, Philadelphia, PA, USA.

MACKENZIE, P. AND REITER, M. 2001b. Networked cryptographic devices resilient to capture. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press, Oakland, CA, 12–25.

MACKENZIE, P. AND REITER, M. 2001c. Two-party generation of DSA signatures. In *Advances in Cryptology—CRYPTO '2001*, J. Kilian, Ed. Number 2139 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 137–154.

MCDANIEL, P. AND RUBIN, A. 2000. A Response to 'Can We Eliminate Certificate Revocation Lists?'. In *Proceedings of the 4rd Conference on Financial Cryptography (FC '00)*, Y. Frankel, Ed. Number 1962 in Lecture Notes in Computer Science. International Financial Cryptography Association (IFCA), Springer-Verlag, Berlin Germany, Anguilla, British West Indies.

Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C. 1999. RFC 2560: Internet public key infrastructure online certificate status protocol—OCSP.

Naor, M. and Nissim, K. 2000. Certificate revocation and certificate update. *IEEE J. Sel. Areas Comm. 18*, 4 (Apr.), 561–570.

Neuman, C. and Ts'o, T. 1994. Kerberos: An authentication service for computer networks. *IEEE Computer 32*, 9 (September).

Nicolosi, A., Krohn, M., Dodis, Y., and Mazières, D. 2003. Proactive two-party signatures for user authentication. In *Symposium on Network and Distributed Systems Security (NDSS '03)*. Internet Society, San Diego, CA.

Shamir, A. 1985. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology—CRYPTO '84*, G. Blakley and D. Chaum, Eds. Number 196 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 47–53.

Shoup, V. and Gennaro, R. 1998. Securing threshold cryptosystems against chosen ciphertext attack. In *Advances in Cryptology—EUROCRYPT '98*, K. Nyberg, Ed. Number 1403 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1–16.

Received January 2003; revised February 2003; accepted June 2003