Int. J. Inf. Secur. (2005)
DOI 10.1007/s10207-005-0076-x

SPECIAL ISSUE PAPER

Gabriel Vanrenen · Sean Smith · John Marchesini

Distributing security-mediated PKI

Published online: 19 July 2005
© Springer-Verlag 2005

Abstract The security-mediated approach to PKI offers several advantages, such as instant revocation and compatibility with standard RSA tools. In this paper, we present a design and prototype that addresses its trust and scalability problems. We use trusted computing platforms linked with peer-to-peer networks to create a network of trustworthy mediators and improve availability. We use threshold cryptography to build a back-up and migration technique which allows recovery from a mediator crashing while also avoiding having all mediators share all secrets. We then use strong forward secrecy with this migration, to mitigate the damage should a crashed mediator actually be compromised.

Keywords SEM · Peer-to-peer · Trusted computing

1 Introduction

The security-mediated approach to PKI (by Boneh et al. [3, 4]) offers many advantages. However, it has some disadvantages with regard to trust and scalability: each user depends on a mediator that may go down or become compromised. In this paper, we apply tools including peer-to-peer computing and trusted computing platforms to distribute the security-mediated approach to PKI, and thus preserve its advantages while overcoming its scalability, reliability, and trust problems. Section 2 reviews the security-mediated approach, and discusses its advantages and disadvantages. Section 3 discusses the tools we apply to this problem. Section 4 discusses the design we build with these tools. Section 5 discusses our prototype. Section 6 discusses some related approaches. Section 7 discusses some conclusions and future work.

G. Vanrenen (✉) · S. Smith · J. Marchesini
Department of Computer Science/PKI Lab, Dartmouth College,
Hanover NH 03755, USA
E-mail: gabriel.vanrenen@alum.dartmouth.org,
sws@cs.dartmouth.edu, carlo@cs.dartmouth.edu

2 SEM

2.1 Motivation

Because it does not require sharing secrets a priori, public key cryptography can enable a variety of secure communications among parties that have never met. For example, in the case of digital signatures, a *keyholder* takes an action with his private key, and the *relying party* verifies this action against the corresponding public key. However, the correctness of the trust decisions a relying party makes, based on this verification, depends on the assumption that the entity knowing the matching private key possesses certain properties (e.g., "is student Alice at Dartmouth College"). In practice, a *certificate* is a signed assertion binding a public key to such properties.

Public key infrastructure (PKI) refers to the surrounding technology and operations necessary for making public key cryptography work in practical applications. A primary role of PKI is creating certificates to indicate such bindings and distributing the certificates to relying parties. When the binding that a certificate expresses ceases to hold, this certificate needs to be *revoked*, and this revocation information needs to propagate to relying parties, lest they make incorrect trust judgments regarding that public key.

Consequently, fast and scalable certificate revocation has been area of active research in recent years (e.g., [24, 25]). In their *Security Mediator*[1] *(SEM)* approach, Boneh et al. [4] proposed a system that revokes the ability of the keyholder to use a private key, instead of (or in addition to) revoking the certificate attesting to the corresponding public key. If a private key operation cannot take place after the binding has been revoked, then a relying party does not need to check the revocation status.

2.2 Architecture

The SEM approach is based on *mediated RSA (mRSA)*, a variant of RSA which splits the private key of a user into

[1] Also referred to as "semi-trusted mediator."

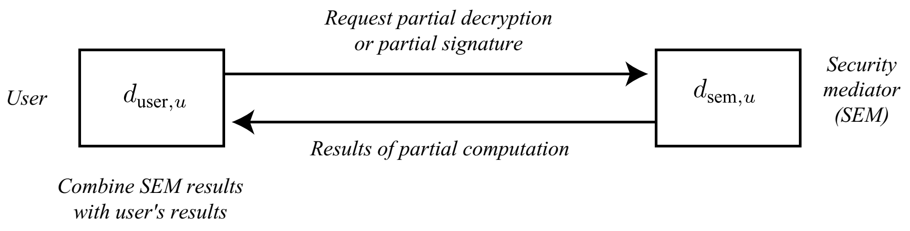
Request partial decryption or partial signature
User
$d_{\text{user},u}$
$d_{\text{sem},u}$
Security mediator (SEM)
Results of partial computation
Combine SEM results with user's results

Fig. 1 In security-mediated RSA, the user has a key pair (as in ordinary RSA). However, the user u's private exponent is divided into two pieces: one piece, $d_{\text{user},u}$, is held by the user, and th other, $d_{\text{sem},u}$, is held by the security mediator. To carry out an operation with the user's private key, both parties must participate

two parts. As in standard RSA, each user has a public key (n_u, e_u) and a private key d_u, where n is the product of two large primes, $\gcd(e_u, \phi(n_u)) = 1$, and $d_u * e_u = 1 \pmod{\phi(n_u)}$. The public key of a user u is the same as in standard RSA, as is the public-key operation. However, in mediated RSA, we divide the private key d_u into two "halves" $d_{\text{sem},u}$ and $d_{\text{user},u}$, where $d_u = d_{\text{sem},u} + d_{\text{user},u} \pmod{\phi(n_u)}$.

In the SEM approach, the half $d_{\text{user},u}$ is held by the user and the half $d_{\text{sem},u}$ is held by the mediator (We note that $d_{\text{sem},u}$ and $d_{\text{user},u}$ are each statistically unique for each user u.) This division of the private key requires changes to the standard RSA key setup because a SEM must not know $d_{\text{user},u}$ and a user must not know $d_{\text{sem},u}$. So, a trusted party (e.g., a CA) performs key setup by generating a statistically unique $\{p_u, q_u, e_u, d_u, d_{\text{sem},u}\}$ for a user u. The private key d_u is generated in the standard manner, but is communicated to neither the SEM nor the user. Instead, $d_{\text{sem},u}$ is chosen as a random integer in $[0, n_u - 1]$, and $d_{\text{user},u}$ is then calculated as $d_{\text{user},u} = d_u - d_{\text{sem},u} \pmod{\phi(n_u)}$.

Because the private key d_u is split into two pieces, private key operations require the participation of both the user and the SEM: e.g., each party raises the message to its share of the exponent, modulo n, and the results are then multiplied, also modulo n (see Fig. 1.) Thus the full private key never needs to be reconstructed; also, since the user initiates these operations, the SEM does not learn exactly what the user is signing or decrypting.

2.3 Advantages

The SEM approach provides several advantages. Since these essentially are standard RSA operations, a SEM PKI is compatible with most legacy public-key cryptography tools. Since the mediator is not involved in the public key operations (encryption and signature verification), those users who do not hold SEM-protected private keys do not even need to know about these mediators. Because of this, the SEM network can be seamlessly integrated with legacy infrastructure. Since a full private-key operation can occur only if the SEM believes the user's key pair is valid, then the system can revoke a key pair by having the SEM refuse to carry out its half of the operation. This approach can reduce or even eliminate (in the case of revocation due to administrative action, such as a user ceasing employment) the need for certificate revocation lists—since, as noted earlier, a private-key operation (such as signature or decryption) cannot occur after revocation.

Furthermore, the SEM itself gains no useful information in servicing users. When decrypting, the SEM receives the ciphertext but is only able to partially decrypt it, so no useful information could be gained by a malicious SEM. For signature generation, a user sends the SEM a hash of the message which the SEM uses to generate the signature. This also contains no information about the cleartext of the message itself, so a user's data is kept confidential.

Additionally, the compromise of a single SEM does not compromise the secret keys of any users. Instead, the attacker is able to revoke the security capabilities for users connected to the SEM. Although Boneh et al. state that attackers could unrevoke revoked certificates, this can be prevented by having honest SEMs permanently delete $d_{\text{sem},u}$ upon revocation.

2.4 Disadvantages

However, the initial SEM approach has scalability disadvantages. In a large-scale distributed system, we must allow for problems such as mobile users, network partitioning, crashing of machines, and occasional compromise of servers. To accommodate a large population, we could add multiple SEMs. However, if, for a user u, the half $d_{\text{sem},u}$ lives on exactly one SEM, then we have many potential problems:

- temporary denial of service if the network is partitioned;
- permanent denial of service if the SEM suffers a serious failure;
- inability to revoke the key pair if an adversary compromises a SEM and learns its secrets.

In their original paper [4], Boneh et al. did propose one way to distribute the SEM architecture by using a stateless model in which a user can connect to any node in a distributed SEM network. However, this approach required that the entire network have a single RSA key pair, so that any node can access a user u's encrypted $d_{\text{sem},u}$ bundled with each request. This network-wide key pair could either be stored on each node through replication or shared securely among islands using threshold cryptography.

– In the first case, compromise of a single node is potentially easier (due to replication) and causes damage to the entire network.
– In the latter case, each user request requires distributed computation among nodes, which hurts performance.

In either case, the user is at risk if she connects to a compromised SEM.

More recently, in [3], Boneh et al. introduced *Multi-SEM* support as another way to increase the robustness of the SEM network. In this approach, a single user is allowed to be served by multiple SEMs. One way to do this is to completely replicate SEM servers; however, such replication does not protect against hostile attacks since an attacker can gain access to all of the secret data from a single server. The other approach described by Boneh et al. is modify the mRSA algorithm slightly to generate k different mRSA key-half pairs so that the user can contact one of k mediators in order to complete a decryption or signature generation operation. This second approach allows the user to have only a single public key and certificate while mitigating the risk of single SEM failure. Additionally, the SEM servers cannot collude to learn the user's half of the private key. However, Boneh et al. do not discuss a revocation scheme for safely distributing revocation status to all the SEM's for a single user. Without a clear scheme, it is unclear whether a compromised SEM could distribute a fake revocation status to the other SEM's that the user can access, thereby denying service to the user. Additionally, this scheme protects only against k compromises (where k is less than or equal to the number of SEM's), after which the CA must become involved again to create another set of uncompromised k key-half pairs. Finally, no protection against collusion between a compromised SEM and a user is given.

3 Tools

To address the problem of distributing SEM, we use several tools.

3.1 Trusted computing platforms

In some sense, PKI is all about *trust*. Relying parties want to know whether they can trust that the party carrying out some private key operation possesses certain properties. From the cryptography, relying parties only know that the operations can only be carried out by parties who know the certain private key; the rest of the components necessary for the trust judgment comes from the technology and operations supporting the keys and the cryptography.

Adding security mediators to PKI adds some new components: relying parties need to be able to trust that a mediator will use and delete each user u's $d_{\text{sem},u}$ when appropriate, and not transmit it further. Given the permeability of current commodity machines (and the fallibility of humans), these assumptions may already be questionable. However, when we add distribution to the picture, these assumptions become even weaker. The more we distribute the SEMs throughout a network, the less foundation relying parties have for trust.

Consequently, the question naturally arises of whether we can augment the technology that comprises the PKI to increase the trust the relying parties might have. In this case, remote machines subject to compromise threaten trust. In recent years, various *trusted computing platform* technologies have been proposed to increase the assurance a stakeholder might have that computation on a remote platform satisfies certain correctness and security properties [34]. Some of these technologies have actually made it out of the laboratory and into commercial production (such as the IBM 4758 *secure coprocessor* [35], and the 1.1b *Trusted Platform Module (TPM)* from the *Trusted Computing Group (TCG)*). Others are looming.

In our design, we consider the potential of such a trusted computing platform. Our basic requirements are a general-purpose computing environment and cryptographic protections, coupled with protection against physical attacks, and an *outbound authentication* (or *attestation*) scheme which lets software applications running on the coprocessor authenticate themselves to remote parties [33]. This platform thus gives us a safe and confidential environment in remote environments. If a user trusts our software is not flawed, then the user can also trust that software executed cannot be altered by adversaries and the user may also remotely authenticate instances of this software.

3.2 Peer-to-peer

We would like to make it easy for users to find SEMs (and for SEMs to find each other), and we would like this functionality to persist despite failures and (potentially) malicious attacks. *Peer-to-peer networking (P2P)* (embodied by technology such as Gnutella) is an attractive choice here. With P2P, communication does not rely on a central entity to forward requests or messages. Rather, each entity either tries to satisfy a request itself, or forwards it to its neighbors, in the spirit of the older distributed concept of *diffusing computation* [2].

This decentralization is a key benefit of the peer-to-peer network, as it removes any central entity necessary for the system to function. Without a central controlling server, the network's survivability increases by causing denial of service attacks to be much more difficult (as the RIAA has found to its dismay). Additionally, the damaging effects of network partitions are potentially alleviated by standard P2P communication algorithms, as a new path to a destination may be found.

3.3 Threshold cryptography

We will also need to distribute critical secrets across multiple SEMs, for resilience against attack. Here, we can use the standard technique of *threshold cryptography* [31]. Given a secret y and parameters $t < k$, we construct a degree t polynomial that goes through the point $(0, y)$, and choose k points on this polynomial as *shares* of y. Any t shares suffices to reconstruct the polynomial and hence y, but fewer than t shares give no information.

3.4 Strong forward security

We need to accommodate the fact that machines (even "trusted computing platforms") may be compromised, and the secrets they store may become exposed to the adversary. To mitigate the damage of such potential exposure, we can use the technique of *strong forward security (SFS)* [5]. We divide time into a sequence of clock periods, and use a cryptographic system such that even if the private key for a given period is exposed, use of the private key in previous or future sessions is still secure. Burmester et al. give two examples of strong forward secure schemes, one for any public key cryptosystem and another for use in an El Gamal key escrow system.

4 Design

4.1 Architecture

Our Distributed SEM architecture consists of the network of server nodes along with software to allow for the distribution of the SEM approach.

4.1.1 Network

In our basic architecture, we envision SEMs as trustworthy *islands* distributed throughout the network. We use a trusted computing platform coprocessor to house each SEM and thus give it a foundation for this trustworthiness. As noted earlier, this technology also lets each island have a key pair and certificate chain that establishes the entity who knows the private key is an instantiation of our island software on an untampered device. Thus, users can authenticate islands, and islands can authenticate each other.

Each island will house resources that enable it carry out services. When a user requests such a service, we use P2P techniques to carry the request to the proper island, and carry the response back to the user. (As we discuss below, individual islands will also house resources other islands need; we can use P2P there as well.)

Our combination of trusted computing and peer-to-peer networking comes from the concept of the *Marianas network* developed at Dartmouth College. Both of those technologies have recently been gaining recognition in both academic and commercial settings and their combination allows us to create a new type of distributed, trustable third party with various benefits. Additionally, it is designed to be scalable and online, ensuring that it fulfills the demands of emerging global infrastructure. Although a "trusted third party" can be regarded as an "Orwellian Big Brother," we aim to alleviate any concerns from the simple fact that an entire Marianas network does not need to be under the control of a single individual or entity. Instead, as long as a node is configured properly, it may be under the control of anyone. With the use of trusted computing at each node and the proper configuration, each node even offers resistance against insider attack.

However, both the network itself and applications built on top of it should allow for occasional compromise of individual nodes. In the future, the Marianas network will also attempt to allow for heterogeneous trusted hardware, although this paper will not deal with issues relating to that. Since a Marianas network is distributed, it will be closer to all clients, ensuring reliable access to its services. Distributing Security-Mediated PKI is just one application that can be implemented on our peer-to-peer network of trusted hardware and is intended to demonstrate the feasibility and benefits of such a network.

4.1.2 Migration

Despite physical protections, an individual island may still become compromised and reveal its data to the adversary. An individual island may also become unavailable, due to crash or partition. To handle these scenarios, we build a *migration* scheme based on threshold cryptography and strong forward security.

Migration aims to be a secure way to avoid replication in any distributed network (and may have applications outside of the scope of this paper). In our extension of the SEM architecture, we use migration to update the secret held by an island and migrate it to another one. The general steps required to set up and execute migration are discussed below. We have designed migration for use in any distributed application and not necessarily on our peer-to-peer network of trusted hardware nodes, although we experiment only on our network and discuss its setup and execution in terms of that.

When we initially create a secret x and transmit it to an island L, we also split it into k shares using threshold cryptography. We securely transmit each share of x to a different island. (Additionally, the shares may be proactively updated using techniques described in the literature [10, 11, 13, 14] so that an attacker may not slowly acquire enough shares to reconstruct x.) The k islands that receive shares of x can be chosen pseudorandomly so that they are most likely distributed evenly in the SEM network, mitigating denial of service risks like network partition. Alternatively, a pseudorandom algorithm could be replaced with a smarter one, such as a load-balancing scheme or other scheme (e.g. based on network proximity to the user). After those steps are complete, the secret is stored both on the primary island L and on k other islands, so an attacker must either compromise L or compromise t of the k islands in order to get x.

When the island L is unavailable to fulfill a request that requires x, then the requester will have to be redirected to another island M, and the shareholders will need to participate in reconstructing x there. However, since the original island L may have been compromised, x must be updated using strong forward security so that the old version on L is rendered useless.

The general migration scheme is executed as follows:

- The user tries to connect to the assigned island L, but fails.
- The user then connects to another island M instead. This M may be chosen using one of the algorithms available for distributing the shares of x as described above (e.g. pseudorandom, load-balancing, etc). Note that M should be chosen by the network in a distributed way such that the user u and L cannot choose M. This is important because we do not want a single compromised node or user to select M.
- The islands that hold shares of x are contacted and this x is updated using strong forward security. As discussed below, this update may or may not involve reconstruction of x, depending on the method chosen. Generally, the strong forward security scheme will vary depending on the how the secret x is generated.
- Strong forward security results in M storing the updated secret.
- Migration is complete and M can then fulfill the user's request.

We summarize the benefits provided by our general migration scheme as follows:

- **Uninterrupted Service:** The use of threshold cryptography to distribute the secret x across the network allows a user to get service even when their node is not available.
- **Secure Service after Node Compromise:** The combination of strong forward security with threshold cryptography allows users to update their secret on another uncompromised node and get secure service even when their assigned node has been compromised.
- **Rare Use of Distributed Computation:** Migration is intended to be used rarely and, in the common case, the user must only interact with a single island L in order to access their full secret x. This facilitates easy and fast access to services provided by our network.

The migration process is aimed at improving the security of any application that uses it, so we now discuss its strengths and weaknesses in various situations. This analysis applies only to the general concept of migration as defined by the combination of strong forward security and threshold cryptography. In specific applications of migration, there may be other caveats involved with the migration of the secret x. Such cases are present in the application introduced in this paper and are examined in Sect. 4.4.

During a migration, there are a few cases that we must consider. Although migration of a secret x can be performed directly from one island L to another island M, let us assume that the secret x must be reconstructed from shares stored on t of the k other islands to cover the case where L has been removed from the Marianas network. Then, to analyze the security of migration we must examine the status of L, M and the other islands that hold shares of x. Depending on which parties have been compromised, the secret x may be compromised before or after the migration. We assume here that forward security can be achieved, as it differs between applications.

If L is compromised, then, before migration, the secret x is known to the attacker that compromised the island. However, forward security updates the secret x in a way such that the new secret y replaces x. Additionally, y is not sent back to L, which may still be compromised, so the attacker's stolen secret becomes useless. Clearly, there is a period between the exposure (or compromise) time of x and the time when that compromise is discovered. However, as discussed in Sect. 3, the use of trusted computing platforms in the Marianas network causes any physical attacks to be detected and stopped by the shutdown of the platform, within some window of time.

If M is compromised, then the secret x is migrated to M so the attacker that compromised M gains access to it. However, as stated above, with trusted computing platforms, the window of time before this compromise is discovered should be small, so another migration from M can be performed, rendering x useless.

If islands holding shares of x are compromised, then, because of the properties of threshold cryptography, as long as less than t of the k islands holding shares of x are not compromised, then the attacker cannot reconstruct x. Since proactive updating of shares limits the time in which at least t of the islands must be compromised, it would be very difficult to reveal x using this method. However, if x is revealed to an attacker in this way, then this situation is the same as the first where L is compromised.

4.2 SEM operations

To use our architecture for SEM, each island acts as a SEM mediator, holding $d_{\text{sem},u}$ for a number of clients. We distribute load across the islands by, at key generation, assigning users to different SEMs. (We could also distribute load via migration.) As with the original SEM architecture, a user's $d_{\text{sem},u}$ is stored in full only on one island.

Key generation

In the original SEM scheme [4], a CA generates key pairs for users and splits d into two halves. In our variant, the CA must additionally share $d_{\text{sem},u}$ to k islands in the network using threshold cryptography (see Fig. 2.) If we're using an online CA during migration, we can get extra protection by encrypting these shares for the CA only.

Also stored with those shares is the user's identity and the revocation status of the user's key pair (initialized to "false," not revoked). For key generation, the CA must be able to prove its identity to the islands; otherwise, the islands will ignore its request. If we desired an escrow service to allow authorized decryption of data after revocation (or if we

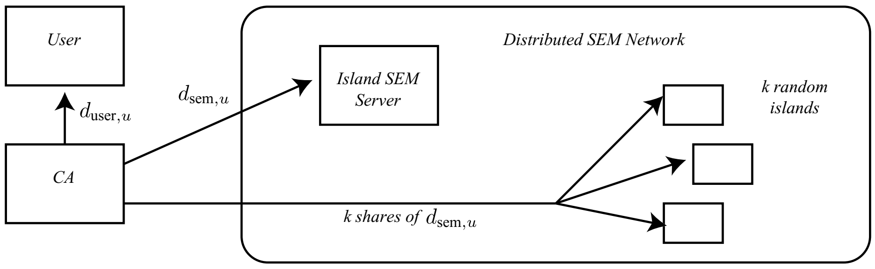

Fig. 2 To distribute the SEM approach to PKI, we establish a network of security mediators. For each user u in addition to assigning the key portion $d_{\text{sem},u}$ to a designated mediator node, we also use Shamir's secret sharing to divide $d_{\text{sem},u}$ into k pieces and distribute each piece to another mediator

do not decide to use the CA during migration, as discussed below), we also distribute shares of the full secret key d_u.

Revocation

If the island that holds $d_{\text{sem},u}$ and revocation information for a user u goes down, then the other islands must be able to determine whether the user's key pair has been revoked. We accomplish this by, during revocation, having the shareholders as well as original island update the revocation status for that key pair. In our initial vision, we delete the shares of $d_{\text{sem},u}$ that are stored on k other islands.

Assume that a user's $d_{\text{sem},u}$ is stored on island L. The network is notified that a user's key pair is to be revoked, and a P2P request is generated to L to revoke the user's key pair. If L is operational, then L notifies all of the other islands that hold shares of $d_{\text{sem},u}$ to delete them and store the fact that $d_{\text{sem},u}$ has been revoked; L also deletes $d_{\text{sem},u}$ and adds the user's serial number to its CRL. Else if L is not operational, then the k islands holding the shares of $d_{\text{sem},u}$ are notified and told to delete their shares. (Note that in this case, migration—see below—has not yet occurred for this user or else an island would have been contacted.)

4.3 SEM migration

If a user u issues a request but the island L holding $d_{\text{sem},u}$ is not available, then the user contacts the SEM network and another island M is selected for migration. As described before, M can be chosen by the network in a pseudorandom or load-balancing way. The best method may differ based on the size of the network and its topology. It is important in some scenarios that the attacker of L and the user cannot predict or choose M. However, even if they can and M turns out to be compromised, migration can always be performed from M to another node.

After that initial step is performed, we have two different approaches, depending on whether a CA exists that can know the full private key d_u. Any communication between the islands is authenticated using the outbound authentication of the secure coprocessors and it is assumed that the online CA also has some mode of outbound authentication to prove the source of its messages.

For added resilience, we can have shareholder islands not participate in migration if they can still ping the original island L.

Using a CA

If we have an online CA, the new island M contacts it and tells it that migration is to occur. (If we constrain the choice of the next M, then the CA must verify that M is a satisfactory candidate.) M sends a request for at least t of the islands in the network that have shares of $d_{\text{sem},u}$ to send those securely to the CA using standard RSA encryption. The CA can then reconstruct $d_{\text{sem},u}$. If the CA stores the full private key for d_u, then it can use that, or obtain it from the user. Otherwise, the shareholders for d_u must also send those, so the CA can reconstruct it as well. The CA generates a random number x in the interval $[0, n_u - 1]$ such that $x \neq d_{\text{sem},u}$, and calculates $y = d_u - x \pmod{\phi(n_u)}$. The CA securely distributes shares of x to the k shareholder islands using threshold cryptography; the shareholder islands delete the old shares for $d_{\text{sem},u}$. The CA securely sends y to the user by encrypting it with the user's public key and then partially decrypting with the old $d_{\text{sem},u}$. The user deletes the old $d_{\text{user},u}$ and sets $d_{\text{user},u} = y$. The CA securely sends the x to M, who deletes the old $d_{\text{sem},u}$ and sets $d_{\text{sem},u} = x$ (see Fig. 3.)

Once the user receives y, it reconnects to the network and performs the operation again, using its new value $d_{\text{user},u}$. At this point, M can complete the request and the migration is complete.

No CA

If no CA exists, then we need to generate a new $d_{\text{sem},u}$, $d_{\text{user},u}$ pair without reconstructing d_u or $\phi(n_u)$, since we do not have a safe place to store them.

Instead, we use a different technique (adapted from idea by the second author with student Z. Le). We have M generate a δ in a range $[-r, r]$, and changing $d_{\text{sem},u}$ to $d_{\text{sem},u} - \delta$, where r is big enough to keep the key halves changing unpredictably, but small enough to be smaller than $d_{\text{sem},u}$ and $d_{\text{user},u}$ for a practically indefinite number of rounds. M sends δ to the user (encrypted with u's public key and then partially decrypted with the old $d_{\text{sem},u}$); the user replaces $d_{\text{user},u}$ with

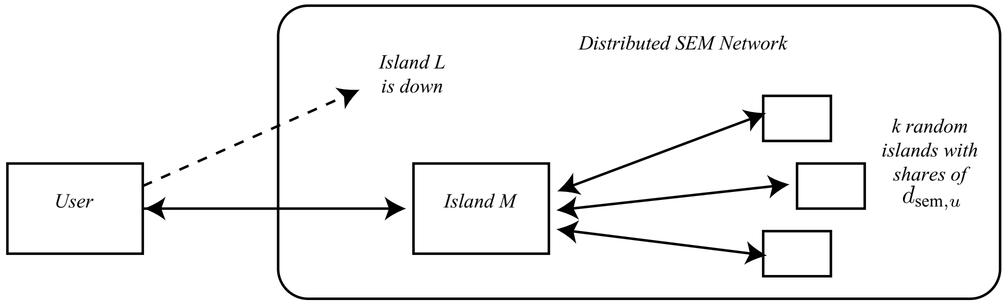

Fig. 3 This diagram shows migration, if the CA is available. If a user u's current mediator L is down, then a new mediator M is selected pseudorandomly. M collects the shares for the old $d_{\text{sem},u}$, the CA reconstructs $d_{\text{user},u}$ and generates new halves for the user and M

$d_{\text{user},u} + \delta$. (We could also have M and u together pick r, to reduce risk from a compromised M.)

This way, neither M nor u need to know $\phi(n_u)$, but the new $d_{\text{user},u}$ and $d_{\text{sem},u}$ remain positive and still sum to d_u. M splits r into k shares and sends each to a $d_{\text{sem},u}$ shareholder; each shareholder uses its piece to update its share (see Fig. 4).

It is tempting to have M pick a new $d_{\text{sem},u}$ directly and distribute shares to the shareholders of d_u, who then calculate the new $d_{\text{user},u}$ in a nicely distributed fashion. However, as of this writing, we can cannot see how to reduce $d_u - d_{\text{sem},u}$ to $d_u - d_{\text{sem},u} \pmod{\phi(n_u)}$ without reconstructing $\phi(n_u)$.

Once the user receives the new $d_{\text{user},u}$, it can compute its half of the normal computation using the new $d_{\text{user},u}$. At this point, M can also complete its half of the computation because it has generated a new $d_{\text{sem},u}$ and the migration is complete.

Renewing user key pairs

As an area of future work, we are also considering incorporating strong forward secrecy into regeneration of the user's private key during regeneration of $d_{\text{sem},u}$. We already have trusted hardware, which is one of the components of some SFS schemes in the literature (e.g., [9, 43]). Furthermore, this would protect against compromise of L by the user u, in order to obtain $d_{\text{sem},u}$ and reconstructing d_u.

Recovery

When an island goes down (or is compromised and subsequently shut down and restarted from a clean state), it has a few options upon reboot.

The island could delete all of the key halves it has stored, and thereby force users to migrate back to it. New users would also be assigned to it. It also deletes all of the shares of x that it stored and requests new shares of those to be generated.

Alternatively, the island could poll the other islands to determine which $d_{\text{sem},u}$ halves have migrated away from it. It then deletes the information for the users that migrated away and continues serving the other users. The island can continue using the shares it has stored, but it must determine whether any of them are out of date. Since the $d_{\text{sem},u}$ shares must be updated during migration, the $d_{\text{sem},u}$ shares could be invalid and the network must be polled to determine whether this is the case. If so, then the outdated shares must be updated.

4.4 Analysis

The migration process is aimed at improving the security of any application that uses it, so we now discuss its strengths and weaknesses in various situations.

Machines

First, we consider compromise of specific entities.

If the CA has been compromised, then we have a serious problem, since the CA generates the users' initial key pairs, and in the CA-migration case, learns the new key pairs as well.

Alternatively, suppose L has been compromised. The island L holds $d_{\text{sem},u}$ for some number of users. Additionally, L stores shares for some other users in the distributed SEM network. We must assume that the attacker has access to all of these values, so now we analyze what privileges are granted by illicit access to them.

– If the attacker acquires $d_{\text{sem},u}$ for another user u, then migration effectively disables this $d_{\text{sem},u}$ because it causes a new $d_{\text{user},u}$ to be issued to the user. (Also, note that the new $d_{\text{user},u}$ is not sent back to L.) Since the new $d_{\text{user},u}$ does not mesh with the old $d_{\text{sem},u}$ due to the mRSA protocols, the old $d_{\text{sem},u}$ is rendered useless.
– If the attacker acquires $d_{\text{sem},u}$ and colludes with that user, then the attacker will be able to compute d_u from $d_{\text{user},u}$ and $d_{\text{sem},u}$, so migration fails to achieve full security in this case (unless, as discussed earlier, we try implementing SFS here as well).
– If the attacker colludes with a user whose key-half is not on that island, then the user and attacker might trick the SEM network to migrating that user's data to L, and thus reconstruct $d_{\text{sem},u}$. The user will then be able to reconstruct d_u, the full private key. This problem can be mitigated by using pings (as stated in Sect. 4.3) to ensure

that the user's main island is unavailable, and also using a non-predictable way to generate the next island for migration (to reduce the chance that M is a valid candidate). However, such problems may be the inevitable cost of higher availability in the distributed SEM network.

– If the attacker acquires shares of a $d_{\text{sem},u}$, the attacker effectively acquires no valuable information, unless the attacker also gains access to enough other shares of either in order to reconstruct them. However, this can be made extremely difficult using proactive share updating.

Clearly, there is a period between the time of compromise time when that compromise is discovered. However, the use of trusted computing platforms will, with some assurance, cause physical attacks to be detected and mitigated by zeroizing sensitive data (or otherwise rendering it inaccessible).

If M is compromised, then the attacker gets access to the new $d_{\text{sem},u}$, so the migration is unsuccessful. However, as long as another migration to an uncompromised island can be performed, the $d_{\text{sem},u}$ acquired by the attacker can be rendered useless as described above. Additionally, in this case the attacker could send the user fake data for the new $d_{\text{user},u}$, but any resulting inconsistencies with decryption or signature generation would just flag M as compromised. As noted above, the stronger the trusted computing platform, the smaller the window of time before this compromise is discovered.

Communications

Since migration involves communication and data transfer between islands in the network, we will now analyze each interaction and discuss its security as it relates to requests by a server that has been compromised but not discovered yet.

If an island N is compromised, then the attacker might attempt to request the shares of $d_{\text{sem},u}$ for any users. If the compromised island succeeds in collecting at least t shares of any $d_{\text{sem},u}$, then the attacker can reconstruct $d_{\text{sem},u}$. However, in order to receive shares, each island that holds a share of must not be able to successfully ping the island that holds the full $d_{\text{sem},u}$—and N must be a valid next candidate. Therefore, in order to receive shares of $d_{\text{sem},u}$, the island that holds the full $d_{\text{sem},u}$ must be unavailable, making the full $d_{\text{sem},u}$ assumed to have been compromised. Therefore, an imminent migration will invalidate the $d_{\text{sem},u}$ and the attacker will gain no useful information.

4.5 Network trust model

The primary parties that require use of the network are the islands that comprise the network itself. Each island must have exclusive access to certain services in order to provide fast revocation of security capabilities. However, the CA (and users and islands) must also be able to gain access to the network services in a restricted way. Clearly, the requests of each party will differ and there must be a clear delineation of capabilities between them. For example, during migration, islands will have to search the network to find other islands that hold the shares of a user's $d_{\text{sem},u}$. Although this operation can be executed by the CA as well (when CAs can perform migration), users should not be able to (easily) determine the location of or acquire shares.

Islands join the network normally and become full members of it. Since each island in the network has a trusted computing platform with outbound authentication or attestation, each member in the peer-to-peer network can prove that it is a trusted island with certain privileges. This creates a trust network in which each island is known to be executing unmolested as long as our software is not flawed (and the platform's security protection works). Once an island has authenticated itself to the system, it can search the network, advertise services, and perform any other command allowed by the peer-to-peer software.

Certificate Authorities can interact with the network in one of two ways: (1) they can connect to an island server that provides an interface to the rest of the network; or (2) they can connect directly to the P2P network, but with limited capabilities (registration and, if implemented using a CA, then migration). For example, the CA must be able to somehow query the network during key generation to determine to which island to assign the user.

Users do not connect directly to the P2P network, but instead communicate with an island that provides indirect access to the services available on the network. For example, during normal operation, users connect directly to their assigned island, but if that fails, then the user must notify the P2P network that migration is necessary. Users do so by connecting to another island (available in a public list) and requesting the migration service. The user is then assigned to an island and further communication occurs directly between that island and the user.

4.6 Legal issues

With the proliferation of PKI mechanisms comes a variety of legal issues associated with the services provided by PKIs, including liability for timely certificate status information. In the common case, our Distributed SEM approach provides instant verification of the validity of a digital signature since the signature could not have been generated if the user's privileges had been revoked. So, liability for timely certificate status information is not usually an issue in our architecture. For something to go wrong an attacker or an administrator must be involved. If an attacker is at fault, then legal action can potentially be brought against the attacker. If a negligent (or malicious) administrator causes a problem on an island, then there must be a legal entity that takes responsibility for the incident. For each Distributed SEM network there must exist a central legal entity covering all nodes that join and that central entity will handle legal issues for the entire network. This will provide incentive for new

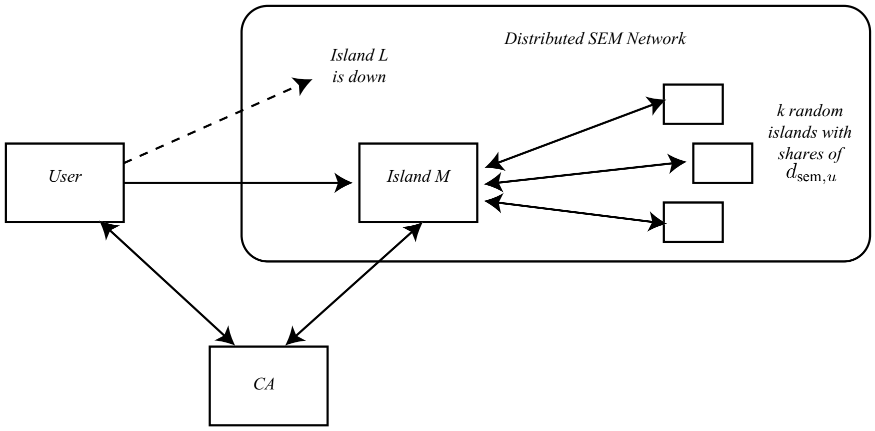

Fig. 4 This diagram shows migration, if no CA is available. If a user u's current mediator L is down, then a new mediator M is selected pseudorandomly. M collects the shares for the old $d_{\mathrm{sem},u}$, and generates a δ with which the user u and mediator M adjust their key halves

nodes to join the network because they are not individually liable.

However, definitive consideration of such law and public policy questions lies outside the scope of this paper.

5 Prototype

5.1 Initial prototype

Initially, we prototyped our distributed SEM approach as a simulation. The current code (2000 lines of Java) deals with the peer-to-peer aspects of key generation and execution of migration (see Fig. 5.)

The *Island Server Code* performs migration and the island's part of decryption and key and signature generation using $d_{\mathrm{sem},u}$. This part also introduces the networking support for the islands. It is a combination of both the original SEM server code, along with our server-related migration code.

The code on each island is divided into two parts. The *P2P network code* consists of the peer-to-peer access layer and the protocols necessary for island communication. We use the Project JXTA open source framework [38] to accomplish this. Each island runs a server that accepts connections from only the SEM server on the same machine. The server application is built on top of JXTA and uses the peer-to-peer protocols, such as searching and pinging, available. It executes the distributed aspects of key generation (distribution of shares of $d_{\mathrm{sem},u}$) and migration.

Specifically, during the distributed SEM operations we use JXTA's discovery service to find islands with the information needed (e.g. shares of $d_{\mathrm{sem},u}$). We plan to leverage the security features of the JXTA framework to secure the P2P activities of the distributed SEM network. These include secure P2P groups to restrict access to the network and secure pipes to allow safe distribution of shares to an island during migration. Furthermore, we envisioned that an underlying trusted computing platform would provide outbound authentication/attestation, that we could use in JXTA XML messages to validate the source of messages generated in the migration and revocation algorithms. We can expose these capabilities in the Java code using the Java Native Interface (JNI) [36].

The *SEM server code* accepts requests from users and handles most of those without using the P2P layer. When a migration or key registration request is received, however, the server code forwards the request to the internal server running JXTA and the internal server completes the request.

This is a modular approach that allows us to change the peer-to-peer implementation in the future. In the current prototype, we have implemented our network code in a simplified version. (Integration with the SEM server code, utilization of security features in the P2P layer, and porting on to the 4758 remain to be done.)

The *Certificate Authority Code* participates in key generation, as in the original SEM architecture. With our additions it is necessary for the CA to connect to an island and initiate a P2P registration request. We have implemented the code to perform registration in the network. (However, it is not yet integrated with the original SEM key generation code.)

The *Client User Code* (still under construction) combines the functionality of the original SEM architecture, which consisted of the user's half of signature generation and decryption, along with with the additional steps required to request migration and process the migration response.

Boneh et al. [3, 4] describe the use of an email client plug-in and an email proxy to easily integrate the client user code on a client's machine. Both provide a high degree of transparency to the user, but the email proxy is favorable because it can be used with all email clients. After installation and configuration, the email proxy can transparently sign and decrypt using Distributed SEM. We envision our additions, such as migration, as transparent operations to the user. This transparency allows less security conscious users to benefit from our Distributed SEM architecture, an important feature in any approach to PKI.

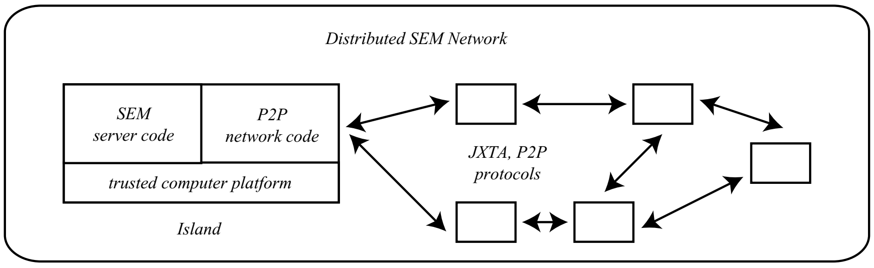

Fig. 5 In our basic software architecture, the distributed SEM network contains a set of islands. Each island consists of a trusted computing platform running the SEM server code and the P2P network code. The islands interact using P2P and the JXTA framework

5.2 Trusted computing platforms

The next step in building a prototype was to choose a trusted computing platform for the code. We saw several primary options.

The IBM 4758 secure coprocessor [35] offered a weak computational environment (486-class CPU) guarded by high assurance protections (the world's first FIPS 140-1 Level 4 validation). Interlocking software and hardware controls assure that only authorized code is loaded—and has access to a statistically unique key pair with a certificate chain identifying that code in that configuration on that device. With high assurance, tamper causes these secrets to be destroyed. However, this assurance comes at a price: the 4758 has retailed for $3K in quantities of one. (A follow-on device with more power—the IBM 4764—is on the market, at a significantly higher price. As of this writing, no software development kit is available.)

As discussed earlier, the TCG (successor to the *Trusted Computing Platform Alliance, TCPA*) designed a TPM: a small chip, added to a commodity motherboard, that witnesses platform configuration via a set of *platform configuration registers (PCRs)* and releases or uses credentials (such as RSA private keys) only when the PCRs have appropriate values [39–41]. The TPM can thus enable *sealed storage*—data available only to trusted software in a trusted configuration on this platform—as well as attestation, obtained initially through use of a TPM-held private key quoting the PCRs.

Compared to the 4758, the TPM approach offers substantially weaker physical security—a TPM platform can be attacked with Xacto knives and logic analyzers, and is also susceptible to time-of-check/time-of-use issues. On the other hand, the TPM approach is substantially cheaper and more ubiquitous—the 1.1b TPM has been shipping for years with IBM Netvistas and ThinkPads.

We stress, however, that the TPM is not itself a secure computing platform, but merely an add-on chip that provides weakly-secure storage of credentials, keyed to platform hashes.

Another approach to is to add physical security enhancements to the CPU itself, and additional modes of operation to essentially turn a portion of the CPU itself into its own secure coprocessor.

In the academic space, the *AEGIS* project from MIT uses *silicon physical unknown functions* to hide secrets from physical adversaries, and offers hardware support so trusted data can be confined to trusted code [37]. Prototypes built on the OpenRISC core will soon be available.

In the commercial space, Intel's *LaGrande* and ARM's *Trustzone* both offer similar designs, but with fewer claims about physical security (and no firm commitment about availability to researchers).

In our work, we initially leaned toward programmable secure coprocessors such as the IBM 4758, because they provide a separate (and much stronger) security domain from their host. However, while such an environment is good from a security viewpoint, a number of factors which make it suboptimal from a practical standpoint. First, there is often little codespace for applications in such devices. Our lab has experimented writing applications with the IBM 4758 [16, 17, 19, 29, 32], and have repeatedly been hindered by the device's internal codespace limit. Second, the high cost of such devices hinders client-side deployment.

Recognizing these shortcomings of highly secure devices, our lab set out to make a low-cost, less-secure, open-source alternative based on the TCPA/TCG hardware. Details of our platform (called *Bear/Enforcer*)can be found in previous work [21, 22], but the basic idea is to utilize the TCPA/TCG's TPM to ensure the integrity of the core kernel and the Enforcer, a Tripwire-like Linux Security Module that measures the integrity of the rest of the system against a signed configuration file, and also protects a key which is used for an encrypted loopback filesystem. When the system is booted, the Enforcer uses the TPM to check if critical system components have changed (e.g., the hardware, bootloader, kernel, etc.). If not, then the Enforcer retrieves its key from the TPM and mounts the encrypted loopback filesystem. As files are opened, the Enforcer performs a run-time integrity check on the file against its security policy. If a mismatch occurs—indicating that someone has tampered with a file—the Enforcer will unmount the loopback filesystem and panic the kernel.

To summarize, in distributing SEM, we form a network of trusted islands. Each island is a computing platform protected by trusted hardware. For our prototype, we used, for each platform, a standard destkop armed with a 1.1b TPM and our Bear/Enforcer code. The Bear/Enforcer code

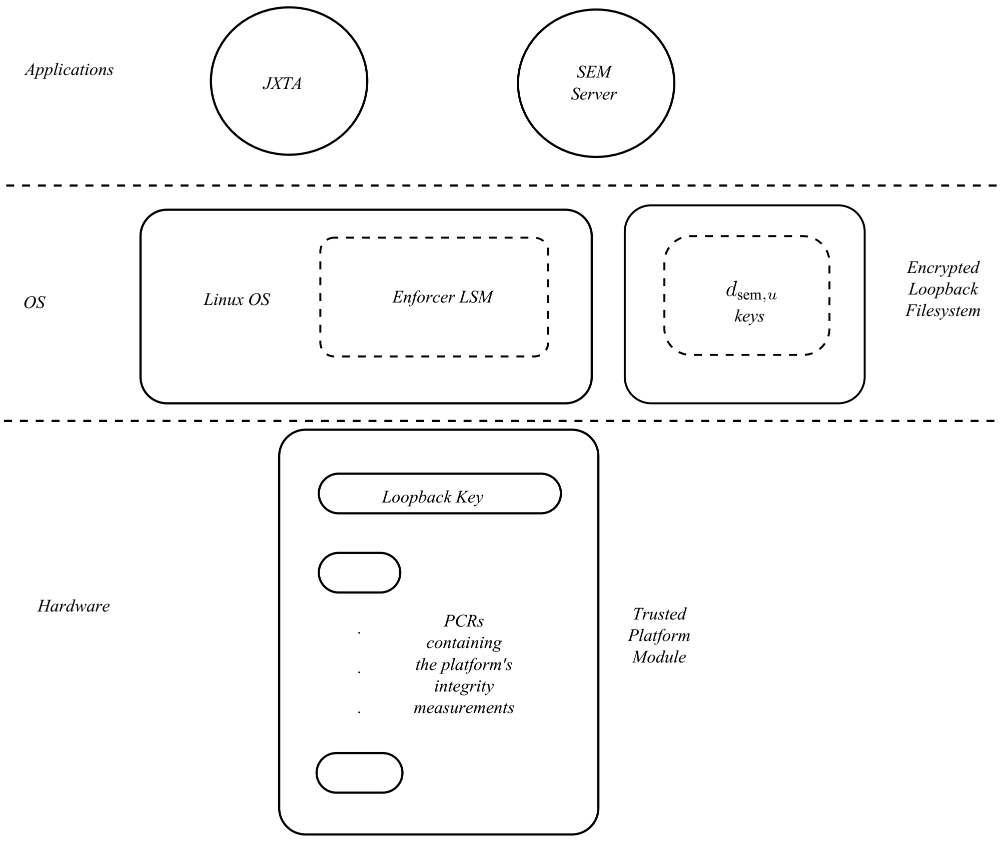
Applications
JXTA
SEM Server
OS
Linux OS
Enforcer LSM
$d_{\mathrm{sem},u}$ keys
Encrypted Loopback Filesystem
Hardware
Loopback Key
PCRs containing the platform's integrity measurements
Trusted Platform Module

Fig. 6 An architectural view of Distributed SEM prototype running on our Bear/Enforcer platform. During the system's boot sequence, the Enforcer LSM will attempt to retrieve the key for the encrypted loopback filesystem. If the platform's integrity measurement allows, the TPM will release the key and the Enforcer will then mount the loopback filesystem. If the Enforcer ever detects a violation of its security policy (i.e., via run-time integrity checking), it will unmount the loopback filesystem, disallowing the Distributed SEM prototype to access its key (shares)

integrates the 1.1b TPM into the broader system architecture in a way that provides a secure execution environment that is more powerful than something like the 4758, but much less secure: in particular, physical attacks, such as using a logic analyzer, will defeat the system. In the future, we plan to explore enhanced-CPU approaches, such as AEGIS, which may provide a better tradeoff of power versus security versus cost. Our monograph [34] provides more details about this emerging space.

5.3 Porting to trusted hardware

Once our Distributed SEM prototype reached a stable state, we placed in on a Bear/Enforcer platform in order to experiment with Distributed SEM and secure hardware. We added to the Enforcer's security policy so that it ensures no Distributed SEM binaries (i.e., SEM server and JXTA `.class` files) are modified. We then placed the key material in the encrypted loopback filesystem. With this configuration, Enforcer ensures that the Distributed SEM code (along with other critical system components and software) adheres to the Enforcer's integrity policy. Should the Enforcer's policy be violated (e.g., by someone tampering with a file in the policy), the the Enforcer will make the Distributed SEM key material unavailable, thus disallowing an attacker to use the portion of Alice's private key held by the SEM (see Fig. 6.)

5.4 Performance

Once we ported our initial prototype to our Bear/Enforcer trusted computing platform, we measured the performance of the migration operation in order to quantify the overhead of running the Distributed SEM code in secure hardware.

Our setup consisted of three nodes: one primary island L, and two islands which received shares of $d_{\mathrm{sem},u}$. We placed L on a Bear/Enforcer platform. The server was an IBM Netvista 8310 server (which includes the TPM hardware) running the Debian/unstable Linux distribution with our Enforcer LSM compiled into a 2.6.5 kernel.

As a baseline, we ran the migration operation on an instance of L *not* protected by Bear/Enforcer. We executed the operation a total of five times from L, and averaged the running times to get a result of 3.656 seconds. We then placed L on a Bear/Enforcer platform, and executed the migration operation five times, yielding an average running time of

Table 1 Performance results for the migration operation

Run	Without Secure HW	With Bear/Enforcer
1	4.396	4.234
2	3.284	3.219
3	3.803	3.580
4	3.648	3.353
5	3.150	5.402
Average	3.656	3.957

3.957 seconds. The normalized slowdown is given by:

$$\frac{3.957 - 3.656}{3.656} = .082$$

Thus, using the Bear/Enforcer platform for security yields an 8.2% slowdown. The results are summarized in Table 1.

5.5 Download

Our code will be available for public download. Since the original SEM code is covered under the GPL, our changes—for migration and support of distributed functionality—are as well. (The Bear/Enforcer source code is already available on Sourceforge.)

6 Related work

6.1 Trusted hardware and P2P

Marchesini and Smith [19] built a *hardened Gnutella* P2P communication system within secure coprocessors; Boneh et al. [12] also considered the potential of combining trusted computing with P2P.

6.2 Strong forward secrecy

Tzeng and Tzeng [43] considered strong forward security with threshold cryptography for El Gamal signatures.

6.3 Standard revocation techniques

Certificate Revocation Lists and variations on this method (e.g., Δ-CRLs) are one of the most common methods of certificate revocation. To revoke a certificate, the CA adds the serial number of the revoked certificate to the CRL and then publishes that CRL to a directory. Since this is only done periodically, CRLs are not a guarantee that a certificate is still valid. Also, since CRLs may be very large, users will generally not want to have to download them very often. In order to check whether a certificate is revoked, a user must potentially download a long CRL. To mitigate this problem, Δ-CRLs only distribute a list of the certificates revoked since the last CRL was distributed. Additionally, Cooper notes that when a new CRL is issued there will be a peak time in which many requests are made to download the CRL from the directory (because everyone wants to make sure that certificates aren't revoked and a CRL expires at the same time for everyone). He suggests spreading out the requests for CRLs over time by "over-issuing" CRLs such that a new CRL is published before the old one expires [7].

Another similar technique is *Windowed Key Revocation*, which uses CRLs but with a twist that certificates are assumed to be valid for a certain "window" of time and that CRLs have a reduced size due to the revocation window [23]. Additionally, in this scheme verifiers can control the allowed "window" time and to check if a certificate is revoked, the verifier checks the windowed CRL issued or grabs a new certificate from the CA.

The *Online Certificate Status Protocol (OCSP)* provides online verification that a certificate is still valid. This requires a CA to generate a digital signature for each request because the response from the CA must be signed [24]. The CA stores an internal log of the status of all certificates or possibly just a CRL that it doesn't publish. So, addition of revoked certificates is quick and the certificate status is updated instantly. A user must be online and must connect to the CA and check the status of a certificate.

Certificate status verification is a computationally expensive operation, as the response from the CA must be digitally signed. If a single validation server performs OCSP, then all requests must be routed to it, potentially overloading the server. Security may be weakened by a distributed environment because if any keys of any OCSP servers are compromised, then the entire system is compromised.

With *Certificate Revocation Trees*, instead of keeping an entire list of revoked serial numbers, we keep a list of ranges of serial numbers that are good or bad. This saves space (better than standard CRLs), but adding a serial can involve a good amount of computation as it can require the entire tree to be recomputed. Still, revocation status can be quickly determined by a fast search through the tree.

6.4 Newer revocation techniques

In [4], Boneh et al. give a discussion of the benefits of the original SEM architecture with regards to other current solutions. Since the publication of that paper, a few other techniques for certificate revocation have been developed. Micali's *NOVOMODO* approach [25] uses one-way hashing and hash chains to show the validity or revocation status of certificates. Centralized NOVOMODO—in which the central secure server responds to all validity requests—is prone to performance issues and denial of service attacks as it is the central source for certificate validity proofs. Micali also presented a distributed version: having one central trusted server send out an array of the current validity proofs for all users to each server in the network. Micali does not discuss solutions to many potential problems that could occur during a distribution, such as bandwidth problems, network partition or untrusted server corruption. Micali states that it should be very difficult to attack the central server, as it does not accept incoming requests, but an attacker could instead attack the network surrounding the server, preventing it from distributing the array. In other words, distributed NOVOMODO still has a central "head" that can be severed (albeit in a more difficult way) in order to shutdown the system.

Ding et al. introduce *Server-Aided Signatures (SAS)*, [8] a technique based on mediated cryptography similar to the SEM architecture with the focus on minimizing client computation load. While Distributed SEM works with both signature generation and decryption, SAS only deals with signature generation. It achieves a performance boost for the user by only requiring the user to compute a hash chain during setup, in a similar fashion to NOVOMODO, but differing in that the user keeps the chain secret. In SAS, the server must be stateful and, for each user, must save their certificate, i, and all of the signatures already generated for that user. This amount of state makes migration infeasible as every signature would have to be distributed on other islands using threshold cryptography. Additionally, in SAS the corruption of a server allows the attacker to produce user signatures because all of the prior signatures are saved on the server.

6.5 SEM

Tsudik [42] and Boneh et al. [3] have also followed up on their original SEM work. Since the multiple-halfpair approach of [3] is essentially orthogonal to our distributed approach, it would be interesting to try them together.

6.6 Trusted computing and PKI

The SEM approach to PKI requires trusting a third party, the mediator. Our distributed SEM approach requires trusting even more of them, and uses trusted computing platforms to help.

In addition to that mentioned above, other work also explores these directions. Perrin et al. propose housing a user's private key on an *online trusted third party* and doing cryptography there [30]. An IETF working group is examining *Securely Available Credentials (SACRED)*, ways for a user to migrate private keys from one platform (which might be a central repository) to another. The Grid community has produced a *MyProxy* key repository; a user's real private key lives there, but certifies a temporary key used on a less secure machine [26]. Follow-on work moved the cryptographic operations onto an IBM 4758 [18].

In our own lab, we are working on *Secure Hardware Enhanced MyProxy (SHEMP)* [20]. Our Bear/Enforcer trusted computing work provides a way to increase assurance of ordinary desktops; the Grid community's *MyProxy* provides a foundation for authentication using temporary key pairs, and OASIS' *XACML* provides a standard way of expressing policies. SHEMP combines these tools to produce a way for users to employ proxy key pairs for signatures and encryption as well, limited by predefined policies geared toward the trustworthiness of the client platform

6.7 Evaluation of certificate status information mechanisms

With the recent advancement and proliferation of PKI techniques, the need for a framework to evaluate the many certificate revocation mechanisms has arisen. Consequently, researchers have developed frameworks, such as [15], to determine the strengths and weaknesses of each. In their paper, Iliadis et al. compare current certificate revocation techniques according to the guidelines of their framework and suggest areas for improvement. The framework described is split into three major categories of criteria that together allow for evaluation of an entire mechanism: management, performance and security. As an area of future work, we could provide further comparison of our Distributed SEM approach with other certificate revocation mechanisms by fitting our solution into the framework and comparing its fulfillment of criteria with the evaluation of other techniques (e.g. CRLs and OCSP) provided in [15]. This would allow an unbiased comparison of our technique with others and provide insight into both weaknesses that need to be addressed in our approach and advantages that our solution uniquely possesses. Recent literature also offers other potential frameworks [6, 27, 28].

7 Conclusions and future work

In this paper we have introduced a method to distribute SEM by using a network that combines the benefits of trusted computing platforms and peer-to-peer networking, and provides efficient and uninterrupted access to private data stored on a trusted third party, even in the event of occasional server compromise. This approach avoids replication of data across the network while also avoiding the common use of distributed computation in order to access the secrets stored.

One of our next areas of interest here will be further performance testing and tuning. The performance of both migration itself and the entire application running on the full P2P network will be reveal much information about our approach to distributed SEM—and the feasibility of this P2P and trusted hardware network. We also hope to extend our basic network to include heterogeneous types of trusted hardware (e.g., 4758s, Bear/Enforcer, and maybe even AEGIS platforms) and build an underlying PKI for these nodes themselves that lets them distinguish and tune for the security levels provided. We also plan to use our framework of P2P on trusted hardware to explore other applications as well. General Byzantine attacks must be considered in the Distributed SEM network and extra steps (e.g., [1]) must be taken to ensure the correct completion of all operations. As our implementation progresses, we anticipate further technical hurdles in combining disparate features of our Distribute SEM network. This could include runtime interactions between proactive share updating and migration and perhaps other unanticipated technical difficulties.

However, at the end of the day, PKI exists to embody human and organizational trust relations in electronic settings. An important part of future work for distributed SEM would be trying the technique in increasingly large pilots with real users, in order to learn where scalability and other problems show up in practice.

PKI is essentially a technology to express trust across organizational boundaries. Trusted computing platforms technology helps enhance trustworthiness of computation across organization boundaries. Looking at the two together has a natural synergy, and we offer our research as a step in that direction.

Acknowledgements A preliminary version of this paper appeared at *EuroPKI* [44]. The authors are grateful to our colleagues and the referees for their helpful comments. This work was supported in part by the Mellon Foundation, by the NSF (CCR-0209144, EIA-9802068), by Internet2/AT&T, and by the Office for Domestic Preparedness, U.S. Department of Homeland Security (2000-DT-CX-K001).

References

1. Alon, N., Kaplan, H., Krivelevich, M., Malkhi, D., Stern, J.: Scalable secure storage when half the system is faulty. Information and Computation **174**(2), 203–213 (2002)
2. Andrews, G.: Paradigms for process interaction in distributed programs. ACM Computing Surveys **23**(1), 49–90 (1991)
3. Boneh, D., Ding, X., Tsudik, G.: Fine-grained control of security capabilities. ACM Transactions on Internet Technology **4**(1), 60–82 (2004)
4. Boneh, D., Ding, X., Tsudik, G., Wong, C.M.: A method for fast revocation of public key certificates and security capabilities. In: 10th USENIX Security Symposium, pp. 297–308 (2001)
5. Burmester, M., Chrissikopoulos, V., Kotzanikolaou, P., Magkos, E.: Strong forward security. In: IFIP-SEC '01 Conference, pp. 109–121. Kluwer (2001)
6. Cooper, D.: A model of certificate revocation. In: 15th Annual Computer Security Applications Conference (ACSAC'99), pp. 256–264. IEEE Computer Society (1999)
7. Cooper, D.A.: A model of certificate revocation. In: Fifteenth Annual Computer Security Applications Conference, pp. 256–264. IEEE Computer Society (1999)
8. Ding, X., Mazzocchi, D., Tsudik, G.: Experimenting with server-aided signatures. In: Network and Distributed Systems Security Symposium (2002)
9. Dodis, Y., Katz, J., Xu, S., Yung, M.: Strong key-insulated public-key schemes. In: Public Key Cryptography—PKC 2003, pp. 109–121. Springer-Verlag LNCS 2567 (2003)
10. Frankel, Y., Gemmell, P., MacKenzie, P.D., Yung, M.: Optimal resilience proactive public-key cryptosystems. In: IEEE Symposium on Foundations of Computer Science, pp. 384–393 (1997). URL `citeseer.nj.nec.com/61609.html`
11. Frankel, Y., Gemmell, P., MacKenzie, P.D., Yung, M.: Proactive RSA. In: Advances in Cryptology—CRYPTO 97, pp. 440–454. Springer Verlag LNCS 1294 (1997). URL `citeseer.nj.nec.com/frankel97proactive.html`
12. Garfinkel, T., Rosenblum, M., Boneh, D.: Flexible OS support and applications for trusted computing. In: 9th Hot Topics in Operating Systems (HOTOS-IX) (2003)
13. Herzberg, A., Jakobsson, M., Jarecki, S., Krawczyk, H., Yung, M.: Proactive public key and signature systems. In: ACM Conference on Computer and Communications Security, pp. 100–110 (1997)
14. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: how to cope with perpetual leakage. In: Advanced in Cryptology—CRYPTO 95, pp. 339–352. Springer Verlag LNCS 963 (1995). URL `citeseer.nj.nec.com/herzberg95proactive.html`
15. Iliadis, J., Gritzalis, S., Spinellis, D., de Cock, D., Preneel, B., Gritzalis, D.: Towards a framework for evaluating certificate status information mechanisms. Computer Communications **26**(16), 1839–1850 (2003)
16. Iliev, A., Smith, S.: Privacy-enhanced credential services. In: 2nd Annual PKI Research Workshop. NIST (2003)
17. Jiang, S., Smith, S., Minami, K.: Securing web servers against insider attack. In: Seventeenth Annual Computer Security Applications Conference, pp. 265–276. IEEE Computer Society (2001)
18. Lorch, M., Basney, J., Kafura, D.: A hardware-secured credential repository for grid PKIs. In: 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (2004)
19. Marchesini, J., Smith, S.: Virtual Hierarchies: An architecture for building and maintaining efficient and resilient trust chains. In: Proceedings of the 7th Nordic Workshop on Secure IT Systems—NORDSEC 2002. Karlstad University Studies (2002)
20. Marchesini, J., Smith, S.: Secure Hardware Enhanced MyProxy. Tech. Rep. TR2005-532, Dartmouth College (2005)
21. Marchesini, J., Smith, S., Wild, O., MacDonald, R.: Experimenting with TCPA/TCG Hardware, Or: How I Learned to Stop Worrying and Love The Bear. Tech. Rep. TR2003-476, Department of Computer Science, Dartmouth College (2003)
22. Marchesini, J., Smith, S., Wild, O., Stabiner, J., Barsamian, A.: Open-source applications of tcpa hardware. In: 20th Annual ACSAC Conference (2004)
23. McDaniel, P., Jamin, S.: Windowed certificate revocation. In: IEEE Symposium on Security and Privacy, pp. 1406–1414 (2000)
24. McDaniel, P., Rubin, A.: A response to "can we eliminate certificate revocation lists?" In: Financial Cryptography (2000)
25. Micali, S.: NOVOMODO: Scalable certificate validation and simplified pki management. In: 1st Annual PKI ResearchWorkshop. NIST (2002)
26. Novotny, J., Tueke, S., Welch, V.: An online credential repository for the grid: MyProxy. In: Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10). IEEE Press (2001)
27. noz, J.M., Forné, J.: Evaluation of certificate revocation policies: OCSP vs. overissued CRL. In: DEXAWorshops 2002: Workshop on Trust and Privacy in Digital Business (TrustBus02), pp. 511–515. IEEE Computer Society (2002)
28. noz, J.M., Forné, J., Esparza, O., Soriano, M.: A test-bed for certificate revocation policies. In: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (2003)
29. Periera, M.: Trusted S/MIME Gateways. Senior Honors Thesis. Also available as Computer Science Technical Report TR2003-461, Dartmouth College (2003)
30. Perrin, T., Bruns, L., Moreh, J., Olkin, T.: Delegated cryptography, online trusted third parties, and PKI. In: 1st Annual PKI Research Workshop. NIST (2002)
31. Shamir, A.: How to share a secret. Communications of the ACM **22**, 612–613 (1979)
32. Smith, S.: WebALPS: A survey of E-commerce privacy and security applications. ACM SIGecom Exchanges **2.3** (2001)
33. Smith, S.: Outbound authentication for programmable secure coprocessors. Int. J. Inf. Secur. **3**(1), 28–41 (2004)
34. Smith, S.: Trusted Computing Platforms: Design and Applications. Springer (2005)
35. Smith, S., Weingart, S.: Building a high-performance, programmable secure coprocessor. Computer Networks **31**, 831–860 (1999)

36. Stearns, B.: Trail: Java Native Interface. Sun Microsystems, Inc. (2004). URL http://java.sun.com/docs/books/tutorial/native1.1/
37. Suh, G., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: AEGIS: Architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the 17 International Conference on Supercomputing, pp. 160–171 (2003)
38. Sun Microsystems, Inc.: Project JXTA: Java Programmers Guide (2001). URL http://www.jxta.org
39. Trusted Computing Platform Alliance: TCPA Design Philosophies and Concepts, Version 1.0 (2001). URL http://www.trustedcomputinggroup.org
40. Trusted Computing Platform Alliance: TCPA PC Specific Implementation Specification, Version 1.00 (2001). URL http://www. trustedcomputinggroup.org
41. Trusted Computing Platform Alliance: Main Specification, Version 1.1b (2002). URL http://www.trusted-computinggroup.org
42. Tsudik, G.: Weak forward security in mediated RSA. In: Security in Computer Networks Conference (2002)
43. Tzeng, Z., Tzeng, W.: Robust Key-Evolving Public Key Encryption Schemes. Crypology Eprint Archive Report (2001). URL http://eprint.iacr.org/2001/009
44. Vanrenen, G., Smith, S.: Distributing security-mediated PKI. In: 1st European PKIWorkshop: Research and Applications, pp. 218–231. Springer-Verlag LNCS 3093 (2004)

Gabriel Vanrenen is currently a software engineer at Wily Technology, Inc. in Brisbane, CA where he works on J2EE application performance management software. He received a B.A. in Computer Science (Summa Cum Laude) from Dartmouth College. At Dartmouth, he researched trusted third parties and PKI with his Senior Honors Thesis advisor Sean Smith.

Sean Smith is on the faculty of the Department of Computer Science at Dartmouth College. His current research and teaching focus on how to build trustworthy systems in the real world. He previously worked as a scientist at IBM T.J. Watson Research Center, doing secure coprocessor design, implementation and validation; and at Los Alamos National Laboratory, doing security designs and analyses for a wide range of public-sector clients. Dr. Smith was educated at Princeton (B.A., Math, but only Magna Cum Laude) and Carnegie Mellon (M.S., Ph.D., Computer Science).

John Marchesini is currently a Ph.D. candidate in the Computer Science Department at Dartmouth College. His advisor is Sean Smith, and his research interests are security, distributed systems, and PKI. Before going to Dartmouth, he worked as a software developer for the BindView Corporation and earned a B.S. in Computer Science from the University of Houston (Summa Cum Laude).