Intrusion Tolerant Cryptographic Techniques

DARPA Contract F30602-97-C-0326

Data Item Number A008: Technical Information Report.

Bill Aiello
aiello@bellcore.com

Dan Boneh
dabo@cs.stanford.edu

Abstract

We describe cryptographic techniques that enable large networked systems to maintain security when a number of hosts are penetrated and the network is snooped. Our main design principle is that at no point in time should long term security information be located at a single site. We use new and existing ideas from Threshold Cryptography to achieve this goal. Classic secret sharing techniques are inadequate since the machine on which the secret is reconstructed violates our design principal. In contrast, techniques of Threshold Cryptography enable distributed signature generation and distributed decryption without ever reconstructing the secret key at a single site. This report describes our approach and gives an overview of the cryptographic tools required. To demonstrate the effectiveness of our techniques we are building prototypes for an Intrusion Tolerant Certification Authority (ITCA) and an Intrusion Tolerant Web Server (ITWS).

1 Introduction

Past experience demonstrates our inability to design large networked systems that are impenetrable. As a result *intrusion detection* mechanisms are often used to detect potential intruders. Unfortunately, intrusion detection is a complex problem and there is currently no complete solution. In this report we describe a complementary approach we call *Intrusion Tolerance*. To compromise the security of an intrusion tolerant system, an attacker must penetrate a large number of hosts within a short pertiod of time. Such large scale attacks are much easier to detect.

Our intrusion tolerance techniques improve system security in two ways. First, they enhance system robustness by eliminating single points of failure. Second, they simplify the intrusion detection problem since only large scale attacks need to be detected. To provide intrusion tolerance we follow a strict design principle: *at no point in time should long term security information be located at a single site.* We apply this principle to long term private keys (as opposed to session-keys) by sharing them across multiple sites. As an added benefit, our techniques provide high availability of security services.

Secret sharing [38] is a classic technique for distributing secret information. Unfortunately, classic secret sharing is inadequate for our purposes. The problem is that in order to use a private key, it must be reconstructed at a *single* site. This site becomes a prime target for an attacker – a single point of failure. Indeed, an attacker that has maintained access to the site while the reconstructed key is in the clear may compromise the key. Using Shamir secret sharing directly it is not possible to satisfy our main design principle. Instead, we must use a different secret sharing paradigm that enables one to split a private key into a number of pieces, so that the key can be used to distributively compute a cryptographic function *without having to reconstruct it.*

To explain our approach we begin by considering the problem of building an intrusion tolerant Certification Authority (CA). Consider a CA that issues standard X.509 certificates using the RSA public key system. The CA has a secret RSA key that must be highly guarded. The key is used to generate signatures on incoming requests for certificates. Often, CA private keys are kept in a tamper resistant device and never leave the device. An example of such a system is BBN's SafeKeyper [36]. Unfortunately, the tamper resistant device is a single point of failure. By compromising it, an attacker can learn the private key and consequently generate certificates at will. Recent results on power cryptanalysis [27] illustrate the dangers of relying solely on tamper resistance for security.

To make the CA intrusion tolerant, one could split the private RSA key among a number of devices. As we shall see, each device can hold a share of the private key, and signature generation can take place without having to reconstruct it. More generally, we are building a system that behaves like the one in Figure **??**. The networked system contains two types of machines: clients and share servers. Share servers store shares of various private keys. Clients can contact the share servers and ask them to apply a certain private key to generate a signature on a given message. Each share server uses its shares of the private key to generate a piece of the signature and sends the result back to the client. The client assembles the pieces and obtains the signature. During the process, the private key is never reconstructed at a single location.

To provide fault tolerance it is desirable that the private keys be shared among the share servers in a t-out-of-n fashion, for some $t \leq n$. Any subset of t servers out of n can be used to generate a signature. A subset of $t-1$ servers has no information as to what the private key is. Hence, even if a small number of servers is offline, the CA can still generate certificates.

At a high level, the signature generation process can be summarized as follows:

Connect: The client connects to all share servers and performs an authentication protocol with each to prove its identity.

Request: The client sends the message to be signed to all share servers and asks them to generate the signature using a particular private key.

Verify: Each share server verifies that the client is authorized to perform the requested operation. This is essentially done using an access control matrix configured by an administrator.

Sign: Each share server applies its own share of the private key to the message to be signed and sends the result back to the client.

Combine: The client receives all responses from the share server and combines them into a signature.

Verify: The client verifies the validity of the resulting signature. If so, the client generates the required certificate.

Recover: If an invalid signature is generated, the client uses robustness properties of the sharing scheme to identify which of the share servers sent an invalid response (sending no response is treated as an invalid response). The client then ignores the corrupt share servers, and asks the remaining servers to generate a signature.

Repeat: This process is repeated until a valid signature is generated. If the signature cannot be generated, then the number of corrupt servers must be greater than the fault tolerance threshold of the system.

We briefly discuss the intrusion tolerance properties of this scheme. While we postpone a detailed description of the sharing scheme to later sections of the report, here we discuss the general issues that come up with such a design.

- Since the private key is shared among n servers in a t-out-of-n scheme, an attacker must penetrate at least t servers to learn the private key. A typical system configuration might use a 3-out-of-4 scheme in which case an attacker must penetrate three machines to recover the key. Due to the proactive update mechanism (discussed below) the attacker must penetrate all three servers within a single time period, e.g. within five minutes. We rely on an intrusion detection mechanism to detect such a large scale attack.

- To disable signature generation an attacker must interfere with the normal operation of more than $n-t$ servers. Interference amounts to crashing a share server, disconnecting a server from the network or corrupting the share stored on a server. Such interference will be immediately detected by an honest client attempting to generate a signature. Once again, when using a 3-out-of-4 scheme, the attacker must corrupt at least two machines.

- An interesting feature of the system is that if $n-t$ or fewer servers are disabled or attacked, the system continues to operate normally as if nothing happened. This enables an administrator to temporarily disconnect a share server which is suspected of having been penetrated, and reconnect it at a later time without interfering with system operation.

- A clever attacker may attempt to attack one of the client machines. By doing so, the attacker may expose one of the keys K_i (see Figure **??**) belonging to a client. Using the client's K_i, the attacker can communicate with the share servers impersonating the client. Note that this attack does not expose the private key. Indeed, to repeatedly use the key, the attacker must repeatedly impersonate the client, increasing the likelihood of detection. We implemented mechanisms that defend against the attack when the adversary is restricted to read-only access to the client (adversary can read any information on the client, but cannot modify anything). The idea is that each client-server pair keep a count of how many times the client initiated a connection to the server. An attacker who is able to discover a client's secret key, may fool the share servers once. However, once the client connects to the share servers, the sequence numbers won't match (since the servers incremented their counter, while the client did not). As a result, the attack is detected, and the client may be asked to update it's private key K_i. We note that an attacker who is able to gain full control of a client machine for an extended period of time, can defeat the sequence number mechanism by updating the counters on the client machine. Indeed, this form of attack cannot be prevented. Nevertheless, we point out (once again) that the attacker does not learn the private key. In this sense our system provides stronger security than the common approach of storing the CA's private key on the client machine (i.e. storing the key in one location, rather than in shared form).

Up until now we focused on signature generation. A similar process is applied when clients wish to decrypt a given RSA ciphertext. However, when the share servers are asked to decrypt a message, all communication between the client machine and the share servers must be protected from eavesdropping. The reason for this difference is simple: when asking the share servers to generate a signature (to be placed inside of a certificate), the final signature value is intended to become publicly available. Consequently, there is no reason to hide the messages that go back and forth between the client machine and the share servers. However, when asking the share servers to decrypt a ciphertext, the resulting plaintext should only be known to the client machine. As a result communication between the client machine and share servers must be protected. This is done by negotiating a session-key when the client first connects to the share servers. Note that although an adversary may penetrate the client, it will only be able to learn messages decrypted while it remains in control of the client. The adversary will not learn the decryption key or the decryption of any message that is decrypted when it does not have access to the client.

An example when the share servers are used in *decryption mode* (as opposed to signature mode) is the Intrusion Tolerant Web Server (ITWS) we are building. Our intrusion tolerance techniques are used to protect the web server's private key. The private key is used during session key negotiation with the browser. Indeed, to establish a session-key, the browser obtains the web server's authenticated public key. It then picks a random session-key, encrypts it using the server's public key and sends the resulting ciphertext to the server. The server decrypts the ciphertext using its private key and obtains the session-key (we note that in actual implementations much more randomness is used. In particular, the server is also involved in picking the session-key). To decrypt the encrypted session-key, the web server communicates with the share servers, since only they know the private key. The resulting decrypted session-key must only be known to the web server. Hence, communication between the web-server and the share-serves must be done over a secure line.

It is important to note that the techniques described above are *transparent to the message sender*. The sender (for instance the entity who asked for a certificate) obtains a standard RSA signature. It is not aware of the special precautions taken in order to protect the private key. This is an important feature of our approach since it requires no change in the certificate structure.

As an extension to this architecture, we note that one of the share servers can be implemented on a smartcard. This can be done by storing one share of the private RSA key on the card. The card can be made a mandatory participant in a successful decryption. Without the card the system does not have enough information to reconstruct the private key. This provides the attractive feature that in case the system has to be shutdown for a few days, a critical piece of the private key can be removed. Hence, even a large scale attack on the offline system is bound to fail. Once the system comes back online, all shares can be refreshed (using the proactive mechanism described below) thus making irrelevant any information an attacker was able to learn.

1.1 Diversity and proactive updates

The mechanism described above has the property that intrusion into a single component will not reveal the private key. In claiming that this design is intrusion tolerant we are implicitly assuming that breaking into two components is harder than breaking into one. To justify the assumption we rely on two complementary approaches: *diversity* and *proactive updates*.

- Diversity implies that the code running on machine S_1 should be different than the one on S_2. At an extreme, one can use different types of machines running different operating systems at geographically distinct sites. Indeed, the tools we are building support both Solaris and Windows-NT. Generally, maintaining software running on multiple operating systems is a problem. Since we intend to use proactive updates such degree of diversity may not be necessary.

- Proactive update was originally described by Ostrovsky and Yung [32] and later extended in [25] for the purpose of updating shares of a secret. Similar techniques can be used in our scenario and we intend to incorporate them into our system. The idea behind proactive update is to force the adversary to break into many components during a single time period in order to recover the secret. To achieve this, every time period (say once a day), the shares stored on the servers S_1, S_2, S_3 are replaced by new shares of the *same* private key. This means that an adversary must not only break into two components of the system, she must do so within one day.

We note that at the end of a proactive update the servers must engage in one trial decryption of a random encrypted message. This ensures that they still properly share the original secret key. The test is necessary to validate proper behavior during the update protocol. It also defeats a spoofing attack by the adversary who may garble some messages in transit or may cause one server to begin a proactive update on its own.

Our use of proactive updates suggests that an extreme degree of diversity of the computing environment may not be necessary. Our objective is now reduced to ensuring that undetected penetration of two hosts takes more than a single time period.

1.2 Generating shared keys

Our strict design principle states the a private RSA key should *never* be located on a single machine. The question is how is the private RSA key generated? A standard solution, is to use a *trusted dealer* to generate a private key and then distribute its share to the share servers. Unfortunately, a trusted dealer is a single point of failure and violates our design principle.

Our system generated shared RSA keys without the help of a trusted dealer. This involves a distributed computation among the share servers. At the end of the computation, all share server have shares of a private key, but none of them know what the private key is. The algorithm used for distributed generation of shared RSA keys is described in [1, 5, 30].

1.3 Roadmap to the rest of the report

The next two section describe the cryptographic tools needed to implement our designs. To understand these tools one must describe various types of secret sharing: *verifiable* secret sharing, *robust* secret and *proactive* secret sharing. In Section 2 we describe these concepts in the context of classic Shamir secret sharing. Then in Section 3 we show how these concepts apply to to distributing a secret RSA key. In particular, we describe protocols for *verifiable* sharing of an RSA key; *robust* sharing of an RSA key and *proactive* sharing of the key. The protocols in Section 3 are the ones used in the Intrusion Tolerant Web Server and Certification Authority we are building.

2 Secret Sharing

2.1 Simple k-out-of-n Secret Sharing

In this section we describe a very simple model for k-out-of-n secret sharing. The model consists of n players called shareholders which we denote by $S_1, \ldots, S_n$. Furthermore, there is a player D who has a secret s_0 unknown to the other players. We call this player the *dealer*. There are one or more players called *combiners*. A combiner may be one of the other players but it need not be. Let C denote a combiner. We assume that all players can communicate via private channels. Such channels can be instantiated in the real world using well-known public key cryptographic techniques [22, 2].

The dealer's responsibility is to run the *dealing protocol* which proceeds as follows. For each other shareholder, S_i, the dealer computes a *share* s_i of s_0 and sends it to S_i on their private channel. The share s_i is a function of s, i and the dealers private coins.

After the dealing protocol, there is a corruption phase carried out by an adversary. The goal of the adversary is either to learn some information about the secret or to prevent the combiners from producing the correct output. The adversary is allowed two types of actions. It can either read the share of a shareholder (but it cannot change the share) or erase the share of a shareholder. The adversary can perform as many of these actions as desired. Such an adversary is called a *fail-stop* adversary.

After the corruption phase, there is a combining protocol. Every shareholder whose share has not been erased sends its share to every combiner. If a combiner does not receive k or more shares it outputs "Too few shares" and halts. Otherwise, the combiner uses the shares to compute $\hat{s}_0$.

The above protocols constitute a simple k-out-of-n secret sharing protocol if the following holds:

1. If a fail-stop adversary erases more than $n - k$ the combiner should output "Too few shares." If the adversary erases at most $n-k$ shares, then the combiners output should equal the secret, i.e., $\hat{s}_0 = s_0$.

2. If the adversary reads $k - 1$ or fewer shares, it learns nothing about the secret.

The first two secret sharing scheme were devised by Shamir [38] and Blakley [4]. The Shamir scheme is based on polynomial interpolation while the Blakley scheme is based on the intersection of hyperplanes. The Shamir scheme is described in detail below.

2.2 Shamir Secret Sharing

Before describing the Shamir Secret Sharing protocol, first consider a degree d polynomial $p(x) = \sum_{i=0}^{d} c_i x^i$ over a finite field with more than $d+1$ elements. We will need the following two facts.

Fact 1. Suppose p is evaluated at $d+1$ distinct points, i.e., $y_i = p(x_i)$ for distinct x_i, $1 \leq i \leq d+1$. For each i, $1 \leq i \leq d+1$, define the degree d polynomial $L_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$. Then p is a linear combination of the L_i:

$$p(x) = \sum_{i=1}^{d+1} y_i L_i(x).$$

This is called the Lagrange interpolation of the points $(x_i, p(x_i))$, for $1 \leq i \leq d+1$.

Fact 2. Suppose the coefficients c_i of p are chosen independently and uniformly. Then, any d' pairs, for $d' \leq d$, $(x_1, p(x_1)), \ldots, (x_{d'}, p(x_{d'}))$, yield no information about $p(x')$ for $x' \neq x_i$, $1 \leq i \leq d'$.

Now Shamir Secret Sharing can be described as follows. In order for a dealer to share ℓ bit secrets with n shareholders, all parties agree on a prime q greater than 2^ℓ and n in advance.

The Dealing Protocol To k-out-of-n share a secret s_0, the dealer:

1. Picks a random $k-1$ degree polynomial $p(x) = \sum_{i=0}^{k-1} c_i x^i$ over F_q with the constraint that $p(0) = s_0$. That is, the dealer chooses $c_1, \ldots, c_{k-1}$ uniformly and independently and sets $c_0 = s_0$.

2. Computes $s_i = p(i)$ and sends s_i to P_i. We will call $s_1, \ldots, s_n$ the (random) Shamir shares, or the (random) polynomial shares of s_0.

The Combining Protocol The shareholders send their shares to each of the combiners. If fewer than k shares are sent to the combiners then the combiners output "Too few shares." If not, each combiner:

1. Chooses an arbitrary set of k shares. Let $\mathcal{G}$ be the indices of the shares chosen, i.e., the chosen shares are s_i for $i \in \mathcal{G}$.

2. Performs Lagrange interpolation on the points (i, s_i) for $i \in \mathcal{G}$ to get the polynomial p and computes $s_0 = p(0)$.

To see that this is indeed a k-out-of-n secret sharing protocol observe that by Fact 1, for a degree $k-1$ polynomial it is enough for the combiner to have k shares in order to perform Lagrange interpolation. And by Fact 2, if the adversary reads $k-1$ shares it learns no information about $p(0) = s_0$.

2.3 Verifiable Secret Sharing

In secret sharing as described above the dealer was trusted. However, secret sharing is often used as a sub-protocol. In such a case the dealer may be a corrupted party. In an effort to gain an advantage, this corrupted dealer might send purported shares $\hat{s}_i$ to the shareholder S_i which are not consistent with *any* secret. Verifiable Secret Sharing (VSS) protocols are designed explicitly to prevent this. A VSS protocol is a secret sharing protocol which detects whether the shares are *consistent*, that is, produced from one secret by the dealer's function for producing the shares. VSS protocols require the assumption of a broadcast channel. A broadcast channel has the property that any player which sends a message on the broadcast channel will be received by every player and every player can verify the player who sent the message. Assuming a robust routing mechanism, such a channel can be implemented using well known cryptographic techniques [3].

Verifiable secret sharing protocols were introduced in [9] and [21]. More practical implementations are due to Feldman [13] and Pedersen [33]. We describe one of Feldman's VSS protocols below.

The dealing protocol of Shamir's secret sharing is amended as follows. The prime q must be chosen so that $q-1$ does not have any small factors (see, for example, [13]). The dealer picks a random generator g of the multiplicative group of the field used in the secret sharing scheme. The honest dealer computes $v_i = g^{c_i}$ for $0 \leq i \leq k-1$. The dealer then broadcasts g and $v_0, v_1, \ldots, v_{k-1}$. Each shareholder S_i, upon receiving these broadcasts values and its share s_i, computes $i^j \bmod q-1$ for $j = 0, \ldots, k-1$ and checks whether $g^{s_i} = \prod_{j=0}^{k-1} (v_j)^{i^j}$ $\pmod q$. Note that if the dealer is honest then the equation holds since

$$g^{s_i} = g^{p(i)} = g^{\sum_{j=0}^{k-1} c_j i^j} = \prod_{j=0}^{k-1} v_j^{i^j} \pmod q$$

If the equation checks, the shareholder broadcasts the message "Consistent Share." If not, that shareholder broadcasts the message "Inconsistent Share." If an "Inconsistent Share" message is broadcasts then all the shareholders abort the protocol.

Notice that Feldman's scheme reveals the value $v_0 = g^s$ where s is the secret being shared. This reveals partial information about s to a polynomially bounded adversary, e.g., the least significant bit of s. However, assuming the hardness of the discrete log problem (i.e., computing discrete logarithms requires superpolynomial time), polynomial time adversaries cannot compute all of s. For example, assuming the hardness of the discrete log problem, no polynomial time adversary can compute any of the $O(\log |s|)$ most significant bits of s [29] . The security of any protocol using Feldman's secret sharing scheme can only be stated relative to the knowledge of g^s. To avoid this assumption the less practical scheme of Pederson's [33] may be used. See [25] for a discussion of these issues.

2.4 Robust Secret Sharing

In simple secret sharing we allowed the adversary to either read the value of a share or erase the value of a share. If we also allow the adversary to corrupt shares by changing their values (with the restriction that at most $n-k$ shares can be erased or corrupted) then the simple combining protocol may not produce the original secret of the dealer. This is true even if the shareholders were distributed consistent shares by the dealer. That is, when a combiner picks an arbitrary subset of k shares to use for Lagrange interpolation, some of those shares may be corrupted and the value of $\hat{s}_0$ computed by the combiner will not equal the share s_0. So, the combiner needs some way to determine which of the shareholders has corrupted its share.

Let a *corrupting* adversary be an adversary that can completely control the behavior of a fixed set[1] of $n-k$ shareholders, where $n-k < n/2$. The restriction on the number of corrupted shareholders ensures a majority of uncorrupted (honest) parties. Without an honest majority robust secret sharing is not possible. A *robust* secret sharing protocol guarantees that the combiner will always produce the correct secret s_0 even in the presence of a corrupting adversary. And furthermore, the corrupting adversary will not learn any useful information about the secret s_0.

.... `I'm sweeping under the rug the distinction between an adaptive adversary and a non-adaptive adversary. Any thoughts on how we should handle this here?` ....

To achieve a robust secret sharing protocol, the dealer uses the dealing protocol of verifiable secret sharing. In addition, the combining protocol is amended as follows. Each shareholder S_i, sends $\hat{s}_i$ to each combiner via a private channel, as before, and in addition, sends $\hat{v}_0^{(i)}, \ldots, \hat{v}_{k-1}^{(i)}$. For an honest shareholder, $\hat{s}_i = s_i$ and $\hat{v}_j^{(i)} = v_j^{(i)} = g^{c_j}$, for all $0 \leq j \leq k-1$. Upon receiving $\hat{v}_0^{(i)}, \ldots, \hat{v}_{k-1}^{(i)}$ from shareholder, S_i, the combiner first checks for each i that these are consistent verifying values, that is, $g^{\hat{s}_i} = \prod_{j=0}^{k-1} (\hat{v}_j^{(i)})^{i^j}$. For any i for which this equation does not hold, all of the values $\hat{v}_0^{(i)}, \ldots, \hat{v}_{k-1}^{(i)}$ are labeled discarded as is the shareholder S_i.

Next the combiner attempts to find the $\hat{v}_j^{(i)}$ which correspond to the $v_j^{(i)}$ supplied by the honest shareholders. For each j, the combiner finds the value $\bar{v}_j$ which appears more than $n/2$ times in the set of non-discarded values of the $\hat{v}_j^{(1)}, \ldots, \hat{v}_j^{(n)}$. Such a value is guaranteed to exist since more than $n/2$ shareholders are assumed to be honest. Moreover, $\bar{v}_j = v_j = g^{c_j}$ for the same reason. The combiner labels as *good* any non-discarded shareholder S_i for which $\bar{v}_j = \hat{v}_j^{(i)}$ for all $0 \leq j \leq k-1$. Now, the combiner can conclude that $\hat{s}_i = s_i$ for all good shareholders S_i. The combiner then performs Lagrange interpolation on a subset of size k of these good values to recover s_0.

2.5 Proactive Secret Sharing

Common security practice calls for secret keys to be refreshed periodically. For example, many systems require users to update their passwords periodically. That way, if an attacker has broken a password, he will only be able to enter the account until the new password is chosen. However, in certain situations, such as public key cryptosystems, long term secret keys are difficult to change on a regular basis. Indeed, the need to keep certain keys secret

[1] An adversary that is restricted to control a fixed set of shareholders in known as a *static* adversary. Ostrovsky and Yung [32] define the notion of an *adaptive* adversary that can control different shareholders at different time steps. The notion of proactive security, defined in the next section, is an attempt to defend against such an adversary.

for a very long period of time is what motivated secret sharing protocols. In order for a secret which is distributed using a k-out-of-n secret sharing protocol to be reconstructed by an adversary, the adversary must break into at least k machines. And this certainly takes more resources than breaking into one machine.

However, suppose a long term secret key has been k-out-of-n shared and an adversary is able to break into the machine of one shareholder over a network in T units of time. Further suppose that adversary wishes to remain as unobtrusive as possible and so does not edit any files. In due time, the network administrator may recognize that the machine has been penetrated but that no files have been changed. He or she may think that there is no cause for alarm. However, if k of the shareholders' machines are all similar, then it is reasonable to assume that the adversary may be able to collect k secrets on k different machines in approximately kT units of time and thus reconstruct the secret. What is needed to prevent this is a protocol to update the shares of the shareholders without changing the secret represented by the shares. Such a protocol is called *proactive* secret sharing. If the time between proactive share updates is much less than kT then the secret will remain hidden.

In proactive secret sharing there is an update phase, in which the update protocol is run, and an operational phase. Together they constitute a round. For simplicity we assume that the adversary breaks into a machine at the beginning of a phase and then leaves at the end of that phase or some later phase. We also assume that if an adversary breaks into a machine in an update phase that it has broken into the machine in the proceeding and following operational phases (At the beginning of the update phase a shareholder holds its share of the previous operational phase and by the end of the update phase it holds the share of the following operational phase. Thus, the adversary may gain access to the shares of both operational phases.)

If the adversary breaks into one set of machines in one phase, it may break into another set of machines in the next phase. The adversary's only restriction is that at any phase, it can break into at most $k-1$ machines. Such an adversary is called a *mobile* adversary and a secret sharing protocol which maintains shares consistent with a single secret against such an adversary is called a k-out-of-n *proactive* secret sharing protocol. If, when the adversary breaks into a shareholder it is only able to read the share or crash the machine, it is called a *fail-stop mobile adversary* (the adversary is not allowed to modify the share). Note, though, that we assume that a crash will always be detected and that the machine will be rebooted by the user by the next round. A proactive secret sharing protocol which maintains a single secret key against a fail-stop mobile adversary is called *simple*. By *robust* proactive secret sharing we will denote a protocol which maintains a single secret even against a *corrupting mobile* adversary, that is, against an adversary which may force a machine to behave arbitrarily upon breakin.

The mobile adversary model was introduced by Ostrovsky and Yung in [32]. They showed how to achieve secure distributed function evaluation when the number of servers was polynomially larger than the number of mobile adversaries. The model was subsequently used by Canetti and Herzberg [7] to maintain a distributed pseudorandom generator.

In order to demonstrate the main idea in proactivity, we will describe here a simple proactive secret sharing protocol for Shamir secret sharing. A robust proactive secret sharing protocol is described in [25]. The protocol is based on the following simple idea. Given the random sharing polynomial $p(x) = \sum_{i=0}^{k-1} c_i x^i$ with $c_0 = s_0$, let $r(x) = \sum_{i=1}^{k-1} d_i x^i$ be a random polynomial with the constant term equal to 0 and let $t_i = r(i)$ for $0 \leq i \leq n$. As before, $t_1, \ldots, t_n$ are the shares of the polynomial r and $t_0 = 0$ is the constant term of r which is the value being shared. Define $p'(x) = p(x) + r(x)$ and let $s'_i = p'(i)$ for $0 \leq i \leq n$, be the shares of the new polynomial p'. Note that $s'_i = p(i) + r(i) = s_i + t_i$. for all $0 \leq i \leq n$. In particular, $s'_0 = s_0$. That is, the polynomial p' is a new random polynomial for sharing s_0; and, the new shares are just the shares of p plus the shares of r.

The update protocol is as follows. Each shareholder S_i holds s_i, its share of the current polynomial p. Each S_i chooses a random polynomial $r^{(i)}$ with a constant term of zero. Then for each other shareholder S_j, S_i computes S_j's share of $r^{(i)}$, $t_j^{(i)} = r^{(i)}(j)$, and sends it to S_j. Each shareholder S_i upon receiving the shares $t_i^{(1)}, \ldots, t_i^{(n)}$ of polynomials $r^{(1)}, \ldots, r^{(n)}$ computes the new share $s'_i = s_i + \sum_{\ell=1}^{n} t_i^{(\ell)}$. This is S_i's share of the polynomial $p' = p + \sum_{\ell=1}^{n} r^{(\ell)}$. Each S_i then deletes s_i and $t_i^{(1)}, \ldots, t_i^{(n)}$.

The protocol just described is sufficient to defeat a fail-stop adversary which is restricted just to read up to $k-1$ shares in a phase. In order to withstand the full power of a fail-stop adversary which is allowed to delete shares, a protocol is needed to regenerate the shares. Based on the tools described in the above sections, a procedure for doing this and for dealing with a corrupting mobile adversary are described in [25].

3 Proactive and Robust RSA Sharing

In secret sharing, a secret is distributed among many players and then deleted. If the combining protocol is not run, then the secret will remain hidden as long as an adversary is not able to attack a large number of players. If the combining protocol is run, then the secret is reconstructed in a combiner. At that point there is no guarantee that the secret has not been exposed as an attacker may have penetrated a combiner. The combiner has become a single point of failure. For very sensitive operations, it may be unwise to use the secret key again after it has been combined. For example, suppose the key is the signing key (i.e., private key) for RSA. If an attacker ever gains access to the key, he is able to forge messages with impunity thereafter. For this reason, secret sharing is often called single secret sharing.

In spite of the fact that a naive application of secret sharing does not provide a strong method for distributing an RSA signing key, we will describe in this section protocols for distributing an RSA signing key *and* signing in such a way that the signing key is *never* reconstructed. Thus, there is no single point of failure and messages can continue to be signed on an ongoing basis.

Distributed RSA signing is an example of what is called *function sharing* or *threshold cryptography*. Threshold cryptography originated in several studies [11, 6, 10, 12]. Robust threshold schemes are known for discrete log based cryptosystems: El Gamal [8, 23], and DSS [20]. A framework for proactivizing discrete log based signatures was given in [24].

Using the results of [20] in [24] achieves a proactive DSS scheme.

As we will see below, threshold RSA is not as straightforward. The first provable scheme was given in [15, 37]. This threshold RSA scheme was made robust in [18, 19]. The first robust proactive RSA scheme was given in [17] using a different threshold RSA scheme as a starting point then that in the above papers. However, this scheme could not tolerate the maximum number of corrupt players. A scheme which is robust against the maximum number of corrupt players was recently given in [16]. More recently, Rabin [34] has shown a robust proactive RSA scheme which is substantially simpler than that in [16].

We will review Rabin's scheme below. In addition, we will describe a threshold RSA scheme developed in [**?**], called the *recursive scheme*, and show how to make this scheme proactive and robust. The recursive scheme does not scale well for a large number of corrupted players. However, when the number of corrupted players is a small constant then the scheme is quite efficient. In addition, as we will see, it has some advantages over Rabin's scheme. In particular, for a k-out-of-n scheme, only k players are needed to reconstruct a signature in the recursive scheme, whereas all n players are needed in Rabin's scheme. Hence $n-k$ machines can be down for maintenance and the remaining k machines can continue to jointly sign messages.

3.1 Simple RSA Sharing

Let us recall the well known RSA scheme. The public and private keys of a party are generated as follows. First, two random primes p and q of length k are generated, where k is the security parameter. Let $N = pq$. The operations will take place in the multiplicative group Z_N^* of order $\phi(N) = (p-1)(q-1)$. A random d is chosen uniformly in the range $1, \ldots, \phi(N) - 1$ such that $(d, \phi(N)) = 1$. The inverse e of d is then calculated, i.e., $ed \equiv 1 \pmod{\phi(N)}$. The party's public key is (N, e) while its private key is (p, q, d). While it is possible to both sign or encrypt with these RSA keys, in this report we will be primarily concerned with signing. Thus, we will call the owner of the private key the signer. The signer computes the signature σ of a message m as $\sigma = D(m) = m^d \bmod N$. A verifier verifies that σ is the signature of a message m by computing $E(\sigma) = \sigma^e \bmod N$ and checking if it is equal to m. Note that if $\sigma = D(m)$ then $E(\sigma) = m$ since $\sigma^d \bmod N = m^{de} \bmod N = m \bmod N$ (since $ed \equiv 1 \pmod{\phi(N)}$). Given only a signer's public key and a large number of signature, message pairs, $(m_1, \sigma_1), \ldots, (m_t, \sigma_t)$, for t a polynomial in k, computing (forging) the signature of a new message $m \notin \{m_1, \ldots, m_t\}$ is considered computationally infeasible [35, 28, 31].

We first describe a very simple scheme for "sharing " the signing function D among n shareholders. Before computing any shared signatures, the dealer[2] first distributes d of the RSA secret key to the shareholders using the following n-out-of-n secret sharing scheme. First, the dealer picks n random variable, $d_1, \ldots, d_n$, uniformly and independently in the range $1, \ldots, R$ and then sets $d_p = d - \sum_{i=1}^{n} d_i$. (The value of R is a parameter of the scheme.

[2]We use the notion of a dealer to simplify the exposition. However, our system does not require a dealer. Indeed, a dealer is a single point of failure that we set out to eliminate. Instead, our system uses distributed algorithms to generate shared RSA keys.

The larger R, the more secure the scheme. For the purposes of this discussion R should be thought of as approximately N). Now the dealer distributes the d_i to the n shareholders, broadcasts d_p (the "public" share) and deletes d. To sign a message m the dealer distributes m to each of the shareholders who then sign the message using their own share of d. That is, player S_i computes $m^{d_i} \bmod N$ which we will call S_i's signature share, σ_i. As in secret sharing, the shareholders now send their signature shares to a combiner. The combiner computes the public signature share, $\sigma_p = m^{d_p} \bmod N$, and then multiplies all the signature shares. That is, the combiner computes

$$\sigma_p \prod_{i=1}^{n} \sigma_i \bmod N = m^{d_p} \prod i = 1^n m^{d_i} \bmod N = m^d \bmod N$$

which is $D(m)$, the signature of m.

Note that d itself is never reconstructed and yet a valid signature was jointly computed by the shareholders and the combiner. Also note that an adversary which has read the shares of $n-1$ shareholders still does not know the value of d. This scheme was first introduced in [14]. The difficulty with this scheme, of course, is that if the adversary erases just one of the shares, the secret key d is lost.

The above scheme illustrates the basic idea of shared RSA. It enables n parties to share the signature generation function in an n-out-of-n sharing. To achieve fault-tolerance one often wishes to enable only some of the shares to be present for signature generation. That is, we want a scheme that enable k-out-of-n signature generation, and does not compromise security if the adversary reads $k-1$ shares. Using k-out-of-n Shamir secret sharing seems natural. However, this approach leads to immediate difficulties. To see this let $d_1, \ldots, d_n$ be the shares of a $k-1$ degree polynomial p with constant term d and coefficients computed $\bmod M$. Suppose the first k shareholders wish to jointly sign the message m. Shareholder S_i would compute $m^{d_i L_i(0)} \bmod N$ where, recall, $L_i(x) = \prod_{j \neq i} \frac{x-j}{i-j}$ for this case. The combiner receives all the signature shares from the first k shareholders and multiplies them. This would yield

$$\prod_{i=1}^{k} m^{d_i L_i(0)} \bmod N = m^d \pmod N.$$

However, in order for the last equality to be true, M must be a multiple of $\phi(N)$. But this is a serious problem. Knowing a multiple of $\phi(N)$ enables the shareholder to factor N. Hence, given M the shareholder could compute d from the public key e and thus break the scheme.

The above discussion shows that a simple Shamir secret sharing cannot be used to build a k-out-of-n sharing of the signature function. In spite of these difficulties, achieving simple k-out-of-n RSA sharing can be achieved in a variety of ways. Here we will describe the two approaches most suitable for our implementations. The first we denote the Recursive Scheme is due to Desmedt, Di Crescenzo and Burmester [**?**] and the second is due to T. Rabin [34].

The Recursive Scheme In the first scheme we will use the n-out-of-n secret sharing scheme described above, but we will not use it directly. Instead, for every subset of size k we will have a k-out-of-k share of the secret key d. Let us first illustrate this with a simple example. To 3-out-of-4 share d the dealer creates four 3-out-of-3 shares of d in the manner described above. Let these shares be denoted $d = d_p + d_2 + d_3 + d_4 = e_p + e_1 + e_3 + e_4 = f_p + f_1 + f_2 + f_4 = g_p + g_1 + g_2 + g_3$. These values are distributed to the four players as shown in Table 1.

	S_1	S_2	S_3	S_4	Public
$\{S_2, S_3, S_4\}$		d_2	d_3	d_4	d_p
$\{S_1, S_3, S_4\}$	e_1		e_3	e_4	e_p
$\{S_1, S_2, S_4\}$	f_1	f_2		f_4	f_p
$\{S_1, S_2, S_3\}$	g_1	g_2	g_3		g_p

Table 1: *A 3-out-of-4 sharing scheme. Each row sums to d. Players S_1, S_2, and S_3 hold the shares in the columns 1, 2, and 3, respectively. The shares in the* Public *column are broadcast. Observe that any triplet of players are able to generate signatures.*

Note that each player holds 3 shares. A similar scheme which requires each shareholder to hold only two shares is shown below in Table 2. When describing the proactive scheme it will be simpler to use the scheme of Table 1. It is possible to proactivize the scheme below using the methods in [25] but we omit the details here.

	S_1	S_2	S_3	S_4	Public
$\{S_2, S_3, S_4\}$		d_1	d_2	d_3	d_p
$\{S_1, S_3, S_4\}$	d_1		d_2	d_3	d_p
$\{S_1, S_2, S_4\}$	e_1	e_2		e_3	e_p
$\{S_1, S_2, S_3\}$	e_1	e_2	e_3		e_p

Table 2: *A 3-out-of-4 sharing scheme. Each row sums to d. Players S_1, S_2, and S_3 hold the shares in the columns 1, 2, and 3, respectively. The shares in the* Public *column are broadcast. As before, every triplet of players can generate a signature, however, each player need only store two shares, rather than three.*

To sign a message m the dealer distributes m to all four shareholders. The combiner then chooses a random subset of size three and broadcasts the name of this subset to the four shareholders. Each shareholder checks if it is in the subset. If it is, it then computes the appropriate signature share and sends it to the combiner.

... . `Can't seem to get the labeling and referencing of the table to match.` ....

The combiner can then compute the public signature share and multiply the four signature shares to produce the signature. For example, if the combiner chooses the subset $\{S_1, S_3, S_4\}$, then S_1, S_3, S_4, and the combiner compute $m^{e_1} \bmod N$, $m^{e_2} \bmod N$, $m^{e_3} \bmod N$, and $m^{e_p} \bmod N$, respectively, the product of which is $m^d \bmod N$.

The simple fail-stop adversary is allowed to read the shares of two parties or crash one party. If one party has crashed and it is in the subset chosen by the combiner then this will be detected by the combiner. The combiner then chooses the unique subset which does not contain the failed party and requests the signature shares for that subset. Also note that if the shares of two parties are known, the adversary does not have any information about the secret key since the outstanding shares of the two 3-out-of-3 secret sharing schemes are uniformly distributed in $\{1, \ldots, R\}$.

We now describe a general $n-1$-out-of-n scheme. The construction is recursive. We will first start by describing the construction for 7-out-of-8 sharing and then we will describe the general construction. To 7-out-of-8 share d, we start with two 2-out-of-2 sharings of d. The first and second shares we denote $d_1, D_1, d_{1,p}$ and $D_2, d_2, d_{2,p}$, respectively, where $d = d_1 + D_1 + d_{1,p} = D_2 + d_2 + d_{2,p}$.

Now consider the first share. We 3-out-of-4 share d_1 among S_1, S_2, S_3, S_4 and we 4-out-of-4 share D_1 among S_5, S_6, S_7, S_8. For the second share we 4-out-of-4 share D_2 among S_1, S_2, S_3, S_4 and we 3-out-of-4 share d_2 among S_5, S_6, S_7, S_8. See Table 3.1 below.

S_1, S_2, S_3, S_4	S_5, S_6, S_7, S_8	Public
3-out-of-4 share d_1	4-out-of-4 share D_1	$d_{1,p}$
4-out-of-4 share D_2	3-out-of-4 share d_2	$d_{2,p}$

Table 3: *A recursive construction for a 7-out-of-8 sharing scheme. The two 2-out-of-2 shares of d are given by the equation $d = d_1 + D_1 + d_{1,p} = D_2 + d_2 + d_{2,p}$. The rows denote how the sharing schemes of d_1, D_1, D_2, d_2 are to be matched up.*

To reconstruct d, the combiner first broadcasts the name of a set of size 7. Using Table 3.1, $\{S_1, S_2, S_3, S_4\}$ know which value of d_1 or D_2 to reconstruct and $\{S_5, S_6, S_7, S_8\}$ know which value of D_1 or d_2 to reconstruct. If a 3-out-of-4 reconstruction is required, the shareholders can use Table 1. So for example, if the subset of size 7 is the set of shareholders excluding S_1, then S_2, S_3, and S_4 reconstruct d_1 and S_5, S_6, S_7, and S_8 reconstruct D_2.

To see that an adversary reading the shares of a subset of size 6 cannot reconstruct d, observe that such a subset can only be split in three ways as subsets of $\{S_1, S_2, S_3, S_4\}$ and $\{S_5, S_6, S_7, S_8\}$: a subset of size 2 and 4, respectively; or, a subset of size 3 and 3, respectively; or, a subset of size 4 and 2, respectively. Consider the first case. The adversary can reconstruct D_1 and d_2. However, the adversary can gain no information on d_1 or D_2 and thus on d. In the second case, an adversary could reconstruct d_1 and d_2 but he cannot gain any information about D_1 and D_2 and thus on d. The third case is symmetric with the first.

We will now describe the general $n-1$-out-of-n scheme. For simplicity we will assume that n is a power of 2 greater than four. We will let I^n denote the set of all players $\{S_1, \ldots, S_n\}$. We will denote the first $n/2$ players by $I_1^{n/2}$ and the second $n/2$ by $I_2^{n/2}$. In general, I_i^m denotes the ith interval of size m, that is, $I_i^m = \{S_j \mid (i-1)m+1 \leq j \leq im\}$. Note that $\cup_{i=1}^{n/m} I_i^m = I^n$. Each interval of players can be subdivided into two equal size intervals, that is, $I_i^m = I_{2i-1}^{m/2} \cup I_{2i}^{m/2}$.

A $n-1$-out-of-n sharing of d among I^n consists of

1. A 2-out-of-2 sharing of d: $d_1 + D_1 + d_{1,p} = d$. Then,

   (a) A $\frac{n}{2} - 1$-out-of-$\frac{n}{2}$ sharing of d_1 among $I_1^{n/2}$. And,

(b) A $\frac{n}{2}$-out-of-$\frac{n}{2}$ sharing of D_1 among $I_2^{n/2}$. Plus,

2. A 2-out-of-2 sharing of d: $D_2 + d_2 + d_{2,p} = d$. Then,

   (a) A $\frac{n}{2}$-out-of-$\frac{n}{2}$ sharing of D_2 among $I_1^{n/2}$. And,

   (b) A $\frac{n}{2} - 1$-out-of-$\frac{n}{2}$ sharing of d_2 among $I_2^{n/2}$.

Using an argument similar to that above for the 7-out-of-8 scheme, the shares of $n-2$ shareholders cannot reconstruct d. The $n/2$-out-of-$n/2$ sharing can be done immediately. The $(n/2)-1$-out-of-$n/2$ sharing can be achieved by further recursion. The general recursive step is given below. To facilitate keeping track of subshares we will index them by the shareholder intervals.

A $m-1$-out-of-m sharing of $d(I_i^m)$ among I_i^m, for m a power of 2 greater than 4, and for $1 \le i \le n/m$ consists of

1. A 2-out-of-2 sharing of $d(I_i^m)$: $d(I_{2i-1}^{m/2}) + D(I_{2i}^{m/2}) + d_{1,p}(I_i^m) = d(I_i^m)$. Then,

   (a) A $\frac{m}{2} - 1$-out-of-$\frac{m}{2}$ sharing of $d(I_{2i-1}^{m/2})$ among $I_{2i-1}^{m/2}$. And,

   (b) A $\frac{m}{2}$-out-of-$\frac{m}{2}$ sharing of $D(I_{2i}^{m/2})$ among $I_{2i}^{m/2}$. Plus,

2. A 2-out-of-2 sharing of $d(I_i^m)$: $D(I_{2i-1}^{m/2}) + d(I_{2i}^{m/2}) + d_{2,p}(I_i^m) = d(I_i^m)$. Then,

   (a) A $\frac{m}{2}$-out-of-$\frac{m}{2}$ sharing of $D(I_{2i-1}^{m/2})$ among $I_{2i-1}^{m/2}$. And,

   (b) A $\frac{m}{2} - 1$-out-of-$\frac{m}{2}$ sharing of $d(I_{2i}^{m/2})$ among $I_{2i}^{m/2}$. Plus,

When $m = 4$, the recursion stops. The 3-out-of-4 sharing for intervals of size four is implemented using Table 1.

Observe that this recursive procedure has the following property. For a fixed m (m a power of two), $4 \le m < n$, and each i, $1 \le i \le n/m$, there is a call for two procedures using the interval of shareholders I_i^m: (a) $m-1$-out-of-m sharing of $d(I_i^m)$ among I_i^m, and (b) m-out-of-m sharing of $D(I_i^m)$ among I_i^m. The later will require each shareholder in I_i^m to store one share of $D(I_i^m)$. The former requires a recursive call. Since each shareholder is in some interval of size m for each $m \in \{n/2, \ldots, 4\}$, each shareholder will have $\log n - 2$ shares due to m-out-of-m sharing for that interval. In addition, each shareholder will need three shares to do 3-out-of-4 sharing in the interval of size 4 to which it belongs at the bottom of the recursion. Thus, the total number of shares required for each shareholder is $\log n + 1$.

This recursive construction for $n-1$-out-of-n sharing can be generalized to a construction for $n-c$-out-of-n for any constant c. The highest level of recursion is given below.

A $n-c$-out-of-n sharing of $d(I^n)$ among I^n, consists of

1. A 2-out-of-2 sharing of $d(I^n)$: $d_c(I_1^{n/2}) + d_0(I_2^{n/2}) + d_{0,p}(I^n) = d(I^n)$. Then,

   1. A $\frac{n}{2} - c$-out-of-$\frac{n}{2}$ sharing of $d_c(I_1^{n/2})$ among $I_1^{n/2}$. And,

2. A $\frac{n}{2}$-out-of-$\frac{n}{2}$ sharing of $d_0(I_2^{n/2})$ among $I_2^{n/2}$. Plus,

2. A 2-out-of-2 sharing of $d(I^n)$: $d_{c-1}(I_1^{n/2}) + d_1(I_2^{n/2}) + d_{1,p}(I^n) = d(I^n)$. Then,

1. A $\frac{n}{2} - (c-1)$-out-of-$\frac{n}{2}$ sharing of $d_{c-1}(I_1^{n/2})$ among $I_1^{n/2}$. And,
2. A $\frac{n}{2} - 1$-out-of-$\frac{n}{2}$ sharing of $d_1(I_2^{n/2})$ among $I_2^{n/2}$. Plus,

$$\vdots$$

c. A 2-out-of-2 sharing of $d(I^n)$: $d_0(I_1^{n/2}) + d_c(I_2^{n/2}) + d_{c,p}(I^n) = d(I^n)$. Then,

1. A $\frac{n}{2}$-out-of-$\frac{n}{2}$ sharing of $d_0(I_1^{n/2})$ among $I_1^{n/2}$. And,
2. A $\frac{n}{2} - c$-out-of-$\frac{n}{2}$ sharing of $d_c(I_2^{n/2})$ among $I_2^{n/2}$. Plus,

The general recursive procedure for $n - c$-out-of-n is easily derived from the procedure above. The number of shares held by each shareholder using this procedure can be shown to be $O(\log^c n)$ for a constant c.

Rabin's Scheme Another approach to RSA sharing was developed by Rabin in [34]. The k-out-of-n scheme is simple to describe. First, the dealer n-out-of-n shares the secret d, as above. That is, player S_i receives a share d_i, and d_p is broadcasts where $d = d_p + \sum_{i=1}^{n} d_i$. Each player then computes a k-out-of-n Shamir secret sharing of its share and sends the shares to the appropriate party. Let S_i's Shamir shares of d_i be $t_1^{(i)}, \ldots, t_n^{(i)}$ where $t_j^{(i)}$ is sent to S_j on a private channel.

As in the previous scheme, to sign a message m, each shareholder computes a signature share $\sigma_i = m^{d_i} \bmod N$ and sends it to the combiner. If the combiner receives all of the signature shares it simply multiplies them all to retrieve the signature $\sigma = m^d \bmod N$. Against a fail-stop adversary $n - k$ or fewer shareholders may crash. Denote this set of shareholders by the set $\mathcal{B}$. With this adversarial model the combiner can easily detect which shareholders have crashed since they simply do not sent a signature share.

Since d was n-out-of-n shared, a signature cannot be computed until the shares of d held by the shareholders in $\mathcal{B}$ are reconstructed. Once the machines in the set $\mathcal{B}$ have been rebooted, each can act as the combiner for the Shamir sharing of its share of d. Suppose S_i is in $\mathcal{B}$. S_i chooses a subset of $\bar{\mathcal{B}}$ of size k which we denote $\mathcal{G}$. S_i retrieves $t_j^{(i)}$ from each S_j in $\mathcal{G}$ and uses these shares to reconstruct d_i. Once all the S_i in $\mathcal{B}$ have reconstructed their shares, joint signing may resume.

3.2 Proactive RSA Sharing

The Recursive Scheme We will now describe how to proactivizing the recursive scheme. Not surprisingly, it will be done recursively. Recall that at the top level of recursion, d is

2-out-of-2 shared in two ways between $I_1^{n/2}$ and $I_2^{n/2}$: $d = d(I_1^{n/2}) + D(I_2^{n/2}) + d_{1,p}(I^n) = D(I_1^{n/2}) + d(I_2^{n/2}) + d_{2,p}(I^n)$. We want to get two new 2-out-of-2 shares of d. Since the shares $d(I_1^{n/2}), D(I_2^{n/2}), D(I_1^{n/2}), d(I_2^{n/2})$ do not reside in any shareholder we must do this in a distributed manner. Each shareholder S_i picks two random sharings given by the following equation:

$$
\begin{aligned}
0 &= (n/2-1)r_i(I_1^{n/2}) + (n/2)R_i(I_2^{n/2}) + r_{i,1,p}(I^n) \\
&= (n/2)R_i(I_1^{n/2}) + (n/2-1)r_i(I_2^{n/2}) + r_{i,2,p}(I^n).
\end{aligned}
$$

S_i then sends $r_i(I_1^{n/2})$ and $R_i(I_1^{n/2})$ to each member of $I_1^{n/2}$, and it sends $R_i(I_2^{n/2})$ and $r_i(I_2^{n/2})$ to each member of $I_2^{n/2}$, and it broadcasts $r_{i,1,p}(I^n)$ and $r_{i,2,p}(I^n)$. Each shareholder S_i in $I_1^{n/2}$ now does the following. To each share whose parent in the recursive construction is $d(I_1^{n/2})$ it adds $\sum_i r_i(I_1^{n/2})$ and to each share whose parent in the recursive construction is $D(I_1^{n/2})$ it adds $\sum_i R_i(I_1^{n/2})$. Likewise, each shareholder S_i in $I_2^{n/2}$ does the following. To each share whose parent in the recursive construction is $d(I_2^{n/2})$ it adds $\sum_i r_i(I_2^{n/2})$ and to each share whose parent in the recursive construction is $D(I_2^{n/2})$ it adds $\sum_i R_i(I_2^{n/2})$. Furthermore, $\sum_i r_{i,1,p}(I^n)$ is added to $d_{1,p}(I^n)$ and $\sum_i r_{i,2,p}(I^n)$ is added to $d_{2,p}(I^n)$.

To see why this works, suppose that the subset of size $n-1$ used to "reconstruct" d consists of $I_1^{n/2}$ plus a subset J of size $n/2-1$ of $I_2^{n/2}$. Before the proactive phase we know that each element of $I_1^{n/2}$ has a share such that the sum of these shares is $D(I_1^{n/2})$. In addition, each element of J has a share such that the sum of these shares is $d(I_2^{n/2})$. After proactivization, each share of $D(I_1^{n/2})$ has been augmented by $\sum_i R_i(I_1^{n/2})$. Thus, the sum of the shares of $I_1^{n/2}$ has been augmented by $(n/2)\sum_i R_i(I_1^{n/2})$. Likewise each share of $d(I_2^{n/2})$ has been augmented by $\sum_i r_i(I_2^{n/2})$. Thus, the sum of the shares of J has been augmented by $(n/2-1)\sum_i r_i(I_2^{n/2})$. In addition, the public value has been augmented by $\sum_i r_{i,2,p}(I^n)$. Thus, $D(I_1^{n/2}) + d(I_2^{n/2}) + d_{2,p}(I^n)$ $(= d)$ has been augmented by

$$\sum_i (n/2)R_i(I_1^{n/2}) + (n/2-1)r_i(I_2^{n/2}) + r_{i,2,p}(I^n) = 0. \tag{1}$$

Rabin's Scheme Rabin's scheme is quite easy to proactivize since n-out-of-n sharing is easy to proactivize. Each player S_i, for $1 \le i \le n$, will pick a random sharing of 0. That is, S_i will choose n random numbers $\delta_1^{(i)}, \ldots, \delta_n^{(i)}$ uniformly and independently in the range $[-R, \ldots, R]$ and send $\delta_j^{(i)}$ privately to each player S_j, for $1 \le j \le n$. In addition, S_i will compute a value $\delta_p^{(i)}$ such that $\delta_p^{(i)} + \delta_1^{(i)} + \cdots + \delta_n^{(i)} = 0$ and broadcast this value.

Now, each player computes the sum of the shares of 0 it received from the other players. That is, S_j computes $\delta_j = \delta_j^{(1)} + \cdots + \delta_j^{(n)}$. S_j then adds δ_j to each of its current share. In addition, the new broadcast value is augmented by $\delta_p = \delta_p^{(1)} + \cdots + \delta_p^{(n)}$. To see that this remains a secret sharing of d, note that share j is augmented by δ_j and thus, the whole sum is augmented by

$$\delta_p + \sum_j \delta_j \quad = \quad \sum_i \delta_p^{(i)} + \sum_j \sum_{i=1}^{n} \delta_j^{(i)}$$

$$= \sum_i (\delta_p^{(i)} + \delta_1^{(i)} + \cdots + \delta_n^{(i)}) = 0.$$

To complete the proactive update, each player must compute and distribute a random k-out-of-n polynomial sharing of its new additive share.

3.3 Robust Proactive RSA Sharing

Rabin's Scheme We will first describe how to make Rabin's simple (non-proactive) scheme robust. We will then describe how to make the proactive updates robust as well. As described above, after a shareholder receives its additive share, it then k-out-of-n polynomially shares it. We make three modifications. First, in the dealing protocol after the dealer gives the additive shares d_i to S_i for each i, the dealer broadcasts $w_i = g^{d_i}, \ldots, w_n = g^{d_n}$, $w_p = g^{d_p}$. These will be the witnesses of the shareholders additive shares.

The second modification is as follows. Rather than performing simple k-out-of-n polynomial sharing of its additive share, each player will perform Verifiable Secret Sharing to k-out-of-n polynomially share its additive share. [3] (We will also assume that each player signs its polynomial shares. This may be automatically done when implementing the private channel assumption.) This prevents an adversary from sending values which are not consistent with a single value. However, in and of itself, it does not prevent a corrupted shareholder from sharing the wrong value. This is where the witnesses w_i comes into play. Recall that if S_i performs VSS, it computes and broadcasts $v_j = g^{c_j}$ where c_j are the coefficients of its random sharing polynomial. In this case $v_0 = g^{d_i} = w_i$. Thus, using the witness broadcast by the dealer, each players can check that S_i sent it a consistent share of d_i.

If a player detects that its share from S_i is not a consistent share of d_i then this player can broadcast a message to disqualify S_i. While a method like this is needed to ensure the robustness of the additive shares, this is too naive. A single corrupt player upon receiving its verifiable polynomial shares of the additives shares of the other players could send a message disqualifying each of them, and thus, destroy the robustness. Thus we will not allow a player to disqualify S_i without letting the other players check if the disqualification is deserved. Instead a player can broadcast a challenge to S_i by broadcasting its (signed) polynomial share. Since S_i had broadcast $v_j = g^{c_j}$ where c_j are the coefficients of its random sharing polynomial, every player can now verify whether the polynomial share of the challenging player is valid. If S_i fails a challenge by any other player it is disqualified.

The first two modifications makes the k-out-of-n sharing of the additive shares robust. The third modification will make the signature sharing robust. In the simple RSA sharing scheme of Rabin's, either the combiner receives correct signature shares from all n shareholders or it fails to receive one or more signature shares and thus knows which shareholders have failed. However, a corrupting adversary may send an incorrect signature share rather than

[3] In Feldman's VSS, arithmetic takes place $\bmod\, p$ for a prime p. For RSA, the arithmetic takes place mod pq for primes p and q and thus arithmetic in the exponent takes place $\bmod \phi(pq)$. However, the adversary must not discern $\phi(pq)$ or else it could factor pq. These differences neccesitate minor changes to Feldman's VSS protocol which are given in [34]. However, these changes do not affect our presentation here.

simply crashing. The combiner can quickly detect that it has received one or more incorrect signature shares simply by computing a trial encryption on the message. That is, if $\tilde{\sigma}$ is the product of the signature shares received by the combiner, then $\tilde{\sigma}^e \bmod N$ will equal the message if all of the signature shares are correct. Thus detecting the existence of one or more faulty signature shares is easy. Determining which shareholder sent the faulty signature is more difficult. If the trial encryption produces a fault, the combiner will engage in a protocol with S_i, for all i, to determine if the signature share it sent is equal to m^{d_i}. One such protocol is given in [19]. Recall that shareholder S_i was given a witness $w_i = g^{d_i} \bmod N$. If S_i was honest then its signature share was $\sigma_i = m^{d_i} \bmod N$. In the protocol, the combiner will ask S_i for a zero-knowledge proof that the discrete log base m of σ_i is equal to the discrete log base g of w_i. The protocol is efficient but requires the primes which compose the RSA modulus to have a special form: $N = pq$ where $p = 2p' - 1$ and $q = 2q' - 1$ for primes p' and q'.

When the RSA modulus is computed distributively, it is not known how to compute primes of this form. Thus, we will use a protocol in [18]. For this protocol, each player S_i will need another witness in addition to w_i. The dealer will compute $w_i' = h^{d_i}$ for a random h and broadcast this value.

After the combiner has received the purported signature share $\hat{s}_i$ from S_i, for all i and detected that one or more is not correct, the following protocol commences.

- For all i: The combiner chooses a random k_i, ℓ_i, ℓ_i' from 1 to N. The combiner sends $z_i = m^{k_i} g^{\ell_i} h^{\ell_i'}$ to S_i.
- For all i: An honest S_i sends $a = H(z_i^{d_i})$ to the combiner, where H is a hash function such as SHA-1.
- For all i: The combiner checks that $a = H\left(\hat{s}_i^{k_i} w_i^{\ell_i} w_i'^{\ell_i'}\right)$.

If all S_i pass step 3, the protocol is repeated. The protocol is repeated for a maximum of $2 \log N$ times. For the purposes of this protocol we assume that the message m is random. (This is achieved by adding a random blinding to a message m' when it is received. That is, the receiving server picks a random r and computes $m = m'r^e \bmod N$. It sends m to the S_i and r to the combiner). With this assumption, it can be shown that with probability at most $O(1/N)$ over m, g, h, k_i, ℓ_i, ℓ_i', an incorrect signature share $\hat{s}_i$ will not be detected.

.... `With probability > 1-x over m, g and h, we will have probability > 1 -` $O(1/N^2)$ `over` k_i, l_i, l_i' `of detecting the fault. I'm not sure what x is here. I haven't added the hash yet.` ....

.... `Unfortunately if p-1 and q-1 are smooth then the probability that two random elements` w_i, w_i' `generate` Z_N^* `is quite small` $(1/\log N)$. `So, we need to make some assumption on` p, q `such as` $\varphi(p-1)/p > \epsilon$ `and the same holds for` q. `Then` $x > \epsilon^2$. `Taking` $\epsilon = 1/10$ `will be satisfied by at least 99% of the primes.` ....

This concludes the changes to the protocol necessary to make the non-proactive version of the scheme robust.

Making the proactive update robust is now straightforward. When a player S_i computes a random n-out-of-n additive sharing of 0, $\delta_1^{(i)}, \ldots, \delta_n^{(i)}$, it also computes witness for these shares, $\hat{w}_1^{(i)} = g^{\delta_1^{(i)}}, \ldots, \hat{w}_n^{(i)} = g^{\delta_n^{(i)}}$, $\hat{w}_p^{(i)} = g^{\delta_p^{(i)}}$. As before S_i sends $\delta_j^{(i)}$ to S_j on a private channel, and in addition, S_i broadcasts $\hat{w}_p^{(i)}$ and the witnesses.

Now each S_j checks that the share it received from each S_i is consistent with a sharing of 0. That is, S_j checks that the witness and the additive share have the right relation: $g^{\delta_j^{(i)}} = \hat{w}_j^{(i)}$. In addition it checks that $\hat{w}_1^{(i)} \cdots \hat{w}_j^{(i)} \cdots \hat{w}_n^{(i)} \cdot \hat{w}_p^{(i)} = g$. (If it does not check, then as above, S_j will challenge S_i by broadcasting $\delta_j^{(i)}$. The other players then perform the check on this value. If S_i fails any challenge then it is considered corrupted and rebooted.) If the values are verified then S_j can add $\delta_j^{(1)}, \ldots, \delta_j^{(n)}$ to its d_j to get d'_j. In addition, S_j computes the new witness of d'_j. This is the old witness times the product $\hat{w}_j^{(1)} \cdots \hat{w}_j^{(n)}$. In addition, all the players update the value of d_p and its witness in the obvious way.

Now each player must polynomial share its new additive share. This is done just as in the robust scheme using VSS and the new witnesses.

Recursive Scheme To make the Recursive Scheme robust, many of the same techniques used above are employed. Let us again first concentrate on what is needed in the non-proactive case before discussing the proactive case.

In this case the Recursive scheme is significantly easier since the shareholders do not polynomially share their additive shares. All we will need to do is provide a method for the combiner to determine which shareholders, if any, have produced faulty signature shares. As before, to do so each shareholder will need a witness for each of its shares. If the dealer is considered honest, then for each share s it deals to a shareholder it also deals the witness g^s. (To check whether the dealer has produced shares consistent with a single secret, the dealer must compute witnesses for the two 2-out-of-2 sharings for each recursive calls. If each of these sharings is broadcast, each shareholder will receive $O(n)$ witnesses and each will be able to check whether the witnesses are consistent with the secret d using the recursive structure of the scheme.)

Once the shareholders have witnesses for their shares, the combining protocol is nearly identical to that for the Rabin scheme with one important difference. In the case of a fail-stop adversary (i.e., at most k shareholders crash but the rest behave properly), the Rabin scheme must halt operation and reconstruct the lost additive shares using the polynomial shares. In the recursive scheme there is still a set of $n - k$ good shareholders and so, by construction, they can still jointly sign a message. Thus, operation need not stop. In the case of a proactive adversary (which we will cover below) which may crash more machines in subsequent time steps, the good surviving shareholders must reconstruct the keys of the failed shareholders. However, this can be done concurrently while the surviving players continue to sign messages.

To guard against a corrupting adversary, upon receiving the signature shares from a subset of size $n-k$ and multiplying them to get a candidate signature, the combiner performs a trial encryption, as in the Rabin scheme. If the trial encryption of the signature yields the message then the combiner sends the signature. If not, the combiner will ask each member of the subset to prove, using the publicly known witnesses of its shares, that its signature share was correct as above.

For making the proactive protocol robust we will consider both fail-stop adversaries and corrupting adversaries. We'll take them in turn. For fail-stop adversaries, as mentioned above, the good surviving subset of size at least $n-k$ must reconstruct the shares of the failed shareholders after the failed shareholders have been rebooted. The procedure to do so is as follows. Recall that given a name of the good subset of size $n-k$ each shareholder in the set has a share. Furthermore, the set of all such shares in the subset of size $n-k$ sum to d. Suppose S_i is in the designated subset and its appropriate share is d_i. S_i acts as the dealer in the recursive procedure to share d_i. Each S_j in the designated set does likewise. Each player, both in and out of the designated set, receives a tuple of shares from each player in the designated set. Each player then adds the tuples component-wise to get a single tuple of shares. This tuple of shares is the new set of shares for that player. Since the shares of the designated set sum to d, the sets of recursive shares for all the players constitute a sharing of d.

To handle corrupting adversaries for the proactive scheme both the share reconstruction procedure above and the proactive update procedure must guard against corrupted players. To guard against corrupted players in the share reconstruction protocol, each shareholder in the designated set must compute witnesses for each recursive share of its additive share. These witnesses, along with the already public witness of the additive share, can be used to ensure that the shareholder is producing recursive shares consistent with its additive share. The proactive phase is handled similarly. Each player computes and broadcasts witnesses for the shares it computes in the proactive phase. The validity of these shares in then checked by multiplying the witnesses and using Eq. 1 in the exponent in the obvious way. As in the Rabin scheme, any player may challenge a sharing player (for either of the above schemes) which is followed by a public checking protocol.

4 Summary

The report describes a number of practical cryptographic tools used for sharing a private RSA key among a number of sites. The shared private key can be used to generate signatures or decrypt messages without having to reconstruct it at a single location. In addition, the techniques described provide the following features:

Robustness When a client (gateway) asks the servers to apply the shared key it can test the responses to test if some of the servers are misbehaving. A corrupt server can be ostrosized and a proper signature (or decryption) can be carried out using the remaining honest servers.

Proactiveness At fixed intervals the servers communicate with each other to refresh their shares of the private key. As a result, an attacker must penetrate a large number of servers within a short period of time in order to expose the private key.

To demonstrate the effectiveness of these techniques we are incorporating them into our intrusion tolerant system. Both our Intrusion Tolerant CA and our Intrusion Tolerant Web Server make use of these tools.

References

[1] W. Aiello, D. Boneh, M. Franklin, and J. Horwitz. Protocols for secure generation of shared rsa keys. In *DARPA Technical Information Report, Data Item A007, Contract F306002-97-C-0326 (Intrusion Tolerance via Threshold Cryptography)*, 1998.

[2] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology—CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 22–26 August 1993.

[3] Shimshon Berkovits. How to broadcast a secret. In D. W. Davies, editor, *Advances in Cryptology—EUROCRYPT 91*, volume 547 of *Lecture Notes in Computer Science*, pages 535–541. Springer-Verlag, 8–11 April 1991.

[4] G. Blakley. Safeguarding cryptographic keys. In *Proceedings of AFIPS 1979 National Computer Conference*, pages 313–317, 1979.

[5] D. Boneh and M. Franklin. Efficient generation of shared rsa keys. In Kaliski Jr. [26], pages 425–439.

[6] C. Boyd. Digital multisignatures. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 241–246. Claredon Press, 1989.

[7] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In Yvo G. Desmedt, editor, *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 425–438. Springer-Verlag, 21–25 August 1994.

[8] M. Cerecedo, T. Matsumoto, and H. Imai. Efficient and secure multiparty generation of digital signatures. *IEICE Trans. Fundamentals*, E76-A(4):532–545, 1993.

[9] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science*, pages 383–395, Portland, Oregon, 21–23 October 1985. IEEE.

[10] R. Croft and S. Harris. Public key cryptography and re-usable shared secrets. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 189–201. Claredon Press, 1989.

[11] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology—CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127. Springer-Verlag, 1988, 16–20 August 1987.

[12] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer-Verlag, 1990, 20–24 August 1989.

[13] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science*, pages 427–437, Los Angeles, California, 12–14 October 1987. IEEE.

[14] Y. Frankel. A practical protocol for large group oriented networks. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology—EUROCRYPT 89*, volume 434 of *Lecture Notes in Computer Science*, pages 56–61. Springer-Verlag, 1990, 10–13 April 1989.

[15] Y. Frankel and Y. Desmedt. Parallel reliable threshold multisignature. In *Department of EE & CS, University of Wisconsin-Milwaukee, Tech. Report TR-92-04-02*, Department of EE & CS, University of Wisconsin-Milwaukee, 1992.

[16] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Optimal-resilience proactive public-key cryptosystems. In *38th Annual Symposium on Foundations of Computer Science*, pages 384–393, Miami Beach, Florida, 20–22 October 1997. IEEE.

[17] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Proactive RSA. In Kaliski Jr. [26], pages 440–454.

[18] Yair Frankel, Peter Gemmell, and Moti Yung. Witness-based cryptographic program checking and robust function sharing. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 499–508, Philadelphia, Pennsylvania, 22–24 May 1996.

[19] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of RSA functions. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 157–172. Springer-Verlag, 18–22 August 1996.

[20] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli Maurer, editor, *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371. Springer-Verlag, 12–16 May 1996.

[21] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, July 1991.

[22] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[23] L. Harn. Group oriented (t, n) digital signature scheme. *IEEE Proc. Comput. Digit. Tech.*, 141(5), Sept 1994.

[24] Amir Herzber, M. Jakobsson, Satnislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In *1997 ACM Conference on Computers and Communication Security*, 1997.

[25] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In Don Coppersmith, editor, *Advances in Cryptology—CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352. Springer-Verlag, 27–31 August 1995.

[26] Burton S. Kaliski Jr., editor. *Advances in Cryptology—CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*. Springer-Verlag, 17–21 August 1997.

[27] Paul Kocher. Differential power analysis. *http://www.cryptography.com/dpa/index.html*, 1998.

[28] Arjen K. Lenstra. Factoring. In Gerard Tel and Paul M. B. Vitányi, editors, *Distributed Algorithms, 8th International Workshop, WDAG '94*, volume 857 of *Lecture Notes in Computer Science*, pages 28–38, Terschelling, The Netherlands, 29 September–1 October 1994. Springer.

[29] Douglas L. Long and Avi Wigderson. The discrete logarithm hides $O(\log n)$ bits. *SIAM Journal on Computing*, 17(2):363–372, April 1988.

[30] M. Malkin, T. Wu, and D. Boneh. Experimenting with shared rsa key generation. In *preprint*, 1998.

[31] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[32] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 51–59, Montreal, Quebec, Canada, 19–21 August 1991.

[33] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology—CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1992, 11–15 August 1991.

[34] T. Rabin. A simpified appraoch to threshold and proactive rsa. In *Advances in Cryptology—CRYPTO '98*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[35] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key crypto-systems. In *Communications of the ACM*, pages 120–126, 1978.

[36] Safekeyper. See http://www.bbn.com.

[37] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely (extended summary). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 522–533, Montréal, Québec, Canada, 23–25 May 1994.

[38] A. Shamir. How to share a secret. In *Communications of the ACM*, pages 612–613, 1979.