MPSS: Mobile Proactive Secret Sharing

DAVID SCHULTZ and BARBARA LISKOV
Massachusetts Institute of Technology
and
MOSES LISKOV
The MITRE Corporation

This article describes MPSS, a new way to do proactive secret sharing. MPSS provides *mobility*: The group of nodes holding the shares of the secret can change at each resharing, which is essential in a long-lived system. MPSS additionally allows the number of tolerated faulty shareholders to change when the secret is moved so that the system can tolerate more (or fewer) corruptions; this allows reconfiguration on-the-fly to accommodate changes in the environment.

MPSS includes an efficient protocol that is intended to be used in practice. The protocol is optimized for the common case of no or few failures, but degradation when there are more failures is modest. MPSS contains a step in which nodes accuse proposals made by other nodes; we show a novel way to handle these accusations when their verity cannot be known. We also present a way to produce accusations that can be verified without releasing keys of other nodes; verifiable accusations improve the performance of MPSS, and are a useful primitive independent of MPSS.

Categories and Subject Descriptors: C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms: Security

ACM Reference Format:
Schultz, D., Liskov, B., and Liskov, M. 2010. MPSS: Mobile proactive secret sharing. ACM Trans. Inf. Syst. Secur. 13, 4, Article 34 (December 2010), 32 pages. DOI = 10.1145/1880022.1880028. http://doi.acm.org/10.1145/1880022.1880028.

1. INTRODUCTION

Malicious attacks are an increasing problem in distributed systems. If a node holds an important secret, that secret could be exposed by an attack in which an intruder gains control of that machine. An example of such a secret is

This research is supported by NSF ITR grant CNS-0428107.
Authors' addresses: D. Schultz (corresponding author) and B. Liskov, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139-4307; email: das@csail.mit.edu; M. Liskov, The Mitre Corporation, 202 Burlington Road, Bedford, MA 01730-1420.

the private key used by a certificate authority (such as VeriSign) to sign its certificates.

A secret can be protected by secret sharing. Secret sharing schemes [Blakley 1979; Shamir 1979] allow a collection of servers to possess *shares* of a secret value, such that any $t + 1$ servers can collaborate to perform operations using the secret, but any t or fewer servers can learn nothing about the secret.

In a long-lived system, however, nodes can become compromised over time, giving an adversary the opportunity to collect more than t shares and recover the secret. Proactive Secret Sharing (PSS) schemes [Cachin et al. 2002; Frankel et al. 1997; Herzberg et al. 1995; Ostrovsky and Yung 1991; Rabin 1998; Zhou et al. 2005] address this problem by using a share regeneration protocol, in which a new set of shares of the same secret is generated and the old shares are discarded, rendering useless any collection of t or fewer old shares the adversary may have learned.

Some PSS schemes reshare the secret among members of the same group. This approach is not sufficient in practice: it assumes a world in which servers that fail are later recovered, but recovery of compromised nodes is problematic. The problem is that resharing schemes use ordinary secret keys to implement secure channels, which are needed to reshare the secret. If the attacker learns these keys or corrupts them while the node is compromised, we cannot use them to communicate securely in the future.

This article describes a new resharing protocol called MPSS (for *Mobile Proactive Secret Sharing*) that addresses this problem by allowing the membership of the group to change. The servers that hold the secret shares periodically execute a handoff protocol that produces a new set of shares for a different (and possibly disjoint) set of servers. We present a novel way to accomplish this transfer that works in an asynchronous network. MPSS protects the secret even when up to t servers in the old group and t servers in the new group are faulty. The number of shareholders in MPSS is $n = 3t + 1$, which is optimal for an asynchronous protocol [Bracha and Toueg 1985].

Additionally, MPSS allows the threshold t to change. This is desirable because the threshold has a meaning: it represents an assumption about how easily nodes can be corrupted. If current events (e.g., a newly discovered vulnerability in Windows) dictate a reevaluation of this assumption, it is better to change the threshold than to start over. To our knowledge ours is the first scheme for changing the threshold that works even when there is up to the threshold number of failures in both the old and new groups.

The MPSS protocol is intended to be efficient for values of t that might occur in practice; analysis shows that the expected range is 1–10 (see Section 3.4). The protocol is designed to minimize latency and bandwidth utilization for the normal case, when fewer than t failures occur, for example, 0 or 1. However, even when there are as many as t failures, performance degradation isn't severe.

Furthermore, our protocol uses an interesting technique for efficient handling of "accusation" messages that may either be true (if sent by an honest node) or false (if sent by a liar), when there is no way to tell which of the accusing and accused nodes is faulty. We also describe an alternative approach in

which the validity of such messages can be evaluated; this approach requires the use of identity-based, forward-secure encryption as discussed further in Section 5, but allows a more efficient protocol.

The main contributions of this article are the following: (1) the introduction of an efficient, asynchronous protocol to reshare a secret to a new group of shareholders; (2) extensions to that protocol to support changing the threshold; and (3) an efficient way of handling verifiable accusations, which are useful for building Byzantine-fault-tolerant systems. In the course of describing the protocol, we show how techniques from several earlier works [Canetti et al. 2003; Castro and Liskov 2002; Herzberg et al. 1995] can be combined in novel ways to achieve our security and performance goals.

The article is organized as follows. We begin by discussing related work. Section 3 describes the network model and assumptions, and also discusses pragmatic issues such as determining how frequently to reshare the secret. Sections 4 and 5 describe our resharing scheme and the MPSS protocol, respectively, and Section 6 explains how to change the threshold. We present a discussion of performance in Section 7 and conclude in Section 8. The appendices contain additional material, including informal correctness arguments.

2. RELATED WORK

Secret sharing was first proposed by Shamir [1979] and Blakley [1979]. Subsequent work by Feldman [1987] and Pedersen [1991] extended Shamir's scheme to allow shareholders to determine whether the dealer sent them valid shares, hence allowing them to come to a consensus regarding whether the secret was shared successfully.

Proactive secret sharing schemes [Frankel et al. 1997; Herzberg et al. 1995, 1997; Ostrovsky and Yung 1991; Rabin 1998; Zhou et al. 2005] attempt to address the problem that shares learned by the adversary are compromised forever by resharing the secret periodically among the same nodes. Proactive secret sharing was first proposed by Ostrovsky and Yung [1991] and shown to be practical in a synchronous network model by Herzberg et al. [1995].

Desmedt and Jajodia [1997] were the first to study redistribution of a secret to a new group. Their work assumes a synchronous network model and is limited because shares are not verifiable; a faulty node in the old group may cause the receiving group to get bad shares. Wong et al. [2002] extend the scheme of Desmedt and Jajodia to be robust against corrupted nodes in the old group and provide verifiable shares. However, they require that all nodes in the new group are nonfaulty, and their protocol takes exponentially many attempts to complete in the worst case.

The protocol of Cachin et al. [2002] is the first efficient PSS scheme designed for the asynchronous model. Whereas the Herzberg scheme computes each new share as a function of a *corresponding* old share, the Cachin scheme is based on *resharing* the shares of the secret and combining the resulting subshares to form new shares of the secret. This scheme may allow the group membership to change, but this issue is not addressed in Cachin et al. [2002]. Also, they rely upon a collection of subprotocols, including a robust threshold signature

scheme and a threshold coin toss protocol which are themselves challenging to implement efficiently.

The scheme of Zhou et al. [2005] is intended to allow the group to move. However, since their construction produces secret shares that are exponentially large in n, the number of shareholders, the communication required to refresh the secret is exponential. The approach works well for small t, for example, for $t = 1$ or $t = 2$. For example, the communication overhead of an implementation of this scheme [Chen 2004] is 47KB for $t = 1$, 3.4MB for $t = 2$, and 220MB for $t = 3$.

Our approach builds on the work of Herzberg et al. [1995], but they assume the group membership is fixed and we allow it to change. Also, we assume a much weaker (asynchronous) network. Our protocol makes use of accusations as part of choosing the new shares, as does Herzberg et al. However, their approach assumes that lack of response implies the sender is faulty; this assumption doesn't work in an asynchronous setting. We show how to overcome this difficulty, both with and without verifiable accusations, and still ensure that honest servers can eventually get their shares.

Various generalizations of secret sharing have been proposed, for example, assigning differing weights to shareholders if some are presumed to be more trustworthy than others [Ito et al. 1987; Shamir 1979], or using separate thresholds for liveness and safety [Cachin et al. 2002]. Although many of the proposed techniques are applicable to MPSS, we do not discuss them here in order to clearly present the features that we believe are most useful in practice.

3. MODEL AND ASSUMPTIONS

In this section we discuss how we model the network and the power of the adversary, as well as the cryptographic assumptions our protocols require. Briefly, we assume that the network is asynchronous and under the control of the adversary. The adversary may adaptively corrupt a limited number of nodes over a period of time, thereby learning all of their stored information and causing them to behave arbitrarily badly.

We assume a system in which each node (or server) has a public signing key and also a public encryption key; the secret keys associated with these public keys are known only to the node. Each node also has a unique nonzero identifier. The correspondence between node ids and public keys is known by all nodes, for example, a node's id might is a cryptographic hash of its public keys; this precludes Byzantine nodes from successfully forging node ids. This information is propagated by some mechanism independent of MPSS, for example, by a Byzantine-fault-tolerant membership service [Cowling et al. 2009; Rodrigues et al. 2007].

3.1 Epochs and Limitation of Corruptions

System execution consists of a sequence of *epochs*, each of which has an associated epoch number. In any given epoch e, a particular group of n nodes in

the system is responsible for holding the shares of the secret. At the end of the epoch, the secret is transferred from these nodes ("the old group") to a new set of nodes ("the new group") and the system moves to the next epoch.

The adversary is assumed to be active and adaptive: it may decide to corrupt nodes at any point. We assume that the adversary can corrupt no more than a threshold t of the nodes in a group holding the secret in epoch e before the end of that epoch. Additional nodes may be corrupted after they have left the epoch. Because absolute clocks are not compatible with an asynchronous network, epochs are defined locally rather than as a global time period; the details are in Section 5.1.

When a node becomes corrupted, its internal state is revealed to the adversary and it can behave arbitrarily: it is no longer constrained to follow the protocol. We assume that corrupted nodes remain corrupted forever. This is reasonable because once the adversary knows a node's secret keys, it would be unsafe to consider that node recovered without changing its keys, in which case it would effectively be a different node.

3.2 Network Assumptions

Nodes communicate over a network by sending messages to one another; we assume messages are point-to-point: there is no broadcast channel. The network is asynchronous and unreliable: messages may be lost or delivered out of order, and messages may appear that were not sent by any party in the system. This model reflects the realities of many wide-scale networks such as the Internet. To capture the worst possible network behavior, we assume an intelligent adversary controls all communication: it receives all messages, and determines what messages are to be delivered, and in what order.

For termination, but not safety, we require a bound on message delivery. In particular, we assume *strong eventual delivery*.

—*Strong eventual delivery*. Assuming a sender keeps retransmitting a message, that message will eventually be delivered with a maximum delay bounded by an unknown variable that does not increase exponentially indefinitely.

This delivery assumption is the one used by Castro and Liskov [2002]. Any proactive secret sharing protocol needs to assume bounded eventual delivery in some form, to guarantee that epochs fit within a bounded time period.

3.3 Cryptographic Assumptions

The resharing protocol requires secure communication channels which we implement using cryptography. All messages are signed by the sender using its secret signing key and can be verified by the recipient using the associated public key. In addition, the content of messages is often encrypted so that only the target can read it.

The adversary can record all messages and examine them later. Furthermore, our constraint on the adversary doesn't limit its behavior once a node has left the epoch in which it was a shareholder. Therefore the adversary is

free to attack a node after this point and if it can recover its decryption key, it will be able to decrypt the old messages and thus learn secret information.

We prevent this attack by using forward-secure encryption [Canetti et al. 2003]. This technique assigns each node a single public encryption key, but the private key changes every epoch. Encryption is done using both the public key and the epoch number, and such a message can be decrypted only by using the secret key for that epoch. When a node leaves an epoch it advances its secret key to the one used for the next epoch, and deletes all information about the previous private key. After this point it will no longer be able to decrypt messages sent to it in the previous epoch.

All proactive secret sharing schemes require some way of changing the encryption keys. This could be achieved by having nodes choose new encryption keys (and delete their previous key) in each epoch, but this requires that nodes in the old group know the new encryption keys for all new nodes. The question that arises is: how do nodes know all these keys? Note that we cannot rely on new nodes telling old nodes their keys, because some of these communications may not happen, for example, if the new node is already corrupted. Forward-secure encryption is an elegant way to ensure that old nodes know how to send messages to new nodes, without needing to learn their new encryption keys.

3.4 Pragmatics

MPSS ensures that the adversary is unable to learn the secret if more than t shares are needed to do this. However, there is also the question of how to set up a system so that the constraint on the adversary works in practice.

There are several parameters that impact the security of the system, including: (1) the duration of epochs d; (2) the threshold t for each epoch; (3) the membership of the group in each epoch; (4) and the membership of nodes in the system, from which each group is chosen. We assume an administrator has configured the system with appropriate values for t and d, given the security and cost goals and the reliability of nodes in the system, but choosing these values is more art than science. Rodrigues et al. [2007] give an analysis of these parameters that provides some guidance. Their results indicate that values of $t > 10$ aren't needed, and that even if nodes are quite reliable, $t > 2$ seems to be needed. For example, suppose servers fail independently, and the probability that any given server is or becomes faulty during an epoch is 2%. With $t = 2$, there is a 23% chance that there will be more than t faults in a group within the first 1000 epochs. However, the probability of failure drops exponentially as t increases. If we increase t to 6, for example, the chance of more than t failures in any of the first 1000 epochs drops to 5×10^{-5}, which is plausible. If $d = 24$ hours, this corresponds to a system that survives for about 3 years; if the probability of failure is reduced to 1%, the system can survive for 20 years.

Practitioners must also be concerned with a number of other issues. For instance, the selected epoch duration must take into account the fact that the adversary may lengthen epochs through denial-of-service attacks that make it impossible for $2t + 1$ nonfaulty nodes to communicate. (The duration of such wide-scale attacks must be limited for any proactive protocol to be effective;

fortunately, historical evidence supports the reasonableness of this assumption.) These deployment considerations apply to many proactive schemes, not just MPSS, and a thorough treatment of the topic is beyond the scope of this article.

An important point is that once the parameters are set and the system has been deployed, the system can perform resharings autonomously. All the administrator must do to maintain the system is to add new nodes to replace ones that may have been corrupted. Since it isn't possible to tell that a node is Byzantine faulty, we recommend a methodology in which nodes in the system are periodically *refreshed*; their secret keys are changed and their code is reinstalled from a trusted disk image, after which they rejoin the system as a new node. The old group can collectively choose members of the new group with a preference for recently added nodes, since these are less likely to have been corrupted already. Each group member in epoch e can independently initiate the resharing protocol after the intended epoch duration d has passed since the start of epoch e, according to its local clock. No clock synchronization assumptions are required, except that the skew in nonfaulty nodes' clock rates must be small enough that the error in d is acceptable.

If the environment changes (e.g., new attacks, new defenses, or changes in security or cost goals), the administrator can instruct the system to change parameters such as t and d. Changing the threshold is discussed in Section 6. The fact that MPSS allows the threshold to change distinguishes it from several earlier schemes, for example, Herzberg et al. [1995] and Cachin et al. [2002].

4. THE RESHARING TECHNIQUE

In this section we describe our resharing technique. We begin by describing earlier work on which our approach is based. We refer to the secret as s and the threshold as t.

4.1 Preliminaries

Shamir's secret sharing scheme. In Shamir's secret sharing scheme [Shamir 1979], the secret is $s = P(0)$, where P is a polynomial of degree t with random coefficients (except for the constant term), computed over a finite field. Each node in the group holding shares knows $P(i)$, where i is the node's unique, nonzero identifier.

With $t + 1$ points on P, we can interpolate to learn P and thus learn $P(0) = s$. However, with only t points, there is still a degree of freedom left; therefore, the secret may be any value, and it is independent of any t points on P at inputs other than 0.

Verifiable secret sharing. In a secret sharing scheme, players must trust that shares they receive are correct. In a *verifiable* secret sharing scheme, additional information is given that allows each player to check whether or not its

share is correct. Each message that must be checked contains additional information in the form of *commitments*. Commitments are sent in the clear and can be used by the recipient to determine whether the rest of the information in the message has the appropriate properties; recipients use them to check that the polynomial used as the basis for the sent information equals zero at some predetermined point and that a point sent to it is on the polynomial. In Appendix A we describe how we use Feldman's VSS scheme [1987]; however, we could just as easily use a different VSS such as Pedersen's scheme [1991].

Herzberg et al.'s proactive secret sharing. Our method for generating the new shares is based on the proactive secret sharing scheme of Herzberg et al. [1995]. Herzberg et al's scheme includes two protocols: a *share renewal protocol* which allows nodes that already know their share to produce a new, independent share, and a *share recovery protocol* which allows a node that has lost its share to learn it.

In Herzberg et al. [1995], share renewal is done by choosing a random polynomial Q such that $Q(0) = 0$; node i's new share is then $P'(i) = P(i) + Q(i)$. This is still a valid sharing of the same secret since $P + Q$ is a random polynomial with constant coefficient s.

The polynomial Q is selected by having each node i produce a random polynomial Q_i such that $Q_i(0) = 0$. Node i then sends a point $Q_i(k)$ to each node k in the group, along with a commitment so that recipients can check that the constraint on Q_i holds. Then the nodes agree on a subset of the Q_i polynomials such that each polynomial in the subset is valid for at least $t + 1$ honest nodes; the polynomial Q is the sum of the polynomials in the subset. Finally, each node i at which the subset is valid computes its new share to be its old share s_i added together with the points it received for the each Q_i in the subset. The details of the protocol ensure that t corrupted nodes cannot learn the difference polynomial Q.

The share recovery protocol is used to recover shares for nodes that have lost their share or where the last resharing didn't work because a proposal in the selected set was invalid for them. The share recovery protocol for node i is done by choosing a random polynomial R_i such that $R_i(i) = 0$; again the protocol prevents corrupted nodes from learning the selected polynomial. Each other node j sends $P(j) + R_i(j)$ to node i, which reconstructs the polynomial $P + R_i$ and evaluates it at i to obtain its share.

4.2 Our Approach

We want to generate new shares for the same secret and move the new shares to a new group of nodes which may be completely disjoint from the old ones. Since the attacker can corrupt up to t nodes in a group, in this system it can control $2t$ nodes between the two groups.

Our approach is based on the Herzberg et al.'s resharing scheme. However, an important point is that if that scheme is unmodified, it is insecure when applied to secret redistribution to a new group. This is because each member of the old group computes a share for the corresponding member of the new group; if t nodes in the old group are corrupted, and a *different* t nodes in the

new group are corrupted, the adversary learns $2t$ of the new shares. We call this a *collusion attack*. This problem does not arise in the setting described by Herzberg et al. because the group membership does not change; the old and new groups are always the same servers.

Our solution to the collusion problem is to combine the resharing and share recovery into a single step. Instead of computing $P(k) + Q(k)$ in the old group and sending it to new node k, the old nodes additionally select, for each new node k, a polynomial R_k such that $R_k(k) = 0$, and each old node i sends $P(i) + \big(Q(i) + R_k(i)\big)$ to k. Upon receiving at least $t+1$ such points, k can interpolate to obtain the polynomial $P + Q + R_k$, then evaluate this polynomial at k to obtain its share of the original secret.

$$P(k) + Q(k) + R_k(k) = P(k) + Q(k) = P'(k)$$

Since R_k is random everywhere except at k, this polynomial provides the new node k no additional knowledge except $P'(k)$.[1] Furthermore, old nodes learn nothing about the new share $P'(k)$ because each old node only knows a single point on any given polynomial $P + Q + R_k$, and this polynomial is random and independent of P' except at k.

The difficulty here is in generating these polynomials Q and R_k in the old group so that no node knows too much about them; in particular, each old node i should learn only $Q(i) + R_k(i)$ for all k. If a node were capable of learning additional points, an adversary could accumulate $t+1$ points and interpolate $Q + R_k$. Then a faulty old node i could learn share $P'(i)$ intended for the new group. Furthermore, nodes must not be able to learn points on Q individually, because t collaborating faulty old shareholders could take their t points plus $Q(0)$, which is known to be zero, and thus interpolate Q. They could then add $Q(i)$ to their old shares $P(i)$ to obtain points on P' which are only supposed to be known only to new shareholders.

Our approach works as follows. Each old node i produces a random polynomial Q_i and also a random polynomial $R_{i,k}$ for each node k in the new group. We require that $Q_i(0) = 0$ and also that $R_{i,k}(k) = 0$ for each k. Node i sends to each old node j a proposal containing point $Q_i(j) + R_{i,k}(j)$, for each $R_{i,k}$; these proposals also contain commitments so that recipients can check their validity.

Old nodes decide on a subset of proposals to use for the resharing. Then they add up the points they received for each of the selected proposals to arrive at the values to send to the new node. The value sent from old node i to the new node k is i's old secret share plus the sum of points the old node received for $Q_l + R_{l,k}$, for each selected proposal l. Thus k receives i's point on the polynomial $P + Q + R_k$. When k receives $t+1$ of these points, it can interpolate to determine its share of the new polynomial $P' = P + Q$.

[1]This assumes that $i \neq k$, or else we have $P'(i) = P(i) + Q(i)$, which is the property of the Herzberg scheme we wish to avoid. We ensure that this is true by mandating that each node's identifier must be unique.

5. THE RESHARING PROTOCOL

This section describes the MPSS protocol for moving the secret to a new group without changing the threshold. Section 6 describes how to change the threshold.

Our system uses BFT [Castro and Liskov 2002] to carry out agreement in the old group. As in BFT, at any moment one of the group members is the *primary*, which directs the activities of the group. The other nodes watch the primary, however, and if it is not behaving properly, for example, not acting on requests when there is work to do, they carry out a view change protocol to select a different node as primary. Nodes are chosen to be the primary in subsequent views round-robin, so that the attacker is unable to control this choice. Details, such as how timeouts are chosen to ensure progress, can be found in Castro and Liskov [2002]. We require, as BFT does, that the size of the group is $n \geq 3t + 1$, which is optimal in an asynchronous network [Bracha and Toueg 1985].

The protocol has two stages. In the first stage, old nodes select polynomials Q and R_k that will be used for resharing the secret. Each old node i generates polynomials Q_i and $R_{i,k}$ and sends proposals describing them to other old nodes; the proposals contain commitments so that nodes can evaluate their validity. The old nodes then agree on a set of proposals that at least $t+1$ honest old nodes consider valid; these proposals define the selected polynomials Q and R_k.

In the second stage, each old node j that knows its share computes a point $P(j) + Q(j) + R_k(j)$ for each new node k. It sends this information to the new nodes, which use it to compute their shares.

As mentioned earlier, all messages are signed. Honest recipients ignore messages that aren't properly signed by the sender or have the wrong format. Additionally messages that contain secret information are encrypted so that only the recipient can see their contents.

The protocol works as follows.

(1) *Old shareholders generate proposals.*
   (a) Each server i in the old group generates random values $q_{i,1}, \ldots, q_{i,t}$ to generate a random degree t polynomial $Q_i(x) = q_{i,1}x + \ldots + q_{i,t}x^t$, where $Q_i(0) = 0$.
   (b) For each k in the new group, server i generates random values $r_{i,k,1}, \ldots, r_{i,k,t}$ to generate n polynomials $R_{i,k}(x) = r_{i,k,1}x + r_{i,k,2}x^2 + \ldots + r_{i,k,t}x^t$, such that $R_{i,k}(k) = 0$.
   (c) Server i computes commitments to the Q_i and $R_{i,k}$ polynomials it generated.
   (d) Server i, for each j in the old group, computes the vector $V_{i,j}$, which consists of $3t + 1$ points $Q_i(j) + R_{i,k}(j)$, one for each $R_{i,k}$ polynomial.
   (e) Server i generates a *proposal*, which consists of all the commitments, along with all the $V_{i,j}$ vectors. Each vector $V_{i,j}$ is encrypted with j's public encryption key; thus only node j will be able to see the points intended for it. Server i signs this proposal and sends it to all other members of the old group.

(2) *The primary collects proposals and sends out a set of them.* The primary waits until it receives $2t+1$ proposals. Once it does, it sends all members of

the old group a message containing cryptographic hashes of each proposal it received.

(3) *Parties check the proposals and accept or reject them.*
   (a) When node j receives the primary's list of hashes, it checks each proposal on the list for validity; if j is missing a proposal on the list (because it did not receive that proposal directly from the sender) it requests the proposal from the sender or the primary. A proposal from i is valid at j if the values in $V_{i,j}$ are consistent with the commitments, and the commitments prove that $Q_i(0) = 0$ and $\forall k. \quad R_{i,k}(k) = 0$.
   (b) If node j does not have its secret share it recovers it by requesting earlier resharing information that it did not receive (see step 11 that follows).
   (c) Once it has its secret share, server j responds to the primary, listing proposals found to be invalid; we say it *accuses* these proposals.

(4) *The primary determines the final set of proposals to use.* The primary processes responses as follows. Initially, the *current-set* of proposals contains the $2t+1$ proposals it selected, d is 0, and the *conflict-list* and *accept-list* are empty.
   (a) If the response is from a node on the conflict-list, it is ignored.
   (b) Else, if the response contains no accusations against proposals in the current-set, the party sending it is considered to be *satisfied* and is added to the accept-list.
   (c) Else, the primary chooses an accused proposal that is in the current-set deterministically (e.g., the one from the lowest numbered accused sender), removes it and the responder's proposal (if any) from the current-set, and increments d. The node that sent the removed proposal and the responder are then both placed on the conflict-list and removed from the accept-list if they were on it.

   When the accept-list contains $2t + 1 - d$ members (including the primary), the proposal selection phase is complete and has led to the identification of a final proposal set S.

(5) *Agreement.* The primary creates a message listing the signed responses (in the order the primary processed them) and uses the BFT agreement protocol [Castro and Liskov 2002] to get the group to agree on this message.

(6) *Computing S.* After agreement is reached, each server j processes the agreed-to message as described before (in step 4) to determine the final proposal set S. If this execution doesn't lead to the step 4 termination condition, the primary is faulty; the group will carry out a view change, elect a new primary, and rerun the protocol.

(7) *Old shareholders send resharing messages to the new shareholders.*
   (a) Node j deletes any resharing messages it received in the previous resharing.
   (b) If node j has its secret share and the proposals in S, it decides whether it finds all proposals in S to be valid. If so, it uses the commitments to each Q_i and $R_{i,k}$ for each i in the final proposal set to compute

commitments to Q and each R_k. Also, for every k in the new group, server j computes $v_{j,k} = s_j + \sum_{i \in S} Q_i(j) + R_{i,k}(j)$, and encrypts it for k. Server j sends the final commitment information along with every encrypted $v_{j,k}$ to each node in the new group.

(c) Node j deletes its secret information.

(8) *New shareholders receive resharing messages.* Each node k in the new group checks that the point $v_{j,k}$ it received was generated correctly, that is, $v_{j,k} = P(j) + Q(j) + R_k(j)$, which it can verify using the public commitments to P, Q, and R_k. If the message is valid, k sends a *receipt* to the sender and retains the message so that it can propagate it to other new nodes.

(9) *Discarding resharing messages*. Each old node that sent resharing messages waits until it receives $t + 1$ receipts and then discards the resharing message it sent to the new group. Deleting the resharing message is safe because we know that at least one honest new node has the message, and it will propagate it to other new nodes.

(10) *New shareholders compute their shares*. Once node k in the new group receives $t + 1$ valid shares, all with the same commitment information, it uses them to interpolate the polynomial $P + Q + R_k$. Then k evaluates this polynomial at k to obtain its new secret share $(P + Q + R_k)(k) = (P + Q)(k)$.

(11) *Share recovery*. Each k in the new group that does not yet have its secret share requests resharing messages from both the old and new shareholders. Servers in the old group reply by sending their resharing message (if they have one). Servers in the new group reply by sending all the resharing messages they received from the old group.

Figure 1 illustrates the message flow for a simple run of the protocol with $t = 1$ and fail-stop failures in both the old and new groups. An important point is that most of the messages are small, except for the proposals and the share transfer messages that are sent to the next group; we discuss performance in Section 7. Figure 2 contains an example that demonstrates some of the subtleties.

5.1 Discussion

This section provides a precise definition of epochs and the constraint on the adversary. Then it discusses the correctness (secrecy, integrity, and termination) of the protocol, assuming the adversary constraint. More details can be found in Appendix C.

Epochs. In an asynchronous system, nodes will not enter and leave epochs at precisely the same time, so we require a careful treatment of the concept of an epoch. We begin by defining a local property.

—*Local Epochs*. A node is *in epoch e* if it has secret shares or secret keys associated with epoch e. It *leaves epoch e* when it wipes out all such information from its memory so that it can no longer be recovered and additionally discards its resharing message if it had one; at this point it *enters epoch e* + 1.

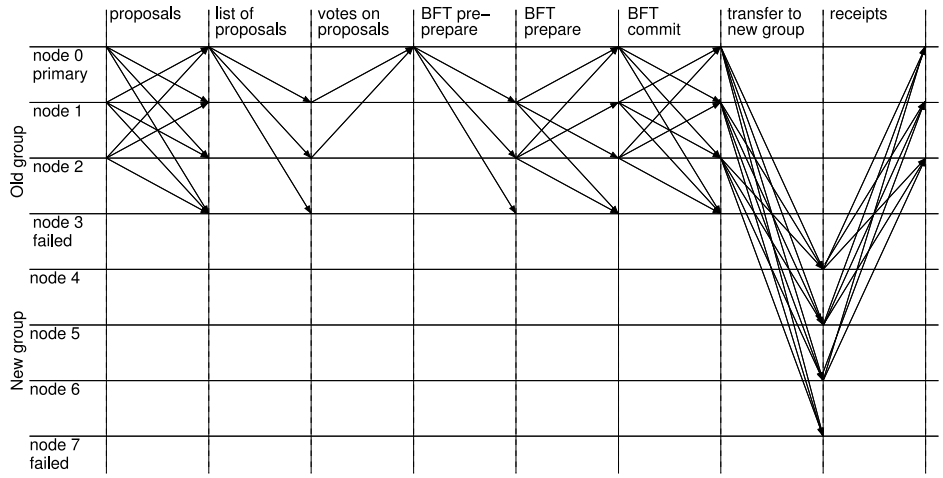

Fig. 1. The diagram illustrates an execution of the resharing protocol with $t = 1$ where node 3 in the old group and node 7 in the new group are faulty and do not send messages. After the nodes in the old group collect $t+1$ receipts, they discard the resharing messages they sent and stop participating in the protocol. If node 7 did not fail and was simply partitioned from the network, it can later use share recovery within the new group to obtain the share transfer messages.

We use G_e to identify the set of nodes responsible for the secret in epoch e. Nodes not in G_e advance to the next epoch by advancing their secret decryption key; nodes in G_e additionally discard their shares and transfer the new information to nodes in G_{e+1}.

Intuitively the adversary must be unable to corrupt more than t nodes in G_e while those nodes have information that would enable it to obtain the secret. One way to define this constraint is to begin with a global property.

—*System Epochs*. The system is in *system epoch e* from the moment the first honest node in G_e enters epoch e until the moment the last honest node in G_e leaves epoch e.

The adversary is constrained as follows.

—*System Compliance Constraint*. The adversary is *system compliant through epoch e* if, for all $i \leq e$, it is able to corrupt no more than t replicas belonging to G_i while the system is in any system epoch up to i.

This definition is sufficient for both safety and liveness. It is appealing for its simplicity, but it restricts the adversary more than necessary. We discuss a weaker constraint in Appendix C.4. Achieving security under the alternative definition requires modifying the resharing protocol to use forward secure signatures.

Secrecy. MPSS preserves secrecy because the Q and R_k polynomials used to generate the new share polynomial P' are random, and the adversary does not know them. Q and R_k are the sum of the proposals selected by the protocol.

(1) Each node in the old group sends all of its proposals to each other node in the old group. The proposals sent by $r1$ and $r2$ are good for $r3$ and $r4$ but bad for the other good nodes.
(2) The primary selects an initial proposal set containing the proposals of $r1 - r5$ and sends this information to the other nodes. Since there are at most t bad replicas, and the initial proposal set contains $2t+1$ proposals, at least $t+1$ proposals from good replicas are included in the initial set; in this case there are 3 good proposals, from $r3 - r5$.
(3) Nodes evaluate the proposal set. Since they can see only the points intended for them, honest nodes cannot tell whether the proposals are good for other nodes. Therefore $r3$ and $r4$ accept all proposals.
(4) The primary receives messages from $r1 - r4$, none of which contains any accusations, so all are placed on the accept-list. Since $2t+1-d$ is greater than the size of the accept-list, step 4 continues.
(5) The primary receives a message from $r5$ accusing both $r1$ and $r2$. It cannot remove both proposals from the current-set, since $r5$ might be a bad replica, but it is safe to remove one proposal: this way each bad replica can eliminate no more than one good proposal, so that at the end the final set will contain at least one good proposal. It removes $r1's$ and $r5's$ proposals from the current-set and removes $r1$ from the accept-list; also it places $r1$ and $r5$ on the conflict-list. Since $2t+1-d$ is greater than the size of the accept-list, step 4 continues.
(6) The primary receives a message from $r6$ accusing both $r1$ and $r2$. It removes $r2's$ proposal from the current-set and removes $r2$ from the accept-list; also it places $r2$ and $r6$ on the conflict-list. Since $2t + 1 - d$ is greater than the size of the accept-list, step 4 continues.
(7) The primary receives a message from $r7$ accusing $r1$ and $r2$. Since neither of these proposals is in the current-set, it places $r7$ on the accept-list and step 4 terminates. At this point the final proposal set contains proposals from $r3$ and $r4$, both of which are good.
(8) The primary runs the agreement protocol on the final proposal set.
(9) Nodes $r3 - r7$ compute the final proposal set by rerunning the primary's computation; since this computation is deterministic, they all arrive at the same final set and thus the same polynomials Q and R_k. Then they send resharing messages based on these polynomials and delete their secret information.
(10) Honest new nodes receive these messages, extrapolate to obtain their shares, and send receipts.
(11) Nodes $r3 - r7$ receive $t + 1$ receipts and discard their resharing messages.

Fig. 2. Example run of the protocol. We have $t = 2$ and a group of size 7. Replicas $r1$ and $r2$ are faulty. The example assumes that all honest nodes have their shares; if not they can obtain them (in step 3) by requesting resharing messages from one another and from members of the earlier group. The example shows that waiting for $2t + 1$ responses in step 4 of the protocol isn't sufficient. When the primary receives $r5's$ message, it can only remove one accused proposal (since $r5$ might be bad). If we completed step 4 at this point, we would have a proposal set containing $r2's$ proposal, so that the only honest nodes that can produce new points are $r3$ and $r4$, which isn't sufficient to allow resharing of the secret.

Proposals from honest nodes are random, so as long as this set contains at least one proposal from an honest old node, Q and R_k will also be random.

MPSS ensures that the set has at least one proposal from an honest node. The initial proposal set contains $2t + 1$ proposals, and therefore at least $t + 1$ of them are from honest nodes. In step 4, each dishonest replica can remove one honest proposal, either by accusing it, or because an honest node's accusation causes its honest proposal to be removed (along with the bad proposal). However, this eliminates at most t honest proposals, since there are at most t bad replicas. Therefore at least one honest proposal remains when step 4 terminates.

Note that nodes must discard their secret decryption keys as part of leaving an epoch. Otherwise, the adversary could record messages and decrypt them later, when it corrupts their recipients.

Integrity. The protocol also ensures that each new node that learns its share, learns a share of the same secret and these can be combined to form the secret. This follows because all honest old nodes compute the same proposal set in step 6, and therefore send points using the same polynomials Q and R_k. Also, new nodes construct their share only after receiving $t + 1$ valid resharing messages, all with identical commitments. Since at least one of these messages is from an honest old node, we can be sure that $P'(0) = P(0)$, where P is the polynomial used in the old group and P' is the polynomial used in the new group.

Termination. Termination requires that each step in the protocol terminates, assuming strong eventual delivery. The critical step here is step 4. First note that in step 4, the primary may need to receive more than $2t+1$ responses, even though there may be only $2f + 1$ honest nodes. However, if all honest replicas have replied, at least $2t + 1 - d$ replicas are on the accept-list. This follows because at least $2t + 1$ nodes are honest and d is an upper bound on the number of honest nodes on the conflict-list (since each time a pair of nodes is added to the conflict-list at least one of them is bad). Therefore if step 4 has not yet terminated, it is safe to wait for another response.

In addition step 4 ensures that the selected proposal set is good for at least $t + 1$ honest old nodes, and therefore honest new nodes will receive enough matching resharing messages so that they can compute their shares. Step 4 terminates when $2t+1-d$ replicas are on the accept-list. Therefore $2t+1-d$ is a lower bound on the number of good replicas that have accepted the proposal set; since $d \leq t$ there are at least $t + 1$ of them. Figure 2 shows that simply waiting for $2t + 1$ replies in step 4 isn't sufficient.

5.2 Verifiable Accusations

The protocol described earlier doesn't check accusations; it simply acts on them. The logic is that if one node accuses another, at least one of them is corrupt, so we remove both. In this section, we describe a scheme in which accusations can be verified. It uses more sophisticated cryptographic primitives (and stronger assumptions) to simplify the protocol.

Verifiable accusations require a way for nodes to check the accuser's claim that information sent to it was invalid. It doesn't work for the accuser to provide the decryption of the bad message, since it might lie. Instead, the accusation includes a key that enables other nodes to decrypt the bad message sent to the accuser, but in a way that does not reveal secret information sent by honest nodes to the accuser. We accomplish this by using *forward-secure identity-based encryption*, described in Figure 3.

Our use of identity-based encryption is atypical, so we first sketch how we use it without taking forward security into account. Each node j has a master key SK_j with corresponding public key PK_j. When i wants to send a message

A forward-secure identity-based encryption scheme is a collection of five algorithms:

—Generate() produces a public key PK and the corresponding master secret key for epoch 0, SK_0.

—Update(SK_e) takes the master secret key for epoch e and produces SK_{e+1}, the master secret key for epoch $e+1$.

—Extract(SK_e, id) takes the master secret key for epoch e and an identity id and produces an identity key $sk_{e,id}$.

—Encrypt(PK, e, id, m) takes an epoch e, identity id, and message m and produces ciphertext c.

—Decrypt($sk_{e,id}, c$) takes an identity key $sk_{e,id}$ and ciphertext c that was encrypted for epoch e and identity id and returns the original message m.

The security properties of this scheme follow immediately from those of forward-secure encryption and identity-based encryption, respectively. Note that this scheme differs from the forward-secure IBE scheme of Yao et al. [2004], wherein Update() can also operate on identity keys—a property that is undesirable for our application. We describe a way to implement identity-based forward-secure encryption in Appendix B.

Fig. 3. Forward-secure identity-based encryption.

to *j*, it encrypts it using PK_j and *i*'s *own* identity. Then *j* can run Extract() to derive the identity key needed to decrypt *i*'s message. However, if *i* sends a bogus message, *j* can reveal the identity key needed to decrypt the message without compromising the secrecy of messages sent to *j* by honest senders.

In the full verifiable accusation scheme, sender *i* encrypts secret message *m* being sent to node *j* as Encrypt(PK_j, e, i, m) where PK_j is *j*'s public encryption key and *e* is the epoch. Node *j* decrypts this message using a secret key based on the pair (e, i); thus the key depends both on the epoch *e* (as in forward-secure encryption) and on the identity of the sender (thus the encryption is identity based). To accuse node *i*, node *j* reveals the identity key for (e, i) in step 3 of the protocol. Other nodes can use this key to determine whether *j*'s accusation is valid, but releasing the key doesn't help the adversary decrypt messages from other senders to *j*, for either the current or later epochs.

Step 4 of the protocol is simpler when verifiable accusations are used. The primary checks whether accusations are valid, and if so removes all accused proposals from the current-set and adds the responder to the accept-list; otherwise, it adds the responder to the conflict-list.

Checked accusations ensure that only invalid proposals are discarded in step 4. Therefore it is sufficient to start with $t+1$ proposals; one of them must be from an honest node and it will not be discarded. Furthermore, honest nodes are always added to the accept-list: once we have $2t+1$ responses, at least $t+1$ of them will be from honest nodes, and they will accept all remaining proposals. Thus, the primary need only wait to receive $2t+1$ responses, rather than for the accept-list to reach a certain size.

6. CHANGING THE THRESHOLD

Our protocol allows the size of the group to change while the threshold t remains constant, but this is uninteresting: The best way to configure the system is to use a group of size $3t+1$, since a larger group cannot survive more failures, but presents more targets for the adversary to attack. Changing the threshold

is interesting, however, since this changes the number of failures the system can tolerate.

To reduce the threshold by some amount υ, we introduce υ "virtual servers," each of which holds a share publicly. The shares of the virtual servers are known at all real servers in the new group. This effectively reduces the threshold, since all nodes know the public shares, and more generally, any group of m nodes will know $m + \upsilon$ shares where υ is the number of virtual servers.

To increase the threshold, if there are more virtual servers than the amount of the increase, we remove that many virtual servers. Otherwise we increase the threshold by increasing the degree of the sharing polynomial.

In the following sections we first discuss how to decrease the threshold. Then we discuss techniques for increasing the threshold when the degree of the polynomial has to increase.

6.1 Reducing the Threshold

To reduce the threshold, we change the protocol as follows. In Step 2, the real servers generate polynomials with the same degree as the current share polynomial P. This polynomial has degree $t+\upsilon$ where υ is the number of virtual servers, and t is the current threshold.

A node does not make proposals on behalf of a virtual server, but does evaluate proposals on its behalf, and makes accusations against any proposal that doesn't pass both its own check and the virtual server's check. Virtual servers are not involved in agreement, and cannot serve as primary. Any real honest node that ends up on the accept-list will be able to do the resharing for its virtual servers as well as for itself and will send points on behalf of itself and the virtual servers to the new nodes; all honest real old nodes will send out identical points for each of the virtual servers. Since at least $t + 1$ honest real servers end up on the accept-list, new nodes will receive $t + \upsilon + 1$ points and therefore can recover their shares.

New nodes send receipts when they have $t + \upsilon + 1$ points (but they need to receive only $t + 1$ valid matching messages to do so). In step 9 old nodes delete the resharing messages when they have received $t' + 1$ receipts, where t' is the threshold in the new group. Since the adversary can never have more than t' corruptions in the new group, having $t' + 1$ receipts implies that at least one good node received the message, and that node will forward the shares so that all good nodes will eventually receive them.

Since virtual servers do not send proposals or transfer information to the new group, they are not involved in the most expensive parts of MPSS. Hence, reducing the threshold from t to t' in this way decreases the protocol overhead, even though the share polynomial still has degree t. The effective security of the resulting configuration is the same as if the system had initially been configured to tolerate t' faults.

6.2 Increasing the Threshold Using the Base Protocol

This section discusses how to increase the threshold when using the base protocol (the one without verifiable accusations). We accomplish this by first

resharing the secret to a larger group without increasing the degree of the polynomial; then the larger group increases the degree of the polynomial and does a second resharing. This approach allows us to overcome two problems that arise when using the base protocol.

First, in a group of exactly size $3t + 1$, we can only increase the threshold by 1. The reason for this limitation is that good nodes can eliminate only one proposal that is bad for them. Suppose the initial proposal set contains t bad proposals, and all bad proposals contain good information except for a specially targeted set of $t - 1$ good nodes. Suppose also that we receive acceptance messages from all good nodes. Even so, the final proposal set will contain one bad proposal, because each of the $t - 1$ targeted nodes will be able to remove just one proposal from the proposal set. Therefore the final proposal set will work only at the $t + 2$ good nodes that weren't targeted and as a result, new nodes can receive only this many points. Since new nodes need to receive $t' + 1$ points, where t' is the new threshold, the threshold can increase by at most one in a resharing.

Second, there is a termination problem: there is no point at which it is safe for old nodes to delete their secret information. Suppose we are increasing the threshold by 1. Then new nodes require $t+2$ resharing messages to reconstruct their shares, which implies that the proposal set must be valid for $t + 2$ honest old replicas. One possibility is that step 4 completes with a final proposal set that works for more than $t + 1$ honest nodes; then the resharing will complete. However, step 4 might complete with a final proposal set that works for exactly $t + 1$ honest nodes, which isn't enough for new nodes to construct their shares. In this case the resharing cannot complete. And there is no way for the primary to distinguish these two situations.

However, if the proposal set works for exactly $t + 1$ honest replicas, there is at least one additional honest replica whose acceptance message hasn't been processed in step 4. Therefore if we are willing to continue to run step 4 in parallel with the later steps of the protocol, we can arrive at a proposal set that works for enough honest old replicas. The idea is that if the primary receives an additional acceptance message that causes a reduction in the proposal set, it does a second round of agreement. After the second agreement we can be certain that the resharing will occur.

The problem is that old nodes don't know when to delete their shares. If they delete them when they send their resharing messages after the first run of the agreement protocol, this might be too early: they may need those shares to send resharing messages for the reduced final proposal set, in case the protocol runs a second time. However, the protocol might not run a second time, so waiting for this to happen also doesn't work.

In general, there is no way to determine whether one or two runs of the protocol will be needed. Thus it is impossible to know when old nodes can discard their secret information, because we cannot be sure whether the proposal selection steps need to run once or twice.

It may seem that old nodes can be informed by new nodes about whether the resharing worked. We could delay this decision point as much as possible: new nodes reply only when they have $t + 2$ resharing messages that work for them,

(1) The protocol runs a first agreement in which the proposal set contains proposals from $o1-o3$. These proposals have been agreed to by these three nodes, but $o1's$ proposal is bad for $o4$.
(2) $o1-o3$ send points to the new nodes based on this proposal set. $o1's$ points are good for $n1-n5$ but bad for $n6$ and $n7$.
(3) $n1-n5$ receive their points and interpolate to compute their shares. Then they send receipts to $o1-o3$.
(4) $o2$ and $o3$ receive $2t+2$ receipts and discard their secret information.
(5) The primary receives a message from $o4$ accusing $o1$. It adjusts the proposal set by removing $o1's$ proposal and reruns the agreement.
(6) $o4$ now sends points to the new nodes based on the final proposal set. However, $o2$ and $o3$ are unable to do this since they discarded their secret information.
(7) $n6$ and $n7$ request resharing messages from old and new nodes and nodes $n1-n5$ forward resharing messages they received previously. However $n6$ and $n7$ cannot interpolate to form their shares since $o1's$ message is bad for them, and $o4's$ message cannot be combined with those of the other old nodes since it concerns a different proposal set.

Fig. 4. Scenario illustrating the termination issue. There are four old nodes, $o1,\ldots,o4$; $o1$ is bad. There are seven new nodes, $n1,\ldots,n7$.

and old nodes wait for $2t+2$ such messages. But all this guarantees is that $t+2$ honest new nodes can reconstruct their shares; there could be another t honest new nodes that aren't able to do this. The reason is that some of the $t+2$ resharing messages that worked at an honest new node might be from bad old replicas and might not work for other honest new nodes: the fact that the new node has its share does not imply that all new nodes will be able to get their shares. This is true in our current protocol too, but in that case other honest old nodes will be sending messages that will work at the other new nodes: our protocol guarantees that enough such resharing messages will be sent. No such guarantee exists here. The problem is illustrated by the scenario in Figure 4.

Moving the secret to just $t+2$ new nodes allows the new nodes to use the secret, for example, to sign messages. But they are unable to transfer the secret in the next epoch, because this requires that *all* honest nodes know their shares. The situation could be resolved if there were a way for the other honest new nodes to recover their shares. This is possible with verifiable accusations as discussed in Section 6.3, but not with the base protocol.

We can overcome these problems by using an *indirect approach*: we first move the secret to an intermediate group, and then the intermediate group reshares the secret to the final group. The intermediate group is larger than the initial group, but has the same threshold t; the final group has threshold $t+c$. The size of the intermediate group is $3t+1+c$; the size of the final group is $3(t+c)+1$.

The resharing to the intermediate group works exactly as now. The main changes to the resharing from the intermediate group to the final group are as follows. First, in step 1 honest nodes produce polynomials of degree $t+c$. Second, step 4 completes when the accept-set is of size $2t+1+c-d$ (where d is the size of the conflict list, as before). This way we ensure that $t+c+1$ good old nodes are able to send resharing messages, and therefore the secret can be transferred. Additionally since the intermediate group is of size $3t+1+c$, it is safe to wait for this many messages in step 4. Finally, in step 9 honest old nodes wait to receive $t+c+1$ receipts before discarding their resharing messages.

6.3 Increasing the Threshold Using Verifiable Accusations

If we use verifiable accusations, we can reshare the secret directly from the old to the new group, without using an intermediate group. We are no longer limited to an increase of 1, because with verifiable accusations honest nodes can eliminate all proposals that are bad for them. However, we still have the problem of possibly needing to do additional iterations because earlier proposal sets didn't work for $t + c$ honest nodes.

The protocol executes in a series of iterations as discussed in Section 6.2, and for the same reason: in step 4 we may not heard from enough honest nodes to allow the secret to be transferred, and we compensate by running the protocol again when additional messages are received. It is no longer necessary to limit the threshold to grow by 1; instead it can grow by c. However, we require $c \leq t$; this is needed because new nodes require $t + c + 1$ resharing messages, and there may be only $2t + 1$ good nodes in the old group. The maximum number of iterations is $c + 1$.

The changes to the protocol are as follows (here $t' = t + c$).

—Run the resharing protocol with verifiable accusations. Each iteration has an iteration number and this number is sent in the resharing messages. Old nodes do not discard their secret information immediately upon generating shares for the new nodes; they delay as discussed shortly.

—The primary runs step 4 in parallel with the earlier iterations. If it receives a new message containing valid accusations of proposals currently in the proposal set, it adjusts the proposal set and starts the next iteration of agreement and resharing.

—When a new node has $t' + 1$ valid resharing messages all for the same iteration, it sends *success* messages to all old servers informing them about this.

—When an old node has $2t' + 1$ success messages all containing the same iteration number, it discards all its secret information for epoch e. Then it concatenates the success messages and sends them as a completion certificate to the new nodes.

—New nodes send receipts for these certificates (if valid) to the old nodes. When an old node has $t' + 1$ such receipts, it can delete its resharing messages and completion certificates.

—New nodes run BFT to agree that the resharing of a particular iteration, for which some of them have received completion certificates, is the one they will use to compute their shares.

The result is that new nodes will produce shares based on the same polynomial P' and old nodes will be able to stop working on the resharing.

However, there is a problem: as discussed in the previous section the aforesaid protocol only guarantees that $t' + 1$ new honest nodes get their shares. The agreed to certificate might be for resharing messages that work at only $t' + 1$ honest new nodes and are bad for the rest. In order to do the next resharing, we need a way for a node that is missing its share to recover it from other replicas. This is possible using verifiable accusations.

A node i that is missing its share runs a *recovery protocol*. This protocol mimics the first steps of the normal resharing protocol.

(1) Node i requests proposals from other nodes.
(2) Honest nodes create random polynomials such that $R(i) = 0$ and send points and verification information about them to i.
(3) Node i waits for $t' + 1$ valid proposals and sends out the proposal set.
(4) Honest nodes respond with acceptance messages containing accusations.
(5) Node i processes accusations and forms the final proposal set when it has received $2t' + 1$ responses. It sends the final proposal set to all nodes, along with the messages it received.
(6) Honest nodes then send points to i, encrypted for it, if they have their share and if the proposal set is good for them.
(7) If i receives $t' + 1$ valid resharing messages, the protocol is finished; it extrapolates using the information to recover its share.

Thus this protocol is like our resharing protocol except that it is run by i and requires no agreement. An additional point is that it may be necessary to continue accepting accusations in step 5 given before and then run steps 6 and 7 again, because the proposal set produced in step 5 might not be good for at least $t' + 1$ honest nodes that have their share. However, eventually i will hear from all of these nodes, and at that point the protocol will be able to complete.

The reason we need verifiable accusations for the recovery protocol is because the resharing protocol can ensure only that $t' + 1$ honest nodes have their share when resharing completes. Therefore the recovery protocol must end up with a proposal set that works for precisely these honest nodes. And this requires that an honest node be able to accuse and remove all proposals that are bad for it. Verifiable accusations allow an accuser to remove multiple proposals; without verifiable accusations this isn't possible.

6.4 Discussion

The indirect approach has the obvious advantage that it can be used with either resharing protocol; it doesn't require the use of verifiable accusations and identity-based forward-secure encryption. However, even when using the verifiable accusations, so that the direct approach is possible, we prefer the indirect scheme. First, it is more efficient than doing a direct resharing. It's true that the indirect scheme requires two runs of the protocol, but the direct protocol might require up to c iterations plus it needs an additional run of BFT (in the new group) and also the recovery protocol. Additionally, the indirect approach is simpler than the direct scheme, and imposes no limit on the amount by which the threshold can grow in a resharing.

Another point is that the drawbacks of the indirect scheme are more apparent than real. The indirect scheme doesn't actually require an additional group of replicas; instead the intermediate group can be a subset of the final group. The intermediate group is slightly more vulnerable to attack because it doesn't have the minimal size of $3t + 1$. However, it is in existence for a

very short time: just long enough to do the resharing. And note that in the direct scheme, old nodes need to keep their secret information for a longer time (through potentially several iterations of the protocol, until they receive the success messages); with the indirect scheme secrets are discarded in step 7 in both the initial and intermediate groups.

7. PERFORMANCE

This section discusses the performance of our system. Our protocol is designed to perform well in the optimistic case and degrade gracefully in the face of failures, so we consider two cases.

Normal case. In the normal case, there are no faults or a small constant number of faults. We expect this to be the common, and hence, the most important case.

Worst case, adaptive adversary. Here we assume an adaptive adversary that behaves arbitrarily badly and can corrupt nodes at any point during the execution of the protocol, up to a maximum of t corruptions.

7.1 Normal Case

In the normal case we would like to have both low bandwidth utilization and low latency to transfer the secret, and we achieve both of these goals.

Our latency is just 7 message delays from the time old nodes start the resharing protocol until new nodes receive their shares (our steps 1–3 and 7, plus the preprepare, prepare, and commit messages of BFT). The total number of messages sent is $4n^2 + 3n$, where n is the size of the group.

Most of the messages we send are small (either constant or $O(n)$ size). The exceptions are the messages sent in steps 1 and 7; these are of size $O(n^2)$. In these steps nodes need to send commitments for n polynomials; for each polynomial, information of size t is needed. The messages are sent to all n nodes, and therefore bandwidth utilization is $O(n^3)$.

Our asymptotic performance is as good as that of the scheme of Cachin et al. [2002], and better than that of the scheme of Zhou et al. [2005], which consumes an exponential amount of bandwidth. Having better asymptotic performance allows our system to work with a larger t than could be supported by the Zhou scheme. We discussed how to choose t in Section 3.4; our scheme is practical across the range of expected values of t. At the upper end of the range, doing a resharing requires hundreds of megabytes of communication, but this is still plausible for a protocol that is run infrequently.

7.2 Worst Case

We would like to ensure that even in the worst case, the protocol consumes a reasonable amount of bandwidth and completes in a reasonable amount of time.

In the pessimistic case up to t view changes may be required to compensate for dishonest primaries.[2] This cost can be avoided by running these rounds in parallel (as is done in the Zhou scheme), but that choice leads to more messages being exchanged in the optimistic case. We have chosen (as was done for BFT [Castro and Liskov 2002]) to avoid the extra costs in the normal case since this is the case that is likely to occur in practice.

Also, in the pessimistic case where there are t faults, in step 3 the (honest) primary may need to resend proposals and commitments that faulty nodes sent to it but not to other nodes, or that faulty nodes claim weren't sent to them. Thus, the primary may have to send t messages of size $O(t^2)$ to each of the n nodes, at a total cost of $O(t^4)$ bytes. However, we have developed a variation on the protocol that avoids this cost. Rather than asking the primary for missing proposals, nodes instead inform the primary of votes on proposals not in the proposal set in step 3. The primary uses these extra votes to evaluate other proposal sets, if the one it selected doesn't reach the termination condition by some timeout. Evaluating other proposals is NP-hard: the primary must consider all possible proposal sets simultaneously. However, the computation is tractable for $t \leq 10$ and thus works fine in expected deployments (e.g., for $t = 4$ or $t = 5$). This combined technique leads to low latency in the normal case, and moderate latency in the worst case; nodes may need to send up to t additional vote messages to the primary in the worst case, but these messages are small.

8. CONCLUSIONS

This article explains why mobility is important for a secret-sharing scheme in a long-lived system and presents MPSS, an efficient and practical scheme for mobile proactive secret sharing in an asynchronous network. In MPSS, the group of shareholders can change completely as the secret is reshared. MPSS also allows the threshold to change when the secret is reshared, allowing the system to be reconfigured on-the-fly to accommodate changes in the environment.

MPSS uses a novel algorithm for handling accusations in an asynchronous network when we cannot determine whether the accuser or accused node is lying. We also present verifiable accusations which improve the efficiency of MPSS, but require the use of identity-based forward-secure encryption. Both of these schemes for handling accusations appear to be useful primitives independent of MPSS.

In addition, MPSS uses an efficient resharing protocol intended to be used in practice; Section 7 discusses its performance. The protocol exchanges mostly small messages, with a few of size $O(n^2)$; it transfers the secret in 7 message delays in the normal case of no or a few failures.

[2]Additional view changes may be required if our timeouts are too short, but Castro and Liskov [2002] point out that spurious view changes are rare enough to be negligible.

APPENDICES

A. EXTENSION OF FELDMAN VSS

In verifiable secret sharing schemes, messages contain extra information in the form of *commitments* that allow recipients to check whether information sent to them is correct.

Here we describe Feldman's VSS scheme [Feldman 1987]. In Feldman's scheme, the secret is shared by generating a random degree t polynomial P such that $P(0)$ is the secret s. The verification information is computed using a cyclic group with generator g, which is a public system parameter. When P is generated, commitments are given for each coefficient: $c_0 = g^{p_0} = g^s$, and generally $c_i = g^{p_i}$. (Because g^s will be public, the actual secret must be included in only the hard-core bits of s. We refer the reader to Herzberg et al. [1995] and Feldman [1987] for a discussion of this issue.) Now each node can compute

$$g^{P(i)} = \prod_{j=0}^{t} c_j^{i^j}$$

and thus check if the share s_i it receives is equal to $P(i)$ by checking if $g^{s_i} = g^{P(i)}$.

We use Feldman scheme extended with some additional check equations. We do not cover the details of these checks here (see Schultz [2007]), but the basic principle is as follows. Since exponentiation is homomorphic, that is, $c = a+b \Leftrightarrow g^c = g^a g^b$, we can verify properties of polynomials we do not know by verifying properties of the commitments. For example, if $Z(x) = z_0 + z_1 x + \cdots + z_t x^t$ and we have Feldman commitments $g^{z_0}, g^{z_1}, \ldots, g^{z_t}$, we can check that $Z(3) = 0$ by verifying that $g^{z_0}(g^{z_1})^3 \cdots (g^{z_t})^{3^t} = g^0 = 1$.

B. FORWARD-SECURE IDENTITY-BASED ENCRYPTION

To allow for verifiable accusations, we use a forward-secure identity-based encryption scheme. The scheme of Yao et al. [2004] would suffice, but provides significantly more powerful security properties than we need, and is much more complex than forward-secure encryption. We prefer to use a straightforward modification of the Canetti et al. scheme [Canetti et al. 2003], which is the first known forward-secure encryption scheme in the literature. Our modification is along the lines of a solution proposed by Yao et al. [2004] but dismissed as not sufficient for their security goals. However it suffices for ours.

Our forward-secure IBE scheme is based on hierarchical identity-based encryption, specifically Binary Tree Encryption (BTE). In BTE, an identity is a sequence of bits: $b_1, \ldots, b_k$. The secret key for identity $b_1, \ldots, b_k$ can be used to derive the secret key for any identity that starts with $b_1, \ldots, b_k$. The public key can be used to encrypt a message for any identity, but only the secret key for that identity can be used to decrypt.

Canetti et al. [2003] describe how to adapt BTE into a forward-secure encryption scheme through the concept of *pebbling*. The epoch e is encoded as a k-bit identity for some fixed k. The scheme keeps track of enough state to derive the secret key for identity e and all identities greater than e, but no identities less than e.

This can easily be extended to allow for identity-based forward-secure encryption, which is parameterized on identity i and epoch e. We use a further level of BTE hierarchy for the identity. Since the key output by the Canetti et al. scheme for epoch e is an ordinary BTE key, we can derive from it the key for the identity $e|i$.

Once we have identity-based encryption, we can use the method described in Section 5.2 to implement provable accusations. Note that the specific secret key for a certain epoch with a certain identity is not hierarchically "above" any key that would be used in the future (because we include both i and e in ID): therefore, revealing it is safe.

There is one caveat here: the properties of identity-based encryption assume that the generator of the key is honest. However, this should not be a problem for our application; it is reasonable to assume that nodes cannot be corrupted before they generate their keys.

C. CORRECTNESS

In this appendix, we discuss what it means for MPSS to be correct, and we prove some of the more interesting claims. The proofs we present are informal and are focused on our protocol and our algorithm for resolving votes and selecting proposals; many of the cryptographic details are relegated to Schultz [2007]. Because our primary purpose here is to argue the correctness of the *protocol*, we do not use standard notions from cryptography such as computational indistinguishability. Strictly speaking, such formalism would be required for most theorems because verifiable secret sharing necessarily relies on computational assumptions.

The purpose of MPSS is to transfer a shared secret to a new group of shareholders, using a new sharing that is distinct from the old one. The old shareholders throw their shares away, so that an adversary that has corrupted up to t nodes in the old group learns nothing about the secret by corrupting additional nodes in the old group in the future. In what follows we show that MPSS is both safe and terminates. Safety requires that we preserve secrecy (the secret is not revealed) and integrity (the secret is not corrupted).

Our proofs depend on the definitions of epochs and the constaint on the adversary given in Section 5.1. We discuss a weaker constraint in Section C.4.

C.1 Secrecy

Informally, the secrecy property says that the adversary never learns the secret, given shares it knows from present and past epochs, and information exposed in the execution of the resharing protocol. Since both MPSS and the scheme of Herzberg et al. [1995] produce a new share polynomial of the form $P' = P + Q$ with $Q(0) = 0$, we do not repeat their secrecy argument here.

Our protocol produces the polynomial Q differently from the Herzberg et al. scheme, however, so we need to justify that the coefficients of $Q + R_k$ are random and independent of anything the adversary knows, even if some of the adversary's proposals are used to construct $Q + R_k$. Intuitively, this is true because the proposal selection algorithm used by the primary always chooses at

least one good proposal, which is random. Combining this proposal with the adversary's proposals via modular addition constitutes a one-time pad.

THEOREM S-1. *Any proposal set that honest nodes agree to in MPSS contains at least one proposal from an honest node.*

PROOF. In step 2, the primary collects $2t + 1$ proposals as an initial proposal set; since no more than t nodes are dishonest, at least $t + 1$ of these proposals come from honest replicas. In step 4, the primary processes accusations and removes accused proposals from the current-set. Each accusation causes the removal of at most one good proposal: a bad node can remove a good proposal by accusing it, and when a good node accuses a bad proposal, its own proposal might also be removed. In either case, both the accuser and the accused node are added to the conflict-list. Since at least one of them must be bad, d is a lower bound on how many bad nodes are in the conflict-list and also an upper bound on the number of good proposals that have been removed from the proposal set. Since $d \leq t$, at least one honest proposal remains. □

A caveat with the preceding secrecy argument is that the adversary must not be able to generate proposals *based on* the proposals from honest nodes. For example, if an honest node generates proposals and sends out $p_1, p_2, \ldots, p_{3t+1}$, with each point encrypted with the appropriate node's key, a bad node B could arrange for the final polynomial to be the zero polynomial by sending out $-p_1, -p_2, \ldots, -p_{3t+1}$ as its points. It could do this without decrypting the points if it knew that flipping a particular bit in the ciphertext resulted in the sign bit flipping in the plaintext. Therefore, we require an encryption scheme that is secure against adaptive chosen ciphertext attacks; it is well-known that this implies nonmalleability, and that therefore, the aforementioned attack will not work.

Furthermore, we must show that nodes in the old group do not learn shares of honest nodes in the new group (unless they are the same node). The Herzberg et al. scheme does not support mobility, that is, the old and new groups are identical, so this is not a problem for them. They must still cope with a mobile adversary, but in a more limited way. In their scheme, an adversary who controls t nodes can choose to uncorrupt a node in epoch e, then later corrupt a new node in epoch $e + 1$, whereas in MPSS, t nodes in the old group and t nodes in the new group are corrupted simultaneously.

THEOREM S-2. *If B is a bad node in the old group and A is an honest node in the new group, B doesn't learn anything about A's share, $P'(A) = P(A) + Q(A)$.*

PROOF. B knows its own share, $P(B)$, and the values of the $Q + R_A$ polynomial at the t or fewer points corresponding to the old nodes the adversary controls, including B. None of these points is $Q(A) + R_A(A) = Q(A)$, because A is an honest node in the new group and not a dishonest node in the old group. By Theorem 1, $Q + R_A$ is a random polynomial of degree t, so the adversary's t points are independent of $Q(A) + R_k(A)$, and hence independent of $P'(A)$. □

Remark. Note that the proposal selection algorithm guarantees that the final proposal set contains at least one proposal from an honest node, but in the worst case it could contain exactly one, for example, the one from C. In this case, C knows $Q + R_A$ and therefore knows the new shares. But since in this case C must be honest, it will simply throw this information away.

The previous theorems take for granted that the adversary is effectively restricted to the information we expect it to be able to learn: messages to corrupted parties, messages sent from corrupted parties, and the shares of corrupted parties. However, our assumptions about the adversary allow a great deal of freedom in the network, including the ability to read and alter *any* message between honest parties. We rely on forward-secure encryption and forward-secure signatures to protect the privacy and integrity of those messages. While this would seem to guarantee the property we need, we need an additional assumption: that selective decryption of messages leaks no unexpected information. This is an open problem in cryptography [Dwork et al. 2003], but is thought to be true, and in any case it is needed to guarantee Byzantine security in a real network. Prior papers in this area avoid this issue by assuming secure channels, but in a real application we must implement secure channels through encryption and signatures. This assumption is needed for another reason also: the public commitments to each party's share, combined with the adversary's ability to later selectively corrupt parties and learn their share, forms another selective decryption scenario. Since we must make this assumption in any case, it is therefore reasonable to use Feldman VSS rather than the more expensive but perfectly-hiding Pedersen VSS scheme [Pedersen 1991].

C.2 Integrity

Integrity requires that each honest node in the new group that computes its new share constructs a share based on the same polynomial P' such that $P'(0) = P(0)$. This ensures that the honest new nodes have shares of the same secret and that these shares can be combined together to recover the secret.

The fundamental reason integrity is preserved is because of the way we use commitments. When the old group agrees to a set of proposals to use, they are actually agreeing to a set of commitments to polynomials whose points are secret. However, each node can verify that the commitments match the points it knows, and that the commitments identify well-formed Q and R_k polynomials, that is, ones with zeros in the correct places. Hence, honest nodes only send consistent points on well-formed $P + Q + R_k$ polynomials to the new group. The new group, furthermore, only uses those points that match the agreed-upon commitments.

THEOREM I-1. *Each new node that computes its share constructs its share based on polynomial P' and furthermore this polynomial represents the same secret as in the old group, that is, $P(0) = P'(0) = s$.*

PROOF. Old node j will accept a proposal point $z_{i,j,k}$ from i only if the commitments prove that it is valid. In particular, i must commit to polynomials Q_i

and $R_{i,k}$ such that $Q_i(0) = 0$ and $R_{i,k}(k) = 0$, which j uses the commitments to verify. Furthermore, the commitments must prove that i sent j the right point, $z_{i,j,k} = Q_i(j) + R_{i,k}(j)$. The old group then agrees to a set of proposals, and if j accepts all the proposals in this set, it sums $Q(j) + R_k(j) = \sum_i Q_i(j) + R_{i,k}(j)$. Note that $Q(0) = 0$ and $R_k(k) = 0$. Then j sends $P(j) + Q(j) + R_k(j)$ to k in the new group. Each honest new node k that computes its share does so based on $t + 1$ resharing messages with matching commitments. Since at least one of these messages must have come from an honest old node, new node k obtains commitments to P, Q, and R_k. It uses the commitments to identify a set of $t+1$ valid points on $P + Q + R_k$, and interpolates to recover this polynomial. Then it computes $P(k) + Q(k) + R_k(k) = P(k) + Q(k) = P'(k)$. Note that $P - P' = Q$ and $Q(0) = 0$, so $P(0) = P'(0) = s$. □

C.3 Termination

To prove the liveness of MPSS, we argue that each of the three phases—proposal selection, agreement, and share transfer—terminate.

C.3.1 *Agreement.* The BFT agreement step terminates assuming the strong eventual delivery assumption of Section 3 holds. See Castro and Liskov [2002] for details.

View changes can occur when there are timeouts at the honest nodes, and in this case it is possible that the primary is actually honest, for example, it sends messages but they are lost. However, even in this case, eventually there will be an honest primary that is given enough time to complete its work; in our case to select the final proposal set and run the agreement. The details are in Castro and Liskov [2002].

C.3.2 *Proposal Selection (steps 1-4).* Termination of proposal selection is nonobvious because even though there may be no more than $2t+1$ honest nodes in the system, proposal selection may wait for *more than* $2t + 1$ votes. However, this only happens if inconsistencies in the votes received so far prove that some of the votes are from faulty nodes, so we are able to argue that the primary will only wait if there are nonfaulty nodes it has yet to hear from.

THEOREM T-1. *The proposal selection step terminates after processing at most* $2t + 1$ *responses from nonfaulty nodes.*

PROOF. When a response contains an accusation against the current set, either the accuser or the accusee is faulty, and we add both to the conflict-list and increment d. Hence d represents an upper bound on the number of nonfaulty servers in the conflict-list. At any point, each server whose response has been processed by the algorithm so far is on the accept-list or the conflict-list. Therefore, upon processing $2t + 1$ responses from honest servers, the size of the accept-list is at least $2t + 1 - d$, which is the termination condition for the algorithm. □

C.3.3 *Share Transfer.* Proving that share transfer terminates has three parts. First, we show that, given that honest nodes know their shares, at least

$t + 1$ honest nodes send share transfer messages to the new group. Next, we show that if at least $t + 1$ honest nodes send share transfer messages to the new group, then all the honest nodes in the *following* epoch can obtain their shares. Then by induction, every resharing is able to terminate as long as the honest nodes in epoch 0 have their shares. The following lemma proves the first property.

LEMMA T-2. *If* $2t+1$ *honest servers in epoch e that know their shares run the MPSS protocol, at least* $t+1$ *honest servers will send share transfer messages to the new group (i.e., the group for epoch* $e + 1$*).*

PROOF. Whenever the primary processes an accusation in step 4, either the accuser or the accusee is faulty, so the primary removes both nodes from the accept-list and increments d. Hence, at most $t - d$ faulty servers are on the accept-list. step 4 terminates when $2t + 1 - d$ nodes are on the accept-list, of which at least $t + 1$ must be honest. These $t + 1$ honest nodes are satisfied with the chosen proposal set and generate share transfer messages in step 7. □

Next, we argue that if $t+1$ honest nodes send share transfer messages to the new group, then the honest nodes in the new group will eventually get their shares. We cannot ensure that this is possible by the time MPSS terminates in the old group, because we want the old nodes to be able to discard their secret information as soon as possible. However, we can ensure that each honest node that doesn't know its share can ask its peers for the encrypted messages that allow it to compute its share.

LEMMA T-3. *If* $t + 1$ *honest nodes in epoch e send share transfer messages to the new group, then each honest node in epoch* $e + 1$ *will be able to obtain its secret share.*

PROOF. In step 9, an old node that sends share transfer messages waits for $t+1$ acknowledgements from $t + 1$ distinct nodes in the new group, at least one of which must be honest. Furthermore, the share transfer messages contain (encrypted) points for *all* the nodes in the new group, so any node k in the new group that doesn't have its share can request the points from its peers at any time in epoch $e + 1$. Honest peers reply by sending the points. Therefore node k will get the points from all the honest old nodes that sent them, either by direct communication with the old nodes, or by obtaining their resharing messages from other new nodes. Given $t+1$ points, k can interpolate to recover its secret share. □

Finally, we combine Theorem T-1 and Lemmas T-2 and T-3 to show that MPSS terminates.

THEOREM T-4. *When MPSS is run in any epoch e, the secret is transferred to the new group and each node enters local epoch* $e + 1$ *in a finite amount of time.*

PROOF. We assume that all honest nodes have their shares in epoch 0, when the system is first initialized. The proof is by induction on e. If $2t + 1$ honest nodes in epoch e know their shares, the primary will receive votes from

these honest nodes and therefore the proposal selection step will terminate by Theorem T-1. By Lemma T-2, at least $t + 1$ honest nodes will be able to use the agreed-upon proposal set. Thus, by Lemma T-3, each honest node in epoch $e + 1$ will be able to obtain its share, completing the induction. □

C.4 Alternative Definition for Epochs

In this section we describe a weaker constraint on the adversary.

—*Weak Compliance Constraint*. The adversary is *weakly compliant through epoch e* if, for every epoch $i \leq e$, it is able to corrupt no more than t replicas in G_i while they are in any epoch up to i.

Note that we do not require a global notion of epoch with this definition.

This definition constrains the adversary less than our original compliance constraint, because the adversary is allowed to corrupt an honest node in G_e after it leaves epoch e even if other honest nodes in the group are still in epoch e. Since weak compliance constrains the adversary less than system compliance, a system that works correctly under weak compliance also works under system compliance but not vice versa.

However, to implement a system that works under the weak compliance constraint, we need to use forward-secure signing [Krawczyk 2000]: when a node leaves an epoch, it advances its secret signing key so that if it is corrupted after that point it will not be possible for the adversary to impersonate it. Otherwise there is an attack that can be launched by the adversary. We call this attack the *isolation attack*. The attack takes advantage of the adversary being allowed to corrupt an honest node after it leaves epoch e!

The isolation attack works as follows. Suppose the adversary has corrupted t replicas in G_e, and additionally it isolates one of the group's good replicas, r, so that r is unable to receive its messages for a long time. During this period the other good replicas in G_e do a resharing and leave epoch e. This resharing is based on a proposal set that contained proposals from each of the bad replicas; all these proposals are known to the adversary and at least some of them are good for r.

The adversary already knows t points on the old polynomial P (since t of the old nodes were corrupt). Subsequently, the attacker corrupts one of the formerly honest nodes, after it has left epoch e. Although it isn't able to learn the secret information of that formerly honest node (since this was discarded when the node left the epoch), it is able to impersonate it, since it now knows the node's signing key. This allows it to rerun the protocol using the original proposal set but with more than t bad nodes. Therefore in step 4 it will be possible for bad nodes to remove all good proposals, leaving us with a proposal set containing only proposals known to the adversary. Since the proposal set looks good to the isolated node it will be willing to send its points to the new nodes; these are points on a polynomial $P' = P + R$, where R is known to the adversary. If just one of the new nodes is corrupt, the adversary will be able to learn the points sent by the isolated node. It can then combine this information

with what it already learned to recover the original polynomial P, and thus the secret.

To our knowledge the definition of weak compliance is new and so is the observation about forward secure signing and its use in avoiding the isolation attack.

ACKNOWLEDGMENTS

We gratefully acknowledge many helpful comments and suggestions from our anonymous reviewers.

REFERENCES

BLAKLEY, G. 1979. Safeguarding cryptographic keys. In *Proceedings of the AFIPS Conference*. Vol. 48, 313–317.

BRACHA, G. AND TOUEG, S. 1985. Asynchronous consensus and broadcast protocols. *J. ACM 32*, 4, 824–240.

CACHIN, C., KURSAWE, K., LYSYANSKAYA, A., AND STROBL, R. 2002. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'02)*. 88–97.

CANETTI, R., HALEVI, S., AND KATZ, J. 2003. A forward-secure public-key encryption scheme. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'03)*. 255–271.

CASTRO, M. AND LISKOV, B. 2002. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst. 20*, 4, 398–461.

CHEN, K. 2004. Authentication in a reconfigurable byzantine fault tolerant system. Master's thesis, MIT.

COWLING, J., PORTS, D. R. K., LISKOV, B., POPA, R. A., AND GAIKWAD, A. 2009. Census: Location-aware membership management for large-scale distributed systems. In *Proceedings of the USENIX Annual Technical Conference*. USENIX.

DESMEDT, Y. AND JAJODIA, S. 1997. Redistributing secret shares to new access structures and its applications. Tech. rep. ISSE TR-97-01, George Mason University.

DWORK, C., NAOR, M., REINGOLD, O., AND STOCKMEYER, L. 2003. Magic functions: In memoriam: Bernard m. dwork 1923–1998. *J. ACM 50*, 6, 852–921.

FELDMAN, P. 1987. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC'87)*. 427–437.

FRANKEL, Y., GEMMELL, P., MACKENZIE, P., AND YUNG, M. 1997. Optimal resilience proactive public-key cryptosystems. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS'97)*. 384–393.

HERZBERG, A., JARECKI, S., KRAWCZYK, H., AND YUNG, M. 1995. Proactive public key and signature systems. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'97)*. 100–110.

HERZBERG, A., JAKOBSSON, M., JARECKI, S., KRAWCZYK, H., AND YUNG, M. 1997. Proactive secret sharing, or how to cope with perpetual leakage. In *Proceedings of the International Cryptology Conference (CRYPTO'95)*. 457–469.

ITO, M., SAITO, A., AND NISHIZEKI, T. 1987. Secret sharing scheme realizing general access structure. In *Proceedings of the IEEE Conference and Exhibition on Global Telecommunications (GlobeCom'87)*.

KRAWCZYK, H. 2000. Simple forward-secure signatures from any signature scheme. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'00)*.

OSTROVSKY, R. AND YUNG, M. 1991. How to withstand mobile virus attacks. In *Proceedings of the Annual ACM SIGOPS Symposium on Principles of Distributed Computing (PODC'91)*. 51–61.

PEDERSEN, T. P. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the International Cryptology Conference (CRYPTO'91)*. 129–140.

RABIN, T. 1998. A simplified approach to threshold and proactive RSA. In *Proceedings of the International Cryptology Conference (CRYPTO'98)*. 89–104.

RODRIGUES, R., LISKOV, B., CHEN, K., LISKOV, M., AND SCHULTZ, D. 2007. Automatic reconfiguration for large-scale distributed storage systems. *IEEE Trans. Depend. Secur. Comput.*

SCHULTZ, D. 2007. Mobile proactive secret sharing. Master's thesis, MIT.

SHAMIR, A. 1979. How to share a secret. *Comm. ACM 22*, 612–613.

WONG, T. M., WANG, C., AND WING, J. 2002. Verifiable secret redistribution for archive systems. In *Proceedings of the International IEEE Security in Storage Workshop (SISW'02)*.

YAO, D., FAZIO, N., DODIS, Y., AND LYSYANSKAYA, A. 2004. ID-Based encryption for complex hierarchies with applications to forward security and broadcast encryption. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'04)*. 354–363.

ZHOU, L., SCHNEIDER, F. B., AND VAN RENESSE, R. 2005. APSS: Proactive secret sharing in asynchronous systems. *ACM Trans. Inf. Syst. Secur. 8*, 3, 259–286.

Received October 2008; revised July 2009; accepted January 2010