Proactive Secret Sharing Or: How to Cope With Perpetual Leakage

Amir Herzberg Stanisław Jarecki* Hugo Krawczyk Moti Yung

IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
{amir,stasio,hugo,moti}@watson.ibm.com

Abstract. Secret sharing schemes protect secrets by distributing them over different locations (share holders). In particular, in k out of n threshold schemes, security is assured if *throughout the entire life-time of the secret* the adversary is restricted to compromise less than k of the n locations. For long-lived and sensitive secrets this protection may be insufficient.
We propose an efficient *proactive* secret sharing scheme, where shares are periodically renewed (*without changing the secret*) in such a way that information gained by the adversary in one time period is useless for attacking the secret after the shares are renewed. Hence, the adversary willing to learn the secret needs to break to all k locations *during the same time period* (e.g., one day, a week, etc.). Furthermore, in order to guarantee the availability and integrity of the secret, we provide mechanisms to detect maliciously (or accidentally) corrupted shares, as well as mechanisms to secretly recover the correct shares when modification is detected.

1 Introduction

Secret sharing schemes protect the secrecy and integrity of information by distributing the information over different locations. For sensitive data these schemes constitute a fundamental protection tool, forcing the adversary to attack multiple locations in order to learn or destroy the information. In particular, in a $(k+1, n)$-threshold scheme, an adversary needs to compromise more than k locations in order to learn the secret, and corrupt at least $n-k$ shares in order to destroy the information. However, the adversary has the *entire life-time of the secret* to mount these attacks. Gradual and instantaneous break-ins into a subset of locations over a long period of time may be feasible for the adversary. Therefore for long-lived and sensitive secrets the protection provided by traditional secret sharing may be insufficient.

A natural defense is to periodically refresh the secrets; however, this is not always possible. That is the case of inherently long-lived information, such as cryptographic master keys (e.g., signature/certification keys), data files (e.g., medical records), legal documents (e.g., a will or a contract), proprietary trade-secret information (e.g., Coca-Cola's formula), and more.

* Laboratory of Computer Science, Massachusetts Institute of Technology

D. Coppersmith (Ed.): Advances in Cryptology - CRYPTO '95, LNCS 963, pp. 339-352, 1995.

Thus, what is actually required to protect the secrecy of the information is to be able to *periodically renew the shares without changing the secret*, in such a way that any information learned by the adversary about individual shares becomes obsolete after the the shares are renewed. Similarly, to avoid the gradual destruction of the information by corruption of shares it is necessary to *periodically recover lost or corrupted shares* without compromising the secrecy of the recovered shares.

These are the core properties of *proactive secret sharing* as presented here. In the proactive approach, the lifetime of the secret is divided into *periods of time* (e.g., a day, one week, etc.). At the beginning of each time period the share holders engage in an interactive *update protocol*, after which they hold completely *new* shares of the *same* secret. Previous shares become obsolete and should be safely erased. As a consequence, in the case of a $(k+1, n)$ proactive threshold scheme, the adversary trying to learn the secret is required to compromise $k+1$ locations during a *single* time period, as opposed to incrementally compromising $k+1$ locations over the *entire* secret life-time. (As an example consider a secret that lives for five years; a weekly refreshment of shares will reduce the time available for the adversary to break the $k+1$ necessary locations from five years to one week.) Similarly, the destruction of the secret requires the adversary to corrupt $n-k$ shares in a *single* time period.

Note that in this setting the adversary is *mobile* and may break into each server multiple times. It nevertheless cannot compromise the secret if at any time period it does not break into more than k locations.

Our solution to the proactive secret sharing problem can support up to $k = n/2 - 1$ corrupted parties at any time period. It assumes the existence of secure encryption and signature functions, as well as the security of the verifiable secret sharing scheme (VSS) based on homomorphic functions [11, 17]. At the system level, we assume a broadcast channel and synchrony (as in VSS). The exact model and assumptions are described in section 2.1.

The mobile adversary setting was originally presented in the context of secure systems by Ostrovsky and Yung [18] with the focus on a theoretical setting of "general distributed function evaluation". That solution allowed large (polynomial) redundancy in the system (redundancy is the ratio of total servers n to the threshold k of simultaneously faulty servers), and used the availability of huge majority of honest servers to achieve the very general task of secure computation in the information theoretic sense. That model was then used in a more practical setting by Canetti and Herzberg [3] who proactively maintained a distributed pseudorandom generator.

Applications: Proactive secret sharing has numerous applications, primarily maintaining data which is long-lived in scenarios where availability and secrecy are crucial. Recently, we employed it to implement "proactive function sharing" where the shares are never combined to reconstruct a single secret, but rather used to collectively compute a function many times and in different inputs. (This follows the *function sharing* model of [7] based on threshold encryption [8].) A particularly attractive application (presented in a forthcoming paper [14]) provides *proactive digital signatures* which achieve the benefits of threshold signatures, with the additional property that the scheme is broken only if the adversary corrupts more than a threshold of the servers in a *single* time-period. For signature keys (e.g., a certification authority) that live for long time and require very strong security, this solution is of particular importance.

Organization: Section 2 presents in some detail the proactive secret sharing model, including the adversary model and the basic definitions of security. It also describes the basic cryptographic tools used in our solution. Section 3 describes the share renewal protocol, and Section 4 describes the share recovery protocol (proofs are omitted from this extended abstract). Section 5 deals with secret reconstruction, and Section 6 shows how to maintain inter-server authentication/decryption keys securely in the proactive setting. Section 7 summarizes the result.

2 Preliminaries

2.1 Model and Assumptions

We assume a system of n servers $\mathcal{A} = \{P_1, P_2, \ldots, P_n\}$ that will (proactively) share a secret value x through a $(k+1, n)$-threshold scheme (i.e., k shares provide no information on the secret, while $k+1$ suffice for the reconstruction of the secret). We assume that the system is securely and properly initialized. The goal of the scheme is to prevent the adversary from learning the secret x, or from destroying it (In particular, any group of $k+1$ non-faulty servers should be able to reconstruct the secret whenever it is necessary).

SERVERS AND COMMUNICATION MODEL. Each server in $\mathcal{A}$ is connected to a common broadcast medium C, called communication channel, with the property that messages sent on C instantly reach every party connected to it. We assume that the system is synchronized, i.e., the servers can access a common global clock, and that each server in $\mathcal{A}$ has a local source of randomness.

TIME PERIODS AND UPDATE PHASES. Time is divided into *time periods* which are determined by the common global clock (e.g., a day, a week, etc.). At the beginning of each time period the servers engage in an interactive *update protocol* (also called *update phase*). At the end of an update phase the servers hold new shares of the secret x.

THE MOBILE ADVERSARY MODEL. The adversary can corrupt servers at any moment during a time period. If a server is corrupted during an update phase, we consider the server as corrupted during both periods adjacent to that update phase. We assume that the adversary corrupts *no more* than k out of n servers in each time period, where k must be smaller than $n/2$ (this guarantees the existence of $k+1$ honest servers at each time).

Corrupting a server means any combination of learning the secret information in the server, modifying its data, changing the intended behavior of the server, disconnecting it, and so on. For the sake of simplicity, we do not differentiate between malicious faults and "normal" server failures (e.g., crashes, power failures etc.). We assume also that the adversary is connected to the broadcast channel C, which means she can hear *all* the messages and inject her own. She cannot, however, modify messages sent to C by a server that she does not control, nor prevent a non-corrupted server from receiving a message sent on C. Additionally, the adversary always knows the non-secret data and the algorithm that each machine performs.

We assume the adversary to be *computationally bounded*, such that it cannot break the underlying cryptographic primitives on which we base our design (this includes public-key encryption and signatures, and verifiable secret sharing – see Section 2.2).

SECURITY OF A PROACTIVE SECRET SHARING SCHEME. We state the security properties of our proactive secret sharing algorithm, relative to the adversary defined above . We only sketch the definition of security, following the notion of *semantic security* introduced in [12]. In the formal definition, to be presented in the complete version of this paper, the adversary is modeled as a computationally bounded interactive probabilistic Turing machine which is fed with all the publicly available information on the secret (e.g., its length, a particular subspace from which the secret is chosen, the value of the secret under a one-way function, and so on), and with the information learned by the adversary during (one or more) runs of the update protocol (this includes all the public communication between servers, secret information of the servers that were corrupted in each of these periods, etc.).

Let κ be a function applicable to the space of secrets x. Let $p_0^{(\kappa)}$ be the probability that the adversary correctly computes the value $\kappa(x)$ when fed with the a-priori (public) information on the secret, and let $p_1^{(\kappa)}$ be the analogous probability but after the adversary is fed with the additional information gathered during the run of the protocol. (The above probabilities depend on the random coins used by the adversary and the servers.) Intuitively, the function $\kappa(x)$ models some knowledge about x, while the difference $p_1^{(\kappa)} - p_0^{(\kappa)}$ quantifies the amount of that knowledge "learned" by the adversary by watching the execution of the protocol and actively intruding the servers.

Definition 1. (sketch) We say that a proactive secret sharing scheme is semantically secure if for any function κ computable on the secret, the difference between the probabilities $p_0^{(\kappa)}$ and $p_1^{(\kappa)}$ is negligible.

The exact notion of "negligible" in the above definition depends on the exact model of the adversary. In the traditional complexity-theoretic setting of a polynomial time adversary, one considers these probabilities as functions of the secret length, and "negligible" stands for any function that decreases faster than any inverse polynomial. A more careful model would bound the difference between $p_0^{(\kappa)}$ and $p_1^{(\kappa)}$ as an explicit function of the (small) probabilities with which the adversary can break the underlying cryptographic primitives.

In some cases, in order to stress the existence of an a-priori public information $\pi(x)$ on the secret x, we will say that the proactive secret sharing scheme is semantically secure relative to $\pi(x)$.

Not only we are interested to preserve the secrecy of x, but also to guarantee its availability and recoverability. This means that we need to prevent the adversary from destroying the secret or impeding its reconstruction by, for example, destroying or modifying shares.

Definition 2. (sketch) A proactive secret sharing scheme that guarantees the correct reconstructibility of the secret at any time is called *robust*.

Notice that for a proactive secret sharing scheme to be robust, one needs to ensure that in any time period the honest servers (which could have been corrupted during previous time periods) have correct shares (i.e., ones that combine to the correct secret x), and that this correctness can be verified by the other servers. This requires that honest servers be able to verify whether each of them stores a correct share. Also, those who do hold correct shares must be able to cooperate in order to recover the shares of the ones that lost them (without exposing the recovered share to anybody except its intended holder).

The focus of this paper is to construct *semantically secure and robust proactive secret sharing scheme* based on the existence of secure public-key encryption [12] and signatures [13], as well as on the existence of verifiable secret schemes [11, 16]. The theorems in this paper are stated relative to these security notions and the above adversary model.

A NOTE ABOUT THE REMOVAL OF AN ADVERSARY FROM A SERVER. We assume that the adversary intruding the servers $\mathcal{A}$ is "removable" (e.g., through a reboot procedure) when it is detected. The responsibility for triggering the reboot operation (or other measures to guarantee the normal operation of a server) relies on the system management which gets input from the servers in the network. In addition to regular detection mechanisms (e.g., anti-virus scanners) available to the system management, our protocols provide explicit mechanisms by which a majority of (honest) servers can detect and alert about a misbehaving server. We assume for simplicity that the reboot operation is performed immediately when attacks or deviations from the protocol are detected. We remark that the initialization of servers and reboot operations require a minimal level of *trust* in the system management, restricted to installation of correct programs and of public keys used for server-to-server communication. Specifically, no secret information is exposed.

2.2 Cryptographic Tools

SHAMIR'S SECRET SHARING. Our secret sharing scheme is based on Shamir's scheme [19]. Let q be a prime number, $x \in Z_q$ be the secret to be shared, n the number of participants (or share holding servers), and $k+1$ the reconstructibility threshold. The dealer D of the secret chooses a random polynomial f of degree k over Z_q subject to the condition $f(0) = x$. Each share x_i is then computed by D as $f(i)$ and then transmitted *secretly* to participant P_i. The reconstruction of the secret can be done by having $k+1$ participants providing their shares and using polynomial interpolation to compute x.

VERIFIABLE SECRET SHARING – VSS. In Shamir's scheme a misbehaving dealer can deal inconsistent shares to the participants, from which they will *not* be able to reconstruct a secret. To prevent such malicious behavior of the dealer one needs to implement a procedure or protocol through which a consistent dealing can be verified by the recipients of shares. Such a scheme is called *verifiable secret sharing* (VSS) [5] . Our work uses these schemes in an essential way. We implement our solution using specific schemes due to Feldman [11] and Pedersen [17]. These schemes are based on hard to invert homomorphic functions and, in particular, on the hardness of computing discrete logarithms over Z_p, for prime p. The first scheme protects the secrecy of the secret in a computational sense, while the second provides information theoretic secrecy. Our solution works with either of these schemes. We choose to present here the solution using Feldman's scheme since it is somewhat simpler. However, the use of Pedersen's scheme strengthens and simplifies the proofs of security.

FELDMAN'S VSS. For completeness we briefly describe Feldman's scheme. Let p be a prime number with $p = mq + 1$, where q is also prime, and m an integer (possibly, a small one like 2, 3, 4). Let g be an element of Z_p of order q. The dealer chooses the polynomial f over Z_q with coefficients $f_0, f_1, \ldots, f_k$ and broadcasts the corresponding values $g^{f_0}, g^{f_1}, \ldots, g^{f_k}$. Then it secretly transmits the value $x_i = f(i)\ (mod\ q)$ to P_i. Each server P_i verifies its own share by checking the following equation:

$$g^{x_i} \stackrel{?}{=} (g^{f_0})(g^{f_1})^i(g^{f_2})^{i^2} \ldots (g^{f_k})^{i^k}\ (mod\ p) \tag{1}$$

If the equation holds, P_i broadcasts a message accepting its share as proper. If all servers find their shares correct then the dealing phase is completed successfully. Indeed, by the homomorphic properties of the exponentiation function (i.e., $g^{a+b} = g^a g^b$) the above equation holds for all $i \in \{1 \ldots n\}$ if and only if the shares were dealt correctly. It is worth noticing that besides allowing the verification of correct dealing of shares, the public values g^{x_i} can be used at time of secret reconstruction to verify that the participating shares are correct (see Section 5).

Notice that Feldman's scheme reveals the value $y = g^x\ (mod\ p)$, where $x = f_0$ is the secret being shared. Although the entire value of x cannot be derived from y (assuming the hardness of discrete logarithm), still there is partial information on x (e.g., its least significant bit) which is exposed by y. Therefore the semantic security of our solution (when based on Feldman's scheme) can be only stated *relative to the knowledge of* g^x $(mod\ p)$. This is acceptable in cases where g^x is known anyway (see [14]); however, in general, one would like to prevent the leakage of partial information. In this case, one encodes the actual secret, say s, into a longer string x and performs the proactive secret sharing using x. The encoding should have the property that given x, the value of s is easy to recover, but given $g^x\ (mod\ p)$ then no information on s can be efficiently computed. Such schemes exist based on *hard core bits* of the exponentiation function (e.g., if s is represented by the $O(\log(|p|))$ most significant bits of x [15]; a more efficient construction can be based on [1]). For more information see [11, 16]. ***This issue can be completely avoided in our solution by replacing Feldman's VSS with the information theoretic scheme of [17].***

PUBLIC-KEY ENCRYPTION AND SIGNATURES. Our solution requires semantically secure encryption [12] and existentially unforgeable signatures [13]. We do not specify or assume any particular implementation of these functions. For a pair of *sender* S and *receiver* R, we denote by $ENC_R(data)$ the probabilistic encryption of *data* under R's public key; and by $SIG_S(data)$ the signature of *data* under S's private key.

3 Periodic Share Renewal Scheme

Here we present a basic component of our solution, namely, the protocols for periodic renewal of shares which preserve the secret, and at the same time make past knowledge obsolete for the adversary. Beyond guaranteeing the secrecy of the shared secret, our scheme is robust in the sense of guaranteeing integrity and availability of the secret in the presence of up to k misbehaving servers.

3.1 Initial Setting: Black-box Public Key Assumptions

Cryptographic solutions in a distributed environment typically require the ability to maintain private and authenticated communication between the servers. This is achieved by the servers having pairs of private and public keys corresponding to public-key cryptosystems with encryption and signature capabilities (e.g., RSA, ElGamal, etc.). However, an adversary that breaks into a server and learns its private key can then impersonate that server for the whole life of that private key. Therefore, to ensure proactive security, it is necessary to maintain these private keys proactively, namely, to renew them periodically.

We will show in section 6 how this can be done in our context (a more general treatment of proactive authentication can be found in [4]). However, for clarity of presentation, we start by making the strong assumption that servers are equipped with a pair of private and public keys in a way that the private key cannot be learned or modified by the adversary, even if this adversary manages to break into the server. While such an intruder will be able to generate legal signatures and decrypt messages using the private key (as a "black-box"), it will not be able to learn or modify the key itself. We will remove this assumption and deal with the proactive maintenance of the private keys in section 6.

3.2 Initialization of Secret Sharing

We assume an initial stage where a secret $x \in Z_q$ (for prime q) is encoded into n pieces $x_1, \ldots, x_n \in Z_q$ using a k-threshold Shamir's secret sharing: Each $P_i, i \in \{1 \ldots n\}$ holds its share x_i, where $x_i = f(i)$ for some k-degree polynomial $f(\cdot)$ over Z_q s.t. $x = f(0)$.

After the initialization, at the beginning of every time period, all honest servers trigger an *update phase* in which the servers perform a *share renewal protocol*. The shares computed in period t are denoted by using the superscript (t), i.e., $x_i^{(t)}, t = 0, 1, \ldots$. The polynomial corresponding to these shares is denoted $f^{(t)}(\cdot)$.

3.3 Share renewal

To renew the shares at period $t, t = 1, 2, \ldots$, we adapt a simplified version of the update protocol presented by Ostrovsky and Yung in [18]. When the secret x is (distributively) stored as a value $f^{(t-1)}(0) = x$ of a k degree polynomial $f^{(t-1)}(\cdot)$ in Z_q, we can update this polynomial by adding it to a k degree random polynomial

$\delta(\cdot)$, where $\delta(0) = 0$, so that $f^{(t)}(0) = f^{(t-1)}(0) + \delta(0) = x + 0 = x$. By the linearity of the polynomial evaluation operation we get the renewal of the shares $x_i^{(t)} = f^{(t)}(i)$ according to:

$$f^{(t)}(\cdot) \leftarrow f^{(t-1)}(\cdot) + \delta(\cdot)\ (mod\ q) \iff \forall_i\ f^{(t)}(i) = f^{(t-1)}(i) + \delta(i)\ (mod\ q)$$

In our system we will have $\delta(\cdot) = (\delta_1(\cdot) + \delta_2(\cdot) + \ldots + \delta_n(\cdot))\ (mod\ q)$, where each polynomial $\delta_i(\cdot)$, $i \in \{1 \ldots n\}$ is of degree k and is picked independently and at random by the ith server subject to the condition $\delta_i(0) = 0$. The share renewal protocol for each server P_i, $i \in \{1 \ldots n\}$, at time period t is as follows:

1. P_i picks k random numbers $\{\delta_{im}\}_{m \in \{1 \ldots k\}}$ from Z_q. These numbers define a polynomial $\delta_i(z) = \delta_{i1} z^1 + \delta_{i2} z^2 + \ldots + \delta_{ik} z^k$ in Z_q, whose free coefficient is zero and hence, $\delta_i(0) = 0$.
2. For all other servers P_j, P_i *secretly* sends $u_{ij} = \delta_i(j)\ (mod\ q)$ to P_j.
3. After decrypting u_{ji}, $\forall j \in \{1 \ldots n\}$, P_i computes its new share $x_i^{(t)} \leftarrow x_i^{(t-1)} + (u_{1i} + u_{2i} + \ldots + u_{ni})\ (mod\ q)$ and *erases* all the variables it used except of its current secret key $x_i^{(t)}$.

This protocol solves the share renewal problem against a ("passive") adversary that may learn the secret information available to corrupted servers but where all servers follow the predetermined protocol. This is proven in the next theorem. Notice that we assume here that the shares are transmitted to the corresponding holders with perfect secrecy. This allows us to prove the information-theoretic secrecy of this scheme. In the next sections we use explicit encryption for the transmission of these shares and then the secrecy of the scheme is reduced to the strength of the encryption.

Theorem 3. *If all servers follow the above share renewal protocol then:*

Robustness: The new shares computed at the end of the update phase correspond to the secret x (i.e., any subset of $k+1$ of the new shares can reconstruct the secret x).

Secrecy: An adversary that at any time period knows no more than k shares (possibly a different set of shares in each period) learns nothing about the secret.

3.4 Share Renewal Protocol in the Presence of Active Attackers

In the above basic share renewal protocol an *active* adversary controlling a server can cause the destruction of the secret by dealing inconsistent share updates or just by choosing a polynomial δ_i with $\delta_i(0) \neq 0$. In order to assure the detection of wrongly dealt shares we add to the above basic protocol a *verifiability* feature. Namely, we adapt to our scenario Feldman's verifiable secret sharing scheme as described in section 2.2. In traditional applications of verifiable secret sharing, the fact that all the share-holders find their shares to be consistent is used as a proof for a correct dealing of the secret. In our case, this is used as a proof for correct dealing of update shares by the servers.

The verifiable share renewal protocol for each server P_i at period t is as follows:

1. P_i picks k random numbers $\{\delta_{im}\}_{m\in\{1...k\}}$ from Z_q to define the polynomial $\delta_i(z) = \delta_{i1}z^1 + \delta_{i2}z^2 + \ldots + \delta_{ik}z^k$. It also computes values $\epsilon_{im} = g^{\delta_{im}} \ (mod\ p)$, $m \in \{1 \ldots k\}$.
2. P_i computes $u_{ij} = \delta_i(j) \ (mod\ q)$, $j \in \{1 \ldots n\}$, and computes $e_{ij} = ENC_j(u_{ij})$, $\forall j \neq i$.
3. P_i broadcast the message $VSS_i^{(t)} = (i, t, \{\epsilon_{im}\}_{m\in\{1...k\}}, \{e_{ij}\}_{j\in\{1...k\}\setminus\{i\}})$, and the signature $SIG_i(VSS_i^{(t)})$.
4. For all messages broadcasted in the previous step by other servers, P_i decrypts the shares intended for P_i (i.e., computes u_{ji} out of e_{ji}, $\forall j \neq i$), and verifies the correctness of shares using the equivalent of the verifiability equation 1 from section 2.2, namely, for all $j \neq i$ it verifies:

$$g^{u_{ji}} \stackrel{?}{=} (\epsilon_{j1})^i (\epsilon_{j2})^{i^2} \ldots (\epsilon_{jk})^{i^k} \ (mod\ p). \tag{2}$$

   (Notice that this equation accounts for the condition $\delta_j(0) = 0$.)
5. If P_i finds all the messages sent in the previous step by other servers to be correct (e.g., all have correct signatures, time period numbers, etc.), and all the above equations to hold, then it broadcasts a signed acceptance message announcing that all checks were successful.
6. If all servers sent acceptance messages then P_i proceeds to update its own share by performing: $x_i^{(t)} \leftarrow x_i^{(t-1)} + (\, u_{1i} + u_{2i} + \ldots + u_{ni}\,) \ (mod\ q)$ and *erases* all the variables it used except for its current share $x_i^{(t)}$.
7. If in the above step 5, P_i finds any irregularities in the behavior of other servers during step 4 then it broadcasts a signed *accusation* against the misbehaving server(s). When to send accusations, and how to resolve them is discussed in the next subsection.

Resolving accusations In step 5 of the above protocol, each server checks the correct behavior and dealing of other servers. If a misbehaving server is found then there are two kinds of actions to take. One is *not* to use the polynomial $\delta_i(\cdot)$ dealt by this server in the renewal of shares in step 6. The second is to alert the system management so that it could take measures to rectify the misbehaving server (e.g., it may be required to reboot the server in order to "expel" the adversary). However, an accusation against a server by another server requires verification, since a misbehaving server could falsely accuse others. For a consistent update of shares, the honest servers need to agree on who the "bad" servers are. We explain below how each server P_i decides on its list B_i of bad processors.

We say that a message from server P_i at period t is *correct* if it complies with the specifications of the above protocol, including all the specified fields and information (e.g., the correct time period number) as well as a correct signature.

We distinguish between three classes of irregularities in the protocol: (a) Formally incorrect messages: wrong time period numbers, numbers out of bounds etc.; (b) Two or more correct messages from the same server containing a valid signature, or no message at all from some server; and (c) a mismatch in equation 2.

Notice that irregularities of the first two types are discovered using public information only, and then are found by all (honest) servers. Any such misbehaving processor is marked as "bad" by each of the other processors. The faults of the third kind are more problematic since they are discovered only locally by a server that receives a share causing a mismatch in equation 2.

When a server P_i finds that equation 2 corresponding to the information sent by P_j does not hold, it needs to broadcast an accusation against P_j. Then the servers decide whether it is P_i or P_j who is cheating. A way to do this is by having P_j "defend" itself. If P_j sent a correct u_{ji}, namely, one that passes equation 2, then it can expose this value and prove that it corresponds to the publicly available encryption value e_{ji} sent by P_j to P_i. To prove this P_j may need to reveal additional information used to compute the encryption (like the random vector used in probabilistic encryption). However, P_j does not need to reveal any private information of itself. Then everybody can check whether u_{ji} and the additional information published by P_j encrypts under P_i's public key to e_{ji} as broadcasted by P_j in step 2. Second, everybody can check whether this u_{ji} matches equation 2. If P_j defends itself correctly then all servers mark P_i as bad, otherwise P_j is marked as bad. Notice that in some encryption schemes (like RSA), the information published by the accuser is sufficient for public verification, which simplifies the above general protocol.

Once all accusations are resolved, every honest server P_i holds the same list of bad servers $\mathcal{B}_i$ (i.e., for each pair (P_i, P_j) of non-faulty servers, $\mathcal{B}_i = \mathcal{B}_j$). Now the computation of the new shares is done by replacing step 6 of the share renewal protocol by:

$$x_i^{(t)} \leftarrow x_i^{(t-1)} + \sum_{j \notin \mathcal{B}_i} u_{ji} \ (mod\ q)$$

3.5 Security Properties

In Theorem 3 we dealt with the share renewal assuming the servers are curious but honest. Here we claim the analogous result for the case that up to k servers are arbitrarily misbehaving in the renewal protocol.

Theorem 4. *If the adversary controls up to k servers during the protocol, then:*

Robustness: The new shares computed at the end of the update phase by honest servers correspond to the secret x (i.e., any subset of $k+1$ of the new shares of honest servers interpolate to the secret x).

Secrecy: The above secret sharing scheme is semantically secure. [2]

4 Share Recovery Scheme

A proactive secret sharing system must be able to check whether a share of each participating server has been corrupted (or lost), and restore the correct share if

[2] In the protocol above we achieve semantic security relative to the a priori knowledge of the exponent $g^x \ (mod\ p)$ of the secret x. This extra knowledge is avoided by using Pedersen's VSS scheme [17] (see discussion on Feldman's VSS in section 2.2).

necessary. Otherwise, an adversary could cause the loss of the secret by gradually destroying $n - k$ shares. Below we present the necessary mechanisms for detection and recovery of corrupted shares.

The share $x_i^{(t)}$ held by processor P_i in period t is called *correct* if $x_i^{(t)} = f^{(t)}(i) \ (mod\ q)$, where $f^{(t)}$ is the current secret sharing polynomial. Otherwise, we say that the share is *incorrect*. A server can have an incorrect share because it was controlled by the adversary during the share renewal protocol (and hence it was prevented to update its share correctly), or because the adversary attacked the server after the update phase and modified the server's secret share. A secret share can also be lost because a server was rebooted or replaced by a new server.

4.1 Detection of Corrupted Shares

How are corrupted shares detected? In some cases it is easy to detect that a server requires to recover its correct share. This is the case of servers that do not participate of an update phase (e.g., due to a crash), or servers that misbehave during that phase. However, if the share of some server is ("silently") modified by the adversary (e.g., after an update phase) then this modification may go undetected. Hence, in the spirit of proactiveness, the system must periodically test the correctness of the local states of the participating servers, detecting in this way lost or modified shares.

To implement the distributed verifiability of shares, we add an invariant that in each time period t, each server P_i stores a set $\{y_j^{(t)}\}_{j\in\{1\ldots n\}}$ of exponents $y_j^{(t)} = g^{x_j^{(t)}} \ (mod\ p)$ of current shares of all servers in $\mathcal{A}$. This is achieved as follows. First, we augment section 3.2 with the requisite that each server stores the values $y_j^{(0)}$ corresponding to the initial shares $x_j^{(0)}, j \in \{1\ldots n\}$ (this can be achieved by performing Feldman's VSS at initialization). Also, using the homomorphism of the exponentiation function, we supplement step 6 of the update protocol in section 3.4 so that each server P_i updates its set $\{y_j\}_{j\in\{1\ldots n\}}$ by computing for every j:

$$y_j^{(t)} \leftarrow y_j^{(t-1)} * (g^{u_{1j}} * g^{u_{2j}} * \cdots * g^{u_{nj}}) \ (mod\ p)$$

(in the general case, the above product is computed using only update shares u_{mj} corresponding to servers that did not misbehave in the update phase, i.e., $m \notin \mathcal{B}_i$).

We extend the ***update phase*** between time periods to include a ***share recovery*** protocol. Its first part is the ***lost share detection protocol*** which works as follows: Every server broadcasts the values $\{y_j^{(t)}\}_{j\in\{1\ldots n\}}$ stored in that server, together with a signature on these values. After collecting these messages from all servers and checking their signatures, each server decides by majority on the current proper set of share exponents. Now each server P_i can decide on a set $\mathcal{B}_i$ of servers which presented an incorrect (i.e., different from majority) exponent of their own share. These are the servers that P_i believes to need a share recovery. (In particular, it can be the case that for some i, $P_i \in \mathcal{B}_i$, which means that server P_i decided that its own share is not correct). It is clear that every pair of non-faulty servers (P_i, P_j) has the same view about who has an incorrect share, i.e., $\mathcal{B}_i = \mathcal{B}_j = \mathcal{B}$, where $|\mathcal{B}| \leq k$.

4.2 Basic Share Recovery Protocol

A straightforward way to reconstruct the shares $x_r = f^{(t)}(r)$ for $r \in \mathcal{B}$, is to let each server in $\mathcal{D} = \mathcal{A} \setminus \mathcal{B}$ send its own share to P_r. However, this would expose the secret x to P_r. Instead, for each $r \in \mathcal{B}$, the servers in the set $\mathcal{D}$ will collectively generate a random secret sharing of x_r in a way analogous to that used to re-randomize the secret sharing of the main secret x in the share renewal protocol: Every server P_i in $\mathcal{D}$ deals a random k-degree polynomial $\delta_i(\cdot)$, such that $\delta_i(r) = 0 \ (mod\ q)$. In this way, a new, random secret sharing $\{x'_i\}_{i \in \mathcal{D}}$ of x_r is obtained. The servers can now send these new shares to P_r, to allow it to compute x_r without letting P_r learn anything about the original shares $\{x_i\}_{i \in \mathcal{D}}$. Also, any coalition of k or less servers, not including P_r, will learn nothing about the value of x_r.

We first present the share recovery protocol stripped of verifiability. It is secure only against an adversary that eavesdrops into k or less servers, but does not change the servers behavior. For each P_r that requires share recovery, the following protocol is performed:

1. Each $P_i, i \in \mathcal{D}$, picks a random k-degree polynomial $\delta_i(\cdot)$ over Z_q such that $\delta_i(r) = 0$, i.e. it picks random coefficients $\{\delta_{ij}\}_{j \in \{1 \dots k\}} \subset Z_q$ and then computes $\delta_{i0} = -\sum_{j \in \{1 \dots k\}} \delta_{ij} r^j \ (mod\ q)$.
2. Each $P_i, i \in \mathcal{D}$, broadcasts $\{ENC_j(\delta_i(j))\}_{j \in \mathcal{D}}$.
3. Each $P_i, i \in \mathcal{D}$, creates its new share of x_r, $x'_i = x_i + \sum_{j \in \mathcal{D}} \delta_j(i)$ and sends it to P_r by broadcasting $ENC_r(x'_i)$.
4. P_r decrypts these shares and interpolates them to recover x_r.

4.3 Full Share Recovery Protocol

In the general case, the adversary not only can eavesdrop into the servers but also cause the corrupted servers to deviate from their intended protocol. To cope with these cases, we add to the above protocol (section 4.2) the necessary "verifiability" properties for the dealing of polynomials $\delta_i(\cdot)$ in step 2 and for reconstruction of x_r from x_i's in step 3 and 4. Due to space constraints we omit the description of this protocol in these Proceedings, but we sketch the properties of this protocol in the following theorem:

Theorem 5. *If the adversary compromises no more than k servers in any time period, then the full share recovery protocol has the following two properties:*

Robustness: Each recovering server that follows the protocol recovers its correct share $x_r = f^{(t)}(r) \ (mod\ q)$.

Secrecy: The semantic security of the secret x is preserved.

5 Secret Reconstruction

In the above section we have shown how to renew shares consistent with the secret, and preserve their integrity such that at any time any $k+1$ parties could reconstruct the secret if desired. However, when coming to actually reconstruct the secret, the participants in the reconstruction protocol (namely, polynomial interpolation as in Shamir's scheme) must be able to detect servers that provide corrupted shares to the reconstruction. This detection is easily accomplished by verification of the submitted shares against values $y_i^{(t)}$ held by the majority of servers.

6 Dynamically Secure Private Keys

We now show how to remove the protected key assumption of section 3.1. We extend the *update phase* between time periods to include a third component, the *private key renewal protocol*, which will be triggered before share recovery and share renewal. As a result of the private key renewal, an adversary that breaks into a server in period t, but which does not control the server at period $t+1$, cannot learn this server's new key.

The private key renewal protocol at the beginning of each update phase works as follows: Each server P_i chooses a new pair of private and public keys $a_i^{(t)}, b_i^{(t)}$. The server then broadcasts its new public key $b_i^{(t)}$ authenticated by its signature using its *previous* private key $a_i^{(t-1)}$. The other servers, having $b_i^{(t-1)}$ from the previous period, can verify this signature. Clearly, an adversary that controlled the server at time period $t-1$, or before, but not during the update phase between periods $t-1$ and t, cannot learn the new private key chosen by the server. However, if the adversary knows $a_i^{(t-1)}$ then, even if she is not controlling P_i during the private key renewal protocol of period t, she can choose her own private key and inject its public counterpart into the broadcast channel, authenticated as if it originated from P_i. But since P_i is not actively controlled by the adversary anymore, it will send its own authenticated public key to the communication channel as well. This will result in two different messages legally authenticated as coming from P_i. This will constitute a public proof of P_i's compromise, after which a reboot procedure must be triggered.

When a server is rebooted, it internally chooses its new private key, publishing only the corresponding public key, which must be then installed on all servers in A.

7 Proactive Secret Sharing: the Combined Protocol

Combining all the above pieces we get our full protocol for proactive secret sharing: At the beginning of every time period, the update phase is trigerred, which consist of three stages: the private key renewal protocol, the share recovery protocol (including lost share detection) and the share renewal protocol. The following theorem summarizes our main result:

Theorem 6. *If there are no more than k corrupted servers in each period (where servers compromised at a renewal phase are considered as compromised in both adjacent periods), then the above protocol constitutes a secure and robust proactive secret sharing scheme.*

References

1. M. Bellare and P. Rogaway, *Optimal Asymmetric Encryption*, Eurocrypt '94.
2. G.R. Blakley, *Safeguarding Cryptographic Keys*, AFIPS Con. Proc (v. 48), 1979, pp 313–317.
3. R. Canetti and A. Herzberg, *Maintaining Security in the Presence of Transient Faults*, Proc. Crypto'94 (LNCS 839), pp. 425-438.
4. R. Canetti and A. Herzberg, *Proactive Maintenance of Authenticated Communication*, manuscript, 1995.
5. B. Chor, S. Goldwasser, S. Micali and B. Awerbuch, *Verifiable Secret Sharing and Achieving Simultaneous Broadcast*, Proc. of IEEE Focs 1985, pp. 335-344.
6. M. Blum and S. Micali, How to Construct Cryptographically Strong Sequences of Pseudorandom Bits. SIAM J. Comp. 13, 1984.
7. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, *How to Share a Function Securely*, ACM STOC 94.
8. Y. Desmedt and Y. Frankel, *Threshold cryptosystems*, In G. Brassard, editor, Advances in Cryptology, Proc. of Crypto '89 (Lecture Notes in Computer Science 435), pages 307–315. Springer-Verlag, 1990.
9. D. Dolev, C. Dwork, O. Waarts, and M. Yung, *Perfectly Secure Message Transmission*, Proc. 31st Annual Symposium on the Foundations of Computer Science, 1990, (also JACM).
10. T. El Gamal, *A Public key cryptosystem and a signature scheme based on discrete logarithm*, IEEE Trans. on Information Theory 31, 465-472, 1985.
11. P. Feldman, *A Practical Scheme for Non-Interactive Verifiable Secret Sharing*, Proc. of the 28th IEEE Symposium on the Foundations of Computer Science, 1987, 427-437
12. S. Goldwasser and S. Micali, *Probabilistic Encryption*, J. Com. Sys. Sci. 28 (1984), pp 270-299.
13. S. Goldwasser, S. Micali and R. Rivest, *A Secure Digital Signature Scheme*, Siam Journal on Computing, Vol. 17, 2 (1988), pp. 281-308.
14. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk and M. Yung, *Proactive Public Key and Signatures Systems*, draft.
15. D.L. Long and A. Wigderson, *The Discrete Log. Problem Hides O(Log N) Bits.* SIAM J. Comp. 17, 1988, pp. 363-372.
16. T. P. Pedersen, *Distributed Provers with Applications to Undeniable Signature*, Proc. Eurocrypt '91 (LNCS 547), pp. 221-242.
17. T. P. Pedersen, *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*, Proc. Crypto'91 (LNCS 576), pp. 129-140.
18. R. Ostrovsky and M Yung, *How to withstand mobile virus attacks*, Proc. of the 10th ACM Symp. on the Princ. of Distr. Comp., 1991, pp. 51-61.
19. A. Shamir. *How to share a secret*, Commun. ACM, 22 (1979), pp 612-613.