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

October 15, 1998

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 funda-

*Massachusetts Institute of Technology

mental 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.

To realize how unsatisfactory such refreshment of the secret is, imagine that one wants to protect a legal document by encrypting it under some initial key and then periodically change that key, decrypting the document with the old key and encrypting it with the new one every time the key changes. Such a solution does not protect the integrity of the document at all, and it also exposes the secrecy to the adversary that happens to attack the server at the moment when the key is changing and the document is being decrypted.

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 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 any 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 for maintaining data which is long-lived in scenarios where availability and secrecy are crucial. It can also be used as a building block in "proactive function sharing" (see section 8).

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 and section 8 discusses applications.

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).

The reason behind this way of counting corrupted servers is that it is impossible or at least very hard to analyze what happens if we differentiated between adversary who moves from one server to another during the update phase and the adversary who just stays in both servers throughout. It is also not a realistic concern in our setting, where the update phase is negligibly short when compared to the length of a time period: If the adversary could move so fast that she could jump between servers during an update, if it continued to jump with the same speed during the time period, it could visit all the servers and destroy the secret sharing system. Furthermore, notice that even during a regular time period, we treat the adversary who jumps from one server to another in the same way as if both of them are corrupted throughout this time period.

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 also assume 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 can she 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*, so that it cannot break the underlying cryptographic primitives on which we base our design, namely a public-key encryption and signature scheme, and a verifiable secret sharing mechanism – see Section 2.3.

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 always detects and alerts about a misbehaving server. We assume for simplicity that the reboot operation is performed immediately when attacks or deviations from the protocol are detected and that it takes less time then a duration of a time period.

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 to the system management. This level of trust regarding integrity of installed information is unavoidable for the initialization of any cryptographic system. It is also worth noticing that if the system management fails to install the communication keys properly, it can threaten the integrity of the secret, but not its secrecy.

We assume that the adversary cannot be cut from the communication channel C. However, we preclude the possibility that the adversary will flood this communication channel with messages and thus prevent servers $\mathcal{A}$ from communicating. Although this is an attack that can happen in real life, there seem to be no cryptographic ways of preventing it.

Erasure of past information. In our protocols we sometimes specify that the servers *erase* some information. This operation (performed by honest servers) is central to the proactive security setting. Not doing so would provide an adversary that attacks a server at a given period with information from a previous period, and the later could enable the adversary to break the system. In many computer systems, what seems like a memory update to the programmer, can in fact result only in an update of a cache, while the main memory or a copy swapped on a disk remains unchanged. It is a formal requirement of our proactive solution that the secret sharing servers be able to reliably erase their local data.

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, namely, an adversary that corrupts at most k servers in each time period and that is computationally limited and incapable of breaking the underlying cryptographic primitives.

We will only sketch the definition of security here, 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 2.1 *(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 2.2 *(sketch) A proactive secret sharing scheme that guarantees the correct reconstructibility of the secret at any time is called* robust.

An alternative way to think about these properties is that a proactive scheme is *robust* if it is secure in the presence of up to k *byzantine* faults per time period. Without robustness, a scheme that preserves the secrecy of the secret would be secure in the presence of up to k *gossip* faults.

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.

2.3 Cryptographic Tools

Shamir's Secret Sharing. Our secret sharing scheme is based on Shamir's scheme [19]. Let q be a prime number, $x \in Z_q$[1] be the secret to be shared, n the number of participants

[1] In fact this can be done over any finite field.

(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 computed by D as $f(i)$ and then transmitted *secretly* to participant P_i (The evaluation point i could be any publicly known value v_i which uniquely corresponds to P_i; we assume $v_i = i$ as the default value). 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.

Our solution works with either of these schemes. We choose to present here the solution using Feldman's scheme since it is somewhat simpler. The use of Pedersen's scheme weakens the protection of integrity of secret x: It makes the robustness of a proactive secret sharing system subject to computational limits of the adversary. However, in a trade-off for integrity, Pedersen's scheme strengthens the protection of secrecy of x: It allows us to achieve a true semantic security, as opposed to a semantic security relative to the knowledge of g^x.

Feldman's VSS. For completeness we briefly describe Feldman's scheme. Let p and q be two prime numbers such that $p = mq + 1$, where m is a small integer (possibly $2, 3, 4$). Let g be an element of Z_p of order q. is that for each share x_i there is a *public* value $y_i = g^{x_i} \ (mod\ p)$ which by the homomorphic properties of the exponentiation function (i.e., $g^a g^b = g^{ab}$) allows every share-holder to verify that its own share is consistent with the public information.

The dealer chooses the polynomial f over Z_q with coefficients f_0, f_1, ..., 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 this equation holds, P_i broadcasts a message saying that it accepts 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 the above equation holds for all $i \in \{1 \ldots n\}$ if and only if the shares were dealt correctly.

If for some i, P_i finds the above equation incorrect then P_i publishes an *accusation* against the dealer. We describe in section 3.4 how honest servers can decide whether it is the dealer or the accuser that misbehaves. We also discuss there the issues of authentication and encryption of the messages in the above protocol.

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).

SECRECY PROTECTION IN VSS. The above scheme makes the value $y = g^x \ (mod\ p)$ public, where $x = f(0)$ is the secret being shared. Therefore the semantic security of our solution (when based on Feldman's scheme) can be only stated relative to the knowledge of $g^x \ (mod\ p)$. Assuming the hardness of the discrete logarithm operation, the entire value of x cannot be derived from y. However, there is partial information on x that can be efficiently derived, and that, consequently, is not protected by the scheme. Such unprotected information includes the value of $g^x \ (mod\ p)$ itself, the least significant bit of x, etc.

However, to stress the usefulness of Feldman's VSS scheme (and ours) we outline here a methodology to apply it to a secret without leaking the partial information: The real secret to be protected, say s, is first encoded into a longer string x (an "envelope" for s) with the property that given x it is easy to recover s, but given $g^x \ (mod\ p)$ it is hard to derive any information on s (i.e., s represents the *hard core* information of x). This provides for *semantic security* of s [12] under the assumption that the exponentiation function is hard to invert on random input.

For example, it is known ([15], see also [6]) that computing the logarithmic number of upper bits of x from $g^x \ (mod\ p)$ is hard. Therefore, for a secret s of length $\log |q|$, one could construct the envelope x (of size $|q|$) as a concatenation of s (as upper most bits) and a random string r. We refer to [1] for a general construction of practical hard core envelopes for *any* one-way permutation (applicable, in particular, to the exponentiation function).

Throughout the paper we refer to x as the secret. Applications in which the exposure of the secret's exponent is unacceptable should use the above envelope method. Other applications (e.g., proactive ElGamal signatures [14] mentioned in section 8) are secure even with this exponent being known. The readers should also notice that *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 the fundamental 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. During a break-in, the adversary could also modify the private or public keys stored on that server, thus disabling it from communicating with others. Also, if the adversary breaks into P_i and replaces P_j's public key (in P_i's storage of other servers' public keys) with her own, she can later spook P_j to P_i. Therefore, to ensure proactive security, it is necessary to maintain the system of private and public communication 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 with a property that the private key cannot be learned or modified by the adversary, even if this adversary manages to break into the server (Similarly, we have to require that during a break-in, the attacker cannot modify the server's view of other servers' public keys). 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 the private key or modify any of the keys. We will remove this assumption and deal with the proactive maintenance of the private/public communication key pairs in section 6.

The security of this public key system is essential for our protocols, because all our communication is implemented as an authenticated broadcast on C, i.e. every message m sent by P_i will have a signature $SIG_i(m)$ attached to it. Also, whenever server P_i will need to send m "privately" to P_j, it will broadcast $m' = (i, j, ENC_j(m))$ (accompanied, of course, by a signature $SIG_i(m')$).

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. $x = f(0)$. We assume that this initialization has been carried out securely (For an example of secure distributed initialization of secret sharing, we refer the reader to [11] or [17]).

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 $t = 1, 2, \ldots$, 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 $f^{(t)}(0) = f^{(t-1)}(0) + \delta(0) = x + 0 = x$. We can renew the shares $x_i^{(t)} = f^{(t)}(i)$ thanks to the linearity of the polynomial evaluation operation:

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

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 the beginning of the 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. (A solution against "active cheaters", i.e. in the presence of *byzantine* faults, is presented in section 3.4). Notice that we assume in step 2 that the shares are transmitted to the corresponding holders with perfect secrecy. Equivalently, we can specify that every u_{ji} are broadcasted to P_i on C in encrypted form (i.e. as $ENC_i(u_{ji})$), where encryption operation $ENC_i(\cdot)$ is a black-box encryption, giving perfect secrecy to those that don't know the secret key of P_i. 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 3.1 *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 interpolate to the secret x).

Secrecy: An adversary that at any time period knows no more than k shares learns nothing about the secret.

Proof: We proceed by an inductive argument: Assume that at initialization the shares correspond to the secret x according to Shamir's scheme. Furthermore, assume that at each time period $r = 1, 2, .., t-1$ the theorem holds. In particular it means that after the update phase of time period $t-1$, the shares correspond to the secret and that the adversary has learned nothing about the secret. We prove that this condition is preserved during time period t:

Correctness: Let S be a set of $k+1$ shares resultant from the t-th update period. For simplicity of notation, assume $S = \{x_1^{(t)}, x_2^{(t)}, \ldots, x_{k+1}^{(t)}\}$. Let $a_1, a_2, \ldots, a_{k+1}$ be the interpolation coefficients such that $\sum_{i=1}^{k+1} a_i x_i^{(t)}$ would recover the secret using Shamir's scheme (the coefficients a_i depend on the particular indices of shares in the set S). We have:

$$\begin{aligned}
\sum_{i=1}^{k+1} a_i x_i^{(t)} &= \sum_{i=1}^{k+1} a_i \left(x_i^{(t-1)} + \sum_{j=1}^{n} \delta_j(i) \right) && \text{(by step 3 of the protocol)} \\
&= \sum_{i=1}^{k+1} a_i x_i^{(t-1)} + \sum_{j=1}^{n} \sum_{i=1}^{k+1} a_i \delta_j(i) \\
&= x + \sum_{j=1}^{n} \delta_j(0) && \text{(by interpolation)} \\
&= x && \text{(because } \forall j, \delta_j(0) = 0)
\end{aligned}$$

Secrecy: Let A be an eavesdropping adversary. Let K_1 be the set of k_1 servers that A eavesdropped into in period $t-1$ but not in period t; let K_2 be the set of k_2 servers that A eavesdropped into both in period $t-1$ and in period t (we may assume that A eavesdropped into these servers during the update phase); and, let K_3 be the set of k_3 servers that A that eavesdropped into in period t but not in period $t-1$. By our assumption on the adversary, e have $k_1 + k_2 \leq k$ and $k_2 + k_3 \leq k$. We will assume a clear worst case when $k_1 = k_3 = k - k_2$. We also denote by S_1 and S_2 the set of shares in period $t-1$ corresponding to the servers in K_1 and K_2, respectively; and by S_2' and S_3 the set of shares in period t corresponding to the servers in K_2 and K_3, respectively.

We now show that the availability of all this information about shares and updates does not provide information about x.

Notice that since we assume that k shares from period $t-1$ are known, then fixing the secret x determines the interpolation polynomial $f^{(t-1)}$ corresponding to period $t-1$. Similarly, from the k known shares of period t, fixing x determines the polynomial $f^{(t)}$. By construction these polynomials are consistent with the available information from the set of shares S_1, S_2, S_2' and S_3. On the other hand, the difference between these polynomials represent a k-degree polynomial with free coefficient zero and its evaluation on the points

corresponding to the servers in K_2 is consistent with the new shares available to the adversary. (They are consistent also with the value of the partial shares corresponding to the polynomials $\delta_i(z)$ which are randomly chosen conditioned only to $\delta_i(0) = 0$).

Since the above argument holds for *any* value of x, then all possible values of x are consistent with the available information. On the other hand, there is no degree of freedom beyond x (i.e., x and the known shares determine all the additional shares), and hence the distribution on x conditioned on the information available to A is uniform. In other words, no information on x is revealed.

■

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 2.3. 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 $\{\delta_{im}\}_{m\in\{1\ldots k\}}$ 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 $e_{ij} = ENC_j(u_{ij})$, $\forall j \neq i$.

3. P_i broadcast the message $VSS_i^{(t)} = (i, t, \{\epsilon_{im}\}_{m\in\{1\ldots k\}}, \{e_{ij}\}_{j\in\{1\ldots n\}\setminus\{i\}})$, 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 2.3, 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.

3.5 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 $\mathcal{B}_i$ 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:

1. Formally incorrect messages: wrong time periods, numbers out of bounds etc.

2. Two or more correct yet different messages from the same server (i.e. containing a valid signature), or no message at all from some server

3. A mismatch in equation 2.

Notice that irregularities of the first two types are discovered using public information only, and therefore, all (honest) servers can always detect them and mark the corresponding servers as "bad". The faults of the third kind cause a problem, since they are discovered only locally by a server that receives a share causing a mismatch in equation 2.

When server P_i finds that equation 2 corresponding to the information sent by P_j does not hold, it has to broadcast an accusation against P_j. The servers must then decide whether it is P_i or P_j who is cheating. A way to do this is by having P_j publicly "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} which was broadcasted by P_j in step 3. 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 the 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 3. 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:

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

3.6 Security Properties

In Theorem 3.1 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 3.2 *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.

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

4 Share Recovery Scheme

In a proactive secret sharing system, participating servers must be able to make sure whether shares of other participating servers have not been corrupted (or lost), and restore the correct share if necessary. Otherwise, an adversary could cause the loss of the secret by gradually destroying $n - k$ shares. In this section 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

[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 2.3).

secret share. A secret share can also be lost because a server was rebooted or replaced by a new server.

Without share recovery, the proactive scheme would not be secure even against adversaries who can change the local state of the servers they attacked, or in general, in any way disable the attacked server from performing the right protocol (by, for example, disconnecting it from the network). In particular, without this mechanism the scheme is insecure in the presence of hardware failures, crashes of the operating system etc. Also, in a practical system, every server whose misbehavior is detected by others will be rebooted, and its share will be lost because of the reboot procedure.

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 in 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 invariant will also provide robustness in the secret reconstruction protocol (see section 5).

The invariant 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).
- Second, 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 corresponding to servers that did not misbehave in the update phase, i.e:

$$y_j^{(t)} \leftarrow y_j^{(t-1)} * \prod_{a\notin\mathcal{B}_i} g^{u_{aj}} \ (mod\ p)$$

  Also, notice that although the servers in protocol 3.4 do not know update shares u_{aj} of other servers, they can compute their exponents from the information broadcasted

publicly using the equation 2, by computing:

$$g^{u_{aj}} \leftarrow \prod_{m \in \{1 \ldots k\}} (\epsilon_{am})^{j^m} \ (mod\ p)$$

Lost Share Detection Protocol. We extend the *update phase* between time periods to include a *share recovery* protocol executed before the *share renewal* protocol. Its first part is the *lost share detection protocol* which works as follows: Every server checks whether its share $x_i^{(t)}$ corresponds to the $y_i^{(t)}$ it stores, i.e. whether $g^{x_i} \stackrel{?}{=} y_i \ (mod\ p)$. If not, it broadcasts a singed request saying that it needs a share recovery. Otherwise, if its x_i and y_i are consistent, P_i broadcasts the values $\{y_j^{(t)}\}_{j \in \{1 \ldots n\}}$ it stores, together with a proper signature. After collecting these messages from all servers and checking their signatures, each server decides by majority on the current proper set $\{y_j^{(t)}\}_{j \in \{1 \ldots n\}}$ (correcting its own set if necessary). 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 addition to servers that broadcasted an explicit request to have their shares recovered (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., $\mathcal{B}_i = \mathcal{B}_j = \mathcal{B}$. From our assumptions about the adversary, there are no more than k servers holding a wrong share at the end of each time period, i.e. $|\mathcal{B}| \leq k$.

4.2 Basic Share Recovery Protocol

The share recovery algorithm is based on the fact that in Shamir's $(k+1, n)$-threshold scheme, any group $\mathcal{D} \subset \mathcal{A}$ of d shares $(k+1 \leq d \leq n-1)$ can be thought of as a $(k+1, d)$-threshold secret sharing of any of the remaining shares $x_r, r \notin \mathcal{D}$.

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, which would allow P_r to recover the whole polynomial $f^{(t)}(\cdot)$ and $f^{(t)}(r)$ in particular. However, this would also expose the secret x to P_r. Instead, for each $r \in \mathcal{B}$, the servers in $\mathcal{D}$ 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 $\delta_i(r) = 0 \ (mod\ q)$. By adding $\delta_i(\cdot)$'s to $f^{(t)}(\cdot)$, 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 can not change the behavior of the servers. 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 $\{\delta_{ij}\}_{j\in\{1\ldots k\}} \subset Z_q$ and then computes $\delta_{i0} = -\sum_{j\in\{1\ldots k\}} \delta_{ij} r^j \ (mod\ q)$.

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:

1. Each $P_i, i \in \mathcal{D}$ picks a random k-degree polynomial $\delta_i(\cdot) \in Z_q[z]$ such that $\delta_i(r) = 0$, i.e., it picks random coefficients $\{\delta_{ij}\}_{j\in\{1\ldots k\}} \subset Z_q$ and then computes $\delta_{i0} = -\sum_{j\in\{1\ldots k\}} \delta_{ij} r^j \ (mod\ q)$.

2. Each P_i verifiably secret-shares its polynomial $\delta_i(\cdot)$ among the set $\mathcal{D}$ using the same mechanism as in the share renewal protocol, i.e., by broadcasting $VSS_i = (i, \{g^{\delta_{im}} \ (mod\ p)\}_{m\in\{0\ldots k\}}, \{ENC_j(\delta_i(j))\}_{j\in\mathcal{D}})$, together with $SIG_i(VSS_i)$.

3. For all servers P_i, P_j in $\mathcal{D}$, P_j checks P_i by locally verifying whether $\delta_i(r) = 0 \ (mod\ q)$:

$$\prod_{m\in\{0\ldots k\}} (g^{\delta_{im}})^{r^m} \stackrel{?}{=} 1 \ (mod\ p) \tag{3}$$

   and whether $\delta_i(j)$ is consistent with exponents of the coefficients of $\delta_i(\cdot)$:

$$g^{\delta_i(j)} \stackrel{?}{=} \prod_{m\in\{0\ldots k\}} (g^{\delta_{im}})^{j^m} \ (mod\ p) \tag{4}$$

4. Depending on the above verification the servers broadcast acknowledgments (if both equations agree) or start accusation protocols (if equation 4 does not hold). By public resolution of the accusations, each server in $\mathcal{D}$ and the server P_r decide on set $\mathcal{D}' \subset \mathcal{D}$ of servers that properly constructed and distributed their re-randomization polynomials $\delta_i(\cdot)$. As in the share-renewal protocol, all honest servers (including the recovering server P_r) will arrive at the same set $\mathcal{D}'$.

5. Each server $P_i, i \in \mathcal{D}'$ creates its new share of x_r, $x_i' = x_i + \sum_{j\in\mathcal{D}'} \delta_j(i)$ and sends it encrypted and signed to P_r by broadcasting $REC_i = (i, ENC_r(x_i'))$ and $SIG_i(REC_i)$ on the communication channel.

6. P_r decrypts all the x_i's and takes the exponents $\{g^{\delta_{jm}} \ (mod\ p)\}_{j\in\mathcal{D}',m\in\{0\ldots k\}}$ that were broadcasted in step (2). Then, for all $i \in \mathcal{D}'$, P_r takes the current valid exponent y_i of P_i's share (P_r knows it from the lost share detection protocol) and verifies whether:

$$g^{x_i'} \stackrel{?}{=} y_i * \prod_{j\in\mathcal{D}'} g^{\delta_j(i)} = g^{x_i+\sum_{j\in\mathcal{D}'}\delta_j(i)} \ (mod\ p) \tag{5}$$

where for all $j \in \mathcal{D}'$:

$$g^{\delta_j(i)} \leftarrow \prod_{m\in\{0\ldots k\}} (g^{\delta_{jm}})^{(i^m)} \ (mod\ p)$$

7. In this way P_r arrives at a set $\mathcal{D}'' \subseteq \mathcal{D}'$ of servers that during step (5) broadcasted correct new shares x_i'. Now P_r can interpolate these shares to recover x_r, because from our assumptions on the adversary, $|\mathcal{D}''| \geq k+1$

We sketch the properties of this protocol in the following theorem:

Theorem 4.1 *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.

Multi-Secret Sharing. Theoretically, instead of recovering each lost share separately by treating the set $\{x_i\}_{i\in\mathcal{D}}$ as a secret sharing of a single x_r for each $r \in \mathcal{B}$, we can treat it as a multi-secret sharing (introduced in [20]) of all $\{x_r\}_{r\in\mathcal{B}}$. The servers $\mathcal{D}$ can recover shares $\{x_r\}_{r\in\mathcal{B}}$ simultaneously, by adding random k-degree polynomials $\delta_i(\cdot)$ such that $\delta_i(r) = 0$ for all $r \in \mathcal{B}$ to their shares and then sending the new shares to servers $\mathcal{B}$ to let them reconstruct their original shares. Even though this "simultaneous" solution allows servers $\mathcal{B}$ to learn each other's shares $\{x_r\}_{r\in\mathcal{B}}$, this scheme will be secure against the adversary we specified in section 2.1: Since we do not distinguish between an adversary that destroys the share and the adversary that learns it, if the servers $\mathcal{B}$ need a share recovery, we can assume their shares are known to the adversary already.

However, in practice, such a solution obviously weakens the security of the system: Even though we do not specify it formally, our proposed solution is secure if in every time period an adversary manages to destroy the shares of k servers without learning them (e.g. by crashing the servers) *and* simultaneously manages to learn k other shares (e.g. by injecting memory-scanning viruses into the servers that store them).

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, the participants in the actual reconstruction protocol (namely, polynomial interpolation as in Shamir's scheme) must be able to detect servers that provide incorrect shares to the reconstruction. This detection is easily accomplished by verification of the submitted shares against values $y_i^{(t)}$ held by each server (because majority of servers stores the same correct set $\{y_i^{(t)}\}_{i\in\{1\ldots n\}}$).

6 Dynamically Secure Private Keys

In the above presentation we have assumed for simplicity that the servers are equipped with ideally protected private / public key pairs used for authentication and encryption of server-to-server communication (see section 3.1). We now show how to remove this protected key assumption. 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 $a_i^{(t)}, b_i^{(t)}$ and broadcasts the new public key $b_i^{(t)}$ authenticated by its signature using its *previous* private key $a_i^{(t-1)}$. The other servers can verify this signature, using $b_i^{(t-1)}$ from the previous time period. 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, which will constitute a public proof of P_i's compromise and must trigger a reboot procedure.

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. At the same time, public keys $\{a_i^{(t)}\}$ of other servers must be installed on the rebooted server. Notice that installing these public authentication keys requires the same degree of trust in the system management as during the initialization of the system.

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:

1. the private key renewal protocol
2. the share recovery protocol (including lost share detection)
3. the share renewal protocol.

The following theorem summarizes our main result:

Theorem 7.1 *If there are no more than k corrupted servers in each period (where servers compromised during an update phase are considered as compromised in both adjacent periods), then the above protocol constitutes a secure and robust proactive secret sharing scheme.*

8 Applications

Proactive secret sharing protocol offers a way of maintaining sensitive information that provides a novel degree of protecting secrecy and integrity.

Instead of direct proactive secret sharing, one could protect the sensitive data by storing it encrypted on multiple servers and proactively secret sharing only the decryption key. To provide integrity, the servers storing the ciphertext of the data would have to periodically perform a *corruption detection protocol* that would be very similar to the *lost share detection protocol*: The servers would compare the hashes of their ciphertexts and decide by majority on the correct version. Since the ciphertext itself is not secret, the servers that have a correct version would just send it to those that lost it.

New important applications of proactive secret sharing are possible when it is extended to *proactive function sharing* (see our forthcoming paper [14]). The idea is that the (proactively maintained) shares are never combined to reconstruct a single secret, but can be instead repeatedly used to collectively compute on any given input a function defined by this shared secret. [3] (This follows the *function sharing* model of [7] based on threshold encryption [8]). Proactive sharing of the decryption function could be employed to maintain secure databases.

A particularly attractive application (presented in [14]) of proactive function sharing 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.

[3] For example, number e can define an exponentiation function $f : Z_p \rightarrow Z_p, \quad f(x) = x^e \ (mod\ p)$

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.

[20] M. Yung, *Communication Complexity of Secure Computation*, STOC'92.