Communication-Optimal Proactive Secret Sharing for Dynamic Groups

Joshua Baron[4], Karim El Defrawy[1], Joshua Lampkins[1,3(✉)], and Rafail Ostrovsky[2,3]

[1] Information and Systems Sciences Laboratory (ISSL), HRL Laboratories, Malibu, CA 90265, USA
kmeldefrawy@hrl.com

[2] Department of Computer Science, UCLA, Los Angeles, CA 90095, USA
rafail@cs.ucla.edu

[3] Department of Mathematics, UCLA, Los Angeles, CA 90095, USA
jlampkins@math.ucla.edu

[4] RAND Corporation, Santa Monica, CA 90401, USA
jbaron@rand.org

Abstract. *Proactive* secret sharing (PSS) schemes are designed for settings where long-term confidentiality of secrets is required, specifically, when *all participating parties may eventually be corrupted*. PSS schemes periodically refresh secrets and reset corrupted parties to an uncorrupted state; in PSS the corruption threshold of parties is replaced with a corruption *rate* which cannot be violated. In *dynamic proactive secret sharing* (DPSS) the group of participating parties can vary during the course of execution. Accordingly, DPSS is ideal when the set of participating parties changes over the lifetime of the secret or where removal of parties is necessary if they become severely corrupted. This paper presents the first DPSS scheme with optimal amortized per-secret communication in the number of parties, n: This paper requires $O(1)$ communication, as compared to $O(n^4)$ or $\exp(n)$ in previous work. We present perfectly and statistically secure schemes with near-optimal threshold in each case. We also describe how to construct a communication-efficient dynamic proactively-secure multiparty computation (DPMPC) protocol which achieves the same thresholds.

Keywords: Proactive security · Secret sharing · Mobile secret sharing · Dynamic groups · Secure multiparty computation

1 Introduction

Secret sharing [6,31] is a foundational primitive in cryptography, especially in secure computation. A secret sharing scheme typically consists of a protocol for sharing a secret (or multiple secrets) and a protocol for reconstructing the

J. Baron—Work performed while at HRL Laboratories.

J. Lampkins—Work performed while at HRL Laboratories.

© Springer International Publishing Switzerland 2015
T. Malkin et al. (Eds.): ACNS 2015, LNCS 9092, pp. 23–41, 2015.
DOI: 10.1007/978-3-319-28166-7_2

shared secret(s). The secret sharing protocol distributes shares of the secret among n parties in the presence of an adversary who may corrupt up to t parties; security of the secret sharing scheme ensures that such an adversary will learn no information about the secret.

However, traditional secret sharing may be insufficient in some real-world settings; specifically, settings that may require a secret to be secured for a long period of time, especially with respect to the ability of an adversary to *eventually corrupt all parties*. Traditional (threshold-based) secret sharing schemes are insecure once $t + 1$ parties have been corrupted. Of particular concern are distributed storage and computing settings in the presence of advanced persistent threats who, given sufficient time, will successfully corrupt enough parties to break the threshold that guarantees security. To address this issue, Ostrovsky and Yung [28] introduced the *proactive security* model. In this model, the execution of the protocol(s) is divided into phases. The adversary is allowed to corrupt and decorrupt parties at will, under the constraint that no more than a threshold number of parties are corrupt in any given phase. This means that every party may eventually become corrupt subject to the corruption rate constraint. Such an adversary is called a *mobile* adversary.[1]

While standard proactively-secure protocols are able to satisfy security requirements of long-term storage and computation, they lack the ability to change the number of parties during the course of the protocol. Such a restraint is particularly challenging in the case of long-term storage or computation, which was one of the reasons that the proactive security model was constructed in the first place. We refer to secret sharing schemes that are both proactively-secure and allow the set of parties to dynamically change as *dynamic proactive secret sharing* (DPSS) schemes. Such schemes have also been the subject of numerous papers [17,30,33,34] but none of them has satisfying (linear or constant) communication complexity. The dynamic setting allows for the reality that some parties (deployed as physical or virtual servers) may be attacked to the point of not being able to be reset to a pristine, uncorrupted state (e.g., they may become physically damaged). When the set of parties can be dynamically changed, this issue could be addressed by excluding the severely corrupted one(s) entirely (and, ideally, include new uncorrupted ones). In addition, DPSS within large distributed systems enables a truly “moving target defense”, where the set of participating nodes is a smaller, dynamically changing subset of the whole distributed system that is therefore more difficult to target for attack.

We argue that adopting efficient DPSS schemes in the future may help prevent large-scale compromises of servers that store user data, often at financial institutions or large enterprises [27,32]. Such breaches show an increasing need for secure long-term storage solutions. Standard secret sharing can address this issue by distributing data to avoid single points of compromise or failure, but

[1] The term “mobile” is heavily used in the secure computation literature: “dynamic” secret sharing, as discussed in this paper, has historically been called “mobile” secret sharing (for instance, see [30]), which is a completely different concept than the mobile adversary definition.

given enough time, an adversary may be able to compromise all the servers that store the data. Proactive secret sharing partially addresses this issue by refreshing and recovering, yet still has no means of securing against a server that becomes "permanently" compromised (e.g., by compromising its boot system and/or firmware). Dynamic proactive secret sharing addresses this issue by allowing the set of servers to change dynamically in response to corruptions and removing permanently compromised servers. Furthermore, the total number of servers may change, thereby increasing the concrete number of servers that would have to be corrupted to exceed the threshold corruption rate. Thus in response to an attack, the threshold may be temporarily raised to increase security, and when the attack is resolved, the threshold may be reduced by reducing the number of participating servers to increase efficiency. Our goal is therefore to construct a communication-efficient DPSS scheme, particularly one that can be used as a building block in a system for storing large data files and where the proactive refresh and recovery of shares becomes a performance bottleneck when the number of parties (or servers) increases. While we acknowledge that several other layers of security must be developed for a complete data storage solution to be secure against a mobile adversary, we argue that constructing an efficient DPSS is an important step towards realizing this goal though.

1.1 Techniques

We first briefly outline the techniques utilized in the rest of the paper.

Batched Secret Sharing. One of the foundational techniques allowing us to achieve optimal amortized communication complexity is batched secret sharing [21]. Such sharings are used to encode a "batch" of multiple secrets as distinct points on a single polynomial, and then distribute shares to each party as in standard Shamir secret sharing [31]. The number of secrets stored in the polynomial (the "batch size") is chosen to be $O(n)$. This allows the parties to share $O(n)$ secrets with $O(n)$ total communication complexity so that the amortized complexity is $O(1)$ per secret.

Hyper-Invertible Matrices. A hyper-invertible matrix [5] satisfies the property that any square submatrix formed by removing rows and columns is invertible. Hyper-invertible matrices are used in our protocol for efficient error detection. If a vector of $n - 3t$ secret sharings is concatenated with t random sharings and then multiplied by a $n \times (n - 2t)$ hyper-invertible matrix, then each party can be given one of the sharings in the resultant vector of n sharings without revealing any information about the $n - 3t$ secrets. Furthermore, if any of the original $n - 2t$ sharings are malformed (meaning that the shares do not lie on a polynomial of correct degree), then at least $2t + 1$ of the resultant n sharings will be malformed. This allows the parties to verify that sharings are correct while preserving the privacy of the secrets. Since $n - 3t$ (which is $O(n)$) sharings are verified by sending n (also $O(n)$) sharings to n parties, this only requires constant amortized communication bandwidth.

Party Virtualization. Party virtualization [8] is a method for transforming a multiparty protocol by replacing each party in the protocol with a "virtual" party. The virtual party is a committee of parties that perform a multiparty protocol to emulate the actions of an individual party in the original (untransformed) protocol. The advantage of this technique is that it allows the corruption threshold to be raised from that of the untransformed protocol. In [15], the authors demonstrate how to raise the corruption threshold to near-optimal while only increasing the communication complexity by a constant factor, which is the approach we take in this paper.

1.2 Contributions

In this paper we present a new communication-optimal *dynamic proactive secret sharing (DPSS)* scheme. In addition to a protocol for distributing shares of a secret and a protocol for reconstructing the secret, a DPSS scheme must also contain a protocol for *refreshing* the shares and (in the case of a malicious adversary) for *recovering* the shares amongst a group of parties that may change from one refresh to the next. A refresh protocol changes the shares held by the parties such that old shares (before the refresh) cannot be combined with new shares (after the refresh) to gain any information about the secret. A recovery protocol allows decorrupted parties to recover shares that may have been destroyed or altered by the adversary. The communication complexity of the refresh and recovery protocols are often a bottleneck for proactive secret sharing schemes.

As will be defined in Sect. 4.1 (Definition 4), a DPSS scheme consists of three protocols: Share, Redistribute, and Open that distribute, redistribute, and reconstruct shares of a secret, respectively. For the protocols Share and Open, we use the protocols RobustShare and Reco (respectively) from [15].

Our main contribution is the construction of a new Redistribute protocol with the following properties: (1) *Optimal (Constant Amortized) Communication Bandwidth:* Out of currently published protocols for DPSS, ours has the lowest amortized communication complexity. We achieve $O(1)$ per-secret amortized communication complexity (measured as the number of field elements).[2] (2) *No Cryptographic Assumptions:* Ours is the first DPSS scheme that provides information-theoretic security without making any cryptographic assumptions. (3) *Eliminating Party Virtualization:* The most efficient DPSS protocol to date is that of [30] where "party virtualization" is utilized when the set of parties is decreased.[3] "Party virtualization" occurs when each real party holds internal data (i.e., shares) corresponding to some virtual party. That is, there are n

[2] We only claim that the amortized communication complexity is optimal. Reducing the non-amortized complexity is a possible area for future work.

[3] Note that the term "party virtualization" has a different meaning in [30] than is typically used, either in Sect. 1.1 or in other secure computation literature such as [15]; we use here the terminology of [30] in quotes and only in this paragraph.

parties, but there are $n + v$ virtual parties, and while each real party gets her own private share, each real party also gets all v shares of all the virtual parties. As stated in [30], this technique is "somewhat unsatisfying theoretically because using this method to reduce the threshold does not reduce the asymptotic computational overhead of the protocol." In this paper, we present a DPSS protocol that does not use party virtualization as in [30] and thus reduces the asymptotic computational and communication overhead of the protocol.

Finally, as an application of our DPSS scheme we briefly describe how to construct a dynamic proactively-secure multiparty computation (DPMPC) protocol.

1.3 Outline

The rest of the paper is organized as follows: In Sect. 2 we discuss related work. The roadblocks facing constructing an efficient DPSS scheme are described in Sect. 3. We give the necessary technical preliminaries in Sect. 4 and then give the details of our DPSS scheme in Sect. 5 (while some of the subprotocols are deferred to the full version of this paper [4]). In Sect. 6 we describe how the threshold may be raised in the statistical security setting. We show how our DPSS scheme can be applied to secure multiparty computation in Sect. 7. Security definitions and proofs are given in the full version of this paper [4].

2 Related Work

The same work [28] introducing the proactive security model also contained the first proactive secret sharing (PSS) scheme and proactively-secure multiparty computation (PMPC) protocol. PSS was the central tool introduced in [28], and there has been significant follow up work on PSS schemes, both in the synchronous and asynchronous network models (see Table 1 for a comparison). Currently the most efficient (non-dynamic) PSS scheme is [3], which has an optimal $O(1)$ amortized communication complexity per secret share, is UC-secure and achieves near-optimal thresholds for both perfect and statistical cases. Currently, the most efficient DPSS scheme is that of [30], which works in asynchronous networks, provides cryptographic security and achieves a corruption threshold of $t/n < 1/3$, but has prohibitive communication complexity in the number of parties, namely $O(n^4)$. Compared to [30], our DPSS protocols require only constant (amortized) communication are perfectly (resp. statistically) secure with near-optimal corruption thresholds of $t/n < 1/3 - \epsilon$ (resp. $t/n < 1/2 - \epsilon$) and work with synchronous networks. Extending our work to asynchronous networks and improving the threshold and communication bounds of [30] is still an open problem.

In addition to proactive secret sharing, proactive security has played a fundamental role in several areas, including proactively secure threshold encryption and signature schemes [7,10,18–20,25,26,29] (and in particular [1], which also sketches a definition of UC security in the proactive framework), intrusion-resilient signatures [24], eavesdropping games [22], pseudorandomness [11], and state-machine replication [12,13].

Table 1. Comparison of Non-Dynamic Proactive Secret Sharing (PSS) and Dynamic Proactive Secret Sharing (DPSS) Schemes. Threshold is for each reboot phase. Our communication complexity is amortized per bit.

Paper	Dynamic	Network	Security	Threshold	Communication complexity
[33]	Yes	synch.	Cryptographic	$t/n < 1/2$	$\exp(n)$
[34]	Yes	asynch.	Cryptographic	$t/n < 1/3$	$\exp(n)$
[9]	No	asynch.	Cryptographic	$t/n < 1/3$	$O(n^4)$
[30]	Yes	asynch.	Cryptographic	$t/n < 1/3$	$O(n^4)$
[23]	No	synch.	Cryptographic	$t/n < 1/2$	$O(n^2)$
[3]	No	synch.	Perfect	$t/n < 1/3-\epsilon$	$O(1)$
[3]	No	synch.	Statistical	$t/n < 1/2-\epsilon$	$O(1)$
This paper	Yes	synch.	Perfect	$t/n < 1/3-\epsilon$	$O(1)$
This paper	Yes	synch	Statistical	$t/n < 1/2-\epsilon$	$O(1)$

The only two known general PMPC protocols are [28] and [3]. The former protocol is proven secure in the stand-alone corruption model and requires at least $O(Cn^3)$ communication complexity (where C is the size of the circuit), while the latter is UC-secure and has near-linear communication complexity of $O(DC\log^2(C)\text{polylog}(n) + D\,\text{poly}(n)\log^2(C))$ (where D is the depth of the circuit). We provide a dynamic PMPC protocol in this paper, whereas neither of the above PMPC protocols is dynamic.

3 Roadblocks in Constructing Communication-Optimal DPSS

The most efficient DPSS scheme to date is that of [30], and the most efficient PSS scheme to date is that of [3]. In this section, we explain why straightforward modifications of either of these would not produce a DPSS scheme with optimal communication requirements.

In [3], the refresh is performed by having the parties generate new polynomials Q to mask the old polynomials H; then each party generates a share of the new polynomial by locally computing her share of $H + Q$ and relabeling $H \leftarrow H + Q$. Although this works in the non-dynamic proactive setting, in the dynamic proactive setting this would allow t corrupt parties in the old group and an additional t' corrupt parties in the new group to learn their shares on the new polynomial (where t' is the corruption threshold in the new group). This could be enough for the adversary to reconstruct the secret(s) rendering the scheme insecure.

In [30], this issue is prevented by constructing the polynomial Q such that no party in the old group knows her share of Q. More specifically, the parties in the old group construct a polynomial R_j for each P'_j in the new group such that $R_j(\beta_j) = 0$. Then the Q and the R_j are generated simultaneously so that each party in the old group only learns her share of $Q + R_j$ for each j. This technique preserves security but would not yield the optimal communication bandwidth that we aim for. Generating one polynomial for each party in the new group would result in a communication complexity of at least $O(n^2)$ for masking $O(n)$ secrets while our goal is $O(1)$ (amortized) communication per secret.

In this paper we provide a solution that generates the polynomials Q without revealing any share of Q to the parties in the old group, and maintains optimal communication efficiency. This technique is one of the main contributions of the paper and is described in detail in Sect. 5.2.

4 Preliminaries

In this section we provide some preliminaries required for the rest of the paper.

4.1 Definitions

We first provide definitions of secret sharing (SS), proactive secret sharing (PSS), and dynamic proactive secret sharing (DPSS) schemes. The definitions below are for perfectly secure protocols; the definitions for statistically secure protocols are the same, except that the termination, correctness, and secrecy properties are allowed to be violated with negligible probability. As our protocols are for sharings of multiple secrets, we write the protocols for a vector of secrets over a finite field $\mathbb{F}$, treating the case in which the vector is of length one as a special case.

Definition 1. *A* secret sharing scheme *consists of two protocols,* Share *and* Open*, which allows a dealer to share a vector of secrets* $\mathbf{s}$ *among a group of* n *parties such that the secrets remain secure against an adversary, and allows any group of* $n - t$ *uncorrupted parties to reconstruct the secrets.*

Assuming that no more than t *parties are corrupt throughout the execution of the protocols, the following three properties hold:*

- *Termination: All honest parties will complete the execution of* Share *and* Open*.*
- *Correctness: Upon completing* Share*, there is a fixed vector* $\mathbf{v} \in \mathbb{F}^W$ *(where* W *is the number of secrets to be shared) such that all honest parties will output* $\mathbf{v}$ *upon completion of* Open*. Furthermore, if the dealer was honest during the execution of* Share*, then* $\mathbf{v} = \mathbf{s}$*.*
- *Secrecy: If the dealer is uncorrupted, then the adversary gains no information on* $\mathbf{s}$*.*

The definition of a PSS scheme is essentially the same as the definition of an SS scheme, with the addition of Refresh and Recovery protocols for securing against a mobile adversary. The Refresh protocol refreshes data to prevent a mobile adversary from learning secrets, and the Recovery protocols allows decorrupted parties to recover their secrets, preventing the adversary from destroying data. Before defining a PSS scheme, we need to define refresh and recovery phases.

Definition 2. *A* refresh phase *(resp.* recovery phase*) is the period of time between two consecutive executions of the* Refresh *(resp.* Recovery*) protocol. Furthermore, the period between* Share *and the first* Refresh *(resp.* Recovery*) is a phase, and the period between the last* Refresh *(resp.* Recovery*) and* Open *is a phase. Any* Refresh *(resp.* Recovery*) protocol is considered to be in both adjacent phases.*

Definition 3. *A* proactive secret sharing scheme *consists of four protocols,* Share*,* Refresh*,* Recover*, and* Open*, which allows a dealer to share a vector of secrets* $\mathbf{s}$ *among a group of* n *parties such that the secrets remain secure against a mobile adversary, and allows any group of* $n-t$ *uncorrupted parties to reconstruct the secrets. The* Refresh *protocol prevents the mobile adversary from discovering the secrets, and the* Recover *protocol prevents the adversary from destroying the secrets.*

Assuming that no more than t *parties are corrupt during any recovery phase, the following two properties hold:*

- *Termination: All honest parties will complete each execution of* Share*,* Refresh*,* Recover*, and* Open*.*
- *Correctness: Same as in Definition 1.*

Assuming that no more than t *parties are corrupt during any refresh phase, the following property holds:*

- *Secrecy: Same as in Definition 1.*

For the definition of a DPSS scheme, we combine the Refresh and Recover protocols into one protocol, Redistribute, which also allows transferring the set of secrets from one group of parties to another and change the threshold. Similarly, we combine *refresh phase* and *recovery phase*, and refer to it simply as a *phase*.

As the number of parties changes, the threshold must change as well. For any given number of parties, n, there is a corresponding threshold, t, which will depend on the particular security and network assumptions of the scheme. Let $\tau(n)$ denote the threshold corresponding to n, and let $n^{(i)}$ denote the number of parties during phase i.

Definition 4. *A* dynamic proactive secret sharing scheme *consists of three protocols,* Share*,* Redistribute*, and* Open*, which allows a dealer to share a vector of secrets* $\mathbf{s}$ *among a group of* $n^{(1)}$ *parties such that the secrets remain secure against a mobile adversary, and allows any group of* $n^{(L)} - t^{(L)}$ *uncorrupted parties to reconstruct the secrets (where* L *is the last phase). The* Redistribute *protocol prevents the mobile adversary from discovering or destroying the secrets, and allows the set of parties and the threshold to change.*

Assuming that for each i, no more than $t^{(i)} = \tau(n^{(i)})$ parties are corrupt during phase i, the following three properties hold:

- *Termination: All honest parties currently engaged in the protocol will complete each execution of* Share*,* Redistribute*, and* Open*.*
- *Correctness: Same as in Definition 1.*
- *Secrecy: Same as in Definition 1.*

4.2 Notation and Technical Details

We assume that there are W secrets in some finite field $\mathbb{F}$ stored among a party set $\mathcal{P}$ of size n. The secrets are stored as follows:

We fix some generator ζ of $\mathbb{F}^*$. Each batch of ℓ secrets is stored in a polynomial H of degree d (where the value of d depends on the security model as described below). The polynomial H is chosen such that $H(\zeta^j)$ is the j^{th} secret for $j \in [\ell]$ and $H(\zeta^{\ell+j})$ is random for $j \in [d-\ell+1]$. (We use the notation $[X]$ to denote the set $\{1, \ldots, X\}$, and we let $[X] \times [Y]$ denote the Cartesian product of the two sets. We let $[A, B]$ denote the set of integers $[A, \ldots, B]$). Each party $P_i \in \mathcal{P}$ is given $H(\alpha_i)$ as her share of the secret. In our scheme we use the protocol RobustShare from [15] to perform the sharing. When the secrets are to be opened, all parties send their shares to some party, who interpolates the shares on the polynomials to reconstruct the secrets. We use the protocol Reco from [15] to perform secret opening.

Our new redistribution protocol given in Sect. 5 redistributes the secrets to a new set of parties $\mathcal{P}'$ of size n'. The parties in $\mathcal{P}'$ are denoted by P'_j for $j \in [n']$. The share of a party $P'_j \in \mathcal{P}'$ is $H(\beta_j)$. We require that $\alpha_i \neq \beta_j$ for each i, j (and that no α_i or β_j is equal to ζ^k for any $k \in [\ell]$). Since we use the labels t, ℓ, and d for $\mathcal{P}$, we use the labels t', ℓ', and d' for $\mathcal{P}'$.

For simplicity of notation, our redistribution protocol below assumes that W is a multiple of $4\ell^2(n-3t)$. If W is not a multiple of $4\ell^2(n-3t)$, we can generate random sharings of batches to make it so. Using RanDouSha from [15], this can be done with poly(n) communication complexity, and since it adds only a poly(n) amount of data to W, this does not affect the overall communication complexity of redistributing W secrets.

In this paper we provide a perfectly secure and a statistically secure version of the redistribution protocol required to construct our DPSS scheme. For the perfectly (statistically) secure protocol, the threshold can be made arbitrarily close to $n/3$ ($n/2$). We describe the threshold, batch size, and degree of polynomials for the two versions below.

In the perfectly secure protocol, we fix three nonzero constants η, θ, and ι that satisfy $\eta + \theta + \iota < 1/3$. The batch size, ℓ, is the highest power of 2 not greater than $\lfloor \eta n \rfloor$; the threshold is $t = \lfloor \theta n \rfloor$; and the degree of the polynomials that share the secrets are $d = \ell + t + \lfloor \iota n \rfloor - 1$. The number of parties may increase or decrease by no more than a factor of 2 at each redistribution. Furthermore, the number of parties cannot decrease so much that the corrupt parties in the old group can interpolate the new polynomials (i.e., $d' - \ell' \geq t$); and the number of parties cannot increase so much that the uncorrupted parties in the old group cannot interpolate the new polynomials in the presence of corrupt shares (i.e., $d' + 2t + 1 \leq n$).

In the statistically secure protocol, we initially pick a low threshold, and then later raise the threshold using the party virtualization[4] technique of [15]. The protocol in Sect. 5 is written as a perfectly secure protocol with a lower threshold, and then this is raised using statistically secure virtualization (see Sect. 6 for a discussion of this). For the initial, low threshold, we select the batch size, ℓ, to be the highest power of 2 not greater than $n/4$; the threshold is $t < n/16$; and the degree of the polynomials is $d = \ell + 2t - 1$. In the statistically secure version, we assume that t will increase or decrease by a factor of no more than 2 at each redistribution (i.e., $t/2 \leq t' \leq t$).

Note that while (theoretically) it may seem that there is no reason to raise n without raising t, in a real world setting one may increase n while fixing t precisely to increase the concrete number of additional servers that an adversary has to corrupt. To simplify demonstration in this paper we assume that n is minimal for a given t (i.e., we assume that n could not be decreased without decreasing t).

Our redistribution protocol requires the use of a hyper-invertible matrix. A *hyper-invertible* matrix is such that any square submatrix formed by removing rows and columns is invertible. It is shown in [5] that one can construct a hyper-invertible matrix as follows: Pick $2a$ distinct field elements $\theta_1, \ldots, \theta_a, \phi_1, \ldots, \phi_a \in \mathbb{F}$, and let M be the matrix be such that if $(y_1, \ldots, y_a)^T = M(x_1, \ldots, x_a)^T$, then the points $(\theta_1, y_1), \ldots, (\theta_a, y_a)$ lie on the polynomial of degree $\leq a - 1$ which evaluates to x_j at ϕ_j for each $j \in [a]$. (In other words, M interpolates the points with x-coordinates $\theta_1, \ldots, \theta_a$ on a polynomial given the points with x-coordinates $\phi_1, \ldots, \phi_a$ on that polynomial.) Then any submatrix of M is hyper-invertible. For our protocol, we let M be some (publicly known) hyper-invertible matrix with n rows and $n - 2t$ columns.

Throughout the protocol, the Berlekamp-Welch algorithm is used to interpolate polynomials in the presence of corrupt shares introduced by the adversary. As was noted in [16], if M is as above and $\boldsymbol{y} = M\boldsymbol{x}$, then we can also use Berlekamp-Welch to "interpolate" $\boldsymbol{x}$ from $\boldsymbol{y}$ if the adversary corrupts no more than t coordinates of $\boldsymbol{y}$.

5 The Redistribution Protocol

In this section, we provide the details of the protocol that redistributes sharings of secrets from one set of parties to another. The first portion of the protocol changes the threshold of the polynomials that share the secret (if the number of servers is changing). Recall that the batch size is the highest power of two not

[4] The term "party virtualization" has a different meaning in [30] than it has in [15].

greater than $\lfloor \eta n \rfloor$ (resp. $n/4$) in the perfectly (resp. statistically) secure protocol. This means that a change in the threshold/number of servers does not necessarily lead to a change in batch size. Thus there are four cases to consider: (1) The threshold is decreasing, and the batch size is not changing; (2) the threshold is decreasing, and the batch size is decreasing; (3) the threshold is increasing, and the batch size is not changing; and (4) the threshold is increasing, and the batch size is increasing. The second portion of the protocol refreshes the sharings and allows parties in the new group to learn their shares.

To simplify exposition, the protocol is broken into several sub-protocols. The four protocols $\mathsf{Threshold\_Change}_i$ for $i = 1, 2, 3, 4$ correspond to the four cases outlined in the previous paragraph. The protocol Refresh_Recovery performs refresh and recovery.

In order to change the set of parties, the current (honest) parties must agree on which parties to remove and which parties to add. This could be determined by the parties jointly invoking a voting algorithm, by a trusted administrator making the decision, or by following some pre-determined schedule. How exactly this is implemented is beyond the scope of this paper.

We now provide an overview and the intuition behind the operation of the protocol.

5.1 Overview of Threshold Change

To simplify the illustration of the operation of the protocol we will treat $\mathsf{Threshold\_Change}_2$ as an example. In this case we are decreasing the threshold and batch size. Since we restrict the batch size to be a power of 2, the batch size will be cut in half (that is, $\ell' = \ell/2$). If the parties had access to an uncorruptible trusted party, then the parties could have the trusted party change the threshold and batch size for a polynomial H as follows:

1. Each party sends all their shares of the degree d polynomial H to the trusted party.
2. The trusted party constructs two new polynomials h_1 and h_2 of degree d' such that $h_1(\zeta^j) = H(\zeta^j)$ and $h_2(\zeta^j) = H(\zeta^{\ell'+j})$ for each $j \in [\ell']$. Fresh randomness is used for to determine the points $h_i(\zeta^j)$ for $i = 1, 2$ and $j = [\ell' + 1, d' + 1]$.
3. The trusted party sends each party their shares of h_1 and h_2.

In the absence of a trusted party, the parties emulate this simplified protocol using hyper-invertible matrices. The parties will take a vector of $n - 3t$ sharings, add to this t extra random sharings, and then via local computations, multiply the vector by a $n \times n - 2t$ hyper-invertible matrix to get a vector of n sharings. Each party is assigned one of these n sharings and is sent all shares of this sharing from the other parties. Then each party acts as the trusted party in the steps above. The fact that the original vector of $n - 3t$ sharings was padded with an extra t sharings prevents the adversary from learning any information on the secrets.

Once each party is done acting as the trusted party, she then sends the shares of the results to the other parties. Each party, upon receiving the n (or fewer) shares, can apply the Berlekamp-Welch algorithm to interpolate the vector of n shares in the presence of errors to reconstruct the pre-image under multiplication by the hyper-invertible matrix, which is a vector of $n-2t$ shares. The first $n-3t$ of these are taken to be the party's shares of the new sharings.

In the case where the trusted party performs the operations, fresh randomness is generated by the trusted party to use in the new sharings. When the parties jointly perform this operation without a trusted party, they instead generate random sharings R, apply a hyper-invertible matrix to these sharings (as they did with the sharings of the actual secrets), and use the points on the resultant sharings as randomness for the new sharing polynomials.

5.2 Overview of Refresh and Recovery

The protocol Refresh_Recovery is a modification of the protocol Block-Redistribute from [3] that is still secure in the dynamic setting (recall that a straightforward adoption is insecure as discussed in Sect. 3). The recovery is performed in essentially the same way as in [3], with the exception that in our scheme the shares are transferred to a new group of parties instead of back to the same group. (The scheme in [3] is for PSS, not DPSS.)

In the dynamic setting, refresh cannot be performed as in [3]. As mentioned in Sect. 3, we need a way for the parties to mask the polynomials H with polynomials Q such that no party in the old group knows a share of $H+Q$ and no party in the new group knows a share of the original H.[5] In [3], the parties generate sharings U that share their shares, and then each party receives a linear combination of these shares that will allow her to recover her shares (if they were corrupted). In our protocol, the parties in the old group generate sharings U that share their shares (just as in [3]), and they additionally generate sharings V, some of which store random data and some of which store a batch of all zeros; then each party in the new group receives a linear combination of the U's and the V's such that this linear combination stores the party's share of $H+Q$ for some masking polynomial Q. Thus the parties in the new group see their shares of $H+Q$ without seeing their shares of H, while the parties in the old group—because the V were generated randomly—do not know any share of Q (and hence they do not know any share of $H+Q$).

[5] However, if there is overlap between the old and new groups of servers, such that $P_i = P'_j$ for some $P_i \in \mathcal{P}$ and some $P'_j \in \mathcal{P}'$, and if $\alpha_i = \beta_j$, then this party will know her share of both H and $H+Q$. Nevertheless, this does not cause a security problem, as it does not cause the threshold to be violated; even in this case, only t parties in the old group know shares of H, and only t' parties in the new group know shares of $H+Q$.

5.3 Protocol Specification

In this section we describe the specification of our redistribution protocol. As stated in Definition 4, a DPSS scheme consists of three protocols, Share, Redistribute (which we describe in this section), and Open. For the protocols Share and Open, we use the protocols RobustShare and Reco (respectively) from [15]. Our contribution is the construction of the redistribution protocol (Fig. 1).

The protocol RobustShare allows the parties to share $O(n^2)$ secrets with $O(n^2)$ communication complexity using batch sharing. This is accomplished with hyper-invertible matrices to ensure robustness. The protocol Reco opens a batch of secrets by sending each share to whichever party is supposed to learn the secret. That party then performs error detection/correction to interpolate the secrets in the presence of (possibly) corrupt shares. The protocol RanDouSha from [15] is also used as a subprotocol in our redistribution protocol. The protocol RanDouSha generates random sharings of degree d and additional sharings of the same secrets using degree $2d$ polynomials with constant amortized communication bandwidth. However, for our protocols we do not use the degree $2d$ sharings. There are some instances in which we require a variant of RanDouSha that generates sharings of batches of all zeros. Modifying the protocol to do this is straightforward, as is the modification of the security proof.

```
The input to the protocol is a $t, \mathcal{P}, \mathcal{C}orr, t', \mathcal{P}'$ and a collection of polynomials $H_a^{(k,m)}$ for $(a, k, m) \in [\ell] \times [n - 3t] \times [B]$ that store the secrets.
1. If $t' \neq t$, then one of the following steps is executed:
    1.1 If $t' < t$ and $\ell' = \ell$, invoke Threshold_Change$_1$.
    1.2 If $t' < t$ and $\ell' < \ell$, invoke Threshold_Change$_2$.
    1.3 If $t' > t$ and $\ell' = \ell$, invoke Threshold_Change$_3$.
    1.4 If $t' > t$ and $\ell' > \ell$, invoke Threshold_Change$_4$.
2. Invoke Refresh_Recovery.
```

Fig. 1. Redistribute.

As seen in Fig. 1, there are four cases for threshold change. To simplify the treatment we only focus on case 2 (see Fig. 2), which is when the threshold is decreasing and the batch size is decreasing, and defer the other three cases to the full version of this paper [4].

The following subprotocol (Fig. 3) describes how refresh and recovery is performed. This subprotocol will be executed at each redistribution regardless of whether the threshold is changing.

After Refresh_Recovery is completed, the parties relabel the $H_a^{(k,m)}$ again so that k varies from 1 to $n' - 3t'$ instead of $n - 3t$. The relabeling is performed in such a way that it preserves lexicographical order as described in the last steps of protocols Threshold_Change$_2$ and Threshold_Change$_4$.

Lowering the Threshold, Batch Size Decreases

Since we assume that the number of parties decreases by no more than a factor of 2, we know that $\ell' = \ell/2$.

1. The parties invoke RanDouSha to generate masking polynomials $H_a^{(k,m)}$ of degree $\leq d$ for $k \in [n-3t+1, n-2t]$ and $a \in [\ell]$, as well as random polynomials $R_a^{(k,m)}$ of degree $\leq d$ for $k \in [n-2t]$ and $a \in [2\ell]$ (where $m \in [B]$).

2. Define $\widetilde{H}_a^{(k,m)}$ for $k \in [n]$ by

$$\left(\widetilde{H}_a^{(1,m)}, \ldots, \widetilde{H}_a^{(n,m)}\right)^T = M\left(H_a^{(1,m)}, \ldots, H_a^{(n-2t,m)}\right)^T,$$

and similarly define $\widetilde{R}_a^{(k,m)}$ for $k \in [n]$. Each party locally computes their shares of these polynomials and sends his share of each $\widetilde{H}_a^{(j,m)}$ and $\widetilde{R}_a^{(j,m)}$ to party P_j.

3. Each P_i uses Berlekamp-Welch to interpolate the shares of $\widetilde{H}_a^{(i,m)}$ and $\widetilde{R}_a^{(i,m)}$ received in the previous step.

4. Each P_i computes (shares of) the unique polynomials $\widetilde{h}_{2a-1}^{(i,m)}, \widetilde{h}_{2a}^{(i,m)}$ of degree $\leq d'$ for $a \in [\ell]$ and $m \in [B]$ that satisfy the following:

4.1 $\widetilde{h}_{2a-1}^{(i,m)}(\zeta^j) = \widetilde{H}_a^{(i,m)}(\zeta^j)$ for $j \in [\ell']$.

4.2 $\widetilde{h}_{2a-1}^{(i,m)}(\zeta^{\ell'+j}) = \widetilde{R}_{2a-1}^{(i,m)}(\zeta^j)$ for $j \in [d'-\ell'+1]$.

4.3 $\widetilde{h}_{2a}^{(i,m)}(\zeta^j) = \widetilde{H}_a^{(i,m)}(\zeta^{\ell'+j})$ for $j \in [\ell']$.

4.4 $\widetilde{h}_{2a}^{(i,m)}(\zeta^{\ell'+j}) = \widetilde{R}_{2a}^{(i,m)}(\zeta^j)$ for $j \in [d'-\ell'+1]$.

5. Each P_i sends each $\widetilde{h}_a^{(i,m)}(\alpha_j)$ to each P_j.

6. If we define $h_a^{(k,m)}$ to be the unique polynomials of degree $\leq d'$ satisfying

6.1 $h_{2a-1}^{(k,m)}(\zeta^j) = H_a^{(k,m)}(\zeta^j)$ for $j \in [\ell']$,

6.2 $h_{2a-1}^{(k,m)}(\zeta^{\ell'+j}) = R_{2a-1}^{(k,m)}(\zeta^j)$ for $j \in [d'-\ell'+1]$,

6.3 $h_{2a}^{(k,m)}(\zeta^j) = H_a^{(k,m)}(\zeta^{\ell'+j})$ for $j \in [\ell']$,

6.4 $h_{2a}^{(k,m)}(\zeta^{\ell'+j}) = R_{2a}^{(k,m)}(\zeta^j)$ for $j \in [d'-\ell'+1]$,

then it is clear that

$$\left(\widetilde{h}_a^{(1,m)}, \ldots, \widetilde{h}_a^{(n,m)}\right)^T = M\left(h_a^{(1,m)}, \ldots, h_a^{(n-2t,m)}\right)^T.$$

So each party uses Berlekamp-Welch to interpolate their shares of the $h_a^{(k,m)}$ from the shares of the $\widetilde{h}_a^{(k,m)}$ received in the previous step.

7. We place a lexicographical order on the polynomials $H_a^{(k,m)}$ by assigning to the polynomial the vector (m,k,a) and using the lexicographical order on these 3-dimensional vectors to induce an ordering on the polynomials. We similarly place a lexicographical order on the polynomials $h_a^{(k,m)}$. To simplify notation throughout the rest of the protocol, we now relabel $\left\{H_a^{(k,m)}\right\}_{\substack{m=1,\ldots,4B \\ k=1,\ldots,n-3t \\ a=1,\ldots,\ell'}} \leftarrow \left\{h_a^{(k,m)}\right\}_{\substack{m=1,\ldots,B \\ k=1,\ldots,n-3t \\ a=1,\ldots,2\ell}}$ in such a way that this map preserves lexicographical order. We then relabel $B \leftarrow 4B$.

Fig. 2. Threshold_Change$_2$.

1. *Double Sharing Batched Secrets*
   - *1.1* The parties generate sharings of ℓtB random sharings by invoking RanDouSha. We will denote these random secrets by $H_a^{(k,m)}$, where a and m range over the same values as before, but $k \in [n-3t+1, n-2t]$.
   - *1.2* Each party batch-shares all of his shares of each $H_a^{(k,m)}$ using RobustShare. That is, P_i chooses polynomials $U^{(i,1,m)}, \ldots, U^{(i,(n-2t),m)}$ of degree $\leq d'$ such that $U^{(i,k,m)}(\zeta^j) = H_j^{(k,m)}(\alpha_i)$ for $j \in [\ell]$ and $U^{(i,k,m)}(\zeta^{\ell'+j})$ is random for $j \in [d'-\ell'+1]$ and shares them via RobustShare.
2. *Verifying Correctness*
   - *2.1* Define $\widetilde{H}_a^{(k,m)}$ and $\widetilde{U}_a^{(k,m)}$ for $k \in [n]$ by

     $$\left(\widetilde{H}_a^{(1,m)}, \ldots, \widetilde{H}_a^{(n,m)}\right)^T = M\left(H_a^{(1,m)}, \ldots, H_a^{(n-2t,m)}\right)^T$$

     and

     $$\left(\widetilde{U}_a^{(1,m)}, \ldots, \widetilde{U}_a^{(n,m)}\right)^T = M\left(U_a^{(1,m)}, \ldots, U_a^{(n-2t,m)}\right)^T.$$

     Each party in $\mathcal{P}$ locally computes their shares of these polynomials.
   - *2.2* Each party in $\mathcal{P}$ sends *all* their shares of $\widetilde{H}_a^{(k,m)}$ and $\widetilde{U}^{(i,k,m)}$ to party P_k for each a, i, and m.
   - *2.3* Each P_k uses Berlekamp-Welch on the shares of each $\widetilde{U}^{(i,k,m)}$ to interpolate $\widetilde{U}^{(i,k,m)}(\zeta^j)$ for each $j \in [\ell']$.
   - *2.4* Each P_k uses Berlekamp-Welch on the shares of each $\widetilde{H}_a^{(k,m)}$. to interpolate $\widetilde{H}^{(i,k,m)}(\alpha_i)$ for each $i \in [n]$.
   - *2.5* Each P_k checks if the shares of $\widetilde{H}_a^{(k,m)}$ are consistent with the interpolation of the polynomial $\widetilde{U}^{(i,k,m)}$. That is, P_k checks if $\widetilde{U}^{(i,k,m)}(\zeta^j) = \widetilde{H}_j^{(k,m)}(\alpha_i)$ for each $j \in [\ell']$. If some $\widetilde{U}^{(i,k,m)}$ does not pass this check, then P_k sends $(P_k, \texttt{accuse}, P_i)$ to each party in $\mathcal{P}'$.
   - *2.6* Each $P_j' \in \mathcal{P}'$ uses the accusations sent in the previous step to determine a set $\mathcal{C}orr_j'$ of parties in $\mathcal{P}$ that might be corrupt. More specifically, P_j' reads through the list of accusations, and adds parties to $\mathcal{C}orr_j'$ according to the following rule: If neither of the parties in the current accusation are in $\mathcal{C}orr_j'$, then add both of them to $\mathcal{C}orr_j'$; otherwise, ignore the accusation.
3. *Share Transfer*

Fig. 3. Refresh_Recovery.

3.1 Each $P'_j \in \mathcal{P}'$ selects a set G_j of parties in $\mathcal{P} - \mathcal{C}orr_j$ such that $|G_j| = n - 2t$. Then P'_j sends this set to each member of G_j.

3.2 For each $P'_j \in \mathcal{P}'$, let $\{z_1^{(j)}, \ldots, z_{n-2t}^{(j)}\}$ denote the set of indices of parties in G_j. Let $\lambda_{j,i}$ denote the Lagrange coefficients for interpolating P'_j's share of a secret from the shares of parties in G_j (i.e. for a polynomial f of degree $\leq d'$, $f(\beta_j) = \lambda_{j,1} f(\alpha_{z_1^{(j)}}) + \cdots + \lambda_{j,n-2t} f(\alpha_{z_{n-2t}^{(j)}})$).

3.3 The parties in $\mathcal{P}$ execute $\mathsf{RanDouSha}$ to generate degree d' polynomials $V^{(j,k,m)}$ for $(j,k,m) \in [\ell'+1, d'+1] \times [n-3t] \times [B]$. The parties in $\mathcal{P}$ also use $\mathsf{RanDouSha}$ to generate degree d' polynomials $V^{(j,k,m)}$ for $(j,k,m) \in [\ell'] \times [n-3t] \times [B]$ that are random subject to the constraint that $V^{(j,k,m)}(\zeta^w) = 0$ for each $w \in [\ell']$.

3.4 Define degree d' polynomials $Q_a^{(k,m)}$ for $(a,k,m) \in [\ell'] \times [n-3t] \times [B]$ by $Q_a^{(k,m)}(\zeta^w) = 0$ for $w \in [\ell']$ and $Q_a^{(k,m)}(\zeta^w) = V^{(w,k,m)}(\zeta^a)$ for $w \in [\ell'+1, d'+1]$. Let $\mu_{j,i}$ denote the Lagrange coefficients for interpolating P'_j's share of a secret from the points at ζ^i for $i \in [d'+1]$ (i.e. for a polynomial f of degree $\leq d'$, $f(\beta_j) = \mu_{j,1} f(\zeta^1) + \cdots + \mu_{j,d'+1} f(\zeta^{d'+1})$.)

3.5 For each $k \in [n-3t]$, each $m \in [B]$, and each $j \in [n']$, each party in G_j sends his share of

$$\lambda_{j,1} U^{(z_1^{(j)},k,m)} + \cdots + \lambda_{j,n-2t} U^{(z_{n-2t}^{(j)},k,m)} \\ + \mu_{j,1} V^{(1,k,m)} + \cdots + \mu_{j,d'+1} V^{(d'+1,k,m)}$$

to P'_j.

3.6 Each P'_j uses Berlekamp-Welch to interpolate the polynomials received in the previous step for each $k \in [n-3t]$ and each $m \in [B]$. Since for each $a \in [\ell']$,

$$\begin{aligned} &\lambda_{j,1} U^{(z_1^{(j)},k,m)}(\zeta^a) + \cdots + \lambda_{j,n-2t} U^{(z_{n-2t}^{(j)},k,m)}(\zeta^a) \\ &\quad + \mu_{j,1} V^{(1,k,m)}(\zeta^a) + \cdots + \mu_{j,d'+1} V^{(d'+1,k,m)}(\zeta^a) \\ &= \lambda_{j,1} H_a^{(k,m)}(\alpha_{z_1^{(j)}}) + \cdots + \lambda_{j,n-2t} H_a^{(k,m)}(\alpha_{z_{n-2t}^{(j)}}) \\ &\quad + \mu_{j,1} Q_a^{(k,m)}(\zeta^1) + \cdots + \mu_{j,d'+1} Q_a^{(k,m)}(\zeta^{d'+1}) \\ &= H_a^{(k,m)}(\beta_j) + Q_a^{(k,m)}(\beta_j). \end{aligned}$$

P'_j has his share of each batch of refreshed data.

Fig. 3. (*continued*)

6 Party Virtualization

As stated in Sect. 1.2, we do not require "party virtualization" as defined in [30]. However for the statistical version of our protocol, we require the use of a party virtualization technique similar to that in [14] (note that these are different techniques as noted before in Sect. 1.2). The technique, initially introduced in [8], replaces an individual party with a committee of parties that emulates the actions of an individual party. This is done such that the number of corrupt committees is lower than the number of corrupt parties. This allows us to raise the threshold in *the statistical case* from the initial threshold of $t < n/16$ to $t < (1/2 - \epsilon)n$ for arbitrary $\epsilon > 0$. In [2], the authors show how to perform party virtualization such that there is a constant number of communication rounds. We refer the reader to [2,14] for details.

Changing the threshold when party virtualization is used is fairly straightforward. The only requirement is that the threshold of the original (non-virtualized) protocol still satisfies $t < n/16$ when the threshold changes. During redistribution, the parties in the new group will be arranged into committees as in the old group, and shares will be transferred from the virtual parties in the old group to the virtual parties in the new group as specified in [2].

7 Dynamic Proactive Multiparty Computation

Our DPSS scheme can be used to construct a dynamic proactive secure multiparty computation (DPMPC) protocol. A secure multiparty computation (MPC) protocol allows a set of parties to compute a function of their private inputs remaining secure against an adversary who may corrupt some of the parties. A DPMPC protocol is an MPC protocol secure against a mobile adversary in which the set of parties performing the computation and the corruption threshold may change during the course of the protocol.[6]

In [3], the authors show how to proactivize the MPC scheme of [14] by executing a refresh and recovery protocol between each layer of circuit computation. To construct our DPMPC scheme, we execute our Redistribute protocol between each circuit layer as in [3].

Acknowledgements. The fourth author is supported in part by NSF grants 09165174, 1065276, 1118126 and 1136174, US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

[6] Although the set of parties may change throughout the course of the protocol, the inputs of the original set of parties are used to compute the circuit.

References

1. Almansa, J.F., Damgård, I., Nielsen, J.B.: Simplified threshold RSA with adaptive and proactive security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, pp. 593–611. Springer, Heidelberg (2006)
2. Baron, J., El Defrawy, K., Lampkins, J., Ostrovsky, R.: How to withstand mobile virus attacks, revisited (full version). Cryptology ePrint Archive, Report 2013/529 (2013). http://eprint.iacr.org/
3. Baron, J., El Defrawy, K., Lampkins, J., Ostrovsky, R.: How to withstand mobile virus attacks, revisited. In: Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing, PODC 2014, pp. 293–302. ACM, New York (2014)
4. Baron, J., El Defrawy, K., Lampkins, J., Ostrovsky, R.: Communication-optimal proactive secret sharing for dynamic groups (full version). Cryptology ePrint Archive, Report 2015/304 (2015). http://eprint.iacr.org/
5. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure MPC with linear communication complexity. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 213–230. Springer, Heidelberg (2008)
6. Blakley, G.R.: Safeguarding cryptographic keys. In: Proceedings of the AFIPS National Computer Conference, vol. 48, pp. 313–317 (1979)
7. Boldyreva, A.: Threshold Signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: Desmedt, Y.G. (ed.) Public Key Cryptography — PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (2002)
8. Bracha, G.: An O(log n) expected rounds randomized byzantine generals protocol. J. ACM **34**(4), 910–920 (1987)
9. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Asynchronous verifiable secret sharing and proactive cryptosystems. In: ACM Conference on Computer and Communications Security, pp. 88–97 (2002)
10. Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive security for threshold cryptosystems. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 98–115. Springer, Heidelberg (1999)
11. Canetti, R., Herzberg, A.: Maintaining security in the presence of transient faults. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 425–438. Springer, Heidelberg (1994)
12. Castro, M., Liskov, B.: Proactive recovery in a byzantine-fault-tolerant system. In: OSDI, pp. 273–288 (2000)
13. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. **20**(4), 398–461 (2002)
14. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 445–465. Springer, Heidelberg (2010)
15. Damgård, I., Ishai, Y., Krøigaard, M., Nielsen, J.B., Smith, A.: Scalable multiparty computation with nearly optimal work and resilience. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 241–261. Springer, Heidelberg (2008)
16. Damgård, I.B., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer, Heidelberg (2007)
17. Desmedt, Y., Jajodia, S.: Redistributing secret shares to new access structures and its applications. Technical Report ISSE TR-97-01, George Mason University, July 1997

18. Frankel, Y., Gemmell, P., MacKenzie, P.D., Yung, M.: Optimal-resilience proactive public-key cryptosystems. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, FOCS 1997, pp. 384, Washington, DC, USA. IEEE Computer Society (1997)
19. Frankel, Y., Gemmell, P.S., MacKenzie, P.D., Yung, M.: Proactive RSA. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 440–454. Springer, Heidelberg (1997)
20. Frankel, Y., MacKenzie, P.D., Yung, M.: Adaptive security for the additive-sharing based proactive RSA. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 240–263. Springer, Heidelberg (2001)
21. Franklin, M., Yung, M.: Communication complexity of secure computation (extended abstract). In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, STOC 1992, pp. 699–710, New York, NY, USA. ACM (1992)
22. Franklin, M.K., Galil, Z., Yung, M.: Eavesdropping games: a graph-theoretic approach to privacy in distributed systems. In: FOCS, pp. 670–679 (1993)
23. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: how to cope with perpetual leakage. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 339–352. Springer, Heidelberg (1995)
24. Itkis, G., Reyzin, L.: SiBIR: signer-base intrusion-resilient signatures. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, p. 499. Springer, Heidelberg (2002)
25. Jarecki, S., Olsen, J.: Proactive RSA with non-interactive signing. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 215–230. Springer, Heidelberg (2008)
26. Jarecki, S., Saxena, N.: Further simplifications in proactive RSA signatures. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 510–528. Springer, Heidelberg (2005)
27. McMillan, R.: $1.2m hack shows why you should never store bitcoins on the internet (2013)
28. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks. In: Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, pp. 51–59. ACM Press (1991)
29. Rabin, T.: A simplified approach to threshold and proactive RSA. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 89–104. Springer, Heidelberg (1998)
30. Schultz, D.: Mobile proactive secret sharing. Ph.D. thesis, Massachusetts Institute of Technology (2007)
31. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
32. Silver-Greenberg, J., Goldstein, M., Perlroth, N.: JPMorgan Chase hacking affects 76 million households (2014). http://dealbook.nytimes.com/2014/10/02/jpmorgan-discovers-further-cyber-security-issues/
33. Wong, T.M., Wang, C., Wing, J.M.: Verifiable secret redistribution for archive system. In: IEEE Security in Storage Workshop, pp. 94–106 (2002)
34. Zhou, L., Schneider, F.B., van Renesse, R.: Apss: proactive secret sharing in asynchronous systems. ACM Trans. Inf. Syst. Secur. **8**(3), 259–286 (2005)