Communication-Optimal Proactive Secret Sharing for Dynamic Groups

Joshua Baron*
RAND Corporation
jbaron@rand.org

Karim El Defrawy
HRL Laboratories
kmeldefrawy@hrl.com

Joshua Lampkins*
Department of Mathematics, UCLA
jlampkins@math.ucla.edu

Rafail Ostrovsky
Department of Computer Science and
Department of Mathematics, UCLA
rafail@cs.ucla.edu

Abstract

Proactive secret sharing (PSS) schemes are designed for settings where long-term confidentiality of secrets has to be guaranteed, 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 t is replaced with a corruption *rate* which cannot be violated. In *dynamic proactive secret sharing* (DPSS) the number of parties can vary during the course of execution. 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 schemes with optimal amortized, $O(1)$, per-secret communication compared to $O(n^4)$ or $\exp(n)$ in number of parties, n, required by existing schemes. 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 [30, 4] 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 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

*The work of this author was performed while at HRL Laboratories.

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 [27] 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. 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, and such schemes have also been the subject of numerous papers [16, 32, 33, 29] 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 [26, 31]. Such breaches show an increasing need for secure long-term storage solutions. Although several other layers of security must be developed for a complete data storage system to be secure against a mobile adversary, DPSS is an important step toward this goal. Standard secret sharing can address this issue by distributing data to avoid single points of compromise or failure, but 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.

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. The idea, introduced in [20], is 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 [30]. 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 $O(n)$ communication complexity so that the amortized complexity is $O(1)$ per secret.

Hyper-Invertible Matrices. A hyper-invertible matrix [3] 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 $n-3t = O(n)$ sharings are verified by sending $n = O(n)$ sharings to parties, this only requires constant amortized communication bandwidth.

Party Virtualization. Party virtualization [6] is a method for transforming a multiparty protocol by replacing each player 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 [14], 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. 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 Section 4.1 (Definition 4), a DPSS scheme consists of three protocols: Share, Redistribute, and Open that distribute, redistribute, and reconstruct shares to a secret, respectively. For the protocols Share and Open, we use the protocols RobustShare and Reco (respectively) from [14].

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).[1] (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 [29] where "party virtualization" is utilized when the set of parties is decreased. "Party virtualization" occurs when each real party holds internal data (i.e., shares) corresponding to some virtual party. That is, there are n 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.[2] As stated in [29], 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 [29] 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 proactive secure multiparty computation (DPMPC) protocol.

1.3 Outline

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

2 Related Work

The same work [27] 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 [27], and there has been significant follow

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

[2]Note that the term "party virtualization" has a different meaning in [29], which is how it is used here, than it has in other secure computation literature such as [14].

Paper	**Dynamic**	**Network**	**Security**	**Threshold**	**Communication Complexity**
[32]	Yes	synch.	cryptographic	$t/n < 1/2$	$\exp(n)$
[33]	Yes	asynch.	cryptographic	$t/n < 1/3$	$\exp(n)$
[7]	No	asynch.	cryptographic	$t/n < 1/3$	$O(n^4)$
[29]	Yes	asynch.	cryptographic	$t/n < 1/3$	$O(n^4)$
[22]	No	synch.	cryptographic	$t/n < 1/2$	$O(n^2)$
[2]	No	synch.	perfect	$t/n < 1/3-\epsilon$	$O(1)$
[2]	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	$t/n < 1/2-\epsilon$	$O(1)$

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.

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 [2], 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 [29], 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 [29], 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 [29] 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 [17, 18, 28, 9, 19, 5, 25, 24] (and in particular [1], which also sketches a definition of UC security in the proactive framework), intrusion-resilient signatures [23], eavesdropping games [21], pseudorandomness [10], and state-machine replication [11, 12].

The only two known general PMPC protocols are [27] and [2]. 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 [29], and the most efficient PSS scheme to date is that of [2]. In this section, we explain why straightforward modifications of either of these would not produce a DPSS scheme with optimal communication requirements.

In [2], 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 [29], 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 Section 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 de-corrupted 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* 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 [14] 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 [14] to perform secret opening.

Our new redistribution protocol given in Section 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 [14], 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[3] technique of [14]. The protocol in Section 5 is written as a perfectly secure protocol with a lower threshold, and then this is raised using statistically secure virtualization (see Section 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 [3] 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 [15], 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 greater than $\lfloor \eta n \rfloor$ (resp. $n/4$) in the perfectly

[3]The term "party virtualization" has a different meaning in [29] than it has in [14].

(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 $\mathsf{Refresh\_Recovery}$ 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 [2] that is still secure in the dynamic setting (recall that a straightforward adoption is insecure as discussed in Section 3). The recovery is performed in essentially the same way as in [2], 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 [2] is for PSS, not DPSS.)

In the dynamic setting, refresh cannot be performed as in [2]. As mentioned in Section 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.[4] In [2], 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 [2]), 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 $H + Q$).

5.3 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 [14]. Our contribution is the construction of the redistribution protocol (Figure 1).

The protocol RobustShare allows the parties to share $O(n)$ secrets with $O(n)$ communication complexity using batch sharing. This is done 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 Ran-DouSha from [14] is also used as a subprotocol in our redistribution protocol. The protocol

[4]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 $H + Q$.

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.

Figure 1: Redistribute.

As seen in Figure 1, there are four cases for threshold change. To simplify the treatment we only focus on case 2 (which is when the threshold is decreasing and the batch size is decreasing) here in Figure 2 and defer the other three cases to Appendix A (Figures 4, 5, and 6).

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 $B \leftarrow 4B$.

Figure 2: Threshold_Change$_2$.

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

1. *Double Sharing Batched Secrets*

1.1 The parties generate sharings of $\ell t B$ 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*

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 $a \in [\ell']$,

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

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

Figure 3: Refresh_Recovery.

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$_4$.

6 Party Virtualization

As stated in Section 1.2, we do not require party virtualization as defined in [29]. However for the statistical version of our protocol, we require the use of a party virtualization technique similar to that in [13] (note that these are different techniques as noted before in Section 1.2). The technique, initially introduced in [6], 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 [13] and [2] for details.

Changing the threshold when player 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.[5]

In [2], the authors show how to proactivize the MPC scheme of [13] 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 [2].

References

[1] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold rsa with adaptive and proactive security. In *Proceedings of the 24th annual international conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, pages 593–611, Berlin, Heidelberg, 2006. Springer-Verlag.

[2] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. How to withstand mobile virus attacks, revisited. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 293–302, New York, NY, USA, 2014. ACM.

[3] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In *TCC*, pages 213–230, 2008.

[4] G. R. Blakley. Safeguarding cryptographic keys. *Proc. of AFIPS National Computer Conference*, 48:313–317, 1979.

[5] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography: Public Key Cryptography*, PKC '03, pages 31–46, London, UK, UK, 2003. Springer-Verlag.

[6] Gabriel Bracha. An O(log n) expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.

[7] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

[8] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. *IACR Cryptology ePrint Archive*, 2000:67, 2000.

[5] 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.

[9] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 98–115, London, UK, UK, 1999. Springer-Verlag.

[10] Ran Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In *CRYPTO*, pages 425–438, 1994.

[11] Miguel Castro and Barbara Liskov. Proactive recovery in a byzantine-fault-tolerant system. In *OSDI*, pages 273–288, 2000.

[12] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[13] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, pages 445–465, 2010.

[14] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, pages 241–261, 2008.

[15] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

[16] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications. *Technical Report ISSE TR-97-01, George Mason University*, July 1997.

[17] Y. Frankel, P. Gemmell, P. D. MacKenzie, and Moti Yung. Optimal-resilience proactive public-key cryptosystems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, pages 384–, Washington, DC, USA, 1997. IEEE Computer Society.

[18] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Proactive rsa. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '97, pages 440–454, London, UK, UK, 1997. Springer-Verlag.

[19] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Adaptive security for the additive-sharing based proactive rsa. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, PKC '01, pages 240–263, London, UK, UK, 2001. Springer-Verlag.

[20] Matthew Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 699–710, New York, NY, USA, 1992. ACM.

[21] Matthew K. Franklin, Zvi Galil, and Moti Yung. Eavesdropping games: A graph-theoretic approach to privacy in distributed systems. In *FOCS*, pages 670–679, 1993.

[22] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO*, pages 339–352, 1995.

[23] Gene Itkis and Leonid Reyzin. Sibir: Signer-base intrusion-resilient signatures. In *CRYPTO*, pages 499–514, 2002.

[24] Stanisaw Jarecki and Josh Olsen. Proactive rsa with non-interactive signing. In Gene Tsudik, editor, *Financial Cryptography and Data Security*, volume 5143 of *Lecture Notes in Computer Science*, pages 215–230. Springer Berlin Heidelberg, 2008.

[25] Stanislaw Jarecki and Nitesh Saxena. Further simplifications in proactive rsa signatures. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 510–528, Berlin, Heidelberg, 2005. Springer-Verlag.

[26] Robert McMillan. $1.2m hack shows why you should never store bitcoins on the internet, 2013. `http://www.wired.com/wiredenterprise/2013/11/inputs/`.

[27] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *In Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59. ACM Press, 1991.

[28] Tal Rabin. A simplified approach to threshold and proactive rsa. In *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '98, pages 89–104, London, UK, UK, 1998. Springer-Verlag.

[29] David Schultz. *Mobile Proactive Secret Sharing.* PhD thesis, Massachusetts Institute of Technology, 2007.

[30] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[31] Jessica Silver-Greenberg, Matthew Goldstein, and Nicole Perlroth. JPMorgan Chase hacking affects 76 million households, 2014. `http://dealbook.nytimes.com/2014/10/02/jpmorgan-discovers-further-cyber-security-issues/`.

[32] Theodore M. Wong, Chenxi Wang, and Jeannette M. Wing. Verifiable secret redistribution for archive system. In *IEEE Security in Storage Workshop*, pages 94–106, 2002.

[33] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. Apss: proactive secret sharing in asynchronous systems. *ACM Trans. Inf. Syst. Secur.*, 8(3):259–286, 2005.

A Threshold Changing Subprotocols

This Appendix section contains the details of the rest of the cases for $\mathsf{Threshold\_Change}_i$ for $i = 1, 3, 4$.

Lowering the Threshold, Batch Size Stays the Same

1. The parties invoke $\mathsf{RanDouSha}$ to generate masking polynomials $H_a^{(k,m)}$ of degree $\leq d$ for $k = [n-3t+1, n-2t]$ (where a and m range over the same values as before).
2. Define $\widetilde{H}_a^{(k,m)}$ for $k \in [n]$ by

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

   Each party locally computes their shares of these polynomials and sends his share of each $\widetilde{H}_a^{(j,m)}$ to party P_j.
3. Each P_i uses Berlekamp-Welch to interpolate the shares of $\widetilde{H}_a^{(i,m)}$ received in the previous step.
4. Each P_i computes (shares of) the unique polynomial $\widetilde{h}_a^{(i,m)}$ of degree $\leq d'$ that agrees with $\widetilde{H}_a^{(i,m)}$ on the evaluation points ζ^1 through $\zeta^{d'+1}$.
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 polynomial of degree $\leq d'$ that agrees with $H_a^{(i,m)}$ on the evaluation points ζ^1 through $\zeta^{\ell+t'}$, 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. To simplify notation in the rest of the protocol, we now set $H_a^{(k,m)} \leftarrow h_a^{(k,m)}$ for $(a, k, m) \in [\ell] \times [n-3t] \times [B]$.

Figure 4: $\mathsf{Threshold\_Change}_1$.

Raising the Threshold, Batch Size Stays the Same

1. The parties invoke $\mathsf{RanDouSha}$ to generate masking polynomials $H_a^{(k,m)}$ of degree $\leq d$ for $k \in [n-3t+1, n-2t]$ (where a and m range over the same values as before).
2. The parties invoke $\mathsf{RanDouSha}$ to generate random polynomials $R_a^{(k,m)}$ of degree $\leq d'$ for $k \in [n-2t]$ (where a and m range over the same values as before).
3. 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.

4. 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.

5. Each P_i computes (shares of) the unique polynomials $\widetilde{h}_a^{(i,m)}$ of degree $\leq d'$ that agrees with $\widetilde{H}_a^{(i,m)}$ on the points ζ^1 through ζ^ℓ and agrees with $\widetilde{R}_a^{(i,m)}$ on the points $\zeta^{\ell+1}$ through $\zeta^{d'+1}$.

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

7. If we define $h_a^{(i,m)}$ to be the unique polynomials of degree $\leq d'$ that agrees with $H_a^{(i,m)}$ on the points ζ^1 through ζ^ℓ and agrees with $R_a^{(i,m)}$ on the points $\zeta^{\ell+1}$ through $\zeta^{d'+1}$, 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.

8. To simplify notation in the rest of the protocol, we now set $H_a^{(k,m)} \leftarrow h_a^{(k,m)}$ for $a \in [\ell]$, $k \in [n-3t]$, and $m \in [B]$.

Figure 5: Threshold_Change$_3$.

Raising the Threshold, Batch Size Increases

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

1. The parties invoke RanDouSha to generate masking polynomials $H_a^{(k,m)}$ of degree $\leq d$ for $k \in [n-3t+1, n-2t]$ (where a and m range over the same values as before).

2. The parties invoke RanDouSha to generate random polynomials $R_a^{(k,m)}$ of degree $\leq d'$ for $k \in [n-2t]$, $a \in [\ell/2]$, and $m \in [B]$.

3. 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.

4. 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.

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

5.1 $\widetilde{h}_a^{(i,m)}(\zeta^j) = \widetilde{H}_{2a-1}^{(i,m)}(\zeta^j)$ for $j \in [\ell]$.

5.2 $\widetilde{h}_a^{(i,m)}(\zeta^{\ell+j}) = \widetilde{H}_{2a}^{(i,m)}(\zeta^j)$ for $j \in [\ell]$.

5.3 $\widetilde{h}_a^{(i,m)}(\zeta^{\ell'+j}) = \widetilde{R}_a^{(i,m)}(\zeta^{\ell'+j})$ for $j \in [d' - \ell' + 1]$.

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

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

7.1 $h_a^{(k,m)}(\zeta^j) = H_{2a-1}^{(k,m)}(\zeta^j)$ for $j \in [\ell]$,

7.2 $h_a^{(k,m)}(\zeta^{\ell+j}) = H_{2a}^{(k,m)}(\zeta^j)$ for $j \in [\ell]$,

7.3 $h_a^{(k,m)}(\zeta^{\ell'+j}) = R_a^{(k,m)}(\zeta^{\ell'+j})$ 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.

8. 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 $\left\{H_a^{(k,m)}\right\}_{\substack{m=1,\ldots,B/4 \\ k=1,\ldots,n-3t \\ a=1,\ldots,\ell'}} \leftarrow \left\{h_a^{(k,m)}\right\}_{\substack{m=1,\ldots,B \\ k=1,\ldots,n-3t \\ a=1,\ldots,\ell/2}}$ in such a way that this map preserves lexicographical order. We then relabel $B \leftarrow B/4$.

Figure 6: Threshold_Change$_4$.

B Security Definition and Proof

We do not provide an ideal functionality for each of the sub-protocols that comprise our redistribution protocol. We note that the functionality for Refresh_Recovery is similar to the functionality for Block-Redistribute from [2]. The functionality for each of the threshold-changing subprotocols is similar. We provide the functionality (Figure 7) and simulator (Figure 8) for Threshold_Change$_2$ below and provide its security proof. Security is proved in the universally composable framework [8].

1. *Input Phase*

1.1 $\mathcal{Z}$ provides each $P_i \in \mathcal{P}$ and each $P'_j \in \mathcal{P}'$ with input t, $\mathcal{P}$, $\mathcal{C}orr$, t', $\mathcal{P}'$, ℓ, and B. Each P_i additionally receives its share of each $H_a^{(k,m)}$.

1.2 If the inputs forwarded by the (dummy) parties to $\mathcal{F}_2$ are inconsistent, then $\mathcal{F}_2$ outputs (`abort`) and aborts.

1.3 $\mathcal{Z}$ initializes the adversary $\mathcal{A}$ with auxiliary input z.

2. *Corruption Phase*
$\mathcal{A}$ may request to corrupt parties in $\mathcal{P}$ or $\mathcal{P}'$ by sending messages (`corrupt`, P_i) or (`corrupt`, P'_i) to $\mathcal{F}_2$. For each party in $\mathcal{P}$ that the adversary corrupts, $\mathcal{F}_2$ sends that party's share of each $H_a^{(k,m)}$ to $\mathcal{A}$, and $\mathcal{A}$ may provide new input shares for that party to $\mathcal{F}_2$. After each corruption, $\mathcal{F}_2$ sends (`corrupt`) to the corrupted (dummy) party, which then forwards this message to $\mathcal{Z}$.

3. *Output Phase*

3.1 $\mathcal{F}_2$ interpolates $H_a^{(k,m)}(\zeta^j)$ for $(a,j,k,m) \in [\ell] \times [\ell] \times [n-3t] \times [B]$ from the shares provided by the honest parties.

3.2 $\mathcal{F}_2$ sends $(\texttt{Shares?}, \ell', B)$ to $\mathcal{A}$.

3.3 $\mathcal{A}$ sends shares $h_a^{(k,m)}(\beta_j)$ to $\mathcal{F}_2$ for each $(a,k,m) \in [2\ell] \times [n-3t] \times [B]$ for each corrupt $P'_j \in \mathcal{P}'$.

3.4 $\mathcal{F}_2$ constructs degree d' polynomials $h_a^{(k,m)}$ for $(a,k,m) \in [2\ell] \times [n-3t] \times [B]$ that are random subject to the constraint that they agree with the shares provided by $\mathcal{A}$ and that $h_{2a-1}^{(k,m)}(\zeta^j) = H_a^{(k,m)}(\zeta^j)$ and $h_{2a}^{(k,m)}(\zeta^j) = H_a^{(k,m)}(\zeta^{\ell'+j})$ for $j \in [\ell']$.

3.5 $\mathcal{F}_2$ relabels $\{H_a^{(k,m)}\} \leftarrow \{h_a^{(k,m)}\}$ as specified in the last step of Threshold_Change$_2$ and relabels $B \leftarrow 4B$.

3.6 $\mathcal{F}_2$ outputs to each honest party her share of each $H_a^{(k,m)}$. $\mathcal{F}_2$ provides outputs for the dishonest parties as specified by $\mathcal{A}$.

Figure 7: Description of $\mathcal{F}_2$.

Note that security here is proved by comparing the ideal execution of Threshold_Change$_2$ with the execution in the hybrid model that uses the ideal functionality for RanDouSha. The ideal functionality for RanDouSha is given in [13].

1. $\mathcal{S}$, emulating the functionality for RanDouSha, sends (`Shares?`) to $\mathcal{A}$.

2. $\mathcal{A}$ sends $\mathcal{S}$ shares for corrupt parties of $H_a^{(k,m)}$ for $(a,k,m) \in [\ell] \times [n-3t+1, n-2t] \times [B]$ and $R_a^{(k,m)}$ for $(a,k,m) \in [2\ell] \times [n-2t] \times [B]$.

3. $\mathcal{S}$ sends messages to $\mathcal{A}$ emulating the honest parties from $\mathcal{P}$ in step 2 of Threshold_Change$_2$. More specifically, $\mathcal{S}$ selects polynomials $\widetilde{H}_a^{(k,m)}$ and $\widetilde{R}_a^{(k,m)}$ for each corrupt $P_k \in \mathcal{P}$ that are random subject to the constraint that $\widetilde{H}_a^{(k,m)}(\alpha_b)$ equals the kth coordinate of $M(H_a^{(1,m)}(\alpha_b), \ldots, H_a^{(n-2t,m)}(\alpha_b))$ for each corrupt P_b (and similarly for $\widetilde{R}_a^{(k,m)}(\alpha_b)$). Then $\mathcal{A}$ iteratively requests $\mathcal{S}$ to send messages for one honest party at a time to all the dishonest parties. At each request, $\mathcal{S}$ sends the shares $\widetilde{H}_a^{(k,m)}(\alpha_i)$ and $\widetilde{R}_a^{(k,m)}(\alpha_i)$ to $\mathcal{A}$ on behalf an honest P_i. After each honest party sends its messages, $\mathcal{A}$ may decide to iteratively corrupt honest parties. If $\mathcal{A}$ wants to corrupt a party from $\mathcal{P}'$, then $\mathcal{A}$ simply sends $(\texttt{Corrupt}, P'_j)$ to $\mathcal{S}$. For each corruption of a party in $\mathcal{P}$, the following steps are executed:

3.1 $\mathcal{A}$ sends $(\texttt{Corrupt}, P_i)$ to $\mathcal{S}$.

3.2 $\mathcal{S}$ sends $(\texttt{Corrupt}, P_i)$ to $\mathcal{F}_2$.

3.3 $\mathcal{F}_2$ sends shares $H_a^{(k,m)}(\alpha_i)$ and $R_a^{(k,m)}(\alpha_i)$ for $(a,k,m) \in [\ell] \times [n-3t] \times [B]$ to $\mathcal{S}$.

3.4 $\mathcal{S}$ chooses shares $H_a^{(k,m)}(\alpha_i)$ for $(a,k,m) \in [\ell] \times [n-3t+1, n-2t] \times [B]$ uniformly at random subject to the constraint that for each (a,m), the w^{th} coordinate of $M(H_a^{(1,m)}(\alpha_i), \ldots, H_a^{(n-2t,m)}(\alpha_i))^T$ equals the value of $\widetilde{H}_a^{(w,m)}(\alpha_i)$ already sent to the adversary for each corrupt $P_w \in \mathcal{P}$. (This can be done because the adversary has seen $\widetilde{H}_a^{(w,m)}(\alpha_i)$ for no more than t different values of w, and because M is a hyper-invertible matrix.) $\mathcal{S}$ similarly chooses shares $R_a^{(k,m)}(\alpha_i)$. These shares are then sent to $\mathcal{A}$.

4. $\mathcal{A}$ sends messages to $\mathcal{S}$ emulating the corrupt parties from $\mathcal{P}$ in step 2 of $\mathsf{Threshold\_Change}_2$.

5. $\mathcal{S}$ chooses shares $h_{2a-1}^{(i,m)}(\alpha_j)$ and $h_{2a}^{(i,m)}(\alpha_j)$ for each corrupt $P_j \in \mathcal{P}$ randomly subject to the constraint that they agree with the shares of $\widetilde{h}_{2a-1}^{(i,m)}$ and $\widetilde{h}_{2a}^{(i,m)}$ already known to $\mathcal{A}$. $\mathcal{S}$ internally computes $\widetilde{h}_{2a-1}^{(i,m)}(\alpha_j)$ and $\widetilde{h}_{2a}^{(i,m)}(\alpha_j)$ for each corrupt $P_j \in \mathcal{P}$ and each uncorrupted $P_i \in \mathcal{P}$ for $(a,m) \in [\ell] \times [B]$ such that the shares are consistent with the chosen shares of $h_{2a-1}^{(i,m)}$ and $h_{2a}^{(i,m)}$.

6. $\mathcal{S}$ sends messages to $\mathcal{A}$ emulating the honest parties from $\mathcal{P}$ in step 5 of $\mathsf{Threshold\_Change}_2$. This is done as in step 3 above, with the adversary adaptively corrupting parties. In addition to what is done in step 3, for each newly corrupted party $\mathcal{S}$ chooses polynomials $\widetilde{H}_a^{(k,m)}$ and $\widetilde{R}_a^{(k,m)}$ that are random subject to the constraint that they are consistent with the shares of $\widetilde{h}_{2a-1}^{(i,m)}$ and $\widetilde{h}_{2a}^{(i,m)}$ already known to $\mathcal{A}$ and sends these shares to $\mathcal{A}$.

7. $\mathcal{S}$ relabels $\{H_a^{(k,m)}\} \leftarrow \{h_a^{(k,m)}\}$ as specified in the last step of $\mathsf{Threshold\_Change}_2$ and relabels $B \leftarrow 4B$.

8. When $\mathcal{S}$ receives $(\texttt{Shares?}, \ell', B)$ from $\mathcal{F}_2$, $\mathcal{S}$ sends shares $H_a^{(k,m)}(\alpha_j)$ for each corrupt $P_j \in \mathcal{P}$ to $\mathcal{F}_2$.

9. $\mathcal{A}$ sends an output for the environment to $\mathcal{S}$, and $\mathcal{S}$ forwards this to $\mathcal{Z}$.

Figure 8: Description of Simulator for $\mathcal{F}_2$.

Proof: In both the ideal and the hybrid execution, $\mathcal{A}$'s view of the polynomials $\widetilde{H}_a^{(k,m)}$ are random given the adversary's view of the shares of $H_a^{(k,m)}$. In the ideal execution, this is because $\mathcal{S}$ chose them randomly (subject to the constraint given in step 3 of the simulator). To see why this holds in the hybrid execution, let T denote some set of t players that contains the corrupt players, and let $k_1 < \cdots < k_t$ denote the indices of players in T. Let

M_C denote the $t \times n-2t$ matrix obtained from selecting the rows of M that correspond to the players in T. Let M_C^L denote the $t \times t$ matrix obtained from selecting the last t columns of M_C, and let M_C^F denote the $t \times n-3t$ matrix obtained from selecting the first $n-3t$ columns of M_C (so that M_C is the concatenation of M_C^F and M_C^L). Then

$$\begin{aligned}&(\widetilde{H}_a^{(k_1,m)}, \ldots, \widetilde{H}_a^{(k_t,m)})^T \\ &\quad = \; M_C^F(H_a^{(1,m)}, \ldots, H_a^{(n-3t,m)})^T + M_C^L(H_a^{(n-3t+1,m)}, \ldots, H_a^{(n-2t,m)})^T.\end{aligned}$$

Since M is hyper-invertible, M_C^L is invertible, so the above equation implies

$$\begin{aligned}&(H_a^{(n-3t+1,m)}, \ldots, H_a^{(n-2t,m)})^T \\ &\quad = \; (M_C^L)^{-1}\left[M_C^F(H_a^{(1,m)}, \ldots, H_a^{(n-3t,m)})^T - (\widetilde{H}_a^{(k_1,m)}, \ldots, \widetilde{H}_a^{(k_t,m)})^T\right].\end{aligned}$$

Thus for each possible set of polynomials $H_a^{(1,m)}, \ldots, H_a^{(n-3t,m)}$ and $\widetilde{H}_a^{(k_1,m)}, \ldots, \widetilde{H}_a^{(k_t,m)}$, there is exactly one choice of polynomials $H_a^{(n-3t+1,m)}, \ldots, H_a^{(n-2t,m)}$ that satisfies the equation. Since the ideal functionality for RanDouSha chooses the polynomials $H_a^{(n-3t+1,m)}, \ldots, H_a^{(n-2t,m)}$ randomly in the hybrid execution, $\mathcal{A}$'s view of the $\widetilde{H}_a^{(k,m)}$ is random given $\mathcal{A}$'s shares of the $H_a^{(k,m)}$.

Since the polynomials $\widetilde{h}_a^{(i,m)}$ can be determined from the polynomials $h_a^{(i,m)}$ (and vice versa), it suffices to show that $\mathcal{A}$'s view of $h_a^{(i,m)}$ is random in both the ideal and hybrid execution. In the ideal execution, this holds because the $h_a^{(i,m)}$ were chosen randomly given the shares of $\widetilde{h}_a^{(i,m)}$ already known to the adversary, and those shares were uniformly random (because the $\widetilde{R}_a^{(i,m)}$ used to generate the randomness for the $\widetilde{h}_a^{(i,m)}$ were chosen randomly). In the hybrid execution, this is because the $R_a^{(i,m)}$ were chosen randomly, and the secrets in the $R_a^{(i,m)}$ are used to determine the randomness in the $h_a^{(i,m)}$.

In both the ideal and the hybrid execution, $\mathcal{A}$'s view of the polynomials $\widetilde{R}_a^{(k,m)}$ are random given the adversary's view of the shares of and $R_a^{(k,m)}$ and the chosen shares of $h_a^{(k,m)}$. This follows from the same matrix argument applied to the $H_a^{(k,m)}$ and by construction of the $h_a^{(k,m)}$.

In the ideal execution, $\mathcal{F}_2$ chooses the honest parties' output shares to be random (given the shares of the corrupt parties provided by the adversary and the secrets being stored). In the hybrid execution, the output shares of the honest parties are determined by the secrets and by the shares they received from the invocation of the functionality for RanDouSha, so again, they are random (given the shares of the corrupt parties provided by the adversary and the secrets being stored). Thus $\mathcal{Z}$ will be unable to distinguish between the ideal and hybrid executions. $\square$