Proactive Secret Sharing with Constant Communication

Brett Hemenway Falk, Daniel Noble, and Tal Rabin

{fbrett, dgnoble, talr}@seas.upenn.edu
University of Pennsylvania

Abstract. This paper presents the first protocols for Proactive Secret Sharing (PSS) that only require constant (in the number of parties, n) communication per party per epoch. By harnessing the power of expander graphs, we are able to obtain strong guarantees about the security of the system. We present the following PSS protocols:

- A PSS protocol that provides *privacy* (but no robustness) against an adversary controlling $\mathcal{O}(n)$ parties per epoch.
- A PSS protocol that provides *robustness* (but no privacy) against an adversary controlling $\mathcal{O}(n)$ parties per epoch.
- A PSS protocol that provides privacy against an adversary controlling $\mathcal{O}(n^a)$ parties per epoch and provides robustness against an adversary controlling $\mathcal{O}(n^{1-a})$ parties per epoch, for any constant $0 \leq a \leq 1$. Instantiating this with $a = \frac{1}{2}$ gives a PSS protocol that is proactively secure (private and robust) against an adversary controlling $\mathcal{O}(\sqrt{n})$ parties per epoch.

Additionally, we discuss how secure channels, whose existence is usually assumed by PSS protocols, are challenging to create in the mobile adversary setting, and we present a method to instantiate them from a weaker assumption.

1 Introduction

Most multiparty protocols provide security as long as no more than a certain threshold of the parties are corrupted, e.g. the Shamir secret-sharing provides security as long as no more than t-out-of-n of the parties are corrupted. These protocols implicitly assume that adversarial corruptions are *static*, i.e., the subset of corrupted parties does not change over time.

The notion of *proactive* security [OY91], considers a *mobile* adversary that can adaptively corrupt different parties, subject to a maximum corruption threshold at a given time. More formally, the model considers a multiparty protocol with n parties, where time is divided into epochs. In each epoch the adversary can corrupt up to t of the n parties, learning their state (and in the malicious model completely controlling their behavior). In the next epoch, the adversary adaptively chooses a new subset of t parties to corrupt, and this continues indefinitely. A protocol that can achieve privacy (or robustness) in the face of this type of mobile adversary is said to be *proactively secure*.

When considering proactive security, it is sufficient to consider Proactive Secret Sharing (PSS), i.e., secret-sharing schemes that can achieve privacy (and/or robustness) in the face of a mobile adversary. This is because any MPC protocol that computes on secret shares can be made proactively secure by simply assuming that each round of the MPC protocol happens *within* a single epoch. With this assumption, the adversary is essentially static with respect to the MPC protocol, and security follows immediately from the proactive security of the underlying secret sharing scheme together with the (static) security of the MPC protocol. Thus, previous works focused on building proactive secret sharing protocols, with the understanding that PSS protocols can be used as the substrate for general secure multiparty computation secure against mobile adversaries.

In addition to the design of the secret sharing protocol, i.e., the refreshing of shares, there is an orthogonal issue which needs to be addressed: the creation and re-establishing of the secure communication channels between the parties after (potential) adversarial corruptions. Previous works either simply assume that an infrastructure for secure channels exists, or have solutions to create secure channels that require (at least) $\Theta(n)$ communication per party per epoch, where n is the number of parties. We detail the prior art in Appendix A with an abridged version in Section 2.

Our Results

Given the communication complexity of prior constructions, the natural question to ask is whether this $O(n)$ communication for PSS is inherent or whether there exist protocols with sublinear communication. In this work, assuming a synchronous network, we present the first PSS protocol for single (unbatched) secrets that achieves sublinear communication. Surprisingly, we show that PSS is possible against passive mobile adversaries corrupting $\Theta(n)$ parties per epoch with only constant (in n) communication per party! Furthermore, we present a PSS protocol that is secure against *active* mobile adversaries corrupting $\Theta(\sqrt{n})$ parties per epoch that also has constant (in n) communication.

Assuming the existence of secure communication channels we show three PSS protocols with constant communication per party. Our first protocol provides secrecy for the shared value, but offers no robustness, i.e. it works only against a passive adversary (Section 5). The second provides robustness but no privacy, that is a malicious adversary cannot corrupt the secret (Section 6). Finally, we combine the first two protocols to provide both secrecy and robustness (Section 7). Our first two protocols are secure against an adversary corrupting $\Theta(n)$ parties per epoch while our third is only secure against an adversary corrupting $\Theta(\sqrt{n})$ parties per epoch. We note, however, that because our per-epoch communication cost is so low, we can set our epoch times to be *much* shorter than existing PSS protocols, which would reduce the number of parties that an adversary can corrupt during an epoch (see Appendix D).

Note that while the number of messages sent per party per epoch is constant, and the size of each message is independent of n, the message sizes do depend on two other parameters. Like any other PSS protocol, our message sizes depend

on the size of the secret, $|\mathbb{F}|$. For notational simplicity we assume $|\mathbb{F}| = \mathcal{O}(1)$. Messages sizes may also depend on the computational security parameter, κ. Assuming secure channels, our first and second protocols do not depend on κ, while our third protocol has messages of size $\mathcal{O}(\kappa)$. Note that some works use batching to combine many secrets to obtain low communication cost per secret. We *do not use batching*; our results hold even if there is only a single secret.

Secure communication channels are required for PSS, so we also develop a method for establishing secure channels between parties that requires only $\mathcal{O}(\kappa)$ communication per party per epoch (Section 8). Using this protocol to instantiate secure channels (instead of simply assuming secure channels exist) increases the communication complexity of our first and second PSS protocols to $\mathcal{O}(\kappa)$ while our third PSS protocol remains $\mathcal{O}(\kappa)$. Our method requires a minimal trusted hardware assumption: that each party has access to a secure signing oracle. The adversary may make the oracle sign arbitrary messages when the party is corrupted, but cannot learn the secret key. This is a *much* weaker assumption than that of secure hardware channels, and is implemented by many common devices such as Yubikeys or iOS Secure Enclaves.

Our third PSS protocol can be easily modified to achieve a different cryptographic primitive called Proactive Pseudorandomness (PP), that is a protocol which enables a set of parties to preserve the ability to generate pseudorandomness in the face of a mobile adversary, despite no access to true randomness. Our protocol requires only $\mathcal{O}(\kappa)$ communication per-party per epoch and maintains (global) pseudorandomness against a mobile adversary controlling $\Theta(n)$ parties per epoch. While this is an interesting development (previous protocols required $\mathcal{O}(n)$ communication), it is not a core contribution of this work so is presented in Appendix F.

Our PSS protocols rely on expander graphs and in Section 4 we provide the properties and theorems for these graphs that we need in our design. Instead of requiring that each party communicate with every other party, each party communicates with only a constant number of neighbors, where the assignment of neighbors is chosen according to an expander graph.

Because each party only communicates with a constant number of other parties, it is possible that an honest party be entirely surrounded by corrupt parties. As such, the adversary may learn the honest party's state (by knowing all messages sent to it) or may cause an honest party to behave incorrectly (by sending it incorrect messages). Our security guarantees therefore will not be local: they will not necessarily apply to every honest party. Instead we prove *global* security properties that hold over the entire system, e.g. that the adversary is not able to learn a secret that has been shared between all parties, or that the adversary cannot cause most parties to behave incorrectly. Intuitively, these global security properties will hold because the expansion property of the communication network ensures that the set of honest parties at different times remain generally well-connected to each other.

2 Related Work

A full related work section appears in Appendix A. Table 1 shows the communication complexity of the works discussed there, as well as our own results. Here we only provide details of a few of the works.

Work	**Communication**	**Threshold**	**Synchrony**
[HJKY95]	$O(n)$	$n = 2t + 1$	sync
[CKLS02]	$O\left(n^3\right)$	$n = 3t + 1$	async
[SLL10]	$O(n)$	$n = 3t + 1$	async
[ZSVR05]	$\exp(n)$	$n = 3t + 1$	async
[BEDLO14]	$O(n/\ell)$	$n = 2(t + \ell) + 1$	sync
[ELL20]	$O(n^2/\ell)$	$n = t + \sqrt{\ell} + 1$	sync
[MZW+19]	$O(n^2)$	$n = 2t + 1$	sync
[YXD22]	$O(n^2 \log n)$	$n = 4t + 1$	async
Protocol 1 (passive adversary)	$O(1)$	$n = (1 + \epsilon)t$	sync
Protocol 3	$O(\kappa)$	$n = (2 + \epsilon)t^2$	sync

Table 1: PSS schemes. ℓ is the size of a batch, $\epsilon > 0$ is a constant.

Proactive secret sharing considers the problem of maintaining the *privacy* and *robustness* of a shared secret in the presence of a *mobile* adversary [OY91]. In the mobile-adversary model, time is divided into "epochs," and the adversary is allowed to corrupt a new subset of parties in every epoch.

In order for PSS to be feasible, we must assume that parties can be securely "rebooted," an operation which leaves them in a fresh (uncorrupted) state. We must also assume that parties can securely delete information, otherwise an adversary corrupting a party in one epoch could learn their shares from previous epochs, which would make it impossible to maintain privacy.

Essentially all PSS protocols are built around the idea of "refreshing" the parties' shares at every epoch. One method of refreshing shares is to simply have all parties generate a random sharing of zero, and then add these shares to the shares of the original secret [HJKY95]. This effectively re-randomizes the shares, and ensures that shares the adversary learns from different epochs cannot be combined. An alternative strategy for refreshing is to have each party re-share their share, then use the linearity of the secret-sharing protocol to have each party locally reconstruct a new share of the original secret [CKLS02]. Other works [ELL20,MZW+19,YXD22] share using bivariate polynomials. To achieve privacy against *malicious* adversaries, the underlying secret sharing protocol can be replaced with a Verifiable Secret Sharing protocol (e.g. [Fel87]).

Some PSS protocols (e.g. [SLL10,BDLO15]) consider *dynamic committees*, i.e., they assume that committees in different epochs may contain different (possibly disjoint) sets of parties, and that the threshold may also change between epochs. Some PSS protocols (e.g. [CKLS02][ZSVR05][SLL10][YXD22]) consider an *asynchronous* model of communication, meaning that although parties are synchronized across epochs, messages can be arbitrarily delayed by the adversary *within* an epoch. In this work, we consider *synchronous communication*.

The goals of PSS protocols are to tolerate a higher corruption threshold (usually $n/2$ or $n/3$) and to reduce communication complexity. Every previous PSS protocol requires all-to-all communication during the refresh phase, and thus every PSS protocol has at least $O(n)$ communication per party per epoch (and many have $O(n^2)$ or even $O(n^3)$). One way to improve *amortized* communication complexity is to consider batches of secrets, which can then be refreshed simultaneously [BDLO15,BDLO15,ELL20]. By considering batches of $O(n)$ secrets, some PSS protocols are able to achieve *amortized* constant in n communication complexity per party per epoch. This work is the first to achieve communication complexity that is constant in n *without amortization* (see Section 7).

One interesting feature of the mobile-adversary model is the problem of how secure channels are created and maintained between the parties. Essentially all multiparty protocols assume the parties are connected via secure, authenticated channels. In most situations, these secure channels can be achieved via a PKI – each party has a key pair for an authenticated encryption scheme. Unfortunately, in the mobile-adversary setting the existence of a PKI can no longer create secure channels, since once an adversary has corrupted a party, they would learn the party's long-term secret keys and would be able to impersonate that party and decrypt all messages to that party in future epochs. This problem was explored in depth in [CHH97], but their solution is rather cumbersome and re-establishing secure channels every epoch requires at least $O(n)$ communication per party. Many PSS protocols (e.g. [OY91,CKLS02,BEDLO14,MZW+19,YXD22]) still assume that all parties are connected via secure channels.

In Section 8 we give a simple solution to the problem of reinstating secure channels in the mobile adversary model, assuming each party has access to a lightweight signing oracle (such as can be found in any modern smartphone or hardware-based cryptocurrency wallet). Our solution for regenerating channel keys can be used with any existing PSS protocol. It is very light—it only requires $\mathcal{O}(\kappa)$ communication to establish a channel—so is compatible with our low-communication PSS protocols.

3 Model

For the full model details see Appendix C.

Secrets and Shares. We assume that there is a single secret, denoted s, from some group $\mathbb{F}$, that is (honestly) distributed by a trusted dealer before the protocol begins, resulting in each party holding a share. In addition, we require that the dealer distributes initial PRG keys.

Epochs. We divide time into *epochs* consisting of two phases, *refresh* and *retain*. The PSS protocol describes the *refresh* phase, while the *retain* phase encompasses what parties do with their share outside of the PSS protocol.

1. **Refresh:**
   (a) Reboot

(b) Establish secure channels
(c) Send messages
(d) Securely delete old share (everything except current private key)
(e) Receive messages
(f) Securely delete keys (everything except new share)

2. **Retain:** Parties may use their share, e.g. in the context of an MPC protocol.

Mobile Adversaries. The set of parties in the protocol is denoted $\{P\}_{i=1}^{n}$, and communication is *synchronous*.

The adversary, $\mathcal{A}$, is *mobile*, which means that it can corrupt t (out of n) parties in each epoch, where t is a function of n. When $\mathcal{A}$ corrupts a party, it is allowed to see all its messages. If $\mathcal{A}$ is malicious, it can cause the party to deviate from the protocol. Furthermore, $\mathcal{A}$ is *rushing*.

We assume parties can securely delete data and have access to fresh randomness. We instantiate secure, authenticated channels between parties using a (hardware-based) signing oracle. Alternatively, we can simply assume the existence of secure channels.

Reboots. To handle such an adversary, we assume that it is possible to remove the adversary's control of a party by a *reboot* operation. Rebooting a party will cause the adversary to lose all access to new information and will cause the party to return to executing the correct program.

A party is *corrupted* if it has been corrupted, but not (yet) been rebooted. It is *honest* otherwise. By periodically applying reboots, we can limit the number of parties that are corrupted at any time.

Counting Corruptions. A party which is corrupted during the retain portion of epoch t is considered corrupt, and counted against the budget of the adversary in epoch t. As in [HJKY95], we consider that when an adversary corrupts a party during the *refresh* phase of epoch t, this counts towards the adversary's corruption budget of epoch t *and* epoch $t-1$.

When the committee in epoch $t+1$ is disjoint from the committee in epoch t, there is no need to double count parties who are corrupted during the refresh phrase. Thus it is typical, when considering dynamic committees, to give the adversary the power to corrupt up to k-out-of-n parties in the old committee as well as k-out-of-n of parties in the new committee.

Security. Most PSS protocols simultaneously achieve both *privacy* and *robustness*. *Privacy* ensures that the adversary gains no advantage in guessing the secret. *Robustness* ensures that the adversary cannot cause the reconstructed value to differ from the secret which was shared. In this work, we will sometimes consider these two properties separately.

For both private and robust protocols, we will show protocols secure against *malicious* (active) adversaries. Our protocols will either provide *perfect* security, ensuring that a property (privacy or robustness) always holds, or they will provide *computationally* security, ensuring that a property holds except with non-negligible probability against a computationally bounded adversary.

Reconstruction. In our protocols it is impossible to guarantee that *every* honest party holds a valid share at every step of the protocol. Since this type of "eclipse attack" is unavoidable in our model, we consider a slightly different form of correctness in our constructions. We consider a PSS protocol secure if, in any given epoch, there exists a reconstruction protocol, which would allow the (honest) parties to reconstruct the original secret. The key distinction here is that the reconstruction procedure may require linear communication (e.g. all parties send their shares to every other party), but since the reconstruction procedure is not actually run in each epoch, the amortized communication per epoch can still be sub-linear.

4 Expander Graphs

The key tool in our protocols is expander graphs. These are graphs which, despite a small number of edges, remain well connected, for certain metrics of connectedness. In particular we will examine *bipartite* graphs, that is $G = (L \cup R, E)$ where $E \subset L \times R$. Our graphs will be *balanced*, that is $|L| = |R| = n$. Furthermore, our graphs will be d-regular, that is every vertex (whether in the "left" side L, or the "right" side R) will have exactly d neighbors, where d is a constant.[1]

The metric of connectedness that is most relevant to our work is *vertex expansion*, which is formally defined below:[2]

Definition 1 (Vertex expansion). *A bipartite graph $G = (L \cup R, E)$, is called a (γ, α)-expander if for every set $S \subset L$, with $|S| \leq \gamma n$, and letting $N(S)$ represent the set of neighbors of vertices in S, we have*

$$|N(S)| \geq \alpha |S| \tag{1}$$

Concretely, we use bipartite d-regular Ramanujan graphs. These are expander graphs that are essentially optimal according to another metric: spectral expansion. Appendix B contains a more detailed explanation of Ramanujan graphs and spectral expansion, as well as standard proofs that they have the properties we require (Theorems 1 and 2 below). Bipartite d-regular Ramanujan graphs can be constructed in polynomial time for all degrees and sizes [MSS13] [MSS18] [Coh16]. Since Ramanujan graphs have optimal spectral expansion, they also have good vertex expansion:

Theorem 1. *A Ramanujan graph is a $\left(\gamma, \frac{1}{(1-\gamma)\frac{4}{d}+\gamma}\right)$ expander $\forall\ \gamma \in [0, 1]$.*

Essentially, the property above will be useful when, if a party has one good neighbor, it will also be good, for some definition of good to be defined later. In other situations, a party will only be good if it has a *majority* of good neighbors. In such cases, we will need the following property of Ramanujan graphs.

[1] A d-regular bipartite graph is always balanced since, $|E| = d|L| = d|R| \Rightarrow |L| = |R|$.

[2] While this definition is valid for the case $\alpha \leq 1$, we will only be interested in the case where $\alpha > 1$, i.e. there is actual *expansion*.

Theorem 2. *Ramanujan graphs have the following property. Let S be a set of size at most δn vertices on the left. Then at most*

$$\frac{4\delta n}{(\frac{1}{2}-\delta)^2 d} \tag{2}$$

right-hand vertices have at least $\frac{1}{2}$ of their neighbors in S.

5 $\mathcal{O}(n)$-Private PSS with Constant Communication

In this section we present a PSS protocol that is perfectly private (but not robust) in the presence of an adversary that can corrupt up to δn parties per epoch, for some constant $0 < \delta < 1$.

Remark 1. In the case of *passive* adversaries, the privacy-only PSS protocol described is actually a full-blown PSS protocol, since passive adversaries cannot modify the shares. In this section, we prove a slightly stronger result, that the protocol achieves privacy in the face of an active (malicious) adversary.

As a warmup, consider the following simple (private-only) PSS protocol. The secret, s, is additively distributed among the players. That is, in epoch t, party P_i holds $s_i^{(t)}$ where $\sum_{i=1}^{n} s_i^{(t)} = s$. To refresh each party additively reshares its share to all other parties. Then, by summing the shares-of-shares it receives, each party gains a new re-randomized share of the secret original secret.

In our protocol, instead of each party additively resharing its share to all other parties, it only reshares to a constant number of neighbors. These neighbors are chosen according to an expander graph.

Definition 2 (Choosing Neighbors according to a Graph). *Let $G = (V, E)$ be a bipartite graph, with parts $L = \{L_1, \ldots, L_n\}$ and $R = \{R_1, \ldots, R_n\}$. If a protocol with parties $P_1, \ldots, P_n$,* chooses neighbors according to *G it means that P_j is a neighbor of P_i iff $(L_i, R_j) \in E$. Note that the neighborhood relation is not reflexive. Let $N(i)$ return the indices of the neighbors of P_i, and $N^{-1}(j)$ return the indices of parties that P_j is a neighbor of.*

Remark 2 (Fixed graph). The graph G will always be public and fixed. Thus, the attacker can therefore choose its corruptions with full knowledge of G.

At the beginning of each epoch, each party holds an additive share of the secret, $s \in \mathbb{F}$. Party $P_i^{(t)}$ will hold a single share $s_i^{(t)} \in \mathbb{F}$, where for every epoch t, $\sum_{1 \le i \le n} s_i^{(t)} = s$. The secret is reshared according to a constant-degree bipartite expander. This makes it very efficient, as each party only has to send a constant number of messages. [3] The expansion property of the underlying graph, G, will

[3] In order to instantiate secure channels as described in Section 8, each party will also have to send messages to its neighboring parties, but this will not change the fact that each party only communicates with $O(1)$ other parties in each epoch.

guarantee that a mobile adversary (controlling a constant fraction of the parties in each epoch) will not learn enough shares to reconstruct the secret.

Protocol 1 describes a scheme that achieves $\Theta(n)$ proactive privacy with only $\Theta(1)$ communication per player, yet it does not provide robustness.

Proactively Private Efficient Resharing

Parameters:
Let $G = (L \cup R, E)$ be a d-regular bipartite Ramanujan (γ, α) expander, with parts $L = \{L_1, \ldots, L_n\}$ and $R = \{R_1, \ldots, R_n\}$. We choose neighbors according to graph G.

1. **Setup:**
   The dealer divides the secret using an additive secret sharing, i.e., $P_i^{(1)}$ receives $s_i^{(1)}$ for $1 \leq i \leq n$, where $s_i^{(1)}$ are chosen uniformly at random from $\mathbb{F}$ subject to the constraint that $\sum_{i=1}^{n} s_i^{(1)} = s$.
2. **Resharing:**
   (a) At the start of epoch t, party P_i generates a share-of-share $s_{i,j}^{(t)}$ for each neighbor P_j and sends the message to P_j. The shares are chosen uniformly at random subject to the constraint $\sum_{j \in N(i)} s_{i,j}^{(t)} = s_i^{(t)}$. P_i sends $s_{i,j}^{(t)}$ to P_j.
   (b) P_j receives values $s_{i,j}^{(t)}$ for each $i \in N^{-1}(j)$. It computes: $s_j^{(t+1)} = \sum_{i \in N^{-1}(j)} s_{i,j}^{(t)}$.

Protocol 1.

Theorem 3. *Protocol 1 is a correct resharing, i.e., the constructed secret would remain the same if all parties follow the protocol.*

Proof. By induction. For epoch 1, $\sum_{i=1}^{n} s_i^{(1)} = s$.

Assume for epoch t, $\sum_{i=1}^{n} s_i^{(t)} = s$. Then for epoch $t+1$,

$$\sum_{j=1}^{n} s_j^{(t+1)} = \sum_{j=1}^{n} \sum_{i \in N^{-1}(j)} s_{i,j}^{(t)} = \sum_{(L_i, R_j) \in E} s_{i,j}^{(t)} = \sum_{i=1}^{n} \sum_{j \in N(i)} s_{i,j}^{(t)} = \sum_{i=1}^{n} s_i^{(t)} = s.$$

We now demonstrate that this protocol maintains privacy against a mobile adversary who can corrupt $\mathcal{O}(n)$ parties per epoch. There are essentially three ways that a mobile adversary can learn a party's share in a given epoch: it can corrupt a party in the current epoch, or it corrupts all the party's neighbors in the previous epoch, or all of the party's neighbors in the subsequent epoch.

To prove the privacy of Protocol 1, let us consider the communication graph, H. We will represent parties as vertices and messages as edges. Since whether a party is corrupt or honest depends on the epoch, we will actually have a different vertex for every party in every epoch. Vertex $H_i^{(t)}$ will represent P_i in epoch t. If P_i is corrupted in epoch t, we also call vertex $H_i^{(t)}$ corrupted; otherwise we call the vertex honest. We let $H^{(t)} = \{H_1^{(t)}, \ldots, H_n^{(t)}\}$, i.e. all vertices that represent parties from epoch t. We call $H^{(t)}$, *layer* t of the graph H. There are therefore at most δn corrupted vertices in each layer of H.

We put a directed edge[4] from $H_i^{(t)}$ to $H_j^{(t+1)}$ if P_i sends a message to P_j in epoch $t+1$. Since communication is according to expander G, edge $(H_i^{(t)}, H_j^{(t+1)})$ exists in H if and only if (L_i, R_j) is an edge in G. To make the graph finite, we set some arbitrarily large upper limit, T on the number of epochs.

We will be able to prove privacy of Protocol 1 by examining *paths* in H. In particular, we are concerned with *honest* paths, which are paths in which every vertex is honest. Recall that edges are directed; paths will follow the same orientation as edges. Since all edges are from a vertex from some layer t to a vertex in layer $t+1$, the vertices in a path will be from contiguous layers. We call a path *ancient* if the first vertex in the path is in $H^{(1)}$.

We now prove some properties of the graph H. This will later allow us to prove the desired security properties of Protocol 1.

Lemma 1. *Let γ and α be constants such that G is a (γ, α) expander. Let H be defined as above. If there are at most δn corrupted vertices per layer, and $\delta \leq \gamma(\alpha - 1)$ then for every t, there exist at least γn vertices in $H^{(t)}$ that are part of ancient honest paths.*

Proof. First, note that for any expander, $\gamma\alpha \leq 1$, so $\delta \leq \gamma(\alpha - 1)$ also implies:

$$\delta \leq \gamma\alpha - \gamma \Rightarrow \delta \leq 1 - \gamma \Rightarrow \gamma \leq 1 - \delta$$

We show by induction that for any $1 \leq t \leq T$, there exist at least γn vertices in layer t that are part of ancient honest paths.

For $t = 1$, any honest vertex is on an ancient honest path consisting only of itself. There are at least $(1 - \delta)n$ honest vertices, and $(1 - \delta) \geq \gamma$.

Assume for epoch t. We now show it holds for epoch $t + 1$. If $H_i^{(t+1)}$ is honest, and is a neighbor of some vertex $H_j^{(t)}$ that is part of an ancient honest path, then appending $H_i^{(t+1)}$ to this path results in a path that is still ancient and honest and includes $H_i^{(t+1)}$. By induction, there are at least γn vertices in epoch $H^{(t)}$ that are part of ancient honest paths. Due to the expansion property, there must be at least $\alpha\gamma n$ vertices in epoch $H^{(t+1)}$ that are neighbors of these

[4] This assumes a secure channel is already established between P_i and P_j. If Protocol 4 is used to re-establish a secure channel, P_j will also need to send messages to P_i, but we do not represent this on the graph. Also, if a corrupted P_i *should* send a message to P_j but doesn't, we consider this as P_i sending some default message.

vertices, at most δn of which are corrupted. Therefore, there are at least $(\alpha\gamma-\delta)n$ vertices in $H^{(t+1)}$ are part of ancient honest paths. $\delta \leq \gamma(\alpha-1)$, so $(\alpha\gamma-\delta)n \geq (\alpha\gamma-(\alpha-1)\gamma)n \geq \gamma n$. Thus, by induction, at least γn vertices in $H^{(t+1)}$ are part of honest ancient paths.

Note that if vertex $H_i^{(t)}$ is on an honest ancient path, this does not guarantee that $\mathcal{A}$ does not learn P_i's share in epoch t. It guarantees that $\mathcal{A}$ did not learn P_i's share directly by corrupting it or by learning all messages it received. However, if $\mathcal{A}$ corrupts all of P_i's neighbors in epoch $t+1$ it will learn all messages P_i sent and thus learn P_i's share in epoch t.

However, the fact that there are honest paths to all future epochs $t' > t$, implies that there is at least 1 vertex in epoch t which is part of these paths, and for which $\mathcal{A}$ did not learn the outgoing messages. This is essentially sufficient to show that privacy is preserved. Formally, Lemma 5 implies the following:

Corollary 1. *If $\delta \leq \gamma(\alpha-1)$ there exists an honest path from $H^{(1)}$ to $H^{(T)}$.*

We will now use this property of H to prove the security of Protocol 1.

Lemma 2. *If there exists an honest path from $H^{(1)}$ to $H^{(T)}$, then for all possible secrets $s_A, s_B \in \mathbb{F}$, the probability that $\mathcal{A}$ guesses output s_A when $s = s_A$ is the same as the probability that $\mathcal{A}$ guesses s_A when $s = s_B$.*

Proof. Recall that H represents the communication network of the protocol. Therefore, the existence of an honest path from $H^{(1)}$ to $H^{(T)}$ means that there are a sequence of parties, $P_{f(1)}, \ldots P_{f(T)}$ such that $P_{f(t)}$ is honest in epoch t and that $P_{f(t+1)}$ is a neighbor of $P_{f(t)}$. This means that $\mathcal{A}$ does not see the shares that these parties hold in the epochs in which they are honest: $s^{(1)}_{f(1)}, \ldots, s^{(T)}_{f(T)}$. Nor does $\mathcal{A}$ see the messages sent between these parties in the epochs in which they are honest: $s^{(1)}_{f(1),f(2)}, \ldots, s^{(T-1)}_{f(T-1),f(T)}$.

Since $\mathcal{A}$ cannot see these messages and shares, it is possible for them to be modified without $\mathcal{A}$ being able to detect it. Clearly, consistency has to be maintained: a share must be the sum of all messages received in that epoch. Likewise the messages sent in an epoch must sum to the share. If these shares and messages were all incremented by some value Δ, consistency would be maintained. Each party on the path would receive one message that was Δ larger, would hold a share that was Δ larger and would send one message that was Δ larger.

We can therefore consider 2 executions. In one, the secret is s_A. In another the secret is s_B and all messages and shares along the path are incremented by $\Delta = s_B - s_A$. All other messages and shares are the same in both executions. Therefore, the information available to $\mathcal{A}$ is the same in both executions.

The probability of the first execution occurring when $s = s_A$ is exactly the same as the probability of the second execution occurring when $s = s_B$. Most parties will have the same inputs and outputs in both executions, and so both events will occur with the same probability. Likewise, $\mathcal{A}$ is not able to see anything different in the two executions, so all actions chosen by $\mathcal{A}$, including the behavior of parties it controls, will be the same in both executions. This is true

whether $\mathcal{A}$ sends correct outputs or not, i.e., it holds true even for a malicious $\mathcal{A}$. The only parties that receive or send different messages are the dealer and $P^{(1)}_{f(1)}, \ldots, P^{(T)}_{f(T)}$.

The dealer generates shares randomly subject to the sum being equal to the secret. Therefore, the probability that it chooses any sequence of initial shares to send to all parties other than $P_{f(1)}$ is equal ($|\mathbb{F}|^{-(n-1)}$) in both executions. The final share, sent to $P_{f(1)}$ is determined by the other shares chosen. Likewise, each honest party on the path chooses shares-of-shares randomly subject to the sum being equal to their secret share. Therefore, the probability of the party choosing any sequence of shares-of-shares to send to parties that are not on the path (namely $|\mathbb{F}|^{-(d-1)}$) is the same in both executions. The share-of-share sent on to the next honest party on the path will be uniquely determined by the other shares-of-shares.

Therefore, for every execution where $s = s_A$ and $\mathcal{A}$ outputs s_A, there is another execution that when $s = s_B$ causes $\mathcal{A}$ to output s_B with the same probability. Summing over the finite set of all possible executions, we have that for all $s_A, s_B \in \mathbb{F}$, $Pr(\mathcal{A} \text{ outputs } s_A | s = s_A) = Pr(\mathcal{A} \text{ outputs } s_A | s = s_B)$.

Lemma 2 implies that $\mathcal{A}$ obtains no advantage in determining the secret by participating in the protocol. This holds provided there is an honest path from $H^{(1)}$ to $H^{(T)}$, which from Corollary 1 we know happens if $\delta \leq \gamma(\alpha - 1)$. Furthermore, since we instantiate with a Ramanujan graph, Theorem 1 shows that $\gamma(\alpha - 1) \geq (1 - \gamma)\frac{d-4}{d-4+\frac{4}{\gamma}}$. Some basic calculus shows that that this is maximized by $\gamma = \frac{2\sqrt{d}-4}{d-4}$, for which the value is $\frac{\left(\sqrt{d}-2\right)^2}{d-4}$. This shows that Protocol 1 provides the following privacy guarantee:

Theorem 4. *If $\delta \leq \frac{\left(\sqrt{d}-2\right)^2}{d-4}$, Protocol 1 provides perfect privacy against (malicious) adversaries controlling at most δn parties per epoch.*

Table 2 presents some example values of δ and the smallest necessary value of d that ensures privacy given δn corruptions per epoch. For instance, for $d = 22$ it is possible to tolerate 40% of parties being corrupted per epoch.

δ	0.1	0.2	0.3	0.4	0.5	0.6	0.7
γ	0.45	0.40	0.35	0.30	0.25	0.20	0.15
d	6	9	14	22	36	64	129

Table 2: Corruption threshold, δ, as a function of the bandwidth cost, d for the privacy-only construction (Theorem 4).

6 $\mathcal{O}(n)$-Robust Only PSS with Constant Communication

The semi-honest construction (Section 5) can be adapted to provide *robustness*, but not *privacy*. In this scheme, the "secret" message, s, is known in the clear.

The scheme aims to ensure that the message is not changed despite a large number of malicious parties and a small amount of communication per party.

Recall that time is divided into epochs. As before, the adversary is allowed to corrupt δn of the parties from each epoch. In this setting, each party P_i, in each epoch t, holds a value, $s_i^{(t)}$. Our protocol will ensure that the majority of nodes in a committee hold the correct value.

Note that (as discussed in Section C.7) because of "eclipse attacks" we cannot guarantee that *all* honest parties hold $s_i^{(t)} = s$ in every epoch t. Instead, we ensure that the *majority* of parties hold the correct value. This allows the true value to be reconstructed by a simple majority vote.

We define *deceived* nodes to be nodes that are honest but hold and send incorrect values because they have received incorrect values. This is a departure from standard PSS and Byzantine models. Due to this relaxation, we are able to obtain asymptotically optimal ($\Theta(n)$) robustness with only $\mathcal{O}(1)$ communication per party. Specifically, we guarantee that the number of *compromised* nodes, that is nodes that are either deceived or corrupt, remains a minority.

Since we guarantee that the majority of nodes are always uncompromised, it is always possible to use an $\mathcal{O}(n)$-communication reconstruction step which will allow each honest node to receive the correct value. If every node broadcasts its value to every other node, then the majority of values any node receives will be correct. If each honest party then sets its value to the most common value it received then every honest party will have the correct value. This step only needs to occur when we wish to return to a situation where every honest node holds the correct value. For the sake of simplicity we omit further discussion of the standard model and will focus on the model where we only need a majority of uncompromised nodes.

The scheme is shown in detail in Protocol 2. It achieves $\Theta(n)$ proactive robustness with only $\Theta(1)$ communication per player. However, it does not provide any privacy as the "secret" is seen by every node.

Robust only

Parameters:
Let $\hat{G}$ be a $\hat{d}$-regular Ramanujan bipartite expander graph with n vertices in each part. Choose neighbors according to $\hat{G}$.

1. **Setup:** s is the "secret". Dealer sends each party P_i the value $s_i^{(1)} = s$.
2. **Resharing:**
   (a) At the start of epoch t, party P_i sends its share to all of its neighbors. Let $s_{i,j}^{(t)}$ denote the message P_i sends to neighbor P_j in epoch t.
   (b) P_j sets its new share to the majority of messages it received, i.e. $s_j^{(t+1)} = \text{majority}_{i \in N^{-1}(j)}(s_{i,j}^{(t)})$. If there is no majority, $s_j^{(t+1)} \stackrel{\text{def}}{=} \perp$.

Protocol 2.

The fact that the protocol has $\Theta(1)$ communication per player is evident from the fact that each player sends a single message to each of its $\hat{d}$ neighbors in the expander and that $\hat{d}$ is constant. We will now show that the protocol provides robustness against a *malicious proactive* adversary controlling δn parties in each epoch, for any constant $0 < \delta < \frac{1}{2}$.

First we will formalize our terminology. A node P_i is *deceived* in epoch t if it is honest in epoch t, but $s_i^{(t)} \neq s$. A node is *compromised* if it is either malicious or deceived.

Theorem 5 (Security of Protocol 2). *Protocol 2 guarantees that in each epoch, there is a majority of uncompromised nodes, provided $\mathcal{A}$ corrupts at most δn nodes in each epoch, for some constant $\delta < \frac{1}{2}$.*

Proof. Select some constant ϵ such that $\delta < \epsilon < \frac{1}{2}$. We show there exists some constant $\hat{d}$ such that, if $\hat{G}$ is a $\hat{d}$-regular Ramanujan bipartite expander, then the number of compromised nodes in any epoch is at most ϵn.

By induction. In epoch 1, there are δn corrupt nodes and no deceived nodes, so there are $\delta n < \epsilon n$ compromised nodes.

Assume that the statement holds until epoch t. Let X be the set of compromised nodes in epoch t. By the inductive hypothesis $|X| \leq \epsilon n$. Let Y be the set of deceived nodes in epoch $t+1$. A node will be deceived only if at least half of the messages it received were incorrect.

Applying Theorem 2, where S is the nodes that are compromised, we obtain:

$$|Y| \leq \frac{4\epsilon n}{\hat{d}\left(\frac{1}{2} - \epsilon\right)^2}$$

The number of corrupt nodes in epoch $t+1$ is at most δn, so the total number of compromised nodes in epoch $t+1$ is at most:

$$\frac{4\epsilon n}{\hat{d}\left(\frac{1}{2} - \epsilon\right)^2} + \delta n$$

If $\hat{d} \geq \frac{4\epsilon}{(\frac{1}{2}-\epsilon)^2(\epsilon-\delta)}$ then the number of compromised nodes in epoch $t+1$ is at most $(\epsilon - \delta)n + \delta n \leq \epsilon n$. Thus, by induction there are at most ϵn compromised nodes in every epoch. Since $\epsilon < \frac{1}{2}$, most nodes in each epoch are uncompromised.

The above proof works for every ϵ satisfying $\delta < \epsilon < \frac{1}{2}$. A simple calculus proof, delegated to Appendix E, shows that the expression is minimized by $\epsilon = \frac{1}{4}\left(\delta + \sqrt{\delta^2 + 4\delta}\right)$. For instance, for $\delta = 0.1$ this yields the requirement that $\hat{d} \geq 88$.

7 $\mathcal{O}(n^a)$-Private, $\mathcal{O}(n^{1-a})$-Robust PSS with $\mathcal{O}(\kappa)$ Communication

The PSS protocols presented in Sections 5 and 6 are extremely limited in that the first protocol does not provide any robustness (a malicious adversary can

modify the secret) and the second does not provide any privacy (every party knows the "secret"). In this section we combine the two protocols to create a protocol that has both privacy and robustness, but still has the desired constant (in n) communication per party per epoch. Specifically, we present a protocol that has privacy against a proactive adversary corrupting $\Theta(n^a)$ nodes each epoch and robustness against a proactive adversary corrupting $\Theta(n^{1-a})$ nodes per epoch, requiring $\Theta(\kappa)$ communication per party per epoch, where κ is a security parameter. The protocol is perfectly robust, and computationally private, such that the adversary's advantage in guessing the secret is negligible in κ. Setting $a = \frac{1}{2}$ provides a constant-communication PSS with both privacy and robustness against a proactive adversary corrupting $\Theta(\sqrt{n})$ nodes per epoch.

At a high-level, we start our construction with the private protocol (Protocol 1) and replicate each party, say P_i, of that protocol some number of times. We consider this set of replicas of P_i as if they are simulating P_i's actions. However, they will do it with a twist; they will utilize the robust protocol (Protocol 2) when they send a message on behalf of P_i. The robust protocol will ensure that no messages or shares are lost and the underlying private protocol will ensure that there is privacy for the global secret, delivering the desired result.

However, things are not straightforward; there are two obstacles which need to be overcome. The first is that for this general idea to work we need to guarantee that the replicas in fact work as replicas. That is, if they are not compromised (i.e. not corrupted or deceived) then they will execute the same steps with the same inputs and randomness, otherwise the replicas will be sending different messages. This is a challenging requirement to satisfy in the proactive setting. The second issue is that we cannot have a replica of one party send message to all the replicas of another party as this will increase the communication complexity beyond our goals. Thus, to deliver a solution we need to address these two problems.

Recall that in Protocol 1, the parties use fresh randomness to generate the shares-of-shares. As described the fresh randomness is unique to each party and is generated locally at the time it is needed. Note that we cannot generate randomness from long-term shared PRG keys, as a proactive adversary can learn all such keys and know the pseudorandomness being used by every party. Thus, it seems that, as we require fresh randomness and at the same time need replicas to have the same randomness, we are stuck in a bind.

To solve this, parties refresh the PRG keys of their neighbors in every epoch. That is, each party, each epoch, sends their neighbors both a share-of-share, and a string, called a re-randomizer. A party combines the re-randomizers it receives to generate a new PRG key. How does a party generate these re-randomizers? It uses its own PRG key for that epoch. This may seem circular since an adversary who corrupts a party will learn the re-randomizers that it sends. Security comes from combining multiple re-randomizers to create the new key, and choosing neighbors using an expander graph. Like Protocol 1 ensured a constant fraction of shares remained private each epoch, this will ensure a constant fraction of keys remain private. Our solution is therefore also Proactive Pseudorandomness

(PP) protocol, that is a protocol that generates pseudorandomness in a way that is indistinguishable from random to a mobile adversary. See Section F for more details on PP and a simplified version of our protocol that just provides Proactive Pseudorandomness.

Since pseudorandomness is generated according to PRG keys, we can consider a *correct* execution in which parties always generate their messages according to the keys. This execution is deterministic given the dealer's initial distribution of keys and secret shares. We can consider the shares and messages of this correct execution as the *correct* shares and messages. To show the robustness of the protocol, we will show that, every epoch, for any party in the privacy-only protocol, most of its replicas hold the correct share.

Having resolved the randomness issue, we have made a step forward towards making replication possible. Now we need to address the issue of not having a replica send messages to all the replicas of its neighbor. To attain robustness, at a low communication cost we will have a replica of a party send its messages only to a small subset of its neighbor's replicas. We will show that robustness is maintained despite this dramatically lower communication.

Concretely, we instantiate Protocol 1 with n^a parties, but in our protocol each of these will be simulated by n^{1-a} replicas. These replicas will be the actual parties running the protocol; the fact that they are simulating an execution of Protocol 1 is a useful abstraction. We label the parties as if they were in a n^a by n^{1-a} grid, with row i holding the replicas of P_i from Protocol 1. $P_{i,j}$ denotes the party in row i and column j. We denote the set of parties in row i as row_i and the set of parties in column j as col_j. If we wish to specify that we are referring to a row (resp. column) in a specific epoch t, we use the notation $row_i^{(t)}$ (resp. $col_j^{(t)}$).

In more detail, examine party P_i from the private protocol that is replicated some number of times. If P_i sent $P_{i'}$ share-of-share $s_{i,i'}^{(t)}$ in epoch t in Protocol 1, then each uncompromised replica of P_i will also send replicas of $P_{i'}$, the share $s_{i,i'}^{(t)}$ in epoch t of the new protocol. We will ensure the majority of replicas of P_i will send the correct share-of-share. Thus, the replicas of P_i in row_i will send messages to the replicas of $P_{i'}$ in $row_{i'}$. Unfortunately, making every party in row_i communicate to every party in $row_{i'}$ causes the communication complexity to scale linearly in the amount of replication.

How many parties do they need to communicate to in order to ensure that the majority of parties in any row always hold the correct share? Surprisingly, a constant number suffices. The argument is almost identical to that of the robustness of Protocol 2. To explain this, let's restrict our view to one replica of P_i, say $P_{i.\ell}$. Examine the replicas of party $P_{i'}$ of which it needs to choose a subset to communicate with. The expander graph of the robust protocol will tell us with which replicas of $P_{i'}$ the replica $P_{i,\ell}$ should talk to, i.e. the columns that identify the subset of the replicas of $P_{i'}$. We state two important points that will aid in the proofs. The replica of P_i also needs to talk to replicas of a party $P_{i''}$, as P_i communicates with $P_{i''}$ in the private protocol. The subset of replicas of $P_{i''}$ will be in exactly the same columns as the replicas of $P_{i'}$. Furthermore, assume

that P_j also communicates with $P_{i'}$ in the private protocol. Then, the replica $P_{j,\ell}$ of P_j will talk to the subset of the same columns as the replica $P_{i,\ell}$. Saying this abstractly we have that *column* col_j will only communicate with column $col_{j'}$ if, in an instantiation of Protocol 2 with n^{1-a} parties P_j would communicate with $P_{j'}$. $P_{i,j}$ only communicates with $P_{i',j'}$ if row row_i communicates with $row_{i'}$ *and* column col_j communications with $col_{j'}$.

One final challenge is that malicious adversaries can choose to send incorrect randomness in an attempt to create related keys for a Related-Key Attack (RKA) on the PRG. To solve this, we use a PRF that is secure against additive RKAs to securely combine the randomness sent to a party. This ensures that if any of the messages is unknown to the adversary, it will be unable to distinguish the PRG seeds from ones that were truly generated at random. We instantiate with the additive-RKA-secure PRF of Bellare and Cash [BC10], which was proven secure under DDH by [ABPP14]. This PRF is a variant of the Naor-Reingold PRF, and like Naor-Reingold it has $\Theta(\kappa^2)$ bits per key ($\Theta(\kappa)$ values from a group where DDH is hard). A simple solution would be for each party to send $\Theta(\kappa^2)$-bit rerandomizers which would be added to form a key for an additive-RKA-secure PRF. However, it is not actually necessary for each party to send $\Theta(\kappa^2)$ bits. In our protocol each party instead sends a κ-bit PRG seed, which the recipient expands to generate the $\Theta(\kappa^2)$-bit rerandomizers, which are then added to create the key for the additive-RKA-secure PRF.

We set the parameters of the protocol as follows: a is a constant such that $0 < a < 1$. n is the number of parties, and n^a and n^{1-a} are both integers. The parties are arranged in an n^a by n^{1-a} grid, and are labeled $P_{i,j}$ for $1 \leq i \leq n^a$ and $1 \leq j \leq n^{1-a}$, such that $P_{i,j}$ is in row i and column j. $P_{i,j}$ in epoch t is represented as $P_{i,j}^{(t)}$. The labels are public.

There are two bipartite expanders of constant degree, G which has n^a nodes in each part and will be used for the private portion, and H which has n^{1-a} nodes in each part and will be used for the robust portion. d_G (d_H) is the degree of G (H), respectively. GR_i (HR_j) represent the sets of indices of right-neighbors of L_i (L_j) in G (H) respectively. Likewise GL_i (HL_j) represent the sets of indices of left-neighbors of R_i (R_j) in G (H) respectively. Expanders are fixed and public.

$\mathbb{F}$ is a group from which the secret is chosen. K_1 is a group from which PRG seeds are chosen, $|K_1| = 2^\kappa$. K_2 is a group from which PRG re-randomizers are chosen, $|K_1| = 2^{\Theta(\kappa^2)}$. $F : K_2 \times X \to K_1$ is a Φ_{add}-RKA-PRF where X can be any PRF input set.

Proactively Private and Robust Efficient Resharing

1. **Setup:**
   The dealer picks $s_1^{(1)}, \ldots, s_{n^a}^{(1)}$ uniformly at random from $\mathbb{F}$ subject to $\sum_{i=1}^{n^a} s_i = s$, where s is the secret.
   The dealer picks $k_1^{(1)}, \ldots, k_{n^a}^{(1)}$ uniformly at random from K_1.

The dealer sends $(s_i^{(1)}, k_i^{(1)})$ to $P_{i,j}^{(1)}$ for all $1 \le i \le n^a$, $1 \le j \le n^{1-a}$.
$P_{i,j}^{(1)}$ sets $(s_{i,j}^{(1)}, k_{i,j}^{(1)})$ to the received value.

2. **Resharing:**
   (a) At the start of epoch t each party $P_{i,j}$ does the following:
   Uses $k_{i,j}^{(t)}$ as a PRG seed to pseudorandomly generate $r_{i,i',j}^{(t)} \leftarrow K_1$, $s_{i,i',j}^{(t)} \leftarrow \mathbb{F}$, for all $i' \in GR_i$, chosen uniformly at random, subject only to $\sum_{i' \in GR_i} s_{i,i',j}^{(t)} = s_{i,j}^{(t)}$.
   Sets $r_{i,i',j,j'}^{(t)} = r_{i,i',j}^{(t)}$, $s_{i,i',j,j'}^{(t)} = s_{i,i',j}^{(t)}$ for all $i' \in GR_i$, $j' \in HR_j$.
   Sends $(r_{i,i',j,j'}^{(t)}, s_{i,i',j,j'}^{(t)})$ to $P_{i',j'}$ for all $i' \in GR_i$, $j' \in HR_j$.
   (b) Each party $P_{i',j'}$ then does the following:
   Receives $(r_{i,i',j,j'}^{(t)}, s_{i,i',j,j'}^{(t)})$ from all $i \in GL_{i'}$, $j \in HL_{j'}$.
   Sets $r_{i,i',j'}^{(t)} = \text{majority}_{j \in HL_{j'}} r_{i,i',j,j'}^{(t)}$
   Sets $\hat{s}_{i,i',j'}^{(t)} = \text{majority}_{j \in HL_{j'}} s_{i,i',j,j'}^{(t)}$.
   Use $r_{i,i',j'}^{(t)}$ as a PRG seed to generate rerandomizers $\hat{k}_{i,i',j'}^{(t)} \leftarrow K_2$ for all $i \in GL_{i'}$.
   Computes a new PRG seed from the provided randomness:
   $\hat{k}_{i',j'}^{(t)} = \sum_{i \in GL_{i'}} \hat{k}_{i,i',j'}^{(t)}$
   $k_{i',j'}^{(t+1)} = F(\hat{k}_{i',j'}^{(t)}, 1)$.
   Combines shares-of-shares to get a new share of the secret:
   $s_{i',j'}^{(t+1)} = \sum_{i \in GL_{i'}} s_{i,i',j'}^{(t)}$.

Protocol 3.

Before proving properties of the protocol, we provide some definitions. A *corrupted row* is one in which there is at least one corrupted party, i.e. row $row_i^{(t)}$ is corrupted if there exists $j \in \{1, \ldots, n^{1-a}\}$ such that $P_{i,j}^{(t)}$ is corrupted. Two rows $row_i^{(t)}$ and $row_{i'}^{(t+1)}$ are *neighbors* if there exist $P_{i,j}^{(t)} \in row_i^{(t)}$, $P_{i',j'}^{(t+1)} \in row_{i'}^{(t+1)}$ such that $P_{i,j}^{(t)}$ sends a message to $P_{i',j'}^{(t+1)}$. This happens exactly when $(i, i') \in G$. We say that $row_{i_w}^{(w)}, row_{i_{w+1}}^{(w+1)}, \ldots, row_{i_{w+x}}^{(w+x)}$ is a *row path* if $row_{i_y}^{(y)}$ and $row_{i_{y+1}}^{(y+1)}$ are neighbors for all $w \le y \le w+x-1$. If a row path consists only of rows that are not corrupted, we say that it is an *honest row path*. Lastly, we call a row path *full* if it stretches from the first epoch (epoch 1) to the last epoch (epoch T), i.e. $row_{i_1}^{(1)}, \ldots, row_{i_T}^{(T)}$ is a full row path for any length-T index set, $i_1, \ldots, i_T$, where $i_t \in \{1, \ldots, n^a\}$ for $1 \le t \le T$. We sometimes refer to a full row path $row_{i_1}^{(1)}, \ldots, row_{i_T}^{(T)}$ simply by the sequence of indices it uses: $i_1, \ldots, i_T$.

These definitions are intentionally analogous to those in the proof of privacy for Protocol 1. The proof of security will also be similar, in that it will be shown that if an honest row path exists throughout the entire protocol execution, then

the privacy is preserved. However, the proof first needs to demonstrate that the adversary is not able to undermine security by using the fact that resharings are generated pseudorandomly.

We prove this by first comparing the adversary's view in two different executions. The first is an execution of Protocol 3. The second is an execution in which all PRG seeds in a full row path are generated truly at random. Now, they cannot be all generated independently. If $\mathcal{A}$ is a passive adversary the PRG seeds in a row will all be the same, but if $\mathcal{A}$ is malicious, the PRG seeds may differ, since $\mathcal{A}$ may provide nodes in the row with different randomness. Thus, we want the alternative execution to have nodes use the same PRG seeds exactly when they would have the same seeds in the original execution. We thus define the executions, or games, as follows.

Let $Game_{Real}$ denote an execution of Protocol 3. Given a full row path $R = row^{(1)}_{R_1}, \ldots, row^{(T)}_{R_T}$, $Game_{1,R}$ denotes an execution almost identical to Protocol 3 except for the way $k^{(t+1)}_{i',j'}$ is generated in part (b) of the Resharing step. If $P^{(t+1)}_{i',j'} \notin row^{(t+1)}_{R_{t+1}}$, it generates $k^{(t+1)}_{i',j'}$ in the normal way. However, if $P^{(t+1)}_{i',j'} \in row^{(t+1)}_{R_{t+1}}$, it communicates with all other parties in $row^{(t+1)}_{R_{t+1}}$ to identify the set of parties which have the same value for $\hat{k}^{(t)}_{i',j'}$. It then collaborates with the parties in this set to generate a new truly random value which all parties in this set then use for their PRG seeds $k^{(t+1)}_{i',j'}$.

Lemma 3. *If $R = R_1, \ldots, R_T$ is a full honest row path then any probabilistic polytime adversary, $\mathcal{A}$, is unable to distinguish $Game_{Real}$ from $Game_{1,R}$ except with negligible probability.*

Proof. By induction on the epoch t. The induction invariant is that $\mathcal{A}$ will know, at most, which parties from a row in the given epoch use the same PRG seeds, but will have a negligible advantage at guessing these values.

The setup does not differ between $Game_{Real}$ and $Game_{1,R}$. So initially the views are identical. Note that $\mathcal{A}$ knows that all values of $k^{(1)}_{R_1,j}$ are identical, but the value was chosen truly at random by the dealer, so $\mathcal{A}$ has no advantage in guessing it.

We now show that a Resharing step followed by a Reconstruct step preserves the invariant. We have that $\mathcal{A}$ knows which parties in $row^{(t)}_{R_t}$ have identical PRG keys. At worst, she learns the result of all messages sent by $row^{(t)}_{R_t}$ except those that are sent to $row^{(t+1)}_{R_{t+1}}$. However, by the security of the PRG, the portion of the PRG output $\mathcal{A}$ observes will give $\mathcal{A}$ negligible advantage in learning the seed. Therefore, this information provides negligible assistance in allowing $\mathcal{A}$ to distinguish the case where the PRG seed, $k^{(t+1)}_{i',j'}$ was generated using the PRF ($Game_{Real}$) and the case where it was generated truly at random ($Game_{1,R}$).

Additionally, the security of the PRG provides her no advantage in guessing the randomness sent from parties in $row^{(t)}_{R_t}$ to those in $row^{(t+1)}_{R_{t+1}}$. Specifically $\hat{k}^{(t)}_{R_t,R_{t+1},j'}$ is generated from a PRG seeded with $r^{(t)}_{R_t,R_{t+1},j'}$. This, in turn was

taken as the most common value of $r^{(t)}_{R_t,R_{t+1},j,j'}$ for $j \in HL_{j'}$. In $Game_{Real}$, these are generated by a PRG on $k^{(t)}_{R_t,j}$, whereas in $Game_{1,R}$ these are generated using a fresh random value (which is the same for any party holding an identical $k^{(t)}_{R_t,j}$). By our inductive hypothesis, these cases are indistinguishable to $\mathcal{A}$. Therefore, by the security of the PRG, the outputs $r^{(t)}_{R_t,R_{t+1},j,j'}$ are indistinguishable from uniformly random to $\mathcal{A}$ except that $\mathcal{A}$ knows (at worst) which are identical, and likewise are the computed values $r^{(t)}_{R_t,R_{t+1},j'}$. Therefore, again by the security of the PRG, the rerandomizer $\hat{k}^{(t)}_{R_t,R_{t+1},j'}$ is indistinguishable from uniformly random (except that $\mathcal{A}$ may learn which parties hold the same value).

Note that $\mathcal{A}$ may, in the worst case, know and be able to influence all other rerandomizers that a given party in $row^{(t+1)}_{R_{t+1}}$ receives. Thus, $P^{(t)}_{R_{t+1},j'}$ computes

$$\hat{k}^{(t)}_{R_{t+1},j'} = \hat{k}^{(t)}_{R_t,R_{t+1},j'} + \sum_{i \in GL_{i'}/\{R_t\}} \hat{k}^{(t)}_{i,R_{t+1},j'}$$

The second term is, at worst, known and controllable by $\mathcal{A}$. However, we have shown that the first term is indistinguishable from uniformly at random to $\mathcal{A}$. Multiple parties in $row^{(t+1)}_{R_{t+1}}$ may receive the same value as the first term, but $\mathcal{A}$ could introduce different values for the second term. This is equivalent to a Related-Key Attack, where the first term is the original key and the second term is an additive modification to the key chosen by $\mathcal{A}$. However, since F is a Φ_{add}-RKA-PRF, the outputs of F on different, additively-related keys are indistinguishable from random outputs. Thus, $\mathcal{A}$ will not be able to distinguish the seeds $k^{(t+1)}_{i',j'}$ in $Game_{Real}$ from the truly randomly generated seeds in $Game_{1,R}$. The outputs of F on identical keys will be the same, and again in $Game_{1,R}$, parties that received the same values of $\hat{k}^{(t)}_{R_{t+1},j'}$ will generate and use the same PRG seeds. Thus the indistinguishability of the two games is preserved after an epoch, and in particular the adversary may learn (at worst) which parties in the honest row path in that epoch have the same PRG seeds, but has no advantage in learning the seeds themselves.

Now, let $Game_{2,R}$ be equivalent to $Game_{1,R}$ except that rather than choosing a truly random seed for the PRG, parties that have the same value for $\hat{k}^{(t)}_{i',j'}$ generate a truly random string in place of the PRG output.

Lemma 4. *A probabilistic polytime adversary is unable to distinguish $Game_{2,R}$ from $Game_{1,R}$, except with negligible probability.*

Proof. This follows immediately from the definition of a PRG. In $Game_{1,R}$ the PRG seeds are chosen truly at random, and the outputs generated from this seed. A PRG has the property that an output of such a PRG is computationally indistinguishable from a truly random output, and thus $Game_{1,R}$ is computationally indistinguishable from $Game_{2,R}$.

We are now essentially in the same position as the proof of Protocol 1. The only difference is that replicas in an honest row may not agree on the same randomness to generate their messages (if $\mathcal{A}$ sends them inconsistent randomizers). Nevertheless, this does not undermine privacy, and we can proceed to prove privacy similar to as for Protocol 1 by considering the case that the secret-shares on the honest path, and all secret-share messages on the honest path, are incremented by some value $s_B - s_A$.

Lemma 5. *If there exists a full honest row path, R, then in $Game_{2,R}$, for all possible secrets $s_A, s_B \in \mathbb{F}$, the probability that $\mathcal{A}$ guesses output s_A when $s = s_A$ is the same as the probability that $\mathcal{A}$ guesses s_A when $s = s_B$.*

Proof. For every execution of $Game_{2,R}$ in which $\mathcal{A}$ outputs s_A when $s = s_A$, there is an execution in which $\mathcal{A}$ outputs s_A when $s = s_B$ that occurs with equal probability.

Let us examine an execution in which $\mathcal{A}$ outputs s_A when $s = s_A$. Now we will examine another execution in which:

- The true secret is s_B, not s_A.
- The initial share sent to $row^{(1)}_{R_1}$ by the dealer was incremented by $s_B - s_A$.
- For all nodes not on path $row^{(1)}_{R_1}, \ldots, row^{(T)}_{R_T}$, the messages received, data held, and messages sent are the same as the original execution. (This means that the data seen by $\mathcal{A}$ and its behavior are identical in the two executions.)
- All secret-shares held by parties in path R are incremented by $s_B - s_A$.
- All shares of secret-shares sent from parties in R to other parties in R are incremented by $s_B - s_A$.

All parties except for the dealer and those in path R view the same thing in both executions and make the same choices, so the probability of them doing so is the same in both executions. This includes $\mathcal{A}$. It remains to show that this is a valid execution for honest parties on the path. The sum of the messages sent by the dealer is equal to the true secret, so this is a valid execution by the dealer. For each party in R, all of the messages they receive from parties in R is incremented by $s_B - s_A$, so, even if these messages disagree, the message they choose as the "correct" message will also be incremented by $s_B - s_A$. Thus the secret share they compute will be incremented by $s_B - s_A$ as required. Lastly, all output messages are the same except those sent to parties in R, and shares-of-shares sent to R are incremented by $s_B - s_A$, so the sum of shares-of-shares output will still equal the share held by the parties. Thus this is a valid execution by honest parties. Since each valid execution by honest parties is equally likely, the probability that this execution occurs is just as likely as the original. Finally, the random choices of all parties on the path R are made independently of all parties not on the path, and in particular of $\mathcal{A}$, so the combined probability of the modified execution occurring is the same as that of the original.

Theorem 6. *If a full honest row path exists, then $\mathcal{A}$ has negligible advantage in guessing the secret.*

Proof. If such a path, R, exists, then there is some game $Game_{1,R}$ which, by Lemma 3 is indistinguishable from $Game_{Real}$ to $\mathcal{A}$. By Lemma 4 this, in turn, is indistinguishable from $Game_{2,R}$ to $\mathcal{A}$. Now, $\mathcal{A}$'s behavior in $Game_{2,R}$ has the same probability if the secret is modified. Thus, $Game_{2,R}$ is (perfectly) indistinguishable to $\mathcal{A}$ from an execution of $Game_{2,R}$ with a modified secret. This in turn is indistinguishable from an execution of $Game_{1,R}$ with a modified secret, which in turn is indistinguishable from an execution of $Game_{Real}$ with a modified secret. Since the indistinguishable relation is transitive, this means that any real execution is indistinguishable to $\mathcal{A}$ from another real execution with a modified secret. Thus, $\mathcal{A}$ has negligible advantage in guessing the secret.

Finally, the proofs about the existence of honest paths for Protocol 1 apply immediately to the case of honest row paths. In particular, as has already been proven in Corollary 1, if a fixed portion δ of the rows are honest, and $\delta \leq \gamma(\alpha - 1)$, (where constants γ and α depend on G) then a full honest row-path exists. Since a dishonest row requires at least one dishonest party, and there are n^a rows, we get the following result:

Corollary 2. *If there are at most δn^a malicious nodes, then there exists a full honest row path.*

Theorem 7. *There exists some constant δ, such that the protocol is computationally-private against a malicious proactive adversary corruption at most δn^a nodes per epoch.*

Now we show that the protocol also has robustness. The approach is very similar to that of Protocol 2, though in this case we show that a constant proportion of *columns* in the grid are holding and sending correct values.

Again, before proceeding we need to introduce some terminology. A *column* is a set of nodes in a given time-step that are in the same column in the grid, i.e. column $col_j^{(t)} = \cup_{i=1}^{n^a} P_{i,j}^{(t)}$. If the adversary corrupts any party in a column (in a given time step), then the column is *corrupt*. Otherwise a column is *honest*. Note that, except for the dealer, all (honest) parties' actions are deterministic. Therefore, given a certain setup by the dealer, we can consider the *correct* value for a data element held, or for a message sent, to be the value that would be sent if all parties follow the protocol. A column is *correct* if all of the data held and messages sent by all parties in the column are correct, and *incorrect* otherwise. Column $col_j^{(t)}$ is a *before-neighbor* of column $col_{j'}^{(t+1)}$ exactly if there exists i, i' such that $P_{i,j}^{(t)}$ is meant to send a message to $P_{i',j'}^{(t+1)}$. This occurs exactly when $(j, j') \in H$.

Lemma 6. *If the majority of an honest column's before-neighbors are correct, then the column will also be correct.*

Proof. Let $col_{j'}^{(t+1)}$ be a column with a majority of correct before-neighbors. Then, for every node $P_{i',j'}^{(t+1)} \in col_{j'}^{(t+1)}$, for every $i \in GL_{i'}$, the majority of

messages $s^{(t)}_{i,i',j,j'}$ it receives are correct. Thus it will compute the correct value for $\hat{s}^{(t)}_{i,i',j'}$ for all $i \in GL_{i'}$ and thus it will also compute the correct value for $s^{(t+1)}_{i',j'}$. Likewise, for every $i \in GL_{i'}$, the majority of messages $\hat{k}^{(t)}_{i,i',j,j'}$ that it receives will be correct, so it will compute the correct value for $\hat{k}^{(t)}_{i,i',j}$ for every $i \in GL_{i'}$ and thus compute the correct value for $k^{(t+1)}_{i',j'}$. Thus all data held by $P^{(t+1)}_{i',j'}$ is correct. Since $s^{(t+1)}_{i',j'}$ and $k^{(t+1)}_{i',j'}$ are both correct, the messages that $P^{(t+1)}_{i',j'}$ sends in the next resharing step will also be correct. Since this is true for all $P^{(t+1)}_{i',j'} \in col^{(t+1)}_{j'}$, then column $col^{(t+1)}_{j'}$ is itself correct.

Theorem 8. *If $\mathcal{A}$ corrupts δn^{1-a} nodes in each epoch, for some constant $\delta < \frac{1}{2}$, then for any constant ϵ satisfying $\delta < \epsilon < \frac{1}{2}$ there exists some constant d such that if H is a d-regular Ramanujan bipartite expander, then at most ϵn^{1-a} columns in every epoch are not correct.*

Proof. By induction. For the first epoch, there are at most δn^{1-a} corrupt columns. The remaining nodes are correct, since they received messages only from the dealer, who is honest. Therefore, the total number of incorrect columns is $\delta n^{1-a} < \epsilon n^{1-a}$.

Now, assume at most ϵn^{1-a} columns are incorrect in epoch t. By Lemma 8, the definition of a Ramanujan d-regular expander and Lemma 6, this means that the number of honest columns in epoch $t+1$ that are incorrect is at most:

$$\frac{4\epsilon n^{1-a}}{d\left(\frac{1}{2}-\epsilon\right)^2}$$

A further δn^{1-a} columns may be corrupt. Therefore, the total number of incorrect columns in epoch $t+1$ is at most

$$\frac{4\epsilon n^{1-a}}{d\left(\frac{1}{2}-\epsilon\right)^2} + \delta n^{1-a} = \left(\delta + \frac{4\epsilon}{d\left(\frac{1}{2}-\epsilon\right)^2}\right) n^{1-a}$$

If $d \geq \frac{4\epsilon}{\left(\frac{1}{2}-\epsilon\right)^2(\epsilon-\delta)}$ then this is at most ϵn^{1-a}.

Setting a concrete value of ϵ leads immediately to the robust security guarantee for Protocol 3.

Theorem 9. *Protocol 3 provides robustness against a proactive adversary corruption $\Theta(n^{1-a})$ nodes in each epoch.*

8 Securing Channels Using Signing Oracles

Our solution for establishing secure channels requires a simple piece of trusted hardware, a secure signing oracle. The secure signing oracle has a (persistent) public verification key, and can be used to sign arbitrary messages. The only

trust assumption is that the private key cannot be extracted from the device. In addition to the signing oracle, we assume that each party has a trusted random number generator, i.e., every party that is not corrupted in the current epoch can generate random numbers that are unpredictable to the adversary.

Such devices are commonly available as external devices (e.g. Yubikeys, or cryptocurrency wallets like the Ledger or Trezor), and are implemented by Apple's Secure Enclave on the iOS.[5]. Suppose, in addition, that the verification keys corresponding to these signing oracles are baked into the read-only memory of every other party.

When a party is corrupted by the adversary, we assume that the adversary has unfettered access to the signing oracle, and can sign arbitrary messages of their choosing.

Secure signing oracles do not immediately yield persistent secure channels on their own, since (1) they do not provide *private* channels, and (2) since the adversary (with access to a signing oracle), can always sign additional messages and inject them into the channels at a later date.

In our solution, we use the persistent key in the signing oracle to bootstrap new keys for each epoch of the protocol. It is not sufficient to simply use the signing oracle to sign new epoch-specific keys, because if an adversary corrupts a party at time t (and gains access to the signing oracle), the adversary can sign new key material, and hold onto these signed keys until after a reboot.

We can eliminate this attack vector with a simple challenge-response protocol. In epoch t, party i will reboot, and generate a new key, $pk_{i,t}, sk_{i,t}$. At the beginning of epoch t, party j will send a challenge $r_{i,j,t}$, to party i. Party i will then sign the pair $(pk_{i,t}, r_{i,j,t})$, using their signing oracle, and then return the signed key to party j. This allows party j to ensure that the new key $pk_{i,t}$ was generated by party i in epoch t (or later).

The formal protocol is described in Protocol 4.

Establishing Secure Channels

Party i holds a secure signing oracle $\mathsf{SO}_i(\cdot)$, and verification keys $\{\mathsf{VK}_j\}_{j\in[N]}$.

- **Challenge:**
  - **Peer-to-peer messaging:** Player i generates a random challenge $r_{i,j,t} \leftarrow_r \{0,1\}^\kappa$ for each party, j, with whom they plan to communicate in epoch t.
  - **Randomness beacon:** In the presence of a trusted randomness beacon that generates a random nonce, $r_t \leftarrow_r \{0,1\}^\kappa$ every epoch, can use this to avoid communication. Instead, every players sets $r_{i,j,t} = r_t$, and players do *not* need to exchange challenges. Thus the

[5] `https://support.apple.com/guide/security/secure-enclave-sec59b0b31ff/web`

presence of a trusted randomness beacon can reduce the communication and round complexity. The rest of protocol proceeds in the same way, whether the challenges $r_{i,j,t}$ were generated and exchanged by the players or provided by the randomness beacon.

- **Key generation:** Player i uses its random number generator to generate a key pair, $pk_{i,t}, sk_{i,t}$, for an *authenticated* encryption scheme.
- **Signing:** Player i uses their trusted signing oracle, SO_i to produce the signature $\sigma_{i,j,t} = \mathsf{SO}_i(i||pk_{i,t}||r_{i,j,t})$
- **Communication:** Player i sends $pk_{i,t}, \sigma_{i,j,t}$ to every player j that the wish to communicate with in epoch t.
- **Verification:** Player j checks that the signature $\sigma_{i,j,t}$ is a valid signature on the message $i||pk_{i,t}||r_{i,j,t}$ using the (persistent) verification key VK_i.

Protocol 4.

Theorem 10. *Protocol 4 is a secure method for establishing channel keys in the mobile adversary model. Specifically, consider a PPT mobile adversary who is allowed to corrupt a (possibly) different subset of parties at every epoch. Then if j is an uncorrupted party in epoch t, and j accepts $pk_{i,t}$, then (with all but negligible probability) $pk_{i,t}$ was generated by party i in epoch t.*

Proof. If party j is honest, and accepts a public key, $pk_{i,t}$ from party i in epoch t, then the signature $\sigma_{i,j,t}$ is valid signature of $pk_{i,t}||r_{i,j,t}$ under party i's persistent verification key, VK_i.

If $pk_{i,t}$, was *not* generated in epoch t, then either (i) the signature $\sigma_{i,j,t}$ was generated by an adversary *with* access to the signing oracle in a prior epoch, or (ii) the signature $\sigma_{i,j,t}$ was generated by an adversary *without* access to the signing oracle in the current epoch.

For case (i), an adversary with access to the signing oracle, would have to guess the challenge $r_{i,j,t}$ (which was generated uniformly at random from $\{0,1\}^\kappa$ in epoch t. Any polynomial-time adversary can make at most a polynomial number of queries to the signing oracle (and store the resulting signatures until epoch t), and thus has at most a negligible probability of querying the oracle with the challenge $r_{i,j,t}$.

For case (ii), an adversary who never had access to the signing oracle would have to guess the signature $\sigma_{i,j,t}$. A polynomial-time adversary can guess at most a polynomial number of signatures $\sigma'_{i,j,t}$ and check (using the public verification key VK_i) whether the signature is valid on $pk'||r_{i,j,t}$. Since the adversary can only make a polynomial number of guesses, the adversary's success probability in this scenario is also negligible.

Thus an adversary (who has not corrupted party i in epoch t) has only a negligible probability of getting party j to accept a public key.

9 Acknowledgements

This research was sponsored in part by ONR grant (N00014-15-1-2750) "SynCrypt: Automated Synthesis of Cryptographic Constructions" and a gift from Ripple Labs, Inc.

References

ABPP14. Michel Abdalla, Fabrice Benhamouda, Alain Passelègue, and Kenneth G Paterson. Related-key security for pseudorandom functions beyond the linear barrier. In *CRYPTO*, pages 77–94. Springer, 2014.

BC10. Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In *CRYPTO*, pages 666–684. Springer, 2010.

BDLO15. Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In *ACNS*, pages 23–41. Springer, 2015.

BEDLO14. Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. How to withstand mobile virus attacks, revisited. In *PODC*, pages 293–302, 2014.

BHNS99. Boaz Barak, Amir Herzberg, Dalit Naor, and Eldad Shai. The proactive security toolkit and applications. In *CCS*, pages 18–27, 1999.

CH94. Ran Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In *CRYPTO*, pages 425–438. Springer, 1994.

CHH97. Ran Canetti, Shai Halevi, and Amir Herzberg. Maintaining authenticated communication in the presence of break-ins. In *PODC*, pages 15–24, 1997.

CHK03. Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271. Springer, 2003.

CKLS02. Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *CCS*, pages 88–97, 2002.

Coh16. Michael B Cohen. Ramanujan graphs in polynomial time. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–281. IEEE, 2016.

ELL20. Karim Eldefrawy, Tancrede Lepoint, and Antonin Leroux. Communication-efficient proactive secret sharing for dynamic groups with dishonest majorities. In *ACNS*, pages 3–23. Springer, 2020.

Fel87. Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–438. IEEE, 1987.

FGMY97. Yair Frankel, Peter Gemmell, Philip D MacKenzie, and Moti Yung. Optimal-resilience proactive public-key cryptosystems. In *FOCS*, pages 384–393. IEEE, 1997.

GRR98. Rosario Gennaro, Michael O Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

Hae95. Willem H Haemers. Interlacing eigenvalues and graphs. *Linear Algebra and its applications*, 226:593–616, 1995.

HJKY95. Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO*, pages 339–352. Springer, 1995.

HO18. Brett Hemenway and Rafail Ostrovsky. Efficient robust secret sharing from expander graphs. *Cryptography and Communications*, 10(1):79–99, 2018.

MAA+16. Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. Intel® software guard extensions (Intel® SGX) support for dynamic memory management inside an enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, pages 1–9. 2016.

MSS13. Adam Marcus, Daniel A Spielman, and Nikhil Srivastava. Interlacing families I: Bipartite Ramanujan graphs of all degrees. In *FOCS*, pages 529–537. IEEE, 2013.

MSS18. Adam W Marcus, Daniel A Spielman, and Nikhil Srivastava. Interlacing families IV: Bipartite Ramanujan graphs of all sizes. *SIAM Journal on Computing*, 47(6):2488–2509, 2018.

MZW+19. Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. CHURP: dynamic-committee proactive secret sharing. In *CCS*, pages 2369–2386, 2019.

Nil91. Alon Nilli. On the second eigenvalue of a graph. *Discrete Mathematics*, 91(2):207–210, 1991.

OY91. Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *PODC*, pages 51–59, 1991.

Par21. Pedro Paredes. *On the Expansion of Graphs*. PhD thesis, Princeton, 2021.

Rab98. Tal Rabin. A simplified approach to threshold and proactive RSA. In *CRYPTO*, pages 89–104. Springer, 1998.

Sha79. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

SLL10. David Schultz, Barbara Liskov, and Moses Liskov. MPSS: mobile proactive secret sharing. *TISSEC*, 13(4):1–32, 2010.

Vad12. Salil Vadhan. *Pseudorandomness*, volume 7. Now Delft, 2012.

YXD22. Yunzhou Yan, Yu Xia, and Srinivas Devadas. Shanrang: Fully asynchronous proactive secret sharing with dynamic committees. IACR ePrint 2022/164, 2022.

ZSVR05. Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. APSS: Proactive secret sharing in asynchronous systems. *TISSEC*, 8(3):259–286, 2005.

Supplemental Material

A Previous Work

The Mobile adversary model is particularly challenging, because eventually a mobile adversary will have corrupted *all* the parties (but not all simultaneously). This means that an adversary who corrupts a party should (1) not be able to read the party's historical state, and (2) should not be able to predict the party's randomness in the future.

This means that at minimum parties need *secure deletes*, since otherwise an adversary who corrupted a party at time t, could read all of the messages received by the party during all previous rounds of the protocol, as well as *fresh randomness*, so that an adversary cannot predict the behavior of parties it has corrupted in the past.

In the original work introducing the mobile adversary [OY91], they imagined removing and adversary (and securely deleting previous state) by imagining a "clean" version of the program sitting in read-only memory, a piece of trusted hardware that would periodically "reboot" the machine to remove the adversary (as well as the history). They also assumed that either "each coin-flip is generated online (which is the practical assumption on generating randomness from physical devices), or, more abstractly, that the entire random tape of the machine is replaced with a new one during reboot."

Our works, like essentially all prior works in the PSS literature assume parties can securely delete state variables, and can be securely "rebooted" to obtain a clean copy of the PSS program.

A.1 PSS Protocols

The *mobile-adversary* model was introduced in [OY91], where they provided an information-theoretic protocol for secure computation. Proactive secret sharing has been widely studied e.g. [OY91,HJKY95,FGMY97,Rab98,CKLS02,ZSVR05] [BHNS99,SLL10,BEDLO14,MZW+19,YXD22] and some works (e.g. [FGMY97,Rab98]) focused on proactive secret sharing of keys for specific cryptosystems (e.g. RSA).

The main challenge in developing a PSS protocol is how to refresh the shares. In the semi-honest model, the linearity of secret sharing schemes like Shamir's scheme [Sha79] make it straightforward to re-randomize shares when parties are *semi-honest*. One method is to have each party generate a fresh sharing of zero, then each party locally adds all the shares they received. So the secret, s, can be shared according to a polynomial $f(x)$, where party i holds $f(i)$, and $f(0) = s$. In this method of refreshing, party i generates a polynomial $g_i(x)$, such that $g_i(0) = 0$, and gives $g_i(j)$ to party j, and party j calculates their new share as $f(j) + \sum_i g_i(j)$. This is the refresh method laid out in [HJKY95].

Another method is to have party i *re-share* their share, i.e., party i generates a new polynomial $g_i(x)$ such that $g_i(0) = f(i)$, and gives $g_i(j)$ to party j. This

re-sharing technique is widely used in Secure Multiparty Computation [GRR98]. Since polynomial interpolation is a linear operation, party j can compute a new sharing of the original secret, s, by doing local, linear operations on the shares $\{g_j\}_i$. This is the refresh method laid out in [CKLS02].

Other works [ELL20,MZW+19,YXD22] share using bivariate polynomials. To obtain security against malicious adversaries (instead of semi-honest adversaries), these simple refresh protocols were combined with Verifiable Secret Sharing (like Feldman VSS [Fel87]), as well as BFT consensus.

The mobile adversary model relies on "epochs" – the adversary is static *within* an epoch – and this introduces some amount of synchrony into the model. It is possible to consider an *asynchronous* model of PSS, where there is still a global notion of epochs, but communication *within* an epoch is asynchronous (and adversarially controlled). PSS protocols that can tolerate asynchronous communication within an epoch include [CKLS02,ZSVR05,SLL10,YXD22]. Some PSS schemes have been implemented [SLL10,MZW+19,YXD22].

As is evident from the brief description of prior works, they all require for each party to carry out a secret sharing in the refresh phase, even in the semi-honest model. This results in an all-to-all communication between the parties during the share refresh. In our work, parties do not have all-to-all communication every epoch, instead they communicate according to an *expander graph*. Expander graphs have been used to build Robust Secret Sharing schemes [HO18], but those constructions only consider a static adversary.

To reduce communication, some protocols can handle batches of independent secrets, which can reduce *amortized* communication complexity. Batched PSS protocols include [BEDLO14,BDLO15,ELL20].

A.2 Refreshing Secure Channels

Most secure multiparty protocols assume that parties can communicate using "secure, authenticated channels." In practice, however, these secure channels are usually secured using public-key encryption, and authenticated using digital signatures. This works well in the *static* adversary model.

In the mobile adversary model, parties cannot use *persistent* keys to secure and authenticate their channels, because once an adversary corrupted a party (and in doing so learned their private keys), the adversary could read *all* messages sent to that party during future rounds of the protocol (using the party's decryption key) and impersonate the party in all future rounds of the protocol (using the party's signing key).

This problem is *not* readily solved. If a party is securely rebooted (and generates new key material), how can they communicate their new public encryption and verification keys to the other parties? They cannot simply sign their new key using their old key, since an adversary (who had corrupted the party in the previous round) could generate a competing key, and sign it using the party's old, valid key.

One way to side-step this problem is to assume that parties are connected via persistent, secure authenticated channels (e.g. secure hardware channels),

thus eliminating the need for key management. This is the approach taken in [OY91] as well as many subsequent works including [CKLS02,BEDLO14], [MZW+19,YXD22].

In [HJKY95] they addressed this problem by assuming that all parties had access to an *uncensorable* broadcast channel. When a party was rebooted, they would generate new key material, and sign the new key using their old key, and broadcast their new (signed) key. As noted above, the adversary could do the same, by generating a new (adversarially controlled) key, and signing this key with the old key. In this case, however, since the broadcast channel is uncensorable, honest parties would see *two* new keys broadcast after the reboot. They would not be able to distinguish which one was valid, but they could refuse to use *either* key until the offending party was rebooted again. This provides a method whereby an adversary could halt the network (by continually broadcasting false keys after a reboot), but could never violate security.

This problem was explored in depth in [CHH97], where they propose a solution involving proactive, threshold signature schemes. Essentially, the construction of [CHH97] works as follows: At the start of the protocol, it is assumed that all parties hold a *share* of private signing key, and the corresponding verification key is baked into their read-only memory. This persistent verification key will then be used to authenticate all short term secrets as follows. When a party reboots, and generates new key material, they will send their new public keys to all parties, at which point the parties will run a byzantine agreement protocol to agree on the party's public key. Then they will use their long-term key shares to generate a threshold signature on the party's new signing key. Unfortunately, this construction rests on a proactive threshold signature scheme, to avoid circularity, they show how to convert any proactive threshold signature scheme (that requires authenticated channels) to one that does not require authenticated channels, using byzantine agreement.

Some PSS protocols (e.g. [SLL10]) consider a *dynamic* committee model, where there is a completely new committee in each epoch (and the public keys of all the new committee members are known in advance), so there is no need to refresh channel keys. This model does allow members from *old* committees to be corrupted (even after their role on a committee is done), so parties use a *forward-secure* cryptosystem [CHK03]. This means that an adversary who corrupts a party cannot decrypt ciphertexts sent to that party in *previous* epochs. Unfortunately, forward-secure cryptosystems do *not* prevent the adversary from learning messages sent in *future* epochs.

[ZSVR05] suggests a few possible approaches for creating persistent secure channels between parties. One approach is to use trusted hardware to implement a signing oracle with a monotonically increasing counter. Every time the oracle signs a message, it would include the counter (that is incremented every epoch), that ensures that the message was sent during the current epoch. They also suggest an alternative approach with a trusted administrator (with a static public key), who can identify each party and sign their new keys after each refresh.

They do not, however, describe how a party can authenticate themself to the trusted administrator after a reboot.

[CKLS02] suggests that if each party has a trusted co-processor (e.g. Intel SGX [MAA+16]), then the co-processor can have a trusted clock (that is timed to the epochs), as well as a persistent signing key. Then the co-processor can generate new session keys every epoch, and sign these new epoch-keys together with the epoch number (from its trusted clock), using its persistent signing key. Now that these trusted co-processors are prevalent in commodity hardware, this is a promising approach. Below (Section 8) we show how to eliminate the need for a full-blown trusted co-processor with a tamper-proof clock.

The assumption that persistent, trusted channels exist (e.g. [OY91,CKLS02], [BEDLO14,MZW+19,YXD22]) is an extremely strong assumption, which we would like to avoid. Weaker assumptions, assuming a censorship resistant broadcast channel as in [HJKY95], or byzantine agreement and threshold secret sharing (as in [CHH97]) are unsatisfactory in our setting because they require *all-to-all* $O(n)$ communication per-party, something that we wish to avoid in our protocol.

In Section 8, we outline a novel solution for re-establishing secure, authenticated channels in the presence of a mobile adversary. Our solution is compatible with any other proactive secret sharing scheme that requires secure channels and has the added benefit that it is compatible with essentially any communication pattern, i.e., it only requires communication between the sender and receiver in order to set up a secure channel between the two parties.

B Ramanujan Expanders

Ramanujan expanders are expanders with essentially optimal spectral expansion. The spectral expansion of a graph is the largest absolute value of an eigenvalue of the adjacency matrix (apart from the trivial eigenvalues $\pm d$). Ramanujan graphs have spectral expansion at most $2\sqrt{d-1}$. This is optimal in the sense that for any $\epsilon > 0$, any infinite family of d-regular graphs contains at least some graphs with spectral expansion greater than $(2\sqrt{d-1} - \epsilon)$ [Nil91].

Definition 3. *A d-regular graph, G, is called a* Ramanujan Graph *if the spectral radius of G is bounded by $2\sqrt{d-1}$, i.e., for every eigenvalue λ of the adjacency matrix of G, if $|\lambda| < d$, then $|\lambda| < 2\sqrt{d-1}$.*

In particular, we use balanced bipartite Ramanujan expanders. Balanced bipartite Ramanujan graphs can be efficiently computed for all degrees and sizes [MSS13] [MSS18] [Coh16].

Ramanujan graphs do not necessarily have *optimal* vertex expansion. It is an open problem to find explicit general constructions of bipartite graphs with near-optimal vertex expansion (see Open Question 6 of Paredes [Par21]). Constructing such graphs would improve the concrete results of this paper. Nevertheless, Ramanujan graphs provide good vertex expansion, which is sufficient for the purposes of this paper.

Below we demonstrate that Ramanujan graphs have the properties our protocols require. Concretely, we prove Theorems 1 and 2. We start with a standard theorem relating spectral and vertex expansion:

Theorem 11 (Spectral expansion implies vertex expansion [Vad12][Theorem 4.6]). *If G is a d-regular graph with second largest eigenvalue λ, then for every $\gamma \in [0,1]$, G is a (γ, α) expander where*

$$\alpha = \frac{1}{(1-\gamma)\frac{\lambda^2}{d^2} + \gamma} \tag{3}$$

Combining Definition 3 and Theorem 11 gives our first required property:

Theorem 1. *A Ramanujan graph is a $\left(\gamma, \frac{1}{(1-\gamma)\frac{4}{d}+\gamma}\right)$ expander $\forall\ \gamma \in [0,1]$.*

We now prove the second property. We are given a d-regular, bipartite expander graph with two sets L and R each of size n. We have a subset $S \subset L$ of nodes of size δn on the left and a value ϵ_1. We want to calculate how many nodes on the right have more that ϵ_1 fraction of their edges connected to the set S.

Lemma 7 (Bipartite Expander Mixing [Hae95][Theorem 5.1]). *Let G be a d-regular bipartite graph with spectral radius λ. Suppose, $S \subset L$, and $T \subset R$, with $|S| = \alpha |L|$, and $|T| = \beta |R|$. Let $e(X,Y) \stackrel{\text{def}}{=} |\{(x,y) \in E \mid x \in X, y \in Y\}|$ then*

$$\left| \frac{e(S,T)}{e(L,R)} - \alpha\beta \right| \leq \frac{\lambda}{d}\sqrt{\alpha\beta} \tag{4}$$

Note, that $e(L,R)$ are all the edges in the graph, i.e. $d|L|$.

Lemma 8. *Given a d-regular bipartite expander with spectral radius λ, suppose a set of δn vertices on the left are in S then at most*

$$\frac{\lambda^2 \delta n}{(\epsilon_1 - \delta)^2 d^2} \tag{5}$$

right vertices have at least an ϵ_1 fraction of left-neighbors in S.

Proof. Let T denote the set of right-hand vertices that have at least an ϵ_1-fraction of left-neighbors in S. Since G has right-degree d, we have $e(S,T) \geq d\epsilon_1 |T|$.

On the other hand, the expander mixing lemma (Lemma 7) tells us that for $\alpha = |T|/n$ and $\beta = \delta$,

$$\left| \frac{e(S,T)}{nd} - \delta\frac{|T|}{n} \right| \leq \frac{\lambda}{d}\sqrt{\delta\frac{|T|}{n}} \quad \Rightarrow \quad \frac{e(S,T)}{d} - \delta|T| \leq \frac{\lambda}{d}\sqrt{n\delta|T|} \tag{6}$$

On the other hand, $e(S,T) \geq d\epsilon_1 |T|$, so we have

$$\epsilon_1 |T| - \delta|T| \leq \frac{\lambda}{d}\sqrt{n\delta|T|} \quad \Rightarrow \quad |T| \leq \frac{\lambda^2 \delta n}{(\epsilon_1 - \delta)^2 d^2} \tag{7}$$

Since Ramanujan graphs have spectral radius at most $2\sqrt{d-1}$, this implies our required property:

Theorem 2. *Ramanujan graphs have the following property. Let S be a set of size at most δn vertices on the left. Then at most*

$$\frac{4\delta n}{(\frac{1}{2}-\delta)^2 d} \tag{2}$$

right-hand vertices have at least $\frac{1}{2}$ of their neighbors in S.

C Model

C.1 Secrets and Shares

In our PSS protocols, we assume that there is a single secret, denoted s, that is (honestly) distributed by a trusted dealer before the protocol begins, resulting in each party holding a share. Our private and robust protocol (Protocol 3) also requires the dealer to distribute initial PRG keys. When we describe the life-cycle of an epoch in this section, such keys are also considered part of a party's share.

C.2 Mobile Adversaries

The set of parties in the protocol is denoted $\{P\}_{i=1}^{n}$, and communication between parties is assumed to be *synchronous*. The protocol is divided into *epochs*, which are fixed time intervals. Each epoch is comprised of two phases, a refresh phase and a retain phase (described in Section C.4).

The adversary, $\mathcal{A}$, is assumed to be *mobile*, which means that the adversary is allowed to corrupt t (out of n) parties in each epoch. There is, however, no limit on the *cumulative* number of parties that the adversary has ever corrupted (across multiple epochs). When an adversary corrupts a party, it is allowed to see all messages that are sent and received by that party. If the adversary is malicious, the adversary can also cause the party to deviate from the protocol (e.g. by sending fraudulent messages). Furthermore, the adversary is *rushing* which means it can wait to receive all incoming messages before sending any messages.

We make the following assumptions that are standard for mobile adversaries. We assume parties can securely delete data. We assume parties have access to fresh randomness. We instantiate secure, authenticated channels between parties using a (hardware-based) signing oracle. See Section 8 for more details about our proposed method for instantiating secure channels. Alternatively, if we simply assume the existence of secure channels, we can drop the assumption that parties can generate fresh randomness.

C.3 Reboots

To handle such an adversary, we assume that it is possible to remove the adversary's control of a party by a *reboot* operation. Rebooting a party will cause the adversary to lose all access to new information (e.g. fresh randomness generated, messages received) and will cause the party to return to executing the correct program. (For instance, the program may be stored in some ROM and reloaded following a reboot.) Although rebooting refreshes the party's program, it does not delete certain state information, specifically the party's share. This means that if a party's share is corrupted (incorrect) before the reboot, it will be incorrect after the reboot.

A party is *corrupted* if it has been corrupted, but not (yet) been rebooted. It is *honest* otherwise. By periodically applying reboots, we can limit the number of parties that are corrupted at any time.

C.4 Epochs

Epochs. Following the standard models of proactive secret sharing, we divide time into *epochs*. Each epoch consists of two phases, *refresh* and *retain*. The PSS protocol describes the *refresh* phase, while the *retain* phase encompasses anything the parties do with their share outside of the PSS protocol.

1. **Refresh:**
   (a) Reboot
   (b) Establish secure channels
   (c) Send messages
   (d) Securely delete old share (everything except current private key)
   (e) Receive messages
   (f) Securely delete keys (everything except new share)
2. **Retain:** Parties may use their share, e.g. in the context of an MPC protocol.

All of our protocols follow the structure above. The protocols will only differ in how a share is used to generate messages to send, and how messages received are combined to form a new share. Therefore, we will only describe the send message and receive message steps in our protocol; all other steps are implicit.

C.5 Counting Corruptions

A party which is corrupted during the retain portion of epoch t is considered corrupt, and counted against the budget of the adversary in epoch t.

It is important to note that an adversary who corrupts a party during the *refresh* phase of epoch t learns both the party's share from the epoch t, as well as the party's share from the previous epoch, i.e. $t-1$. In fact, an adversary who corrupts a party during the refresh phase of epoch t has slightly more power than an adversary who corrupts the same party in *both* the retain phases of epoch $t-1$ and epoch t because the adversary who corrupts during the refresh phase

can see (and potentially modify) the messages sent during the refresh phase in epoch t.

For this reason, as is standard [HJKY95], we consider that when an adversary corrupts a party during the *refresh* phase of epoch t, this counts towards the adversary's corruption budget of epoch t *and* epoch $t-1$.

Some PSS protocols consider *dynamic committees* (e.g. [ZSVR05,SLL10], [BDLO15,MZW+19]). In this dynamic-committee PSS the set of parties (and possibly the corruption threshold) can change from epoch to epoch. One of the key differences between a static-committee and dynamic committee PSS protocol is the way corruptions are counted. When the committee in epoch $t+1$ is different from the committee in epoch t, there is no need to double count parties who are committed during the refresh phrase. Thus it is typical, when considering dynamic committees, to give the adversary the power to corrupt up to k-out-of-n parties in the old committee as well as k-out-of-n of parties in the new commiteee. This means that in the dynamic committee model a security threshold of k-out-of-n is actually stronger than in the static committee model where the adversary can only corrupt $k/2$ parties (in the refresh phase) of every epoch. Our protocols work for either method of counting corruptions.

C.6 Security

Most PSS protocols simultaneously achieve both *privacy* and *robustness*. *Privacy* ensures that the secret is not learned by the adversary. *Robustness* ensures that the adversary cannot cause the reconstructed value to differ from the secret which was shared. In this work, we will sometimes consider these two properties separately. Note that in the case of *passive* adversaries, every protocol is robust since the adversary cannot introduce corruptions into the shares.

For both private and robust protocols, we will show protocols secure against *malicious* (active) adversaries who may cause parties that they corrupt to behave arbitrarily during the epoch in which they were corrupted. The number of parties that the adversary can corrupt in an epoch will be some function of the number of parties, n. For instance, $\mathcal{O}(n)$ security means that a security property holds if the adversary corrupts at most δn parties in an epoch for some constant δ.

We will be concerned with two types of security. Our protocols that just ensures privacy (Section 5) or robustness (Section 6) will provide *perfect security* which means that the security guarantees always hold, regardless of the computational resources of the adversary. Our protocol that ensures both privacy and robustness (Section 7) provides *computational security*, that is, an adversary with bounded computational power has an insignificant probability of undermining the security guarantees.

C.7 Reconstruction

In this work, we are focused on building PSS protocols with *sublinear* communication. In particular, this means that parties in our protocols typically do *not* communicate with every other party in every epoch. This makes it impossible

to guarantee that *every* honest party holds a valid share at every step of the protocol. To see this, consider an honest party in epoch t, an adversary (who can corrupt a constant fraction of parties in each epoch) could have corrupted *all* of the target party's communication partners in the previous epoch. In which case, the target party would not have communicated with a single honest party (since its last reboot) and could not be have an uncorrupted state.

Since this type of "eclipse attack" is unavoidable in our model, we consider a slightly different form of correctness in our constructions. We consider a PSS protocol secure if, in any given epoch, there exists a reconstruction protocol, which would allow the (honest) parties to reconstruct the original secret. The key distinction here is that the reconstruction procedure may require linear communication (e.g. all parties send their shares to every other party), but since the reconstruction procedure is not actually run in each epoch, the amortized communication per epoch can still be sub-linear.

D Epoch length

We present a maliciously-secure PSS protocol in Section 7 that can only tolerate $\Theta(\sqrt{n})$ corruptions per epoch, which may seem low compared to existing PSS protocols (e.g. [HJKY95] and [SLL10]) that can tolerate $\Theta(n)$ corruptions per epoch.

What this comparison hides is we are free to choose the length of an epoch by choosing how frequently we run the refresh protocol. Decreasing the length of an epoch will increase the communication cost (per unit time), but should decrease the number of parties an adversary can corrupt in a given epoch.

To see this in play, imagine that instead of allowing the adversary to corrupt $\delta \cdot n$ parties per epoch (as is standard in the PSS literature), we assumed the adversary had a fixed corruption *rate*, i.e., the adversary could corrupt one party every $t(n)$ units of time. A traditional PSS protocol (tolerating δn corruptions per epoch), would be secure in this model by setting the epoch length $T = \delta \cdot n \cdot t(n)$.

But now, consider the communication cost. A traditional PSS protocol, tolerating δn corruptions per epoch, and requiring $\Theta(n)$ communication per refresh, would have amortized communication cost of $\Theta\left(\frac{1}{t(n)}\right)$ per unit time. By contrast, our protocol, which requires only $\Theta(\kappa)$ communication per epoch, but can "only" tolerate $\Theta(\sqrt{n})$ corruptions could set a much lower epoch time, $T = \Theta(t(n) \cdot \sqrt{n})$, which would make the amortized communication cost of our protocol $\Theta\left(\frac{\kappa}{t(n)\sqrt{n}}\right)$ per unit time, which is much lower for sufficiently large n.

Furthermore, this ignores the costs of establishing secure channels in the (normal) case that secure hardware channels do not exist. Authentication between parties requires $\Omega(\kappa)$ communication (see Section 8 for our instantiation). This would increase the amortized communication cost of traditional protocols to $\Theta\left(\frac{\kappa}{t(n)}\right)$ per unit time, but the amortized cost of our maliciously-secure PSS protocol would remain $\Theta\left(\frac{\kappa}{t(n)\sqrt{n}}\right)$.

What this means is that (for sufficiently large n) we can achieve a *lower* amortized communication cost per unit time, while achieving the *same* level of security.

E Proof of Lemma 9

Lemma 9. *The equation* $f(x) = \frac{4x}{(\frac{1}{2}-x)^2(x-a)}$ *where* $0 < a < \frac{1}{2}$ *is minimized over the range* $a < x < \frac{1}{2}$ *by* $x = \frac{1}{4}(a + \sqrt{a^2+4a})$.

Proof. First, observe that over the range $a < x < \frac{1}{2}$, $f(x)$ is continuous, differentiable and positive. Therefore, any minimum point of $f(x)$ over $a < x < \frac{1}{2}$ is also a maximum point of $g(x) = \frac{4}{f(x)}$ over the same range. So we will now instead find the maximum point(s) of $g(x)$ over this range.

$$g(x) = \frac{(\frac{1}{2}-x)^2(x-a)}{x} = \frac{x^3 - ax^2 - x^2 + ax + \frac{1}{4}x - \frac{1}{4}a}{x} = x^2 - (a+1)x + (a+\frac{1}{4}) - \frac{1}{4}\frac{a}{x}$$

Now $g(a) = 0$, $g(\frac{1}{2}) = 0$ and $g(x)$ is positive over $a < x < \frac{1}{2}$, so $g(x)$ is not maximized over $a < x < \frac{1}{2}$ at the end-points. It must be maximum at a point, v, where the first derivative is 0.

$$g'(v) = 2v - (a+1) + \frac{a}{4v^2} = 0 \Rightarrow 2v^3 - (a+1)v^2 + \frac{a}{4} = 0 \Rightarrow (v - \frac{1}{2})(2v^2 - a - \frac{a}{2}) = 0$$

The solutions are $v = \frac{1}{2}$, and $v = \frac{a \pm \sqrt{a^2+4a}}{4}$. Only $v = \frac{a+\sqrt{a^2+4a}}{4}$ is in the range $a < x < \frac{1}{2}$, so this value minimizes $g(x)$ and maximizes $f(x)$ over this range.

F $\mathcal{O}(n)$-secure Proactive Pseudorandomness with $\mathcal{O}(\kappa)$ Communication

In Section 7, we required replicas to have local PRG keys which were generally indistinguishable from random to $\mathcal{A}$. Using pseudorandomness rather than fresh, local randomness was necessary to allow replicas to send identical messages.

A simplified version of Protocol 3 can instead be used for a different objective, replacing the need for fresh, local randomness altogether. In [CH94], Canetti and Herzberg presented the problem of generating *Proactive Pseudorandomness*. They argued that sometimes a source of fresh, local randomness is not available. They presented a protocol that replaces randomness by pseudorandomness generated by PRGs. In order to ensure that the pseudorandomness remains indistinguishable from random to a mobile adversary, each party, every epoch, sends every other party a randomizer. Each party combines the randomizers it receives to construct a new PRG seed. As long as the adversary is unaware

of any one of these randomizers, a party's new PRG key will be indistinguishable from random to $\mathcal{A}$. [CH94] argue that this removes the need for local, fresh randomness in Proactive protocols.

Like [CH94] we present a protocol that removes the need for fresh, local randomness in proactive protocols *provided secure hardware channels exist.* Unlike [CH94], each party communicates with only $\mathcal{O}(1)$ parties per epoch rather than all n parties. Since each party only communicates with a constant number of other parties in each epoch, honest parties are susceptible to "eclipse attacks," where the adversary corrupts all of the party's communication partners. Thus, we consider a slightly more relaxed notion of PP security than [CH94]. The original PP protocol guarantees that every party that is honest in a given epoch has pseudorandomness that is unpredictable to the adversary. Our protocol will instead guarantee that in every epoch, at least γn parties will have pseudorandomness that is unpredictable to the adversary, where γ is a constant.

The protocol is obtained by simplifying Protocol 3 as follows. We no longer need replication, so we set $a = 1$. We are concerned only with keys and key re-randomizers, so we remove all messages and variables related to shares. Additionally, since there are no replicas, we do not need to worry about related-key attacks. We therefore do not need a Φ_{add}-RKA secure PRF to combine the re-randomizers, simply adding the re-randomizers to generate a new key suffices. The resulting protocol is presented in Protocol 5.

Proactive Pseudorandomness

Parameters:
Let $G = (L \cup R, E)$ be a d-regular bipartite (γ, α) expander, with parts $L = \{L_1, \ldots, L_n\}$ and $R = \{R_1, \ldots, R_n\}$. We choose neighbors according to graph G. (See Section 5 for the definition of choosing neighbors.)

1. **Setup:**
   Each party, P_i, is provided an initial truly random seed k_i^1 from a trusted source.
2. **Re-randomizing:**
   (a) At the start of epoch t, each party P_i, for each of its neighbors P_j, generates re-randomizer $r_{i,j}^t = PRF_{k_i^{t-1}}(j)$ and sends it to P_j.
   (b) P_j receives a re-randomizer from each party of which it is a neighbor. It computes its new key as:
   $k_j^t = \bigoplus_{i \in N^{-1}(j)} r_{i,j}^t$.

Protocol 5.

To prove the security of this protocol, we first observe that the communication pattern in Protocol 5 is identical to that of Protocol 1. Thus as in the proof

of security of Protocol 1, we can define a layered graph H, where vertex $H_i^{(t)}$ represents P_i at epoch t. Recall that vertex $H_i^{(t)}$ is considered *honest* if P_i is honest in epoch t. Recall that the edges of H exactly represent the communication pattern of the protocol.

With these definitions, we can prove the following lemma.

Lemma 10. *If $H_i^{(t)}$ has an honest backward path to $H^{(1)}$, then k_i^t is indistinguishable from random to $\mathcal{A}$.*

Proof. Now, if there exists an honest forward path to $H_i^{(t)}$, there is a sequence of parties, $P_{f(1)}, \ldots, P_{f(t)}$, where $f(t) = i$, such that $P_{f(u)}$ is honest in epoch u, for all $1 \leq u \leq t$, and $P_{f(u+1)}$ is a neighbor of $P_{f(u)}$ for $1 \leq u \leq t-1$. We can now show that this implies that k_i^t is indistinguishable from random to the adversary.

By induction. $k_{f(1)}^1$ is given by a trusted dealer, so is truly random. $P_{f(1)}$ is honest in epoch 1, and $k_{f(1)}^1$ is deleted at the start of epoch 2, so the adversary could only learn $k_{f(1)}^1$ based on messages sent by $P_{f(1)}$ in epoch 2. However, all of the messages sent are outputs of a PRF, and by the security of the PRF, these leak no information about the key. Thus, $k_{f(1)}^1$ is indistinguishable from random to the adversary (not only in epoch 2, but at any point in the future).

Assume that $k_{f(u)}^u$ is indistinguishable from random to $\mathcal{A}$. This means that $r_{f(u),f(u+1)}^{u+1}$ is also indistinguishable from random when it is generated. (Recall $P_{f(u)}^{(u)}$ is honest.) It is sent to $P_{f(u+1)}$ at the start of epoch $u+1$, and then promptly deleted. Thus $\mathcal{A}$ cannot learn the value of $r_{f(u),f(u+1)}^{u+1}$ by later corrupting $P_{f(u)}$. $P_{f(u+1)}^{(u+1)}$ is honest, so $\mathcal{A}$ does not learn $r_{f(u),f(u+1)}^{u+1}$ from it. The message is sent over an encrypted, authenticated channel, so $\mathcal{A}$ cannot make any of the other re-randomizers sent to $P_{f(u+1)}^{(u+1)}$ be correlated with $r_{f(u),f(u+1)}^{u+1}$. Thus, $\bigoplus_{i \in N^{-1}(f(u+1))} r_{i,f(u+1)}^{u+1}$ is also indistinguishable from random to $\mathcal{A}$.

This now allows us to prove the security of our Proactive Pseudorandomness protocol:

Theorem 12. *Let there be a malicious mobile adversary that controls at most δn parties per epoch, where $\delta \leq \gamma(\alpha - 1)$. Protocol 5 ensures that each epoch contains γn parties whose keys are indistinguishable from random to the adversary.*

Proof. Lemma 5 in Section 5 that if $\delta \leq \gamma(\alpha - 1)$ then for every t, there exist at least γn nodes in $H^{(t)}$ that have honest backward-paths to $H^{(1)}$.

Combining Lemma 5 and Lemma 10 gives the desired security property.

Remark 3 (Malicious adversaries). Recall that since our first PSS protocol (Protocol 1) provided privacy but no robustness, an adversary, $\mathcal{A}$, could *change* the secret by sending an incorrect message, but it could not *learn* the secret.

In our PP protocol, a corrupted party can similarly send incorrect re-randomizers. This will change the PRF keys generated in later epochs, however it will not help

$\mathcal{A}$ *learn* these keys. This will mean that the state of the system is not simulatable based on its initial keys, but it does not undermine the pseudorandomness generated. Therefore, our PP protocol has security guarantees that are still useful in practice against malicious, mobile adversaries.

Remark 4 (Secure channels). As discussed in Section A.2, the secure channels required by PSS (and PP) protocols cannot be instantiated with static cryptographic keys, since a mobile adversary could learn the channel keys in one epoch, and then continue to read messages on the channel in future epochs.

In Section 8 we gave a simple solution for re-generating channel keys (assuming trusted signing oracle). Unfortunately, PP protocols are *not* sufficient to instantiate secure channels in the mobile adversary model.

To see this, note that $\mathcal{A}$ is able to see all (potentially encrypted) messages sent to a party P_i (even when P_i is not corrupted). Now, suppose P_i corrupted, and $\mathcal{A}$ learns the complete state of P_i, and then P_i is rebooted.

At this point, the protocol needs to do two things: (1) P_i needs to generate (and authenticate) new channel keys and (2) the other parties need to send P_i new re-randomizers. Unfortunately, we cannot do either first. If we try to generate a new random encryption public key before the re-randomizers are sent, $\mathcal{A}$ will continue be able to simulate P_i and can learn the corresponding secret key. If we try to send the re-randomizers to P_i, the adversary will be able to decrypt them and will continue to know the full state of P_i.

In short, if there are no secure hardware channels and no internal sources of fresh randomness, once $\mathcal{A}$ corrupts a party, $\mathcal{A}$ will always be able to continue simulating a party. This is true even if there are hardware channels that are authenticated but not private. As long as $\mathcal{A}$ is able to see all (potentially encrypted) messages sent to a party, once the party is corrupted, $\mathcal{A}$ will always be able to simulate the party's state. Note that this is not just an issue with our protocol: this inherent limitation applies to all proactive protocols. Thus, without secure (hardware) channels or fresh local randomness, it is impossible to attain any privacy in the mobile adversary model.

Furthermore, even if parties do have access to fresh local randomness, if there are no trusted hardware or authenticated hardware channels, once $\mathcal{A}$ corrupts a party, that party will never be able to authenticate itself. When $\mathcal{A}$ corrupts a party, it learns the entire state of the party at that point in time, and can therefore pretend to be the party in all future interactions. While the party may generate fresh local randomness, $\mathcal{A}$ can choose randomness from an identical distribution. Other parties will thus be unable to distinguish $\mathcal{A}$ from the real party. Therefore, without (hardware) authenticated channels or local secure hardware, it is impossible to authenticate parties in the mobile adversary model.