On Regenerating Codes and Proactive Secret Sharing: Relationships and Implications

Karim Eldefrawy[1], Nicholas Genise[2], Rutuja Kshirsagar[3], and Moti Yung[4]

[1] SRI International, USA,
karim.eldefrawy@sri.com,
[2] Duality Technologies**, USA,
ngenise@dualitytech.com,
[3] Virginia Tech, USA,
rutujak@vt.edu,
[4] Google and Columbia University, USA
motiyung@google.com

Abstract. We look at two basic coding theoretic and cryptographic mechanisms developed separately and investigate relationships between them and their implications. The first mechanism is Proactive Secret Sharing (PSS), which allows randomization and repair of shares using information from other shares. PSS enables constructing secure multi-party computation protocols that can withstand mobile dynamic attacks. This self-recovery and the redundancy of uncorrupted shares allows a system to overcome recurring faults throughout its lifetime, eventually finishing the computation (or continuing forever to maintain stored data). The second mechanism is Regenerating Codes (RC) which were extensively studied and adopted in distributed storage systems. RC are error correcting (or erasure handling) codes capable of recovering a block of a distributively held codeword from other servers' blocks. This self-healing nature enables more robustness of a code distributed over different machines. Given that the two mechanisms have a built-in self-healing (leading to stabilizing) and that both can be based on Reed Solomon Codes, it is natural to formally investigate deeper relationships between them. We prove that a PSS scheme can be converted into an RC scheme, and that under some conditions RC can be utilized to instantiate a PSS scheme. This allows us, in turn, to leverage recent results enabling more efficient polynomial interpolation (due to Guruswami and Wooters) to improve the efficiency of a PSS scheme. We also show that if parameters are not carefully calibrated, such interpolation techniques (allowing partial word leakage) may be used to attack a PSS scheme over time. Secondly, the above relationships give rise to extended (de)coding notions. Our first example is mapping the generalized capabilities of adversaries (called generalized adversary structures) from the PSS realm into the RC one. Based on this we define a new variant of RC we call Generalized-decoding Regenerating Code (GRC) where not all network servers have a uniform sub-codeword (motivated by non-uniform probability of attacking different servers case). We finally highlight several interesting research direc-

** This work was done entirely at SRI International.

tions due to our results, e.g., designing new improved GRC, and more adaptive RC re-coding techniques.

1 Introduction

Many times in the past, it was found that different areas and concepts in computing are related in some fashion. Uncovering such relations, in turn, led to information flowing and ideas transferring across areas. In this work we deal with two areas that are fundamental to major aspects of safe (and secured) self-healing distributed computing. The first area, *Coding Theory*, is one of the oldest and most fundamental branches of modern computing and communication, dealing with keeping integrity of data in various situations (originally communication channels, then storage media, and finally distributed fault tolerant storage). In the context of modern distributed computing (clusters, storage systems, cloud computing, etc.), a fundamental mechanism to perform fault-tolerant distributed storage is to encode the data to be stored using regenerating codes (RC) [1] (our first area's concentration). RC are a class of error correcting codes (ECC) invented in 2010, which, besides the traditional task of ensuring reliability of data recovery, also provide for efficient repair (regeneration) of failed nodes (holding blocks of the codeword) in a distributed storage system, where recovery is from information stored at other nodes, assuring better maintenance capability of the distributed codeword. We note that most RC work is in the erasure codes model, where servers can fail-stop, but some are in the error correcting mode where servers' memory can be maliciously modified.

The second area is *Secret Sharing Schemes*: given that data in distributed cryptographic systems may be sensitive (e.g., a redundant storage of cryptographic keys), these systems may be the target of byzantine/malicious corruption (not only random or fail-stop faults) which attempt to violate confidentiality of the stored data and its availability. To that end, (threshold) secret sharing (SS) was originally proposed as a fundamental cryptographic defense technique in the late 70s. It distributes a secret with redundancy into shares, where, in order to recover the secret, one needs a threshold of shares, while less than the threshold of shares reveals no information about the secret; the mechanism was employed heavily in security protocols. The security of the standard, and first, SS technique (known as Shamir's SS [2]) is based on polynomial interpolation, and is closely related to Reed-Solomon codes [3]. This relation was the first hint regarding certain connections between coding theoretic methods and sharing secrets procedures, yet this, early uncovered relationship did not continue to attract much attention since then.

In 1991, motivated by malicious adversaries being mobile as in the spread of malware, SS was extended to what became known as *Proactive Secret Sharing (PSS)* [4]. PSS (our second area's concentration), in fact, protects against a mobile adversary that can change the subset of corrupted nodes overtime and thus may eventually compromise all involved nodes over a long period of time (while the standard SS notion only assumes that a subset of the nodes can

be corrupted, even over a long period of time). PSS, in fact, adds to SS the ability to t-wise randomize the information held by the sharing servers, and allows for the recovery of the current state held by a server in case a share has been previously destroyed by the adversary which moved away. Since then, PSS was extended [5, 6] to more generic settings, beyond threshold adversaries, i.e., what is often called general adversary structures. PSS has also been employed in developing secure multi-party computations against mobile adversaries [6], and for building threshold cryptosystems, say, for supporting distributed certification authorities, and in recent years in various Blockchain based protocols, e.g., [7].

In this paper, renewing connections between coding theory and distributed secret sharing methods, we explore fundamental connections between RC and PSS. It is a natural question to ask, due to the fact that both notions involve reconstruction of information (shares or codewords) held by servers using information (shares or codewords) held at other servers , and that some schemes in both areas are, in fact, related to Reed Solomon (or other algebraic) codes. We also suggest (and demonstrate) utilization of the observed connections to imply new useful paradigms and extensions in one area (RC), building on related paradigms existing in the second area (PSS).

Our Contributions: This paper is the first systematic study of the close relation between *proactive* secret sharing (PSS) and regenerating codes (RC), and makes the following concrete contributions:

1. We show how security of common PSS schemes (treating/restricting leakage as/to full shares only leakage) fails to hold in simple generalized leakage models – models allowing leaks of smaller pieces of shares. This is accomplished by developing a new generalized model (Section 4) to reason about PSS and analyze its relation to RC. This new model takes into account partial leakage of information about shares of non-compromised nodes as opposed to complete information leaked (all or nothing) from compromised nodes.
2. We demonstrate a (conditional[5]) equivalence between PSS and RC. We provide two main theorems (Theorems 1 and 2 in Section 5) as a simple starting point demonstrating a conditional equivalence between PSS and RC. This allows for a flow of ideas and constructions between the two areas.
3. As a first demonstration of a flow of constructions and ideas from RC to PSS, we show that due to our result proving the equivalence between the two notions, recent techniques for efficient polynomial interpolation due to Guruswami and Wootters [8] may improve efficiency of several bottleneck sub-protocols in PSS. We also show that such efficient interpolation may cause a threat if parameters are not carefully calibrated, i,e., such interpolation technique may be used to attack PSS over a long period of time, (Corollary 1 in Section 5); further studying such attacks may be of independent interest.

[5] Our condition is that an RC code is MDS. This is to simplify this first treatment of the topic, we note that there is more work required to understand what conditions on the RC side imply certain types of security on the PSS side.

4. Conversely (i.e., considering ideas flowing in the other direction), we map (in Section 6) adversarial capabilities – called general adversary structures – from the PSS realm into the RC realm by defining a new notion of RC, called Generalized-decoding Regenerating Code (GRC). In a GRC, the decoding structure is captured by a collection of specific subsets of nodes that may decode, as opposed to the usual case where any set larger than a given threshold can decode. We also show how to construct a GRC scheme based on PSS and Theorem 1. We emphasize a general decoding structure is needed in many applications since network nodes often differ greatly in reliability, trust, and connections.

Paper Outline: Section 2 provides the necessary background and notation, whereas Section 3 overviews related work in PSS and RC; Section 4 describes a new generalized model to reason about PSS and which allows us to analyze its relation to RC; Section 5, in turn, contains the main technical results mapping RC to PSS and PSS to RC, while Section 6 discusses how relationships between the PSS and RC notions can further give rise to new extended notions in the RC realm. Our first example of such implications is mapping the generalizations of the capabilities of adversaries (called general adversary structures) from the PSS realm into the RC one, and define a new notion of RC which we call Generalized-decoding Regenerating Code (GRC). Finally, we conclude in Section 7 with a discussion of open questions and future research directions.

2 Background and Notation

In this section we define the underlying mathematical and protocol notions employed in this work. Due to space constraints, we have further background material in the Appendix.

Finite Fields. We denote finite fields as F, L always such that F is a finite extension of L ($L \leq F$). Let $GF(p)$ be the underlying prime field and let $t = [F : L]$ be $F's$ degree over L. The ring of polynomials with coefficients in F is denoted as $F[x]$ and the subset of polynomials with degree at most $k-1$ is denoted as $F[x]^{\leq k-1}$. The latter is a k-dimensional vector space over F. Whenever we need the fields' cardinalities, we say $F = GF(q^t)$ and $L = GF(q)$ (where $q = p^d$ such that d is some positive integer).

Definition 1. *Let $F = GF(q^t)$ be a field extension of $L = GF(q)$ with degree t. Then, the* field trace *is defined as*

$$tr_{F/L}(\alpha) = \alpha + \alpha^q + \alpha^{q^2} + \cdots + \alpha^{q^{t-1}}.$$

2.1 Reed-Solomon Codes

Definition 2. *An* (n, k) Reed-Solomon code *with distinct evaluation points* $A = \{\alpha_1, \ldots, \alpha_n\} \subset F$ *is the subspace of* F^n *defined as*

$$RS(A, k) := \{(f(\alpha_1), \ldots, f(\alpha_n)) | f \in F[x]^{\leq k-1}\}.$$

We call n the *block-length* and k the *dimension* of the code. Reed-Solomon codes are *Maximal Distance Separable* (MDS) codes since they achieve the Singleton bound. That is, their minimum distance is $n - k + 1$ and any collection of k code symbols, $f(\alpha_i)$, can be used to efficiently recover the original message, f. One can also efficiently decode Reed Solomon codes in the presence of $k < n/3$ errors using the Berlekamp-Welch algorithm [9].

2.2 Regenerating Codes

An (n, k, d, α, β) regenerating code [1] distributes a file, represented as a polynomial f in $F[x]^{\leq k-1}$, by encoding it and sending elements of the encoding to n nodes where each node stores α bits of data. A failed node can recover (repair) its share by accessing size β data from d surviving nodes, and we denote the *repair bandwidth* as $\gamma = d\beta$. Any k nodes are able to reconstruct the original file f when using their collective stored data. For example, an (n, k) Reed-Solomon code gives $\alpha = \log_2 |F|$ at each node and using the trivial reconstruction to repair a node yields $d = k$, $\beta = \log_2 |F|$, and $\gamma = k \log_2 |F|$. We give the formal definition in Definition 3 for completeness.

Definition 3. *Let* $\mathcal{P}_1, \ldots, \mathcal{P}_n$ *be labeled as nodes and* $\mathcal{S}$ *be labeled as a share-generator. An* (n, k, d, α, β) regenerating code *is a tuple of three protocols (*Encode, Repair, Decode*) defined as follows:*

- Encode: *This protocol has* $\mathcal{S}$ *take a file/string, represented by* $f \in F[x]^{\leq k-1}$*, as input and distributes a codeword* $\mathbf{c} = (c_1, \ldots, c_n) \in F^n$ *to the storage nodes according to their index (*c_i *to* $\mathcal{P}_i$*).*
- Repair: *Here, a failed node* $\mathcal{P}_j$ *contacts* d *other nodes, each of which sends it* β *bits of data. Then, the node computes a function on their sent data* $(\delta_1, \ldots, \delta_d)$ *to generate its new local storage.*
- Decode: *This protocol accesses any* k *nodes to reconstruct the original file/string.*

It is clear from the construction that each failed node can be reconstructed by accessing full data from at least k other nodes. However, this does not provide the optimum bandwidth. Regenerating codes facilitate the failed nodes to access fewer bits (β) from more than k surviving nodes for reconstruction.

2.3 Secret Sharing Schemes

Here we assume that all secret sharing schemes operate over finite fields. In general, we use the term *secret sharing (SS) scheme* to denote the following:

Definition 4. *Let F be a finite field. A* (k, n, F) information-theoretically secure secret threshold sharing scheme *over F is a pair of protocols used between servers labeled as, the unique and fixed sharing node $\mathcal{S}$ and the set of storage nodes $\mathcal{A} := \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$:*

> *Share(s_0): On input $s_0 \in F$, $\mathcal{S}$ randomly generates n shares $x_1, \ldots, x_n \in F$ and returns x_i to server $\mathcal{P}_i$.*
> *Reconstruct$(\mathbf{s})$: Any $k+1$ nodes combine their shares, represented as a vector $\mathbf{x} \in F^{k+1}$, to reconstruct the secret $s' \in F$.*

Let $H(\cdot)$ be the classical Shannon entropy function. For information-theoretic security, we assume s_0 is a non-trivial random variable over F. Then, the scheme's correctness and security is defined as follows:

> *Security: If $\mathbf{x} \in F^k$ is any (k)-sized subset of shares $x_{j_1}, \ldots, x_{j_k}$, then $H(s_0|\mathbf{x}) = H(s_0) > 0$.*
> *Correctness: If $\mathbf{x} \in F^{k+1}$ is any $(k+1)$-sized subset of shares $x_{j_1}, \ldots, x_{j_{k+1}}$, then $H(s_0|\mathbf{x}) = 0$.*

In the following definition, we break up the timeline into distinct phases once the shares are distributed. Each phase is represented by a positive integer σ.

Definition 5. *A* proactive secret sharing scheme *(PSS) is a secret sharing scheme as in definition 4 with the following additional algorithms:*

> *Refresh: All storage nodes $\mathcal{P}_1, \ldots, \mathcal{P}_n$ use their respective shares from phase t to generate new random shares; $x_1^{(t+1)}, \ldots, x_n^{(t+1)}$ (for the same secret). Then, it distributes $x_i^{(t+1)}$ to $\mathcal{P}_i$.*
> *Recover: A corrupted node, $\mathcal{P}_r$, contacts d uncorrupted nodes which combine their shares to compute (potentially with new randomness).*

Next, we define Shamir's secret sharing scheme and its accompanying proactive protocols [2, 10][6]. We assume the finite field is at least the size of the number of storage nodes plus one, $|F| \geq n + 1$. We denote the set of nodes needing to recover their share as $\mathcal{B}$, with $|\mathcal{B}| \leq k$, and the non-corrupted nodes as $\mathcal{D} := \mathcal{A} \setminus \mathcal{B}$. The set of evaluation points $A = \{\alpha_1, \ldots, \alpha_n\} \subset F$ is fixed beforehand and known to all nodes.

Definition 6. *The* proactive (n, k) Shamir secret sharing scheme *over a finite field F with evaluation points $A = \{\alpha_1, \cdots, \alpha_n\} \subseteq F$ is defined as follows:*

> *Share(s_0): On input $s_0 \in F$, $\mathcal{S}$ randomly generates a degree k polynomial f over F conditioned on $f(0) = s_0$. Then, $\mathcal{S}$ sends $x_i^{(0)} := f(\alpha_i) \in F$ to $\mathcal{P}_i$.*
> *Reconstruct$(\mathbf{x})$: Any $k + 1$ nodes interpolate their shares, represented as a vector $\mathbf{x} \in F^{k+1}$, to reconstruct the secret $s' \in F$.*

[6] There are more efficient PSS schemes, [11] for example, but we describe the scheme in [10] for its clear relation to Reed-Solomon codes.

Refresh: Each storage node $\mathcal{P}_i$ generates a random polynomial δ_i of degree k conditioned on $\delta_i(0) = 0$ and sends $\delta_i(\alpha_j)$ to $\mathcal{P}_j$ for all $j \neq i$. Then, each storage node $\mathcal{P}_i$ updates their share as $x_i^{t+1} \leftarrow x_i^t + \sum_j \delta_j(\alpha_i)$ (and erases all intermediate values used to compute x_i^{t+1}).
Recover: For each corrupted node $\mathcal{P}_r \in \mathcal{B}$, each $\mathcal{P}_i \in \mathcal{D}$ does the following. Generate a uniformly random polynomial of degree k, ξ_i, such that $\xi_i(\alpha_r) = 0$. Then, send $\xi_i(\alpha_j)$ to $\mathcal{P}_j$ for all $\mathcal{P}_j \in \mathcal{D}$. Each $\mathcal{P}_j \in \mathcal{D}$ updates their share as $x_j^t \leftarrow x_j^t + \sum_{i,\mathcal{P}_i \in \mathcal{D}} \xi_i(\alpha_j)$. Finally, each $\mathcal{P}_i \in \mathcal{D}$ sends its updated share x_i^t to $\mathcal{P}_r$ and $\mathcal{P}_r$ interpolates them to get its original share, x_r^t.

2.4 Leakage Model

Leakage is fundamental to modeling secret sharing schemes. Here we define leakage functions using the terminology of [12]. We restrict the output of the leakage function to field elements, either in some large field F or some subfield $L \leq F$.

Definition 7. *Let $L \leq F$ such that $t = [F : L]$, and fix a secret sharing scheme over F, denoted by $\mathsf{Share} : F \to F^n$. We denote $\mathsf{Leak}^L = (\mathsf{Leak}_1^L, \ldots, \mathsf{Leak}_n^L)$ for a* length l *L-*leakage function *with $l < t$, where $\mathsf{Leak}_i : F \to L^l$ is a leakage function, possibly randomized, defined for each node. When $(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(s_0)$, we denote the collection of leakage outputs as $(b_1, \ldots, b_n) \leftarrow \mathsf{Leak}^L(x_1, \ldots, x_n)$ where $b_i = \mathsf{Leak}^L(x_i)$. In addition, for any $S \subseteq [n]$ we define $\mathsf{Reveal}_S : F^n \to F^n$ as revealing an entire share at node i if $i \in S$. That is, $\mathsf{Reveal}_S(i) = \mathbb{1}_{i \in S} \cdot \mathsf{Share}_i$ where $\mathbb{1}_{i \in S}$ is the indicator function for S.*

For epoch i, we denote $F^{(i)} \subset [n]$ as the set of nodes which leak full shares and $L^{(i)}$ as the set of nodes which leak partial shares (elements in L^l).

3 Related Work

Now that we have introduced the basic notions and mechanisms we need and employ, let us review further related earlier work.

Proactive Secret Sharing. The notion of proactive security was first proposed by Ostrovsky and Yung [4], and subsequently utilized to protect cryptographic keys by secret sharing them and computing RSA signatures in a distributed manner [10]. Specifically, Proactive Secret Sharing (PSS) aims to protect against a mobile adversary that can change the subset of corrupted parties over time and therefore may eventually compromise all involved parties over a long period of time; the model assumes that such a mobile adversary is limited to simultaneously corrupting no more than t parties during the same period though. PSS initially only considered static groups and for settings with honest majorities, Dynamic Proactive Secret Sharing (DPSS) schemes are both proactively secure and allow the set of parties to dynamically change over time. The dynamic group problem has been addressed [13–19], but mostly for the honest majority and non-proactive settings, and only in [20] in the proactive setting.

In the dishonest majority setting most of the PSS literature [21, 22] assumes a static group of parties, i.e., unchanged during the secret lifetime. PSS protocols for *dynamic* groups with dishonest majorities were only recently constructed [23, 6]. As for any secret sharing against dishonest majorities, security is only computational. In addition to efficiently handling dynamic groups, recent work [23] introduces a notion of batched PSS that retain fairness against mixed (passive and active) adversaries and reduces the communication complexity of DPSS from $O(n^4)$ to $O(n^2)$ when batching is used and $O(n^3)$ in the single secret setting.

Regenerating Codes. A large file can divided into pieces, each of which is stored at a different node using distributed storage. Server corruption can lead to loss of information. Error-correction coding[7] techniques allow the recovery of information stored in a corrupted node using the information stored in other nodes. Regenerating codes were first introduced by Dimakis et al. in [1] to improve the repair bandwidth for distributed storage systems. The aim is to recreate the information stored in a corrupted node without recreating the entire encoded information. Given a file of size $\mathcal{M}$, it can be divided into k pieces of size $\mathcal{M}/k$ which are stored in n nodes using an (n, k) MDS code. Each node stores α symbols. Information stored in a corrupted node can be recovered by accessing β sub-symbols from d surviving nodes. This can be done in the following three ways.

- *Exact repair:* The encoded block is regenerated exactly as before.
- *Functional repair:* The corrupted nodes are regenerated such that the new system represents an MDS code of length n.
- *Exact repair of systematic parts:* This is a hybrid between the above two repair schemes. The code contains exactly one replica of the information. Systematic parts of the code are regenerated using the exact repair scheme and the non-systematic parts are regenerated using the functional repair scheme.

The trade-off between storage efficiency and repair bandwidth is a point of interest. Two special cases of regenerating codes are given by the optimal cases: minimum-storage regenerating (MSR) codes and minimum-bandwith regenerating (MBR) codes.

Other works. Though there are other works combining regenerating codes and secret sharing, we emphasize that this paper is the first to do so in the *proactive* secret sharing setting. First, Huang and Bruck [24] apply the GW paradigm to threshold secret sharing schemes and prove optimality. They stay in the simplest security model, ignoring leakage, and are not concerned with proactive schemes. A previous work by Huang et al. [25] studies the communication complexity of threshold secret schemes and presents a scheme with optimal decoding bandwidth based on Reed-Solomon codes. However, [25] does not concern repairing

[7] Regenerating codes are studied from the point of view of erasure recovery in coding theory literature. However, here we refer to it as a subset of error-correction because we also care about corruption of nodes along with node failures.

in secret sharing schemes, leakage, randomizing shares (proactive schemes), or generalized decoding structures.

4 Leakage and Reconstruction: Old Models, New Lens

This section is to show how considering a simple leakage model affects the security of proactive secret sharing (PSS) schemes across epochs. We show the connection between repairing Reed-Solomon codes (Subsection 2.1) and PSS schemes using the leakage model described in (Subsection 2.4). In other words, we show how the algorithms comprised in [8] can be used to attack an incorrectly-implemented PSS scheme.

Model. Let $(\mathsf{Share}, \mathsf{Reconstruct}, \mathsf{Refresh}, \mathsf{Recover})$ be a Shamir-based (n, k, F) PSS scheme (Definition 6) with evaluation points $A' \subset F$. Our model is simple. During each epoch of a PSS scheme an adversary $\mathcal{A}$ receives either nothing, a collection of $l < t$ small field elements ($\alpha \in L^l$), or a full field element from each node. Let $l^{(i)}$ be the number of subfield elements leaked in epoch i, and let $f^{(i)}$ be the number of full field elements leaked during epoch i. The key notion throughout this section is that the linear transformations used in [8]'s reconstruction algorithms are *independent* of the polynomial f representing the distributed data (potentially secret). In the notation of Definition 7, the polynomials $\mu_{\zeta,\alpha}(x)$ only depend on A[8]. Lastly, we extend the evaluation points to include 0 in order to bridge Definitions 6 and 7: $A := A' \cup \{0\}$.

4.1 Static Leakage.

Here we look at the case to where the leakage function Leak^L is static between epochs.

Proposition 1. *Let b be the repair bandwidth of the Reed-Solomon linear exact repair scheme also being used as a Shamir-based PSS scheme. There is an efficient adversary which receives $l^{(i)}$ leaked subfield elements and $f^{(i)}$ leaked full field elements which needs $b-(tf^{(i)}+l^{(i)})$ subfield elements during a single epoch to reconstruct the secret $s_0 \in F$ for a static leakage function between epochs.*

The above proposition shows how an adversary can be under the security threshold for whole shares, $f^{(i)} < k$, during each epoch but can still reconstruct the secret!

[8] For example, Corollary 9 in [8] constructs these polynomials as $\mu_{\zeta,\alpha}(\alpha^*) = p(\alpha) \cdot \frac{\prod_{\beta \in A \setminus \{\alpha^*\}}(\beta - \alpha^*)}{\prod_{\beta \in A \setminus \{\alpha\}}(\beta - \alpha)}$ where p is a polynomial dependent only on the evaluation points A.

4.2 Dynamic Leakage.

Here we consider a leakage function that changes between epochs. This setting represents the case where storage nodes fail to completely erase some data used in computing the Refresh protocol.

Proposition 2. *Let b be the repair bandwidth of the Reed-Solomon linear exact repair scheme also being used as a Shamir-based PSS scheme. Then, there is an efficient adversary which needs b leaked subfield elements in order to reconstruct the secret across epochs assuming the leaked nodes store a non-zero value.*

Note that this is a strong leakage model. However, updating the leakage function via multiplying and inverting finite field elements at each node is a plausible scenario which should be known to those utilizing and implementing PSS schemes.

5 On the Equivalence of Regenerating Codes and Proactive Secret Sharing

In this section we prove the equivalence between Regenerating Codes (RC) and threshold Proactive Secret Sharing (PSS) under certain conditions. Theorem 1 treats the PSS to RC direction, while Theorem 2 treats the reverse one, restricted to linear, MDS codes. These properties of the RC are required for threshold security and to construct a simple Refresh protocol.

Theorem 1. *For each (t, n) proactive secret sharing scheme represented as a tuple of algorithms (Share, Reconstruct, Recover, Refresh) which stores α bits at each node and contacts d nodes in the $\mathsf{Recover}_{PSS}$ protocol (Definition 5), each sending β bits of data to the failed node, there is an erasure $(n' = n, k' = t+1, d' = d, \alpha' = \alpha, \beta' = \beta)$ regenerating code represented as a tuple of algorithms (Encode, Decode, Repair), as in Definition 3.*

Remark 1. We note that the rate given implicitly in Theorem 1 is only $1/n$. However, there are clear ways to achieve a better rate. The first is when the PSS scheme is linear. Here, the encoding procedure usually involves a linear code of dimension $t+1$ and t of the input symbols are uniformly random elements in F. This randomness is only for security so we can replace these t symbols with data elements. The second is the case of batching [20], employing a basic technique from [26], where the PSS scheme takes a secret as an element in F^l. The proof of Theorem 1 for the batching case is the same except $\mathbf{s}, \mathbf{s}' \in F^l$.

The other direction, from RC to PSS only makes sense if we are able to show threshold security. This was proven for non-proactive secret sharing schemes with repair in [24, Theorems 1 and 2]. Here we extend this result to the proactive setting.

Theorem 2. *For every $(n+1, k+1, d, \alpha, \beta)$ MDS linear regenerating code over F, there is a (k, n) proactive secret sharing scheme whose $\mathsf{Recover}_{PSS}$ protocol contacts d nodes, each sending β bits to the damaged node.*

Theorem 2 and the result of Guruswami and Wootters [8], Theorem 4 in the special case where the number of parties equals the degree of the extension, together imply the existence of an alternative $\mathsf{Recover}_{PSS}$ protocol that only receives symbols in a subfield for the Shamir-based PSS scheme. This may be advantageous in settings with restricted communication during the recovery phase. This is summarized in the following Corollary.

Corollary 1. *The Shamir-based PSS scheme, Definition 6 [2, 10], over a finite field F with subfield $L \leq F$ with degree $t = [F : L]$ has an alternative $\mathsf{Recover}_{PSS}$ protocol which contacts the remaining $n-1$ nodes and receives t symbols in L from each node in order to recover the lost share.*

We emphasize that Corollary 1's efficient recover protocol is information-theoretically secure in the setting illustrated by Theorem 4, without leakage. This is because Theorem 4's repair algorithm only sends a user t subfield elements and its secret is exactly comprised of t subfield elements.

6 From General Adversary Structures to General Decoding Structures

In this section we demonstrate how, due to the relationships between the notions, an interesting useful paradigm in one area (PSS) can induce an interesting new notion in the other area (RC). Secret sharing (SS) schemes, and proactive SS (PSS) in particular, can be extended [20, 23, 22, 6] to accommodate for different adversaries beyond a threshold one (i.e., more general secure subsets of shares), and to deal with dynamic change of the servers holding the shares (dynamic groups). Such extensions make sense in the case of a network of nodes/processors, holding shares jointly. Since RC is a coding theoretic technique for a network of nodes/processors to hold sub-codewords, the extended notions make sense in storage-oriented RC codes. Here we demonstrate such translation of a relevant extension.

The most generic adversarial capabilities are captured (since PODC'97 [5]) using the *general adversary structure (GAS)* in the secure computation literature. The GAS notion is a more general (and practically motivated via different availability of different types of servers in a network) and more flexible one when modeling adversaries, compared to only the threshold limitation on corruptions. GAS applies to various scenarios, for example when only special combinations of nodes is required to reconstruct a secret, when some nodes are authorized by some authority and others by another authority and combination of authorities is required, etc.

Let $2^{\mathcal{P}}$ denote the set of all the subsets of nodes ($\mathcal{P}$) involved in a secret sharing scheme. A subset of $2^{\mathcal{P}}$ is *qualified* if nodes in the subset can reconstruct

the secret, while a subset of $2^{\mathcal{P}}$ that nodes in the set obtain no information about the secret is called *ignorant*. Every subset of $\mathcal{P}$ is either qualified or ignorant[9]. The secrecy condition is stronger: even if any ignorant set of nodes hold any kind of partial information about the shared value, they must not obtain any additional information about the shared value.

The *access structure* Γ is the set of all qualified subsets of $\mathcal{P}$ and the *secrecy structure* Σ is the set of all ignorant subsets of $\mathcal{P}$. Naturally, Γ includes all supersets of each element in it (so often called *monotone access structure*), while Σ includes all subsets of each element in it. We call such minimum or maximum sets as *basis structure*, and denote it with $\tilde{\cdot}$. i.e., the basis access structure $\tilde{\Gamma}$ is the set of all minimal subsets in Γ, and the basis secrecy structure $\tilde{\Sigma}$ is the set of all maximal subsets in Σ.

The *adversary structure* $\Delta \subseteq \Sigma$ is a set of subsets of nodes that can be potentially corrupted. The adversary can choose a set in Δ and corrupt all the nodes in the set. Note that the adversary structure in t-threshold SS is the set of all subsets of $\mathcal{P}$ of at most t nodes and GAS extends this to non-threshold models. A GAS includes all of these structures, (Γ, Σ, Δ). We define a family of security properties on the sets Δ by a covering condition: $\Delta \in Q^k(\mathcal{P}, \Delta)$ if for all distinct $A_1, \ldots, A_k \in \Delta$, $A_1 \cup \cdots \cup A_k \neq \mathcal{P}$ [5, 28]. That is, no k different adversary patterns can cover the set of nodes in the protocol. Further, we are focused on the $Q^2(\mathcal{P}, \Delta)$ setting: $\forall\ A, B \in \Delta, A \cup B \neq \mathcal{P}$.

One can extend the definition of secret sharing for the threshold structure (see Definition 4) to the GAS as follows:

Definition 8 (Secret Sharing for GAS). *Let F be a finite set. An (n, F)* information-theoretically secure secret threshold sharing scheme *over F for a GAS (Γ, Σ, Δ) is a pair of protocols used between servers labeled as the, unique and fixed, sharing node $\mathcal{S}$ and the set of storage nodes $\mathcal{A} := \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$:*

> *$\mathsf{Share}(s_0)$: on input $s_0 \in F$, $\mathcal{S}$ randomly generates n shares $x_1, \ldots, x_n \in F$ and returns x_i to server $\mathcal{P}_i$.*
> *$\mathsf{Reconstruct}(\mathbf{s})$: Any set of nodes in Γ can combine their shares, represented as a vector of elements $\mathbf{x}$, to reconstruct the secret $s' \in F$.*

For information-theoretic security, we assume s_0 is a non-trivial random variable over F. Then, the scheme's correctness and security is defined as follows:

> *Security: if $\mathbf{x}$ is any subset of shares corresponding to a set in Σ, then $H(s_0|\mathbf{x}) = H(s_0) > 0$.*
> *Correctness: if $\mathbf{z}$ is any subset of shares with indices corresponding to a set of nodes in Γ then $H(s_0|\mathbf{z}) = 0$.*

When extending the definition to the proactive setting, we break up the timeline into distinct phases once the secret shares are distributed. Each phase is represented by a positive integer σ (analogous to Definition 5).

[9] That is, we only consider *perfect* secret sharing schemes [27, Definition 11.59].

Definition 9 (Proactive Secret Sharing for GAS). *A* proactive secret-sharing scheme *for a GAS (Γ, Σ, Δ) is a secret-sharing scheme as in definition 8 with the following additional algorithms:*

> *Refresh: All storage nodes $\mathcal{P}_1, \ldots, \mathcal{P}_n$ use their respective shares from phase t to generate new random shares (for the same secret), $x_1^{(t+1)}, \ldots, x_n^{(t+1)}$. Then, it distributes $x_i^{(t+1)}$ to $\mathcal{P}_i$.*
> *Recover: A corrupted node, $\mathcal{P}_r$, contacts d uncorrupted nodes which combine their shares to compute (potentially with new randomness) a new share for the corrupted node. $\mathcal{P}_r$ receives their new (recovered) share.*

Lemma 1. *There exists a proactive secret sharing (PSS) scheme for general adversary structures (GAS) as defined in Definition 9 with $Q^2(\mathcal{P}, \Delta)$ adversaries.*

Proof. This follows a constructive proof, from a PSS for a GAS given in [6].

6.1 Generalized Decoding in Regenerating Codes

Let $\mathcal{P}$ be a set of all servers involved in distributed storage and $2^{\mathcal{P}}$ all its subsets. A subset of $2^{\mathcal{P}}$ is available if servers in the subset can be accessed to reconstruct the original data, while a subset of $2^{\mathcal{P}}$ that cannot be used for data reconstruction is called unavailable. Every subset of $\mathcal{P}$ is either available or unavailable. Therefore, let Γ_{RC} be the sets of servers which can decode when available. We call Γ_{RC} the *decoding structure*. Any decoding structure is monotone, i.e., it is closed under taking supersets.

The analogy between Γ on the secret sharing side and Γ_{RC} on the regenerating code side is clear: these are the sets which can decode the secret. So, there must remain one which is not tampered with, or, equivalently, an erasure pattern, υ, can come from $\mathcal{P}$ as long as there exists $B \in \Gamma$ such that $\upsilon \cap B = \varnothing$. On the other hand, the coding analogy of the sets of tolerable active adversaries, Δ, is more subtle. It is, however, the sets of error patterns which the code can withstand and still decode. In addition, we define the error property $Q^k_{RC}(\mathcal{P}, \Delta)$ analogously in the coding setting: For all distinct $A_1, \ldots, A_k \in \Delta_{RC}$, $A_1 \cup \cdots \cup A_k \neq \mathcal{P}$.

Definition 10 (Generalized-decoding Regenerating Code (GRC)). *Let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be labeled as servers and $\mathcal{S}$ be labeled as a share-generator. Let F be a finite set. An (n, k, d, α, β) general error and erasure regenerating code for a decoding structure for $(\Gamma_{RC}, \Delta_{RC})$, $\varnothing \neq \Gamma_{RC}, \Delta_{RC} \subset 2^{\mathcal{P}}$ and $\Gamma_{RC} \cap \Delta_{RC} = \varnothing$, is a tuple of three protocols (Encode, Repair, Decode) which are defined as follows:*

> *Encode: This protocol has $\mathcal{S}$ take a file, represented by $f \in F[x]^{\leq k-1}$, as input and distributes a codeword $\mathbf{c} = (c_1, \ldots, c_n) \in F^n$.*
> *Repair: Here, a failed node $\mathcal{P}_j$ contacts d other uncorrupted, each of which sends it β bits of data. Then, the node computes a function on their sent data $(\delta_1, \ldots, \delta_d)$ to generate its new local storage.*

Decode: This protocol accesses any set of servers in Γ_{RC} to reconstruct the original file.

Moreover, we require the following correctness constraints:

Errors: for all error patterns in Δ_{RC}, Decode correctly decodes to the original message.
Erasures: for all erasure patterns υ such that there exists a $B \in \Gamma_{RC}$ such that $\upsilon \cap B = \varnothing$, Decode correctly decodes to the original message.

Now we show the existence of any GRC with Q^2_{RC} error patterns via the mapping applied in Theorem 1's proof. We emphasize that the following theorem is an existence result and not optimized for parameters. We leave optimization open for future works.

Theorem 3. *For every $(\Gamma_{RC}, \Delta_{RC})$ with property $Q^2_{RC}(\mathcal{P}, \Delta_{RC})$, there exists a linear GRC as defined in Definition 10 over a finite field F. Moreover, all erasure patterns, υ, such that there is a $B \in \Gamma_{RC}$ such that $\upsilon \cap B = \varnothing$ can be correctly decoded.*

7 Conclusion and Future Directions

In this paper we conduct, to the best of our knowledge, the first systematic study of the relationship between Regenerating Codes (RC) and Proactive Secret Sharing (PSS), two subareas with meaningful applications in distributed computing (fault tolerant storage, and secure multi-party computation protocols, resp.). We show that a PSS scheme can be converted to a RC, and that under some conditions a RC can be utilized to instantiate a PSS scheme. Proving this connection allows us to leverage recent results for repairing codes (i.e, efficient polynomial interpolation) to improve efficiency of bottlenecks in PSS. We also show that if parameters are not carefully calibrated, these new interpolation techniques may be used to attack PSS when secrets are maintained, and thus the scheme also attacked, for a long period of time. We also demonstrate how the relationships we uncover allow one to translate extensions of PSS to relevant novel extensions in the RC realm. Our work, being the first to point out these relationships, paves the way for several interesting research directions, among them: (1) further investigating the new notion of Generalized-decoding Regenerating Code (GRC), and developing more efficient coding schemes satisfying this notion (especially under specific structures that are most relevant to networks underlying storage systems), and ideally understanding the limits on achievable rates for GRC; (2) defining and realizing GRC with generalized repair structures that are different than the generalized decoding sets; and finally, (3) dealing with dynamic groups for both threshold and the general adversaries cases in the context of GRC (most relevant when network nodes move in and out of the storage system), and other possible extensions, e.g., verifiable RC as an analogue notion to verifiable Secret Sharing [29].

References

1. A. G. Dimakis, B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, 2010.
2. A. Shamir, "How to share a secret," *Commun. ACM*, 1979.
3. R. J. McEliece and D. V. Sarwate, "On sharing secrets and reed-solomon codes," *Commun. ACM*, 1981.
4. R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks (extended abstract)," in *PODC*, 1991.
5. M. Hirt and U. M. Maurer, "Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract)," in *PODC*, IEEE, 1997.
6. K. Eldefrawy, S. Hwang, R. Ostrovsky, and M. Yung, "Communication-efficient (proactive) secure computation for dynamic general adversary structures and dynamic groups," in *SCN*, 2020.
7. S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song, "Churp: Dynamic-committee proactive secret sharing," CCS '19, 2019.
8. V. Guruswami and M. Wootters, "Repairing reed-solomon codes," in *STOC*, 2016.
9. E. R. Berlekamp, "Bounded distance+1 soft-decision reed-solomon decoding," *IEEE Trans. Inf. Theory*, 1996.
10. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *CRYPTO*, 1995.
11. J. Baron, K. Eldefrawy, J. Lampkins, and R. Ostrovsky, "How to withstand mobile virus attacks, revisited," in *PODC*, ACM, 2014.
12. J. B. Nielsen and M. Simkin, "Lower bounds for leakage-resilient secret sharing," in *EUROCRYPT*, 2020.
13. I. Damgård and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," in *CRYPTO*, 2007.
14. I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. Smith, "Scalable multiparty computation with nearly optimal work and resilience," in *CRYPTO*, 2008.
15. I. Damgård, Y. Ishai, and M. Krøigaard, "Perfectly secure multiparty computation and the computational overhead of cryptography," in *EUROCRYPT*, 2010.
16. Y. Desmedt and S. Jajodia, "Redistributing secret shares to new access structures and its applications," *Technical Report ISSE TR-97-01, George Mason University*, July 1997.
17. D. Schultz, *Mobile Proactive Secret Sharing*. PhD thesis, Massachusetts Institute of Technology, 2007.
18. T. M. Wong, C. Wang, and J. M. Wing, "Verifiable secret redistribution for archive system," in *IEEE Security in Storage Workshop*, 2002.
19. L. Zhou, F. B. Schneider, and R. van Renesse, "Apss: proactive secret sharing in asynchronous systems," *ACM Trans. Inf. Syst. Secur.*, 2005.
20. J. Baron, K. Eldefrawy, J. Lampkins, and R. Ostrovsky, "Communication-optimal proactive secret sharing for dynamic groups," in *ACNS* (T. Malkin, V. Kolesnikov, A. B. Lewko, and M. Polychronakis, eds.), 2015.
21. S. Dolev, K. Eldefrawy, J. Lampkins, R. Ostrovsky, and M. Yung, "Proactive secret sharing with a dishonest majority," in *SCN*.
22. K. Eldefrawy, R. Ostrovsky, S. Park, and M. Yung, "Proactive secure multiparty computation with a dishonest majority," in *SCN*, 2018.
23. K. Eldefrawy, T. Lepoint, and A. Leroux, "Communication-efficient proactive secret sharing for dynamic groups with dishonest majorities," in *ACNS* (M. Conti, J. Zhou, E. Casalicchio, and A. Spognardi, eds.), 2020.

24. W. Huang and J. Bruck, "Secret sharing with optimal decoding and repair bandwidth," in *ISIT*, IEEE, 2017.
25. W. Huang, M. Langberg, J. Kliewer, and J. Bruck, "Communication efficient secret sharing," *IEEE Trans. Inf. Theory*, 2016.
26. M. K. Franklin and M. Yung, "Communication complexity of secure computation (extended abstract)," in *STOC*, 1992.
27. R. Cramer, I. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.
28. M. Hirt and U. M. Maurer, "Player simulation and general adversary structures in perfect multiparty computation," *J. Cryptol.*, 2000.
29. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract)," in *FOCS*, 1985.

A Additional Background

General background A *protocol* is any efficient (polynomially-bounded number of rounds in the input's bit-length) joint computation between probabilistic polynomial-time (PPT) Turing machines. Protocols will be written in sans serif font (Share, for example). A *scheme* is a collection of protocols, often with a set order of when the protocols are executed. We denote the protocol's participants as $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ and we may use $\mathcal{S}$ to denote a special participant (e.g. secret-share generator). We denote a random variable being sampled from some distribution as $Y \leftarrow D$. When D is a finite set, $Y \leftarrow D$ represents Y being sampled from the uniform distribution over D. Shannon entropy is represented as $H(Y) := -\sum \log_2(p(y))$ where the sum is over the domain of Y and $p(y) = Pr\{Y = y\}$.

Communication Model We assume participants have access to a universal clock (i.e., a synchronous model) throughout protocols and each server can access a broadcast channel as well as a channel to another server. Bilateral channels are modeled as private and they send messages instantly. When needed, we will break up the timeline into *phases* or *epochs*.

A.1 Repairing Reed-Solomon Codes

Given any (n, k) Reed-Solomon code, classical erasure correction requires access to k codeword symbols to determine $f(\alpha_i)$. Guruswami and Wootters ingeniously showed in [8] that the bandwidth can be improved by accessing fewer sub-symbols of more than k codeword symbols. Consider the following definition of a linear exact repair scheme.

The following theorem summarizes the result for Reed-Solomon codes over general field extensions.

Theorem 4 (Lemma 4 in [24], Implicit in [8]). *Let F be a t-degree extension of a finite field L, let f be a polynomial of degree at most $k-1$ over F, and let $f(\alpha_1), \ldots, f(\alpha_n)$ be evaluations of f on n distinct points $\alpha_1, \ldots, \alpha_n$. Let α_0 be an*

element in F and let $g_1(x), \ldots, g_t(x)$ be t distinct polynomials over F of degree at most $n-k$ such that $\{g_i(\alpha_0)\}_{i\in[t]}$ is a basis for F over L. Then, it suffices to know the set of values $\bigcup_{i\in[n]}\{tr_{F/L}(g_j(\alpha_i)f(\alpha_i))\}_{j\in[t]}$ in order to recover $f(\alpha_0)$.

Definition 11. *[8, Definition 5] Let C be a linear code over F of length n and dimension k, given by a set of function $\mathcal{F}$ ($\mathcal{F} = F[x]^{\leq k-1}$ in the Reed-Solomon case) and a set of evaluation points $A \subseteq F$. A* linear exact repair scheme *for C over a subfield $L \leq F$ consists of the following.*

- *For each $\alpha^* \in A$, and for each $\alpha \in A\backslash\{\alpha^*\}$, a set of queries $Q_\alpha(\alpha^*) \subseteq F$.*
- *For each $\alpha^* \in A$, a linear reconstruction algorithm that computes $f(\alpha^*) = \sum \lambda_i\nu_i$, for coefficients $\lambda_i \in L$ and a basis $\nu_1, \nu_2, \ldots, \nu_t$ for F over L, so that the coefficients λ_i are L-linear combinations of the queries:*

$$\bigcup_{\alpha\in A\backslash\{\alpha^*\}} \{tr_{F/L}(\gamma f(\alpha)) : \gamma \in Q_\alpha(\alpha^*)\}.$$

The repair bandwidth,

$$b = \max_{\alpha^*\in A} \sum_{\alpha\in A\backslash\{\alpha^*\}} |Q_\alpha(\alpha^*)|,$$

of the exact repair scheme is the total number of sub-symbols in L returned by each node α.

Given input set A of points of evaluation, $\alpha^* \in A$ a failed node, and for some $f \in \mathcal{C}$, access to linear queries of the for $tr_{F/L}(\gamma f(\alpha))$ the Guruswami-Wootters algorithm outputs $f(\alpha^*)$. The algorithm is summarized in [8, Algorithm 1].

B Missing Proofs

Here we give the proofs for the accompanying statements in the paper.

B.1 Proof of Proposition 1

Proof. In terms of the leakage model, the function

$$\begin{aligned}\mathsf{Leak}_i^L(f^{(i)}(\alpha_i)) = (&tr_{F/L}(\gamma_1 f^{(i)}(\alpha_i)), tr_{F/L}(\gamma_2 f^{(i)}(\alpha_i)), \ldots \\ &\ldots, tr_{F/L}(\gamma_l f^{(i)}(\alpha_i)))\end{aligned}$$

for $\gamma_j \in \tilde{Q}_\alpha(\alpha^*)$ and $\alpha^* = 0$. Each full symbol $f(\alpha_j)$ allows the adversary to compute the queries $\{tr_{F/L}(\gamma f(\alpha_j)) : \gamma \in Q_\alpha(\alpha^*)\}$ locally. The proposition follows from the adversary receiving the remaining queries in $Q_\alpha(\alpha^*)$.

B.2 Proof of Proposition 2

Proof. Fix an epoch h. In terms of the leakage model, the function

$$\mathsf{Leak}_i^L(f^{(i)}(\alpha_i)) = (tr_{F/L}(\gamma_1^h f^{(i)}(\alpha_i)), tr_{F/L}(\gamma_2^h f^{(i)}(\alpha_i)), \ldots \\ \ldots, tr_{F/L}(\gamma_l^h f^{(i)}(\alpha_i)))$$

for $l \in \tilde{Q}_\alpha(\alpha^*)$, $\alpha^* = 0$, and $\gamma_j^h := \gamma_j \cdot f^{(0)}(\alpha_i)(f^{(i)}(\alpha_i))^{-1}$ if $f^{(i)}(\alpha_i) \neq 0$.

B.3 Proof of Theorem 1

Proof. We prove this by computational reduction on the scheme/protocols.

- **Encode_{RC}:** Let $\mathcal{S}$ be the encoding/sharing node in regenerating code. It has oracle access to the Share_{PSS} protocol from the PSS scheme. On input s, it calls $\mathsf{Share}_{PSS}(s)$, collects the n shares and distributes them as codewords to the n nodes.
- **Decode_{RC}:** Here, the protocol gathers $t+1$ shares/ codewords, $\mathbf{x}$, and inputs them into the $\mathsf{Reconstruct}_{PSS}$ protocol which outputs some $s' \in F$.
- **Repair_{RC}:** Lastly, the Repair protocol on input node i, the damaged node, runs the $\mathsf{Recover}_{PSS}$ which contacts d uncorrupted nodes and outputs the recovered share.

From the reduction, it is clear that $n' = n$, $k' = t+1$, $d' = d$, $\alpha' = \alpha$, $\beta' = \beta$. Further, correctness follows from the correctness of $\mathsf{Reconstruct}_{PSS}$ and $\mathsf{Recover}_{PSS}$.

B.4 Proof of Theorem 2

Proof. Without loss of generality, assume the code is represented by a generator matrix in systematic form: $\mathbf{G} = [\mathbf{I}_{k+1}|\mathbf{P}] \in F^{(k+1)\times(n+1)}$.

- **$\mathsf{Share}_{PSS}(s)$:** Given $s \in F$, sample k uniformly random elements in F , denoted as r_i, to form the vector $\mathbf{y} = (s, r_1, \ldots, r_k)$. Then, encode the vector $\mathbf{y}$ to get $(x_0, x_1, \ldots, x_n)^t \leftarrow \mathbf{y}^t\mathbf{G}$ and distribute the shares as $x_1, \ldots, x_n$ to the respective nodes.
- **$\mathsf{Reconstruct}_{PSS}(\mathbf{z})$:** Here we take $\mathbf{z} \in F^{k+1}$ representing k shares and call $\mathsf{Decode}_{RC}(\mathbf{z})$ to get $\mathbf{y}$. Return $y_0 = s$.
- **$\mathsf{Recover}_{PSS}(i)$:** Run the Repair_{RC} protocol. By definition, $\mathcal{P}_i$ has their share restored after the protocol.
- **$\mathsf{Redistribute}_{PSS}$:** node i runs $(0, x_{i,1}, \ldots, x_{i,n}) \leftarrow \mathsf{Share}_{PSS}(0)$ locally and sends x_j to node $\mathcal{P}_j$. Then, each node $\mathcal{P}_i$ updates their share as $x_i^{t+1} \leftarrow x_i^t + \sum x_{i,j}$ where $x_{i,j}$ is the i-th entry of the vector generated by node $\mathcal{P}_j$.

Security follows from the MDS property: any k coordinates of the code is isomorphic to F^k. Therefore any k collection of shares are uniformly random. The $\mathsf{Redistribute}_{PSS}$ protocol only works with passive adversaries. The new shares after the $\mathsf{Redistribute}$ protocol form a uniformly random encoding of s by linearity[10].

[10] The relationship between threshold secret sharing schemes and MDS codes is well-established. See Section 11.12.3 of [27] for further details.

B.5 Proof of Theorem 3

Proof. We prove this by computational reduction on the scheme/protocols. First, we take the existence of a PSS on the sets $(\Gamma := \Gamma_{RC}, \Sigma, \Delta := \Delta_{RC})$ from Lemma 1.

- **Encode**$_{RC}$**:** Let $\mathcal{S}$ be the encoding node in GRC. It has oracle access to the Share_{GAS} protocol from the PSS scheme with GAS. On input $s \in F$, it calls $\mathsf{Share}_{GAS}(s)$, collects the n shares and distributes them to the n nodes.
- **Decode**$_{RC}$**:** Here, a decoding node calls the $\mathsf{Reconstruct}_{GAS}$ protocol which outputs some $s' \in F$.
- **Repair**$_{RC}$**:** Lastly, the Repair protocol on input node i, the damaged node, runs the $\mathsf{Recover}_{GAS}$ which contacts d uncorrupted nodes and outputs the recovered share.

Correctness in error and erasure correction follows by contradiction. If there were an error pattern in Δ_{RC} which does not decode correctly, then there is an active adversary which can break the underlying PSS scheme. For erasures, this follows from the definition of Γ.

For Repair_{RC}, the damaged node must be repaired, otherwise Definition 8 would be contradicted.

C Example

Consider a $(4, 2, 3, 2, 1)$ regenerating code: $((A_1, A_2), (B_1, B_2), (A_1 + B_1, A_2 + B_2), (A_2 + B_1, A_1 + A_2 + B_2))$. If the node (A_1, A_2) fails, it can be recovered by accessing B_2, $A_2 + B_2$, and $A_1 + A_2 + B_2$ from the 3 other surviving nodes. The above code can also be viewed as a $(1, 4)$ secret sharing scheme. The corrupted share, say (A_1, A_2) can be recovered by accessing the remaining 3 shares. Furthermore, the information at each node can be modified in a timely manner so that we obtain the same MDS code (by property of regenerating codes). That is, the shares of each party can be updated regularly. The security follows from [24, Theorems 1 and 2].