Efficient Secret Sharing for Large-Scale Applications

Sarvar Patel
Google
New York, USA
sarvar@google.com

Giuseppe Persiano
Università di Salerno and Google
Salerno and New York, Italy and USA
giuper@gmail.com

Joon Young Seo
Google
New York, USA
jyseo@google.com

Kevin Yeo
Google and Columbia University
New York, USA
kwlyeo@google.com

ABSTRACT

Threshold secret sharing enables distributing a message to n parties such that no subset of fewer than t parties can learn the message, whereas any subset of at least t parties can recover the message. Despite being a fundamental primitive, secret sharing still suffers from one significant drawback, where its message reconstruction algorithm is computationally expensive for large privacy thresholds t. In this paper, we aim to address this significant drawback.

We study general (t, c)-ramp secret sharing schemes where the number of parties c needed to reconstruct the secret may be larger than t. We present a ramp secret sharing scheme whose reconstruction time is 2-7.8x faster than prior constructions suitable against adversaries that adaptively corrupt parties. For $t = 2^{20}$, our new protocol has reconstruction time of 5 seconds whereas prior work requires nearly half a minute. We see improvements starting from as small as $t = 256$. Furthermore, we obtain correctness threshold as small as $c \geq 1.05t$. To obtain our construction, we first improve the secret sharing frameworks by Cramer *et al.* (EUROCRYPT'15) and Applebaum *et al.* (CRYPTO'23) from erasure codes. Our new framework obtains secret sharing schemes that may be used against adversaries with adaptive corruptions while requiring only weaker correctness guarantees from the underlying erasure code with a distributed generation property. Furthermore, our new framework also maintains the linear homomorphism of the prior works. Afterwards, we present a concretely efficient erasure code from random band matrices that satisfies the distributed generation property.

We show that our secret sharing scheme can improve many real-world applications. In secure aggregation protocols for federated learning, we obtain up to 22% reductions in computational cost by replacing Shamir's scheme with our construction. We extend our protocol to obtain a verifiable ramp secret sharing scheme where each party can verify the consistency of the shares. Our new verifiable ramp secret sharing has 8.2-25.2x faster sharing and 2.7-23.2x faster reconstruction time compared to prior works. Finally, we present an improved distributed verifiable random function that may be used for decentralized randomness beacons.

1 INTRODUCTION

Secret sharing, introduced by Blakley [12] and Shamir [70], considers the problem of sharing an input message m amongst n parties. The original works considered *threshold secret sharing* that guaranteed that any subset of at most $t-1$ parties could not learn message m, whereas any subset of at least t parties could reconstruct m.

Ramp secret sharing is a generalization of threshold secret sharing introduced in [13] with two parameters: *privacy threshold* t and *correctness threshold* $c \geq t$. Privacy of the input message m remains identical: no subset of at most $t-1$ parties can learn anything about m. For decoding, ramp schemes guarantee that any subset of at least c parties can reconstruct m. Note that threshold schemes are a special case of ramp schemes with $c = t$. Since ramp schemes provide weaker recovery guarantees when $c > t$ (more parties are needed to decode the input message) one would expect more efficient constructions than threshold schemes.

Despite being introduced more than 40 years ago, Shamir's scheme [70] remains widely used today in both academia and industry (some examples include Binance [60], cryptocurrency wallets [11] and secure vault storage [71]). In academic works, Shamir's scheme is utilized in many state-of-the-art constructions including secure aggregation [5, 16] and multi-party computation (see, for example, [7, 23, 41]). Even though Shamir's secret sharing is widely used, it suffers from a significant bottleneck when scaling to larger use cases. In particular, the message reconstruction algorithm is computationally expensive as it requires interpolating a $(t-1)$-degree polynomial. Many implementations of Shamir's secret sharing utilize Lagrange interpolation using $O(t^2)$ time that is very expensive for large choices of t. Even if one uses more efficient $O(t \log^2 t)$ polynomial interpolation [17, 73], message reconstruction in Shamir's scheme is still very expensive for large t as we will show. We note that prior works have studied secret sharing with near-linear reconstruction (such as [14, 29, 36, 54]), but the majority have considered theoretical constructions. The only work that considers a practical instantiation is Applebaum *et al.* [2] that obtains non-negligible reconstruction error even against weaker adversaries with static corruption.

This slow reconstruction bottleneck is unfortunately inherited by cryptographic primitives relying on secret sharing. One example is verifiable secret sharing (VSS) where each of the n parties can check the consistency of their share. State-of-the-art synchronous VSS constructions [40, 51, 64, 73] build on secret sharing. Distributed verifiable random functions (DVRF) [18, 42] are another such cryptographic primitive where many constructions rely on secret scheme to combine randomness from multiple sources. We can also consider important applications that rely on any of these primitives. One application is secure aggregation for federated learning [5, 16] that use secret sharing to reconstruct values from dropout users as deployed at Google [39]. Another example is decentralized randomness beacons [25] enabling multiple parties

to generate trusted randomness that have been deployed by CloudFlare [28], DFINITY [48] and drand [35]. All these applications inherit the slow reconstruction times of the secret sharing scheme.

1.1 Our Contributions

Our contributions in this paper are threefold. First, we present an improved framework for constructing secret sharing from erasure codes with weak correctness properties that still protect against adaptive corruptions. Next, we construct a more efficient linear erasure code from random band matrices with efficient decoding and negligible error to obtain our secret sharing scheme RB-SS. Finally, we use RB-SS to improve multiple applications.

Secret Sharing Framework. To build our secret sharing scheme, we start from the framework of Cramer *et al.* [29] that builds linear secret sharing from a linear erasure code and a linear universal hash function. However, this construction required a strong correctness guarantee of the underlying erasure code that the original input could be recovered for any choice of erasures in the codeword. A recent work by Applebaum *et al.* [2] presented a modified variant that considered erasure codes with weaker correctness properties and replaced the universal hash function with a custom efficient randomness extractor. The weaker correctness guaranteed that, for any choice of erasures, the original input could be recovered except with some small probability. The weaker correctness requirements enabled the usage of more practically efficient erasure codes. However, the modified framework of Applebaum *et al.* [2] comes with the downside that reconstruction correctness guarantees only hold against a static adversary that corrupts parties before the start of the secret sharing scheme. In particular, it is essential that the corruptions are chosen independently of the erasure code. In other words, the adversary must compromise parties before the erasure code is randomly generated in the protocol.

In our work, we present a new framework to build secret sharing schemes that can rely on linear erasure codes with weaker correctness guarantees but still provide reconstruction success against adaptive adversaries. For this, we make the following modification. In both prior works [2, 29], the sparse generator matrix $\mathbf{M}$ related to the linear erasure code was published at the beginning of the protocol. Instead, we take the following approach where each row of $\mathbf{M}_i$ is generated independently and can be presented using a succinct representation. The succinct representation of the i-th row is distributed to the i-th party along with their share. Critically, the entire matrix $\mathbf{M}$ is never revealed. An adversary can only learn about $\mathbf{M}$ by corrupting parties. Furthermore, each row of $\mathbf{M}$ is generated independently from all other rows. We denote this property as *distributed generation* for erasure codes. With this property, we show that an erasure code with weak correctness properties is now sufficient to obtain constructions suitable for adaptive corruptions. In particular, we leverage the fact that an adversary that adaptively corrupts parties cannot learn information about rows owned by other parties (without compromising them).

Finally, prior works suggested the usage of linear universal hash functions [29] or a randomness extractor based on the inner product with a small-integer vector [2]. Our framework could use either option that would result in a linear secret sharing scheme. We also provide a third option of using a random oracle that is slightly more efficient at the cost of linearity. Nevertheless, we note there are several important applications of secret sharing that do not require linearity (such as federated learning [16]). We stress our framework may be instantiated with any of the three options depending on the necessity of linearity.

Linear Erasure Codes from Random Band Matrices. We will use *random band matrices* [32] to build a linear erasure code with distributed generation. In particular, we will focus on building a *distributed generator matrix* where each row may be generated independently. As originally presented, random band matrices of dimension $n \times \ell$ embed a short w-length band of random entries into a random location in each row. The remaining $n - w$ entries in the row will be zero. If $n = (1 - \eta)\ell$ for some constant $\eta > 0$, then random band matrices have full rank n and the associated linear system can be solved in time $O(nw)$ with high probability [32].

We consider a variant of the random band matrix where we study $n \times \ell$ matrix with more rows than columns, $n > \ell$. Each row is generated in the same way except that we now permit wrap-around in the row. For example, if the w-length band picks the last location in the row, the last $w - 1$ entries of the band will appear at the start of the row. Note, this satisfies the distributed generation property as each row is randomly sampled independently. For any subset of $(1+\alpha)\ell$ rows for some $\alpha > 0$, we show that the resulting sub-matrix has full column rank of ℓ except with negligible probability with $w = O(\lambda)$. Furthermore, we prove that Gaussian elimination can solve the associated linear system in time $O(\ell \cdot \lambda)$ that is corroborated by our experimental evaluation. In our experiments, we also show that $\alpha = 0.05$ is sufficient. Surprisingly, our modified random band matrices require a different set of analytical techniques compared to [32] and show our wrap-around modification is necessary.

We can use our modified random band matrix as a low-density (sparse) generator matrix to immediately obtain an erasure code that obtains negligible error. As the main benefit, our erasure code has distributed generation and its the decoding algorithm is more efficient than those in prior constructions. We note prior works can be viewed in similar lenses using the generator matrix. For example, Shamir's scheme uses Reed-Solomon codes with the Vandermonde matrix. Recent work by Applebaum *et al.* [2] suggest practical instantiations using low-density parity check (LDPC) matrices that enable efficient reconstruction through a peeling algorithm (such as [58]). A recurring theme is that random band matrices are more efficient than both Vandermonde and peeling-based matrices. For example, this phenomenon has been observed in oblivious key-value stores [43] where random band matrix solutions [9] outperform Vandermonde and peeling-based solutions [43, 67]. The same has occurred for filter data structures where ribbon filters using random band matrices [33] outperform cuckoo filters using peeling-based matrices [37]. Therefore, it is not surprising that our erasure code outperforms prior schemes. We present a comparison of practical erasure codes in Figure 1.

Secret Sharing Construction. Next, we combine our framework and linear erasure code to obtain a (t, c)-ramp secret sharing scheme, RB-SS, with $O(t\lambda)$ message reconstruction for $c = 1.05t$. Our experiments shows that reconstruction time in RB-SS is 2-7.8x faster than Shamir's scheme [70] that is still the most widely used secret sharing scheme. With $t = 2^{20}$, RB-SS requires 5 seconds while

	Decoding Time	Correctness Guarantee	Error	Distributed Generation
Reed-Solomon	$O(n \log^2 n)$	Strong	0	×
Luby [58]	$O(n)$	Weak	$2^{-\Theta(1)}$	×
Ours (RB)	$O(n(\lambda + \log n))$	Weak	$2^{-\lambda}$	✓

Figure 1: Comparison of practical linear erasure codes. Strong correctness means that there exists no choice of erasures that could cause decoding failures except with the error probability. Weak correctness states that, for any choice of erasures, decoding fails with the error probability.

	Total Time	Correctness Threshold	Error	Adaptive Corruption
Shamir [70]	$O(n \log^2 n)$	$c = t$	0	✓
ANP [2]	$O(n)$	$c = 1.22t$	$2^{-\Theta(1)}$	×
Ours (RB-SS)	$O(n(\lambda + \log n))$	$c = 1.05t$	$2^{-\lambda}$	✓

Figure 2: Practical secret sharing with near-linear time.

Shamir's scheme requires nearly half a minute. Therefore, RB-SS may be scaled towards much larger thresholds t and more parties n without message reconstruction becoming a bottleneck. Furthermore, we see improvements for t as small as 256. Even though we require a gap between the correctness and privacy thresholds of $c \geq 1.05t$, RB-SS still fits the requirements for many important applications including federated learning, Byzantine agreement and secure multi-party computation (MPC).

We also compare RB-SS to the practical instantiation of Applebaum *et al.* [2] that we denote as ANP. As mentioned earlier, RB-SS may be used against adversaries with adaptive corruptions while ANP may only be used in the setting of static corruptions. Furthermore, as acknowledged by the authors in [2], ANP only provides non-negligible error guarantees even in the static corruption case. In contrast, RB-SS provides provably negligible error even against adaptive corruptions. Even with the stronger guarantees, RB-SS is still more efficient than ANP as shown in our experimental evaluation in Section 6. Finally, we note that ANP utilize peeling-based codes that have very well known tight thresholds of success rates (see [46, 74] for example) that mean that $c \geq 1.22t$. RB-SS is more flexible and can be instantiated with much smaller correctness thresholds $c \geq 1.05t$. See Figure 2 for more comparisons.

We note that Applebaum *et al.* [2] present other constructions addressing these issues. In particular, they obtain negligible error against adaptive corruptions and arbitrarily small gaps between the privacy and correctness thresholds. To our knowledge, these constructions remain theoretical in nature without efficient instantiations in practice. Instead, we consider the practical constructions (denoted ANP) suggested by the authors (see Section 1.4 in [2]).

Applications. First, we show that replacing Shamir's scheme with RB-SS in the secure aggregation protocol of Bonawitz *et al.* [16] deployed at Google for federated learning [39]. As a result, we reduce time spent for secret sharing by 3.6-6.6x. This results in a decrease of 13%-22% in computation compared to [16].

Next, we show that RB-SS can be utilized to improve other primitives. First, we extend RB-SS with the techniques of either Feldman [40] or Pedersen [64] to obtain *verifiable* ramp secret sharing schemes (VSS). Specifically, we obtain VSS schemes with ϵ error probability, verification time $O(\log(1/\epsilon) + \log t)$, and public parameters of size $O(t)$. Our VSS schemes obtains 8.2-25.2x faster sharing and 2.7-23.2x faster reconstruction time compared to state-of-the-art works [21, 51, 73]. Furthermore, our Pedersen-based scheme obtains up to 9% smaller communication during dealing. As a caveat, our VSS schemes only provide correctness guarantees against non-adaptive adversaries. It turns out that privacy still holds even in the adaptive setting. We leave it as an open problem to enable adaptive correctness. See Figure 8 in Appendix C.4 for further comparisons.

Finally, we utilize our VSS to construct a distributed verifiable random function (DVRF) that, in turn, may be used to construct decentralized randomness beacons (DRB). As we build off our reconstruction-efficient VSS, our DRB schemes obtains 2-6x improvement for each randomness generation round compared to prior DVRF-based DRB schemes [42, 48].

2 DEFINITIONS

We will denote column vectors as $\mathbf{v}$ and row vectors as $\mathbf{v}^\intercal$. The i-th entry of $\mathbf{v}$ is denoted by $\mathbf{v}_i$. We will consider all primitives to be one-indexed. For example, we set $[n] = \{1, \ldots, n\}$. For two n-length vectors $\mathbf{v}, \mathbf{v}' \in \mathbb{F}^n$, we denote the dot product by $\mathbf{v} \cdot \mathbf{v}' = (\mathbf{v}_1 \cdot \mathbf{v}'_1) + \ldots + (\mathbf{v}_n \cdot \mathbf{v}'_n)$. For any $n \times t$ matrix $\mathbf{M} \in \mathbb{F}^{n \times t}$, we denote the i-th row of the matrix by $\mathbf{M}_i$ for any $i \in [n]$. The j-th element of $\mathbf{M}_i$ by $\mathbf{M}_{i,j}$ for any $j \in [t]$. We denote the matrix-vector multiplication of $\mathbf{M} \in \mathbb{F}^{n \times t}$ and $\mathbf{v} \in \mathbb{F}^t$ as $\mathbf{M} \cdot \mathbf{v} = [\mathbf{M}_1 \cdot \mathbf{v}, \ldots, \mathbf{M}_n \cdot \mathbf{v}]^\intercal$. For a matrix $\mathbf{M} \in \mathbb{F}^{n \times t}$ and a vector $\mathbf{y} \in \mathbb{F}^n$, we say we solve the linear system for $\mathbf{M}$ and $\mathbf{y}$ as finding a vector $\mathbf{x} \in \mathbb{F}^t$ such that $\mathbf{M} \cdot \mathbf{x} = \mathbf{y}$.

We model all hash functions $H : X \to Y$ as random oracles unless otherwise specified. Random oracles idealize hash functions such that each output is truly random. For any $x \in X$, $H(x)$ will be a uniformly random element from Y.

2.1 Linear Erasure Codes

An erasure code consists of a pair of algorithms (Encode, Decode) is defined over an alphabet Σ. The erasure code encodes an input of length γ, $x \in \Sigma^\gamma$, into a codeword of length η, $\mathsf{Encode}(x) \in \Sigma^\eta$. For any choice of z erasures in the codeword $\mathsf{Encode}(x)$, Decode will successfully recover the input x except with some probability p. Note, this is the weaker correctness guarantee. A stronger correctness guarantee would be that there does not exist any choice of z erasures of $\mathsf{Encode}(x)$ such that the input x cannot be recovered except with probability p. Throughout our work, we will only consider the weaker correctness guarantee.

All linear erasure codes can be viewed through a generator matrix $\mathbf{M}$ of dimension $\eta \times \gamma$. Then, $\mathsf{Encode}(x) = \mathbf{M}x$ is the matrix-vector multiplication of the generator matrix $\mathbf{M}$ and the input x. Consider any subset $S \subseteq [\gamma]$ of erasures. Let $\mathbf{M}_{-S}$ and $\mathsf{Encode}(x)_{-S}$ be the generator matrix and input with the corresponding erased codeword symbols removed. For $\mathbf{M}_{-S}$, these are the rows corresponding to the erased codeword symbols. Then, Decode is equivalent to solving the linear system $\mathbf{M}_{-S} \cdot x = \mathsf{Encode}(x)_{-S}$ for x. The input is successfully recovered as long as there is a unique solution to the linear system. That is, $\mathbf{M}_{-S}$ has full column rank of γ. One can view the correctness guarantee as follows. For any choice of S, $\mathbf{M}_{-S}$ must have rank γ except with probability p over the random

choice of the generator matrix $\mathbf{M}$. We will consider erasure codes through generator matrices for the remainder of the paper.

Finally, we will introduce the *distributed generation* property of linear erasure codes necessary for our secret sharing framework. In particular, we require that each row of the generator matrix $\mathbf{M}$ may be generated independently from every other row. We point readers to Section 3 for our formal definition.

2.2 Ramp Secret Sharing

We define ramp secret sharing (SS) schemes with computational privacy following [6]. A (t, c)-ramp secret sharing for message space $\mathcal{M}$ consists of a tuple of algorithms, (Init, Share, Reconstruct). First, Init produces public parameters pp used by all algorithms. The dealer splits the message $m \in \mathcal{M}$ as shares, $s_1, \ldots, s_n$, to n parties by executing $\mathsf{Share}(\mathsf{pp}, m)$. Afterwards, Reconstruct takes a subset of shares and aims to output the input message m. For subset $I \subset [n]$, we denote the subset of shares by $s_I = \{s_i\}_{i \in I}$.

A ramp secret sharing scheme against a malicious adversary that may adaptively corrupt parties must satisfy:

(1) **Privacy**: An adversary adaptively corrupting at most $t - 1$ parties cannot learn any information about the message.
(2) **Correctness**: An adversary that adaptively corrupts at most $n-c$ parties cannot cause the reconstruction algorithm to fail or output the wrong message when receiving any subset of c shares from the remaining honest parties.

In other words, no subset of $t-1$ parties can learn the input message but any subset of c honest parties can reconstruct the input message. For formal definitions, see Appendix A.1.

Linear Homomorphism. Linearity is an important property for many applications of secret sharing. We can consider the setting where multiple messages $m, m' \in \mathcal{M}$ are shared to n parties such that the i-th party receives shares s_i and s'_i. Roughly speaking, linearity ensures that each party can add their shares $s_i + s'_i$ to obtain a new share. These new shares may be reconstructed to the message $m + m'$ that is the sum of the two original messages. We will refer to such schemes as linear (ramp) secret sharing schemes.

2.3 Verifiable Secret Sharing

Ramp secret sharing assumes that the dealer computes the shares honestly and that all players reveal the share received from the dealer. In this section, we consider *verifiable* (t, c)-ramp secret sharing (*VSS*) that detects dishonest dealers and parties. We define VSS following prior works [51, 73] that consider malicious (Byzantine) adversaries that may corrupt the dealer as well as at most $t - 1$ parties. We also add a modification for ramp access structures for correctness. We assume all parties are connected with a broadcast channel and any two parties (including the dealer) are connected by a private synchronous channel.

A VSS scheme consists of the same tuple (Init, Share, Reconstruct). However, in VSS, sharing is implemented by a protocol consisting of multiple rounds. Our constructions will consider three round sharing consisting of a dealing, verification and complaint round identical to prior works [51, 73]. The general definition below accommodates any number of rounds. A (t, c)-VSS scheme against a malicious adversary must satisfy the following:

(1) **Privacy**: An adversary that adaptively corrupts at most $t - 1$ parties cannot learn anything about the message.
(2) **Correctness (Honest Dealer)**: An adversary that statically corrupts at most $n-c$ parties cannot cause any honest party to reconstruct the wrong message.
(3) **Correctness (Malicious Dealer)**: Every honest party will output the same message m' or $\perp$, even in the presence of an adversary that statically corrupts the dealer and at most $n - c$ parties.

Any subset of $t - 1$ cannot learn the message m. When the dealer is honest, every honest party should correctly reconstruct the input message m. When the dealer is malicious, every honest party will output a consistent message m' or detect the malicious dealer by returning $\perp$. For formal definitions, we point readers to [51].

Static Corruption Correctness. Our VSS schemes maintain privacy even against adaptive adversaries. However, correctness only holds with respect to static adversaries. In other words, an adaptive adversary can prevent honest parties from reconstructing the secret correctly. We leave it as an open problem to provide correctness guarantees against adaptive adversaries.

Other Properties. Prior works have also studied secret sharing with other properties beyond linearity and verifiability including robustness [29, 66] and integrity [49]. The main goal of our work is to improve computational costs that we show may be obtained for secret sharing satisfying privacy, correctness and even verifiabililty. We leave it as an open problem to obtain other properties.

3 LINEAR ERASURE CODES

In this section, we present our linear erasure code. First, we will present another definition of erasure codes through generator matrices that will be useful for our secret sharing framework. Afterwards, we present our construction using random band matrices.

3.1 Distributed Generator Matrix

Throughout the rest of the paper, we will consider linear erasure codes through the lens of their corresponding generator matrix (see Section 2.1 for details on the relationship). We will consider a definition of generator matrix families that enable *distributed generation*. A family of distributed generator matrices $\mathcal{F}$ consists of the tuple $\mathcal{F} = (\mathsf{RandGen}, \mathsf{ExpandRow}, \mathsf{Solve})$. The *random generation* algorithm, RandGen, will be responsible for sampling n uniformly random seeds, $\mathsf{seed}_1, \ldots, \mathsf{seed}_n$, that will be used to randomly generate a matrix $\mathbf{M} \in \mathbb{F}^{n \times \ell}$ with dimension $n \times \ell$ over a field $\mathbb{F}$. In particular, RandGen outputs $(\mathsf{seed}_1, \ldots, \mathsf{seed}_n)$ such that seed_i can be used by ExpandRow to generate the i-th row of $\mathbf{M}$. By our definition, each row of $\mathbf{M}$ is generated independently from all other rows. Roughly speaking, RandGen simply determines the length of each random seed generated.

The solve algorithm, Solve, will receive a sub-matrix of $\mathbf{M}$ consisting of a subset of rows whose indices are in subset $I \subseteq [n]$ as well as a vector $\mathbf{y}$ of length $|I|$. For $I = \{i_1, \ldots, i_{|I|}\}$, we define

$$\mathbf{M}_I = \begin{bmatrix} \mathbf{M}_{i_1} \\ \cdots \\ \mathbf{M}_{i_{|I|}} \end{bmatrix}.$$

The goal of Solve is to output a vector $\mathbf{x}$ such that $\mathbf{M}_I \cdot \mathbf{x} = \mathbf{y}$. In other words, Solve should solve the linear system associated with the sub-matrix $\mathbf{M}_I$. The goal for $\mathcal{F}$ is to generate random matrices $\mathbf{M}$ such that $\mathbf{M}_I$ has a unique solution and that solution may be efficiently found for sufficiently large enough subsets I. In other words, this allows the corresponding erasure code using $\mathcal{F}$ as its generator matrix to recover the input $\mathbf{x}$. We say $\mathcal{F}$ is a distributed (K, ϵ)-*generator matrix family* if the above is true except with probability ϵ for any subsets $|I| \geq K(\ell)$.

Definition 1. *Let $0 \leq \epsilon \leq 1$ and let $K(\cdot)$ be an integer function such that $K(\ell) \geq \ell$, for all $\ell \geq 1$. A (K, ϵ)-*distributed generator matrix family $\mathcal{F} = (\mathsf{RandGen}, \mathsf{ExpandRow}, \mathsf{Solve})$ *is defined as:*

- $(\mathsf{seed}_1, \ldots, \mathsf{seed}_n) \leftarrow \mathsf{RandGen}(1^\lambda, n, \ell)$ *receives as input the security parameter, the number of rows and the number of columns ℓ and outputs n uniformly random seeds $(\mathsf{seed}_1, \ldots, \mathsf{seed}_n)$ that succinctly encode the n row vectors of an $n \times \ell$ matrix $\mathbf{M} \in \mathbb{F}^{n \times \ell}$.*
- $\mathbf{M}_i \leftarrow \mathsf{ExpandRow}(\mathsf{seed}_i)$ *receives seed_i and outputs the i-th row vector of the matrix, $\mathbf{M}_i \in \mathbb{F}^\ell$.*
- $\mathbf{x} \leftarrow \mathsf{Solve}(\mathbf{M}_I, \mathbf{y}_I)$ *receives a sub-matrix $\mathbf{M}_I \in \mathbb{F}^{|I| \times \ell}$ of $|I|$ rows with indices in $I \subseteq [n]$ and the vector $\mathbf{y}_I \in \mathbb{F}^{|I|}$ of length $|I|$.* Solve *outputs a vector $\mathbf{x}$ of length ℓ such that $\mathbf{M}_I \cdot \mathbf{x} = \mathbf{y}_I$.*

$\mathcal{F}$ must satisfy the following correctness *requirement. For every subset $I \subseteq [n]$ such that $|I| \geq K(\ell)$,*

$$\Pr[\mathbf{M}_I \text{ has rank } \ell \mid (\mathsf{seed}_1, \ldots, \mathsf{seed}_n) \leftarrow \mathsf{RandGen}(1^\lambda, n, \ell)] \geq 1 - \epsilon$$

where $\mathbf{M}_i \leftarrow \mathsf{ExpandRow}(\mathsf{seed}_i)$ for all $i \in I$. The probability is taken over the random coin tosses of RandGen.

Note that since $\mathbf{M}_I$ has $|I| \geq K(\ell) \geq \ell$ rows and ℓ columns, $\mathbf{M}_I$ having rank ℓ means it has full column rank. Therefore, for every vector $\mathbf{y}_I$ of size $|I|$, the linear system $\mathbf{M}_I \cdot \mathbf{x} = \mathbf{y}_I$ has at most one solution except with probability ϵ. In other words, this implies that each $\mathbf{y}_I$ will result in the same $\mathbf{x}$ except with probability ϵ. Going back to erasure codes, the Solve algorithm is used to try and decode the original input. We critically use the fact that $\mathbf{M}_I$ has full column rank to ensure a unique solution $\mathbf{x}$ for each $\mathbf{y}_I$.

Finally, we did not formally introduce the notion of the efficiency in the above definition of (K, ϵ)-generator matrix families. However, an important goal of a matrix family $\mathcal{F}$ is to ensure that all algorithms are efficient. In most cases, this is more difficult with respect to Solve that needs to solve the linear system associated to $\mathbf{M}_I$. One universal way to do this is to utilize Gaussian elimination to solve the $|I| \times \ell$ matrix that would require $O(|I| \cdot \ell^2)$ time. For our construction, we will end up using matrix families equipped with Solve algorithms that find solutions much faster.

Strong vs. Weak Correctness. We revisit our prior notions of strong and weak correctness formally. For strong correctness, we mean that there exists no choice of erasures that could cause decoding failures except with some probability ϵ. In contrast, weak correctness guarantees that, if one chooses corruptions independent of the matrix, decoding failures occur with some small probability. We define erasure codes in Definition 1 with weak correctness.

Later, we show that weak correctness still suffices for adaptive correctness in secret sharing by relying on the distributed generation property. One major benefit of relying only on weak correctness is that we obtain more efficient constructions.

Algorithm 1 RB.RandGen algorithm

Input: 1^λ: security parameter
Output: $\mathsf{seed}_1, \ldots, \mathsf{seed}_n$: seeds to generate n rows
  for $i \in [n]$ **do**
    Randomly sample $\mathsf{seed}_i \leftarrow \{0,1\}^\lambda$
  return $(\mathsf{seed}_1, \ldots, \mathsf{seed}_n)$

Algorithm 2 RB.ExpandRow algorithm

Input: seed: seed to generate row
Output: $\mathbf{v}$: row vector of matrix
  Compute starting location $x \leftarrow H_1(\mathsf{seed})$ with $x \in [\ell]$.
  Compute w-bit band $\mathbf{b} \leftarrow H_2(\mathsf{seed})$ with $\mathbf{b} \in \{0,1\}^w$.
  Initialize $\mathbf{v} = [0]^t$.
  for $i \in [w]$ **do**:
    Set $\mathbf{v}_{x+i-1} \leftarrow \mathbf{b}_i$. ▷ $x + i - 1$ computed mod ℓ.
  return $\mathbf{v}$

Algorithm 3 RB.Solve algorithm

Input: $\mathbf{M}, \mathbf{y}$: matrix and vector
Output: $\mathbf{x}$: unique solution if it exists
  Sort rows of $\mathbf{M}$ by starting location of random band.
  Execute Gaussian elimination on sorted $\mathbf{M}$.
  if $\mathbf{M}$ has full column rank **then**
    Compute $\mathbf{x}$ such that $\mathbf{M} \cdot \mathbf{x} = \mathbf{y}$.
    return $\mathbf{x}$
  return $\perp$

Discussion about Distributed Generation. At a high level, the distributed generation property means that each of the n row vectors may be sampled independently. We note that knowledge of any subset of rows of $\mathbf{M}$ does not reveal any information about the remaining rows. This property will be critical to obtain secret sharing for adaptive corruptions.

3.2 Our Construction

We present a distributed generator matrix (and, thus, linear erasure code with distributed generation) using random band matrices.

Original Random Band Matrix Family. To construct our generator matrix family (and, thus, linear erasure code), we adapt the *random band matrix* family introduced by Dietzfelbinger and Walzer [32] that we will denote by RB. The family of $n \times \ell$ matrices in RB is also parameterized by a band length that we denote by w.

The family of random band matrices is generated in the following way. Each of the n rows is generated by embedding exactly one consecutive w-length random binary band at a random starting location. More formally, suppose we wish to generate the i-th row vector $\mathbf{M}_i$ of a random band matrix $\mathbf{M}$ and let H_1 and H_2 be two random functions. For $\mathbf{M}_i$, we first pick a random column denoting the start of the band is picked uniformly at random from the set $x_i \in [\ell - w]$. Additionally, a random w-bit band is generated uniformly at random from $\mathbf{b}_i \in \{0,1\}^w$. Then, the first entry of $\mathbf{b}_i$ becomes the x_i-th entry of $\mathbf{M}_i$, the second entry of $\mathbf{b}_i$ becomes the $(x_i + 1)$-th entry of $\mathbf{M}_i$ and so forth. All $\ell - w$ columns outside of the random

band will be zero. Therefore, the number of non-zero entries is at most $|\mathbf{M}|_0 \leq n \cdot w$ as each row vector has at most w non-zero entries. The algorithms of RB are formally described with pseudocode in Algorithms 1, 2 and 3. We use two random functions H_1 and H_2 that take as input a λ-bit seed and output the starting location $x \in [\ell - w]$ and a w-bit random band, respectively.

Next, we describe the algorithm used to solve the linear system associated with sub-matrices of random band matrices. Consider any random band matrix $\mathbf{M}$ and any subset $I \subseteq [n]$ of rows. The input to Solve is the $|I| \times \ell$ sub-matrix as well as some solution vector $\mathbf{y} \in \mathbb{F}^{|I|}$. We will rely on the fact that, for any random band matrix $\mathbf{M}$, any sub-matrix consisting of subsets of rows $\mathbf{M}_I$ is also a random band matrix. Therefore, we can use the algorithms described by Dietzfelbinger and Walzer [32] to try and solve the linear system associated to $\mathbf{M}_I$. The first step is to sort the rows of $\mathbf{M}_I$ according to the starting band location. Afterwards, standard Gaussian elimination can be executed in an attempt to reduce $\mathbf{M}_I$ to row echelon form. However, the structure of random band matrices ensure that Gaussian elimination runs much faster. In each row reduction step, the number of non-zero entries that need to be considered will be either or nearby the w-bit band. In other words, each of the row reduction operations will require $O(w)$ time on average meaning the entire Gaussian elimination algorithm requires $O(|I| \cdot w)$ time that is faster than the time of Gaussian elimination on arbitrary $|I| \times \ell$ matrices that requires $O(|I|^2 \cdot \ell)$.

Our Modified Random Band Matrix Family. Dietzfelbinger and Walzer [32] analyzed random band matrices of dimension $(1 - \alpha)\ell \times \ell$ with more columns than rows. Their work showed that such random band matrices are expected to have full row rank with high probability. For our work, we require matrices of dimension $|I| \times \ell$ where $|I| \geq \ell$. While these differences seem subtle, it turns out that we must modify the algorithms and provide new proof techniques to use random band matrices with more rows than columns.

First, we modify the sampling of random bands in $\mathbf{M}$. Previously, the starting column of a random w-length band for each row vector, $\mathbf{M}_i$, was chosen from $[\ell - w]$. In our work, we modify this to choose the starting column from $[\ell]$ with wrap-around. Suppose the starting location was i such that $i + w > \ell$. Then, the remaining $w - (\ell - i)$ entries of the w-length band will be placed starting from the first column again. The algorithm used to solve the linear system remains the same: first sort rows by starting location and then execute Gaussian elimination. We will later show that this wrap-around modification is necessary (see Appendix B). We present our pseudocode in Algorithms 1, 2 and 3.

Next, we also need to show that $\mathbf{M}_I$ has full column rank to enable recovery of the unique solution $\mathbf{x}$. We show that, for any subset $|I| \geq (1 + \alpha)\ell$ for some constant $\alpha > 0$, $\mathbf{M}_I$ has full column rank except with probability with $2^{-\kappa}$ if we set the band length to be $w = O(\kappa + \log \ell)$. Furthermore, we prove that the running time of Solve is $O(\ell \cdot w)$ for any subset $|I| = (1+\alpha)\ell$ except with probability $O(2^{-\kappa})$. Our above modification is essential to ensure $(1 + \alpha)\ell \times \ell$ random band matrices have full column rank except with negligible probability. Otherwise, it turns out that it is highly unlikely that the first few columns will be used as a pivot during Gaussian elimination (see the full version for further details). Furthermore, in our experiments, we show that solving the linear system $\mathbf{M}_I$ (decoding) is much faster than other known linear erasure codes (see Section 6).

To our knowledge, the above properties were not previously known. Note that Dietzfelbinger and Walzer [32], instead, proved properties for $(1-\alpha)\ell \times \ell$ random band matrices that we were unable to directly relate to our setting of $(1+\alpha)\ell \times \ell$ random band matrices. For our adaptation, we have to resort to new proof techniques. We point readers to Appendix B for the proof.

Theorem 1. *The $n \times \ell$ random band matrix family* RB *with w-bit band length satisfies the following properties:*

- RandGen *requires $O(n)$ time.*
- ExpandRow *requires $O(w)$ time.*
- *Every* $\mathbf{M} \in$ RB *has at most $n \cdot w$ non-zero entries.*
- *If $w = O(\kappa + \log \ell)$ and $K(\ell) = (1 + \alpha)\ell$ for some constant $\alpha > 0$, then* RB *is a $(K, 2^{-\kappa})$-generator matrix family. Furthermore,* Solve *requires $O(\ell \cdot w)$ time.*

Note, the above obtains a near-optimal erasure code rate requiring $(1+\alpha)\ell$ codewords to recover the original symbol. Furthermore, it obtains negligible error for $w = O(\kappa + \log \ell)$. Nevertheless, RB emits faster algorithms than the erasure codes used by Applebaum *et al.* [2] that only obtain non-negligible error.

Comparison with Other Erasure Codes. We note that our linear erasure code was built specifically with the requirements of our secret sharing framework. In particular, we desired distributed generation, a very fast decoding algorithm and negligible error probability. To our knowledge, prior erasure codes used in secret sharing do not satisfy the distributed generation property.

Reed-Solomon codes [68] used in Shamir's scheme [70] do not satisfy this property as rows are highly correlated. However, they can be used to build secret sharing tolerating adaptive corruption due to their strong correctness guarantees.

In the peeling-based codes used in practical instantiations of ANP [2], the matrices where each row has 3 non-zero entries and each column has 6 non-zero entries. As the rows are correlated (by the column requirement), it does not satisfy the distributed generation property. One could also try to extend ANP using a different peeling-based code with a distributed generation property. For example, one could consider slightly modified matrices with 3 non-zero entries in each row and no restrictions per column. Even if such an approach obtains the distributed generation property, we note it still has three drawbacks. First, the practical instantiations of these codes still have non-negligible error. This translates to adaptive correctness guarantees but with non-negligible error in the secret sharing scheme. Secondly, this approach still require larger gaps between correctness and privacy of $c \geq 1.22t$ that is inherent in any peeling-based code (see [46, 74]). In contrast, random band matrices obtain $c = 1.05t$. Finally, these peeling-based codes require more computation than random band matrices. Even when compared with the best peeling-based code without distributed generation, random band matrices are more computationally efficient (see Section 6.1 for experimental evaluation). We expect this to remain true when using other variants of peeling-based codes that may achieve distributed generation. To our knowledge, we are unaware of any techniques for extending ANP with peeling-based codes to obtain similar guarantees achieved by random band matrices of distributed generation, negligible error and efficiency.

Algorithm 4 $\mathcal{F}$-SS.Init algorithm

Input: $1^\lambda, n, t$: security parameter, number of parties and privacy threshold.
Output: pp: public parameters.
Sample random function $H : \mathbb{F}^t \to \mathcal{M}$.
return $\mathsf{pp} = (1^\lambda, n, c \leftarrow K(t), t, H)$.

4 SECRET SHARING

We present our framework for building ramp secret sharing using linear erasure codes. As a reminder, we build off the frameworks of Cramer *et al.* [29] and Applebaum *et al.* [2]. However, our framework is stronger as it may be used in the setting of adaptive corruptions. We also show that our framework may be instantiated to maintain the linear homomorphism of the prior frameworks. Afterwards, we instantiate our framework with our generator matrix family from Section 3.2 to obtain our secret sharing scheme.

4.1 Our Improved Framework

In this section, we construct secret sharing schemes from any (K, ϵ)-distributed generator matrix family $\mathcal{F}$. We denote the resulting secret sharing scheme as $\mathcal{F}$-SS. The formal description of $\mathcal{F}$-SS using $\mathcal{F}$ in a blackbox manner is presented in Algorithms 4, 5 and 6.

Let $\mathcal{F}$ be a (K, ϵ)-generator matrix family. The Init algorithm of the ramp secret sharing associated with $\mathcal{F}$ receives the security parameter λ, the total number of parties n and the privacy threshold $t \leq n$. The correctness threshold is set equal to $c = K(t)$. For security parameter λ, we assume that the message space $\mathcal{M}$ is a field of size $|\mathcal{M}| = 2^\lambda$ (see later discussion for handling large messages). All shares will contain an element from a field $\mathbb{F}$ of size $|\mathbb{F}| = 2^{O(\lambda)}$ where we will pick the field size later. Finally, we use a random hash function H mapping t field elements, $\mathbb{F}^t$, to the message space, $\mathcal{M}$.

Our framework enables various options for the hash function H. One option for H is any linear randomness extractor including those built from universal hash functions [29] or the dot-product with a small integer vector [2]. When H is a linear randomness extractor (see Definition 4 for more details), the resulting secret sharing scheme is linear. We also provide a third option later where H could instead be a random oracle. This slightly increases efficiency, but will lose the linearity feature. Surprisingly, linearity is not necessary for some important applications such as federated learning [16] (see Section 5.3). For the remainder of this section, we assume that the hash function H is a linear randomness extractor.

Initialization of Public Parameters. The initialization algorithm Init for $\mathcal{F}$-SS will sample a random hash function $H : \mathbb{F}^t \to \mathcal{M}$ that will be assumed to be publicly available to the dealer and all n parties. The public parameters also contain $1^\lambda, n, c$ and t. Note that t can be chosen arbitrarily and the resulting correctness threshold must satisfy $c \geq K(t)$, which is a function of t and the underlying (K, ϵ)-generator matrix family $\mathcal{F}$.

Share Construction. The goal of the Share algorithm is to construct n shares $s_1, \ldots, s_n$ for input message m. Share picks a matrix from family $\mathcal{F}$ with dimension $n \times t$.

Algorithm 5 $\mathcal{F}$-SS.Share algorithm

Input: m, pp: input message and public parameters.
Output: $(s_1, \ldots, s_n)$: n shares.
Parse $\mathsf{pp} = (1^\lambda, n, c, t, H)$.
Sample random vector $\mathbf{x} \leftarrow \mathbb{F}^t$
Compute $\mathsf{mask} \leftarrow m + H(\mathbf{x})$.
Compute $(\mathsf{seed}_1, \ldots, \mathsf{seed}_n) \leftarrow \mathcal{F}.\mathsf{RandGen}(1^\lambda)$.
for $i \in [n]$ **do**
  Compute $\mathbf{M}_i \leftarrow \mathcal{F}.\mathsf{ExpandRow}(\mathsf{seed}_i)$.
  Compute $\mathbf{y}_i \leftarrow \mathbf{M}_i \cdot \mathbf{x}$.
  Set $s_i \leftarrow (\mathsf{mask}, \mathsf{seed}_i, \mathbf{y}_i)$.
return $(s_1, \ldots, s_n)$

Algorithm 6 $\mathcal{F}$-SS.Reconstruct algorithm

Input: $(s_i)_{i \in I}$, pp: subset of shares and public parameters.
Output: m: reconstructed message.
Parse $\mathsf{pp} = (1^\lambda, n, c, t, H)$.
for $i \in I$ **do**
  Parse $s_i = (\mathsf{mask}, \mathsf{seed}_i, \mathbf{y}_i)$.
  Compute $\mathbf{M}_i \leftarrow \mathcal{F}.\mathsf{ExpandRow}(\mathsf{seed}_i)$.
Compute $\mathbf{x} \leftarrow \mathcal{F}.\mathsf{Solve}(\mathbf{M}_I, \mathbf{y}_I)$.
if $\mathbf{x} = \perp$ **then**
  return $\perp$
Compute $m \leftarrow \mathsf{mask} - H(\mathbf{x})$.
return m

First, the dealer generates a random vector $\mathbf{x} \in \mathbb{F}^t$ such that each $\mathbf{x}_i$ is a uniformly random field element from $\mathbb{F}$. Using $\mathbf{x}$, the dealer will construct a masked message by computing $\mathsf{mask} = m + H(\mathbf{x})$. That is, mask is the sum in the field $\mathcal{M}$ of the input message m and the output of the hash function H on input $\mathbf{x}$, $H(\mathbf{x})$. Recall that we have chosen $H : \mathbb{F}^t \to \mathcal{M}$ in Init. Next, the dealer executes the RandGen algorithm of $\mathcal{F}$ to produce n seeds, $(\mathsf{seed}_1, \ldots, \mathsf{seed}_n)$. Afterwards, the dealer constructs the associated random matrix $\mathbf{M} \in \mathbb{F}^{n \times t}$ by computing the i-th row $\mathbf{M}_i \leftarrow \mathsf{ExpandRow}(\mathsf{seed}_i)$, for $i \in [n]$. Note, the number of rows is equal to the number of parties n and the number of columns is at least the privacy threshold t. Finally, the dealer generates shares in the following. The dealer computes the matrix-vector multiplication $\mathbf{y} = \mathbf{M} \cdot \mathbf{x}$ where $\mathbf{y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n] \in \mathbb{F}^n$. For all $i \in [n]$, the share for the i-th party will be $s_i = (\mathsf{mask}, \mathsf{seed}_i, \mathbf{y}_i)$. Critically, the dealer will not publish the matrix $\mathbf{M}$. Instead, each of the n parties will receive one row of the matrix through seed_i. As a result, an adversary with adaptive corruption parties only learns the matrix $\mathbf{M}$ after compromising parties. As each row is independent of others, the adversary cannot learn about any row of $\mathbf{M}$ without compromising the party with the corresponding seed. As a result, we can leverage this property against adaptive corruptions.

The masked message mask is the same value across all n parties. With a slight abuse, mask may be viewed as a public parameter and simply appended to pp. This would reduce each share size to only contain seed_i and the field element $\mathbf{y}_i \in \mathbb{F}$.

Message Reconstruction. Finally, we present the message reconstruction algorithm Reconstruct that receives the shares $\{s_i\}_{i \in I}$ for

some subset of parties denoted by $I = \{i_1, \ldots, i_{|I|}\} \subseteq [n]$. First, Reconstruct executes ExpandRow on input the seed_{i_j} found in share s_{i_j}, for $j \in [|I|]$, to obtain the sub-matrix $\mathbf{M}_I$ of row vectors with indices in I. So, Reconstruct receives the masked message mask, the sub-matrix $\mathbf{M}_I$ and the vector $\mathbf{y}_I = [\mathbf{y}_{i_1}, \ldots, \mathbf{y}_{i_{|I|}}]^\intercal$ consisting of entries of $\mathbf{y}$ whose indices also appear in I.

Reconstruct will solve the linear system to compute the vector $\mathbf{x}$ satisfying: $\mathbf{M}_I \cdot \mathbf{x} = \mathbf{y}_I$. To do this, we execute the Solve algorithm of $\mathcal{F}$ to get $\mathbf{x} \leftarrow \mathsf{Solve}(\mathbf{M}_I, \mathbf{y}_I)$. If $\mathcal{F}$ is a distributed (K, ϵ)-generator matrix family, then we know that $\mathbf{x}$ satisfies $\mathbf{M}_I \cdot \mathbf{x} = \mathbf{y}_I$ and $\mathbf{x}$ is the unique solution with probability at least $1 - \epsilon$ as long as $|I| \geq K(t)$. Therefore, we set the correctness threshold as $c \geq K(t)$. To recover the message, we compute $H(\mathbf{x})$ and retrieve $m = \mathsf{mask} - H(\mathbf{x})$. In practice, we instantiate H using a hash function (such as SHA256) and convert $\mathbf{x} \in \mathbb{F}^t$ into a string of $t \cdot \lambda$ bits.

Correctness. For correctness, we will first assume that the underlying matrix family $\mathcal{F}$ is a (K, ϵ)-generator matrix family. Then, we note that the only requirement for correct share reconstruction is that the correct random vector $\mathbf{x} \in \mathbb{F}^t$ that was uniformly generated in Share is retrieved in Reconstruct. We critically utilize the fact that an adversary that adaptively corrupts parties only learns the matrix rows corresponding compromised parties. In particular, the adversary cannot learn any information about rows of honest parties. Even with an adversary that adaptively corrupts parties, we show that Reconstruct still successfully reconstructs the secret except with ϵ probability.

Theorem 2. *If $\mathcal{F}$ is a distributed (K, ϵ)-generator matrix family, then $\mathcal{F}$-SS is an (t, c)-ramp secret sharing against adaptive adversaries with at most ϵ error for any correctness threshold $c \geq K(t)$.*

Proof. In Reconstruct, share reconstruction is correct as long as $\mathsf{Solve}(\mathbf{M}_I, \mathbf{y}_I)$ outputs the same vector $\mathbf{x}$ that was randomly generated in Share. As $\mathcal{F}$ is a (K, ϵ)-generator matrix, we know that, for any subset I, $\Pr[\mathbf{x} = \mathsf{Solve}(\mathbf{M}_I, \mathbf{y}_I)] \geq 1 - \epsilon$. However, this does eliminate the fact that there may exist some subset I such that $\mathbf{x} \neq \mathsf{Solve}(\mathbf{M}_I, \mathbf{y}_I)$. In fact, it is possible such a subset I may be efficiently computable if given the matrix $\mathbf{M}$.

Instead, we utilize the fact that our construction reveals each row vector $\mathbf{M}_i$ only to the i-th party. Consider the correctness experiment for secret sharing (see Appendix A.2). We consider a hybrid experiment with the following modification. The challenger will delay sampling the matrix $\mathbf{M}$. The challenger will still sample $\mathbf{x}$ and compute $\mathsf{mask} \leftarrow m + H(\mathbf{x})$. Instead, it will sample each row as the adversary chooses to corrupt parties. For example, if the adversary compromises the i-th party, the challenge generates seed_i to compute $\mathbf{M}_i \leftarrow \mathsf{ExpandRow}(\mathsf{seed}_i)$ and $\mathbf{y}_i \leftarrow \mathbf{M}_i \cdot \mathbf{y}_i$. Finally, the challenge will reveal $(\mathsf{mask}, \mathsf{seed}_i, \mathbf{y}_i)$ to the adversary. We note that the games are indistinguishable from the view of the adversary. Clearly, mask is independent of $\mathbf{M}$ and seed_i is a uniformly random string. We note that $\mathbf{y}_i$ is essentially a linear combination with some random vector chosen by $\mathsf{ExpandRow}(\mathsf{seed}_i)$ as $\mathcal{F}$ since $\mathcal{F}$ is a distributed generator matrix and each row is independently generated. Therefore, the experiments are indistinguishable.

We consider a final hybrid experiment where the challenger samples initially samples n seeds: $\mathsf{seed}_1, \ldots, \mathsf{seed}_n$. We will consider the subset $I = \{K(t) + 1, \ldots, n\}$. The challenge also computes $\mathsf{mask} = m + H(\mathbf{x})$. As the adversary adaptively corrupts, we will simply reveal seeds in sequence (independent of the corrupted party's index) and compute the corresponding share as before. The remaining seeds $\mathsf{seed}_{K(t)+1}, \ldots, \mathsf{seed}_n$ will be used to generate the shares for the honest parties. As all n seeds are uniformly random and independent strings, we can see that this experiment is indistinguishable from the prior experiment as the only difference is that the challenger sampled the seeds ahead of time. Finally, we note that this experiment returns 1 if and only if $\mathbf{x} = \mathsf{Solve}(\mathbf{M}_I, \mathbf{y}_I)$ for the fixed subset $I = \{K(t) + 1, \ldots, n\}$. As $\mathcal{F}$ is a distributed (K, ϵ)-generator matrix, for the fixed subset $I = \{K(t) + 1, \ldots, n\}$, we know that $\Pr[\mathbf{x} \neq \mathsf{Solve}(\mathbf{M}_I, \mathbf{y}_I)] \leq \epsilon$ completing the proof. □

Efficiency. To analyze efficiency, we consider several properties of the matrix family $\mathcal{F} = (\mathsf{RandGen}, \mathsf{ExpandRow}, \mathsf{Solve})$. First, we will assume the number of non-zero entries of any matrices that are potential outputs of RandGen. That is, any possible matrix output $\mathbf{M}$ of RandGen will contain at most z non-zero entries, $|\mathbf{M}|_0 \leq a$. As $\mathcal{F}$ consists of $n \times t$ matrices, we can always set $a = n \cdot t$ as a trivial upper bound on the norm. However, in our concrete instantiations, we will consider explicit families $\mathcal{F}$ where a is much smaller, $a \ll n \cdot t$. We also assume that the RandGen, ExpandRow and Solve algorithms run in time $f_{\mathsf{RandGen}}(n, t)$, $f_{\mathsf{ExpandRow}}(t)$ and $f_{\mathsf{Solve}}(t)$ respectively. Then, we can upper bound the running times of $\mathcal{F}$-SS as follows:

Theorem 3. *Suppose $\mathcal{F} = (\mathsf{RandGen}, \mathsf{ExpandRow}, \mathsf{Solve})$ is a (K, ϵ)-generator matrix family such that* RandGen *outputs matrices with at most a non-zero entries and* RandGen, ExpandRow *and* Solve *run in time $f_{\mathsf{RandGen}}(n, t)$, $f_{\mathsf{ExpandRow}}(t)$ and $f_{\mathsf{Solve}}(t)$ respectively. Then, we get:*

- Init: $O(\lambda)$.
- Share: $O(f_{\mathsf{RandGen}}(n, t) + n \cdot f_{\mathsf{ExpandRow}}(t) + a + t)$.
- Reconstruct: $O(t \cdot f_{\mathsf{ExpandRow}}(t) + f_{\mathsf{Solve}}(t) + t)$.

Privacy. We show that any adversary $\mathcal{A}$ running in time $\mathsf{poly}(\lambda, n)$ and adaptively corrupt at most $t-1$ parties cannot learn any information about the input message except with negligible probability for sufficiently large fields $\mathbb{F}$ and $\mathcal{M}$. Roughly speaking, privacy follows by showing the output $H(\mathbf{x})$ is shown to be a highly unpredictable random string from the view of the adversary. This is similar to prior works [2, 29] where they assumed H is a randomness extractor. In particular, we assume that H is $(\log|\mathbb{F}|, 2^{-\lambda})$-randomness extractor that takes an input $\mathbf{x}$ with min-entropy $H_\infty(\mathbf{x}) \geq \log|\mathbb{F}|$. The output $H(\mathbf{x})$ has statistical distance at most $\delta \leq 2^{-\lambda}$ from a uniformly random element in $\mathcal{M}$. We present formal proofs in Appendix A.2 as well as formal definitions of extractors.

Theorem 4. *Suppose H is a $(\log|\mathbb{F}|, 2^{-\lambda})$-randomness extractor. For any $c \geq K(t)$, $\mathcal{F}$-SS is a (t, c)-ramp secret sharing scheme that is δ-secure with $\delta \leq 2^{-\lambda}$ against adaptive adversaries when $|\mathcal{M}| \geq 2^\lambda$ and $|\mathbb{F}| = |\mathcal{M}| \cdot 2^{2\lambda}$.*

We also show that it suffices for H to be a random oracle where the field size $|\mathbb{F}|$ may be smaller where $|\mathbb{F}| = |\mathcal{M}|$.

Theorem 5. *Suppose H is a random oracle. For any $c \geq K(t)$, $\mathcal{F}$-SS is a (t, c)-ramp secret sharing scheme that is δ-secure with $\delta \leq 2^{-\lambda(1-o(1))}$ against adaptive adversaries when $|\mathbb{F}| = |\mathcal{M}| \geq 2^\lambda$.*

We note that $\mathcal{F}$-SS inherits the gap between the privacy t and correctness threshold c from the rate of the underlying erasure code. If $K(t) = t$, then we can obtain a threshold scheme without any gap. When $K(t) = (1+\alpha)t$ for near-optimal rate erasure codes, the gap becomes $c \geq (1+\alpha)t$.

Linearity. We show that our framework $\mathcal{F}$-SS results in a linear ramp secret sharing scheme assuming that the underlying hash function H is linear. In other words, we will rely on the fact that $H(\mathbf{x}) + H(\mathbf{x}') = H(\mathbf{x} + \mathbf{x}')$. This follows similar from prior frameworks [2, 29] using the observation that the same generator matrix corresponding may be used to share multiple secrets. Furthermore, both the share and reconstruction algorithms are linear. Consider sharing any two messages $m, m' \in \mathcal{M}$ using the same generator matrix $\mathbf{M}$. Note, this can be achieved by having each party use the same seed_i to generate their corresponding row in $\mathbf{M}$. The shares of the i-th party will be $s_i = (\mathsf{mask}, \mathsf{seed}_i, \mathbf{y}_i)$ and $s'_i = (\mathsf{mask}', \mathsf{seed}_i, \mathbf{y}'_i)$ where the two seeds are the same. Then, we can compute sum of shares as $t_i = (\mathsf{mask} + \mathsf{mask}', \mathsf{seed}_i, \mathbf{y}_i + \mathbf{y}'_i)$. That is, the sum of the first and third components of the shares (the seed remains the same). If one attempts to run Reconstruct on t_I for a sufficiently large subset $I \subseteq [n]$, then the result would be the sum $m + m'$.

Theorem 6. *If H is a linear function, then $\mathcal{F}$-SS satisfies linearity.*

Some options for a linear randomness extractor H are a linear universal hash function [29] or the dot-product with a small integer vector [2] from prior works.

Non-Linear Hash Function. For privacy, we only require that H is a randomness extractor. The linearity is only critical if we wish for a linear ramp secret sharing scheme. If one does not require linearity, we note that H can be replaced with any non-linear hash function such as SHA256 (modelled as a random oracle). This results in a slightly more efficient construction at the cost of the linear homomorphism property (see Theorem 5). In Section 5.3, we present applications that do not require linearity.

Different Message Sizes. In $\mathcal{F}$-SS, the message space is size $|\mathcal{M}| = 2^\lambda$. One could consider larger messages with more than λ bits. We note that we can adapt the technique of Krawczyk [52]. Instead of sharing the message directly, we derive an encryption key K of λ bits that we share using $\mathcal{F}$-SS. Each party is provided an encryption of the message m under key K. To retrieve the message, the parties first reconstruct the encryption key K using $\mathcal{F}$-SS to decrypt the message m. $\mathcal{F}$-SS still only shares encryption keys of λ bits.

We note the shares in $\mathcal{F}$-SS still require λ bits for smaller messages (such as 1-bit messages). We leave it as an open problem to improve share sizes in $\mathcal{F}$-SS for smaller messages. Although, this phenomenon appears in prior works such as Shamir's schemes that also requires shares of size $\log n$ bits for single-bit messages.

4.2 Our Construction: RB-SS

Finally, we will instantiate our final ramp secret sharing construction using the framework $\mathcal{F}$-SS from Section 4.1 with our linear erasure code $\mathcal{F} = \mathsf{RB}$ in Section 3.2. Plugging in the properties of RB into our framework, we obtain the following construction:

Theorem 7. *There exists a constant $\alpha > 0$ such that for any $c \geq (1+\alpha)t$ and band length $w = O(\kappa + \log t)$, RB-SS is a (t, c)-ramp linear secret sharing with $2^{-\kappa}$ error and $\mathsf{negl}(\lambda)$-security in the random oracle model.* Init *runs in time $O(\lambda)$,* Share *runs in time $O(nw)$ and* Reconstruct *runs in time $O(tw)$.*

Choosing α and w. For our construction, we will need to choose concrete values of constant $\alpha > 0$ as well as the band length $w = O(\kappa + \log t)$. In Appendix B.1, we use experimental evaluation to pick α and w in an attempt to minimize $\alpha = 0.05$ while ensuring error probability at most 2^{-40}. For $\alpha = 0.05$, our empirical analysis shows that it is sufficient to choose w using the equation: $w(t, \kappa) = (0.9282 \log_2(t) + \kappa - 6.867)/0.1325$. For $t = 2^{10}$, we may set $w = 150$ and for $t = 2^{22}$, we can use $w = 400$.

We picked $\alpha = 0.05$ through experimentation and balancing several factors. One could choose any α that would result in different band lengths w. Larger α would result in faster algorithms (smaller bands), but larger correctness threshold $c = (1+\alpha)t$. In contrast, smaller α results in smaller correctness threshold, but slower algorithms (larger bands). We chose $\alpha = 0.05$ as a good balance of being both faster than prior work while having more flexible correctness thresholds. This means that RB-SS has smaller correctness thresholds of $c = 1.05t$ as opposed to $c \geq 1.22t$ in ANP [2].

Practical Running Time. The above running time assumes that each entry of the random band matrix is processed independently. In practice, we rely on SIMD operations and pack multiple entries into a single word (integer). In particular, an entire band of length $w = O(\kappa + \log t)$ can fit into $O(1)$ words in practice. For example, we use band lengths of $w = 150$ and $w = 400$ that fits into a few registers in standard modern architectures (where registers are at least 64 or 128 bits in size). Similarly, we can also pack multiple shares into the a single register to process simultaneously. This means the total running time of Share and Reconstruct will be $O(n)$ and $O(t)$ word operations respectively.

Runtime Comparison with ANP [2]. Recall that the reconstruction time of ANP [2] is $O(n)$. In contrast, RB-SS requires $O(n(\kappa + \log t))$ time that is asymptotically larger. Even with this, we note RB-SS is faster in practice for several reasons. The core reason ends up being memory locality. As mentioned above, the usage of SIMD operations effectively means RB-SS requires $O(n)$ word operations in practice (same with ANP). Even more critical, all the non-zero entries of the generator matrix in RB-SS are grouped together. In contrast, ANP places three non-zero values in three random locations in each row. The locality of RB-SS enables faster memory access resulting in faster times in ANP. We note this phenomenon where the cache locality of random band matrices results in faster concrete times than peeling-based matrices (see [9] for example). In Section 6, we show that RB-SS is more concretely efficient than prior state-of-the-art schemes [2, 70].

5 APPLICATIONS

In this section, we consider applications of RB-SS to various important cryptographic primitives that utilize secret sharing. We will consider the simplest setting for these primitives including non-adaptive adversaries and the synchronous network model. Our goal is to showcase that RB-SS can be used to improve applications that already rely on secret sharing. We leave it as future work to consider more complex settings and other properties.

Algorithm 7 $\mathcal{F}$-F-VSS.Init algorithm

Input: $1^\lambda, n, t$: security parameter, number of parties and privacy threshold.
Output: pp: public parameters.
Sample random function $H : \mathbb{F}^t \to \mathcal{M}$.
return pp $\leftarrow (H, 1^\lambda, n, c \leftarrow K(t), t)$.

Algorithm 8 $\mathcal{F}$-F-VSS.Share protocol

Input: for dealer: public parameters pp and message m.
Input: for party i: public parameters pp.
Output: for party i: share s_i and public key pk.

Dealing round (executed by dealer):
Parse pp $= (H, 1^\lambda, n, c, t)$.
Sample random vector $\mathbf{x} \leftarrow \mathbb{F}^t$.
Compute pk $\leftarrow [g^{\mathbf{x}_1}, \ldots, g^{\mathbf{x}_t}]$.
Compute mask $\leftarrow m + H(\mathbf{x})$.
Randomly select coin tosses $R \leftarrow \{0,1\}^\lambda$.
Compute $(\mathsf{seed}_1, \ldots, \mathsf{seed}_n) \leftarrow \mathcal{F}.\mathsf{RandGen}(1^\lambda; R)$.
for $i \in [n]$ **do**
  Compute $\mathbf{M}_i \leftarrow \mathcal{F}.\mathsf{ExpandRow}(\mathsf{seed}_i)$.
  Compute $y_i \leftarrow \mathbf{M}_i \cdot \mathbf{x}$.
  Set $s_i \leftarrow (i, \mathsf{seed}_i, y_i)$.
  Send s_i to party i over private channel.
Broadcast (pk, mask, R) to all parties.

Verification round (executed by each party $i \in [n]$):
Receive $s_i = (i, \mathsf{seed}_i, y_i)$ and (pk, mask, R).
Compute $b_i \leftarrow \mathcal{F}$-F-VSS.Verify$(s_i, \mathsf{pk}, R)$.
If $b_i \neq 1$, broadcast a complaint.

Complaint round (executed by each party $i \in [n]$):
Let C be the complaining parties. If $|C| \geq t$, output $\perp$.
The dealer broadcasts $(s_j)_{j \in C}$.
If $\mathcal{F}$-F-VSS.Verify$(s_j, \mathsf{pk}) \neq 1$, for any $j \in C$, output $\perp$.
Otherwise, output s_i.

Algorithm 9 $\mathcal{F}$-F-VSS.Reconstruct algorithm

Input: pp, $(s_i)_{i \in [n]}$, pk, mask: public parameters, shares, public key, and mask.
Output: m: reconstructed message.
Parse pp $= (H, 1^\lambda, n, c, t)$.
Compute subset $I \subset [n]$ of size $|I| = c$ such that, for all $i \in I$, $\mathcal{F}$-F-VSS.Verify$(s_i, \mathsf{pk}) = 1$.
return $\mathcal{F}$-SS.Reconstruct(s_I, pp).

5.1 Verifiable Secret Sharing

We extend our framework, $\mathcal{F}$-SS, to enable verifying shares and detecting malicious dealer behavior. We will consider the synchronous setting where an adversary statically (non-adaptive) corrupts at most $t-1$ parties and the dealer. Our construction will consist of an interactive Share consisting of three rounds for dealing, verifying and complaining (same as prior works [51, 73]) extending techniques of Feldman [40] and Pedersen [64]. See Figure 8 in Appendix C.4 for detailed comparisons of different VSS schemes.

Algorithm 10 $\mathcal{F}$-F-VSS.Verify algorithm

Input: s, pk, R: share, public key, and coin tosses.
Output: $b \in \{0,1\}$: verification output.
If $s = \perp$, **return** 0.
Parse $s = (i, \mathsf{seed}, y)$ and pk $= [g^{\mathbf{x}_1}, \ldots, g^{\mathbf{x}_t}]$.
Run $\mathcal{F}$.RandGen using coin tosses R and check by using j that seed is correct.
Compute $\mathbf{m} \leftarrow \mathcal{F}.\mathsf{ExpandRow}(\mathsf{seed})$.
Compute $C \leftarrow \prod_{j \in [t]} (g^{\mathbf{x}_j})^{\mathbf{m}_j}$.
return 1 iff $C = g^y$.

5.1.1 Feldman-based VSS. The VSS scheme of Feldman [40] used a discrete logarithm-based commitment that we adapt. At a high level, we will add a commitment to the random vector $\mathbf{x} \in \mathbb{F}^t$ generated by the dealer by broadcasting pk $= [g^{\mathbf{x}_1}, \ldots, g^{\mathbf{x}_t}]$. This enables each party to verify that their shares are correctly formed. Recall that the i-th share is $s_i = \mathbf{M}_i \cdot \mathbf{x}$ for the matrix $\mathbf{M}$ used by the dealer. Each party can verify that their share is well-formed by checking the following: $g^{s_i} = (g^{\mathbf{x}_1})^{\mathbf{M}_{i,1}} \cdots (g^{\mathbf{x}_t})^{\mathbf{M}_{i,t}}$. The remainder of the VSS scheme follows similarly to prior synchronous VSS schemes [51, 73]. We present the formal description of $\mathcal{F}$-F-VSS in Algorithms 7-10.

Verification of a party's share depends only on the non-zero entries in $\mathbf{M}_i$. When using random band matrices with $\mathcal{F} = \mathsf{RB}$, each row vector contains $w = O(\kappa + \log t)$ non-zero entries. Furthermore, all entries of $\mathbf{M}_i$ are zero or one. Therefore, verification only requires $O(\kappa + \log t)$ group multiplications (and no exponentiations).

Theorem 8. *For any $c \geq (1+\alpha)t$ and if discrete logarithm is hard, then* RB-F-VSS *is a (t, c)-VSS in the random oracle model against non-adaptive adversaries with error at most $2^{-\kappa}$.*

Broadcasting M. In $\mathcal{F}$-F-VSS, we broadcast the matrix $\mathbf{M}$ by publishing the randomness R used to execute $\mathcal{F}$.RandGen. This is necessary to verify that all parties use the same matrix $\mathbf{M}$. For random band matrices $\mathcal{F} = \mathsf{RB}$, note that RB.RandGen essentially outputs n random seeds. In practice, we can use R as $O(\lambda)$-bit key of a PRF F and compute $\mathsf{seed}_i \leftarrow F(R, i)$. This is more efficient than broadcasting $O(n)$ seeds or the entire matrix $\mathbf{M}$.

Correctness Guarantees. Due to the fact that $\mathbf{M}$ must be broadcast, our scheme no longer provides negligible error against adaptive corruptions. An adaptive adversary may prevent honest parties from reconstructing the secret correctly. We leave it as an open problem to obtain correctness against adaptive corruptions for verifiable secret sharing using weakly correct erasure codes.

5.1.2 Pedersen-based VSS. We present another construction based on the VSS scheme of Pedersen [64]. In this case, there are two generators g and h. Along with the random vector $\mathbf{x}$, another vector $\mathbf{v}$ is generated uniformly at random for blinding. The dealer broadcasts $\mathsf{pk} = [g^{\mathbf{x}_1} \cdot h^{\mathbf{v}_1}, \ldots, g^{\mathbf{x}_t} \cdot h^{\mathbf{v}_t}]$. Each share is modified to contain both $\mathbf{y}_i = \mathbf{M}_i \cdot \mathbf{x}$ and $\mathbf{u}_i = \mathbf{M}_i \cdot \mathbf{v}$. Then, each party verifies their share by checking: $g^{\mathbf{y}_i} \cdot h^{\mathbf{u}_i} = (g^{\mathbf{x}_1} \cdot h^{\mathbf{v}_1})^{\mathbf{M}_{i,1}} \cdots (g^{\mathbf{x}_t} \cdot h^{\mathbf{v}_t})^{\mathbf{M}_{i,t}}$. We defer the description of $\mathcal{F}$-P-VSS to Appendix C.2 in Algorithms 11-14.

Similar to $\mathcal{F}$-F-VSS, we note the verification algorithm runs in $O(\kappa + \log t)$ time when using random band matrices $\mathcal{F} = \mathsf{RB}$ and requires only group multiplications. Consider any $g^{\mathbf{x}_i} \cdot h^{\mathbf{v}_i}$. If $\mathbf{v}_i$ is chosen uniformly at random from $\mathbb{Z}_p$ assuming the underlying

group as prime order p, then we note that $\mathbf{v}_i$ is statistically binding for $\mathbf{x}_i$. In other words, it impossible to determine $\mathbf{x}_i$ even when $\mathbf{x}_i \in \{0, 1\}$ is binary. We present the pseudocode in Appendix C.2.

Theorem 9. *For any $c \geq (1+\alpha)t$, if discrete logarithm is hard for a bilinear pairing group, then* RB-P-VSS *is a (t, c)-VSS in the random oracle model against non-adaptive adversaries with error at most $2^{-\kappa}$.*

Note that RB-P-VSS is more computationally expensive than RB-F-VSS. In particular, the dealing round and verification requires twice as many group multiplications. We note that RB-F-VSS does not require trusted setup unlike RB-P-VSS. In contrast, RB-P-VSS provides the benefit of statistical binding.

5.2 Distributed Randomness Generation

Next, we show that our secret sharing scheme RB-SS can be used to help generate randomness in distributed settings.

5.2.1 Distributed Verifiable Random Function. A *(t, c)-distributed verifiable random function* (DVRF) that enables any n parties to jointly compute the output of a verifiable random function (VRF) on input message m, even in the presence of $t-1$ corrupted parties. We present new DVRF constructions from our verifiable secret sharing (VSS) schemes in Section 5.1 (see Appendix D).

5.2.2 Decentralized Randomness Beacons. Finally, we note that there is a straightforward approach to construct a decentralized randomness beacon (DRB) using DVRF schemes as done in prior works [25, 42, 48]. At a high level, we execute Setup of DVRF to share secret keys across all n parties. For round r, we will use the evaluation of the DVRF at input r as the random output for round r denoted by Ω_r. To do this, each individual user broadcasts its partial evaluation. Afterwards, the final random value can be reconstructed. We point readers to Appendix E for more details.

5.3 Non-Homomorphic Applications

In our framework in Section 4.1, we showed that there were several ways to instantiate the hash function used in the secret sharing scheme. One way is to use a random oracle that would no longer enable linear homomorphism. The benefit of this approach is slightly faster algorithms and smaller shares. We present some large-scale applications that do not require homomorphism.

5.3.1 Federated Learning. In federated learning [50], the goal is to train a global model under coordination by a central server over the private data of many users. A core piece is secure aggregation [5, 16] where the server learns the sum of inputs from a cohort. Secret sharing is used to reconstruct the values of dropouts. Shares are never algebraically manipulated before reconstruction. Therefore, homomorphism is not necessary. The usage of RB-SS can also increase cohort size and improve accuracy (see Appendix F).

5.3.2 Distributed Storage. Many applications (such as cryptocurrency wallets) rely on secret sharing to share private keys across multiple machines so that an attacker must compromise at least t to learn the private key. The above can be used for distributed secure storage of any information beyond private keys. In this case, secret sharing is only used for reconstruction amongst a subset of machines and, thus, non-homomorphic secret sharing may be used. The benefit of our scheme for distributed storage is that RB-SS can enable higher privacy thresholds efficiently. Higher privacy thresholds means that attackers must gain access to even more machines making successful compromises more challenging.

6 EXPERIMENTAL EVALUATION

Choosing RB-SS **Parameters.** We use experimental evaluation to determine parameters, α and w, for RB-SS with error probability $\epsilon = 2^{-40}$ ($\kappa = 40$). In our experiments, we fix $\alpha = 0.05$ and use the formula $w(t, \kappa) = (0.9282 \log_2(t) + \kappa - 6.867)/0.1325$ for the band width selection. We also plot the error probabilities and explain how we empirically derived the function $w(t, \kappa)$ in Figure 7 in Appendix B.1. For RB-SS, we set $c = (1+\alpha)t = 1.05t$.

Setup. We implemented RB-SS, RB-F-VSS, RB-P-VSS, RB-F-DVRF and RB-P-DVRF in C++ using 3000 lines of code. The implementation of all hash functions are SHA256-based with 16-byte seeds. We rely on the BoringSSL implementations of SHA256 and secp256r1 as the elliptic curve for our VSS schemes. For our DVRF schemes, we use the implementation of 254-bit BN curves with a type-III bilinear pairing from [55]. We ran all experiments using a Ubuntu PC with 12 cores, 3.7 GHz Intel Xeon W-2135 and 64 GB of RAM. Our experiments enable AVX2 and AVX-512 instruction sets with SIMD instructions. We do not consider network costs or delays. All reported results use single-thread execution as the average of at least 10 trials with standard deviation less than 10% of the average.

Message Size. In our evaluation, we will only consider messages of $b = 128$ bits (i.e., $b = \lambda$ bits). This suffices as one can use the techniques in [52] to handle larger messages by sharing a single encryption key. Therefore, we will focus our evaluation on message spaces of size $|\mathcal{M}| = 2^\lambda$ as the additional encryption and decryption costs are the same for all secret sharing schemes with this technique.

6.1 Secret Sharing

We compare our construction RB-SS with prior works. We report all results in Figure 3. For our experimental evaluation, we will compare with the schemes of Shamir [70] and Applebaum *et al.* [2] that we denote as ANP. We implemented ANP ourselves in C++ as no public implementation was available at the time. Even though RB-SS provides stronger guarantees against adaptive corruption, we show that RB-SS is more efficient than ANP that only handles static corruptions. For proper comparison, both RB-SS and ANP use the same linear extractor described in [2].

We note there are other prior works studying near-linear time schemes including [2, 29, 36, 54], but only Applebaum *et al.* [2] consider practical instantiations. Another line of work [24, 53] consider secret sharing using only XOR operations that are more efficient. However, all these schemes require $\Omega(nt)$ time for sharing and/or reconstruction that are already worse than Shamir's scheme.

To our understanding, Shamir's scheme remains the most used secret sharing construction in practice. We consider naive evaluation and FFT for sharing. For reconstruction, we consider Lagrange interpolation and fast modular transforms [17] as done in [73]. We rely on [56] for implementations of [17].

Share Construction. First, we note that both FFT and the share construction of RB-SS are significantly faster than the trivial share construction in Shamir's scheme. Sharing with RB-SS is slightly

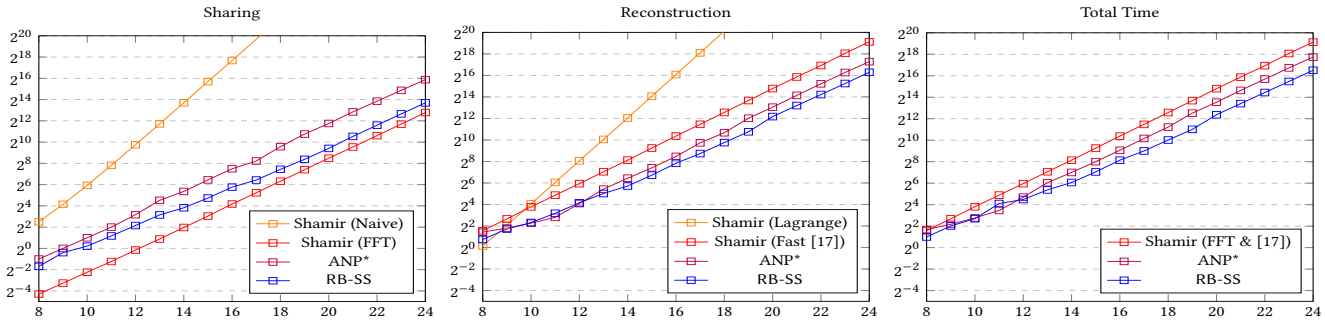

Figure 3: Computational cost of secret sharing with $n = 2t$. X-axis are $\log_2(t)$ and y-axis are milliseconds (ms). We denote ANP with an asterisk(*) as it is only suitable for settings with static corruption unlike the other two constructions.

n	t	[16] Secret Sharing	[16] Total	RB-SS Secret Sharing	RB-SS Total
1000	300	58	329	**16**	**281**
1500	450	172	781	**37**	**629**
2000	600	401	1,477	**61**	**1,134**

Figure 4: Comparison of secure aggregation protocol [16] plugging in RB-SS. All times reported in seconds.

slower than FFT, but the total time of RB-SS will be smaller due to significantly faster reconstruction. We note that RB-SS has significantly smaller sharing time that is 2.4x faster than ANP .

Message Reconstruction. We see that fast modular transforms outperform Lagrange interpolation for larger values of t in Shamir's scheme. However, both are significantly slower than the message reconstruction scheme of RB-SS by 2-7.8x. With privacy threshold $t = 2^{20}$, Shamir's scheme requires more than 28 seconds to reconstruct the message whereas RB-SS requires less than 5 seconds. RB-SS has 1.5-6.8x smaller total sharing and reconstruction time starting from t as small as 256. Finally, we note that RB-SS is 2x faster than ANP with improvements starting from $t = 2^{13}$.

Total Time. In terms of total time, RB-SS is the fastest while Shamir's scheme is the slowest. RB-SS is up to 6.1x faster than Shamir's scheme in total time with improvements starting from as small as $t = 256$. RB-SS has total time that is 2.3x faster than ANP on top of the fact that RB-SS also withstands adaptive corruptions.

Share Size. We note that our framework increases the share size as we need to include the seed to generate the matrix row and the masked message into each share. In practice, the masked message can be a part of the public parameters as it is identical for all parties. Therefore, we only increase the share size by a small 128-bit seed that is used as input to a pseudorandom generator.

Erasure Code. We can also compare the underlying erasure code performance. In particular, the erasure code's encoding and decoding are the majority of the computational cost in the sharing and reconstruction of the corresponding secret sharing scheme. As a result, we can see that our linear erasure code from random band matrices, RB, is more efficient than the erasure codes used in [2].

6.2 Secure Aggregation for Federated Learning

We plug RB-SS into the secure aggregation implementation by Bonawitz *et al.* [16]. We choose this protocol as it is deployed at Google [39] along with a public implementation [38]. We believe using RB-SS would improve other secure aggregation protocols such as [5]. Our results are reported in Figure 4. We consider time used in secret sharing as well as total time. We set $t = 0.3n$ following prior experiments [16]. RB-SS reduces time spent during secret sharing by 3.6-6.6x. In turn, the total time decreases by 13%-22%. Note that the savings increase as n (and, thus, $t = 0.3n$) increases. We picked choices of n consistent with current cohort sizes [15]. For larger cohort sizes n, RB-SS will provide even more improvement. The implementation [38] uses Lagrange interpolation as opposed to the faster algorithms in [17]. However we note that Lagrange interpolation outperforms fast modular transform [17] for $t \in \{300, 450, 600\}$ from Figure 3. RB-SS outperforms both algorithms in this regime.

6.3 Verifiable Secret Sharing

We compare RB-F-VSS and RB-P-VSS with eVSS [51], SCRAPE [21], AMT VSS [73] and DHPVSS [22]. All results are reported in Figure 5. eVSS used improved polynomial commitments reducing time to verify shares in $O(1)$ time, but sharing required $O(nt)$ time. AMT VSS balanced the costs requiring $O(\log t)$ time to verify a single share but reduced sharing time to $O(n \log t)$. For reconstruction both eVSS and AMT VSS may use fast interpolation [17]. We compare with eVSS and AMT VSS using the implementations from [56]. We implement SCRAPE and DHPVSS in C++ for comparison with the underlying Shamir's scheme using FFT and fast modular transforms. See Figure 8 (Appendix C.4) for more VSS comparisons.

Sharing Time. We evaluate the time of the dealing round. As the other two rounds depend on the number of invalid shares, we evaluate verification of a single share. The complaint and verification round times can be extrapolated by the number of complaints and verification time. The dealing time of RB-F-VSS is 8.2-25.2x smaller than the other schemes. RB-P-VSS is twice as expensive compared to RB-F-VSS but significantly faster than eVSS and AMT VSS. This is not surprising as both only require t or $2t$ exponentiations and executing the sharing of RB-SS. The setup of AMT VSS [73] requires executing multipoint polynomial evaluation (identical to fast interpolation [17]) while eVSS has $O(nt)$ time setup. Sharing in SCRAPE [21] and DHPVSS [22] require performing $4n$ and $3n$ exponentiations respectively as well as Shamir's sharing algorithm. Both components are larger than required by RB-F-VSS and RB-P-VSS.

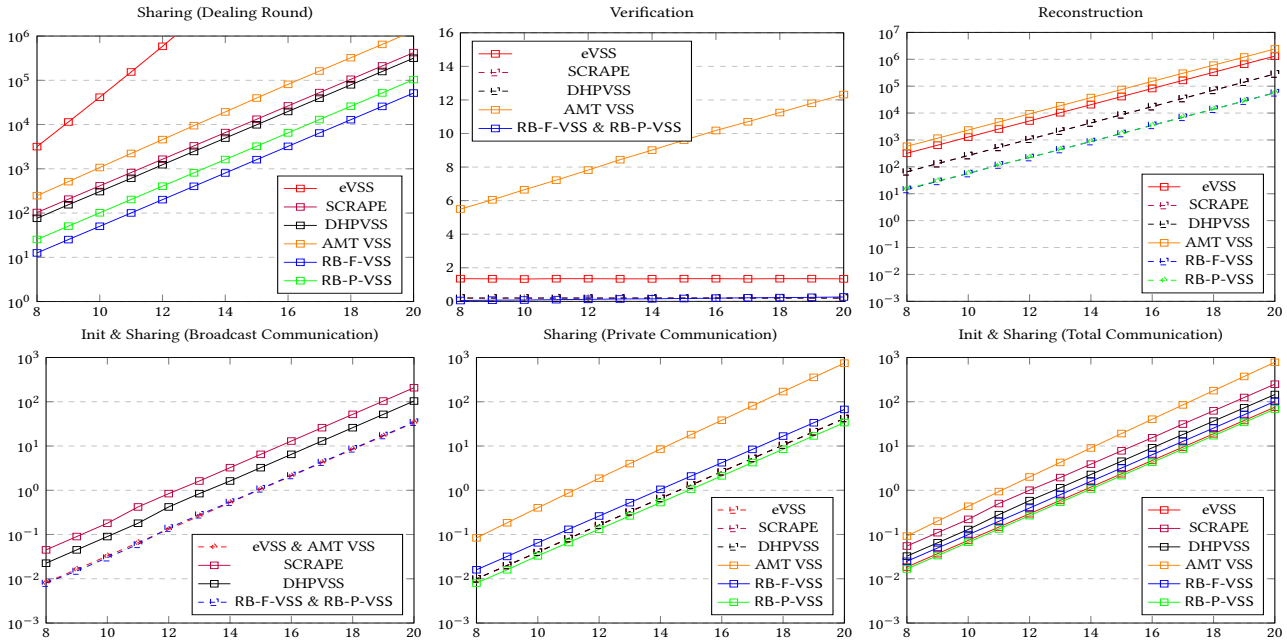

Figure 5: Evaluation of VSS with $n = t + c$. All x-axis are $\log_2(t)$ and y-axis are milliseconds (ms) or megabytes (MB).

For verification, both RB-F-VSS and RB-P-VSS are faster. Even though verification requires $O(\lambda+\log t)$ group multiplications growing in t, we stress that RB-F-VSS and RB-P-VSS perform no exponentiations. Both RB-F-VSS and RB-P-VSS uses 5x less time than eVSS. Note, eVSS is slower due to the use of BN curves with bilinear pairings instead of sec256r1. As SCRAPE and DHPVSS use only $O(1)$ exponentiations, SCRAPE's verification time is essentially similar to RB-F-VSS and RB-P-VSS. As t grows larger, SCRAPE and DHPVSS will eventually be superior as the verification time of RB-F-VSS and RB-P-VSS requires $O(\lambda + \log t)$ time. Although, this requires very large t with no notable difference for $t \leq 2^{20}$.

Communication. We evaluate broadcast and private communication. We assume all public parameters are broadcast. RB-F-VSS and RB-P-VSS have nearly no initialization communication as they only need to broadcast a single hash function. In contrast, eVSS and AMT VSS broadcast t public keys while SCRAPE and DHPVSS broadcast n public keys. During sharing, both RB-F-VSS and RB-P-VSS broadcast t public keys while eVSS and AMT VSS broadcast a single commitment. So, the total broadcast costs are similar for all four primitives except for SCRAPE and DHPVSS that broadcast n public keys meaning it has worse communication than the others. The private communication during sharing is $O(n)$ equivalent to the underlying secret sharing scheme: RB-SS (for RB-F-VSS and RB-P-VSS) or Shamir's scheme (for eVSS and SCRAPE). The exception is AMT VSS requiring $O(n \log t)$ private communication. RB-F-VSS and eVSS have similar communication while SCRAPE and DHPVSS are slightly larger due to broadcasting n public keys. RB-P-VSS obtains at least 9% less communication than the rest.

Message Reconstruction. We consider message reconstruction in the best case where only t shares need verification. If more shares need verification, RB-F-VSS and RB-P-VSS will provide even better improvements due to their smaller verification times compared to eVSS and AMT VSS. There will not be significant differences compared to SCRAPE and DHPVSS when verifying more shares since verification times are similar. RB-F-VSS and RB-P-VSS enjoy 2.7-23.2x faster reconstruction compared to the rest.

6.4 Distributed Verifiable Random Functions

Finally, we evaluate RB-F-DVRF compared to state-of-the-art DVRF-based schemes. All results are in Figure 6. We compare against DVRF-based schemes with small number of rounds. One scheme uses threshold BLS signatures as first detailed by DFINITY [48] that is also deployed by CloudFlare [28] and the drand project [35]. A recent work [42] considers space-time trade-offs of threshold BLS protocols presenting GLOW-DVRF and DDH-DVRF. For our evaluation, we use the implementations available at [34]. We omit comparisons with RandHerd [72] as it has high round complexity (the reason it was not used in the drand project [62]) and Strobe [4] due to the lack of a public implementation.

Setup. We compare the total setup time for each party that includes generating all necessary information to send to other parties and verifying inputs from other parties. In our experiments, we choose $n = t + c$. For RB-F-DVRF, this means $n = (2 + \alpha)t$. For the other schemes, we use $n = 2t$. We assume that all inputs verify correctly and no work is needed to handle complaints. We see that RB-F-DVRF has significantly faster setup time than the other schemes. This is expected as the other schemes have $O(nt)$ setup time whereas RB-F-DVRF uses only $O(n\lambda)$ time during setup.

Randomness Generation Round. We consider three separate components of randomness generation: partial evaluation, verification of partial evaluations and final reconstruction. For partial evaluation, we consider a single party's computation. Similarly, for partial verification, we consider a single share. In practice, each round could require verifying c shares in the best case and n shares

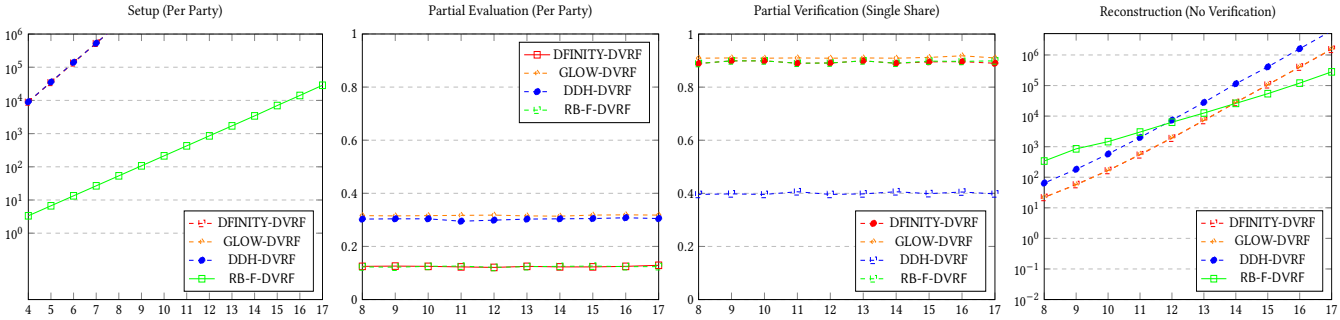

Figure 6: Computational cost of DVRF-based DRBs with $n = t + c$. All x-axis are $\log_2(t)$ and y-axis are milliseconds (ms).

in the worst case. As this is the same for all protocols, we only benchmark verifying a single share for comparison. We see that RB-F-DVRF has identical times to DFINITY-DVRF. GLOW-DVRF and DDH-DVRF have higher evaluation times and either similar or faster verification. For reconstruction, we note that the public implementation [34] uses the secret keys to reconstruct (i.e., assume that the reconstructing party is trusted). In the same setting, RB-F-DVRF is 2-6x faster than the other constructions.

7 RELATED WORKS

Secret Sharing. Threshold secret sharing was introduced independently by Blakley [12] and Shamir [70]. Krawczyk [52] showed that shares may be reduced using computational assumptions. Recent work has considered secret sharing with efficient sharing and reconstruction algorithms. For example, schemes with near-linear time were studied in [2, 29, 36, 54]. To our knowledge, all of these works are theoretical nature except for a single practical instantiation presented in [2]. Recent work [3] ruled out the existence of $O(n)$ time algorithms for binary shares under certain settings. Another line of work considers building secret sharing schemes utilizing only XOR operations. The first XOR-based scheme [53] required $O(n^3t^3)$ reconstruction time. For smaller t, [24] presented a scheme with $O(nt\lambda)$ sharing and reconstruction. To our knowledge, all the schemes require at least quadratic in t time.

Blakley and Meadows [13] presented the notion of ramp secret sharing. A series of works have improved the share sizes of ramp schemes [53, 57, 61] as well as studying the required share size [20].

Verifiable Secret Sharing. VSS schemes were introduced by Chor *et al.* [26] where parties can verify the consistency of their shares. Two early VSS schemes are by Feldman [40] and Pedersen [64]. More efficient VSS [8, 27, 51, 73] as well as those with stronger properties [1, 59] were presented in recent years.

Distributed Randomness Generation. Randomness beacons were first introduced by Rabin [65]. Many prior works have studied decentralized randomness beacons in the trusted setup [4, 8, 18, 19, 42, 47, 72] as well as transparent setup [30, 31, 69] settings to list some examples. We point to [25] for more details and comparisons.

Random Band Matrices. Random band matrices were first introduced in [32]. Its first application was in ribbon filters [33] that outperformed prior filter data structures (such as bloom filters). In recent years, random band matrices have found several applications in cryptography including keyword PIR [63], oblivious key-value stores [9] as well as oblivious ciphertext compression [10].

8 CONCLUSIONS

We present a generic framework to build ramp secret sharing schemes from any family of linear erasure codes with distributed generator matrices. In particular, we present a framework allowing one to build secret sharing schemes against adaptive corruption from erasure codes with weak correctness guarantees. We build a distributed generator matrix RB from random band matrices that is more efficient than other erasure codes used in practical secret sharing instantiations. Using our framework, we instantiate RB-SS with RB that is more efficient than prior state-of-the-art schemes. We show RB-SS has potential implications towards many applications by presenting improved protocols for federated learning, verifiable secret sharing and distributed verifiable random functions.

Acknowledgements. The authors would like to thank the anonymous CCS reviewers for providing feedback to improve our paper.

REFERENCES

[1] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. 2023. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation. In *CRYPTO 2023*.
[2] Benny Applebaum, Oded Nir, and Benny Pinkas. 2023. How to Recover a Secret with O(n) Additions. In *CRYPTO 2023*. Springer-Verlag, 236–262.
[3] Marshall Ball, Alper Çakan, and Tal Malkin. 2021. Linear threshold secret-sharing with binary reconstruction. In *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
[4] Donald Beaver, Konstantinos Chalkias, Mahimna Kelkar, Lefteris Kokoris Kogias, Kevin Lewi, Ladi de Naurois, Valeria Nicolaenko, Arnab Roy, and Alberto Sonnino. 2021. STROBE: Stake-based Threshold Random Beacons. Cryptology ePrint Archive, Paper 2021/1643. https://eprint.iacr.org/2021/1643
[5] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 1253–1269. https://doi.org/10.1145/3372297.3417885
[6] Mihir Bellare and Phillip Rogaway. 2007. Robust computational secret sharing and a unified account of classical secret-sharing goals. In *ACM CCS 2007*.
[7] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *20th ACM STOC*. ACM Press, 1–10. https://doi.org/10.1145/62212.62213
[8] Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. 2021. Randpiper–reconfiguration-friendly random beacons with quadratic communication. In *ACM CCS 2021*.
[9] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Near-Optimal Oblivious Key-Value Stores for Efficient PSI, PSU and Volume-Hiding Multi-Maps. In *USENIX Security 23*. USENIX Association, Anaheim, CA, 301–318.
[10] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2024. Batch PIR and Labeled PSI with Oblivious Ciphertext Compression. In *USENIX Security*.
[11] Bitcoin Armory [n. d.]. https://btcarmory.com/.
[12] George Robert Blakley. 1979. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*. IEEE Computer Society.
[13] G. R. Blakley and Catherine Meadows. 1984. Security of Ramp Schemes. In *CRYPTO'84 (LNCS, Vol. 196)*, G. R. Blakley and David Chaum (Eds.). Springer,

Heidelberg, 242–268.
[14] Andrej Bogdanov, Yuval Ishai, Emanuele Viola, and Christopher Williamson. 2016. Bounded Indistinguishability and the Complexity of Recovering Secrets. In *CRYPTO 2016, Part III (LNCS, Vol. 9816)*, Matthew Robshaw and Jonathan Katz (Eds.). Springer, Heidelberg, 593–618. https://doi.org/10.1007/978-3-662-53015-3_21
[15] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. *Proceedings of machine learning and systems* 1 (2019), 374–388.
[16] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 1175–1191. https://doi.org/10.1145/3133956.3133982
[17] Allan Borodin and Robert Moenck. 1974. Fast modular transforms. *J. Comput. System Sci.* 8, 3 (1974), 366–386.
[18] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *Journal of Cryptology* 18, 3 (July 2005), 219–246. https://doi.org/10.1007/s00145-005-0318-0
[19] Jan Camenisch, Manu Drijvers, Timo Hanke, Yvonne-Anne Pignolet, Victor Shoup, and Dominic Williams. 2021. Internet Computer Consensus. Cryptology ePrint Archive, Paper 2021/632. https://eprint.iacr.org/2021/632
[20] Ignacio Cascudo, Ronald Cramer, and Chaoping Xing. 2013. Bounds on the threshold gap in secret sharing and its applications. *IEEE Transactions on Information Theory* 59, 9 (2013), 5600–5612.
[21] Ignacio Cascudo and Bernardo David. 2017. SCRAPE: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security*. Springer, 537–556.
[22] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. 2022. YOLO YOSO: Fast and Simple Encryption and Secret Sharing in the YOSO Model. In *ASIACRYPT 2022, Part I (LNCS)*. Springer, Heidelberg, 651–680. https://doi.org/10.1007/978-3-031-22963-3_22
[23] David Chaum, Claude Crépeau, and Ivan Damgård. 1988. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *20th ACM STOC*. ACM Press, 11–19. https://doi.org/10.1145/62212.62214
[24] Liqun Chen, Thalia M Laing, and Keith M Martin. 2016. Efficient, XOR-based, ideal threshold schemes. In *Cryptology and Network Security: 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*. Springer.
[25] Kevin Choi, Aathira Manoj, and Joseph Bonneau. 2023. SoK: Distributed Randomness Beacons. In *2023 IEEE Symposium on Security and Privacy (SP)*.
[26] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. 1985. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *26th FOCS*. IEEE Computer Society Press, 383–395. https://doi.org/10.1109/SFCS.1985.64
[27] Ashish Choudhury. 2020. Optimally-resilient unconditionally-secure asynchronous multi-party computation revisited. *Cryptology ePrint Archive* (2020).
[28] Cloudflare Randomness Beacon [n. d.]. https://developers.cloudflare.com/randomness-beacon/.
[29] Ronald Cramer, Ivan Bjerre Damgård, Nico Döttling, Serge Fehr, and Gabriele Spini. 2015. Linear Secret Sharing Schemes from Error Correcting Codes and Universal Hash Functions. In *EUROCRYPT 2015, Part II (LNCS, Vol. 9057)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Heidelberg, 313–336. https://doi.org/10.1007/978-3-662-46803-6_11
[30] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. 2022. Spurt: Scalable distributed randomness beacon with transparent setup. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2502–2517.
[31] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018, Part II (LNCS, Vol. 10821)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, Heidelberg, 66–98. https://doi.org/10.1007/978-3-319-78375-8_3
[32] Martin Dietzfelbinger and Stefan Walzer. 2019. Efficient Gauss Elimination for Near-Quadratic Matrices with One Short Random Block per Row, with Applications. In *ESA 2019*.
[33] Peter C Dillinger and Stefan Walzer. 2021. Ribbon filter: practically smaller than Bloom and Xor. *arXiv preprint arXiv:2103.02515* (2021).
[34] Distributed Verifiable Random Functions: an Enabler of Decentralized Random Beacons [n. d.]. https://github.com/fetchai/research-dvrf.
[35] Drand: Distributed randomness beacon [n. d.]. https://drand.love.
[36] Erez Druk and Yuval Ishai. 2014. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In *ITCS 2014*, Moni Naor (Ed.). ACM, 169–182. https://doi.org/10.1145/2554797.2554815
[37] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. 2014. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*.
[38] Federated Compute Platform [n. d.]. https://github.com/google/federated-compute.
[39] Federated Learning Blog [n. d.]. Federated Learning: Collaborative Machine Learning without Centralized Training Data. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.
[40] Paul Feldman. 1987. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *28th FOCS*. IEEE Computer Society Press, 427–437. https://doi.org/10.1109/SFCS.1987.4
[41] Matthew K. Franklin and Moti Yung. 1992. Communication Complexity of Secure Computation (Extended Abstract). In *24th ACM STOC*. ACM Press, 699–710. https://doi.org/10.1145/129712.129780
[42] David Galindo, Jia Liu, Mihair Ordean, and Jin-Mann Wong. 2021. Fully distributed verifiable random functions and their application to decentralised random beacons. In *2021 IEEE European Symposium on Security and Privacy*.
[43] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2021. Oblivious Key-Value Stores and Amplification for Private Set Intersection. In *CRYPTO 2021, Part II (LNCS, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer, Heidelberg, Virtual Event, 395–425. https://doi.org/10.1007/978-3-030-84245-1_14
[44] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *Journal of Cryptology* 20, 1 (Jan. 2007), 51–83. https://doi.org/10.1007/s00145-006-0347-3
[45] Michel Goemans. 2015. Chernoff bounds, and some applications. https://math.mit.edu/~goemans/18310S15/chernoff-notes.pdf.
[46] Michael T Goodrich and Michael Mitzenmacher. 2011. Invertible bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 792–799.
[47] Jens Groth. 2021. Non-interactive distributed key generation and key resharing. *Cryptology ePrint Archive* (2021).
[48] Timo Hanke, Mahnush Movahedi, and Dominic Williams. 2018. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548* (2018).
[49] Bailey Kacsmar, Chelsea Komlo, Florian Kerschbaum, and Ian Goldberg. 2020. Mind the Gap: Ceremonies for Applied Secret Sharing. *PoPETs* 2020, 2 (April 2020), 397–415. https://doi.org/10.2478/popets-2020-0033
[50] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* 14, 1–2 (2021), 1–210.
[51] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT 2010 (LNCS, Vol. 6477)*, Masayuki Abe (Ed.). Springer, Heidelberg, 177–194. https://doi.org/10.1007/978-3-642-17373-8_11
[52] Hugo Krawczyk. 1994. Secret Sharing Made Short. In *CRYPTO'93 (LNCS, Vol. 773)*, Douglas R. Stinson (Ed.). Springer, Heidelberg, 136–146. https://doi.org/10.1007/3-540-48329-2_12
[53] Jun Kurihara, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. 2008. A new (k, n)-threshold secret sharing scheme and its extension. In *Information Security: 11th International Conference, ISC 2008*. Springer, 455–470.
[54] Yuan Li. 2023. Secret Sharing on Superconcentrator. *arXiv preprint arXiv:2302.04482* (2023).
[55] libff: C++ library for Finite Fields and Elliptic Curves [n. d.]. https://github.com/scipr-lab/libff.
[56] libpolycrypto [n. d.]. https://github.com/alinush/libpolycrypto.
[57] Fuchun Lin, Mahdi Cheraghchi, Venkatesan Guruswami, Reihaneh Safavi-Naini, and Huaxiong Wang. 2019. Secret Sharing with Binary Shares. In *ITCS 2019*, Avrim Blum (Ed.), Vol. 124. LIPIcs, 53:1–53:20. https://doi.org/10.4230/LIPIcs.ITCS.2019.53
[58] Michael G Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A Spielman. 2001. Efficient erasure correcting codes. *IEEE Transactions on Information Theory* 47, 2 (2001), 569–584.
[59] Atsuki Momose, Sourav Das, and Ling Ren. 2023. On the Security of KZG Commitment for VSS. In *ACM CCS 2023*.
[60] Multi-Party Threshold Signature Scheme [n. d.]. https://github.com/bnb-chain/tss-lib.
[61] Wakaha Ogata, Kaoru Kurosawa, and Shigeo Tsujii. 1993. Nonperfect secret sharing schemes. In *Advances in Cryptology—AUSCRYPT'92: Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, December 13–16, 1992 Proceedings 3*. Springer, 56–66.
[62] Origins of drand [n. d.]. https://drand.love/about/#origins-of-drand.
[63] Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Don't be Dense: Efficient Keyword PIR for Sparse Databases. In *USENIX Security 2023*.
[64] Torben P. Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91 (LNCS, Vol. 576)*, Joan Feigenbaum (Ed.). Springer, Heidelberg, 129–140. https://doi.org/10.1007/3-540-46766-1_9
[65] Michael O. Rabin. 1983. Transaction protection by beacons. *J. Comput. System Sci.* 27, 2 (1983), 256–267.
[66] Tal Rabin and Michael Ben-Or. 1989. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *21st ACM STOC*. ACM

Press, 73–85. https://doi.org/10.1145/73007.73014
[67] Srinivasan Raghuraman and Peter Rindal. 2022. Blazing Fast PSI from Improved OKVS and Subfield VOLE. ACM Press, 2505–2517. https://doi.org/10.1145/3548606.3560658
[68] Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics* 8, 2 (1960).
[69] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. 2020. Hydrand: Efficient continuous distributed randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 73–89.
[70] Adi Shamir. 1979. How to Share a Secret. *Communications of the Association for Computing Machinery* 22, 11 (Nov. 1979), 612–613.
[71] Shamir Seals [n. d.]. https://developer.hashicorp.com/vault/docs/concepts/seal#shamir-seals.
[72] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. 2017. Scalable Bias-Resistant Distributed Randomness. In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 444–460. https://doi.org/10.1109/SP.2017.45
[73] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. 2020. Towards scalable threshold cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE.
[74] Stefan Walzer. 2021. Peeling close to the orientability threshold–spatial coupling in hashing-based data structures. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2194–2211.

A RAMP SECRET SHARING

A.1 Ramp Secret Sharing Definitions

Definition 2 (Ramp Secret Sharing). *A* ramp secret sharing scheme *over message space* $\mathcal{M}$ *consists of the following tuple of algorithms* $\mathsf{SS} = (\mathsf{Init}, \mathsf{Share}, \mathsf{Reconstruct})$ *such that:*

- Init *receives security parameter* 1^λ*, number of parties* n*,* correctness threshold $c \le n$ *and* privacy threshold $t \le c$. Init *outputs* public parameters pp.
- Share *receives the message* $m \in \mathcal{M}$ *and public parameters* pp *and outputs an* n*-tuple of shares,* $(s_1, \ldots, s_n)$.
- $\mathsf{Reconstruct}$ *receives a subset of shares* s_I *for some subset* $I \subseteq [n]$ *and public parameters* pp *and outputs a message* $m' \in \mathcal{M}$.

Next, we move onto defining privacy of a ramp secret sharing scheme SS with respect to a stateful, polynomial time adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$. Our notion of privacy corresponds to the CSS privacy definition in [6]. We use experiments PriExp^b, for $b = 0, 1$, that are parametrized by the secret sharing scheme SS and by the adversary $\mathcal{A}$. In both experiments, $\mathcal{A}$ chooses two challenge messages m_0 and m_1 after seeing the public parameters pp. For our schemes, one can imagine pp to be public hash functions (random oracles) and the choice of the challenge messages may depend on the hash functions. Then, in PriExp^b, $\mathcal{A}$ receives up to $t-1$ shares of the challenge message m_b for adaptively corrupted parties of $\mathcal{A}$'s choice and then, finally, $\mathcal{A}$ outputs one bit. We say that the scheme is δ-private if the probabilities of outputting 1 in $\mathsf{PriExp}^0_{\mathsf{SS},\mathcal{A}}$ and $\mathsf{PriExp}^1_{\mathsf{SS},\mathcal{A}}$ differ at most δ.

$\mathsf{PriExp}^b_{\mathsf{SS},\mathcal{A}}(\lambda, n, c, t)$:

(1) Challenger executes $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, n, c, t)$.
(2) Adversary picks two messages, $(m_0, m_1) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pp})$.
(3) Challenger computes $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}(m_b, \mathsf{pp})$.
(4) Adversary $\mathcal{A}_2$ adaptively corrupts at most $t-1$ parties by submitting index $i \in [n]$ and receiving s_i.
(5) Adversary outputs $\eta \leftarrow \mathcal{A}_3()$.
(6) Return η.

Next, we define experiment CorExp that we use to formalize the notion of correctness of a ramp secret sharing scheme. Here we let adversary $\mathcal{A}$ adaptively corrupt up to $n-c$ parties and learn their shares with the goal of forcing the remaining uncorrupted parties to reconstruct the wrong message. We say that SS has ϵ-error if no adversary has probability greater than ϵ of succeeding. The experiment is parameterized by the secret sharing scheme SS and by the polynomial time stateful adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. The notion we obtain corresponds to the CR0 recoverability definition in [6].

$\mathsf{CorExp}_{\mathsf{SS},\mathcal{A}}(\lambda, n, c, t)$:

(1) Challenger executes $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, n, c, t)$.
(2) Adversary picks challenge message, $m \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pp})$.
(3) Challenger computes $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}(\mathsf{pp}, m)$.
(4) Adversary $\mathcal{A}_2$ adaptively corrupts at most $n-c$ parties by submitting index $i \in [n]$ and receiving s_i. Let $I \subset [n]$ be the set of parties corrupted by $\mathcal{A}_2$.
(5) Return 1 iff $\mathsf{Reconstruct}(\mathsf{pp}, s_{[n]\setminus I}) \neq m$.

We are now ready to define ramp secret sharing.

Definition 3 ((ϵ, δ)-Ramp Secret Sharing). *A ramp secret sharing scheme* $\mathsf{SS} = (\mathsf{Init}, \mathsf{Share}, \mathsf{Reconstruct})$ *is a* (t, c)-ramp secret sharing scheme with ϵ-error and δ-privacy *if the following hold:*

- Correctness*: For any polynomial time adversary* $\mathcal{A}$ *and sufficiently large* λ *and* n

$$\Pr[\mathsf{CorExp}_{\mathsf{SS},\mathcal{A}}(\lambda, n, c, t) = 1] \le \epsilon.$$

- Privacy*: For any polynomial time adversary* $\mathcal{A}$ *and sufficiently large* λ *and* n

$$|\Pr[\mathsf{PriExp}^0_{\mathsf{SS},\mathcal{A}}(\lambda, n, c, t) = 1] - \Pr[\mathsf{PriExp}^1_{\mathsf{SS},\mathcal{A}}(\lambda, n, c, t) = 1]| \le \delta.$$

For a ramp scheme, it is possible that $c > t$ parties are needed to reconstruct. We note that one can obtain *threshold schemes* by consider (t, t)-ramp schemes where $c = t$.

A.2 Proofs for Ramp Secret Sharing

All missing proofs for RB-SS are presented here. We start with correctness and efficiency of $\mathcal{F}$-SS below.

Proof of Theorem 3. We note that Init only samples a random hash function that requires $O(\lambda)$ time. For Share, we generate $\mathbf{M}$ that requires $O(f_{\mathsf{RandGen}}(n, t) + n \cdot f_{\mathsf{ExpandRow}}(t))$ time. Generating $\mathbf{x}$ requires $O(t)$ time. Computing mask requires $O(t)$ time to compute $H(\mathbf{x})$ and $\mathsf{mask} = m + H(\mathbf{x})$. Afterwards, we compute the matrix-vector multiplication $\mathbf{M} \cdot \mathbf{x}$ that requires $O(a)$ time. So, the total running time of Share is $O(f_{\mathsf{RandGen}}(n, t) + n \cdot f_{\mathsf{ExpandRow}}(t) + a + t)$. Finally, consider $\mathsf{Reconstruct}$ that executes $\mathsf{ExpandRow}$ to obtain $\mathbf{M}_I$ requiring $O(t \cdot f_{\mathsf{ExpandRow}}(t))$ time. Executing Solve requires $O(f_{\mathsf{Solve}}(t))$ time. Afterwards, $\mathsf{Reconstruct}$ computes $H(\mathbf{x})$ and $\mathsf{mask} + H(\mathbf{x})$ requiring $O(t)$ time. So, the total time is $O(t \cdot f_{\mathsf{ExpandRow}}(t) + f_{\mathsf{Solve}}(t) + t)$. □

Proof of Theorem 7. To obtain the desired bounds, we use the random band matrix family RB of dimension $n \times \ell$ with band length $w = O(\kappa + \log \ell)$ over a field $\mathbb{F}$ of size 2^λ and setting $\ell = t$. By choosing the appropriate constant $\alpha > 0$, we get that RB is a

$((1+\alpha)t, 2^{-\kappa})$-generator matrix. Therefore, RB-SS is a $(t, (1+\alpha)t)$-ramp secret sharing scheme with error at most $2^{-\kappa}$ and security advantage negligible in λ, by Theorem 4. Finally, we obtain the running times by plugging in the fact that every random band matrix has at most $O(nw)$ non-zero entries and that Solve executes in $O(tw)$ time. □

Privacy. Next, we move to privacy and proving Theorem 4. Here, we will rely on certain properties of the underlying hash function H. Recall that we provided several options for H. Prior works have instantiated H using linear universal hash functions [29] and the dot-product with a small-integer vector [2] that are randomness extractors. We introduced a third option where H is a random oracle. We will proceed in two steps. First, we show that using H as a random oracle enables privacy. Afterwards, we extend the framework to generality assuming that H is a randomness extractor.

If H is a random oracle, we show that any adversary $\mathcal{A}$ that can only make $q = \mathsf{poly}(\lambda, n)$ queries to H can distinguish experiments $\mathsf{PriExp}^0_{t,n}$ and $\mathsf{PriExp}^1_{t,n}$ with negligible probability for sufficiently large choices of field sizes for $\mathbb{F}$ and $\mathcal{M}$. Roughly speaking, we show that H can be viewed as a randomness extractor when H is a random oracle and the adversary is computationally bounded. Let us proceed more formally.

Theorem 10. *$\mathcal{F}$-SS is a $(t, K(t))$-ramp secret sharing scheme that, for security parameter λ, is δ-secure with $\delta \le q^2/|\mathcal{M}| = q^2/2^\lambda$ when adversary $\mathcal{A}$ makes at most q queries to H that is modeled as a random oracle.*

Proof. Let I be the set of at most $t-1$ parties corrupted by adversary $\mathcal{A}$ and let Y_I be the vector of the components y_i from the shares of the corrupted parties. With a slight abuse of terminology, we will refer to y_i as the *share* of party i. Let us fix a set J of $z = t - |I| \ge 1$ parties such that $Z = I \cup J$ has size $|Z| = t$ and the matrix M_Z has rank t.

Next we define the set $\mathbb{X}$ of *compatible vectors* in the following way. For every sequence A of $z \ge 1$ shares for the parties in J, we consider the vector Y_A obtaining by appending the z shares of A to Y_I and we let X_A be the vector such that $Y_A = M_Z \cdot X_A$. Since M_Z has full rank, for every Y_A, there exists a unique such vector X_A and $A \neq A'$ implies that $X_A \neq X_{A'}$. Therefore, the set $\mathbb{X}$ of the X_A's as A ranges over the sets of $z \ge 1$ shares consists of exactly $|\mathbb{F}|^z \ge 2^\lambda = |\mathcal{M}|$ elements. Note also that the set $\mathbb{X}$ of compatible vectors is, for any fixed I and Y_I, independent of the set J chosen to augment I.

Let us consider the experiment Hybrid^b that coincides with PriExp^b with the exception that the restriction of H to the set $\mathbb{X}$ is one-to-one and onto $\mathcal{M} = \mathbb{F}^\lambda$. That is, if $A \neq A'$, not only $X_A \neq X_{A'}$ but also $H(X_A) \neq H(X_{A'})$. Then, the views of $\mathcal{A}$ in Hybrid^0 and Hybrid^1 coincide.

On the other hand, unless the random oracle H is queried by $\mathcal{A}$ at two points $X_A \neq X_{A'}$ for which $H(X_A) = H(X_{A'})$, the views of $\mathcal{A}$ in PriExp^b and Hybrid^b also coincide. Therefore the distance between PriExp^b and Hybrid^b is upper bounded by the probability that, for random H, two queries over compatible vectors issued by $\mathcal{A}$ collide. The probability that any two queries collide is $1/|\mathcal{M}|$. If $\mathcal{A}$ makes q queries, by the union bound, the probability that at least 2 of the q queries collide is at most $q^2/2 \cdot 1/|\mathcal{M}|$. Therefore, the statistical distance between $\mathsf{PriExp}^0_{t,n}$ and $\mathsf{PriExp}^1_{t,n}$ is at most $q^2/|\mathcal{M}| = q^2/2^\lambda$. □

Next, we bound the adversary's running time to finish the proof when H is a random oracle.

Proof of Theorem 5. Follows by Theorem 10 and observing that a polynomial adversary makes $q \le \mathsf{poly}(\lambda, n)$ queries. □

The theorems above hold also for adversaries that have unbounded computational power but can query the random oracle on at most $\mathsf{poly}(\lambda, n)$ inputs. Finally, we note that our construction is built against adaptive adversaries.

Next, we show that the framework yields a private secret sharing scheme assuming that H is any randomness extractor. We start by defining randomness extractors as follows.

Definition 4. *$H : \{0,1\}^s \times X \to \{0,1\}^b$ is a (k, γ)-randomness extractor if, the random variable X satisfies min-entropy $H_\infty(X) \ge k$ and for uniformly random $s \in S$, then the statistical distance of distributions $(S, H(S, X))$ and (U_s, U_b) is at most γ where U_s and U_b are uniformly random strings of s and b bits.*

Another way to view the above definition is to consider a family of hash functions. Drawing a random hash function is equivalent to picking a random s-bit seed. This view aligns with our constructions where we sample random hash functions and do not consider random seeds. Therefore, we will use this view where we consider H to be randomly drawn for a hash family and omit the seed.

The proof proceeds nearly identically except we only make the assumption that H is a randomness extractor. Recall that the hash function $H : \mathbb{F}^t \to \mathcal{M}$ receives t field elements as input $\mathbf{x} \in \mathbb{F}^t$ and outputs a random element $H(\mathbf{x})$ from the message space. Roughly speaking, at least one of the t elements will be random from the adversary's views meaning that the min-entropy of the input $\mathbf{x}$ is $H_\infty(\mathbf{x}) \ge \log|\mathbb{F}|$. We suppose that H is a $(\log|\mathbb{F}|, 2^{-\lambda})$-randomness extractor whose output will have statistical distance at most $2^{-\lambda}$ from a random element in $\mathcal{M}$.

We assume that $|\mathbb{F}| \ge |\mathcal{M}| \cdot 2^{2\lambda}$. This matches prior works [2, 29] that assume randomness extractors that match the leftover hash lemma. These extractors are able to output $k - 2\lambda$ bits whose statistical distance is at most $2^{-\lambda}$ from a uniformly random string assuming the input has min-entropy k.

Proof of Theorem 4. The proof follows identically to the proof of Theorem 10 with only a few modifications. First, we use the same argument to show that the input to H has min-entropy at least $\log(|\mathbb{F}|^z) \ge \log(|\mathbb{F}|)$ for the $z \ge 1$ missing shares. Afterwards, we use the property that the output of H has statistical distance at most $2^{-\lambda}$ from a random element from $\mathcal{M}$. This translates into the statistical distance between the two privacy experiments. This is sufficient to show that $\mathcal{F}$-SS is δ-secure for $\delta \ge 2^{-\lambda}$. □

Linearity. Finally, we prove that our framework yields linear secret sharing schemes when the hash function H is linear.

Proof of Theorem 6. Consider sharing two messages $m, m' \in \mathcal{M}$ using the same seeds used to expand rows of the generator matrix $\mathbf{M}$. For each party $i \in [n]$, they would receive shares $s_i = (\mathsf{mask}, \mathsf{seed}_i, \mathbf{y}_i)$ and $s'_i = (\mathsf{mask}', \mathsf{seed}_i, \mathbf{y}'_i)$ that are the distributed

outputs of Share. Next, we compute the addition of the two shares as the following: $t_i = (\mathsf{mask} + \mathsf{mask}', \mathsf{seed}_i, \mathbf{y}_i + \mathbf{y}'_i)$.

Suppose that $\mathbf{M}$ is the generator matrix for $\mathsf{seed}_1, \ldots, \mathsf{seed}_n$. Then, we can see the following:

$$\mathbf{y} + \mathbf{y}' = \mathbf{M}\mathbf{x} + \mathbf{M}\mathbf{x}' = \mathbf{M}(\mathbf{x} + \mathbf{x}').$$

Suppose there is a sufficiently large subset of parties $I \subseteq [n]$ that attempt to reconstruct using shares $t_I = \{t_i\}_{i \in I}$. The first step of Reconstruct attempts to solve the following linear system:

$$(\mathbf{y} + \mathbf{y}')_I = \mathbf{M}(\mathbf{x} + \mathbf{x}')_I$$

with the goal to compute $\mathbf{x} + \mathbf{x}'$. Regardless of whether we are working with sums of shares or regular shares, the properties of $\mathbf{M}$ (that is, full rank and efficiently finding a solution) remain the same. Therefore, Reconstruct will retrieve $\mathbf{x}+\mathbf{x}'$ even when working over sums of shares. Finally, Reconstruct will compute

$$\begin{aligned}\mathsf{mask} + \mathsf{mask}' - H(\mathbf{x} + \mathbf{x}') &= \mathsf{mask} + \mathsf{mask}' - H(\mathbf{x}) - H(\mathbf{x}') \\ &= (\mathsf{mask} - H(\mathbf{x})) + (\mathsf{mask}' - H(\mathbf{x}')) \\ &= m + m'\end{aligned}$$

where we use that H is linear and $H(\mathbf{x} + \mathbf{x}') = H(\mathbf{x}) + H(\mathbf{x}')$. □

B RANDOM BAND MATRIX ANALYSIS

In this section, our goal is to analyze random band matrices with dimension $(1+\alpha)t \times t$ and band length $w = O(\kappa + \log t)$. In particular, our goal is to prove that these random band matrices with more rows than columns have full column rank of t with probability at least $1 - 2^{-\kappa}$. Furthermore, solving the associated linear system can be done very efficiently in time $O(t \cdot w)$. For the modified version of Gaussian elimination that we will analyze, we refer readers back to the description in Section 3.2. To our knowledge, these properties were not previously known. The original work [32] focused on random band matrices with dimension $(1-\alpha)t \times t$ with the goal of proving full row rank. We are unaware of a direct reduction from this setting of more columns to our problem with more rows.

For our analysis, we will utilize the connection between random band matrices and coin-flipping Robin Hood hashing that was presented in [32] (this reduction was independent of the dimensions of the random band matrix). We start by describing standard Robin Hood hashing with n items inserted into m entries with displacement at most $w-1$. Each of the n items is randomly assigned to one of the first $m - w + 1$ entries. When inserting an item, it performs linear probing from its assigned entry and its $w - 1$ entries to the right. Standard Robin Hood hashing will insert the item into the first empty entry found in the linear probing. In the coin-flipping variant, each time an item encounters an empty entry, a random coin is flipped to decide whether to insert the item into the empty entry or move onto the next entry.

Dietzfelbinger and Walzer [32] showed that one can directly relate random band matrices to coin-flipping Robin Hood hashing. In particular, each of the $n = (1+\alpha)t$ rows may be viewed as items. Furthermore, the t columns may be viewed as the entries of the hash table. Consider the i-th row of the random band matrix that will be isomorphic to the i-th item in coin-flipping Robin Hood hashing. Then, the starting location of the w-length band for the i-th row is equivalent to the assigned entry for the i-th item. Suppose that the i-th item in coin-flipping Robin Hood hashing is inserted into the j-th entry. This turns out be equivalent that the i-th row in the Gaussian elimination using the j-th column as the pivot. As a result, we can use coin-flipping Robin Hood hashing to directly analyze random band matrices.

Lemma 1. *Pick a sufficiently large constant $\alpha > 0$ and let $w = O(\kappa + \log t)$. Then, a $(1+\alpha)t \times t$ random band matrix has full column rank of t except with probability at most $2^{-\kappa}$.*

Proof. We will first rely on Lemma 3 in [32] that enables us to analyze the column rank of random band matrices through the success of coin-flipping Robin Hood hashing. In particular, each of the $(1+\alpha)t$ rows may be viewed as items to be inserted and each of the t columns may be viewed as hash table entries. If the i-th item is placed into the j-th entry, this is equivalent to the pivot of the i-th row of the random band matrix being the j-th column. Therefore, it suffices for us to prove that by inserting at most $(1+\alpha)t$ items in coin-flipping Robin Hood hashing, each of the t entries becomes occupied. This is equivalent to each of the t columns being picked as a pivot by one row. This immediately implies that the $(1+\alpha)t \times t$ random band matrix has full column rank.

Consider the j-th column for any $j \in [t]$. We will analyze the probability that coin-flipping Robin Hood hashing with $(1+\alpha)t$ items will never insert an item into the j-th entry. First, we analyze the number of items that are candidates to be inserted into the j-th entry. This set of candidates are those items that are assigned to positions in the set $S = \{j - w + 1, j - w + 2, \ldots, j\} \subset [t]$. Note that all entries in S are computed modulo t (the number of columns) to incorporate wrap-around. The probability that an item is assigned to a position in the set S is at most $|S|/(t-w) = w/(t-w) \geq w/t$. Denote the random variable be such that $X_i = 1$ if and only if the i-th item is assigned to an entry in S. Otherwise, $X_i = 0$. Note that $\Pr[X_i] \geq n/w$ for all $i \in [(1+\alpha)t]$. Let $X = X_1 + \ldots + X_{(1+\alpha)t}$. Then, the expected value is $\mu = \mathbf{E}[X] \geq (1+\alpha)t \cdot (w/t) = (1+\alpha)w$. By the Chernoff bound (see [45] as an example for the statement), we get that

$$\Pr[X < (1-\beta)\mu] \leq e^{-(\mu\beta^2)/2}.$$

By picking $w = O(\kappa + \log t)$ sufficiently large, we can get that

$$\Pr[X < w + \kappa + \log t] \leq 2^{-\kappa - \log t - 1}.$$

In other words, we pick w such that $(1+\alpha)(1-\beta)w \geq w + \kappa + \log t$ and $e^{-(\mu\beta^2)/2} \leq 2^{-\kappa - \log t - 1}$. For now, let us assume that $X \geq w + \kappa + \log t$. In the worst case, $w-1$ of these items will be inserted in columns before the j-th column. Therefore, at least $\kappa + \log t + 1$ items will arrive at the j-th column. The probability that none of these $\kappa + \log t + 1$ items will be inserted into the j-th column is at most $2^{-\kappa - \log t - 1}$. Therefore, the probability that the j-th column is empty after inserting $(1+\alpha)t$ items is at most $2^{-\kappa - \log t}$. Finally, we take a union bound over all t entries to get that any of the t entries are empty is at most $t \cdot 2^{-\kappa - \log t} \leq 2^{-\kappa}$ completing the proof. □

Necessity of Modification. Recall that we modified the random band matrices from [32] such that starting locations for bands are chosen uniformly at random from $[t]$ and the w-length bands may wrap-around from the last column to the first column. Suppose we considered the original approach to simply sample the starting location uniformly at random from $[t - w]$ without wrap-around. Consider the first column. Note, the probability that any of the

$(1+\alpha)t$ rows will pick the first column as the starting location is $1/t$. Therefore, the probability that no row picks the first column as the starting location is

$$\left(1-\frac{1}{t}\right)^{(1+\alpha)t} = e^{-(1+\alpha)}.$$

In other words, the probability that none of the $(1+\alpha)t$ rows chooses the first column as the starting location is constant as α is constant. In this case, it is impossible for the resulting matrix to have full column rank. Therefore, our modification with wrap-arounds was necessary to ensure full column rank.

Next, we prove that the running time of Gaussian elimination after sorting rows by the start location of the random band is efficient. In particular, we show that $(1+\alpha)t \times t$ dimension random band matrices may be converted into row echelon form using Gaussian elimination in time $O(t \cdot w)$ where w is the length of the band. Once again, the prior analysis in [32] only proves this for $(1-\alpha)t \times t$ dimension random band matrices. We are unaware of a way to utilize the analysis in [32] directly. Instead, we prove this same fact for $(1+\alpha)t \times t$ random band matrices using a different proof technique relying on the fact that $(1+\alpha)t \times t$ random band matrices almost always have full column rank.

Lemma 2. *Pick a sufficient large constant $\alpha > 0$ and let $w = O(\kappa + \log t)$ such that $w \leq t/2$. Suppose that the word size is $\Omega(\kappa + \log t)$. Then, a $(1+\alpha)t \times t$ random band matrix may be reduced to row echelon form using Gaussian elimination after sorting rows by band start location in time $O(tw)$ except with probability at most $2^{-\kappa+1}$.*

Proof. First, we apply Lemma 1. By picking constant $\alpha > 0$ and $w = O(\kappa + \log t)$ appropriately, we can guarantee that $(1+\alpha)t \times t$ random band matrices have full column rank except with probability at most $2^{-\kappa}$. From now on, we assume the random band matrix has full column rank.

Next, we apply the following observation. Consider the j-th column for any $j \in [t]$. As soon as the i-th row, for some $i \in [(1+\alpha)t]$, chooses the j-th column as a pivot, then we know that all other rows following the i-th row will always have a zero in the j-th column. Therefore, we simply need to bound the number of rows that may have non-zero entries in the j-th column before the i-th row uses the j-th column as the pivot.

To do this, we use a similar approach as done in our prior proof of Lemma 1 for most columns. Fix the j-th column for any $j \in [t]$ such that $j \geq w$. In other words, we consider columns where every row with a non-zero entry in the j-th column will not wrap-around from the last column to the first column. Then, we can clearly see that only rows where the starting location is in the set $S = \{j-w+1, j-w+2, \ldots, j\} \subset [t]$ may have a non-zero entry in the j-th column. Clearly, the j'-th row where $j' < j-w+1$ will not have a zero entry as the random band appears strictly before the j-th column. For the j'-th where $j' > j$, we already assumed that the matrix has full column rank. Therefore, some row with a starting location in S will already have chosen the j-th column as the pivot. Therefore, the j'-th row where $j' > j$ will also have a zero entry in the j-th column. As $j \geq w$, we know that these rows will be processed consecutively during Gaussian elimination after sorting rows by their starting location. Therefore, we simply need to bound the total number of rows with starting locations in S.

First, we denote X_i to be the random binary variable denoting whether the i-th row has a starting location in S. If the i-th row has a starting location in S, we set $X_i = 1$. Otherwise, we set $X_i = 0$. Note, $\Pr[X_i = 1] = |S|/(t-w) = w/(t-w) \leq 2w/t$ as we assumed that $w \leq t/2$. Set $X = X_1 + \ldots + X_{(1+\alpha)t}$ and we know that $\mu = \mathbf{E}[X] \leq (1+\alpha)t \cdot (2w/t) = 2(1+\alpha)w$. Finally, we apply the Chernoff bound (see [45] for example) and get that

$$\Pr[X > (1+\beta)\mu] < e^{-(\mu\beta^2)/(2+\beta)}$$

for any constant $\beta > 0$. Once again, we pick $w = O(\kappa + \log t)$ and β sufficiently large such that

$$\Pr[X > \gamma w] < 2^{-\kappa - \log t}$$

for some constant $\gamma > 0$. Therefore, we know that the number of rows that will process the j-th column is at most $O(w)$ except with probability $2^{-\kappa-\log t}$. Applying a union bound across all t columns, we get that all columns will be processed by at most $O(w)$ rows except with probability $t \cdot 2^{-\kappa-\log t} \leq 2^{-\kappa}$. Since each of the $O(w)$ rows processes the j-th column $O(w)$ times, the j-th column is processed $O(w^2)$ times. In other words, the last $t-w$ columns are processed $O(tw^2)$ times in total. Since the word size is $\Omega(\kappa + \log t) = \Omega(w)$ and each band row fits in $O(1)$ words, performing a row reduction takes $O(1)$ time. Noting that each row reduction processes w columns, we get that the running time is $O(tw)$. For the final probability, recall that we conditioned on the fact that the random matrix has full column rank. Therefore, the algorithm has running time at most $O(tw)$ except with probability at most $2^{-\kappa+1}$.

Finally, we handle the first w columns. We cannot use the argument above as the set of rows that may contain a non-zero entry in the first w columns are not processed consecutively. However, we note that the worst case is that each of these w columns are processed in all $(1+\alpha)t = O(t)$ rows only adding an additional $O(tw)$ computational cost. □

In the above lemma, we made the assumption that each w-bit length band fits into $O(1)$ words. Note, this mirrors our practical evaluation since $w \leq 400$ for all our experimented values and modern machines have word sizes (registers) of at least 64 bits, but more reasonably 128-256 bits. Furthermore, we note this is a reasonable comparison with other algorithms such as the fast polynomial interpolation algorithm in [17] that assumes $O(1)$ time for multiplications of two word-sized integers.

Finally, we prove the main result about random band matrix families, RB, that we needed in Section 3.2.

Proof of Theorem 1. Note that RandGen requires $O(n)$ to generate all $O(n)$ seeds. ExpandRow requires $O(w)$ time to generate the single w-bit band in each row. The fact that RB is a distributed $((1+\alpha)t, 2^{-\kappa})$-generator matrix follows directly from Lemma 1. The running time of the algorithm follows directly from Lemma 2. □

B.1 RB-SS Parameters

In Figure 7, for $\alpha = 0.05$ and various values of t, we plot the graphs of w versus κ, where the error probability ϵ equals $2^{-\kappa}$. All experimental data points were collected using at least 2^{13} runs and up to 2^{25} runs. Solid lines are constructed from experimental data

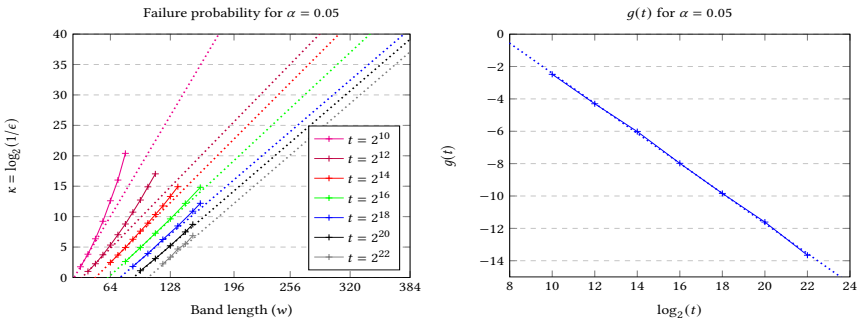

Figure 7: Left graph plots log error probability based on w. Right graph plots $g(t)$ when plugging in data points from left graph into $\kappa = 0.1325w + g(t)$.

points, and dotted lines are plotted as pessimistic approximation of the underlying function. From the plot, we observe that failure probability is a superlinear function (for small t) with respect to the band width, and converges to a linear function as t gets larger. In particular, for $t = 2^{18}, 2^{20}$ and 2^{22}, a line with slope 0.1325 provides good approximations on a lower bound on the necessary band length w for a certain security parameter.

For a fixed t, we will approximate the underlying function as a linear function with slope 0.1325. Assuming the underlying function is a superlinear function (as shown in Figure 7), this will give us pessimistic approximation of κ. In other words, the linear function is a base line for the minimum of some choice of w to obtain specific κ, but it may actually obtain even larger values of κ. We will approximate as $\kappa(w, t) = 0.1325w + g(t)$ where $g(t)$ is the y-intercept. For each t, we then solve for $g(t)$ by plugging in the first data point to equation $\kappa = 0.1325w + g(t)$. We plot these values in Figure 7. We observe that the function looks like a linear function with respect to $\log_2(t)$. Using linear regression, we obtain $g(t) = -0.9282 \log_2(t) + 6.867$. Thus, we obtain approximation

$$\kappa(w, t) = 0.1325w - 0.9282 \log_2(t) + 6.867.$$

As a sanity check, we plug in the data point $t = 2^{20}$, $w = 144$ into the previous equation and get $\kappa = 7.383$ which is not far from the actual value 7.49 from the experiment. For smaller t, by plugging in the data point $t = 2^{14}$, $w = 88$ we obtain $\kappa = 5.532$ whereas the experiment gave us $\kappa = 6.28$. The calculated value is smaller than the data point but still shows that $\kappa(w, t)$ is suitable as a pessimistic approximation.

To find a suitable band width w with respect to t and κ, we rearrange the equation to obtain $w(t, \kappa) = (0.9282 \log_2(t) + \kappa - 6.867)/0.1325$. Finally, we note that this corroborates with Theorem 1 proving that $w = O(\kappa + \log t)$ is sufficient to obtain $2^{-\kappa}$ error probability.

Other Choices of α. For other values of α, we expect the plots to look similar to Figure 7, except for the "steepness" of the functions. In particular, larger the α, steeper the functions would look. Intuitively, with t and w fixed, larger values of α would result in the matrix to have full column rank with higher probability. This implies a smaller w is sufficient to obtain a fixed error probability of $\epsilon = 2^{-\kappa}$.

C VERIFIABLE SECRET SHARING

C.1 Verifiable Secret Sharing Definitions

Definition 5 (Verifiable Ramp Secret Sharing). *A (t, c)-VSS over message space $\mathcal{M}$ consists of a tuple* vSS = (Init, Share, Reconstruct) *such that:*

- Init *receives security parameter 1^λ, number of parties n,* correctness threshold *$c \le n$, and* privacy threshold *$t \le c$, and outputs* public parameters pp.
- Share *is an interactive protocol between the dealer and n parties. The dealer and the n parties receive the public parameters* pp. *The dealer also receives message $m \in \mathcal{M}$ as an additional private input. The private output of party i is either a share s_i or $\perp$. In the latter case, we say that party i has* disqualified *the dealer.*
- Reconstruct *receives a subset of shares s_I for some subset $I \subseteq [n]$ and public parameters* pp, *and outputs a message $m' \in \mathcal{M}$ or $\perp$ where the latter detects malicious behavior.*

A VSS scheme vSS must satisfy the privacy notion which we formalize by experiments $\mathsf{vPriExp}^b_{\mathsf{vSS},\mathcal{A}}$, for $b = 0, 1$ and a probabilistic polynomial time adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$. The experiment is similar to PriExp^b with the only exception that Share is now an interactive protocol. For the privacy requirement, we assume that the dealer is honest as, otherwise, an adversarial dealer receive the message in plaintext as input.

$\mathsf{vPriExp}^b_{\mathsf{vSS},\mathcal{A}}(\lambda, n, c, t)$:

(1) Challenger executes $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, n, c, t)$.
(2) Adversary picks two messages, $(m_0, m_1) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pp})$.
(3) Challenger executes Share on input (pp, m_b) with the n players, at most $t - 1$ of which can be adaptively corrupted by adversary $\mathcal{A}_2$.
(4) Return $\eta \leftarrow \mathcal{A}_3()$.

For privacy, a VSS must still satisfy the original correctness guarantees of secret sharing with an honest dealer. Additionally, it must satisfy a stronger correctness notion if the dealer is corrupted by the adversary. In this case, any subset of c honest parties must give the same output that could be a message m' or $\perp$. The two conditions above must hold even if a static adversary corrupts up to $n - c$ parties. Formally, we have the following experiment for the

Algorithm 11 $\mathcal{F}$-P-VSS.Init algorithm

Input: $1^\lambda, n, t$: security parameter, number of parties and privacy threshold.
Output: pp: public parameters.
Randomly select $g, h \leftarrow \mathbb{G}$.
Sample random function $H : \mathbb{F}^t \to \mathcal{M}$.
return $\mathsf{pp} = (H, 1^\lambda, n, c \leftarrow K(t), t, g, h)$.

strong correctness notion for VSS with respect to any polynomial time adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

$\mathsf{vCorExp}_{\mathsf{vSS},\mathcal{A}}(\lambda, n, c, t)$:

(1) Challenger executes $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, n, c, t)$.
(2) $\mathcal{A}_1(\mathsf{pp})$ decides whether to corrupt the dealer and decides on the set of at most $n - c$ corrupted parties.
If the dealer is not corrupted, $\mathcal{A}_1$ outputs also a message $m \in \mathcal{M}$.
(3) The Share protocol is executed. During the execution, $\mathcal{A}_2$ can maliciously interact as any corrupted party, including the dealer if corrupted.
If instead the dealer is not corrupted, it takes part to Share using m and pp as input.
(4) All parties broadcast their shares. Each honest party executes the Reconstruct algorithm and computes its output.
(5) If the dealer is corrupt, the experiment returns 1 unless all honest parties give the same output (that could be any message or $\perp$).
(6) If the dealer is not corrupt, the experiment returns 1 unless all honest parties give m as output.

In other words, an adversary wins the above game if it can force at least one honest party to output an inconsistent message without detecting a malicious dealer (denoted by $\perp$). Using the above experiment, we can define a VSS scheme as follows:

Definition 6 ((ϵ, δ)-Verifiable Ramp Secret Sharing). *A ramp secret sharing scheme* $\mathsf{SS} = (\mathsf{Init}, \mathsf{Share}, \mathsf{Reconstruct})$ *is a* (t, c)-verifiable ramp secret sharing scheme with ϵ-error and δ-privacy *if the following hold:*

- Correctness*: For any polynomial time adversary* $\mathcal{A}$ *and sufficiently large* λ *and* n

$$\Pr[\mathsf{vCorExp}_{\mathsf{SS},\mathcal{A}}(\lambda, n, c, t) = 1] \leq \epsilon.$$

- Privacy*: For any polynomial time adversary* $\mathcal{A}$ *and sufficiently large* λ *and* n

$$|\Pr[\mathsf{vPriExp}^0_{\mathsf{SS},\mathcal{A}}(\lambda, n, c, t) = 1] - \Pr[\mathsf{vPriExp}^1_{\mathsf{SS},\mathcal{A}}(\lambda, n, c, t) = 1]| \leq \delta.$$

C.2 Pedersen-based Verifiable Secret Sharing

We present the formal description and pseudocode for our Pedersen-based VSS, $\mathcal{F}$-P-VSS, in Algorithms 11-14. See Section 5.1.2 for detailed description. We also present the proofs associated with the VSS scheme based Pedersen commitments.

Proof of Theorem 9. Let RB be the (K, ϵ)-generator matrix family. The adversary can corrupt at most $(t-1)$ players and for RB-P-VSS we need $c = (1+\alpha)t$ honest players to reconstruct.

Algorithm 12 $\mathcal{F}$-P-VSS.Share protocol

Input: for dealer: public parameters pp and message m.
Input: for party i: public parameters pp.
Output: for party i: share s_i and public key pk.

Dealing round (executed by dealer):
Parse $\mathsf{pp} = (H, 1^\lambda, n, c, t, g, h)$.
Sample random vectors $\mathbf{x}, \mathbf{v} \leftarrow \mathbb{F}^t$.
Set $\mathsf{pk} = [g^{\mathbf{x}_1} \cdot h^{\mathbf{v}_1}, \ldots, g^{\mathbf{x}_t} \cdot h^{\mathbf{v}_t}]$.
Compute $\mathsf{mask} \leftarrow m + H(\mathbf{x})$.
Randomly select coin tosses $R \leftarrow \{0,1\}^\lambda$.
Compute $(\mathsf{seed}_1, \ldots, \mathsf{seed}_n) \leftarrow \mathcal{F}.\mathsf{RandGen}(1^\lambda; R)$.
for $i \in [n]$ **do**
  Compute $\mathbf{M}_i \leftarrow \mathcal{F}.\mathsf{ExpandRow}(\mathsf{seed}_i)$.
  Compute $y_i \leftarrow \mathbf{M}_i \cdot \mathbf{x}$ and $u_i \leftarrow \mathbf{M}_i \cdot \mathbf{v}$.
  Set $s_i \leftarrow (i, \mathsf{seed}_i, y_i, u_i)$.
  Send s_i to party i over private channel.
Broadcast $(\mathsf{pk}, \mathsf{mask}, R)$ to all parties.

Verification round (executed by each party $i \in [n]$):
Receive s_i and $(\mathsf{pk}, \mathsf{mask}, R)$.
Compute $b_i \leftarrow \mathcal{F}$-P-VSS.$\mathsf{Verify}(s_i, \mathsf{pk}, R)$.
If $b_i \neq 1$, broadcast a complaint.

Complaint round (executed by each party $i \in [n]$):
Let C be the complaining parties. If $|C| \geq t$, output $\perp$.
The dealer broadcasts $(s_j)_{j \in C}$.
If $\mathcal{F}$-P-VSS.$\mathsf{Verify}(s_j, \mathsf{pk}, R) \neq 1$ for any $j \in C$, output $\perp$. Otherwise, output s_i.

Algorithm 13 $\mathcal{F}$-P-VSS.Reconstruct algorithm

Input: $\mathsf{pp}, (s_j)_{j \in [n]}, \mathsf{pk}, \mathsf{mask}$: public parameters, shares, public key, and mask.
Output: m: reconstructed message.
Parse $\mathsf{pp} = (H, 1^\lambda, n, c, t, g, h)$.
Compute subset $J \subset [n]$ of size $|J| = c$ such that, for all $j \in J$, $\mathcal{F}$-P-VSS.$\mathsf{Verify}(s_j, \mathsf{pk}) = 1$.
Let Y_J be the set of (seed_j, y_j) from $s_j = (\mathsf{seed}_j, \mathbf{y}_j, \mathbf{z}_j)$
return $\mathcal{F}$-SS.$\mathsf{Reconstruct}(s_J, \mathsf{pp})$.

Algorithm 14 $\mathcal{F}$-P-VSS.Verify algorithm

Input: s, pk, R: share, public key, and coin tosses.
Output: $b \in \{0,1\}$: verification output.
If $s = \perp$, then **return** 0.
Parse $s = (j, \mathsf{seed}, y, u)$.
Run $\mathcal{F}.\mathsf{RandGen}$ using coin tosses R and check by using j that seed is correct. If not, return 0.
Parse $\mathsf{pk} = [g^{\mathbf{x}_1} \cdot h^{\mathbf{v}_1}, \ldots, g^{\mathbf{x}_t} \cdot h^{\mathbf{v}_t}]$.
Compute $\mathbf{m} \leftarrow \mathcal{F}.\mathsf{ExpandRow}(\mathsf{seed})$.
Compute $C \leftarrow \prod_{j \in [t]} (g^{\mathbf{x}_j} \cdot h^{\mathbf{v}_j})^{\mathbf{M}_j}$.
return 1 iff $C = g^y \cdot h^u$.

Correctness. For a dealer to be disqualified, at least one honest party must file a complaint. This implies that a honest dealer will

not be disqualified during the Share protocol. If the dealer is not disqualified then every honest party i has a share s_i that passes the verification and thus $s_i \neq \perp$. Let us now consider the Reconstruct algorithm. We observe that, for every party i, only the share $s_i = (\mathbf{y}_i, \mathbf{u}_i)$ produced by the dealer will pass the Verify algorithm. Indeed note that pk and row $\mathbf{M}_i$ (all this information is public) determine the commitment C_i of $\mathbf{y}_i$ with randomness $\mathbf{u}_i$ and, by the computational binding property, no adversary can open C_i as $\mathbf{y}'_i \neq \mathbf{y}_i$, under the discrete log assumption. Therefore malformed shares can be identified and, since the adversary can corrupt at most $n - c$ parties, at least c parties contribute their honest shares. By the property of the RB family, if the probability that reconstructions fails is thus most ϵ.

Privacy. The proof of privacy is similar to the one of Theorem 4 with the difference that the adversary's view also contains pk consisting of t commitments $\mathsf{pk}_1, \ldots, \mathsf{pk}_t$. As a reminder, the first step of the proof of Theorem 4 consists in showing that given $t-1$ shares, for each possible value A of the missing $z \geq 1$ shares, there exists a unique vector $\mathbf{x}_A$ that is consistent with the view of the adversary; in the case of Theorem 10, the view consists of the $t-1$ shares. Moreover, no two values $a \neq a'$ have the same vector; that is $\mathbf{x}_a \neq \mathbf{x}_{a'}$. If we assume that the random oracle H is one-to-one and onto on the set of vectors $\mathbf{x}_a$, then the adversary has no advantage in distinguishing m_0 from m_1; the proof terminates by showing that a random oracle that is queried q times will show this exact behaviour except with probability $O(q^2/|\mathbb{F}|)$. Therefore, the adversary's distinguishing probability is $O(q^2/|\mathbb{F}|)$.

To prove privacy of $\mathcal{F}$-P-VSS, we will thus show that, given any $t-1$ shares *and* the public key pk, for each possible choice of the missing $z+1$ shares, there exists a unique vector $\mathbf{x}_A$ and a unique vector $\mathbf{v}_A$ consistent with the adversary's view. Moreover the vectors $\mathbf{x}_A$ and $\mathbf{v}_A$ are different for different A. Having proved this property, then the proof continues as for Theorem 10.

Let (g, h) be the pair used to compute the Pedersen's commitments and let d be such that $h = g^d$. Suppose that the adversary has shares $s_i = (y_i, u_i)$ for $i \in I$, with $|I| = t-1$, and we assume that the $((t-1) \times t)$-matrix $\mathbf{M}_I$ has full rank $t-1$. We denote by Y_I the vector $[y_i]_{i \in I}$. In addition to the shares, the adversary has also the public information $\mathsf{pk} = [\mathsf{pk}_1, \ldots, \mathsf{pk}_t]$ and we let a_j be such that $\mathsf{pk}_j = g^{a_j}$, for $j = 1, \ldots, t$. Since the shares have been correctly computed, the verification is successful and thus we have that, for $i \in I$, $\sum_{j=1}^{t} \mathbf{M}_{i,j} \cdot a_j = y_i + d \cdot u_i$. Now let J be a set of $z+1$ parties with $J \cap I = \emptyset$ so that $K = I \cup J$ has cardinality t. Moreover, let us assume that the $(t \times t)$-matrix $\mathbf{M}_K$ has full rank. For a fixed vector A of length $z+1$ consisting of one value $y_j \in \mathbb{F}$ for each $j \in J$, we let Y_A be the vector of length t obtained by appending A to the vector Y_I. Then, by the full rank of $\mathbf{M}_K$, there exists a unique vector $\mathbf{x}_A$ such that $\mathbf{M}_K \cdot \mathbf{x}_A = Y_A$. Now, define the vector $\mathbf{v}_A$ of length t by setting $\mathbf{v}_j = (a_j - \mathbf{x}_j)/d$, for $j \in K$. Note that this guarantees that $g^{\mathbf{x}_j} \cdot h^{\mathbf{v}_j} = g^{\mathbf{x}_j + d\mathbf{v}_j} = g^{a_j} = \mathsf{pk}_j$ which implies that $\mathbf{x}_A$ and $\mathbf{v}_A$ are compatible with pk_j. Finally, let $Z' = \mathbf{M}_K \cdot \mathbf{v}_A$. Observe that, for $i \in I$, we have

$$\begin{aligned} u'_i &= \sum_{k \in K}^{t} \mathbf{M}_{i,k} \mathbf{v}_k \\ &= \frac{1}{d} \left(\sum_{j \in I}^{t} \mathbf{M}_{i,j} a_j - \sum_{j \in K} \mathbf{M}_{i,j} \mathbf{x}_j \right) \\ &= \frac{1}{d} \left(y_i + d u_i - y_i \right) = u_i \end{aligned}$$

which implies that the u'_i obtained from $\mathbf{v}_A$ are compatible with the ones found in the shares from I.

C.3 Feldman-based Verifiable Secret Sharing

Let $\mathcal{F}$ be a generator matrix family. In this section, we give more details about $\mathcal{F}$-F-VSS that uses the *deterministic* commitment based on discrete logarithm and is inspired by Feldman's VSS [40] but it differs from it in a crucial aspect.

Let $\mathbb{G}$ be a group of prime order p in which the discrete logarithm is hard and let g be a generator of $\mathbb{G}$. We consider the deterministic commitment algorithm that commits to a value s by computing $C = g^s$. Note that the commitment function is linear in the sense that $\mathsf{com}(s_0) \cdot \mathsf{com}(s_1) = \mathsf{com}(s_0 + s_1)$ and that $\mathsf{com}(s)^r = \mathsf{com}(r \cdot s)$. Because of linearity, the commitments $C_1 = g^{x_1}, \ldots, C_t = g^{x_t}$ of the components of $\mathbf{x}$ are sufficient to construct the commitment of each y_i and thus each player can check the validity of y_i received as part of the share and thus correctness is guaranteed. The commitment is perfectly binding (that is, unlike Pedersen's, the commitment uniquely determines the value committed) and this can be used to prove soundness in a straightforward way. For the privacy requirement, we note that the commitment is not even computationally hiding as it is trivial to distinguish a commitment of s_0 from a commitment of s_1. On the other hand, the commitment is one-way secure (given a commitment it is difficult to extract the committed value) and this should be sufficient as, roughly speaking, the messages committed, the components of $\mathbf{x}$, have sufficiently large entropy, even conditioned on $t-1$ shares and, more importantly, are chosen independently from the secret being shared. We note that is not the case for Feldman's VSS [40]. There, the VSS was constructed on top of Shamir's secret sharing scheme which implies that the commitments of the coefficients of the polynomial (that play a role analogous to the vector $\mathbf{x}$ in our construction) included a commitment to the secret being shared. Therefore, the construction of [40] needed to resort to a randomized commitment based on discrete logarithm.

Next we sketch the proof of Theorem 8. Let $\mathcal{A}$ be an adversary that receives shares $s_i = (\mathsf{mask}_i, \mathsf{seed}_i, i, y_i)$ for $i \in I$, with $|I| = t-1$, and $\mathsf{pubV} = (\mathsf{mask}, C_1, \ldots, C_t)$. Recall that we only consider non-adaptive adversaries $\mathcal{A}$ here. Let $\mathbf{x}$ be the vector such that $C_j = g^{\mathbf{x}_j}$, for $j = 1, \ldots, t$. Note that, since the shares have been correctly constructed, we have that $y_i = \sum_{j=1}^{t} \mathbf{M}_{i,j} \cdot b x_j$, for all $i \in I$, where $\mathbf{M}_i = \mathsf{ExpandRow}(\mathsf{seed}_i)$.

Now observe that if $\mathcal{A}$ does not query the random oracle on $\mathbf{x}$ then mask hides m_b information theoretically. Indeed, $\mathsf{mask} = m_b + H(\mathbf{x})$ and $H(\mathbf{x})$ is randomly chosen over $\mathbb{F}$. On the other hand, any probabilistic polynomial-time adversary $\mathcal{A}$ that receives in

Scheme	Trusted Setup	Dealing Round Time	Verification Round Time	Complaint Round Time	Recons. Time (No Verification)	Init Comm. (Broadcast)	Dealing Comm. (Broadcast)	Dealing Comm. (Private)
Feldman [40]	×	$O(n \log t)$	$O(t)$	$O(t^2)$	$O(t \log^2 t)$	$O(1)$	$O(t)$	$O(n)$
Pedersen [64]	✓	$O(n \log t)$	$O(t)$	$O(t^2)$	$O(t \log^2 t)$	$O(1)$	$O(t)$	$O(n)$
eVSS [51]	✓	$O(nt)$	$O(1)$	$O(t)$	$O(t \log^2 t)$	$O(t)$	$O(1)$	$O(n)$
SCRAPE [21]	✓	$O(n)$	$O(1)$	$O(t)$	$O(t \log^2 t)$	$O(n)$	$O(n)$	$O(n)$
AMT VSS [73]	✓	$O(n \log t)$	$O(\log t)$	$O(t \log t)$	$O(t \log^2 t)$	$O(t)$	$O(1)$	$O(n \log t)$
RB-F-VSS	×	$O(n\lambda)$	$O(\lambda)$	$O(t\lambda)$	$O(t\lambda)$	$O(1)$	$O(t)$	$O(n)$
RB-P-VSS	✓	$O(n\lambda)$	$O(\lambda)$	$O(t\lambda)$	$O(t\lambda)$	$O(1)$	$O(t)$	$O(n)$

Figure 8: Comparison of per-player or dealer worst-case asymptotic costs of VSS schemes.

input $t-1$ shares $(s_i)_{i\in I}$ and pubV and queries H on $\mathbf{x}$ with non-negligible probability can be used to solve the discrete logarithm problem. Privacy follows by combining the two observations. Let us now sketch a proof for the second observation.

Suppose that $\mathcal{A}$ is a polynomial-time algorithm that, on input $t-1$ shares and commitments $C_1, \ldots, C_t$ of the components of $\mathbf{x}$, queries H on $\mathbf{x}$ with non-negligible probability. Now consider the following probabilistic polynomial-time algorithm $\mathcal{B}$ that receives in input a challenge for the discrete logarithm problem consisting of a generator g and $C = g^x$ with a random x. $\mathcal{B}$ runs $\mathcal{A}$ and simulates $\mathsf{vPriExp}^b_{n,t}$ for $\mathcal{A}$ as follows. Note that all $\mathcal{A}$'s random oracle queries are handled by $\mathcal{B}$. As a first step $\mathcal{A}$ outputs m_0, m_1 and I. Then $\mathcal{B}$ randomly chooses seed_i, computes $\mathbf{M}_i = \mathsf{ExpandRow}(\mathsf{seed}_i)$, and randomly chooses y_i, for $i \in I$. Then $\mathcal{B}$ sets $C_1 = C$ and computes $C_2, \ldots, C_t$ so that $y_i = \sum_{j=1}^{t} \mathbf{M}_{i,j} \cdot \mathbf{x}_j$, for all $i \in I$, where $\mathbf{x}_j$ is the discrete logarithm of C_j. $\mathcal{B}$ can construct the commitments by solving the system of $t-1$ equations $y_i = \sum_{j=1}^{t} \mathbf{M}_{i,j} \cdot \mathbf{x}_j$ for $i \in I$ in the t unknowns $\mathbf{x}_1, \ldots, \mathbf{x}_t$ using $\mathbf{x}_1$ as a parameter. In other words, each $\mathbf{x}_j$, for $j > 1$, is expressed as a linear functions of $\mathbf{x}_1$ with coefficients from the y_i's. Then $C_j = g^{\mathbf{x}_j}$ is computed using the y_i's and the challenge $C = C_1$ received in input. Finally, $\mathcal{B}$ picks a random $\mathsf{mask} \leftarrow \mathbb{F}$ and constructs the shares $s_i = (\mathsf{mask}, i, \mathsf{seed}_i, y_i)$ for $i \in I$ and public verification information $\mathsf{pubV} = (\mathsf{mask}, C_1, \ldots, C_t)$ for $\mathcal{A}$. Note that shares and pubV have the same distribution as in $\mathsf{vPriExp}^0_{n,t}$ and $\mathsf{vPriExp}^1_{n,t}$. Then $\mathcal{B}$ monitors the queries $\mathbf{x}'$ for H and for each queries $\mathcal{B}$ checks if $g^{\mathbf{x}'_1} = C$. If the check is successful, $\mathcal{B}$ stops and outputs $\mathbf{x}'_1$. Note that if $\mathcal{A}$ has a non-negligible probability of querying for vector $\mathbf{x}$ committed to by $C_1, \ldots, C_t$ then $\mathcal{B}$ has the same non-negligible probability of computing the discrete logarithm of C to the base g that would be a contradiction.

By instantiating $\mathcal{F}$ with the random band matrix family RB, we obtain a VSS with very efficient verification (see the discussion for RB-VSS with Pedersen).

Note that, unlike the proofs of Theorem 4 and Theorem 9, the proof of privacy for $\mathcal{F}$-F-VSS crucially uses the fact that not only the number q of $\mathcal{A}$'s queries to the random oracle is polynomial but also that $\mathcal{A}$'s running time is polynomially bounded.

C.4 Verifiable Secret Sharing Comparison

We present a comparison of the asymptotic costs of VSS schemes in Figure 8. We assume all schemes utilize FFT for polynomial evaluation and fast interpolation [17] for polynomial interpolation where applicable. We assume all public parameters are broadcast during initialization.

D DISTRIBUTED VERIFIABLE RANDOM FUNCTIONS

D.1 Definition

In this section, we define the notion of a (t, c)-*distributed verifiable random function* (DVRF) that enables any n parties to jointly compute the output of a verifiable random function (VRF) on input message m, even in the presence of $t-1$ corrupted parties. A DVRF first runs an initial setup phase to generate a secret share and a public key for each of the n parties. Afterwards, the parties can jointly compute the output of a verifiable random function (VRF) on input m. To do this, each party will individually compute a partial evaluation on input m and each partial evaluation may be individually verified using public keys. Finally, any c partial evaluations that were successfully verified can be combined to construct the final output. We adapt our definition of DVRF with respect to the ramp threshold structure. The adversary statically corrupts at most $t-1$ parties, but at least c parties are required to generate random values. Therefore, we will assume that $n \geq c + t - 1$ to guarantee there are at least c honest parties.

A (t, c)-DVRF scheme is required to satisfy the following properties against a malicious, static adversary:

(1) **Consistency**: Any subset of c correctly verified partial evaluations for input m reconstructs to the same output.
(2) **Robustness**: An adversary corrupting at most $t-1$ parties cannot cause the reconstruction protocol to abort early or output $\perp$ on any input.
(3) **Uniqueness**: For every input m, there exists a unique output y such that reconstruction always outputs y.
(4) **Pseudorandomness**: For any input, an adversary corrupting at most $t-1$ parties cannot distinguish between the output of the reconstruction algorithm and the output of a truly random function.

For formal definitions, we point to prior works [18, 42].

Finally, we also consider (t, c)-*decentralized randomness beacons* (DRB) where n parties wish to jointly generate random values. Any c parties should reconstruct the randomness while at most $t-1$ parties cannot learn the randomness or influence the random outputs. We note that DRBs can be built from DVRFs in a straightforward way. We present formal definitions in Appendix E.

D.2 Our Construction

We show that DVRF schemes may be built from each of $\mathcal{F}$-F-VSS and $\mathcal{F}$-P-VSS. Each DVRF is obtained by running n parallel executions of a VSS scheme. Afterwards, the reconstruction algorithm can be adapted to enable verified evaluations. We start by presenting $\mathcal{F}$-F-DVRF from $\mathcal{F}$-F-VSS. The pseudocode of $\mathcal{F}$-F-DVRF may be found in Appendix D.2. We present an informal description below.

We consider a DVRF with ℓ_{in}-bit inputs m returning ℓ_{out}-bit values v. Our construction uses a prime-order cyclic group $\mathbb{G}$ with generator g, equipped with a non-degenerate bilinear function $\mathsf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ that maps pairs of elements from $\mathbb{G}$ to the target group $\mathbb{G}_T$. We also use hash H' mapping ℓ_{in}-bit strings to $\mathbb{G}$.

During Init, a random matrix $\mathbf{M}$ is selected according to a distributed generator matrix family $\mathcal{F}$. For Setup, the main idea is for each party i to select a random vector $\mathbf{x}^i$ and to verifiably share it to the other $n-1$ parties using $\mathcal{F}$-F-VSS with the matrix $\mathbf{M}$ from Init. In other words, each party i will send $s^i_j = \mathbf{M}_j \cdot \mathbf{x}^i$ to party j. At the end of Setup, all honest parties agree on a set Q of non-disqualified parties. The final secret is $\mathbf{x} = \sum_{i \in Q} \mathbf{x}^i$ and the final share for party j will be the sum of shares for all non-disqualified parties in Q, $s_j = \sum_{i \in Q} s^i_j$. Note that this still satisfies $\sum_{i \in Q} \mathbf{M}_j \cdot \mathbf{x}^i = \mathbf{M}_j \cdot \sum_{i \in Q} \mathbf{x}^i = \mathbf{M}_j \cdot s_j$. In other words, if one executes $\mathcal{F}$-F-VSS.Reconstruct with c correctly-formed shares, the output would be $\mathbf{x} = \sum_{i \in Q} \mathbf{x}^i$. Finally, we note that each party can also compute public keys $\mathsf{pk}_j = g^{s_j}$ for all non-disqualified parties $j \in Q$.

For partial evaluation on input m, each party will hash the input m to the group $\mathbb{G}$, $H'(m)$ and and return $y_j = H'(m)^{s_j}$ using its secret s_j. To verify a partial evaluation, a party can use the public key $\mathsf{pk}_j = g^{s_j}$ by checking whether $\mathsf{e}(y_j, g) = \mathsf{e}(H'(m), \mathsf{pk}_j)$. Note this succeeds iff $y_j = H'(m)^{s_j}$ as $\mathsf{pk}_j = g^{s_j}$.

Finally, we adapt the reconstruction from $\mathcal{F}$-F-VSS to enable combining partial evaluations on input m without revealing the secret $\mathbf{x}$. To do this, we execute the $\mathcal{F}$-F-VSS.Reconstruct in the exponent using $y_j = H'(m)^{s_j}$ as the shares. We execute reconstruction except for the final hash evaluation. At this point in the reconstruction algorithm, one obtains $[H'(m)^{\mathbf{x}_1}, \ldots, H'(m)^{\mathbf{x}_t}]$ where $\mathbf{x}$ was the original secret. The final random value will be $H(H'(m)^{\mathbf{x}_1}, \ldots, H'(m)^{\mathbf{x}_t})$. Using the same privacy argument from $\mathcal{F}$-SS, an adversary with $t-1$ input shares will be unable to predict the final random value without the final missing share.

Unbiasiability. Prior work [44] showed that building discrete logarithm-based distributed key generation from parallel executions of Feldman's VSS results in biased secrets. In particular, these constructions must output s and g^s. They showed that g^s is not uniformly random, but s is uniformly random. This same issue does not appear in our DVRF as we output s that is uniformly random.

Pedersen DVRF. We note a similar approach can be used to construct a DVRF using our Pedersen-based VSS, $\mathcal{F}$-P-VSS. Using the same extension to DVRF, we can obtain $\mathcal{F}$-P-DVRF. The main benefit of $\mathcal{F}$-P-DVRF is the statistical guarantees of the commitments. However, there will be no significant improvement during partial evaluation or reconstruction. So, there will be no improvement for randomness generation in DRB schemes. Thus, we omit the description of the Pedersen DVRF.

Algorithm 15 $\mathcal{F}$-F-DVRF.Init algorithm

Input: $1^\lambda, n, t, \ell_{\text{in}}, \ell_{\text{out}}$: security parameter, number of parties, privacy threshold, and input and output length.
Output: pp: public parameters.
Sample random function $H : \mathbb{G}^t \to \{0,1\}^{\ell_{\text{out}}}$.
Sample random function $H' : \{0,1\}^{\ell_{\text{in}}} \to \mathbb{G}$.
Sample coin tosses $R \leftarrow \{0,1\}^\lambda$ for $\mathcal{F}$.RandGen.
return $\mathsf{pp} \leftarrow (1, \lambda, H, H', n, t, c \leftarrow K(t), R)$.

Algorithm 16 $\mathcal{F}$-F-DVRF.Setup protocol

Input: pp, 1^λ: public parameters and security parameter.
Output: for party i, $(s_i, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$: one share and n public keys.

Dealing round (executed by each party $i \in [n]$):

Parse $\mathsf{pp} = (1^\lambda, H, H', n, t, c, R)$.
Sample random vector $\mathbf{x}^i \leftarrow \mathbb{F}^t$.
Compute and broadcast $g^{\mathbf{x}^i} = [g^{\mathbf{x}^i_1}, \ldots, g^{\mathbf{x}^i_t}]$.
Compute $(\mathsf{seed}_1, \ldots, \mathsf{seed}_n) \leftarrow \mathcal{F}.\mathsf{RandGen}(1^\lambda; R)$.
for $j \in [n]$ **do**
  Compute $\mathbf{M}_j \leftarrow \mathcal{F}.\mathsf{ExpandRow}(\mathsf{seed}_j)$.
  Compute $\mathbf{y}^i_j \leftarrow \mathbf{M}_j \cdot \mathbf{x}^i$.
  Send $s^i_j \leftarrow \mathbf{y}^i_j$ to party j over private channel.

Verification round (executed by each party $i \in [n]$):

for $j \in [n]$ **do**
  If $\mathcal{F}$-F-VSS.Verify$((i, \mathsf{seed}_i, s^j_i), g^{\mathbf{x}^j}, R) \neq 1$, broadcast a complaint for party j.

Complaint round (executed by each party $i \in [n]$):

Let C_j be the parties complaining about party $j \in [n]$.
for $j \in C_i$ **do**
  Broadcast s^i_j.
Set $Q = [n]$.
for $j \in [n]$ **do**
  If $|C_j| \geq t$, disqualify j, $Q \leftarrow Q \setminus \{j\}$ and $\mathsf{pk}_j \leftarrow \perp$.
for $j \in Q$ **do**
  for $k \in C_j$ **do**
    If $\mathcal{F}$-F-VSS.Verify$((k, s^j_k, \mathsf{seed}_k), g^{\mathbf{x}^j}, R) \neq 1$, disqualify j, $Q = Q \setminus \{j\}$ and $\mathsf{pk}_j \leftarrow \perp$.
Set $s_i \leftarrow \sum_{j \in Q} s^j_i$.
for $j \in Q$ **do**
  Compute $\mathbf{M}_j \leftarrow \mathcal{F}.\mathsf{ExpandRow}(\mathsf{seed}_j)$.
  Set $\mathsf{pk}_j \leftarrow \prod_{a \in [t]} \prod_{j \in Q} (g^{\mathbf{x}^j_a})^{\mathbf{M}_{j,a}}$. ▷ $\mathsf{pk}_j = g^{s_j}$

Algorithm 17 $\mathcal{F}$-F-DVRF.PartialEval algorithm

Input: m, s: input and share.
Output: y: partial output.
return $y \leftarrow H'(m)^s$.

E DECENTRALIZED RANDOMNESS BEACONS

We consider (t, c)-*decentralized randomness beacons* (DRB) where n parties wish to jointly generate random values. In particular, the outputs should be computationally indistinguishable from random

Algorithm 18 $\mathcal{F}$-F-DVRF.PartialVerify algorithm

Input: m, y, pk: input, partial output and public key.
Output: $b \in \{0,1\}$: verification output.
return 1 iff $\mathsf{e}(H'(m), \mathsf{pk}) = e(y, g)$.

Algorithm 19 $\mathcal{F}$-F-DVRF.Reconstruct algorithm

Input: $m, (y_j, \mathsf{pk}_j)_{j \in J}$: input, partial evaluations and public keys.
Output: y: output.
Compute subset $I \subset J$ such that $|I| = c$ and, for all $j \in I$, $\mathcal{F}$-F-DVRF.PartialVerify$(y_j, \mathsf{pk}_j) = 1$.
Execute $\mathcal{F}$-SS.Reconstruct in the exponent until the final hash function to obtain $H'(m)^{\mathbf{x}_1}, \ldots, H'(m)^{\mathbf{x}_t}$.
return $y \leftarrow H(H'(x)^{\mathbf{x}_1}, \ldots, H'(x)^{\mathbf{x}_t})$.

even in the presence of a malicious adversary that corrupts $t-1$ parties. DRB schemes operate in rounds where the n parties will generate fresh random outputs in each round. We denote the output of the r-th round as Ω_r. Once again, we adapt DRB to ramp threshold structures by requiring c parties to generate random values each round. Therefore, we will assume that $n \geq c + t - 1$ so there will be at least c honest parties.

A (t, c)-DRB scheme in the synchronous setting against malicious, static adversaries must satisfy these properties:

(1) **Unbiasability**: An adversary corrupting at most $t-1$ parties cannot distinguish between the outputs of the DRB protocol with only honest parties and the DRB protocol executed with the corrupted parties.
(2) **Liveness**: An adversary corrupting at most $n - c$ parties cannot cause the DRB scheme to abort early or output $\Omega_r = \perp$ in any round r.
(3) **Unpredictability**: Given the output of the r-th round Ω_r, an adversary corrupting at most $t-1$ parties cannot predict any property about the output of the r'-th round $\Omega_{r'}$ for any round $r' \geq r + 1$.

For formal definitions, we refer readers to [25].

DRB and DVRF. We note that the definitions of decentralized random beacons (DRB) and distributed verifiable random functions (DVRF) are similar. In fact, there is a straightforward to build DBR schemes using a DVRF where the output of round r is the evaluation of the DVRF on input r used in prior works [25, 42].

Other DRB Protocols. There are other DRB schemes using primitives other than DVRFs. See [25] for more comparisons. As these DRB schemes still rely on Shamir's scheme, we believe our techniques may be used to improve their computational cost as well. However, we leave this to future work.

Unpredictability with Network Delay. Prior works have considered unpredictability definitions with network delays of Δ. See [25] for full definitions.

- α-**intra-unpredictability**: An adversary cannot predict the output of round r at α time units before the end of round r.
- β-**inter-unpredicability**: An adversary cannot predict the output of round $r + \beta'$ for any $\beta' \geq \beta$ at the end of round r.

It is straightforward to show that $\mathcal{F}$-F-DVRF satisfies the notions of $O(\Delta)$-intra-unpredictability and 1-inter-predictability in the presence of Δ network delay. This is identical to other DVRF-based DRB schemes [42, 48].

F NON-HOMOMORPHIC APPLICATIONS

Federated Learning. A significant benefit of using RB-SS is that federated learning may use larger cohorts without sacrificing efficiency. Current systems use cohorts of size in the 100-10,000 [15] where share reconstruction is performed by surviving users in each cohort. RB-SS would enable scaling to larger cohorts without share reconstruction becoming a bottleneck. In turn, larger cohorts significantly reduce the impact of dropouts for federated learning. To see this, suppose there are n users, 25% dropout rates and we wish to report results when no more than half users dropout from any cohort. If there are $n/100$ cohorts of size 100, 51 users can drop from $g = \lfloor n/(4 \cdot 51) \rfloor$ different cohorts. As a result, the contributions of $100 \cdot g$ users are lost. In contrast, if all n users were in one cohort, no individual's contribution would be lost. We use RB-SS to scale to larger cohorts (see Section 6.2 for the experimental evaluation).