Communication-efficient, Fault Tolerant PIR over Erasure Coded Storage

Andrew Park*, Trevor Leong*, Francisco Maturana*, Wenting Zheng*, K. V. Rashmi *
*Carnegie Mellon University
andrewpark@cmu.edu, tmleong@andrew.cmu.edu, fmaturan@cs.cmu.edu, wenting@cmu.edu, rvinayak@cs.cmu.edu

Abstract—**Private information retrieval (PIR) is a technique for a client to retrieve an item from a public database without revealing to an adversarial server the item that was queried. While multi-server PIR has been well-studied in order to obtain better communication and computation relative to single-server schemes, there are far fewer fault-tolerant PIR schemes which can remain functional even in the presence of malicious adversaries. In this paper, we present a solution that combines techniques from both the cryptography and information theory communities to design robust PIR protocols that obtain better computation, communication, and storage compared to prior state-of-the-art schemes. Our results show that our PIR protocols achieve up to $9.1\times$ lower latency, at least $39.2\times$ less total communication, and up to $7.3\times$ less computation than the state-of-art robust PIR protocols for a database 4GB in size and can withstand two malicious servers, and continually outperform the robust PIR baselines for a variety of parameter configurations and failure scenarios.**

1. Introduction

Private information retrieval (PIR) is a set of cryptographic techniques that allows clients to retrieve values from an untrusted data storage such that the storage servers do not learn anything about the value being retrieved. PIR is an important fundamental primitive for enabling many applications such as anonymous messaging [16], [23], private database querying [50], encrypted search [17], and privacy-preserving advertising [42].

One particular category of protocols that has garnered recent interest are *multi-server* PIR schemes [10], which avoid expensive cryptographic operations like homomorphic encryption required of single-server PIR schemes by distributing trust over multiple servers. This allows for protocols that are more computation-efficient compared to single-server approaches. When considered for practical usage, however, existing multi-server approaches have one major drawback. Namely, most existing multi-server PIR protocols are not robust to either fail-stop or Byzantine servers. Specifically, though the protocols guarantee that the adversarial servers do not learn anything about the query, they do not necessarily prevent these servers from returning incorrect results. Prior works [17] have used MACs to guarantee integrity of the result. However, such solutions require additional burden on the client to store and manage a secret key. Additionally, even if the client does catch malicious behavior, the client will still not be able to recover her desired result.

Existing fault-tolerant PIR protocols, however, also contain drawbacks when considered for practice. For example, the works of Goldberg [28] and Devet et al. [19] provide fault-tolerance at the cost of communication inefficiency. In particular, both of these schemes require large queries that are linear in the size of the database, and such inefficiency becomes a big limiting factor in scaling to larger databases. An orthogonal line of work by Woodruff et al. [52], is communication-efficient, but is only able to provide this efficiency at the cost of more expensive server computation.

Other multi-server PIR protocols developed in the information theory community have leveraged *erasure-coding* to not only design robust PIR schemes ([1], [3], [49], [53]), but also minimize server storage and download cost ([2], [4], [9], [25], [26], [37], [46], [48], [54]). Erasure coding is a powerful tool for providing error correction (even in the Byzantine setting) without resorting to full data replication, thus significantly saving on server-side storage and computation. However, erasure coded robust PIR schemes suffer from the same drawbacks as the schemes in [19], [28], namely linear-sized PIR queries that prevent practical implementations.

In this paper, we design novel PIR protocols that achieve better storage, communication, and computation overheads compared to the prior state-of-art robust PIR protocols. To achieve this, we first leverage *distributed point functions* (DPF [11], [12], [27]), a lightweight cryptographic primitive which can be used in PIR to provide significant query bandwidth savings by compressing PIR queries. Next, we compose DPF with erasure coding as an effective means of achieving fault tolerance, because it allows for significant storage and computation savings at the cost of modestly more servers.

Because our schemes require less storage, communication, and computation compared to prior works, but require additional servers, we believe our new schemes could have use in many *decentralized* applications, such as decentralized search [38], anonymous communication networks [31], and decentralized storage [51]. These scenarios typically have a large number of users whose storage and compute capabilities are smaller than a cloud provider's. In addition, fault tolerance against Byzantine failures will allow our schemes to remain functional even in the presence of offline or malicious servers.

In order to design our schemes, the first challenge we need to tackle is to devise new ways to make existing DPF designs fault tolerant. Therefore, we design three new DPF protocols that can tolerate fail-stop faults. First, we extend the two-server, tree-based DPF scheme [11] to the multi-server setting that tolerates $p-2$ out of p failures by carefully replicating control bits and increasing the number of correction words without leaking extra information. Next, we leverage *covering designs* to the XOR-sharing based multi-server DPF scheme ([12]) in order to improve communication. This significantly improves upon a naive approach of directly using a more common approach such as replicated secret sharing (RSS), because a big issue of applying RSS is the extra communication overhead that varies based on the total number of servers and the security threshold. Finally, we design a third, fault-tolerant *information-theoretic* DPF scheme based on Shamir secret sharing. The base design requires a large number of servers to serve queries when the number of items in the database is increased. To loosen this restriction, we use Hermite interpolation to decrease the number of servers by using more computation and more response overhead.

The second part of our design focuses on composing our new fault-tolerant DPF designs with erasure-coded storage in order to design three new end-to-end, communication-efficient and fault-tolerant PIR protocols. Erasure-coding is a powerful technique that not only provides fault-tolerance, but also leads to less storage and computation compared to a replicated scheme. However, our second challenge is that directly composing our new DPF schemes with erasure-coded storage is challenging because DPF-based PIR schemes do not directly work over erasure coding due to DPF's *additive reconstruction*. This reconstruction implies a PIR protocol that computes independent dot products between the database values and the function output shares followed by a final summation of the results at the client, which requires each server to hold replicated copies of the database. When the database is erasure coded across multiple servers, however, the servers no longer hold the same items. To address this challenge, we design new techniques to compose our fault-tolerant DPF schemes by transforming the outputs into the appropriate algebraic structures that can be used to query erasure coded storage.

Our techniques leverage Reed-Solomon codes in two different ways: either within each item or across items. Both methods of erasure-coding were studied in the PIR context in [2], [26], [48], [49] using linear-sized queries, and we demonstrate how to transform our sublinear tree-based DPF and Shamir-based DPF to be compatible with these approaches. Second, we observe that an additive DPF scheme that is made fault-tolerant using covering designs can be composed with the erasure-code is applied across items. Therefore, we develop a third PIR protocol that integrates the covering design-based DPF protocol with an XOR-based MDS encoded storage. Finally, we also take advantage of erasure code's properties to support malicious servers that attempt to return wrong results, and use these properties to recover from Byzantine faults.

We implement and evaluate both the baseline schemes

		(1)	(2)	(3)
Prior Work	Boyle et al. DPF Tree Based PIR [10]	✗	✓	✗
	Boyle et al. Multiparty Based PIR [10]	✗	✓	✗
	Devet et al. Robust PIR [19]	✗	✗	✓
	Tajeddine et al. Coded PIR [49]	✓	✗	✓
	Woodruff et al. Robust PIR [52]	✗	✓	✓
Ours	Robust DPF Tree Based PIR (Section 5.2)	✓	✓	✓
	Shamir DPF Based PIR (Section 5.3)	✓	✓	✓
	Covering Design Based PIR (Section 5.4)	✓	✓	✓

TABLE 1: Comparison with prior works: (1) Erasure coded storage (2) Sublinear query size (3) Robustness to fail-stop and Byzantine failures

and our solutions with a variety of parameters and scenarios. In terms of storage, our schemes achieve better per-server overhead, and achieves better *total storage* when compared to the prior state of art replicated robust PIR baselines (up to 72% less storage). Second, in terms of computation, our schemes achieve up to 9.1× better per-server computation, and up to 7.3× better total computation. In addition, when testing our protocols in failure scenarios, our protocols have very minimal overhead (∼22 ms for our Shamir-based scheme, <1 ms for the other two), and achieve up to 9.1× less latency compared to the state-of-art erasure coded PIR protocols which use linear-sized queries. As a downside, our schemes require a larger number of servers when compared to a replicated baseline because of our use of erasure-coding. Overall, our solutions ultimately suggest that combining erasure coding with cryptographic solutions in the context of PIR seem to be promising, demonstrating that we can indeed achieve both communication efficiency and fault tolerance.

2. Background

2.1. Preliminaries

We first describe the notation we use for the rest of the paper. We define p to be the number of servers that store a (potentially encoded) public database of size $N = 2^n$. The parties are enumerated from 1 to p. We assume that the adversary can compromise up to t servers. We assume that up to r servers can be non-responsive, while up to b servers are Byzantine and can fail arbitrarily.

For our MDS code, we use k to represent the reconstruction threshold of the $[p, k]$-MDS code, and G to represent the k-by-p generator matrix associated with the MDS code. The values range over some finite field $\mathbb{F}$.

We represent the i'th item in a database X as $X[i]$, where $X[i]$ comprises m field elements. Each $X[i]$ can also be written as k *shards*, $X[i]_1 \| X[i]_2 \| \cdots \| X[i]_k$, where each shard $X[i]_j$ comprises $\frac{m}{k}$ field elements. Finally, we use $\vec{e_i}$ to represent the i'th standard basis vector, and $\vec{r}$ to represent a uniformly random vector over a field $\mathbb{F}$.

2.2. Secret Sharing

A (p, t)-secret sharing scheme, is a protocol that allows a party to share a secret s in an arbitrary field $\mathbb{F}$ to p parties such that no collusion of t parties can learn anything about s, while any subset of $t+1$ or more (honest) parties are able to reconstruct the secret.

One standard technique to do this is *Shamir secret sharing* [45]. The client generates a random degree t polynomial $g(x) = s + \sum_{i=1}^{t} r_i x^i, r_i \in \mathbb{F}$ and gives each party j the value $g(j) \in \mathbb{F}$. Any subset of t parties cannot learn anything about s because there exists a degree t polynomial passing through their t points and any possible value of s, while $t+1$ or more parties can use interpolation on the function to recover the value $g(0)$.

Another (p, t)-secret sharing scheme is *replicated secret sharing* (RSS) [34]. Let S be the set of all p parties. Given p parties who wish to withstand against a collusion of up to t, RSS shares a secret $s \in \mathbb{F}$ by splitting the secret s into $k = \binom{p}{t}$ shares in any arbitrary field $\mathbb{F}$. We denote each share as $s_i \in \mathbb{F}$ with the property that $s = \sum_{i=1}^{k} s_i$. The shares are distributed by enumerating through all subsets A_i of size t and giving s_i to all parties *not* in A_i. Reconstruction requires a subset of $t+1$ parties since any subset of R of size $t+1$ must overlap with all subsets $S \setminus A_i$, which means that R has all s_i shares. Security is maintained because if a subset T of size less than or equal to t is corrupted, this subset must be contained within some A_i, which implies that it does not hold share s_i and thus cannot reconstruct the entire secret s.

2.3. Distributed Point Function

A distributed point function (DPF) ([11], [12], [27]) is a technique that allows a client to split a point function $f(x)$ (functions that have one nonzero output at a special index α) into keys $k_1, \ldots, k_p$, such that any subset of keys reveals nothing about $f(x)$, while the aggregation of the key evaluations at a point x result in $f(x)$. At a high level, this means that the keys represent secret shares of a vector size $N = 2^n$, where all the elements except the element at index i is equal to 0. A DPF is defined by the following set of algorithms:

- $\mathsf{Gen}(f) \rightarrow (\mathcal{K}_1, \ldots, \mathcal{K}_p)$ which on input $f \in \{0,1\}^*$, outputs a p-tuple of keys.
- $\mathsf{Eval}(i, \mathcal{K}_i, x) \rightarrow y_i$ is a polynomial-time evaluation algorithm, which on input $i \in [p]$ (party index), $\mathcal{K}_i$ (key corresponding to party i) and $x \in \{0,1\}^n$, outputs an element y_i such that $\sum_{j=1}^{p} \mathsf{Eval}(j, \mathcal{K}_j, x) = f(x)$

As an example of how the DPF is useful for the PIR setting, assume we have $p = 2$ servers, each with a replicated copy of the database X. When retrieving an item $X[i]$, the client generates $\mathsf{DPF.Gen}(1^\lambda, f) \rightarrow (\mathcal{K}_1, \mathcal{K}_2)$, where $f(x)$ if the point function $f(x) = 1$ if $x = i$, $f(x) = 0$ otherwise. After receiving the key $\mathcal{K}_j$, server j evaluates the key at every index and sends back the response $\sum_{m=1}^{N} \mathsf{Eval}(j, \mathcal{K}_j, m) \cdot X[m]$ to the client. Adding the responses together allows the client to recover $\sum_{m=1}^{N} (\mathsf{Eval}(1, \mathcal{K}_1, m) + \mathsf{Eval}(2, \mathcal{K}_2, m)) \cdot X[m] = \sum_{m=1}^{N} f(m) \cdot X[m] = X[i]$ as desired.

2.4. Maximum Distance Separable (MDS) Codes

A $[p, k]$-MDS code encodes k symbols of $\mathbb{F}$ into p symbols of $\mathbb{F}$, such that the original k symbols can be recovered even if some of the p symbols are erased or corrupted. The most common type of MDS code is the Reed-Solomon code [39], which interprets the k data symbols as coefficients of a $(k-1)$-degree polynomial, and encodes data by evaluating this polynomial at p distinct points in $\mathbb{F}$. Alternatively, a $[p, k]$ Reed-Solomon code (or any other linear MDS code) can be described through a *generator matrix* $G \in \mathbb{F}^{k \times p}$ that encodes $m \in \mathbb{F}^k$ as mG, and has the property that the submatrix formed by any k columns of G is invertible. Several efficient algorithms exist for decoding Reed-Solomon codes [39].

For our schemes, we apply erasure codes to the unencoded files in two different ways - within each item, or across items. When erasure-coding within each item, each item f is split into k parts $f = f_1, \ldots, f_k$ and the generator matrix is applied to get each erasure-coded item. For example, for $p = 3, k = 2$, the 3 erasure coded items are $f_1, f_1 + f_2, f_1 + 2f_2$. When erasure-coding across items, we shard the original database into k parts $X = X_1 \| \cdots \| X_k$, where each X_i contains $\frac{N}{k}$ items. We can then apply the generator matrix of the $[p, k]$-MDS code to these shards to get the contents of each of the p servers. As a concrete example, consider a $[3, 2]$-MDS code applied to a database $X = X_1 \| X_2$. The encoded databases would then be $\mathcal{X}_1 = X_1, \mathcal{X}_2 = X_2, \mathcal{X}_3 = X_1 \oplus X_2$.

3. Threat Model and Security Guarantees

3.1. Fault-tolerant DPF definitions

At a high level, our security definition guarantees that a static adversary who compromises no more than t out of p parties will not learn anything about the secret query index i. This generalizes the security notion than the ones presented in [11], [12], allowing us to design robust querying schemes that will be presented in the following sections.

Definition 3.1 (Point Function)**.** For $\alpha \in \{0,1\}^n, \beta \in \{0,1\}^m$, the point function $f_{\alpha,\beta} : \{0,1\}^n \rightarrow \{0,1\}^m$ is defined such that $f(\alpha) = \beta$, and $f(\alpha') = 0$ for all $\alpha' \neq \alpha$.

Definition 3.2 (Fault-tolerant DPF:Syntax)**.** A (p, t, r)-robust distributed point function is a tuple of algorithms $(\mathsf{DPF.Gen}, \mathsf{DPF.Eval}, \mathsf{DPF.Rec})$:

- $\mathsf{DPF.Gen}(1^\lambda, \alpha, \beta) \rightarrow (\mathcal{K}_1, \ldots, \mathcal{K}_p)$ which on input 1^λ (security parameter) and parameters of point function $f_{\alpha,\beta} : \{0,1\}^n \rightarrow \{0,1\}^m$ outputs a p-tuple of keys.
- $\mathsf{DPF.Eval}(i, \mathcal{K}_i, x) \rightarrow y_i$, which on input $i \in [p]$ (party index), $\mathcal{K}_i$ (key corresponding to party i), and $x \in \{0,1\}^n$, outputs a group element $y_i \in \{0,1\}^m$
- $\mathsf{DPF.Rec}((i_1, y_1), \ldots, (i_{t'}, y_{t'})) \rightarrow \mathsf{res}$, which on input of $t' \geq p - r$ tuples i (party index) and y_i (party output share), is able to recover the desired value res
  - $\mathsf{DPF.Gen}(1^\lambda, \alpha, \beta) \rightarrow (\mathcal{K}_1, \ldots, \mathcal{K}_p)$

– $\mathsf{DPF.Eval}(i, \mathcal{K}_i, x) \to y_i$
– $\mathsf{DPF.Rec}((i_1, y_1), \ldots, (i_{t'}, y_{t'})) \to \mathsf{res}$

Definition 3.3 (Fault-tolerant DPF:Security)**.** Let $\mathcal{F}$ be a function family and let Leak be a function specifying the allowable leakage for a function $f \in \mathcal{F}$. A (p, t, r)-robust DPF is a tuple of algorithms $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Rec})$ that satisfies the following requirements:

- **Correctness**: For all point functions $f_{\alpha,\beta} \in \mathbb{F}$ describing $f_{\alpha,\beta} : \{0,1\}^n \to \{0,1\}^m$ and for all $x \in \{0,1\}^n$, if $(\mathcal{K}_1, \ldots, \mathcal{K}_m) \leftarrow \mathsf{Gen}(1^\lambda, \alpha, \beta)$, for all subsets $S \subset [p]$ such that $|S| \geq p - r$, then $\mathbb{P}[\mathsf{Rec}(\{(i, \mathsf{Eval}(i, \mathcal{K}_i, x)) : i \in S\}) = f(x)] = 1$.
- **Secrecy**: For all sets of corrupted parties $C \subset [p]$ of size $\leq t$, there exists a PPT algorithm Sim such that for all functions $f \in \mathcal{F}$, the outputs of the following experiments Real and Ideal are indistinguishable:
  – $\mathsf{Real}(1^\lambda, \alpha, \beta) : (\mathcal{K}_1, \ldots, \mathcal{K}_m) \leftarrow \mathsf{Gen}(1^\lambda, \alpha, \beta)$
  – $\mathsf{Ideal}(1^\lambda) :$ Output $\mathsf{Sim}(1^\lambda, \mathsf{Leak}(f))$

3.2. Fault-tolerant PIR definitions

We generalize prior multi-server PIR definitions to a *threshold* version, where the index remains hidden even when t out of p servers are corrupted. In exchange, our definition achieves correctness even when r of the servers are *non-responsive* and b servers are *Byzantine*, where r and b are pre-determined.

Definition 3.4 (Fault-tolerant PIR: Syntax)**.** A (p, t, r, b)-robust coded PIR protocol P is a tuple of algorithms $(\mathsf{PIR.Encode}, \mathsf{PIR.Query}, \mathsf{PIR.Eval}, \mathsf{PIR.Rec})$:

- $\mathsf{PIR.Encode}(X) \to (\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_p)$, which on input of an unencoded database X, outputs p erasure coded databases $\mathcal{X}_1, \ldots, \mathcal{X}_p$.
- $\mathsf{PIR.Query}(p, t, X, \alpha) \to (\sigma_1, \ldots, \sigma_p)$, which on input number of parties p, security threshold t, and index in the original DB α, outputs query vectors $(\sigma_1, \ldots, \sigma_p)$.
- $\mathsf{PIR.Eval}(j, \sigma_j, \mathcal{X}_j) \to r_j$, which for party index j and their corresponding encoded database and query vector, outputs a response r_j.
- $\mathsf{PIR.Rec}((j_1, r_{j_1}), \ldots, (j_{t'}, r_{j_{t'}})) \to X[i]$, which on input of $t' \geq p - r$ tuples j (party index), r_j (party output), is able to recover the value $X[i]$.

We wish our protocols to satisfy the following notions of correctness and security. Note that our correctness definition guarantees that the output is correct even in the case where all of the parties are not behaving as expected.

Definition 3.5 (Fault-tolerant PIR: Security)**.**

- **Correctness** For all databases X and set of responses $\mathsf{res} = \{(j, \mathsf{r}_j)\}$, $|\mathsf{res}| \geq p - r$, if for at most b responses $r_j \neq \mathsf{Eval}(j, \sigma_j, S_j))$, then:
$$\mathbb{P}\left[\begin{smallmatrix} (S_1,\ldots,S_p)\leftarrow\mathsf{Encode}(X) \\ (\sigma_1,\ldots,\sigma_p)\leftarrow\mathsf{Query}(p,X,i) \\ \mathsf{Rec}(\{(j,r_j)\in\mathsf{res}\})=X[i] \end{smallmatrix}\right] = 1$$
- **Secrecy** For all subsets of corrupted parties $C \subset [p]$ of size $\leq t$, there exists a PPT algorithm Sim such that for all databases X, $|X| = N$, the outputs of the following experiments are computationally indistinguishable:
  – $\mathsf{Real}(1^\lambda, p, X, i) : (\sigma_1, \ldots, \sigma_p) \leftarrow \mathsf{Query}(1^\lambda, p, X, i)$
  – $\mathsf{Ideal}(1^\lambda) :$ Output $\mathsf{Sim}(1^\lambda, p, N)$

4. Fault Tolerant DPF Schemes

In this section, we present three new fault-tolerant DPF constructions. Existing DPF designs ([11], [12], [27]) are not robust to failures because all servers need to respond in order to reconstruct the result. However, our three schemes are designed to be robust to non-responsive (r) failures.

4.1. Multi-server DPF with no collusion

We first present our first fault-tolerant, tree-based DPF scheme based on [12] with collusion threshold $t = 1$. Recall that the goal during the $\mathsf{DPF.Gen}$ phase in [12] is to design keys that represent secret shares of a vector of size $N = 2^n$ with the value β at the special index α, such as $\vec{r}$ and $\vec{r} + \beta \cdot \vec{e}_\alpha$. Each party's DPF key is a binary tree of depth $\log N$, where the root node of the tree is a random seed of length λ. It is then repeatedly input into a length-doubling pseudorandom generator (PRG) to achieve a GGM-style pseudorandom function (PRF) [29]. Each leaf value is the output of the evaluation path corresponding to an input i. In order to transform this into a distributed point function, Boyle et al. construct mechanisms to achieve the following invariants. Each node has a label, which consists a seed (similar to the GGM tree) and an extra control bit. The α input corresponds to a special evaluation path. Outside the evaluation path, the labels on the two tree are identical. Since a PRG will evaluate identical seeds to the same value, two equal roots will cause their entire subtrees to evaluate to be the same. On the special evaluation path, however, the seeds are indistinguishable from random, and the two control bits are different. The control bits decide whether correction words CW_l is applied to each node, and will only be applied if the node is on the special path. There is a unique correction word CW_l per level of the tree, which is used to trigger nodes off the special path to become identical with the same control bits, and those in the special path to become pseudorandom with different control bits.

To extend this to the multi-server setting where $p > 2$, our goal is to extend the above construction in order to design p logarithmic-sized keys that represent the linear sized vectors $\vec{r}, \vec{r}+\beta\cdot\vec{e_i}, \cdots, \vec{r}+(p-1)\beta\cdot\vec{e_i}$. Note that any 2 out of p of these vectors is sufficient to recover the value β. Our key insight is that, under the assumption of no collusion, it is possible to maintain the invariant across more than two parties by replicating one of the control bits across more servers. In order to extend to p parties with potential failures of up to $p - 2$ (this is the worst possible situation in which our scheme can still reconstruct the result), we additionally need to make use of $p - 1$ control bits for each party, as well as $p - 1$ corresponding correction words. Each set of control bits has following invariant: the ith control bits are the same

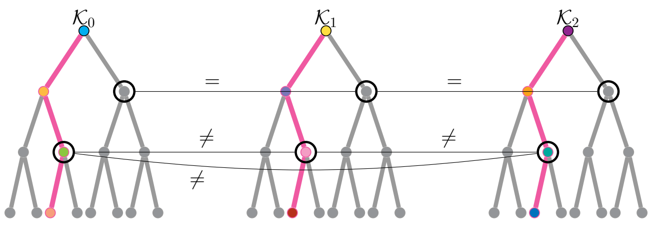


Figure 1: DPF Tree construction. The grey lines represent equal values across all 3 keys, while the pink line represents the special path. Correction words are applied at each node.

across all parties for non-special nodes; for special nodes, they should XOR to 1 for each pair of (P_i, P_j) servers such that $i \neq j$. Essentially, the ith control bits (and their corresponding correction words) are able to tolerate any $p-2$ failures, conditioned on the fact that server i does not fail. Therefore, given $p-1$ unique control bit/correct word sets, we can recover from up to any $p-2$ failures. We construct each of our $p-1$ correction words using the same method as Boyle et al. between key 0 and key i. At the last level of the tree, all non-special leaves will hold the same seed and control bits, while the special index will hold random seeds and control bits maintaining the invariant discussed above. We refer the reader to Figure 1, which demonstrates the invariant among each of the p keys. The pink line represents the special path, where each of the seeds are random to the server. As the server evaluates down the tree, the nodes off the special path are corrected such that they are all equal to each other. We present the full algorithm in Algorithm 1.

4.2. Multi-Server DPF using Covering Designs

Our second construction utilizes a key distribution scheme such as RSS to design a multi-server DPF. This allows us to transform a non-fault tolerant $(q, q-1)$-multi-server DPF (such as the one in [10]) into a fault tolerant version by replicating the DPF keys across multiple servers. We refer the reader to [10] for the specifics of their $(q, q-1)$-multi-server DPF construction, but at a high level, their DPF keys consists of two parts: $\sqrt{N} \cdot 2^{q-1}$ *seeds* and 2^{q-1} *correction words*. This means that the size of the key is sublinear in the size of the database, but exponential with the number of keys q.

To convert this non-robust scheme into a fault tolerant version, we increase the number of parties that receive each key. In this way, if a party fails to respond, then we can still retrieve the value associated with that key from another party. In particular, in our base scheme, the client runs the DPF.Gen phase of this protocol to generate $q = \binom{p}{t}$ keys $\mathcal{K}_1, \cdots, \mathcal{K}_q$ and distribute them to the servers according to RSS, as described in Section 2.2. The expansions of the individual q keys $\vec{r_1}, \cdots, \vec{r_q}$ satisfy $\sum_{i=1}^{q} \vec{r_i} = \vec{e_\alpha}$.

During evaluation, each server sends back one response for each key it receives. If the client queries enough parties to cover all q keys, then she can use the result to reconstruct her desired result. Any subset of t parties will not have access to one of the DPF keys due to the security of RSS.

Algorithm 1 Fault-tolerant DPF scheme with no collusion

```
DPF.Gen$(1^\lambda, \alpha, \beta) \rightarrow (\mathcal{K}_0, \ldots, \mathcal{K}_{p-1})$:
 1: Parse $\alpha = \alpha_1 \| \alpha_2 \| \ldots \| \alpha_n \in \{0,1\}^n$
 2: Sample random seeds $s_0^{(0)}, \ldots, s_{p-1}^{(0)} \leftarrow \{0,1\}^\lambda$
 3: for $i = 0, \ldots, p-1$ and $j = 1, \ldots, p-1$ do
 4:     Set $t_{i,j}^{(0)} = 1$ if $i = j$, 0 otherwise.
 5: for $i = 1, \ldots, n$ do
 6:     for $j = 0, \ldots, p-1$ do
 7:         $s_j^L \| \{t_{j,k}^L\}_{k=1}^{p-1} \| s_j^R \| \{t_{j,k}^R\}_{k=1}^{p-1} \leftarrow G(s_j^{(i-1)})$
 8:     $(\mathsf{Keep}, \mathsf{Lose}) \leftarrow \begin{cases} (L, R), & \text{if } \alpha_i = 0 \\ (R, L), & \text{if } \alpha_i = 1 \end{cases}$
 9:     for $k = 1, \ldots, p-1$ do
10:         $s_{CW[k]} \leftarrow s_0^{\mathsf{Lose}} \oplus s_k^{\mathsf{Lose}}$
11:         for $m = 1, \ldots, p-1$ do
12:             $t_{CW[k],m}^L \leftarrow \begin{cases} t_{0,m}^L \oplus t_{k,m}^L \oplus \alpha_i \oplus 1, & \text{if } k = m \\ t_{0,m}^L \oplus t_{k,m}^L, & \text{otherwise} \end{cases}$
13:             $t_{CW[k],m}^R \leftarrow \begin{cases} t_{0,m}^R \oplus t_{k,m}^R \oplus \alpha_i, & \text{if } k = m \\ t_{0,m}^R \oplus t_{k,m}^R & \text{otherwise} \end{cases}$
14:         $CW[k]^{(i)} \leftarrow s_{CW[k]} \| \{t_{CW[k],m}^L\}_{m=1}^{p-1} \| \{t_{CW[k],m}^R\}_{m=1}^{p-1}$
15:     $CW^{(i)} \leftarrow \{CW[k]^{(i)}\}_{k=1}^{p-1}$
16:     for $b = 0, \ldots, p-1$ do
17:         $s_b^{(i)} \leftarrow s_b^{\mathsf{Keep}} \bigoplus_{k=1}^{p-1} t_{b,k}^{(i-1)} \cdot s_{CW[k]}$
18:         $t_{b,m}^{(i)} \leftarrow t_{b,m}^{\mathsf{Keep}} \bigoplus_{k=1}^{p-1} t_{b,k}^{(i-1)} \cdot t_{CW_k,m}^{\mathsf{Keep}}$ for $m=1, \ldots, p-1$
19: for $k = 1, \ldots, p-1$ do
20:     $CW[k]^{(n+1)} \leftarrow (-1)^{t_{k,k}^{(n)}} [k\beta - \mathsf{Conv}(s_0^{(n)}) + \mathsf{Conv}(s_k^{(n)})]$
21: $CW^{(n+1)} \leftarrow \{CW[k]^{(n+1)}\}_{k=1}^{p-1}$
22: $\mathcal{K}_j \leftarrow s_j^{(0)} \| CW^{(1)} \| \ldots \| CW^{(n+1)}$ for $j = 0, \ldots, p-1$
23: return $(\mathcal{K}_0, \ldots, \mathcal{K}_{p-1})$

DPF.Eval$(i, \mathcal{K}_i, x)$:
 1: Parse $x = x_1 \| \ldots \| x_n$
 2: For $j = 1, \ldots, p-1$, let $t_j^{(0)} = 1$ if $j = i$, 0 otherwise.
 3: Parse $\mathcal{K}_i = s^{(0)} \| CW^{(1)} \| \ldots \| CW^{(n+1)}$
 4: for $j = 1, \ldots, n$ do
 5:     Parse $CW^{(j)} = \{CW[k]\}_{k=1}^{p-1}$
 6:     $\tau^{(j)} \leftarrow G(s^{(j-1)}) \bigoplus_{k=1}^{p-1} t_k^{(j-1)} \cdot CW[k]$
 7:     Parse $\tau^{(j)} = s^L \| \{t_k^L\}_{k=1}^{p-1} \| s^R \| \{t_k^R\}_{k=1}^{p-1}$
 8:     if $x_j = 0$ then $s^{(j)} \leftarrow s^L$, $\{t_k^{(j)}\}_{k=1}^{p-1} \leftarrow \{t_k^L\}_{k=1}^{p-1}$
 9:     else $s^{(j)} \leftarrow s^R$, $\{t_k^{(j)}\}_{k=1}^{p-1} \leftarrow \{t_k^R\}_{k=1}^{p-1}$
10: Parse $CW^{(n+1)} = \{CW[k]\}_{k=1}^{p-1}$
11: return $\mathsf{Conv}(s^{(n)}) + \sum_{k=1}^{p-1} t_k^{(n)} \cdot CW[k]$

DPF.Rec$((i_1, y_1), \cdots, (i_t, y_{t'}))$:
 1: return IP$((i_1, y_1), \cdots, (i_t, y_{t'}))$
```

Reducing key size with covering designs

One big drawback to RSS is its high communication cost. Directly using RSS requires that each server receive $O(p^t)$ keys per query, where each key is of size $O(2^{\binom{p}{t}}\sqrt{N})$. This quickly becomes prohibitively expensive since the cost grows exponentially in the number of keys generated. We therefore introduce an optimization using *covering designs* [30], which were previously used in the context of secure computation [7]. In our setting, we can use covering designs to reduce the number of total keys generated, which also reduces the concrete key size for each party.

A $(p, m, t)-$covering design is a collection of q subsets $A_1, \ldots, A_q$, each of size m, such that for any subset $T \subset [p]$ of size t, there exists an A_i such that $T \subseteq A_i$. The goal of covering designs is to minimize q, the total number of subsets, and we let $C(p, m, t)$ to denote the smallest possible q achieved in these covering designs. Our insight is to increase the *covering size* m and exploit a gap between m and the collusion threshold t to allow each server to receive fewer keys per query. While $C(p, t, t) = \binom{p}{t}$ is precisely the setting of RSS, using $m > t$ gives way for $C(p', m, t) < \binom{p}{t}$. For example, for $p = 6, t = 2$, RSS requires generating $q = \binom{5}{2} = 15$ keys, while using a $(7, 3, 2)$-covering design only requires $q = 7$.

Covering designs can thus be used to improve communication efficiency of RSS. Given a (p, m, t)-covering design that achieves the minimum $q = C(p, m, t)$, the client runs the DPF.Gen protocol to create q key shares and distribute each $\mathcal{K}_i$ to $[p]\backslash A_i$. The collusion threshold is t since a subset T of this size must be contained in some set A_j, and thus must not hold $\mathcal{K}_j$. One potential downside is that a closed form for covering designs for arbitrary p, m, t is still an open problem, and deterministic algorithms for generating covering designs are not known. Fortunately, the number of subsets q only needs to be computed once for fixed p, m, t. The best known covering designs also exist for the range of values p, m, t that we consider in this paper.

4.3. Information-theoretic multi-server DPF

Our final construction is an *information-theoretic* DPF that uses Shamir secret sharing to hide the special value α. The full protocol is presented in Algorithm 2.

A straw man approach is to use the Shamir secret sharing scheme described in 2.2 to directly share each element of the α'th standard basis vector $\vec{e_\alpha}$, and distribute these sharings to the p servers. Each Shamir share encodes either 0 or 1, and 1 is only encoded in the α-th index. During evaluation, each server executes a dot product between their Shamir shares and the corresponding data items in the replicated database. The result is a Shamir share of $X[\alpha]$.

We can further compress the query by considering the database a d-dimensional hypercube. Each index $m \in [N]$, we treat it as a d-dimensional tuple $m = (m_1, \ldots, m_d)$, $m_j \in [N^{\frac{1}{d}}]$. Letting our special index be $\alpha = (\alpha_1, \ldots, \alpha_d)$, we can then use the (p, t)-secret sharing scheme to share each of the dimensions separately, where each dimension

Algorithm 2 Shamir Based Multi-Party DPF $O(n^{\frac{1}{d}})$

Notation: d represents the pre-determined branching factor of the Shamir DPF.

```
DPF.Gen($1^\lambda, \alpha, \beta$):
1: Let $\alpha = \alpha_1 || \cdots || \alpha_d$
2: for $i = 1, \ldots, d$ do
3:     for $k = 1, \ldots, n^{\frac{1}{d}}$ do
4:         If $k = \alpha_i$ sample deg-$t$ $r_i[k](x)$, $r_i[k](0) = 1$.
5:         Else, samp. deg $t$ func. $r_i[k](x)$, $r_i[k](0) = 0$.
6:     Set $\sigma_j = \{r_i[k](j)\}_{k=1}^{N^{\frac{1}{d}}}$, $j = 1, \cdots, p$
7: Set $\mathcal{K}_j = \sigma_1 || \cdots || \sigma_d$
8: return $(\mathcal{K}_1, \ldots, \mathcal{K}_p)$

DPF.Eval($j, \mathcal{K}_j, x$):
1: Parse $\mathcal{K}_j = \sigma_1 || \cdots || \sigma_d$
2: Parse $\sigma_i = \{r_i[k](j)\}_{k=1}^{N^{\frac{1}{d}}}$ for $i = 1, \cdots, d$
3: Let $x = x_1 ||, ..., || x_d$
4: Initialize $y_j = 1$.
5: for $l = 1, \cdots, d$ do
6:     Update $A_l = \prod_{l' \in [d], l' \neq l} r_{l'}[x_{l'}](j)$
7:     Set $y_j = y_j \cdot r_l[x_l](j)$
8: return $\mathsf{res}_j = A_1 || \cdots || A_d || y_j$

DPF.Rec($(i_1, \mathsf{res}_1), \ldots, (i_{t'}, \mathsf{res}_{t'})$):
1: Parse $res_j = A_{1,j} || ... || A_{d,j} || y_j$ for $j = 1, \cdots, t'$
2: Compute value $y'_j$ from $i_j, (A_{1,j} || ... || A_{d,j})$, and sampled random functions from DPF.Gen
3: return Hermite Interpolation of the $2t'$ points $(i_1, y_1), \cdots, (i_{t'}, y_{t'}), (i_1, y'_1), \cdots, (i_{t'}, y'_{t'})$
```

is a $O(N^{\frac{1}{d}})$-vector of all 0's with a 1 in the α_j'th position. We denote the i'th Shamir share polynomial in dimension $k \in [d]$ as $r_k[m_i](x)$. In order to reconstruct each evaluation index, the server can take the product of the corresponding d Shamir sharings. For an index $m \in [N] = (m_1, \ldots, m_d)$, the Shamir shared index is $r_1[m_1](x) \times r_1[m_2](x) \ldots \times r_1[m_d](x)$. To query a database X, we simply apply a dot product between the Shamir responses and each individual database item $X[i]$: $\sum_{m=1}^{N}(r_1[m_1](x) r_2[m_2](x) \ldots r_d[m_d](x)) \cdot X[m]$.

This response is a sharing of a degree dt polynomial that encodes 1 if and only if x is the secret index, and 0 otherwise. However, the number of servers needs to be greater than $dt + 1$, compared to $t + 1$ for the strawman scheme.

Reducing Number of Servers with Hermite Interpolation
A major disadvantage of the above scheme is the high number of servers needed. The number of servers required is at least $dt + 1$, where d is the branching factor and t is the collusion threshold. In this section, we leverage Hermite interpolation to reduce the number of necessary servers.

Hermite interpolation [52], which generalizes Lagrange interpolation, computes a function of mn degree using n evaluations as well as the $m - 1$ first derivatives.

In our setting, we observe that it is possible to use

Hermite interpolation to reduce the number of servers required to interpolate the fault-tolerant DPF. Particularly, we additionally require each server to calculate auxiliary information for the client to compute additional derivatives of the response function client side. Specifically, for $\mathsf{res}(x) = \sum_{m=1}^{N}(r_1[m_1](x)\cdots r_d[m_d](x))\cdot X[m]$, the client is able to reconstruct $\mathsf{res}(x)$ with only $\frac{dt+1}{2}$ servers, if it can also compute the corresponding first order derivatives.

We now show how to calculate the value $\mathsf{res}'(x)$. First note that the j'th server has access to the values $r_1[m_1](j),\ldots,r_d[m_d](j)$ for all m. These are the evaluation points that correspond to the Shamir polynomials. By the chain rule, we have that $\mathsf{res}'(j) = \sum_{m=1}^{N}(r_1[m_1]'(j)\cdots r_d[m_d](j))\cdot X[m] + \cdots + \sum_{m=1}^{N}(r_1[m_1](j)\cdots r_d[m_d]'(j))\cdot X[m]$.

Consider the first term $\sum_{m=1}^{N}(r_1[m_1]'(j)\cdots r_d[m_d](j))\cdot X[m]$. It is insecure for the server to actually receive the value $r'_{m_1}(j)$ as this gives a collusion of t servers enough information to recover $r_{m_1}(j)$, breaking the security of the Shamir DPF. Because of this, we instead calculate intermediate values that allows the client to calculate this sum herself. In particular, since $m_1 \in [N^{\frac{1}{d}}]$, we can calculate the vector $\{\sum_{m|m_1=k}(r_2[m_2](j)\cdots r_d[m_d](j))\cdot X[m]\}_{k\in[N^{\frac{1}{d}}]}$. The same vector can be similarly calculated for all of the terms, allowing the client to calculate the derivative locally.

5. Communication-efficient PIR over erasure coded databases

5.1. Erasure code PIR building block

We first present a state-of-the-art PIR protocol over erasure coded storage [49] that uses *linear-sized* queries. We then explain how to adapt our DPF constructions to compress the query size to be sublinear while still being compatible over erasure coded storage. Note that we describe a particular version of their scheme in which the client only needs to send one linear query per server in order to download the entire item, at the cost of additional servers. It is also possible to generalize this scheme so that the protocol requires fewer servers at the cost of additional linear queries/ per-server computation. We refer the reader to [49] for additional details.

Encoding Each item in the database is erasure coded using a $[p,k]$-MDS code. Concretely, the encoded database stored at server j is $\mathcal{X}_j = \{X[i]_1 + X[i]_2\cdot j + \cdots + X[i]_k\cdot j^{k-1}\}_{i=1}^{N}$.

Query Assume that the client is querying for the desired index α. For each server, the client prepares a linear sized query. Specifically, they prepare a linear sized vector of secret shares $\vec{q(x)} = [g_1(x),\ldots,g_N(x)]$. Given random $r_{i,j} \overset{\$}{\leftarrow} \mathbb{F}$, each of the $g_i(x)$ is defined:

$$g_i(x) = \begin{cases} r_{i,1} + \cdots + r_{i,t}x^{t-1} + x^{t-1+k} & i = \alpha \\ r_{i,1} + \cdots + r_{i,t}x^{t-1} & o/w \end{cases}$$

The client then sends this vector of shares $\vec{q(j)} = [g_1(j),\ldots,g_N(j)]$ to each server j.

Evaluation During evaluation, each server simply returns the dot product between its erasure coded database and the linear query vector it has received. Concretely, the response from server j is the response res_j. Note that the dot product is equal to the following polynomial evaluated at j:

$$\begin{aligned} h(j) = &\sum_{i'=1,i'\neq\alpha}^{N} g_{i'}(j)\cdot(X[i']_1 + \cdots + X[i']_k\cdot j^{k-1}) \\ &\quad + g_\alpha(j)\cdot(X[\alpha]_1 + \cdots + X[\alpha]_k\cdot j^{k-1}) \\ &= r'(j) + \underbrace{X[\alpha]_1\cdot j^{t-1+k} + \cdots + X[\alpha]_k\cdot j^{t+2k-2}}_{s(j)} \end{aligned}$$

where $r'(x)$ is a random degree $t+k-2$ degree polynomial.

Reconstruction Note that given the at least $2k+t-1$ (honest) responses, the client can locally interpolate the polynomial $h(x)$. From this polynomial, the first $t+k$ coefficients are random and can be discarded. However, the last k coefficients are precisely the k shards $X[\alpha]_1,\ldots,X[\alpha]_k = X[\alpha]$, which is the desired item.

In summary, in the above scheme, the client prepares $2k+t-1$ linear-sized queries, and each server performs a linear scan over a database of size $\frac{N}{k}$.

5.2. Fault-tolerant PIR protocol 1—no collusion

In this section, we design a new fault-tolerant PIR protocol with communication overhead of $O(\log N)$. We achieve this by composing our fault-tolerant DPF scheme described in Section 4.1 and the building block from [49] described in Section 5.1. Naively combining these protocols does not work as the query structure required by [49] is different from what Section 4.1 requires.

We discuss how to adapt our DPF scheme from Section 4.1 with a simple insight. Specifically, Recall that our DPF construction gives us a technique such that, on an input β, outputs logarithmic sized keys that are compressed representations of the linear-sized vectors $\vec{r}, \vec{r} + \beta\cdot\vec{e_i},\ldots,\vec{r} + (p-1)\beta\cdot\vec{e_i}$. We observe that rather than taking as input only a single value of β in the final correction word, it is actually possible to directly input $p-1$ chosen values $\mathsf{val}_1,\ldots,\mathsf{val}_{p-1}$ in the final correction word of the protocol (see line 20 of Algorithm 1). This is due to the fact that the security of the scheme is independent of these chosen values, which are eventually masked by the randomness of the independently generated seed. This adaptation of our DPF scheme allows us to create DPF keys that represent sublinear sized representations of the vectors $\vec{r}, \vec{r} + \mathsf{val}_1\cdot\vec{e_i},\ldots,\vec{r} + \mathsf{val}_{p-1}\cdot\vec{e_i}$ for client-chosen values $\mathsf{val}_1,\ldots,\mathsf{val}_{p-1}$.

Using this modified functionality, we now show how this can be used to be compatible with the coded PIR protocol in the following section.

We observe that in the Tajeddine et al. [49] protocol in the case where $t = 1$, the linear query sent to the j'th party is of the form $\vec{r} + j^k \cdot \vec{e_i}$, for $j \in \{1, \dots, p\}$. In the case of $j = 1$, this vector is equal to $\vec{r'} = \vec{r} + \vec{e_i}$. With this in mind, choosing the $p - 1$ values $\{\mathsf{val}_1, \dots, \mathsf{val}_{p-1}\} = \{2^m - 1, \dots, p^m - 1\}$ allows us to have p logarithmic-sized keys that represent a set of p vectors $\vec{r}+\vec{e_i}, \vec{r}+2^m\cdot\vec{e_i}, \dots, \vec{r}+p^m \cdot \vec{e_i}$. These are precisely the query vectors needed for the Tajeddine et al. protocol in the case $t = 1$, giving us a way to compress the size of the queries to be logarithmic. The full protocol is presented in Algorithm 3.

Algorithm 3 $(p, 1, r, b)$ - Coded PIR Protocol

Notation: Let C be a $[p, k]$-MDS code with generator G, where $p \geq t+1+r+2b$. We use the DPF.{Gen, Eval, Rec} algorithms described in Algorithm 1.

```
PIR.Encode($X$):
1: for $X[i] \in X$ do
2:     Parse $X[i] = X[i]_1 \| \cdots \| X[i]_k$
3:     Let $[f_{i,1}, \dots, f_{i,p}] = [X[i]_1, \dots, X[i]_k] \cdot G$
4: Let $\mathcal{X}_j = \{f_{i,j}\}_{i=1}^n$
5: return $(\mathcal{X}_1, \dots, \mathcal{X}_p)$

PIR.Query($p, t, X, \alpha$):
1: Let $\mathsf{val}_j = (j-1)^k - 1$ for $j = 1, \dots, p-1$
2: $(\mathcal{K}_1, \dots, \mathcal{K}_p) \leftarrow \mathsf{Gen}(1^\lambda, \alpha, \{\mathsf{val}_1, \dots, \mathsf{val}_{p-1}\})$
3: return $(\mathcal{K}_1, \dots, \mathcal{K}_p)$

PIR.Eval($j, \mathcal{K}_j, \mathcal{X}_j$):
1: return $\mathsf{res}_j = \sum_{m=1}^n \mathsf{Eval}(j, \mathcal{K}_j, m) \cdot \mathcal{X}_j[m]$

PIR.Rec($(j_1, \mathsf{res}_{j_1}), \cdots, (j_{t'}, \mathsf{res}_{j_{t'}})$):
1: Let $h(x) \leftarrow \mathsf{IP}((j_1, \mathsf{res}_{j_1}), \cdots, (j_{t'}, \mathsf{res}_{j_{t'}}))$
2: Parse $X[\alpha]_1, \dots, X[\alpha]_k$ from coefficients of $h(x)$ as
   described in 5.1.
3: return $X[\alpha] = X[\alpha]_1, \dots, X[\alpha]_k$
```

5.3. Fault-tolerant PIR protocol 2 — Shamir-based PIR

While the previous PIR protocol is communication-efficient, it cannot tolerate any collusion. We describe how to compose the recursive Shamir DPF described in Section 4.3 with Section 5.1 to design a PIR protocol with sublinear query size and can tolerate server collusion. We describe the protocol in the case $d = 2$, which leads to $O(\sqrt{N})$−sized keys, but note that the same technique can be directly extended to any generalized d. We present the generalized case in Algorithm 4 in the Appendix.

Recall from Section 4.3 that our Shamir DPF keys are $O(\sqrt{N})$-length vectors such that when evaluated at an index i, the server evaluates a $(p, 2t)$-secret share of 0 or 1, and then utilizes Hermite interpolation such that the client only requires $\frac{2t+1}{2}$ responses to interpolate the result. Directly combining our DPF scheme with the Tajeddine et al. coding scheme does not work for multiple reasons. First, the shares in the Tajeddine et al. protocol are of the form: $g(x) = r_1 + r_2 x + \cdots + r_t x^{t-1} + b x^{t-1+k}$ where b is 0 or 1. Their shares have a gap of degree k between the second to last and last terms, which is used to retrieve all k shards simultaneously. However, our Shamir shares do not have this same gap, which is required to retrieve the item. Second, the encoded database stores items of the form $f_i(x) = X[i]_1 + X[i]_2 \cdot x + \cdots + X[i]_k \cdot x^{k-1}$, so taking the dot product with the encoded database gives us a secret share of the function $\sum_{m=1}^N (r_1[m_1](x) \cdot r_2[m_2](x)) f_m(x)$. If we wish to use Hermite interpolation directly on this share, this would require also knowing the derivative of $f_m(x)$, which cannot be calculated locally based on the information stored on the server.

To address the first issue, we create secret shares for the DPF key directly using the function $r_{i'}(x) = r_{i',1} + r_{i',2} x + \cdots + r_{i',t} x^{t-1} + b x^{t-1+k}$ instead of using standard Shamir sharing. Note that this is still a (p, t)-secure secret sharing scheme since these shares would still contain t random coefficients, so privacy is still maintained. Another property that this shares with Shamir secret sharing is that the product of two of these (p, t)-secret shares is additionally still a valid $(p, 2t)$-secret share. These two properties allow us to use this secret sharing scheme directly with our Shamir DPF scheme described in Section 4.3.

To address the second issue, we additionally require the server to store the values of $f'_i(x) = X[i]_2 + X[i]_3 \cdot x + \cdots + (k-1) \cdot X[i]_k \cdot x^{k-2}$ for all i. This can be easily calculated during the PIR.Encode phase and stored on each server. Although this does incur extra storage overhead compared to the prior work of Tajeddine et al. [49] by 2, we note that this still represents an overall storage reduction per server provided that $k > 2$.

Below we provide a concrete example of the protocol in the case $d = 2$, and refer the reader to Algorithm 4 in the Appendix for more details. For a database X such that $|X| = N$, each Server i stores the following $2N$ values:

$$S_i = \begin{cases} \{f_i(x) = X[i]_1 + X[i]_2 \cdot x + \cdots + \cdot X[i]_k \cdot x^{k-1}\}_{i=1}^N \\ \{f'_i(x) = X[i]_2 + \cdots + (k-1) \cdot X[i]_k \cdot x^{k-2}\}_{i=1}^N \end{cases}$$

When querying for an index $\alpha = (\alpha_1 || \alpha_2)$, such that $\alpha_1 \sqrt{N} + \alpha_2 = \alpha$, the client prepares a key according to Algorithm 2. On each Server j, the server now has the necessary information to compute the following $\sqrt{N} + 2$ values, which are sent to the client:

- $\mathsf{val(j)} = \sum_{i=1}^N r_1[i_1](j) r_2[i_2](j) f_i(j)$
- $A_{i_1}(j) = \sum_{i_2=1}^{\sqrt{N}} r_2[i_2](j) f_{i_1||i_2}(j)$ for $i_1 = 1, \dots, \sqrt{N}$
- $B_{i_2}(j) = \sum_{i_1=1}^{\sqrt{N}} r_1[i_1](j) f_{i_1||i_2}(j)$ for $i_2 = 1, \dots, \sqrt{N}$
- $\mathsf{val'(j)} = \sum_{i=1}^N r_1[i_1](j) r_2[i_2](j) f'_i(j)$

Note that the client now has the values $\mathsf{res(j)}_{j=1}^p = \mathsf{val(j)}_{j=1}^p$, and has enough information to locally compute the values:

$$\begin{aligned}\mathsf{res}'(\mathrm{j}) &= \sum_{i=1}^{N} r_1[i_1]' r_2[i_2] f_i(j) + \sum_{i=1}^{N} r_1[i_1] r_2[i_2]' f_i(j) \\ &+ \sum_{i=1}^{N} r_1[i_1] r_2[i_2] f_i'(j) \\ &= \sum_{i_1=1}^{\sqrt{N}} r_1'[i_1](j) A_{i_1}(j) + \sum_{i_2=1}^{\sqrt{N}} r_2'[i_2](j) B_{i_2}(j) + \mathsf{val}'(j)\end{aligned}$$

With the $2p$ values res(j) and $\mathsf{res}'(j)$, the client can now perform Hermite interpolation to recover their desired result.

Algorithm 4 (p, t, r, b) - Shamir Based Robust PIR Protocol

Notation: Let C be a $[p, k]$-MDS code with generator G and corresponding matrix G', where $p \geq \frac{dt}{2} + k + r + 2b$. We use the DPF.{Gen, Eval, Rec} algorithms described in Algorithm 2.

```
PIR.Encode($X$):
1: for $X[i] \in X$ do
2:     Parse $X[i] = X[i]_1 \| \cdots \| X[i]_k$
3:     Let $[f_{i,1}, \dots, f_{i,p}] = [X[i]_1, \dots, X[i]_k] \cdot G$
4:     Let $[f'_{i,1}, \dots, f'_{i,p}] = [X[i]_1, \dots, X[i]_k] \cdot G'$
5: Let $\mathcal{X}_j = \{f_{i,j}\}_{i=1}^N \| \{f'_{i,j}\}_{i=1}^N$
6: return $(\mathcal{X}_1, \dots, \mathcal{X}_p)$

PIR.Query($p, t, X, i$):
1: $(\mathcal{K}_1, \dots, \mathcal{K}_p) \leftarrow$ DPF.Gen$(i, 1)$
2: return $(\mathcal{K}_1, \dots, \mathcal{K}_p)$

PIR.Eval($j, \mathcal{K}_j, S_j$):
1: Initialize $d$ arrays of size $N^{\frac{1}{d}}$, Arr$_1 \dots$, Arr$_d$
2: Initialize val $= 0$, val$' = 0$
3: for $m = 1, \dots, N$ do
4:     Parse $m = m_1 \| \cdots \| m_d$
5:     $A_1 \| \cdots \| A_d \| y_m \leftarrow$ DPF.Eval$(j, \mathcal{K}_j, m)$
6:     for $l = 1, \dots, d$ do
7:         Update Arr$_l[m_l]$ += $A_l \cdot f_{m,j}$
8:     Set val$^{(r)}$ += $y_m \cdot f_{m,j}$
9:     Set val$'^{(r)}$ += $y_m \cdot f'_{m,j}$
10: Set res$_j$ = Arr$_1 \| \cdots \|$Arr$_d \|$val$\|$val$'$
11: return res$_j$

PIR.Rec($(j_1, \mathsf{res}_1), \cdots, (j_{t'}, \mathsf{res}_{t'})$):
1: Parse res$_j = A_{1,j} \| \cdots \| A_{d,j} \| y_j$ for $j = 1, \dots, t'$.
2: Compute value $y'_j$ from $i_j, (A_{1,j} \| \cdots \| A_{d,j})$, and sampled random functions from DPF.Gen
3: Recover $f(x)$ from Hermite Interpolation of the $2t'$ points $(i_1, y_1), \dots, (i_{t'}, y_{t'}), (i_1, y'_1), \dots, (i_{t'}, y'_{t'})$
4: Parse $X[\alpha]_1, \dots, X[\alpha]_k$ from coefficients of $h(x)$ as described in 5.1.
5: return $X[\alpha] = X[\alpha]_1, \dots, X[\alpha]_k$
```

5.4. Fault-tolerant PIR protocol 3 — leveraging covering design-based DPF

For our final scheme, we observe that rather than erasure coding the database at the item-level, erasure coding at the database level allows us to utilize our querying scheme effectively. We first present a building block that will be helpful in our construction. In particular, we utilize the following building block: given any linear vector $\vec{v}$ and p servers that store $[p, k]$-MDS encoded data $\mathcal{X}_1, \dots, \mathcal{X}_p$, where the database was encoded across items, then given any k *honest* responses $[r_{i_1}, \dots, r_{i_k}] = [\vec{v} \cdot \mathcal{X}_{i_1}, \dots, \vec{v} \cdot \mathcal{X}_{i_k}]$, it is always possible to recover the value $\vec{v} \cdot X_\delta$ for any δ.

If we assume we receive k honest responses from servers indexed by the indices in the set $I = \{i_1, \dots, i_k\}$, we can then write the k responses from these servers as

$$[r_{i_1}, \dots, r_{i_k}] = [\vec{v} \cdot (\sum_{j=1}^{k} X_j G_{i_1,j}), \dots, \vec{v} \cdot (\sum_{j=1}^{k} X_j G_{i_k,j})] \tag{1}$$

$$= [\vec{v} \cdot X_1, \dots, \vec{v} \cdot X_k] \cdot G[I] \tag{2}$$

where $G[I]$ corresponds to the k-by-k matrix representing the k columns of G indexed by I. By the property of MDS codes, there exists an inverse to this square matrix, which we denote $G[I]^{-1}$. Multiplying this inverse to each side gives us

$$[r_{i_1}, \dots, r_{i_k}] \cdot G[I]^{-1} = [\vec{v} \cdot X_1, \dots, \vec{v} \cdot X_k]$$

We use $\vec{v} \cdot X_\delta \leftarrow \mathsf{Retrieve}(\vec{v}, I, \{r_{i_1}, \dots, r_{i_k}\})$ to represent this building block above.

We now use the above building block in order to construct our final coded PIR scheme. Assume that the database is encoded across the p parties as encoded databases $\mathcal{X}_1, \dots, \mathcal{X}_p$, and without loss of generality, assume that our desired item is located at index $\alpha \in X_1$.

The client first generates q keys $(\mathcal{K}_1, \dots, \mathcal{K}_q) \leftarrow$ DPF.Gen$(\alpha, 1)$, where q is the number of keys required by the $C(p, m, t)$ covering design in Section 4.2. We then distribute the q keys to the p parties according to the covering design (or RSS). The security property of this key distribution protocol guarantees no collusion of t servers learns anything about the special index.

For each key that the server received, the server can evaluate the key at every index and perform a dot product with the expanded key and the database contents. Concretely, each server sends back the response $\sum_{m=1}^{N}$ DPF.Eval$(j, \mathcal{K}, m)\mathcal{X}_j[m]$ for each key they received during the distribution phase.

During the decoding phase, since each key $\mathcal{K}_i$, $i \in [q]$ was received by $p - m$ parties (which the construction of covering design guarantees), the client can recover the value $\sum_{j=1}^{N}$ DPF.Eval$(\mathcal{K}_i, j)X_1[j]$ by the Retrieve building block described above (provided that $p - m \geq k$.) By the properties of DPF, taking the sum of these q intermediary values allows the client to retrieve the value $X_1[\alpha]$ as desired. The full algorithm is discussed in Algorithm 5.

Algorithm 5 (p, t, r, b)-Covering Design Based Coded Fault-tolerant PIR Protocol

Notation: Let C be a $[p, k]$-MDS code with generator G, where $p \geq m + k + r + 2b$. Let RSS.Share be the algorithm described in Section 2.2, Retrieve be the building block described in Section 5.4, and $A_1, \cdots, A_k$ as the enumeration of the q covering sets of $C(p, m, t)$ We make black box use of the multi-party DPF algorithm DPF.{Gen, Eval} described in [10].

```
PIR.Encode($X$):
1: Partition $X$ into $k$ equal-sized matrices $X = X_1 \| \cdots \| X_k$.
2: Let $[\mathcal{X}_1, \ldots, \mathcal{X}_p] = [\mathcal{X}_1, \ldots, \mathcal{X}_k] \cdot G$
3: return $(\mathcal{X}_1, \ldots, \mathcal{X}_p)$

PIR.Query($p, t, X, i$):
1: Let $q = \binom{p}{p-t}$
2: Compute $(\mathcal{K}_1, \ldots, \mathcal{K}_q) \leftarrow \mathsf{DPF.Gen}(1^\lambda, i, 1)$
3: Compute $(\sigma_1, \ldots, \sigma_p) \leftarrow \mathsf{RSS.Share}(\mathcal{K}_1, \ldots, \mathcal{K}_q)$
4: return $(\sigma_1, \ldots, \sigma_p)$

PIR.Eval($j, \sigma_j, \mathcal{X}_j$):
1: return $r_j = \{\sum_{i=1}^{n} \mathsf{DPF.Eval}(j, \mathcal{K}, i) \cdot \mathcal{X}_j[i] \mid \mathcal{K} \in \sigma_j\}$

PIR.Rec($(j_1, r_{j_1}), \ldots, (j_{t'}, r_{j_{t'}})$):
1: Let $X[i] \in X_\delta$
2: for $m = 1, \ldots, q$ do
3:     for ind in $A_m$ do
4:         Let $\mathsf{res_{ind}} = \mathcal{K}_m \cdot S_{\mathsf{ind}} \in r_{\mathsf{ind}}$
5:     $\mathcal{K}_m \cdot X_\delta \leftarrow \mathsf{Retrieve}(\mathcal{K}_m, i, A_m, \{\mathsf{res_{ind}} \mid \mathsf{ind} \in A_m\})$
6: return $\sum_{m=1}^{q} \mathcal{K}_m \cdot X_\delta = X[i]$
```

5.5. Extending to Malicious Security

While prior work [17] leveraged cryptographic techniques such as MACs to check the integrity of server responses, our approach allows the client to detect errors in the servers' responses directly, letting us avoid the extra MAC storage as well as the need for a client key.

First note that our use of Reed-Solomon codes guarantees each of the responses in all of the above protocols are evaluations of some degree γ function $h(x)$ at a party index. This means assuming $\gamma + 1$ (honest) responses, the client can simply use standard interpolation to retrieve their desired value, whether that is an item shard in Algorithm 3 or Algorithm 4, or the value $v \cdot X_\delta$ in Algorithm 5. This algebraic structure of the response values allows us to extend standard coding techniques to account for non-responsive (r) and malicious (b) servers. To do so, we leverage known techniques from information theory that allows us to efficiently correct errors, specifically the Berlekamp-Welch algorithm [8] for error correction. We refer the reader to [8] for details, but at a high level, Berlekamp-Welch provides a way for the client to interpolate a degree γ function $h(x)$ with $\gamma + 2b + 1$ responses, where at most b of the responses are Byzantine. We emphasize this number is *optimal* due to properties of MDS codes.

In addition, to account for r non-responsive servers, we can simply add an extra r servers to the system, which provides the redundancy required to guarantee the client can still recover her result. Specifically, if there are $\gamma + r + 2b + 1$ servers, r of these can be non-responsive which still guarantees the $\gamma + 2b + 1$ responses need for interpolation.

6. Implementation

We implement our above protocols in $\sim$5,000 lines of C++ and Go. We utilize the OpenSSL library [40] for cryptographic functions, and AES for PRF evaluations. Our tree-based and multiparty DPF implementations closely follow the implementations provided in DORY [17] and Splinter [50], respectively. For the erasure coding and polynomial evaluations, we use $\mathsf{GF}(2^8)$ as the underlying field. This is the standard field for erasure code implementations because additions are equivalent to XORs and multiplications are just table lookups in a 256 by 256 table. We use the lookup table provided in Intel's ISA-L erasure-coding library [32], as well as the associated functions. ISA-L is the standard tool used for implementing erasure codes in data storage systems [41], [47].

Optimized DPF Tree Implementation Naively implementing the tree-based Robust DPF from Section 4.1 requires k separate DPFs for each query, one to retrieve each of the k shards. This leads to a query of size $O(kp \log N)$ and kN AES evaluations per PIR query. We instead implement an optimized DPF, with key length of $O(p \log N) + O(k)$ and only requires N AES evaluations. Our insight is that we can pack the k correction words in each leaf position together, and only use one seed per leaf to hide the contents of the packed correction words. This leads us to only having to expand the entire tree once per evaluation, which is the main bottleneck of the DPF Tree protocol.

7. Evaluation

The evaluation aims to answer the following questions:

- What are the theoretic storage, communication, and compute costs of our erasure coded PIR schemes compared to prior robust PIR baselines? (Section 7.1)
- How does our protocols compare to prior robust PIR baselines? (Section 7.2)
- How does erasure-coding improve the performance of PIR compared to replicated storage? (Section 7.2)
- What is the cost of protecting against malicious servers? (Section 7.2)
- How do the protocols' performance change as we vary parameters? (Section 7.3)

Table 3 provides a summary of all of the parameters in our protocols, which we refer to throughout our evaluation.

Scheme	Storage Rate	No. Servers Required	Key Length	Response Length	# AES Evals	# Mults
Devet et al. [19]	1	$p \geq t + 1 + r + 2b$	$O(N)$	$O(1)$	0	N
Tajeddine et al. [49]	$1/k$	$p \geq t + k + r + 2b$	$O(N)$	$O(1)$	0	$\frac{N}{k}$
Woodruff et al. [52]	1	$p \geq dt + 1 + r + 2b$	$O(N^{\frac{1}{d}})$	$O(1)$	0	$O(\sqrt{N}) \cdot N$
Tree-Based PIR (Algorithm 3)	$1/k$	$p \geq 1 + k + r + 2b$	$O(p \log N)$	$O(1)$	$\frac{N}{16k}$	$\frac{N}{k}$
Shamir Based PIR (Algorithm 4)	$2/k$	$p \geq \frac{dt}{2} + k + r + 2b$	$O(N^{\frac{1}{d}})$	$O(N^{\frac{1}{d}})$	0	$\frac{5N}{k}$
CD-Based PIR (Algorithm 5)	$1/k$	$p \geq m + k + r + 2b$	$O(2^q \sqrt{N})$	$O(q)$	$\frac{\sqrt{N}}{k} \cdot 2^q$	$\frac{q(p-m)}{p} * \frac{N}{k}$

TABLE 2: Theoretic Performance of Coded PIR protocols, where $q < \binom{p}{t}$ is the size of the (p, m, t)-covering design. In our evaluation, we set $d = 2$ for the WY and Shamir schemes. The amount of computation is the amount performed on each server per query.

Parameter	Meaning
N	No. items in database
p	No. servers
r	No. stragglers/fail-stop servers
b	No. Byzantine servers
t	Collusion threshold
k	Reconstruction threshold of MDS code

TABLE 3: Parameters used in our Coded PIR protocols

Setup We evaluate our protocols on AWS r5.4xlarge instances with 16 cores and 128GB of memory. All server computation is parallelized across the 16 virtual cores on each machine. We place the client machine on the west coast, and the rest on the east coast. All communication is performed over TLS on a 2Gbps network latency. All results are computed as the average over ten trials.

Baselines In our evaluation we consider the following *robust* baseline schemes and compare them to our three protocols:
Devet et al. (DGH): An implementation of a robust PIR scheme over replicated storage [19], [28]. This scheme achieves robustness, but utilizes replicated storage and linear-sized queries.
Tajeddine et al. (TGK+): An implementation of the linear querying scheme described in Section 5.1 [49]. This scheme achieves robustness with erasure-coded storage, but utilizes linear-sized queries.
Woodruff et al. (WY): An implementation of the PIR scheme described in [52]. This scheme achieves robustness and sublinear-sized queries, but utilizes replicated storage.

7.1. Theoretic Performance

Table 2 shows the theoretical performance of the baselines and our three protocols for the three main performance metrics that we consider in this paper: storage, communication, and computation.

Storage We evaluate the storage based on the schemes' per-server storage rate and the total storage across all servers. The *storage rate* is the ratio between the storage on the erasure-coded server and the original, unencoded server. For example, the storage rate of a replicated scheme is 1.

Each of our schemes achieves a per-server storage rate of less than 1 because of our use of erasure coding. Our tree and covering design schemes storage rates of $\frac{1}{k}$, while for the Shamir based PIR scheme, the storage rate is $\frac{2}{k}$, which means that our storage overhead is concretely smaller than replicated storage for $k > 2$.

While the storage reduction comes at a cost of additional servers (discussed below), the concrete *total storage* compared to a replicated storage baseline is also smaller. This is seen in the following section when comparing the total storage between the DGH and WY baselines and our schemes. This stems from the fact that our use of MDS codes allows us to provide robustness with less redundancy than merely replicating across multiple servers, allowing us to provide better per-server and total storage overheads. In addition, for larger values of k and b, our schemes will asymptotically have less total storage, because each additional server added to provide more robustness will have less storage than adding additional servers in a replicated setup. Each additional server in the replicated setting increases total storage by N, while total storage is only increased by $\frac{N}{k}$ in the erasure-coded setting.

When compared to the TGK+ scheme, however, our Shamir and covering design schemes do incur larger total storage overhead. For the CD scheme, this stems from the fact that covering designs require additional servers in order to achieve better concrete communication.. The Shamir incurs extra storage from the use of Hermite interpolation.

No. Servers Erasure coded PIR schemes require additional servers when compared to a replicated baseline. This increase in servers stems from the fact that the number of servers increase linearly in k, which is not a parameter is replicated robust PIR schemes.

Communication The communication cost of our schemes are all sublinear in the number of items, which is asymptotically better than the linear-sized queries required by both the DGH and TGK+ schemes. The Tree-based PIR scheme has a $O(p \log N)$ query size since it is derived from the two-party tree-based DPF [10], though at the cost of assuming no collusion among parties. Asymptotically, the Shamir based PIR scheme has a communication cost of $O(N^{\frac{1}{d}})$ (we set $d = 2$ by default), but has a higher response length of $O(N^{\frac{1}{d}})$ compared to other schemes. For the covering design PIR scheme, the query size is additionally exponential in the in the number of parties, which is inherent in the use of the multiparty DPF scheme in [10].

Compared to the WY scheme, we achieve the same or better asymptotic performance but worse practical key size due to larger constant factors in our schemes. However, to achieve this smaller key size, the WY scheme requires a larger amount of computation on each server compared to our schemes, which inhibits performance in practice (see Table 2 in the following section). Because the per-server compute is often the bottleneck in practice, we find that our schemes still perform better in our evaluation.

Computation For per-server computation, one can think of the computation required in two parts: first, the cost of expanding a sublinear key, and then the cost of performing a linear scan across the server contents. For example, for the DGK scheme, which sends a linear query vector to each server, the server does not require any additionally computation to expand each vector, but must perform a scan across a database of size N. The per-server compute is seen in Table 2. We separate the costs of multiplications (i.e. for the linear scan) and the AES evaluations.

When comparing the compute costs of our schemes to the replicated baseline schemes, our schemes have a lower per-server compute cost as k gets larger. Specifically, since each of the schemes must perform a linear scan across the database contents, which is the main computation bottleneck, erasure coding allows each server to scan across a smaller sized database, significantly reducing the computation on each server when compared to a replicated scheme. Compared to the TGK+ scheme, however, our schemes do incur a larger per-server compute cost. This is because their scheme also inherits the benefits of erasure-coding and additionally does not require any computation to perform a key expansion. However, note that the linear query size negatively affects the performance of their scheme in practice, which is seen in the following section.

Our schemes also achieve asymptotically better *total compute*. While this is empirically shown in the following section, this is because the number of servers only grows linearly with k, but per-server compute decreases *multiplicatively* in k, as seen in Table 2. Thus, although increasing k would increase the number of servers required, it would also decrease the total compute.

7.2. Cost Breakdown

In this section, we present cost breakdowns of the different protocols for two different database configurations of varying sizes. We run the various protocols under the parameters $k = 4, b = 2$ unless otherwise stated. We break down the end to end latency into four components: upload time (including the key generation), server compute time, download time, and the decoding time. We refer the reader to Table 4 to see the cost breakdown in each of the protocols. We look at the effects of changing these parameters in the following section (Section 7.3).

Comparison to Prior Work Baselines As seen in Table 4, our schemes outperform both the replicated robust PIR baselines (DGH [19] and WY [52]), as well as the state of art erasure coded PIR baseline (TGK+ [49]). For a 4GB sized database, our schemes demonstrate an improvement of 4.22–9.09$\times$ in end to end latency. This improvement stems from two places. First, when compared to the DGH and TDK+ schemes, our schemes save in the Upload Time because these schemes are bottlenecked by their linear key generation and query upload time. Second, when compared to the WY scheme, our computation time on each server is much smaller because of our use of erasure coding. Although their work is able to achieve key sizes concretely smaller than ours, this is at the cost of a more expensive server computation on each server. Because of this, our schemes are more efficient in practice.

Furthermore, the performance of our protocols are nearly unaffected even in the presence of Byzantine servers. The decoding time in the Byzantine setting takes less than 1 ms for the tree and multi-party schemes, and $\sim$22 ms for the Shamir scheme (which must compute the derivative locally as well as perform the Berlekamp-Welch algorithm).

Although our use of erasure coding comes at the cost of additional servers when compared to the replicated robust PIR baselines, we improve on both *total storage* and *total compute* when compared to the replicated baseline. First, the *total storage* is still less when compared to the replicated setting. In the configuration we consider in Table 4, for example, our schemes use 28–72% less total storage in our evaluation when compared to the WY scheme, and up to 7–64% when compared to the DGH scheme.

When comparing the **total computation** of the WY scheme to our schemes in the 4GB setting, our schemes have a 2.68–7.30$\times$ improvement, even with the additional servers. This is due to the fact that the WY scheme has a much more expensive computation phase in their protocol, and must also scan across a replicated server.

Compared to the DGH and TGK+ scheme, our schemes incur a modest overhead in total compute. This stems from the fact that our schemes require per-server computation in order to expand out the key to a linear vector even before performing a linear scan, while the prior schemes due not since the linear vector is sent directly. This leads to a modest overhead of $< 34\%$ and $< 50\%$ when compared to the DGH and TGK+ schemes, respectively. However, since the DGH scheme operates over replicated storage, further increasing k would also continually reduce the total compute (since the number of servers grows linearly in k, but per-server compute reduces multiplicatively).

In Table 6, we also compare the estimated costs of running a PIR query in each of the baseline and our schemes based on AWS cost estimates for compute and communication. Apart from the Tree scheme (0.6 cents per query) and the WY scheme (4.3 cents per query), the cost of the other schemes are similar ($1.1 - 1.5$ cents per query). The cost of the schemes are highly correlated with the total computation of the schemes, (as compute cost is in general more expensive than communication).

Comparison to Replicated Servers

Table 5 displays the performance of our schemes when used over both replicated and erasure-coded storage. As expected, our schemes perform much better over erasure-coded storage than in their replicated counterpart, exhibiting a 2.2–4.3$\times$ latency improvement.

Again, while erasure coding does require an increased

Database	Metric	Baselines			Our Schemes		
		DGH	TDK+	WY	Tree	Shamir	CD
	Number of Servers	7	13	9	12	13	16
	Storage Rate	1	0.25	1	0.25	0.5	0.25
	Total Storage	7×	3.25×	9×	3×	6.5×	4×
2^{22} × 512B (2GB)	Key Size	4MB	4MB	2897B	8387B	4kB	340kB
	Response Size	512B	512B	512B	512B	525kB	768B
	Upload Time	1.11s	1.59s	0.13s	0.13s	0.14s	0.40s
	Compute Time	1.56s	0.85s	2.24s	0.38s	1.16s	0.87s
	Download Time	0.02s	0.02s	0.02s	0.02s	0.32s	0.02s
	Decode Time	$<$ 1ms	$<$ 1ms	25ms	$<$ 1ms	28ms	$<$ 1ms
	Total Time	2.70s	2.47s	2.81s	0.55s	1.65s	1.30s
2^{25} × 128B (4GB)	Key Size	32MB	32MB	8kB	10kB	12kB	836kB
	Response Size	128B	128B	128B	128B	393kB	192B
	Upload Time	6.05s	9.62s	0.13s	0.13s	0.19s	0.52s
	Compute Time	6.81s	3.30s	19.15s	1.97s	4.06s	4.04s
	Download Time	0.03s	0.05s	0.04s	0.03s	0.33s	0.03s
	Decode Time	$<$ 1ms	$<$ 1ms	47ms	$<$ 1ms	22ms	$<$ 1ms
	Total Time	12.89s	12.97s	19.37s	2.13s	4.61s	4.59s

TABLE 4: Breakdown of performance for baseline and our schemes for $k = 4$ and $b = 2$.

Database	Metric	Replicated Storage			Erasure-Coded Storage		
		Tree	Shamir	CD	Tree	Shamir	CD
	Number of Servers	6	9	9	12	13	16
	Storage Rate	1	1	1	0.25	0.5	0.25
	Total Storage	6×	9×	9×	3×	6.5×	4×
2^{22} × 512B (2GB)	Key Size	2881B	4kB	2.8MB	8387B	4kB	340kB
	Response Size	512B	2.1MB	512B	512B	525kB	768B
	Upload Time	0.13s	0.13s	0.66s	0.13s	0.14s	0.40s
	Compute Time	1.48s	3.59s	5.00s	0.38s	1.16s	0.87s
	Download Time	0.02s	0.43s	0.02s	0.02s	0.32s	0.02s
	Decode Time	$<$ 1ms	58ms	$<$ 1ms	$<$ 1ms	28ms	$<$ 1ms
	Total Time	1.64s	4.20s	5.69s	0.55s	1.65s	1.30s
2^{25} × 128B (4GB)	Key Size	3271B	12kB	6.7MB	9641B	12kB	836kB
	Response Size	128B	1.5MB	640B	128B	393kB	192B
	Upload Time	0.13s	0.19s	0.94s	0.13s	0.19s	0.52s
	Compute Time	6.96s	9.46s	18.78s	1.97s	4.06s	4.04s
	Download Time	0.02s	0.44s	0.02s	0.03s	0.33s	0.03s
	Decode Time	$<$ 1ms	42ms	$<$ 1ms	$<$ 1ms	22ms	$<$ 1ms
	Total Time	7.12s	10.13s	19.75s	2.13s	4.61s	4.59s

TABLE 5: Breakdown of PIR schemes in both replicated and erasure-coded settings. The erasure-coded setting uses $k = 4$.

DGH	TGK+	WY	Tree	Shamir	CD
1.2¢	1.1¢	4.3¢	0.6¢	1.3¢	1.5¢

TABLE 6: Cost estimates for the 4GB setting in Table 4 based on AWS costs (\$1.5 × 10^{-5} per core-second of computation, \$0.09 per GB of communication.)

number of servers when compared to the replicated baseline, we improve on both total storage and total computation. First, when comparing total storage, our schemes utilize 28-59% less total storage. Note that as we increase b (the number of Byzantine servers), erasure coding would further improve on the total storage overhead for the theoretical reasons mentioned in the previous section.

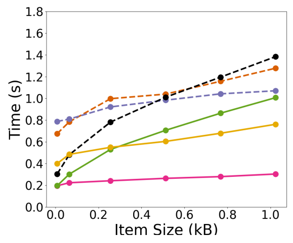
Second, when comparing the *total computation* of the schemes, we find that erasure codes help improve the total computation of the schemes by $1.61 - 2.61\times$. This improvement comes from the fact that the cost of the linear scan is greatly reduced by erasure coding. In particular, for the covering design scheme, which must do multiple linear scans (based on the number of keys each server receives), the cost incurred by each scan is greatly reduced, leading to the large decrease of computation required of each server. Further increasing k would help further reduce the cost of the linear scan at the cost of addition servers.

7.3. Effect of Parameters on Performance

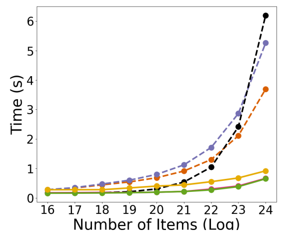
In this section, we discuss how the schemes scale with individual parameters. We evaluate our protocols using the parameters $k = 4, r = 0, b = 2$ unless otherwise stated.

Item Size: Figure 2a and Figure 2c show the performance of the protocols for 2^{20} and 2^{22} items as we scale item size.

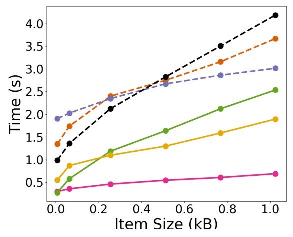
Our schemes outperform the robust PIR baselines across item sizes in both settings. For a fixed number of items, the query does not change, so the only performance difference is



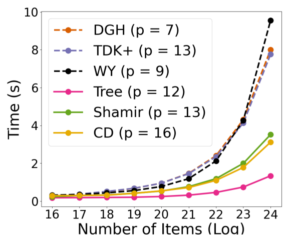
(a) Latency for 2^{20} items with varying item sizes.



(b) Latency for varying number of items for item size $= 8B$



(c) Latency for 2^{22} items with varying item sizes.



(d) Latency for varying number of items for item size $= 256B$

Figure 2: Latency of protocols for various parameter configurations. We use $k = 4, b = 2$ across all figures.

the amount of work incurred by the linear scan on the server, which only depends on the item size. This means that for a fixed number of items, the latency difference across schemes will always be the same, since the latency difference only comes from the query generation and upload time.

Each of the schemes scales *linearly* in the item size, with the two replicated baselines having the worst scaling slope. The DGH and WY schemes have the worst scaling, because each server in these schemes must perform a linear scan across a replicated database. We refer the reader to Table 2 to see the break down of per-server computation in terms of the PRF evaluations and the linear scan costs.

The Shamir based scheme, however, scales in a larger constant factor with respect to item size. This stems from the fact that the Shamir scheme requires the server to additionally calculate $\sqrt{N}$ intermediate sums that are later used by the client to reconstruct the derivative for Hermite interpolation. Since the size of these intermediary sums are dependent on the item size, the Shamir based PIR scheme scales at a linear factor larger than the other schemes. In particular, the Shamir PIR scheme would begin to underperform the TGK+ scheme as the item size grew larger. This breakpoint would increase for larger number of items.

No. Items N: Figure 2b and Figure 2d show the performance of the schemes as the number of items grows, where the item size is fixed at $8B$ and $256B$ respectively.

Our figures show that our schemes outperform the robust PIR baselines across item sizes as the number of items increases. This improvement stems from two sources. When compared to the DGH and TGK+ schemes, our schemes achieve improvement from our sublinear key size. As seen in the previous section, the linear key generation and upload time is the main bottleneck of those schemes. Second, when compared to the WY scheme, our use of erasure coding allows us to save on the server computation of each server. As the number of items grows larger, this compute savings allows us to achieve the latency gain seen in the figure.

MDS Code Reconstruction Threshold k:

Section 7.3 shows the performance of the erasure coding compatible schemes for $k = 2$ and $k = 4$. A larger k will reduce the per-server and total storage overhead at the cost of increasing the number of servers required. For this section, we compare our three schemes only to TGK+, the only baseline that supports erasure coded storage.

For various values of k, our schemes outperform the TGK+ baseline. This is largely due to the linear upload key generation present in their scheme. Hence, as k increases, the computation on each server decreases but their scheme is still bottlenecked by a linear key generation time. Interestingly, we actually see a small increase in the latency for their scheme as we increase k because the client must additionally prepare a larger number of linear-sized queries (because of the increase in the number of servers required).

Our schemes, in contrast, are bottlenecked by the server compute, which decreases with increasing k, hence saving on overall end-to-end latency.

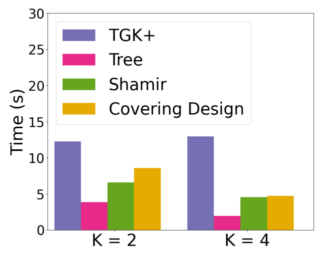


Figure 3: Latency for $k = 2$ and $k = 4$ for 2^{25} items of size $128B$

Comparison to DPF+MACs MAC verification extends any PIR protocol to support malicious *detection* as follows: 1) The client MACs every item in the database using a secret key and uploads the MAC values to the server along with the file payloads. We use a standard size of 256 bits for each MAC. 2) The server runs the PIR protocol on both the item contents as well as the MAC values. 3) After retrieving the item and its associated MAC, the client locally verifies that the MAC of the retrieved item equals the retrieved MAC value using the same secret key. The security guarantees of the MAC ensures that the item has been correctly retrieved.

Table 7 shows the total time comparing the two approaches. As expected, our approach outperforms the MAC-based approach for all of our protocols, albeit slightly as the item sizes grow larger. This is because the PIR protocols must perform a linear scan across the entire database and the additional MAC storage adds a non-trivial increase to this workload, particularly in the case where the database is smaller. Note that for the tree-based protocol, the savings is very small as the key expansion is the bottleneck of the computation, not the linear scan. For the Shamir protocol, the large drop in performance stems from the fact that the computation in Shamir is *superlinear* in the item size (a linear scan is required for each of the $O(\sqrt{N})$ intermediary sums for the Hermite interpolation).

The table does not display the setup or precomputation costs of using MACs, which is the main computation overhead of this approach (since the client must MAC every item). In addition, note that in a standard multi-client application, the system must either maintain a secret key for each client, or implement a protocol that allows synchronization of the key among multiple clients. This is a difficult problem that our approach does not face, since our protocol does not require the client to maintain any sort of secret state locally. Our approach also allows for error correction, while the MAC approach only allows for error detection.

	Tree		**Shamir**		**CD**	
	EC	MAC	EC	MAC	EC	MAC
Extra Storage (MB)	0	128	0	128	0	128
No. Servers	5	4	6	5	7	5
Total Time (s)						
2^{22} items, 8B	0.40	0.48	0.36	0.73	0.67	1.20
2^{22} items, 64B	0.54	0.56	0.85	1.3	1.27	1.31
2^{22} items, 256B	0.78	0.80	1.62	2.60	1.76	1.82
2^{22} items, 1024B	1.20	1.22	4.10	7.24	3.27	3.35

TABLE 7: EC stands for Erasure-Coded. We use $k = 2$ and $b = 1$ and a standard size of 32 bytes for each MAC. Extra storage is per-server.

8. Related Work

PIR Private information retrieval was originally proposed in [15]. Since then, various work have focused on reducing the download communication cost in the original setting with replicated servers [4], [5], [22], [46]. Non-responsive and Byzantine servers were first considered in [36], which protects the client from malicious responses, but cannot perform error correction.

PIR on Erasure Coded Storage This line of work focuses on performing PIR on data that has already been erasure coded and distributed for the purpose of failure tolerance with low storage-overhead. This problem was first investigated by Shah et al. in [44]. Since then, there has been a long line of work [2], [9], [14], [25], [26], [35], [37], [48], [49], [54], [55] proposing new constructions and investigating the fundamental tradeoffs between storage overhead, collusion threshold, and download communication cost.

An orthogonal line of work in the cryptography community has also looked at *batch codes* [33], which uses coding theory techniques to batch multiple queries together in order to amortize computation and communication costs.

Robust PIR This line of work considers the multi-server setting in which the client should be able to retrieve her desired result even when some of the servers are non-responsive. This was first studied by Beimel et al. in [6]. The bounds were then improved in [52]. The Byzantine setting was considered in [19], [28]. These works all operate in the replicated database setting. The work by Bunn et al. [13] also considers a threshold version of DPF for PIR, but their techniques are not compatible with erasure coded storage.

A complementary line of work in the information theory community has studied robust PIR on coded storage. The problem of PIR with byzantine servers has been studied in [1], [3]. The problem of PIR with byzantine *and* unresponsive servers has been studied in [49], [53]. These works optimize for storage overhead and/or download rate, but have linear query communication.

DPF in Systems Riposte [16] and Express [23] use DPFs to provide metadata-hiding anonymous communication. DORY [17] and DURASIFT [24] use DPFs to perform encrypted search queries without leaking access patterns. DPFs were used private database queries in systems such as Splinter [50] and Waldo [18]. Floram [21] is a multi-server ORAM that uses DPFs to beat other ORAM approaches for medium database sizes. DPFs have also recently used for systems to implement private contract tracing [20] and private nearest neighbor search [43].

9. Conclusion

We present novel robust PIR protocols that provide better computation, communication, and storage compared to prior state-of-art robust PIR schemes.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This material is based upon work supported by the National Science Foundation Award NSF SaTC 2326312, Sloan Research Fellowship 2023 and gift funding from Cisco and VMware.

References

[1] Daniel Augot, Françoise Levy-dit Vehel, and Abdullatif Shikfa. A storage-efficient and robust private information retrieval scheme allowing few servers. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis Askoxylakis, editors, *Cryptology and Network Security*, pages 222–239, Cham, 2014. Springer International Publishing.

[2] Karim Banawan and Sennur Ulukus. The capacity of private information retrieval from coded databases. *IEEE Transactions on Information Theory*, 64(3):1945–1956, 2018.

[3] Karim Banawan and Sennur Ulukus. The capacity of private information retrieval from byzantine and colluding databases. *IEEE Transactions on Information Theory*, 65(2):1206–1219, 2019.

[4] Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. In *International Colloquium on Automata, Languages, and Programming*, pages 912–926. Springer, 2001.

[5] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and J-F Raymond. Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 261–270. IEEE, 2002.

[6] Amos Beimel and Yoav Stahl. Robust information-theoretic private information retrieval. In *International Conference on Security in Communication Networks*, pages 326–341. Springer, 2002.

[7] Fabrice Benhamouda, Elette Boyle, Niv Gilboa, Shai Halevi, Yuval Ishai, and Ariel Nof. Generalized pseudorandom secret sharing and efficient straggler-resilient secure computation. In *Theory of Cryptography Conference*, pages 129–161. Springer, 2021.

[8] E. Berlekamp and L. Welch. Error correction for algebraic block codes, 1983.

[9] Simon R. Blackburn, Tuvi Etzion, and Maura B. Paterson. PIR schemes with small download complexity and low storage requirements. *IEEE Transactions on Information Theory*, 66(1):557–571, 2020.

[10] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 337–367. Springer, 2015.

[11] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 337–367. Springer, 2015.

[12] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1292–1303, New York, NY, USA, 2016. Association for Computing Machinery.

[13] Paul Bunn, Eyal Kushilevitz, and Rafail Ostrovsky. CNF-FSS and its applications. Cryptology ePrint Archive, Paper 2021/163, 2021. https://eprint.iacr.org/2021/163.

[14] Terence H Chan, Siu-Wai Ho, and Hirosuke Yamamoto. Private information retrieval for coded storage. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 2842–2846. IEEE, 2015.

[15] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[16] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.

[17] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. DORY: An encrypted search system with distributed trust. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1101–1119, 2020.

[18] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. Waldo: A private time-series database from function secret sharing. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2450–2468. IEEE, 2022.

[19] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally robust private information retrieval. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 269–283, 2012.

[20] Samuel Dittmer, Yuval Ishai, Steve Lu, Rafail Ostrovsky, Mohamed Elsabagh, Nikolaos Kiourtis, Brian Schulte, and Angelos Stavrou. Function secret sharing for psi-ca: With applications to private contact tracing. *arXiv preprint arXiv:2012.13053*, 2020.

[21] Jack Doerner and Abhi Shelat. Scaling ORAM for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 523–535, 2017.

[22] Zeev Dvir and Sivakanth Gopi. 2-server PIR with subpolynomial communication. *Journal of the ACM (JACM)*, 63(4):1–15, 2016.

[23] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1775–1792, 2021.

[24] Brett Hemenway Falk, Steve Lu, and Rafail Ostrovsky. DURASIFT: A robust, decentralized, encrypted database supporting private searches with complex policy controls. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, WPES'19, page 26–36, New York, NY, USA, 2019. Association for Computing Machinery.

[25] Arman Fazeli, Alexander Vardy, and Eitan Yaakobi. Codes for distributed PIR with low storage overhead. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 2852–2856. IEEE, 2015.

[26] Ragnar Freij-Hollanti, Oliver W Gnilke, Camilla Hollanti, and David A Karpuk. Private information retrieval from coded databases with colluding servers. *SIAM Journal on Applied Algebra and Geometry*, 1(1):647–664, 2017.

[27] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 640–658, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[28] Ian Goldberg. Improving the robustness of private information retrieval. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 131–148. IEEE, 2007.

[29] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[30] Daniel M Gordon and Douglas R Stinson. Coverings. In *Handbook of Combinatorial Designs*, pages 391–398. Chapman and Hall/CRC, 2006.

[31] Shaddi Hasan, Yahel Ben-David, Giulia Fanti, Eric Brewer, and Scott Shenker. Building dissent networks: Towards effective countermeasures against {Large-Scale} communications blackouts. In *3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI 13)*, 2013.

[32] Intel. Intel® intelligent storage acceleration library. https://www.intel.com/content/www/us/en/developer/tools/isa-l/overview.html. Accessed: 2022-10-11.

[33] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 262–271, 2004.

[34] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

[35] Siddhartha Kumar, Eirik Rosnes, and Alexandre Graell i Amat. Private information retrieval in distributed storage systems using an arbitrary linear code. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 1421–1425. IEEE, 2017.

[36] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings 38th annual symposium on foundations of computer science*, pages 364–373. IEEE, 1997.

[37] Julien Lavauzelle. Private information retrieval from transversal designs. *IEEE Transactions on Information Theory*, 65(2):1189–1205, 2019.

[38] Mingyu Li, Jinhao Zhu, Tianxu Zhang, Cheng Tan, Yubin Xia, Sebastian Angel, and Haibo Chen. Bringing decentralized search to decentralized services. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pages 331–347, 2021.

[39] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[40] OpenSSL project authors. OpenSSL: Cryptography and SSL/TLS toolkit. https://www.openssl.org/. Accessed: 2022-10-11.

[41] K. V. Rashmi, Mosharaf Chowdhury, Jack Kosaian, Ion Stoica, and Kannan Ramchandran. EC-Cache:Load-Balanced, Low-Latency Cluster Caching with Online Erasure Coding. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 401–417, 2016.

[42] Sacha Servan-Schreiber, Kyle Hogan, and Srinivas Devadas. AdVeil: A private targeted advertising ecosystem. *Cryptology ePrint Archive*, 2021.

[43] Sacha Servan-Schreiber, Simon Langowski, and Srinivas Devadas. Private approximate nearest neighbor search with sublinear communication. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 911–929. IEEE, 2022.

[44] Nihar B. Shah, K. V. Rashmi, and Kannan Ramchandran. One extra bit of download ensures perfectly private information retrieval. In *2014 IEEE International Symposium on Information Theory*, pages 856–860, 2014.

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

[46] Hua Sun and Syed Ali Jafar. The capacity of private information retrieval. *IEEE Transactions on Information Theory*, 63(7):4075–4088, 2017.

[47] Apache Hadoop Distributed File System. HDFS Erasure Coding. https://hadoop.apache.org/docs/r3.2.4/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html. Accessed: 2023-12-08.

[48] Razane Tajeddine, Oliver W Gnilke, and Salim El Rouayheb. Private information retrieval from MDS coded data in distributed storage systems. *IEEE Transactions on Information Theory*, 64(11):7081–7093, 2018.

[49] Razane Tajeddine, Oliver W Gnilke, David Karpuk, Ragnar Freij-Hollanti, and Camilla Hollanti. Private information retrieval from coded storage systems with colluding, byzantine, and unresponsive servers. *IEEE Transactions on information theory*, 65(6):3898–3906, 2019.

[50] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 299–313, 2017.

[51] Shangping Wang, Yinglong Zhang, and Yaling Zhang. A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems. *Ieee Access*, 6:38437–38450, 2018.

[52] David Woodruff and Sergey Yekhanin. A geometric approach to information-theoretic private information retrieval. In *20th Annual IEEE Conference on Computational Complexity (CCC'05)*, pages 275–284. IEEE, 2005.

[53] Yiwei Zhang and Gennian Ge. Private information retrieval from MDS coded databases with colluding servers under several variant models, 2017.

[54] Yiwei Zhang and Gennian Ge. A general private information retrieval scheme for MDS coded databases with colluding servers. *Des. Codes Cryptogr.*, 87(11):2611–2623, 2019.

[55] Ruida Zhou, Chao Tian, Hua Sun, and Tie Liu. Capacity-achieving private information retrieval codes from MDS-coded databases with minimum message size. *IEEE Transactions on Information Theory*, 66(8):4904–4916, 2020.

Appendix A. Correctness and Security Proofs of DPF Protocols

A.1. Section 4.1

Theorem 1. The DPF in Algorithm 1 is a $(p, 1, p-2)$-robust distributed point function.

Proof Sketch We prove the correctness and security of the tree-based DPF described in Section 4.1. We show the correctness and security of the scheme separately.

Correctness To prove correctness of $\mathsf{DPF.Gen}(1^\lambda, \alpha, \beta) \to (\mathcal{K}_0, \dots, \mathcal{K}_{p-1})$, it satisfies to show for pair of keys $\mathcal{K}_0, \mathcal{K}_j$, where $j \in [1, \dots, p-1]$ that the expanded versions of these keys represent the expanded random vectors $\vec{r}$ and $\vec{r} + j\beta \cdot \vec{e_\alpha}$. Note that if this holds, then the correctness of the full DPF directly follows, as this guarantees that all pairs of keys $\mathcal{K}_i, \mathcal{K}_j$ have the expanded random vector values $\vec{r} + i\beta \cdot \vec{e_\alpha}$ and $\vec{r} + j\beta \cdot \vec{e_\alpha}$.

To prove this, we demonstrate how to reduce the correctness guarantee to the same correctness argument in [10]. To do this for the specific pair of keys, there are three main steps. First, we show that the seeds and the j'th correction bit satisfies the same invariant as [10]. Second, we show the additional invariant that the other correctness bits are secret shares of zero. Finally, we show that the final conversion step indeed satisfies the above correctness requirement.

For the first two conditions, consider the two keys $\mathcal{K}_0 = s_0 || CW^{(1)} || ... || CW^{(n+1)}$ and $\mathcal{K}_j = s_j || CW^{(1)} || ... || CW^{(n+1)}$. Now, view the current seeds and correction bits as *secret shares* of a global seed. Specifically if the current seed at party 0 is s_0 and the current seed at party j is s_j, we can also view this as secret shares of some seed $[s] = s_0 \oplus s_j$. Additionally, we can view the correction bits are secret shares that obey the invariant $[t_j] = 1$, and $[t_k] = 0$ for all $k \neq j$.

We demonstrate that the invariants hold during the tree expansion via induction. Note that the base case clearly satisfy the invariants. Now consider the tree expansion at an arbitrary level in the tree on the special path. By our invariant, all correction bits except for the j'th one are identical. This implies that the contributions of everything except the j'th correction word are equivalent to adding secret shares of zero. This reduces the problem to a single pair of secret-shared seeds and a single secret-shared correction bit, which is equivalent to the original DPF construction in [10]. This is enough to claim that the j'th correction bit indeed satisfies the required invariant. Next, note that all other correction bits still satisfy the additional invariant since the values of those correction bits are not affected by the j'th correction word.

Finally, on the last step of the protocol, the values of the tree are converted to an output value. Note that on all non-special indices, the seeds and correction bits are fixed so the outputs will also be identical because the conversion is deterministic. Finally, for the special index, since all correction bits apart from the j'th one are secret shares of zero, they do not contribute anything to the final output value.

Now consider the contribution of the j'th correction word. There are two cases. Either $t_0^{(n)} = 0$ and $t_j^{(n)} = 1$ or vice versa.

1) In the first case, note that the output of key 0 is $\mathsf{Conv}(s^0)$ and the output of key j is $\mathsf{Conv}(s_j^n) = k * \beta$, which satisfies our correctness condition.
2) In the second case, the output of key 0 is $\mathsf{Conv}(s_j^{(n)})$ and the output of key j is $\mathsf{Conv}(s_j^{(n)}) + k\beta$, which also satisfies the correctness condition.

Security Our proof of security follows the blueprint of [10]. Specifically, we show that each party's key k_p is pseudorandom, assuming that an adversary only has access to a single key. This can be done using a standard hybrid argument, where in each step we replace one of the correction words $CW^{(i)} = \{CW_k^{(i)}\}_{k=1}^{p-1}$ from being the honestly generated correction word to a truly random string.

Explicitly, define the hybrid experiment in Algorithm 6, for $0 \leq j \leq n+1$. Note when $j = 0$ this is precisely the functionality described in DPF.Gen, while if $j = n+1$, then the key is truly random. Thus, it satisfies to show that two adjacent hybrid experiments are indistinguishable based on the security of PRG.

Algorithm 6 Hyb_j algorithm

```
$\mathsf{Hyb}_j(1^\lambda, p, \alpha, \beta)$:
1: Let $\alpha = \alpha_1, ..., \alpha_n \in \{0,1\}^n$ be the bit decomposition of $\alpha$.
2: Sample random seeds $s_0^{(0)}, ..., s_{p-1}^{(0)} \leftarrow \{0,1\}^\lambda$
3: for $i = 0, ..., p-1$ do
4:     For $d = 0, ...p-1$, set $t_{i,d}^{(0)} = 1$ if $i = d$, 0 otherwise.
5: for $i = 1, ..., n$ do
6:     $s_p^L || \{t_{p,k}^L\}_{k=1}^{p-1} \,||\, s_p^R || \{t_{p,k}^R\}_{k=1}^{p-1} \leftarrow G(s_p^{(i-1)})$
7:     If $\alpha_i = 0$ then $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$. Else $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
8:     If $i < j$, then uniformly sample $CW^{(i)} \leftarrow \{0,1\}^{(p-1)\lambda+2(p-1)}$
9:     Else, compute $CW^{(i)}$ according to Algorithm 1
10: if $j = n+1$, then $CW^{(n+1)} \leftarrow \mathbb{G}^{p-1}$. Else $CW_k^{(n+1)} \leftarrow (-1)^{t_{k,k}^{(n)}}[k \cdot \beta - \mathsf{Convert}(s_0^{(n)}) + \mathsf{Convert}(s_k^{(n)})]$ for $k = 1, ..., p-1$
11: Let $CW^{(n+1)} = \{CW_k^{(n+1)}\}_{k=1}^{p-1}$
12: Let $k_p = s_p^{(0)} || CW^{(1)} || ... || CW^{(n+1)}$
13: return $k_p$
```

To do so, consider a hybrid game distinguishing adversary $\mathcal{A}$ that is able to distinguish between Hyb_{j-1} and Hyb_j for arbitrary j. We construct a corresponding PRG adversary $\mathcal{B}$, in which the adversary $\mathcal{B}$ is given a value r which is either truly random, or sampled from a PRG with some random seed. Then the PRG adversary $\mathcal{B}$, for a fixed α, β can be constructed as follows:

- Let $s_p^{(0)} \leftarrow \{0,1\}^\lambda$ be random seeds, and define $\{t_{p,j}^{(0)}\}_{j=0}^{p-1}$ as defined in DPF.Gen.
- The first j correction words $CW^{(j)} = \{CW_k^{(j)}\}_{k=1}^{p-1}$ are chosen uniformly at random.
- for $i \leq j-1$ the seeds and control bits are generated honestly.
- For $i = j$, the seeds and control bits are set according to the random string r given from the PRG adversary, which is either a truly random string, or generated according to a PRG.
- For $i > j$, all remaining values are computed as defined in DPF.Gen, as a function of previous values.
- Output key k_p.

Consider $\mathcal{B}$'s success in the PRG game as a function of $\mathcal{A}$'s success in distinguishing Hyb_{j-1} from Hyb_j. If r was computed pseudorandomly, then it is clear that the generated k_p is distributed as Hyb_{j-1}.

Similarly, if r was sampled at random, then the generated k_p is distributed as Hyb_j. Specifically, given that r is random, then the computed values of $s_p^{(j)}$ and $CW^{(j)}$ are distributed randomly conditioned on the values of $s_p^{(0)}, CW^{(0)}, \ldots, CW^{(j-1)}$, and the t_p's satisfy the invariant as described in Section 4.1. This mechanically follows the steps taken in [10], which essentially demonstrates that the true random string behaves as a one-time pad on the existing values.

Combining these pieces together, we have that if r is random, then the resulting distribution of k_p is precisely distributed as Hyb_j. Thus, the advantage of $\mathcal{B}$ is equivalent to the advantage of $\mathcal{A}$, showing that the advantage of $\mathcal{A}$ is negligible as desired.

A.2. Correctness and Security of Section 4.3

Theorem 2. The DPF in Section 4.3 is a (p, t, r)-robust distributed point function for $p \geq \frac{dt}{2} + r + 1$.

Correctness Correctness directly follows from the correctness of Hermite interpolation.

Security Security directly follows from the fact that each DPF key is composed of $O(dN^{\frac{1}{d}})$ independently sample Shamir secret shares, which are information-theoretically secure.

A.3. Correctness and Security of Section 4.2

Theorem 3. The DPF in Section 4.2 is a (p, t, r)-robust distributed point function for $p \geq m + r + 1$, $m \geq t$.

Proof We prove the correctness and security of the covering design-based DPF described in Section 4.1. First, recall the definition of covering design:

Definition A.1. A $(p, m, t)-$covering design is a collection of q subsets $A_1, \ldots, A_q$, each of size m, such that for any subset $T \subset [p]$ of size t, there exists an A_i such that $T \subseteq A_i$. We now show the correctness and security of the scheme separately.

Correctness Recall that the covering design based DPF scheme generates q keys with black-box use of a multiparty DPF scheme. It thus satisfies to show that for each of the q keys $\mathcal{K}_i$, the client is able to retrieve the evaluation from at least one of the $p - r$ responding servers.

Consider any arbitrary multiparty-DPF key $\mathcal{K}_j, j \in [q]$. By construction of our covering design scheme, this key is received by exactly $p - m \geq (m + r + 1) - m \geq r + 1$ parties. Thus, by pigeonhole principle, at least one of the $p - r$ responding srevers contains the evaluation of $\mathcal{K}_j$.

Because this is true for every key, correctness immediately follows due to the correctness of the underlying $(p, p-1)$-multiparty DPF.

Security We reduce the security of the DPF scheme described in Section 4.2 to the security of the underlying $(p, p-1)$-multiparty DPF, which was used in a black-box manner in the scheme.

Consider any arbitrary adversary that corrupts any random subset of t parties $T = \{i_1, ..., i_t\} \in [p]$. By definition of covering design, there exists at least one out of the q keys, $\mathcal{K}_j$, that is not held by any of these t parties. This is due to the fact that for any arbitrary subset, there exists a corresponding A_j such that $T \in A_j$ by definition of covering design. Because A_j is a cover for the corrupted parties and the corresponding key is only given to the parties not contained in the cover, this implies that the adversary only holds at most $q-1$ out of q multiparty-DPF keys. Hence, security immediately follows due to the security of the underlying multiparty DPF scheme.

Appendix B. Correctness of PIR Protocols

We now prove the correctness of each of our PIR schemes. Note that the security of these schemes immediately follows from the underlying DPF protocols, whose security were proven in the previous section.

B.1. Correctness of Algorithm 3

Theorem 4. The scheme in Algorithm 3 is a (p, t, r, b)-robust PIR protocol.

Proof We prove the result for a specific case that only requires the client to prepare one set of DPF keys. Note that this proof that can be generalized so that the protocol requires fewer servers at the cost of additional linear queries / per-server computation. We refer the reader to [49] for details, and note that the proof can be directly generalized to this setting.

Assume that the client is querying for an index i. Then, consider the key $\mathcal{K}_j$ and the server contents $\mathcal{X}_j$ at some Server j. It satisfies to show that the response from this server is a degree k polynomial evaluated at j, and k of the coefficients are equal to $X[i]_1, ..., X[i]_k$. Let $\mathsf{DPF.Eval}(\mathcal{K}_j)$ be equivalent to the complete expansion of the key on every index from 1 to N. More specifically, $\mathsf{DPF.Eval}(\mathcal{K}_j) = \vec{r} + j^k \cdot \vec{e_i}$ for some random vector $\vec{r}$.

Next, note that this means that the response from Server j is equal to the value

$$\begin{aligned}\mathsf{resp}_j &= \mathsf{DPF.Eval}(\mathcal{K}_j) \cdot \mathcal{X}_j \\ &= (\vec{r} + j^k \cdot \vec{e_i}) \cdot \mathcal{X}_j\end{aligned}$$

Letting $f_i(j) = X[i]_1 + X[i]_2 \cdot j + ... + X[i]_k \cdot j^{k-1}$ equal the degree $k-1$ polynomial that is equal to the erasure-coded representation of file i stored at Server j, we can rewrite the response of Server j as follows:

$$\begin{aligned}\mathsf{resp}_j &= (\vec{r} + j^k \cdot \vec{e_i}) \cdot \mathcal{X}_j \\ &= \sum_{i=1}^{N} r_i \cdot f_i(j) + j^k \cdot f_i(j) \\ &= \sum_{i=1}^{N} r_i \cdot f_i(j) + X[i]_1 j^k + ... + X[i]_k j^{2k-1}\end{aligned}$$

The above is a degree $2k-1$ polynomial where the coefficients of the last k terms are indeed the k shards of the file looking to be recovered. Hence, with $2k + 2b$ responses, the file can be recovered via Berlekamp-Welch with probability 1.

B.2. Correctness of Algorithm 4

Proof We prove the correctness result for the specific case $d = 2$ for ease of readability, but correctness for general d is proven identically. For a database X, each Server i, $i \in [\frac{dt}{2} + k + r + 2b]$ stores the following $2N$ values:

$$S_i = \begin{cases} \{f_i(x) = X[i]_1 + X[i]_2 \cdot x + \cdots + \cdot X[i]_k \cdot x^{k-1}\}_{i=1}^N \\ \{f_i'(x) = X[i]_2 + \cdots + (k-1) \cdot X[i]_k \cdot x^{k-2}\}_{i=1}^N \end{cases}$$

When querying for an index $\alpha = (\alpha_1 || \alpha_2)$, such that $\alpha_1 \sqrt{N} + \alpha_2 = \alpha$, the client prepares a key according to Algorithm 2. On each Server j, the server now has the necessary information to compute the following $\sqrt{N} + 2$ values, which are sent to the client:

- $\mathsf{val(j)} = \sum_{i=1}^{N} r_1[i_1](j) r_2[i_2](j) f_i(j)$
- $A_{i_1}(j) = \sum_{i_2=1}^{\sqrt{N}} r_2[i_2](j) f_{i_1||i_2}(j)$ for $i_1 = 1, \ldots, \sqrt{N}$
- $B_{i_2}(j) = \sum_{i_1=1}^{\sqrt{N}} r_1[i_1](j) f_{i_1||i_2}(j)$ for $i_2 = 1, \ldots, \sqrt{N}$
- $\mathsf{val'(j)} = \sum_{i=1}^{N} r_1[i_1](j) r_2[i_2](j) f_i'(j)$

Since at most r servers are non-responsive, the client receives at least $q = \frac{dt}{2} + k + 2b$ responses, where at most b of these are Byzantine. WIthout loss of generality, assume that these responses are from the first q parties. Then note that the client now has the values $\mathsf{res(j)}_{j=1}^q = \mathsf{val(j)}_{j=1}^q$, and has enough information to locally compute the values:

$$\begin{aligned}\mathsf{res'(j)} &= \sum_{i=1}^{N} r_1'[i_1] r_2[i_2] f_i(j) + \sum_{i=1}^{N} r_1[i_1] r_2'[i_2] f_i(j) \\ &\quad + \sum_{i=1}^{N} r_1[i_1] r_2[i_2] f_i'(j) \\ &= \sum_{i_1=1}^{\sqrt{N}} r_1'[i_1](j) A_{i_1}(j) + \sum_{i_2=1}^{\sqrt{N}} r_2'[i_2](j) B_{i_2}(j) + \mathsf{val}'(j)\end{aligned}$$

With the $2q$ values $\mathsf{res(j)}$ and $\mathsf{res}'(j)$, the client can now perform Hermite interpolation to recover their desired result.

B.3. Correctness of Algorithm 5

Theorem 5. The scheme in Algorithm 5 is a (p, t, r, b)-robust PIR protocol for $p \geq m + k + r + 2b, m \geq t$

Proof Assume that the client is querying for the value $X[i]$, and it is located in the δ'th portion of the database. First, recall that the response from each server j is the following:

$$\mathsf{resp}_j = \{\sum_{i=1}^{\frac{N}{k}} \mathsf{DPF.Eval}(\mathcal{K}_\alpha, i) \cdot \mathcal{X}_j[i] \mid \mathcal{K}_\alpha \in \sigma_j\}$$

where σ_j represents the set of keys received by party j. This is equivalent to the following:

$$\mathsf{resp}_j = \{f_\alpha(j) \mid \mathcal{K}_\alpha \in \sigma_j\}$$

where $f_\alpha(j) = \sum_{i=1}^{\frac{N}{k}} \mathsf{DPF.Eval}(\mathcal{K}_\alpha, i) \cdot \mathcal{X}_j[i]$ is a degree $k-1$ polynomial.

Consider the portions of the responses that correspond to an arbitrary key $\mathcal{K}_\alpha$. By the properties of covering design, this key is received by $p - m = k + r + 2b$ parties. Because at most r parties are non-responsive, this implies that the client receives at least $p' = k + 2b$ responses for this key. Denote these parties $i_1, ..., i_{p'}$.

Consider the set of p' values $f_\alpha(i_1), ..., f_\alpha(i_{p'})$. These are evaluations of the function

$$\begin{aligned} f_\alpha(x) &= \sum_{i=1}^{\frac{N}{k}} \mathsf{DPF.Eval}(\mathcal{K}_\alpha, i) \cdot \mathcal{X}_x[i] \\ &= \sum_{i=1}^{\frac{N}{k}} \mathsf{DPF.Eval}(\mathcal{K}_\alpha, i) \cdot \mathcal{X}_x[i] \\ &= \sum_{i=1}^{\frac{N}{k}} \mathsf{DPF.Eval}(\mathcal{K}_\alpha, i) \cdot (X_1[i] + ... + X_k[i] \cdot x^{k-1}) \end{aligned}$$

Letting $\vec{v}$ represent the expanded vector of the DPF key, we can rewrite the above as follows:

$$f_\alpha(x) = \vec{v} \cdot X_1 + (\vec{v} \cdot X_2)x + ... + (\vec{v} \cdot X_k)x^{k-1}$$

Because this is a degree $k - 1$ polynomial, $k + 2b$ responses are enough to interpolate this polynomial via Berlekamp-Welch. Hence, we can retrieve the value $\vec{v} \cdot X_\delta$ for any δ. This means that the client can retrieve the value $\sum_{i=1}^{\frac{N}{k}} \mathsf{DPF.Eval}(\mathcal{K}_\alpha, i) \cdot X_\delta[i]$. for any δ.

Since the above holds true for each of the q keys, correctness immediately follows due to the correctness of the underlying multiparty DPF, allowing the client to retrieve the value $X[i]$ as desired.

.

Appendix C. Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

This paper presents new fault-tolerant PIR constructions with smaller communication costs, along with lower server-side storage overheads and computation costs, compared to prior work. To design these schemes, the authors combine techniques from the cryptography literature (namely, distributed point functions, which give rise to the most communication-efficient PIR schemes) and from the information-theory literature (namely, maximum distance separable codes, which reduce the server-side storage). The downside of these schemes is that they rely on a relatively large number of non-colluding servers to host the PIR database, though this setting is motivated in decentralized, peer-to-peer applications.

C.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field

C.3. Reasons for Acceptance

1) A prior version of this submission was invited to be resubmitted following major revisions during the 2023 review cycle. The authors put significant effort into the reviewers requests. In comparison to the original paper, this work emphasizes fault tolerance for multi-server PIR protocols as the main contribution of the work and provides clearly comparisons to past protocols. The authors also expanded their evaluation and reconfigured experiments to be more realistic, as well as providing an experimental comparison between the costs of the new PIR schemes and the costs of using DPFs + MACs. Reviewers appreciated the expanded treatment of related work and the additional experimentation.