Computationally Secure Aggregation and Private Information Retrieval in the Shuffle Model

Adrià Gascón
Google

Yuval Ishai
Technion

Mahimna Kelkar
Cornell University

Baiyu Li
Google

Yiping Ma
University of Pennsylvania

Mariana Raykova
Google

June 1, 2024

Abstract

The *shuffle model* has recently emerged as a popular setting for differential privacy, where clients can communicate with a central server using anonymous channels or an intermediate message shuffler. This model was also explored in the context of cryptographic tasks such as *secure aggregation* and *private information retrieval* (PIR). However, this study was almost entirely restricted to the stringent notion of information-theoretic security.

In this work, we study *computationally secure* aggregation protocols and PIR in the shuffle model. Our starting point is the insight that the previous technique of shuffling additive shares can be improved in the computational setting. We show that this indeed holds under the standard learning parity with noise (LPN) assumption, but even better efficiency follows from plausible conjectures about the *multi-disjoint syndrome decoding* (MDSD) problem that we introduce and study in this work.

We leverage the above towards improving the efficiency of secure aggregation and PIR in the shuffle model. For secure aggregation of long vectors, our protocols require $9\times$–$25\times$ less communication than the previous information-theoretic solutions. Our PIR protocols enjoy the simplicity and concrete efficiency benefits of multi-server PIR while only requiring a single server to store the database. Under the MDSD assumption, they improve over recent single-server PIR constructions by up to two orders of magnitude.

Contents

1 Introduction

Anonymous communication, as a means for protecting the identities of communicating parties, is an important privacy tool that has been extensively studied from the perspectives of both constructions and applications [Cha81, Cha88, GT96, GRS99, Nef01, FM02, DMS04, GJ04, CL05]. Somewhat less expectedly, anonymous communication can also serve to enhance the *efficiency* of privacy primitives that seem unrelated to hiding identities. This idea was first put forward by Ishai, Kushilevitz, Ostrovsky and Sahai [IKOS06], who suggested the use of anonymity for enhancing secure computation and private information retrieval protocols.

More recently, in the context of differential privacy, several works observed that allowing clients to simultaneously send anonymous messages to a central server can lead to significantly improved privacy-utility tradeoffs [BEM+17, EFM+19, CSU+19, BBGN19]. A simple practical implementation that was suggested in this context is via the use of a *shuffler*: an intermediate entity that receives from each client one or more encrypted messages (under a secret key known only to the server), and forwards a random permutation of the received ciphertexts to the server[1]. The simplicity and efficiency of this approach inspired a rich line of work on differential privacy in the *shuffle model*; see, e.g., [Che21, CZ22] and references therein.

Almost all of the above works (with the exception of a feasibility result from [IKOS06], see Section 1.2) only consider the stringent notion of information-theoretic security in the shuffle model. In the present work, we study the concrete efficiency benefits of settling for *computational security.* We show that this relaxation unlocks much more efficient solutions for problems that involve processing inputs or requests from a large number of clients, including secure aggregation and private information retrieval.

The split-and-mix construction. A core building block in our solutions is a "split-and-mix" construction from [IKOS06] that takes c inputs from a finite Abelian group, generates k additive shares of each, and outputs the shuffled ck shares. This construction can be naturally used to realize secure aggregation in the shuffle model. The main technical question is how large k should be, as a function of c and the group size, such that the shuffled additive shares reveal essentially nothing except the sum of the c inputs. In the information-theoretic setting, a nearly tight answer was given in [IKOS06, GMPV20, BBGN20]. In this work, we initiate the study of the computational version of this problem. We show that the parameters of the information-theoretic solution can be improved under standard assumptions on the hardness of the *syndrome decoding* problem [BMvT78, McE78], an equivalent dual version of the *learning parity with noise* (LPN) problem [BFKL94].

Towards further improving efficiency, we formulate and study the *multi-disjoint syndrome decoding (MDSD)* problem, a variant of the previously studied multi-syndrome decoding (MSD) problem [Sen11], and show that it tightly captures the computational security of the split-and-mix construction. We conjecture that, in the relevant parameter regime, the MDSD problem is roughly as hard as MSD, for which the best known attack is the *Decoding One Out of Many* (DOOM) algorithm from [Sen11]. This conjecture is guided by the intuition that the weak extra structure in MDSD (compared to standard MSD) is similar in spirit to the weak extra structure in the regular-noise variant of the standard LPN problem [AFS03]. The latter is commonly believed to be as hard as standard LPN in most parameter regimes, even though only loose security reductions are known [LWYY22]. In light of the above, we propose concrete security parameters for split-and-mix that suffice to defeat the DOOM attack. We extend our analysis to other variants of the construction that compress the first $k-1$ additive shares of each secret by using short seeds, and include an additional dummy secret that may be split into a bigger number of shares. These variants are motivated by the applications we discuss next.

The split-and-mix construction is at the heart of the two main applications we consider in this work: *secure aggregation* and *private information retrieval* (PIR) in the shuffle model. In both cases we require anonymous communication, which may be implemented by a shuffler, between many clients and a single server. For secure aggregation, we only require one-way communication from the clients to the server. For

[1]This assumes that the shuffler and the server do not collude, which is similar to non-collusion assumptions made in other cryptographic contexts. This assumption can be realistic when the shuffler and server are run by different entities. It can be further relaxed by concatenating two or more shufflers, as is done in the context of mix networks and voting protocols [Cha81].

PIR, we further require that the server can respond to clients via the same shuffler. We now discuss the two applications in more detail.

Secure aggregation in the shuffle model. A secure aggregation protocol enables the server to learn the sum of the inputs provided by many clients without learning anything else about the inputs. While this summation can be carried out in any finite Abelian group, we will mostly consider aggregation of long vectors in $\mathbb{Z}_p^n$. Aggregating long vectors is commonly seen in applications like private histogram statistics [CGB17, BBCG+21, AGJ+22, ZMA22], or the context of federated learning [MMR+17], where aggregate gradient updates (often of dimension 10K or more) needs to be summed up for model training. Existing standard-model single-server constructions [BIK+17, BBG+20, MWA+23, BGL+23, LLPT23] require multiple rounds of interaction between the server and the clients, or a subset of clients that can handle a large communication proportional to the number of users and is available for online interactions throughout the execution of the protocol. There are also protocols that require only one message from each client but rely on two non-colluding servers [CGB17, BBCG+21]. In the setting where we require only differential privacy guarantees for the aggregated output, there is a line of works leveraging the shuffle model to obtain non-interactive solutions with a single server. These solutions [GMPV20, BBGN20] rely on the information-theoretic variant of the split-and-mix problem.

Our improved parameters for the computational setting directly yield similar improvements for secure aggregation. The resulting construction requires each client to split its input vector into additive shares (compressing all but one share), and send the shares to the server through the anonymous communication channels. Asymptotically, the number of shares per client in our protocol is $\mathsf{poly}\log(n)$ in the input length n, and by compressing all but one shares we achieve nearly the optimal communication rate 1. In contrast, the optimal information theoretic analysis achieves communication rate of $\log(c)/n$. For concrete settings, our protocol achieves $9\times$ to $25\times$ saving in terms of communication, even when optimizing the information-theoretic solution by compressing all but one of the shares, or aggregating each vector entry separately. Moreover, similarly to the information-theoretic solution, this aggregation protocol can make a black-box use of an arbitrary Abelian group (we conjecture that the MDSD parameters for $\mathbb{Z}_2$ suffice in the general case). This black-box use of groups can support, for example, point addition in bilinear groups, which is required by recent shuffle model protocols from [HIKR23]. See Section 4 for more details on our secure aggregation protocol and Section 6 for concrete evaluation results.

We also show how to reduce aggregation on long inputs to aggregation of short inputs by leveraging RWLE encryption [Reg05] which provides both key and message homomorphism. The core idea is to have clients submit their inputs encrypted under different keys and run secure aggregation to provide the server with the aggregated key; this lets the server decrypt only the aggregated input. Note however that unlike the direct MDSD-based construction, which makes a black-box use of the underlying group, the performance of the RLWE-based construction is more sensitive to the choice of group. Section 4.1 details the RLWE-based construction while Section 6 compares our different aggregation constructions.

PIR in the shuffle model. A PIR scheme [CGKS95, KO97] enables a client to retrieve a selected item from a large database while hiding the identity of the retrieved item from one or more servers storing the database. The main challenge in PIR research is to make the communication cost sublinear in the database size, while also minimizing the server computation overhead. There has been a flurry of recent works on practical solutions to PIR, either with or without database preprocessing; see Section 1.2 for a survey of some relevant works.

Our starting point is a simple technique for multi-server PIR from [CGKS95], whose communication cost is roughly the square root of the database size. Viewing the database as a matrix $X \in \mathbb{F}_2^{m \times m}$, which is held by $k \geq 2$ servers, the client randomly splits a unit vector $q \in \mathbb{F}_2^m$ into k additive shares q_i, and each server responds with the matrix-vector product $X \cdot q_i$. Adding up the answers, the client obtains an entire column of X without revealing the identity of the chosen column using $O(m)$ bits of communication.

The above PIR scheme is attractive due to its lightweight server computation. Without preprocessing of the database, the computational cost of PIR must inherently be linear in the database size [BIM00]. Here each server only needs to compute the XOR of (roughly) half the columns of X. The main downside, compared to single-server PIR schemes, is that the servers need to hold synchronized copies of the database, and are further assumed not to collude.

Our simple observation is that the above multi-server scheme can be implemented almost directly using a single server in the shuffle model. Here the same server plays the role of all k servers in the multi-server scheme, where security is achieved by mixing the sub-queries of different clients. Using a small number of additional dummy shares to hide the sum of the client inputs, the security of this construction reduces to the MDSD assumption.

Compared to the multi-server scheme, this solution has the big advantage of only requiring a single server to store the database. Furthermore, while a shuffler-based implementation still needs to assume that the server and the shuffler do not collude, this is arguably more realistic than assuming non-collusion of servers that need to maintain synchronized copies of the same database. The non-collusion assumption can be further relaxed by employing multiple shufflers or other anonymous communication techniques.

The concrete efficiency of this construction is directly influenced by the choice of k. Based on our conjectured parameters for the MDSD problem, $k \approx 20$ suffices with a large number of clients. We achieve further improvements by applying several optimization techniques. The bandwidth is reduced by compressing $k-1$ out of the k sub-queries sent by each client. The computational overhead is improved by batch-processing sub-queries and by processing the first $k-1$ sub-queries of each client offline, before the online query is known and during idle times of the server. Even without these optimizations, the simplicity of this approach and the small number of bit-operations performed by the server make it competitive with state-of-the-art single-server PIR schemes, including ones that employ expensive preprocessing. The online phase of our optimized implementation outperforms the best current single-server PIR schemes [MW22, HHC$^+$23, LMRSW23] by one to two orders of magnitude with respect to most relevant metrics. See Section 6 for a more detailed comparison.

PIR with proxy. If the shuffle model is instantiated in practice with a dedicated proxy that mixes all queries, a natural question is what we can achieve by having the proxy do more than shuffling but still without requiring it to store the database. We show a proxy-PIR construction which enables the proxy to generate PIR queries from an encryption of the query index sent by the client, and then combine the answers from the server into a single response sent to the client. The proxy still learns nothing about the query index or the response. Our construction achieves optimal upload and download size per query. See Section B for details.

PIR with variable-sized records. Real-world databases typically contain records in a variety of sizes, and revealing the record size can reveal sensitive information about the record identity. Unfortunately, in the usual PIR setting, nothing better can be done apart from padding all records to the same size and having clients retrieve the padded records; this poses an undue communication cost if the majority of clients only wish to retrieve small records.

In Section 7, we demonstrate that the shuffle model enables PIR for variable-sized records with significant savings in total communication, leaking *only* the total size of all the queried records. We show novel approach for clients to split a query for record size ℓ into $O(\log \ell)$ queries for records of smaller sizes, in which the size of any individual retrieved record can be hidden by shuffling without any database padding. Moreover, our construction only makes a black-box use of single-server PIR protocols.

On realizing and extending the shuffle model. While there are many different approaches to realize anonymous communication channels, a number of practical applications assume a dedicated non-colluding shuffler party that processes the clients' messages [App, Chr, Clo]. With this shuffler instantiation, our secure aggregation and PIR protocols can be further optimized by leveraging the shuffler to insert and shuffle dummy shares together with the clients' inputs. These dummy shares are only necessary for the security of the protocol with respect to the server, but do not influence the outputs. This helps reduce the communication of the clients by requiring a smaller number of shares for each input, and can be used to guarantee a sufficient level of security even when there is a small number of participating clients. Finally, the shuffler can potentially perform other actions beyond inserting dummy shares. This is exploited in the proxy-PIR construction discussed above.

1.1 Our Contributions

Our new techniques and constructions include:

Reducing Split-and-Mix to MDSD. We initiate the study of the computational security of the "split-and-mix" construction, which is core to multiple applications. We formulate the problem of multi-disjoint syndrome decoding (MDSD) and we show that it suffices for the security of this primitive.

Security of MDSD. We show a reduction from the well studied syndrome decoding problem to MDSD. Since this reduction is not tight, we also put forward a conjecture that the DOOM attack provides a tight bound on the complexity of MDSD. We provide the intuition and analysis for this conjecture.

Secure aggregation in the shuffle model. We present new analysis for shuffle-model secure aggregation constructions in the computational setting, which have been analyzed before only in the information-theoretic setting. The proof of security leverages our core "split-and-mix" lemma. This construction approaches the optimal communication rate of 1. We also present a reduction from secure aggregation on long vectors to aggregation on short vectors with the help of RLWE encryption. The resulting construction reduces the computation cost, typically with a small increase of communication cost.

PIR in the shuffle model. As a second application of our computational "split-and-mix" analysis, we present a new single-server PIR protocol in the shuffle model, namely where the server obtains a shuffle of all queries made by the clients. Our protocol offers significant efficiency advantages over existing single-server PIR protocols without requiring database replication, but at the cost of requiring an additional shuffler who does not collude with the server. We propose several optimizations that improve the concrete efficiency. In Section 7, we additionally demonstrate the possibility of leveraging the shuffle model towards better solutions for PIR with *variable-size records*, revealing only the total size of all retrieved records.

We implement our constructions, conduct extensive benchmarks and compare their concrete efficiency with the state of the art.

Empirical Results. We implemented our secure aggregation and PIR protocols and evaluated their concrete efficiency. For secure aggregation, we measured the improvement compared to the previous information-theoretic analysis. Our advantage increases with the number of clients and the input size. For example, for input vectors of dimensions 2^{15}–2^{23} we get 9–25× improvement in communication. For many realistic choices of parameters, the expansion ratio[2] of our construction is very close to 1 which is optimal. Our RLWE hybrid-variant only requires secure aggregation of short vectors (of length 2^{10} to 2^{11}) combined with RLWE encryptions of the inputs. The concrete communication for this hybrid protocol is worse than our MDSD-based construction, which has close to optimal communication. However, it achieves better computation for the server with 170×–590× speedups for different parameters.

For the PIR benchmarks, we considered databases of total size in the range of 0.2–8 GiB, arranged in different shapes determined by record size vs. number of records. Both the (amortized) communication and computation costs of our protocol improve as the number of clients increases. Since several of the single-server schemes that we compared with are in the preprocessing model, where there is a significant offline cost of communicating hints, we also split our communication into online and offline. More specifically, each client's query consists of several sub-queries, all but one of which can be sent and processed before the query is available and thus the online communication consists of a single sub-query. The distinction between online and offline costs makes sense even if those do not amortize across queries from the same client, since offline queries can typically be run during idle periods for the server. We note that this kind of per-query preprocessing is qualitatively worse than the one-time hint-based preprocessing required in several other PIR schemes. However, even when adding this preprocessing cost to the online cost, our construction is still competitive. We expect that in a typical situation where the server load has high variability, off-peak times can be used to process the offline queries and enable very fast processing of online queries during peak times.

Existing single-server PIR constructions obtain different trade-offs between communication and computation costs. Spiral [MW22] minimizes communication, but at the cost of relatively high computation, while SimplePIR [HHC+23] optimized throughput at the cost of large preprocessing communication costs. The

[2]Expansion ratio measures the communication overhead of the protocol over the input vector size.

online throughput of shuffle PIR improves the throughput of SimplePIR by 5–7× (20–25× with batched processing). Spiral is more than 10× slower than SimplePIR and thus 200–300× times slower than the online computation for ShufflePIR, and is also substantially slower even if we consider the total online and offline computation for shuffle PIR. Concretely, the online computation of shuffle PIR in our experiments is 7–220ms per client.

In terms of communication, our online communication is close to that of Spiral, which is 3–13× less than the online communication of SimplePIR for different parameter settings. The total online and offline communication of ShufflePIR is better than the total communication for SimplePIR when clients have small number of queries. Our proxy PIR constructions achieve upload cost of a single ciphertext encrypting the query index and download of a single ciphertext encrypting the single retrieved record.

1.2 Related Work

In this section we discuss the relevant works for our comparison.

Shuffle Model. The shuffle model has been widely studied in the literature, spanning across differential privacy, secure aggregation and anonymous systems. In recent years, this model has seen increasing popularity due to its use in differential privacy applications (see, e.g., [BEM+17, Che21]). In this work we consider PIR and secure aggregation, and study the extent to which settling for *computational security* can be used to improve previous results from [BBGN20, GMPV20, IKOS06]. We expect our approach to have relevance to differential privacy as well, and leave this direction to future work.

Secure Aggregation. A number of works consider the problem of enabling a server to learn the sum of inputs from a set of clients without learning anything else. These works can be categorized by their communication models. Some constructions [BIK+17, BBG+20, BGL+23] (designed for distributed learning settings) assume clients can only communicate with the server and are multi-round protocols. New committee-based construction [MWA+23, LLPT23] in the similar communication model alleviate the multi-round requirement for most of the clients but at the cost of an assumption that a subset of the clients can be available online throughout the execution of the protocol and be able to handle communication linear in the number of parties.

Differently, single-server aggregation in the shuffle model can be made non-interactive (single round). Ishai et al. [IKOS06] first proposed an information-theoretic protocol for adding c inputs over a group of size p in the shuffle model, where statistical security of $2^{-\sigma}$ can be achieved by using $k = O(\log c + \log p + \sigma)$ additive shares per input. A line of subsequent works improved on this bound [IKOS06, BEM+17, EFM+19, CSU+19, BBGN19], culminating in a nearly tight analysis [GMPV20, BBGN20] of the "split-and-mix" protocol in the information-theoretic setting, showing that $k = \Theta(\frac{2\sigma+\log p}{\log c})$ suffices. In this work we show that, settling for computational security, better efficiency follows from the LPN or MDSD assumptions.

When inputs are high-dimensional, our construction's communication outperforms these works by orders of magnitude. This is particularly useful for machine learning applications to aggregate model weights. Note that there are two straightforward optimizations to the information-theoretic solution: compressing all but one shares using PRG seeds, and breaking a long vector into shorter segments that are aggregated separately. Our construction still outperforms the information-theoretic solution even when these optimization are equally applied, thanks to the much smaller number of shares ($\mathsf{poly}\log(n)$ vs. n) per input for any given input length n to guarantee the computational hardness of MDSD problem.

Private Information Retrieval. While the shuffle model is widely used in the area of differential privacy, only few works have studied PIR in this model. Here, the clients are granted the ability to make anonymous PIR queries to a single server. Ishai et al. [IKOS06] proposed a computationally secure construction under a non-standard assumption (the hardness of reconstructing noisy low-degree curves in a low-dimensional space), where the main goal is obtaining single-server PIR with sublinear server computation per query following a one-time polynomial-time preprocessing. This problem was recently solved by Lin et al. [LMW23] in the standard model under the standard RLWE assumption. An unpublished recent work by Ishai, Kelkar, Lee and Ma [IKLM] gave an *information-theoretic* construction for PIR in the shuffle model. However, all these schemes are prohibitively expensive in practice, either having concretely high server storage and computation overhead [IKOS06, LMW23] or requiring a huge number of clients to query simultaneously [IKLM]. Our goal is to obtain PIR schemes with *concretely* fast server computation, even without preprocessing.

Another line of work [TDG16, AIVG22] considers the differential privacy (DP) notion for PIR. Here, clients send their queries anonymously to the server, and privacy is guaranteed by the shuffling of the queries along with some noise queries. The DP notion guarantees that the server cannot distinguish neighboring sets of queries (i.e., differing in exactly one client). Unfortunately, DP is substantially weaker than the standard PIR security definition and therefore insufficient in any applications where client queries can be strongly correlated, which is often the case in practice.

To show the efficiency improvement, we compare our scheme with state-of-the-art single-server PIR constructions including 1) Simple PIR [HHC+23], which optimizes server online computation at the cost of preprocessing; 2) Hintless PIR [LMRSW23], which removed the preprocessing of Simple PIR with some additional online cost; and 3) Spiral PIR [MW22], which optimized online communication. FrodoPIR [DPC23] is a concurrent work similar to SimplePIR, which has better download size but inferior throughput. There are other works [PPY18, ZPSZ23] that have fast server computation, but require clients to download the entire database in an offline phase (but with sublinear client storage).

2 Preliminaries

We use κ to denote the computational security parameter, and σ, the statistical security parameter. For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. By default, vectors in this paper are assumed to be column vectors. For matrix $\mathbf{A}$, we use $\mathbf{a}_1, \mathbf{a}_2, \ldots$ to denote its columns, and $\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \ldots$ for its rows. $\mathbf{A}$ can also be written as the horizontal concatenation $[\mathbf{a}_1, \ldots, \mathbf{a}_n]$ of its columns, or the vertical concatenation $(\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)})$ of its rows.

For a distribution $\mathcal{D}$, we use $s \leftarrow \mathcal{D}$ to denote sampling from $\mathcal{D}$. Given a set S, we use $s \leftarrow\!\!\$\; S$ to denote uniformly sampling from S. We use $\mathbb{I}$ to denote the characteristic function of a random variable. For two distributions $\mathcal{D}_1, \mathcal{D}_2$, we use $\mathsf{s.d.}(\mathcal{D}_1, \mathcal{D}_2)$ to denote the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$.

2.1 Coding Theory

We will need some basic results in coding theory. Let $\mathbb{F}$ be a finite field. The Hamming distance between any two vectors $\mathbf{v}, \mathbf{u} \in \mathbb{F}^n$, denoted by $\Delta(\mathbf{v}, \mathbf{u})$, is the number of coordinates that they differ: $\Delta(\mathbf{v}, \mathbf{u}) := |\{i \mid v_i \neq u_i\}|$. The Hamming weight of a vector $\mathbf{v} \in \mathbb{F}^n$, denoted by $\Delta(\mathbf{v})$, is the number of non-zero entries in $\mathbf{v}$. Clearly, we have $\Delta(\mathbf{u}, \mathbf{v}) = \Delta(\mathbf{u} - \mathbf{v}) = \Delta(\mathbf{v} - \mathbf{u})$ for all $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$.

A linear code $\mathcal{C}$ of length m and dimension k is a subspace of $\mathbb{F}^m$, and its elements are usually called codewords. The distance of a linear code $\mathcal{C}$, denoted by $\Delta(\mathcal{C})$, is the minimum Hamming weight of non-zero codewords of $\mathcal{C}$. Since $\mathcal{C}$ is a linear subspace, we have $\Delta(\mathcal{C}) = \min_{\mathbf{u} \neq \mathbf{v} \in \mathcal{C}} \Delta(\mathbf{u}, \mathbf{v})$. The code $\mathcal{C}$ can be represented by its generator $\mathbf{G} \in \mathbb{F}^{k \times m}$, whose columns form a basis of $\mathcal{C}$, such that $\mathcal{C} = \{\mathbf{x}^T\mathbf{G} \mid \mathbf{x} \in \mathbb{F}^k\}$. Given a matrix $\mathbf{G} \in \mathbb{F}^{k \times m}$, we write $\mathcal{C}(\mathbf{G}) = \{\mathbf{x}^T\mathbf{G} \mid \mathbf{x} \in \mathbb{F}^k\}$ for the code generated by $\mathbf{G}$. Alternatively, $\mathcal{C}$ can also be represented by its *parity check matrix* $\mathbf{H} \in \mathbb{F}^{m \times (m-k)}$ whose kernel is $\mathcal{C}$, i.e. $\mathbf{H} \cdot \mathbf{y} = 0$ for all $\mathbf{y} \in \mathcal{C}$. Note that $\mathbf{G} \cdot \mathbf{H} = \mathbf{0}$, and that the minimal distance $\Delta(\mathcal{C}(\mathbf{G}))$ is exactly the minimal number of columns of $\mathbf{H}$ that are linearly dependent. Furthermore, for randomly sampled generator matrix $\mathbf{G}$, its corresponding parity check matrix $\mathbf{H}$ is also uniformly random over $\mathbb{F}^{m \times (m-k)}$. Note that we choose to write the generator $\mathbf{G}$ in the "transposed" form where rows of $\mathbf{G}$ spans the code; this is more convenient for us since we will mostly working with its parity check matrix $\mathbf{H}$.

For finite field $\mathbb{F}_q$, define the entropy function h_q as

$$h_q(x) = x \log_q(q-1) - x \log_q(x) - (1-x) \log_q(1-x).$$

This is a generalization of the binary entropy function $h_2(x) = -x \log(x) - (1-x) \log(1-x)$ to arbitrary finite field $\mathbb{F}_q$. For random linear codes over $\mathbb{F}_q$, the celebrated Gilbert-Varshamov lemma gives a high probability bound on the existence of a solution $\mathbf{e} \in \mathbb{F}_q^m$ such that $\mathbf{H} \cdot \mathbf{e} = \mathbf{z}$ for a random $\mathbf{H}$ and a given $\mathbf{z}$.

Lemma 1 (Gilbert-Varshamov Bound for Random Linear Code)**.** *Let $m > 1$, q a prime power, $\delta \in [0, 1 - 1/q)$, and $0 < \epsilon < 1 - h_q(\delta)$. If $k = \lceil (1 - h_q(\delta) - \epsilon)m \rceil$, then*

$$\Pr_{\mathbf{G} \leftarrow \mathbb{F}_q^{k \times m}} \{\Delta(\mathcal{C}(\mathbf{G})) \geq \delta m\} \geq 1 - q^{-\epsilon m + 1}.$$

Definition 1 (Gilbert-Varshamov Distance of Linear Code)**.** *For a linear code $\mathcal{C}$ with parity-check matrix $\mathbf{H} \in \mathbb{F}^{n \times m}$, the* Gilbert-Varshamov distance *of $\mathcal{C}$ is the smallest integer d such that $\binom{m}{d} \geq 2^n$.*

2.2 LPN and Syndrome Decoding

Fix a field $\mathbb{F}$, let $m, k > 0$ be integers, and let $0 \leq \tau \leq 1$. We use $\mathsf{Ber}_p(\mathbb{F})$ to denote the Bernoulli distribution over $\mathbb{F}$ that returns a uniform non-zero element with probability p and 0 with probability $1 - p$. We use $\mathsf{HW}_w^n(\mathbb{F})$ to denote the exact error distribution of Hamming weight w over $\mathbb{F}^n$, that is, $\mathsf{HW}_w^n(\mathbb{F})$ is the uniform distribution over the subset $\{\mathbf{x} \in \mathbb{F}^n \mid \Delta(\mathbf{x}) = w\}$. The *primal* (m, k, τ)-LPN distribution over $\mathbb{F}$, parameterized by secret $\mathbf{s} \in \mathbb{F}^k$ is

$$\{(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T) \mid \mathbf{A} \leftarrow\$\, \mathbb{F}^{m \times k}, \mathbf{e} \leftarrow \mathsf{Ber}_\tau^m\}.$$

We usually refer to k as the secret dimension, m the number of samples, τ the error rate, and $w = m\tau$ the expectation of Hamming weight of the error vector $\mathbf{e}$.

Definition 2 (Primal LPN Assumption [BFKL94])**.** *Let $m, k > 0$ and $0 \leq \tau \leq 1$. The* (primal) (m, k, τ)-LPN assumption *states that, for any secret vector $\mathbf{s} \leftarrow\$\, \mathbb{F}^k$, the (m, k, τ)-LPN distribution is indistinguishable from uniform distribution over the same output domain:*

$$\{(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T) \mid \mathbf{A} \leftarrow\$\, \mathbb{F}^{m \times k}, \mathbf{e} \leftarrow \mathsf{Ber}_\tau^m\} \approx_c \{(\mathbf{A}, \mathbf{b}^T) \mid \mathbf{A} \leftarrow\$\, \mathbb{F}^{m \times k}, \mathbf{b} \leftarrow\$\, \mathbb{F}^m\}.$$

In the above formulation, we can consider $\mathbf{A}$ as the generator matrix of a (m, k)-linear code $\mathcal{C}$ over $\mathbb{F}$. The vector $\mathbf{b}^T = \mathbf{s}^T\mathbf{A} + \mathbf{e}^T$ is thus a received word with error $\mathbf{e}^T$. Let $n = m - k$ and $\mathbf{H} \in \mathbb{F}^{n \times m}$ be the parity check matrix of $\mathbf{A}$, *i.e.*, $\mathbf{A}\mathbf{H}^T = 0$, we then have $\mathbf{H}\mathbf{b} = \mathbf{H}(\mathbf{A}^T\mathbf{s} + \mathbf{e}) = \mathbf{H} \cdot \mathbf{e}$. The vector $\mathbf{H}\mathbf{e}$ is called the *syndrome* of the received word $\mathbf{b}$. The *dual* form of the LPN assumption is now on the distribution on $\mathbf{H}\mathbf{e}$ being pseudorandom for $\mathbf{H} \leftarrow\$\, \mathbb{F}^{(m-k) \times m}$.

The dual LPN assumption is closely related to the *Syndrome Decoding* problem, and in fact it is equivalent to the $(m, m - k, w)$-*Decisional Syndrome Decoding* assumption.

Definition 3 (Decisional Syndrome Decoding (dual-LPN) Assumption [BFKL94, AIK07])**.** *Let $n, m > 0$ and $0 \leq \tau \leq 1$. The* Decisional (m, n, τ)-Syndrome Decoding assumption *states that the following two distributions are computational indistinguishable:*

$$\begin{aligned}\mathsf{SD}_{m,n,\tau}(\mathbb{F}) &= \{(\mathbf{H}, \mathbf{H} \cdot \mathbf{e}) \mid \mathbf{H} \leftarrow\$\, \mathbb{F}^{n \times m}, \mathbf{e} \leftarrow \mathsf{Ber}_\tau^m\} \\ &\approx_c \{(\mathbf{H}, \mathbf{y}) \mid \mathbf{H} \leftarrow\$\, \mathbb{F}^{n \times m}, \mathbf{y} \leftarrow\$\, \mathbb{F}^n\}.\end{aligned}$$

For our purpose, it is more convenient to use the exact error distribution $\mathsf{HW}_w^n(\mathbb{F})$ for fixed Hamming weight w instead of the Bernoulli distribution. The corresponding syndrome decoding problem with a fixed Hamming weight error distribution is usually referred to as the *Exact*-Syndrome Decoding, and we use $\mathsf{SD}_{n,m,w}$ to denote this variant. Note that the hardness of search version of $\mathsf{SD}_{n,m,w}$, which asks the adversary to recover the error vector $\mathbf{e}$, follows directly from the hardness of the search $\mathsf{SD}_{n,m,\tau}$ problem [Pie12]. See [LWYY22] for a recent analysis on the relations between the exact SD / LPN problems and the Bernoulli variant.

2.3 Additive Secret Sharing

We use an additive secret sharing scheme (Share, Recon) over a vector space. Formally, let $\mathbb{F}$ be a finite field, and let $n > 0$. The additive secret sharing scheme over $\mathbb{F}^n$ is defined as follows.

- $\mathsf{Share}(\mathbf{x} \in \mathbb{F}^n; k \geq 2) = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$: On input a secret $\mathbf{x}$, sample uniform and i.i.d. $\mathbf{s}_1, \ldots, \mathbf{s}_{k-1} \leftarrow_{\$} \mathbb{F}^n$, and let $\mathbf{s}_k = \mathbf{x} - \sum_{i=1}^{k-1} \mathbf{s}_i$.
- $\mathsf{Recon}(\mathbf{s}_1, \ldots, \mathbf{s}_k) = \sum_{i=1}^{k} \mathbf{s}_i$: On input shares $\mathbf{s}_1, \ldots, \mathbf{s}_k$, output their sum as the reconstructed secret.

Additive secret sharing is information-theoretically secure in the sense that any subset of shares of size at most $k-1$ is uniformly random. We formulate an interesting computational problem about mixing many additive secret sharings in Section 3. Looking ahead, this will be central in the analysis of our constructions.

2.4 Computational Bit Security

We restate the bit-security framework [MW18] for cryptographic primitives.

Definition 4 (Non-adaptive Decision Game)**.** *Let $\{\mathcal{D}_\theta^0\}_\theta$ and $\{\mathcal{D}_\theta^1\}_\theta$ be two distribution ensembles indexed by a security parameter θ. A non-adaptive decision game $\mathcal{G}$ (implicitly parameterized by θ) is defined as follows:*

- *$\mathcal{C}$ samples a bit $b \leftarrow_{\$} \{0, 1\}$;*
- *$\mathcal{A}$ sends $q \in \mathbb{N}$ to $\mathcal{C}$;*
- *$\mathcal{C}$ samples $y_1, \ldots, y_q \leftarrow \mathcal{D}_\theta^b$ and sends $(y_1, \ldots, y_q)$ to $\mathcal{A}$;*
- *$\mathcal{A}$ outputs either a bit b' or $\perp$.*

The output of the game is defined as $\mathcal{G}[\mathcal{A}] := (b' = b)$.

Definition 5 (Bit Security)**.** *For any adversary $\mathcal{A}$ playing a decision game $\mathcal{G}$ and outputting b', we define its output probability as $\alpha^{\mathcal{A}} := \Pr\{b' \neq \perp\}$ and its conditional success probability as $\beta^{\mathcal{A}} := \Pr\{b' = b \mid b' \neq \perp\}$, where the probabilities are taken over the randomness of the game $\mathcal{G}$ and internal randomness of $\mathcal{A}$. We define $\mathcal{A}$'s conditional distinguishing advantage as $\delta^{\mathcal{A}} := 2\beta^{\mathcal{A}} - 1$, and the advantage of $\mathcal{A}$ as $\mathsf{Adv}^{\mathcal{A}} := \alpha^{\mathcal{A}}(\delta^{\mathcal{A}})^2$.*

The bit security of a primitive modeled by $\mathcal{G}$ is $\min_{\mathcal{A}} \log \frac{T(\mathcal{A})}{\mathsf{Adv}^{\mathcal{A}}}$, where $T(\mathcal{A})$ is the running time of $\mathcal{A}$.

Bit security provides a uniform measure about hardness of a computational problem. The following lemma allows us to determine the bit security of hybrid arguments.

Lemma 2 (Hybrid Argument [MW18])**.** *Let $\{\mathcal{H}_i\}_{i=1}^k$ be a series of k distributions and $\mathcal{G}_{i,j}$ be the decision game instantiated with $\mathcal{H}_i$ and $\mathcal{H}_j$. Let $\varepsilon_{i,j} = \max_{\mathcal{A}} \mathsf{Adv}^{\mathcal{A}}$ over all T-bounded adversaries $\mathcal{A}$ against $\mathcal{G}_{i,j}$. Then $\varepsilon_{1,k} \leq 3k \sum_{i=1}^{k-1} \varepsilon_{i,i+1}$.*

2.5 Key-Message Homomorphic Encryption

For our proxy and secure aggregation constructions, we use a symmetric encryption scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$ and message space $\mathbb{F}^\ell$, such that it is both key and message homomorphic, i.e. $\mathsf{Enc}(s_1, \mathbf{x}_1) + \mathsf{Enc}(s_2, \mathbf{x}_2) = \mathsf{Enc}(s_1 \oplus s_2, \mathbf{x}_1 + \mathbf{x}_2)$, where $\oplus$ and $+$ are addition over $\mathcal{K}$ and $\mathbb{F}^\ell$, respectively. Furthermore, let $\mathcal{E}' = (\mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ be a public key additively homomorphic encryption scheme with message space $\mathcal{K}$.

We can instantiate $\mathcal{E}$ and $\mathcal{E}'$ with the above properties using lattice-based encryption schemes [Reg06]. One natural instantiation is the following: Assume $\mathbb{F} = \mathbb{F}_p$ is a finite field of prime order. Let $R_{Q_1} = \mathbb{Z}_{Q_1}/(X^{N_1} + 1)$ be a quotient ring with $N_1 \geq \ell$ a power of two and $Q_1 > p$, and let $R_{Q_2} = \mathbb{Z}_{Q_2}/(X^{N_2} + 1)$ be a quotient ring such that $N_2 \geq N_1$ is a power of two and $Q_2 > Q_1$. For some $\sigma > 0$, let $\chi_\sigma(R)$ be the distribution over polynomial ring R where each coefficient is independent Gaussian with parameter σ. The key space of $\mathcal{E}$ is $\mathcal{K} = R_{Q_1}$, and we interpret any vector $\mathbf{x} \in \mathbb{F}_p^\ell$ as the coefficients of a polynomial in $\mathbb{Z}_p/(X^{N_1} + 1)$. The symmetric encryption scheme $\mathcal{E}$ with public random polynomial $a \leftarrow_{\$} R_{Q_1}$ consists of the following algorithms:

- $\mathsf{Enc}(s, \mathbf{x}) \ {}_{\$}\!\!\rightarrow a \cdot s + e + \lfloor Q_1/p \cdot \mathbf{x} \rceil$: where $e \leftarrow \chi_\sigma$;

- $\mathsf{Dec}(s, y) = \lfloor (y - a \cdot s)/p \rceil$.

The public key encryption scheme $\mathcal{E}'$ can be defined as:

- $\mathsf{KeyGen}'() \overset{\$}{\to} (\mathsf{sk}, \mathsf{pk} = (u, -u \cdot \mathsf{sk} + e') \pmod{Q_2})$, where $u \leftarrow_\$ R_{Q_2}$ and $\mathsf{sk}, e' \leftarrow \chi_\sigma$;
- $\mathsf{Enc}'(\mathsf{pk}, s) \overset{\$}{\to} (v \cdot \mathsf{pk}_0 + e_0 + \lfloor Q_2/Q_1 \cdot s \rceil, v \cdot \mathsf{pk}_1 + e_1)$, where $v, e_0, e_1 \leftarrow \chi_\sigma$.
- $\mathsf{Dec}'(\mathsf{sk}, \mathsf{ct}) = \lfloor (\mathsf{ct}_0 \cdot \mathsf{sk} + \mathsf{ct}_1)/Q_1 \rceil$.

To securely instantiate $\mathcal{E}$ and $\mathcal{E}'$, we can set lattice parameters (N_1, Q_1, σ) and (N_2, Q_2, σ) with the desired security level.

To combine a KMAHE scheme with a shuffle-model secure aggregation protocol as in Construction 2, it must satisfy a special leakage-resilient security. We now formally define such security.

Definition 6. *Let $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ be an KMAHE encryption scheme with key space $\mathcal{K}$. Let $c > 0$ be any positive integer. We say that $\mathcal{E}$ is* c-IND-secure under leakage of sum of secret keys *if any efficient adversary $\mathcal{A}$ has negligible advantage in winning the following experiment.*

$$\begin{aligned}
&\mathsf{Expr}[\mathcal{A}](1^\kappa) = (b == b') :\\
&\qquad b \leftarrow_\$ \{0, 1\}\\
&\qquad (\{x_i^{(0)}\}_{i=1}^c, \{x_i^{(1)}\}_{i=1}^c) \leftarrow \mathcal{A}(1^\kappa)\\
&\qquad \textit{for all } i = 1, \dots, c : \mathsf{sk}_i \leftarrow_\$ \mathcal{K}, \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, x_i^{(b)})\\
&\qquad b' \leftarrow \mathcal{A}(\sum_{i=1}^c \mathsf{sk}_i, \mathsf{ct}_1, \dots, \mathsf{ct}_c)
\end{aligned}$$

3 Split-and-Mix Core Lemma

In this section, we formulate a core *split-and-mix* lemma which we will use in the rest of our constructions; this formalizes the security properties we need from shuffling additive shares. To formalize the shuffle model, let $\Pi = \{\Pi_m\}_{m \in \mathbb{N}}$ be a distribution ensemble where Π_m is a distribution over the symmetric group $\mathfrak{S}_m$ (recall that this is the group of all permutations of $\{1, \dots, m\}$). When Π is unspecified, we assume that each Π_m is a uniform distribution over $\mathfrak{S}_m$; we refer to this as the uniform or perfect shuffler.

We begin by considering the problem of distinguishing between (i) a single additive sharing of $\sum_{i=0}^c \mathbf{x}_i$ into m shares; and (ii) the union of individual additive sharings of $\mathbf{x}_i$ into a smaller (k or k_0) number of shares, with $c \cdot k + k_0 = m$. More formally, let $\mathbb{F}$ be a finite field, let c, n, k, k_0 be positive integers, and let $m = c \cdot k + k_0$. For any $\mathbf{x}_0, \dots, \mathbf{x}_c \in \mathbb{F}^n$, the *split-and-mix* problem is to distinguish the following two distributions:

$$\mathcal{D}_0 = \left\{ \pi(\mathbf{s}_1^{(0)}, \dots, \mathbf{s}_{k_0}^{(0)}, \mathbf{s}_1^{(1)}, \dots, \mathbf{s}_k^{(1)}, \dots, \mathbf{s}_1^{(c)}, \dots, \mathbf{s}_k^{(c)}) \;\middle|\; \begin{array}{r} (\mathbf{s}_1^{(0)}, \dots, \mathbf{s}_{k_0}^{(0)}) \leftarrow \mathsf{Share}(\mathbf{x}_0 \in \mathbb{F}^n, k_0), \\ \forall 1 \le i \le c.\ (\mathbf{s}_1^{(i)}, \dots, \mathbf{s}_k^{(i)}) \leftarrow \mathsf{Share}(\mathbf{x}_i \in \mathbb{F}^n, k), \\ \pi \leftarrow_\$ \Pi_m \end{array} \right\},$$

and

$$\mathcal{D}_1 = \left\{ \pi(\mathbf{s}_1, \dots, \mathbf{s}_m) \;\middle|\; \begin{array}{r} (\mathbf{s}_1, \dots, \mathbf{s}_m) \leftarrow \mathsf{Share}(\sum_{i=0}^c \mathbf{x}_i \in \mathbb{F}^n, m), \\ \pi \leftarrow_\$ \Pi_m \end{array} \right\}.$$

We formally present the indistinguishability result in the split-and-mix problem in Lemma 4, which is our formal *split-and-mix* lemma.

Here we briefly discuss the split-and-mix problem. Without loss of generality, we assume that the first $k-1$ shares in $\mathsf{Share}(\mathbf{x}, k)$ are independently sampled, *e.g.*, $\mathbf{s}_1^{(0)}, \dots, \mathbf{s}_{k_0-1}^{(0)} \leftarrow_\$ \mathbb{F}^n$ and $\mathbf{s}_{k_0}^{(0)} = \mathbf{x}_0 - \sum_{i=1}^{k_0-1} \mathbf{s}_i^{(0)}$. Looking ahead, $\mathbf{x}_0$ may play a different role than the rest of the input vectors and we may set $k_0 \gg k$:

In the aggregation protocol where a proxy implements the shuffle, $\mathbf{x}_0$ is the $\mathbf{0}$ vector, and we may let the proxy insert a large k_0 shares of $\mathbf{0}$ into the mix. In PIR, $\mathbf{x}_0$ is a random vector where the generation of the large shares $\mathbf{s}_1^{(0)}, \ldots, \mathbf{s}_{k_0-1}^{(0)}$ is distributed among the clients, to make it hard for the server to locate shares of real input $\mathbf{x}_i$ for $i = 1, \ldots, c$. In both cases, we would like to set k as small as possible while keeping the split-and-mix problem hard.

When $c = 1$, the split-and-mix problem reduces from Syndrome Decoding $\mathsf{SD}_{n,m-2,k-1}$ over $\mathbb{F}$. We formally present the reduction in the lemma below.

Lemma 3. *For any integers $n > 0$, $m > 2$, and $k > 1$, and for any $\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{F}^n$, if the decisional $\mathsf{SD}_{n,m-2,k-1}$ is hard, then the split-and-mix problem over $\mathbf{x}_0, \mathbf{x}_1$ with $c = 1$ is also hard.*

Proof. Assume $\mathcal{A}$ solves the split-and-mix problem with $c = 1$. We build the following reduction to solve $\mathsf{SD}_{n,m-2,k-1}$ using $\mathcal{A}$ as an oracle. Given a uniformly random parity check matrix $\mathbf{H} \leftarrow \mathbb{F}^{n\times(m-2)}$ and a vector $\mathbf{y} \in \mathbb{F}^n$, the reduction computes $\mathbf{z} = \mathbf{x}_1 - \sum_j \mathbf{h}_j + \mathbf{y}$ and runs $\mathcal{A}$ on a set of randomly permuted vectors $\pi(\mathbf{h}_1, \ldots, \mathbf{h}_{m-2}, -\mathbf{y} + \mathbf{x}_0, \mathbf{z})$. Clearly, if $(\mathbf{H}, \mathbf{y})$ is a $\mathsf{SD}_{n,m-2,k-1}$ sample and hence $\mathbf{y} = \mathbf{H} \cdot \mathbf{e}$ for some random $\mathbf{e}$ with Hamming weight $k - 1$, then the input to $\mathcal{A}$ follows $\mathcal{D}_0$; if $(\mathbf{H}, \mathbf{y})$ is instead a uniformly random sample, then the input to $\mathcal{A}$ is an additive sharing of $\mathbf{x}_0 + \mathbf{x}_1$ of size m and follows $\mathcal{D}_1$. $\square$

We are interested in the case where $c > 1$ and $k_0 \geq k$. The conjectured hardness of this decision problem will serve as the basis for applications in the shuffle model, including secure aggregation and PIR. Later in this section, we will show that this hardness conjecture is implied by the hardness of a generalized version of syndrome decoding.

3.1 Multi-Disjoint Syndrome Decoding

As alluded to above, there are close connections between the split-and-mix problem and the problem of decoding random linear codes, and in the degenerate case $c = 1$, there is a simple reduction from syndrome decoding. For the general case, we introduce the *Multi-Disjoint Syndrome Decoding* problem and study its hardness and relations with existing problems.

First, let us define an error distribution for this new problem.

Definition 7 (Disjoint Error)**.** *Let $\mathbb{F}$ be a finite field, and let $m, c, w > 0$. The (m, c, w)-*Disjoint Error Set *$\mathsf{DisError}_{m,c,w}$ is the following set of matrices over $\mathbb{F}$:*

$$\mathsf{DisError}_{m,c,w} = \{\mathbf{E} \in \mathbb{F}^{m\times c} \mid \forall i \neq j, \mathbf{e}_i \cap \mathbf{e}_j = \emptyset \wedge \forall i, \Delta(\mathbf{e}_i) = w\}.$$

Now we define the multi-disjoint syndrome decoding problem.

Definition 8 (Multi-Disjoint Syndrome Decoding)**.** *Let $\mathbb{F}$ be a finite field, and let $n, m, c, w > 0$. The* (n, m, c, w)-Multi-Disjoint Syndrome Decoding *distribution over $\mathbb{F}$ is*

$$\mathsf{MDSD}_{n,m,c,w} = \{(\mathbf{H}, \mathbf{H} \cdot \mathbf{E}) \mid \mathbf{H} \leftarrow_{\$} \mathbb{F}^{n\times m}, \mathbf{E} \leftarrow_{\$} \mathsf{DisError}_{m,c,w}\}.$$

The (n, m, c, w)-Decisional Multi-Disjoint Syndrome Decoding *(MDSD) problem is to distinguish the distribution $\mathsf{MDSD}_{n,m,c,w}$ from the uniform distribution over $\mathbb{F}^{n\times m} \times \mathbb{F}^{n\times c}$.*

The next lemma states that the hardness of the MDSD problem implies the indistinguishability of the split-and-mix distributions $\mathcal{D}_0$ and $\mathcal{D}_1$. To see the link first we will consider shares of the zero vector. Let $w = k - 1$, $w_0 = k_0 - 1$, and let $m = cw + w_0$. Let $\mathbf{H}$ be a $n \times m$ matrix whose columns are the shuffled random vectors $\{\mathbf{s}_j^{(0)}\}_{j=1}^{w_0}$, $\{\mathbf{s}_j^{(1)}\}_{j=1}^{w}$, $\ldots$, $\{\mathbf{s}_j^{(c)}\}_{j=1}^{w}$. Let $\mathbf{E}$ be a 0-1 matrix of dimension $m \times c$ such that, for $1 \leq i \leq c$, the coordinates of 1's in the i'th column $\mathbf{e}_i$ are exactly the column indices of $\mathbf{s}_1^{(i)}, \ldots, \mathbf{s}_w^{(i)}$ in $\mathbf{H}$. That is, if $\mathbf{Y} = \mathbf{H} \cdot \mathbf{E}$ is a matrix with c columns $\mathbf{y}_1, \ldots, \mathbf{y}_c$, then we have $\mathbf{y}_i = \sum_{j=1}^{w} \mathbf{s}_j^{(i)}$. It follows that, for $1 \leq i \leq c$, $(\mathbf{s}_1^{(i)}, \ldots, \mathbf{s}_w^{(i)}, -\mathbf{y}_i)$ is an additive secret sharing of $\mathbf{0}$ of size k. The columns in $\mathbf{H}$ that are not selected by $\mathbf{E}$ are exactly $\mathbf{s}_1^{(0)}, \ldots, \mathbf{s}_{w_0}^{(0)}$. So, we set $z = \mathbf{x}_0 - \mathbf{s}_{k_0}^{(0)} = \mathbf{x}_0 - (\sum_{j=1}^{m} \mathbf{h}_j - \sum_{i=1}^{c} \mathbf{y}_i)$, and thus $(\mathbf{s}_1^{(0)}, \ldots, \mathbf{s}_{k_0-1}^{(0)}, \mathbf{z})$ form an additive secret sharing of $\mathbf{x}_0$ of size k_0. At this point, we can think of $\mathbf{H}, \mathbf{X} - \mathbf{H} \cdot \mathbf{E}, z$, where $\mathbf{X}$ has columns $\mathbf{x}_1, \ldots, \mathbf{x}_c$, as the *shuffled* shares of the vectors $\mathbf{x}_0, \ldots, \mathbf{x}_c$, which is distribution $\mathcal{D}_0$. On the other hand, if we have a random matrix $\mathbf{Y}$ with c columns $\mathbf{y}_1, \ldots, \mathbf{y}_c$ and we set $\mathbf{z} = \mathbf{x}_0 - (\sum_{j=1}^m \mathbf{h}_j - \sum_{i=1}^c \mathbf{y}_i)$ where $\mathbf{h}_j$ are the columns of $\mathbf{H}$, then $\mathbf{H}, \mathbf{X} - \mathbf{Y}, \mathbf{z}$ are shuffled shares of $\sum_{i=0}^c \mathbf{x}_i$, which is the distribution $\mathcal{D}_1$.

Lemma 4 (Split-and-Mix)**.** *For any $n, m, c, w > 0$, and for any set of vectors $\mathbf{x}_0, \ldots, \mathbf{x}_c \in \mathbb{F}^n$, where $\mathbf{X}$ has as columns $\mathbf{x}_1, \ldots, \mathbf{x}_c$, if the decisional $\mathsf{MDSD}_{n,m,c,w}$ is hard, then the following two distributions are computationally indistinguishable.*

$$D_0 = \left\{ (\mathbf{H}, \mathbf{X} - \mathbf{Y}, \mathbf{z} = \mathbf{x}_0 - \sum_j \mathbf{h}_j + \sum_i \mathbf{y}_i) \;\middle|\; \begin{array}{l} \mathbf{H} \leftarrow\!\!\$\; \mathbb{F}^{n \times m}, \\ \mathbf{E} \leftarrow\!\!\$\; \mathsf{DisError}_{m,c,w} \\ \mathbf{Y} = \mathbf{H} \cdot \mathbf{E} \end{array} \right\},$$

$$D_1 = \left\{ (\mathbf{H}, \mathbf{X} - \mathbf{Y}, \mathbf{z} = \mathbf{x}_0 - \sum_j \mathbf{h}_j + \sum_i \mathbf{y}_i) \;\middle|\; \begin{array}{l} \mathbf{H} \leftarrow\!\!\$\; \mathbb{F}^{n \times m}, \\ \mathbf{Y} \leftarrow\!\!\$\; \mathbb{F}^{n \times c} \end{array} \right\}.$$

Proof. Notice that all elements in the two distributions can be computed given $\mathbf{H}$ and $\mathbf{Y}$ in additions to $\mathbf{x}_0, \ldots, \mathbf{x}_c$. So a reduction from $\mathsf{MDSD}_{n,m,c,w}$ is straightforward. □

The distributions D_0 and D_1 in Lemma 4 contain smaller shuffle than in $\mathcal{D}_0$ and $\mathcal{D}_1$: in D_0 the matrix $\mathbf{H}$ is a shuffle of the first w shares of each $\mathbf{x}_i$, for $1 \le i \le c$, as well as the first w_0 shares of $\mathbf{x}_0$, where the permutation is determined by the columns of the error matrix $\mathbf{E}$; in D_1 the matrix $\mathbf{H}$ is a shuffle of the first $m-1$ share vectors of $\sum_{i=0}^c \mathbf{x}_i$. In particular, if D_0 and D_1 are indistinguishable, then so is $\mathcal{D}_0$ and $\mathcal{D}_1$; this is because any random permutation π in $\mathcal{D}_0$ and $\mathcal{D}_1$ can be decomposed into the permutation implied by the columns of $\mathbf{E}$ and another permutation in $\mathfrak{S}_m$. Looking ahead, this limited form of shuffle in D_0 and D_1 allows to use PRG in applications to compress all but one shares for each $\mathbf{x}_i$.

3.2 Reduction from Syndrome Decoding to MDSD

In this section we show a reduction from the standard syndrome decoding problem to our new multi-disjoint version. We start with the following technical lemma that establishes a statistical bound.

Lemma 5. *Let $n, m, w > 0$, and let $\mathbb{F}_q$ be a finite field. If $n \ge h_q(\frac{2w}{m}) \cdot m + \frac{\sigma}{\log q}$, then*

$$\Pr_{\mathbf{H} \leftarrow\$\, \mathbb{F}_q^{n\times m}} \left\{ \exists \mathbf{e}, \mathbf{e}' \in \mathbb{F}_q^m, \mathbf{e} \neq \mathbf{e}', \mathbf{He} = \mathbf{He}', \Delta(\mathbf{e}) = \Delta(\mathbf{e}') = w \right\} \le 2^{-\sigma}.$$

Proof. Assume there exist distinct $\mathbf{e}, \mathbf{e}' \in \mathbb{F}_q^m$ such that $\mathbf{He} = \mathbf{He}'$ and $\Delta(\mathbf{e}) = \Delta(\mathbf{e}') = w$. Let $\mathbf{y} = \mathbf{e} - \mathbf{e}'$. Since $\mathbf{H} \cdot \mathbf{y} = \mathbf{0}$ and $\mathbf{y} \neq \mathbf{0}$, we have that $\mathbf{y}$ is a non-zero codeword of $\mathcal{C}(\mathbf{H}^\perp)$ and that $\Delta(\mathcal{C}(\mathbf{H}^\perp)) \le \Delta(\mathbf{y}) \le 2w$. Let $\mathbf{G} = \mathbf{H}^\perp$ be the generator matrix of the code. By Lemma 1, the probability that $\mathcal{C}(\mathbf{G})$ has distance at most $2w$ is bounded by

$$\Pr_{\mathbf{G} \leftarrow\$\, \mathbb{F}_q^{k\times m}} \{\Delta(\mathcal{C}(\mathbf{G})) \le 2w\} \le q^{-\epsilon m + 1},$$

where $k = m - n$ and $\epsilon m \le (1 - h_q(\frac{2w}{m})) \cdot m + 1 - k$. By our condition on n, we can bound the above probability by $2^{-\sigma}$. □

We are now ready to present our reduction from SD to MDSD.

Lemma 6. *Fix a finite field $\mathbb{F}_q$, and let $n, m, c, w > 0$ such that $m = c \cdot w + \lambda$ for some $\lambda > 0$. Assume for all $1 \le i \le c$, $\mathsf{SD}_{n, i\cdot w+\lambda, w}$ is κ_i-bit computationally hard and $n \ge h_q(\frac{2w}{m}) \cdot m + \frac{\kappa_i}{\log q}$. Then $\mathsf{MDSD}_{n,m,c,w}$ is κ-bit computationally hard for $\kappa = -\log(3c) - \log(\sum_{i=1}^c 2^{-\kappa_i})$.*

Proof. We proceed with a hybrid argument reducing the $\mathsf{MDSD}_{n,m,c,w}$ problem from the standard $\mathsf{SD}_{n,wh+\lambda,w}$ problem with different codeword lengths $wh+\lambda$ for $h = 1, \ldots, c$. In the MDSD problem, the adversary wants to distinguish between $(\mathbf{H}, \mathbf{H}\cdot\mathbf{E})$ and a uniformly random instance $(\mathbf{H}, \mathbf{Y})$, where $\mathbf{E} \leftarrow_{\$} \mathsf{DisError}_{m,c,w} \subset \mathbb{F}_q^{m\times c}$ consists of "error vectors" that encode the locations of linearly dependent columns of the matrix $(\mathbf{H}, \mathbf{H}\cdot\mathbf{E})$.

Our hybrids are defined as follows:

- $\mathsf{Hyb}^{(0)} = (\mathbf{H}, \mathbf{y}_1, \ldots, \mathbf{y}_c)$ where $\mathbf{H} \leftarrow_{\$} \mathbb{F}_q^{n\times m}$ and $\mathbf{y}_1, \ldots, \mathbf{y}_c \leftarrow_{\$} \mathbb{F}_q^n$: This is a uniformly random instance.

- $\mathsf{Hyb}^{(i)} = (\mathbf{H}, \mathbf{He}_1, \ldots, \mathbf{He}_i, \mathbf{y}_{i+1}, \ldots, \mathbf{y}_c)$, where $\mathbf{H} \leftarrow_{\$} \mathbb{F}_q^{n\times m}$, $\mathbf{E}^{(i)} \leftarrow_{\$} \mathsf{DisError}_{m,i,w}$, and $\mathbf{y}_{i+1}, \ldots, \mathbf{y}_c \leftarrow_{\$} \mathbb{F}_q^n$ are independently and uniformly sampled. Note that $\mathsf{Hyb}^{(c)}$ is exactly the MDSD distribution.

In the following we denote using $\mathbf{1}^w$ for a vector in $\mathbb{F}_q^w$ whose components are all 1's, and $\mathbf{0}^w$ for a vector of w many 0's. Row vectors are denoted using notations like $(v_1, \ldots, v_m)$, and by default we assume vectors are column vectors.

- $\mathsf{Hyb}^{(0)}$ vs $\mathsf{Hyb}^{(1)}$: Note that $\mathsf{Hyb}^{(1)} = (\mathbf{H} \leftarrow_{\$} \mathbb{F}_q^{n\times m}, \mathbf{y}_1 = \mathbf{He}_1, \mathbf{y}_2 \leftarrow_{\$} \mathbb{F}^n, \ldots, \mathbf{y}_c \leftarrow_{\$} \mathbb{F}^n)$, where $\mathbf{e}_1$ is sampled from $\mathsf{DisError}_{m,1,w}$. Distinguishing $\mathsf{Hyb}^{(0)}$ and $\mathsf{Hyb}^{(1)}$ is exactly the decisional $\mathsf{SD}_{n,m,w}$ problem.

- For any $i \geq 1$, we can reduce $\mathsf{SD}(n, (c-i)w + \lambda, w)$ to distinguishing $\mathsf{Hyb}^{(i)}$ and $\mathsf{Hyb}^{(i+1)}$, and in addition a statistical argument. We have

$$\begin{aligned}\mathsf{Hyb}^{(i)} &= (\mathbf{H} \leftarrow_{\$} \mathbb{F}_q^{n\times m}, \mathbf{y}_1 = \mathbf{He}_1, \ldots, \mathbf{y}_i = \mathbf{He}_i, \mathbf{y}_{i+1} \leftarrow_{\$} \mathbb{F}^n, \mathbf{y}_{i+2} \leftarrow_{\$} \mathbb{F}^n \ldots, \mathbf{y}_c \leftarrow_{\$} \mathbb{F}^n),\\ \mathsf{Hyb}^{i+1} &= (\mathbf{H} \leftarrow_{\$} \mathbb{F}^{n\times m}, \mathbf{y}_1 = \mathbf{He}_1, \ldots, \mathbf{y}_i = \mathbf{He}_i, \mathbf{y}_{i+1} = \mathbf{He}_{i+1}, \mathbf{y}_{i+2} \leftarrow_{\$} \mathbb{F}^n, \ldots, \mathbf{y}_c \leftarrow_{\$} \mathbb{F}^n).\end{aligned}$$

The difference between these two hybrids is in how $\mathbf{y}_{i+1}$ is generated.

Assume $\mathcal{A}$ is any efficient distinguisher of $\mathsf{Hyb}^{(i)}$ and $\mathsf{Hyb}^{(i+1)}$. We build a reduction $\mathcal{B}(\bar{\mathbf{H}}, \bar{\mathbf{y}})$ from $\mathsf{SD}(n, (c-i)w + \lambda, w)$, where the input to $\mathcal{B}$ is either a sample $(\bar{\mathbf{H}} \leftarrow_{\$} \mathbb{F}^{n\times((c-i)w+\lambda)}, \bar{\mathbf{y}} = \bar{\mathbf{H}}\bar{\mathbf{e}})$ from the syndrome decoding distribution $\mathsf{SD}_{n,(c-i)w+\lambda,w}$ for $\bar{\mathbf{e}} \leftarrow \mathsf{Ber}_{w/(w+\lambda)}^{m+\lambda}$, or it is a uniformly random sample $(\bar{\mathbf{H}} \leftarrow_{\$} \mathbb{F}_q^{n\times((c-i)w+\lambda)}, \bar{\mathbf{y}} \leftarrow_{\$} \mathbb{F}_q^n)$.

```
$\mathcal{B}(\bar{\mathbf{H}}, \bar{\mathbf{y}})$ :  Sample $\mathbf{H}^* \leftarrow_{\$} \mathbb{F}^{n\times iw}$, and $\pi \leftarrow_{\$} \Pi_{cw+\lambda}$;
          For $j \in \{1, \ldots, i\}$:
            let $\mathbf{y}_j = \mathbf{H}^* \cdot \mathbf{f}_j$ for $\mathbf{f}_j^T = (\mathbf{0}^{(j-1)w}, \mathbf{1}^w, \mathbf{0}^{(i-j)w}) \in \mathbb{F}^{iw}$;
          Sample $\mathbf{y}_{i+2}, \ldots, \mathbf{y}_c \leftarrow_{\$} \mathbb{F}_q^n$;
          Let $\mathbf{H} = \pi(\bar{\mathbf{H}} \mid \mathbf{H}^*)$, where we apply $\pi$ to randomly permute columns of $(\bar{\mathbf{H}} \mid \mathbf{H}^*)$;
          Run $\mathcal{A}(\mathbf{H}, \mathbf{y}_1, \ldots, \mathbf{y}_i, \bar{\mathbf{y}}, \mathbf{y}_{i+2}, \ldots, \mathbf{y}_c)$, and return its result.
```

Fix an input $(\bar{\mathbf{H}}, \bar{\mathbf{y}})$. We consider two cases:

1. $\bar{\mathbf{y}} = \bar{\mathbf{H}}\bar{\mathbf{e}}$ is generated as a syndrome of $\bar{\mathbf{e}} \in \mathbb{F}_q^{iw+\lambda}$ such that $\Delta(\mathbf{e}) = w$. We consider two sub-cases:

   – There exists a unique solution $\mathbf{e} \in \mathbb{F}_q^m$ such that $\mathbf{He} = \bar{\mathbf{y}}$. By the construction of $\mathbf{H} = \pi(\bar{\mathbf{H}}|\mathbf{H}^*)$ and the assumption that $\bar{\mathbf{H}}\bar{\mathbf{e}} = \bar{\mathbf{y}}$, we must have $\mathbf{e}^T = \pi((\bar{\mathbf{e}}, \mathbf{0}^{(i-1)w})^T)$. By definition, we also have $\mathbf{y}_j = \mathbf{He}_j$ for $\mathbf{e}_j^T = \pi(\mathbf{0}^{(c-i)w+\lambda}, \mathbf{f}_j^T)$, $j = 1, \ldots, i$. For any distinct $j, j' < i$, since $\mathbf{f}_j \cap \mathbf{f}_{j'} = \mathbf{0}$ by definition, we have $\mathbf{e}_j \cap \mathbf{e}_{j'} = \mathbf{0}$ are also disjoint. Furthermore, $\mathbf{e} \cap \mathbf{e}_j = \mathbf{0}$ are disjoint for all $j \leq i$. So, $\mathbf{E} = (\mathbf{e}_1, \ldots, \mathbf{e}_i, \mathbf{e})$ is an element of $\mathsf{DisError}_{m,i+1,w}$, and hence the input given to $\mathcal{A}$ is exactly $\mathsf{Hyb}^{(i+1)}$.

   – There exist distinct $\mathbf{e}' \neq \mathbf{e}$ such that $\mathbf{He}' = \mathbf{He} = \bar{\mathbf{y}}$ and $\Delta(\mathbf{e}') = \Delta(\mathbf{e}) = w$. WLOG we can assume $\mathbf{e}^T = \pi((\bar{\mathbf{e}}, \mathbf{0}^{iw})^T)$ is a solution. Then the minimal distance of the code generated by $\mathbf{H}^\perp$ is less than $2w$, which happens with negligible probability $2^{-\sigma}$ by Lemma 5.

2. $\bar{\mathbf{y}}$ is uniformly sampled from $\mathbb{F}_q^n$. In this case the input given to $\mathcal{A}$ is exactly $\mathsf{Hyb}^{(i)}$.

It follows that $\mathsf{Hyb}^{(i)}$ and $\mathsf{Hyb}^{(i+1)}$ are κ_i-bit computationally indistinguishable and σ-bit statistically indistinguishable.

By definition, any adversary $\mathcal{A}$ with running time T and against $\mathsf{Hyb}^{(i)}$ and $\mathsf{Hyb}^{(i+1)}$ has advantage $\varepsilon_i \leq T/2^{-\kappa_i}$. According to Lemma 2, any adversary of the $\mathsf{MDSD}_{m,n,c,w}$ problem has advantage at most $3c\sum_{i=1}^{c} T/2^{-\kappa_i}$. Therefore $\mathsf{MDSD}_{m,n,c,w}$ is at least $(-\log(3c) - \log\sum_{i=1}^{c}\frac{1}{2^{\kappa_i}})$-bit computationally hard. $\square$

Concrete security. When w is smaller than the GV-distance of the code $\mathcal{C}^{\perp}(\mathbf{H})$, the best known algorithms that solve the syndrome decoding problem are the class of Information-Set Decoding (ISD) algorithms such as Prange [Pra62], Stern [Ste88], Dumer [Dum91], and recent ISD variants such as May-Meurer-Thomae [MMT11], Becker-Joux-May-Meurer [BJMM12], May-Ozerov [MO15], and Both-May [BM18]. In the setting of our shuffle PIR protocol, we would like to keep w small to minimize the client communication cost and the server's computation cost, and we will always set w to be smaller than the GV-distance. For sublinear $w = o(m)$, it is known [TS16] that the asymptotic complexity of these ISD algorithms is $2^{-\log(n/m)\cdot w\cdot(1+o(1))}$.

In the formulation of Lemma 6, there are λ columns of $\mathbf{H}$ that correspond to 0 entries of $\mathbf{E}$, and hence these columns are independent in an MDSD sample $(\mathbf{H}, \mathbf{H}\cdot\mathbf{E})$. These vectors "hide" the remaining, correlated shares. Assume $\mathsf{SD}_{n,w+\lambda,w}$ has κ bits of security. We may use $\kappa - 2\log c$ as a rough upper bound on the bit security of $\mathsf{MDSD}_{n,cw+\lambda,c,w}$ problem. In order for our shuffle PIR protocol to achieve κ' bits of security against ISD attacks, we should set $w = \frac{\kappa'+2\log c}{\log\lambda - \log n}$. In particular, this estimation requires $\lambda > n$.

3.3 Security of MDSD Problem

In the previous subsection we show that there is a $\mathsf{poly}(c)$ time reduction from standard SD to the $\mathsf{MDSD}_{m,n,c,w}$ problem, which, however, is not tight. In particular, it does not seem to correspond to a very efficient distinguishing attack on the MDSD problem. In this section, we study the hardness of distinguishing the $\mathsf{MDSD}_{m,n,c,w}$ distribution from uniformly random.

3.3.1 Search-to-decision Reduction

Consider the search version of MDSD problem: given a $\mathsf{MDSD}_{n,m,c,w}$ sample $(\mathbf{H}, \mathbf{Y} = \mathbf{H}\cdot\mathbf{E})$, the adversary is asked to return *at least* one column of the error matrix $\mathbf{E}$. Note that, in order to distinguish an MDSD sample with uniform $(\mathbf{H}, \mathbf{Y})$, it is sufficient to recover just one error vector $\mathbf{e}$ such that $\mathbf{y}_i = \mathbf{H}\cdot\mathbf{e}$ for some $1 \leq i \leq c$, as the probability of the existance of $\mathbf{e}$ such that $\mathbf{H}\cdot\mathbf{e} = \mathbf{y}$ is negligible for random $\mathbf{H}$ and $\mathbf{y}$.

On the other hand, Lemma 6 implies a reduction from the search MDSD problem to its decision variant. This can be seen as follows. Assume there exists a distinguisher $\mathcal{D}$ for $\mathsf{MDSD}_{n,m,c,w}$ with advantage $\varepsilon > 0$. Then there must exist $0 \leq i < c$ and a distinguisher $\mathcal{D}^{(i)}$ for the hybrids $\mathsf{Hyb}^{(i)}$ and $\mathsf{Hyb}^{(i+1)}$ as in the proof of Lemma 6, such that $\mathcal{D}^{(i)}$'s advantage is at least ε/c. Now, $\mathcal{D}^{(i)}$ can be used to build a distinguisher for $\mathsf{SD}_{n,m',c,w}$ and the uniform distribution, where $m' = m - (c-i)w$, and by the search-to-decision reduction of Syndrome Decoding (and its dual problem LPN) [BFKL94, AIK07, KSS10, MM11], we can then build an adversary that recovers $\mathbf{e}_i$.

3.3.2 Security of Search MDSD Problem

In the MDSD distribution, the syndrome vectors (columns of $\mathbf{H}\cdot\mathbf{E}$) are generated from correlated error vectors (columns of $\mathbf{E}$), where distinct error vectors $\mathbf{e}_i, \mathbf{e}_j$ do not have overlapping entries of 1's. If we instead consider *independent* error vectors, then the problem becomes more general, and it is commonly referred to as the *Decoding One Out of Many* (DOOM) [Sen11], or the Multi-Syndrome Decoding problem.

Definition 9 (Multi Syndrome Decoding)**.** *Let $\mathbb{F}$ be a finite field, and let $n, m, c, w > 0$. The (n, m, c, w)-*Multi Syndrome Decoding *distribution is*

$$\mathsf{MultiSD}_{n,m,c,w} = \left\{ \left(\mathbf{H}, \mathbf{H} \cdot \mathbf{e}_1, \ldots, \mathbf{H} \cdot \mathbf{e}_c \right) \middle| \begin{array}{l} \mathbf{H} \leftarrow_{\$} \mathbb{F}^{n \times m}, \\ \mathbf{e}_1 \leftarrow \mathsf{Ber}_\tau^m, \ldots, \\ \mathbf{e}_c \leftarrow \mathsf{Ber}_\tau^m \end{array} \right\}.$$

*The (n, m, c, w)-*Decisional Multi Syndrome Decoding *problem is to distinguish the distribution* $\mathsf{MultiSD}_{n,m,c,w}$ *from the uniform distribution over $\mathbb{F}^{n \times m} \times \mathbb{F}^{n \times c}$.*

The search version of Multi-Syndrome Decoding (MSD) asks the adversary to recover at least one error vector $\mathbf{e}_j$. Clearly, MDSD is a restricted variant of MSD, and any MDSD attacker can automatically solve the MSD problem. So, the complexity of an MSD solver is a lower bound on the concrete security for MDSD.

For the MDSD parameters that we are interested in, where $w = O(\log m)$ is the Hamming weight of error vectors, since w is smaller than the GV-distance of the code $\mathcal{C}(\mathbf{H}^\perp)$, Information Set Decoding (ISD) algorithms are the most efficient for solving standard SD and MSD problems. For sublinear $w = o(m)$, it is known [TS16] that the asymptotic complexity of the ISD algorithms for solving $\mathsf{SD}_{n,m,w}$ is $2^{-\log(n/m) \cdot w \cdot (1+o(1))}$. Assume T_{SD} is the concrete running time of an ISD algorithm for solving $\mathsf{SD}_{n,m,w}$. As shown in [Sen11], such algorithm can be converted to solve $\mathsf{MultiSD}_{n,m,c,w}$ with running time $T_{\mathsf{MultiSD}} = T_{\mathsf{SD}}/c^\gamma$, where $\gamma < 1/2$. In practice, when m is large, the factor γ is much smaller than $1/2$.

We conjecture that the DOOM algorithm [Sen11] is the most efficient one for breaking the MDSD problem. As a generic extension of ISD algorithms, DOOM maintains two sets of vectors computed from the multiple syndrome vectors in order to perform collision tests more efficiently. Although the error vectors in MDSD have additional structure that the non-zero entries are correlated, such information does not seem to be useful in the DOOM algorithm. A similar situation is the relation between the LPN and the regular LPN problems: while the error vector in regular LPN has an regular noise pattern, such structure does not seem to allow a significant speedup in the attacking algorithms for the parameter regimes similar to ours. As analyzed in [HOSS22], ISD algorithms are (among combinatorial attacks) the most efficient in attacking regular LPN when the error Hamming weight is below the GV-distance, and they do not perform better on regular LPN comparing to standard LPN with the same dimension. The recent algebraic attack to regular LPN [BØ23] seems to be limited to the cases where the code rate $m/(m-n)$ is small, which does not apply to our proposed parameters. One plausible approach to study our conjectured hardness of MDSD is to apply the linear test framework of [BCG+20], by checking if there are noticable bias when applying any linear function to the syndrome vectors. We do not know how to bound such bias for the MDSD distribution, and we leave it as an open question.

4 Secure Aggregation

In this section, we show how we can use the split-and-mix core lemma from Section 3 to construct a secure aggregation protocol. We start with a definition of secure aggregation in the shuffle model.

Definition 10 (Secure Aggregation in the Shuffle Model)**.** *Let $\mathbb{F}$ be a finite field, and let $n, c \in \mathbb{N}$. A (single-server)* secure aggregation *protocol over c clients in the* shuffle model *is a tuple of algorithms* $\mathsf{ShAgg} = (\mathsf{Share}, \mathsf{Mix}, \mathsf{Aggregate})$*:*

- $\mathsf{Share}(\mathbf{x} \in \mathbb{F}^n) \xrightarrow{\$} S$*: a randomized algorithm executed by the client that takes in an input $\mathbf{x} \in \mathbb{F}^n$, and outputs a set of shares*[3] *S.*

- $\mathsf{Mix}(S_1, \ldots, S_c) \xrightarrow{\$} \hat{S}$*: a randomized algorithm executed by the shuffler that takes in the sets of shares from c clients $S_1, \ldots, S_c$, and output a set of shuffled shares $\hat{S}$.*

- $\mathsf{Aggregate}(\hat{S}) \to a$*: a deterministic algorithm executed by the server that takes in the shuffled elements and outputs an aggregated answer a.*

[3]In the context of secure aggregation protocol, we follow the convention to call client output *shares* of its input.

Correctness. For all $n, c \in \mathbb{N}$, a secure aggregation protocol over c clients is *correct* if for all $\mathbf{x_1}, \ldots, \mathbf{x_c} \in \mathbb{F}^n$,

$$\Pr\left[a = \sum_{i=1}^{c} \mathbf{x_i} \;\middle|\; \begin{array}{rcl} S_i & \leftarrow & \mathsf{Share}(\mathbf{x_i})\ \forall i \in [c], \\ \hat{S} & \leftarrow & \mathsf{Mix}(S_1, \ldots, S_c), \\ a & \leftarrow & \mathsf{Aggregate}(\hat{S}) \end{array} \right] = 1.$$

Security. For all $n, c \in \mathbb{N}$, an aggregation protocol over c clients in the shuffle model is *secure* if for all $\mathbf{x_1}, \ldots, \mathbf{x_c}, \mathbf{x'_1}, \ldots, \mathbf{x'_c} \in \mathbb{F}^n$ such that $\sum_{i=1}^{c} \mathbf{x_i} = \sum_{i=1}^{c} \mathbf{x'_i}$, the following two distributions V and V' are computationally indistinguishable:

$$V = \left\{ \hat{S} \;\middle|\; \begin{array}{rcl} S_i & \leftarrow & \mathsf{Share}(\mathbf{x_i})\ \forall i \in [c], \\ \hat{S} & \leftarrow & \mathsf{Mix}(S_1, \ldots, S_c) \end{array} \right\},$$

$$V' = \left\{ \hat{S}' \;\middle|\; \begin{array}{rcl} S'_i & \leftarrow & \mathsf{Share}(\mathbf{x'_i})\ \forall i \in [c], \\ \hat{S}' & \leftarrow & \mathsf{Mix}(S'_1, \ldots, S'_c) \end{array} \right\}.$$

Our secure aggregation construction has clients additively share their input vectors into $w + 1$ shares, for some public parameter $w \in \mathbb{N}$.

Construction 1 (Computationally Secure Aggregation in the Shuffle Model)**.** *Let $\mathbb{F}$ be a finite field, let $n, c, w > 0$. We define a single-server aggregation protocol in the* shuffle model *as follows.*

- $\mathsf{Share}(\mathbf{x} \in \mathbb{F}^n) \overset{\$}{\rightarrow} (\mathbf{s}_0, \ldots, \mathbf{s}_w)$*: samples $\mathbf{s}_i \overset{\$}{\leftarrow} \mathbb{F}^n$ for $i = 0, \ldots, w - 1$, and let $\mathbf{s}_w = \mathbf{x} - \sum_{i=0}^{w-1} \mathbf{s}_i$.*
- $\mathsf{Mix}(S_1, \ldots, S_c) \overset{\$}{\rightarrow} \hat{S}$*: samples a uniform random distribution $\pi \overset{\$}{\leftarrow} \Pi_{\sum_{i=1}^{c} |S_i|}$, and outputs $\hat{S} := \pi(S_1, \ldots, S_c)$.*
- $\mathsf{Aggregate}(\hat{S}) \rightarrow a$*: outputs $a := \sum_{\mathbf{s} \in \hat{S}} \mathbf{s}$.*

Theorem 1. *Let $n, c, w > 0$, and let $\{\mathbf{x_i}\}_{i=1}^{c}$ and $\{\mathbf{x'_i}\}_{i=1}^{c}$ be any two sequences of vectors over a finite field $\mathbb{F}$ such that $\sum_{i=1}^{c} \mathbf{x_i} = \sum_{i=1}^{c} \mathbf{x'_i}$. Assume the decisional $\mathsf{MDSD}_{n,m,c,w}$ is hard over $\mathbb{F}$. Then the aggregation protocol of Construction 1 is computationally secure.*

The security of the above constructions follows from the split-and-mix Lemma 4, which states that the server's view on any input set is indistinguishable from the distributions of shuffled shares of the sum of the input vectors.

Proof. Let $\mathbf{X} \in \mathbb{F}^{c \times c}$ be a matrix with columns $\mathbf{x}_1, \ldots, \mathbf{x}_c$, and $\mathbf{X}'$ a matrix with columns $\mathbf{x}'_1, \ldots, \mathbf{x}'_c$. Furthermore, let $m = c \cdot w$. The server's view given inputs $\{\mathbf{x}_i\}_{i=0}^{c}$ and $\{\mathbf{x}'_i\}_{i=0}^{c}$ are shuffles of samples from the following distributions D and D':

$$D = \left\{ (\mathbf{H}, \mathbf{X} - \mathbf{Y}, \mathbf{z} = \mathbf{x}_0 - \sum_{j=1}^{m} \mathbf{h}_j + \sum_{i=1}^{c} \mathbf{y}_i) \;\middle|\; \begin{array}{l} \mathbf{H} \overset{\$}{\leftarrow} \mathbb{F}^{n \times m}, \\ \mathbf{E} \leftarrow \mathsf{DisError}_{m,c,w} \\ \mathbf{Y} = \mathbf{H} \cdot \mathbf{E} \end{array} \right\},$$

$$D' = \left\{ (\mathbf{H}, \mathbf{X}' - \mathbf{Y}, \mathbf{z} = \mathbf{x}'_0 - \sum_{j=1}^{m} \mathbf{h}_j + \sum_{i=1}^{c} \mathbf{y}_i) \;\middle|\; \begin{array}{l} \mathbf{H} \overset{\$}{\leftarrow} \mathbb{F}^{n \times m}, \\ \mathbf{E} \leftarrow \mathsf{DisError}_{m,c,w} \\ \mathbf{Y} = \mathbf{H} \cdot \mathbf{E} \end{array} \right\}$$

First notice that, by Lemma 4 and setting $w_0 = w$, D is indistinguishable from

$$\tilde{D} = \left\{ (\mathbf{H}, \mathbf{X} - \mathbf{Y}, \mathbf{z} = \mathbf{x}_0 - \sum_{j=1}^{m} \mathbf{h}_j + \sum_{i=1}^{c} \mathbf{y}_i) \;\middle|\; \begin{array}{l} \mathbf{H} \overset{\$}{\leftarrow} \mathbb{F}^{n \times m}, \\ \mathbf{Y} \overset{\$}{\leftarrow} \mathbb{F}^{n \times c} \end{array} \right\}.$$

Now, since $\mathbf{Y}$ in $\tilde{D}$ is uniformly random, and since $\sum_{i=0}^{c} \mathbf{x}'_i = \sum_{i=0}^{c} \mathbf{x}_i$, the distribution $\tilde{D}$ is equivalent to

$$\tilde{D} \equiv \left\{ (\mathbf{H}, \mathbf{X}' - \mathbf{Y}, \mathbf{z} = \mathbf{x}'_0 - \sum_{j=1}^{m} \mathbf{h}_j + \sum_{i=1}^{c} \mathbf{y}_i) \;\middle|\; \begin{array}{l} \mathbf{H} \leftarrow \mathbb{F}^{n \times m}, \\ \mathbf{Y} \leftarrow \mathbb{F}^{n \times c} \end{array} \right\}.$$

By Lemma 4 again, we see that $\tilde{D}$ is indistinguishable from D'. $\square$

Optimizations. We can compress all but one shares in an input by using PRG seeds. This improves the communication cost and helps us reach optimal expansion ratio (see Section 6). We will explicitly prove the security of this optimized approach in the random oracle model for our secure aggregation construction, and an analogous proof follows for our shuffle PIR construction.

Besides, in the proxy model for shuffle PIR where all the client shares are shuffled by a proxy party who does not collude with the server, we can have the proxy insert $w_0 \gg w$ many dummy shares whose sum is the zero vector. The security of this proxy-mode construction also follows from the split-and-mix Lemma 4. Since ISD attacks have asymptotic complexity $2^{-\log(n/m)\cdot w}$ where $m = w_0 + w \cdot c$, this proxy-inserting reduces the number of shares w from each client.

4.1 Reducing Long-Vector Aggregation to Short-Vector Aggregation

As the input vector becomes longer, the number of shares per input must grow in order to maintain security level. We now discuss a reduction that allows to keep the number of shares low while working with long inputs, with the help of a key- and message-additive homomorphic encryption (KMAHE) scheme $\mathcal{E}$ (see Section 2.5) with a special leakage resilience property. This idea was leveraged by Bell et al. [BGL+23] in their interactive single server secure aggregation constructions, and we show how it can also be applied in the shuffle model. Roughly speaking, we require the KMAHE scheme $\mathcal{E}$ to have IND security even when the *sum* of symmetric keys is leaked. In the resulting shuffle-model aggregation protocol, each client encrypts its input using a fresh (symmetric) encryption key of $\mathcal{E}$ and sends such ciphertext to the server, and then all clients invoke the aggregation protocol in Construction 1 on their encryption keys. The server can then decrypt the sum of KMAHE ciphertexts using the aggregated encryption keys, and learn the aggregated input. Below we formally present such hybrid construction.

Construction 2 (Secure Aggregation with KMAHE in the Shuffle Model)**.** *For any $c \in \mathbb{N}$, let $\mathcal{E}$ be an KMAHE encryption scheme with key space $\mathcal{K}$ and plaintext space $\mathbb{F}^n$. Let $\mathsf{ShAgg} = (\mathsf{Share}, \mathsf{Mix}, \mathsf{Aggregate})$ be a secure aggregation protocol for c clients in the shuffle model with message space $\mathcal{K}$. Define a new aggregation protocol $\mathsf{ShAgg}' = (\mathsf{Share}', \mathsf{Mix}', \mathsf{Aggregate}')$ over $\mathbb{F}^n$ as follows:*

- $\mathsf{Share}'(\mathbf{x} \in \mathbb{F}^n) \overset{\$}{\rightarrow} (\mathsf{ct}, S)$*: samples a random key* $\mathsf{sk} \leftarrow_\$ \mathcal{K}$ *and encrypts* $\mathsf{ct} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{sk}, \mathbf{x})$*. Invokes* $S \leftarrow \mathsf{ShAgg}.\mathsf{Share}(\mathsf{sk})$*. Outputs* (ct, S)*.*
- $\mathsf{Mix}'((\mathsf{ct}_1, S_1), \ldots, (\mathsf{ct}_c, S_c)) \overset{\$}{\rightarrow} (\mathsf{ct}_1, \ldots, \mathsf{ct}_c, \hat{S})$*: invokes* $\hat{S} \leftarrow \mathsf{ShAgg}.\mathsf{Mix}(S_1, \ldots, S_c)$*. Outputs* $(\mathsf{ct}_1, \ldots, \mathsf{ct}_c, \hat{S})$*.*
- $\mathsf{Aggregate}'(\hat{\mathsf{ct}}, \hat{S}) \rightarrow a$*: invokes* $a_S \leftarrow \mathsf{ShAgg}.\mathsf{Aggregate}(\hat{S})$*. Outputs* $a = \mathcal{E}.\mathsf{Dec}(a_S, \sum_{i=1}^{c} \mathsf{ct}_i)$*.*

Note that the $\mathcal{E}$ ciphertexts are not required to be anonymous, and they can be sent directly to the server.

Theorem 2. *For any positive integers $c, n > 0$, let $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ be a KMAHE encryption scheme with key space $\mathcal{K}$, message space $\mathbb{F}^n$ and ciphertext space $\mathcal{C}$, and assume it is c-IND secure under leakage of sum of secret keys as in Definition 6. Let $\mathsf{ShAgg}_{\mathcal{K},c,\lambda} = (\mathsf{Share}, \mathsf{Mix}, \mathsf{Aggregate})$ be the aggregation protocol of Construction 1 with input space $\mathcal{K}$. Then, the protocol in Construction 2 is a c-client secure aggregation protocol in the shuffle model, with per-client communication $O(n \cdot |\mathcal{C}|) + \tilde{O}(|\mathcal{K}|)$.*

Under the standard RLWE assumption, Bell et al [BGL+23, Appendix A] builds an KAHE scheme with the leakage security property of Definition 6 for any polynomially bounded c. We now sketch a proof of Theorem 2.

Proof(Sketch). Fix any two sequences of input vectors $\{\mathbf{x_i}\}_{i=1}^{c}$ and $\{\mathbf{x'_i}\}_{i=1}^{c}$ over $\mathbb{F}^n$ such that $\sum_{i=1}^{c} \mathbf{x_i} = \sum_{i=1}^{c} \mathbf{x'_i}$. The server's views under $\{\mathbf{x_i}\}_{i=1}^{c}$ is

$$V = \left\{ (\mathbf{H}, -\mathbf{Y} + \mathbf{S}, \mathbf{z}, \{\mathsf{ct}_i\}_{i=1}^{c}) \;\middle|\; \begin{array}{r} \mathbf{H} \leftarrow_\$ \mathbb{F}^{\ell \times m}, \mathbf{E} \leftarrow \mathsf{DisError}, \mathbf{Y} = \mathbf{H} \cdot \mathbf{E}, \\ \forall 1 \le i \le c.\ \mathsf{sk}_i \leftarrow_\$ \mathcal{K}, \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, \mathbf{x_i}), \\ \mathbf{S} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_{c-1}), \mathbf{z} = -\sum_j \mathbf{h}_j + \sum_i \mathbf{y}_i + \mathsf{sk}_c \end{array} \right\}.$$

By Lemma 4, V is indistinguishable from the distribution $\mathsf{Hyb}^{(1)}$ over $(\mathbf{H}, -\mathbf{Y}+\mathbf{S}, \mathbf{z}, \{\mathsf{ct}_i\}_{i=1}^c)$ where $\mathbf{Y}$ is uniformly random and other components are identically distributed as in V. Next, since $\mathbf{Y}$ is uniform in $\mathsf{Hyb}^{(1)}$, $\mathsf{Hyb}^{(1)}$ is identical to $\mathsf{Hyb}^{(1)}$ where we replace $-\mathbf{Y}+\mathbf{S}$ with $\mathbf{Y}$ and let $\mathbf{z} = -\sum_j \mathbf{h}_j + \sum_i \mathbf{y}_i + \sum_{i=1}^c \mathsf{sk}_i$. Now, by the leakage security property of $\mathcal{E}$, $\mathsf{Hyb}^{(1)}$ is indistinguishable from the distribution $\mathsf{Hyb}^{(2)}$ over $(\mathbf{H}, \mathbf{Y}, \mathbf{z}, \{\mathsf{ct}'_i\}_{i=1}^c)$ where $\mathsf{ct}'_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, \mathbf{x}'_\mathbf{i})$ for all i. Finally, by a symmetric argument, we see that $\mathsf{Hyb}^{(2)}$ is indistinguishable from the server's view V' on input $\{\mathbf{x}'_\mathbf{i}\}_{i=1}^c$. $\square$

5 PIR in the Shuffle Model

5.1 Definitions

We now formally define single-server PIR in the shuffle model; we consider a single server but many query-making clients. Importantly, we do not assume any coordination among clients. In this section, we require the shuffler to only provide a random permutation; Appendix B expands the role of the shuffler to perform additional computation.

Definition 11 (PIR in the shuffle model)**.** *Let Σ be a finite alphabet and $n \in \mathbb{N}$. PIR in the shuffle model on database Σ^n is a tuple* $\mathsf{ShPIR} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Recon})$*, run by a server and a set of clients defined as below.*

- $\mathsf{Setup}(x) \to P_x$*: a deterministic algorithm executed by the server that takes in an n-entry database $x \in \Sigma^n$ and outputs its encoding P_x.*
- $\mathsf{Query}(i; n) \xrightarrow{\$} ((q_1, \dots, q_k), \mathsf{st})$*: a randomized algorithm (parameterized by n) executed by the client that takes in an index $i \in [n]$, and outputs sub-queries $q_1, \dots, q_k$ and a state* st*. Note that k may be a (randomized) function of n.*
- $\mathsf{Answer}(P_x, q_\ell) \to a_\ell$*: a deterministic algorithm executed by the server that takes in the encoding P_x and a sub-query q_ℓ, and outputs an answer a_ℓ.*
- $\mathsf{Recon}((a_1, \dots, a_k), \mathsf{st}) \to x_i$*: a deterministic algorithm executed by the client that takes in answers $a_1, \dots, a_k$ and a state* st*, where for all $\ell \in [k]$, a_ℓ is the answer to the client's sub-query q_ℓ; and outputs $x_i \in \Sigma$.*

Correctness. *For all $n \in \mathbb{N}$, any database $x = (x_1, \dots, x_n) \in \Sigma^n$, and every $i \in [n]$,*

$$\Pr\left[\mathsf{Recon}(a_1, \dots, a_k) = x_i \;\middle|\; \begin{array}{r} P_x = \mathsf{Setup}(x) \\ (q_1, \dots, q_k) \leftarrow \mathsf{Query}(i; n) \\ (a_1, \dots, a_k) = (\mathsf{Answer}(P_x, q_\ell))_{\ell=1}^k \end{array}\right] = 1.$$

Security. *We will parameterize security by a shuffler Π and a minimum number of client queries c. For given n, Π, and c, and given a tuple $I = (i_1, \dots, i_c) \in [n]^c$ of client query indices, define the distribution*

$$\widetilde{\mathcal{D}}_{n,\Pi,c}(I) = \left\{ \pi(\mathbf{q}) \;\middle|\; \begin{array}{r} (q_1^{(1)}, \dots, q_k^{(1)}) \xleftarrow{\$} \mathsf{Query}(i_1; n) \\ \cdots \\ (q_1^{(c)}, \dots, q_k^{(c)}) \xleftarrow{\$} \mathsf{Query}(i_c; n) \\ \mathbf{q} \leftarrow (q_1^{(1)}, \dots, q_k^{(1)}, \dots, q_1^{(c)}, \dots, q_k^{(c)}) \\ \pi \xleftarrow{\$} \Pi_{kc} \end{array} \right\}.$$

Then, we say that ShPIR *is (Π, c, ϵ) computationally secure if for every $n \in \mathbb{N}$ and all $c^* \geq c(n)$, and $I, I' \in [n]^{c^*}$, and any probabilistic polynomial-time adversary $\mathcal{A}$, it holds that:*

$$|\Pr[\mathcal{A}(\widetilde{\mathcal{D}}_{n,\Pi,c^*}(I)) = 1] - \Pr[\mathcal{A}(\widetilde{\mathcal{D}}_{n,\Pi,c^*}(I')) = 1]| \leq \epsilon(n),$$

where the probability is taken over the randomness of Π *and all the algorithms in* ShPIR*. For ease of presentation, we will parameterize a* ShPIR *protocol running on database* Σ^n *with* c *clients as a tuple* (Σ, c, n, k)*, and given a such tuple, there will be a corresponding security level* κ*. Later in our construction, we may split* k *into two parameters to differentiate the sub-query types.*

Remark 1 (Randomized number of sub-queries)*.* Unlike in standard multi-server PIR, k may be a function of n; this is possible since the shuffle model does not require k physical servers. In the shuffle model, all sub-queries will be sent to one server.

5.2 Construction

We start by describing the key idea of our construction. Fix a finite field $\mathbb{F}$ such that Σ can be encoded. On input $i \in [n]$, the client generates $w + 1 + z$ vectors in $\mathbb{F}^n$, which are called *sub-queries* in our protocol, and sends them to the shuffler who will shuffle sub-queries from all clients and forward them to the server. Each set of sub-queries consists of $k = w + 1$ *real* sub-queries $\mathbf{s}_0, \ldots, \mathbf{s}_w$ which are additive secret shares of the indicator vector $\mathbf{u}_i$ encoding the index i, as well as z *dummy* sub-queries $\mathbf{r}_1, \ldots, \mathbf{r}_z$ which are independent and uniformly random vectors in $\mathbb{F}^n$. We index the real sub-queries using subscripts starting from 0 for convenience of stating our security assumption, which will become clear in Section 5.3. For the dummies, the total number of dummy sub-queries $\lambda = z \cdot c$ will play an important role in our security analysis, and so we index them starting from 1.

Once the sub-queries are shuffled by the shuffler, the server processes each sub-query, which simply computes the inner product between the database (as a vector in $\mathbb{F}^n$) and the sub-query vector as the response. All the responses are routed back through the shuffler to the originating client. Then client can recover the requested database element by computing the sum of responses corresponding to the real sub-queries.

This basic protocol has each sub-query being a vector of length n (linear in the database size). We can reduce the query size by compressing all but one sub-queries using PRG seeds, and the server can expand them into vectors in $\mathbb{F}^n$ to answer the sub-queries. [4] We give the protocol description in Construction 3.

Construction 3 (Shuffle PIR Protocol)**.** *Let* $\mathbb{F}$ *be a field and* $c, n, w, z \in \mathbb{N}$*, and let* $\mathbf{x} \in \mathbb{F}^n$*. Assume* $\mathsf{G} : \{0,1\}^s \to \mathbb{F}^n$ *is a random oracle with output space* $\mathbb{F}^n$*. The PIR protocol parameterized by* $(\mathbb{F}, c, n, w, z)$ *consists of the following algorithms.*

- $\mathsf{Setup}(\mathbf{x}) \to \mathbf{x}$*: The setup algorithm does not modify the database and treats it as a vector of length* n *over* $\mathbb{F}$*.*
- $\mathsf{Query}(i; n) \overset{\$}{\to} (\mathbf{s}_0, \mathsf{seed}_1, \ldots, \mathsf{seed}_w, \mathsf{seed}'_1, \ldots, \mathsf{seed}'_z)$*: The query algorithm samples* $w + z$ *independent PRG seeds* $\mathsf{seed}_1, \ldots, \mathsf{seed}_w$*,* $\mathsf{seed}'_1, \ldots, \mathsf{seed}'_z$*, and computes* $\mathbf{s}_0 = \mathbf{u}_i - \sum_{i=1}^{w} \mathsf{G}(\mathsf{seed}_i)$*, where* $\mathbf{u}_i \in \mathbb{F}^n$ *is the indicator vector of query index* i*. It outputs the tuple* $(\mathbf{s}_0, \mathsf{seed}_1, \ldots, \mathsf{seed}_w, \mathsf{seed}'_1, \ldots, \mathsf{seed}'_z)$*.*
- $\mathsf{Answer}(\mathbf{x}, \mathbf{q}) \to \langle \mathbf{x}, \mathbf{q} \rangle \in \mathbb{F}$*. Given a sub-query* $\mathbf{q} \in \mathbb{F}^n$*, the server computes the inner product between the database* $\mathbf{x}$ *and the sub-query* $\mathbf{q}$*, and returns it to the client.*
- $\mathsf{Recon}(a_0, \ldots, a_w, d_1, \ldots, d_z) \to \sum_{i=0}^{w} a_i \in \mathbb{F}$*: To reconstruct the requested element, the client computes the sum of* $a_0, \ldots, a_w$ *that correspond to the real sub-queries* $\mathbf{s}_0, \mathsf{seed}_1, \ldots, \mathsf{seed}_w$*. Answers* $d_1, \ldots, d_z$ *corresponding to the dummy sub-queries are discarded.*

Remark 2*.* Note that the shuffle model only ensure anonymity, so here messages are not necessarily hidden from the communication intermediary between the server and the clients. If we want to further hide the query, we can have clients encrypt the query and the answer as follows. For query, the clients use server's public key. For answers, the client sends with each share (a sub-query) a fresh symmetric key to be used to encrypt the answer. This guarantees that the server will not link shares of the same client.

We can view the sub-queries (after expanding using PRG) received by the server as vectors in $\mathbb{F}^n$. Thus, the server's view in our protocol with c clients is a matrix in $\mathbb{F}^{n \times c(w+z+1)}$.

[4]As we will prove, this compression technique is secure when the PRG is modeled as a random oracle.

5.3 Security

We now prove that our PIR protocol is computationally secure in the shuffle model, where the security can either be based on standard hardness assumptions about decoding random linear codes, or on our new assumption on the hardness of solving MDSD problem.

Theorem 3 (Security). *Let $n, m, c, w > 0$ such that $m = c \cdot w + \lambda$ for some $\lambda > 0$. Assume $\mathsf{MDSD}_{n,m,c,w}$ over $\mathbb{F}$ is κ-bit computationally hard (c.f. Definition 5). Furthermore, assume G is modeled as a random oracle. Then the shuffle PIR protocol in Construction 3 parameterized by $(\mathbb{F}, c, n, w, \lceil \lambda/c \rceil)$ is $(\Pi, c, 2^{-\kappa})$ computationally secure in the shuffle model and random oracle model.*

Proof. Let $I = (i_1, \ldots, i_c), \hat{I} = (\hat{i}_1, \ldots, \hat{i}_c) \in [n]^c$ be arbitrary sequences of query indices. Let $\mathbf{u}_1, \ldots, \mathbf{u}_c$ be the indicator vectors corresponding to indices in I. Similarly let $\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_c$ be the corresponding indicator vectors of indices in $\hat{I}$. The server's view for queries I is the following distribution V:

$$\left\{ \pi \left(\mathsf{seed}^{(1)}, \mathsf{seed}'^{(1)}, \ldots, \mathsf{seed}^{(c)}, \mathsf{seed}'^{(c)}, \mathbf{s}_0^{(1)}, \ldots, \mathbf{s}_0^{(c)} \right) \;\middle|\; \begin{array}{r} \forall j. (\mathbf{s}_0^{(j)}, \mathsf{seed}^{(j)}, \mathsf{seed}'^{(j)}) \leftarrow \mathsf{Query}(i_j), \\ \pi \leftarrow_{\$} \Pi \end{array} \right\}.$$

where, for each $1 \leq j \leq c$, we denote using $\mathsf{seed}^{(j)}$ the tuple of seeds corresponding to the real sub-queries and $\mathsf{seed}'^{(j)}$ the tuple of seeds corresponding to the dummy sub-queries in $\mathsf{Query}(i_j)$. Similarly, the server's view given the indices $\hat{I}$ is $\hat{V}$:

$$\left\{ \pi \left(\widehat{\mathsf{seed}}^{(1)}, \widehat{\mathsf{seed}}'^{(1)}, \ldots, \widehat{\mathsf{seed}}^{(c)}, \widehat{\mathsf{seed}}'^{(c)}, \hat{\mathbf{s}}_0^{(1)}, \ldots, \hat{\mathbf{s}}_0^{(c)} \right) \;\middle|\; \begin{array}{r} \forall j. (\hat{\mathbf{s}}_0^{(j)}, \widehat{\mathsf{seed}}^{(j)}, \widehat{\mathsf{seed}}'^{(j)}) \leftarrow \mathsf{Query}(\hat{i}_j), \\ \pi \leftarrow_{\$} \Pi \end{array} \right\}.$$

Let $\mathbf{X} = (\mathbf{u}_1, \ldots, \mathbf{u}_c)$, and define the distribution $\tilde{V}$ as

$$\left\{ \pi(\mathbf{H}, -\mathbf{H} \cdot \mathbf{E} + \mathbf{X}) \;\middle|\; \begin{array}{r} \mathbf{E} \leftarrow \mathsf{DisError}_{m,c,w}, \\ \mathbf{H} = \tau_{\mathbf{E}}(\mathsf{G}(\mathsf{seed}^{(1)}), \mathsf{G}(\mathsf{seed}'^{(1)}), \ldots, \mathsf{G}(\mathsf{seed}^{(c)}), \mathsf{G}(\mathsf{seed}'^{(c)})), \\ \pi \leftarrow_{\$} \Pi \end{array} \right\},$$

where by $\mathsf{G}(\mathsf{seed}^{(i)})$ we mean the vectors $\mathsf{G}(\mathsf{seed}_1^{(i)}), \ldots, \mathsf{G}(\mathsf{seed}_w^{(i)})$ obtained by invoking RO on all w seeds in $\mathsf{seed}^{(i)}$ independently, and similarly for $\mathsf{G}(\mathsf{seed}'^{(i)})$; and $\tau_{\mathbf{E}}$ is the permutation derived from the error matrix $\mathbf{E}$ such that the non-zero entries of the j'th column $\mathbf{e}_j$ correspond to the positions of $\mathsf{G}(\mathsf{seed}^{(i)})$ under the permutation $\tau_{\mathbf{E}}$. Such permutation $\tau_{\mathbf{E}}$ makes sure that, for each $1 \leq j \leq c$, the columns $\{\mathbf{h}_k : \mathbf{e}_{j,k} \neq 0\}$ together with $-\mathbf{H} \cdot \mathbf{e}_j + \mathbf{u}_j$ form an additive sharing of $\mathbf{u}_j$. Similarly, let $\hat{\mathbf{X}} = (\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_c)$, and define the distribution $\tilde{\tilde{V}}$ as

$$\left\{ \pi(\mathbf{H}, -\mathbf{H} \cdot \mathbf{E} + \hat{\mathbf{X}}) \;\middle|\; \begin{array}{r} \mathbf{E} \leftarrow \mathsf{DisError}_{m,c,w}, \\ \mathbf{H} = \tau_{\mathbf{E}}(\mathsf{G}(\widehat{\mathsf{seed}}^{(1)}), \mathsf{G}(\widehat{\mathsf{seed}}'^{(1)}), \ldots, \mathsf{G}(\widehat{\mathsf{seed}}^{(c)}), \mathsf{G}(\widehat{\mathsf{seed}}'^{(c)})), \\ \pi \leftarrow_{\$} \Pi \end{array} \right\}.$$

In $\tilde{\tilde{V}}$, the columns $\{\mathbf{h}_k : \mathbf{e}_{j,k} \neq 0\}$ together with $-\mathbf{H} \cdot \mathbf{e}_j + \tilde{\mathbf{u}}_j$ form an additive sharing of $\tilde{\mathbf{u}}_j$.

Since G is a random oracle, and since all seeds are independently sampled, the matrix $\mathbf{H}$ in $\tilde{V}$ and $\tilde{\tilde{V}}$ is uniform over $\mathbb{F}^{n \times m}$. Furthermore, any distinguisher $\mathcal{D}$ of V and $\hat{V}$ can be converted into a distinguisher of $\tilde{V}$ and $\tilde{\tilde{V}}$ in ROM, where the reduction receives either a sample from $\tilde{V}$ or $\tilde{\tilde{V}}$, replaces columns of $\mathbf{H}$ with independently sampled seeds and programs the random oracle such that it outputs $\mathbf{h}_j$ when given as input the corresponding seed, for all $j = 1, \ldots, m$, and then invokes the distinguisher $\mathcal{D}$.

Now, we can rewrite $\tilde{V}$ and $\tilde{\tilde{V}}$ as

$$\tilde{V} \equiv \left\{ \pi(\mathbf{H}, -\mathbf{H} \cdot \mathbf{E} + \mathbf{X}) \;\middle|\; \mathbf{H} \leftarrow_{\$} \mathbb{F}^{n \times m}, \mathbf{E} \leftarrow \mathsf{DisError}_{m,c,w}, \pi \leftarrow_{\$} \Pi \right\},$$

$$\tilde{\tilde{V}} \equiv \left\{ \pi(\mathbf{H}, -\mathbf{H} \cdot \mathbf{E} + \hat{\mathbf{X}}) \;\middle|\; \mathbf{H} \leftarrow_{\$} \mathbb{F}^{n \times m}, \mathbf{E} \leftarrow \mathsf{DisError}_{m,c,w}, \pi \leftarrow_{\$} \Pi \right\}.$$

It remains to show that $\tilde{V}$ and $\tilde{\tilde{V}}$ are indistinguishable. By Lemma 4, $\tilde{V}$ is indistinguishable from the following distribution

$$W = \left\{ \pi(\mathbf{H}, -\mathbf{Y} + \mathbf{X}) \;\middle|\; \mathbf{H} \leftarrow\!\$\, \mathbb{F}^{n\times m}, \mathbf{Y} \leftarrow\!\$\, \mathbb{F}^{n\times c} \right\}.$$

This can be seen as follows: extend $\tilde{V}$ by letting $\mathbf{Y} = \mathbf{H} \cdot \mathbf{E}$ and add to the tuple an extra element $\mathbf{z} = -\sum_j \mathbf{h}_j + \sum_i \mathbf{y}_i - \sum_i \mathbf{u}_i$; by Lemma 4, the tuple $(\mathbf{H}, -\mathbf{H}\cdot\mathbf{E}+\mathbf{X}, \mathbf{z})$ is indistinguishable from $(\mathbf{H}, -\mathbf{Y}+\mathbf{X}, -\sum_j \mathbf{h}_j + \sum_i \mathbf{y}_i - \sum_i \mathbf{u}_i)$ where $\mathbf{Y} \leftarrow\!\$\, \mathbb{F}^{m\times c}$ is uniformly sampled. Next, since $\mathbf{Y}$ is uniform in W, we see that W is equivalent to the following distribution $\tilde{W}$:

$$\tilde{W} = \left\{ \pi(\mathbf{H}, -\mathbf{Y} + \tilde{\mathbf{X}}) \;\middle|\; \mathbf{H} \leftarrow\!\$\, \mathbb{F}^{n\times m}, \mathbf{Y} \leftarrow\!\$\, \mathbb{F}^{n\times c} \right\}.$$

By a symmetrical argument, $\tilde{W}$ is indistinguishable from $\tilde{\tilde{V}}$. Now our proof is complete. □

As we have shown in Section 3, there is a reduction from SD to the MDSD problem. This allows us to rely on standard hardness assumption about decoding random linear codes.

Corollary 1. *Let $n, m, c, w > 0$ such that $m = c \cdot w + \lambda$ for some $\lambda > 0$. Assume $\mathsf{SD}_{n,i\cdot w+\lambda,w}$ is κ_i-bit computationally hard for all $i = 1,\dots,c$. Then the shuffle PIR protocol in Construction 3 is $(\Pi, c, 3c\sum_{i=1}^{c} \frac{T}{2^{\kappa_i}})$-secure against all adversaries with running time at most T.*

We now give the efficiency of our construction below.

Theorem 4 (Efficiency)**.** *Let $n, \lambda, w, c \in \mathbb{N}$ and let $m = c \cdot w + \lambda$. Let $\mathbb{F}_q$ be a field of size q, and let $\mathbf{x} \in \mathbb{F}_q^n$. Let κ be a computationally security parameter for PIR. Assuming there exists a PRG $\mathsf{G} : \{0,1\}^s \to \mathbb{F}_q$ with seed size s bits that can be modeled as random oracle. Let $\mathsf{MDSD}_{n,m,c,w}$ over $\mathbb{F}_q$ be an instance that is κ-bit computationally hard. Then there exists a PIR protocol in the shuffle model with κ bits of security, parameterized by $(\mathbb{F}_q, c, n, w, \lceil \lambda/c \rceil)$, such that for each query:*

- *Upload size is $n \log q + s(w + \lceil \lambda/c \rceil)$ bits;*
- *Download size is $(w + \lceil \lambda/c \rceil) \log q$ bits;*
- *Server computation per client query is dominated by $w + \lceil \lambda/c \rceil$ PRG invocations and $n(w + \lceil \lambda/c \rceil + 1)$ multiplications in $\mathbb{F}_q$.*

6 Evaluation

We now present our evaluation results of the secure aggregation and PIR protocols of Sections 4 and 5.

6.1 Setup and Parameters

We ran our experiments on a laptop with an Intel i7 1185G CPU and 32 GB RAM, and our experimental programs take advantage of the SIMD instructions such as AVX-512. All our experiments were executed using a single thread.

Aggregation. For our aggregation protocol, we consider input vectors over $\mathbb{F}_2$ as well as over larger finite fields $\mathbb{F}_{65537}$ of 16 bits and $\mathbb{F}_{4294967311}$ of 32 bits. Note that the complexity of ISD algorithms increases as the field becomes larger, and as observed in [LWYY22], ISD algorithms are the most efficient for our settings where the noise rate is low. We estimated the complexity of the DOOM attacks on $\mathsf{MDSD}_{n,m,c,w}$, based on ISD algorithms such as Prange [Pra62], Stern [Ste88], Dumer [Dum91], and Lee-Brickell [LB88], using the Syndrome Decoding Estimators [EB22] for $\mathbb{F}_2$ and [EVZB23] for larger fields. We then set parameters such that the protocol has at least 128 bits of computational security based on the conjectured hardness of $\mathsf{MDSD}_{n,m,c,w}$ problem as discussed in Section 3.3. We also set parameters achieving 100 bits of security with lower numbers of shares, based on the conjectured $\mathsf{MDSD}_{n,m,c,w}$ hardness. In addition, we consider conservative parameters in both the primitive shuffle model and the extended shuffle model, where the shuffler also generates dummy shares of the zero vector, such that the security is based on the reduction from SD problems.

PIR. We instantiate our PIR protocol from Definition 3 over $\mathbb{F}_2$. Let n be the number of records in the database, and let L be the bit length of each record. The database is arranged into a $n_1 \times n_2$ matrix, where $n = n_1 n_2$, and the query dimension n_1 is divided into blocks of size d (see Appendix A.2 for details). Recall that each query consists of $w + 1$ real and z dummy sub-queries, where all except one sub-queries are sent as AES-128 seeds. So, each query has size $n_1 + 256(w + z)n_1/d$ bits, and each response has size $nL(w + z + 1)/d$ bits assuming the shuffler simply forwards all responses back to the client. We rely on the hardness of $\mathsf{MDSD}_{d,m,c,w}$ problem where $m = c(w + z)$. For concrete database dimensions, we adjust n_1, n_2, and d to minimize the communication and computation costs.

We further optimize the number of database reads by batch-processing multiple queries—the trick is to simultaneously compute several inner-products with just one sequential pass over the database (see Appendix A.1). Through experimentation, we observe that the optimal batch size depends on the record size and the CPU cache size. For small records, we batch up to the sub-queries in each client query, effectively handling queries one at a time. For large records and when the number of shares is large, each query is processed in several batches.

To set protocol parameters, we rely on the conjectured hardness of $\mathsf{MDSD}_{d,m,c,w}$ problem. We use the Syndrome Decoding Estimator [EB22] to estimate the complexity of the DOOM attack on $\mathsf{MDSD}_{d,m,c,w}$ such that our PIR protocol has at least 128 bits of security. In particular, we consider ISD attacks with access to up to 2^{60} bits of memory, and as our parameter scales are large, we consider the logarithmic memory access cost model. In addition, we consider parameters achieving 100 bits of security, as well as more conservative parameters based on the reduction from SD problems as in Lemma 6.

6.2 Experimental Results

We now discuss the benchmarks for our aggregation and PIR protocols.

Aggregation. We implemented our secure aggregation construction, where all but one of the input shares were compressed to AES-128 seeds and the last share was sent in full length. We compared with the state of the art shuffle-based aggregation protocol with information-theoretic security of Balle et al. [BBGN19] applying the same compression optimization. Table 1 contains the benchmarks of our secure aggregation protocol achieve 128 bits of security. It shows that for most parameters settings of number of clients, input size and field, our protocol achieves close to the optimal expansion ratio 1, which measures the bandwidth used by the aggregation protocol per client over the plaintext input size. This results in 11–24× improvement over Balle et al. under different numbers of clients for inputs of length $2^{15}, 2^{20}$ over prime fields of 1, 16 and 32 bits. In Tables 2 we present parameters targeting a lower, 100-bit security level based on the conjectured MDSD hardness, where the expansion ratio is very close to 1 in all input settings.

In Table 3 we present the parameters and benchmark results for which there is a reduction from SD problem to MDSD. To set parameters according to Lemma 6, we consider $\lambda = w$ and hence we must have $2w > n$; otherwise SD problem over a $n \times 2w$ random matrix can be easily solved by Gaussian elimination. Due to this constraint, the share size is much larger than the parameters relying on the conjectured MDSD hardness. Nonetheless, we were still be able to choose parameters such that the communication cost is smaller than the information theoretical construction of [BBGN20].

In Table 4, we present results of our aggregation protocol in the extended shuffle model, where the shuffler generates $\lambda > 0$ shares of the zero vector. We set parameters based on the conservative hardness of MDSD problem. In particular, we chose a relatively large number λ of dummy shares, and as a result, we were able to set the share size lower than in the primitive shuffle model with heuristic parameters. Again, in the extended shuffle model, the communication rate is close to 1.

We also evaluated the hybrid constructions which combines a RLWE-based KMAHE scheme with the shuffle-model secure aggregation for short vectors. Table 5 contains the benchmark results. We compare it

Input $\times$ Field	Input size (KiB)	# Clients	Aggregation Protocol in Construction 1			Info-Theoretic Protocol [BBGN20]		
			# Shares	Upload size (KiB)	Expansion ratio	# Shares	Upload size (KiB)	Expansion ratio
$2^{15} \times \mathbb{F}_2$	4	100	405	10	2.57	6317	103	25.67
		1000	88	5	1.34	3856	64	16.06
		10000	37	5	1.14	2775	47	11.84
$2^{15} \times \mathbb{F}_{65537}$	64	100	410	70	1.10	100819	1639	25.61
		1000	77	65	1.02	61525	1025	16.02
		10000	33	65	1.01	44271	756	11.81
$2^{15} \times \mathbb{F}_{4294967311}$	128	100	410	134	1.05	201621	3278	25.61
		1000	77	129	1.01	123039	2050	16.02
		10000	33	129	1.00	88533	1511	11.81
$2^{20} \times \mathbb{F}_2$	128	100	10576	293	2.29	201621	3278	25.61
		1000	1124	146	1.14	123039	2050	16.02
		10000	169	131	1.02	88533	1511	11.81
$2^{20} \times \mathbb{F}_{65537}$	2048	100	10568	2213	1.08	3225684	52449	25.61
		1000	1116	2065	1.01	1968454	32805	16.02
		10000	159	2050	1.00	1416403	24179	11.81
$2^{20} \times \mathbb{F}_{4294967311}$	4096	100	10563	4261	1.04	6451352	104898	25.61
		1000	1110	4113	1.00	3936897	65610	16.02
		10000	153	4098	1.00	2832797	48358	11.81

Table 1: Parameters and communication costs of our aggregation protocol over $\mathbb{F}_q^n$ in the primitive shuffle model, for q with bit sizes 1, 16 and 32. We set parameters based on the conjectured MDSD hardness for 128-bit security. For each input vector $\mathbf{x} \in \mathbb{F}_q^n$, we generate $w + 1$ additive shares over $\mathbb{F}_q^n$, where the first w shares are compressed using AES-128 seeds. For comparison, we include the information-theoretic construction from [BBGN20]. In both protocols, the expansion ratio measures the communication overhead of the protocol over the input vector size.

Input $\times$ Field	Input size (KiB)	# Clients	Aggregation Protocol in Construction 1			Info-Theoretic Protocol [BBGN20]		
			# Shares	Upload size (KiB)	Expansion ratio	# Shares	Upload size (KiB)	Expansion ratio
$2^{15} \times \mathbb{F}_{65537}$	64	100	371	70	1.09	100819	1639	25.61
		1000	66	65	1.02	61525	1025	16.02
		10000	25	64	1.01	44271	756	11.81
$2^{15} \times \mathbb{F}_{4294967311}$	128	100	371	134	1.05	201621	3278	25.61
		1000	64	129	1.01	123039	2050	16.02
		10000	22	128	1.00	88533	1511	11.81
$2^{20} \times \mathbb{F}_{65537}$	2048	100	10528	2212	1.08	3225684	52449	25.61
		1000	1087	2065	1.01	1968454	32805	16.02
		10000	137	2050	1.00	1416403	24179	11.81
$2^{20} \times \mathbb{F}_{4294967311}$	4096	100	10528	4260	1.04	6451352	104898	25.61
		1000	1087	4113	1.00	3936897	65610	16.02
		10000	136	4098	1.00	2832797	48358	11.81

Table 2: Parameters and communication costs of our aggregation protocol over $\mathbb{F}_q^n$ in the primitive shuffle model, targeting 100-bit security, for q with bit sizes 16 and 32. Security is based on the conjectured MDSD hardness. For each input vector $\mathbf{x} \in \mathbb{F}_q^n$, we generate $w + 1$ additive shares over $\mathbb{F}_q^n$, where the first w shares are compressed using AES-128 seeds. For comparison, we include the information-theoretic construction from [BBGN20]. In both protocols, the expansion ratio measures the communication overhead of the protocol over the input vector size.

Input $\times$ Field	Input size (KiB)	# Clients	Aggregation Protocol in Construction 1			Info-Theoretic Protocol [BBGN20]		
			# Shares	Upload size (KiB)	Expansion ratio	# Shares	Upload size (KiB)	Expansion ratio
$2^{15} \times \mathbb{F}_{65537}$	64	100	16712	325	5.08	100819	1639	25.61
		1000	16712	325	5.08	61525	1025	16.02
		10000	16712	325	5.08	44271	756	11.81
$2^{20} \times \mathbb{F}_{4294967311}$	4096	100	524682	12294	3.00	6451352	104898	25.61
		1000	524682	12294	3.00	3936897	65610	16.02
		10000	524682	12294	3.00	2832797	48358	11.81

Table 3: Parameters and communication costs of our secure aggregation protocol over $\mathbb{F}_q^n$ in the primitive shuffle model, where the parameters are based on the reduction from the SD problem of Lemma 6, targeting 128-bit security level. For each input vector $\mathbf{x} \in \mathbb{F}^n$, we generate an additive share of size $w+1$ over $\mathbb{F}_q^n$, where the first w shares are sent using AES-128 seeds. For comparison, we include the information-theoretic construction from [BBGN20].

Input $\times$ Field	Input size (KiB)	# Clients	Our Aggregation Protocol					Info-Theoretical Protocol [BBGN20]		
			d	λ	Share size w	Upload size (KiB)	Expansion ratio	Share size w	Upload size (KiB)	Expansion ratio
$2^{20} \times \mathbb{F}_2$	128	1000	2^{16}	200000	101	153	1.20	123039	2050	16.02
		10000	2^{16}	2000000	94	151	1.18	88533	1511	11.81

Table 4: Parameters and communication costs of our secure aggregation protocol over $\mathbb{F}_q^n$ in the extended shuffle model. Security is based on the conservative hardness of MDSD problem by reducing from SD as in Lemma 6. For each input vector $\mathbf{x} \in \mathbb{F}_q^n$, we first split into blocks of length d, and then generate an additive share of size $w+1$ for each block over $\mathbb{F}_q^d$, where the first w shares are sent using AES-128 seeds. For comparison, we include the information theoretical construction in [BBGN20].

against running the shuffle construction directly on the long input vectors. This hybrid construction has slightly worse expansion ratio than our direct shuffle construction from MDSD. However, its aggregation phase runs much faster than the direct aggregation protocol. Such improvement is because aggregating short RLWE keys requires a much smaller number of shares per input, and hence the hybrid protocol spends much less on the relatively more expensive AES operations and replace with fast polynomial operations in RLWE. In our benchmarks on input length 2^{20} and 2^{23} over 16-bit and 32-bit prime fields, the RWLE-based hybrid protocol is 170–590$\times$ faster than the direct aggregation protocol.

PIR. We selected some typical database configurations: a baseline database of $2^{20} \times 256$ bytes, databases of one billion small records: $2^{30} \times 1$ bytes and $2^{30} \times 8$ bytes, and a databases of long records: $2^{18} \times 32$ KB. Table 6 shows our parameters as well as the communication cost in the primitive shuffle model, with 128-bit security under the MDSD conjecture. For all databases we set $z = 1$, i.e. each query contains one dummy sub-query. In Table 7, we present the corresponding experimental results on server running times. we consider our protocol in the offline-online mode: during the offline stage the client does not know the query indices yet, and it sends all except one sub-queries (as AES-128 seeds); during the online stage, the client then sends the last sub-query vector of full length.

For comparison, we considered two single-server PIR protocols in the standard model with minimal setup, namely HintlessPIR [LMRSW23] and Spiral [MW22], as well as SimplePIR [HHC+23] which has the state-of-the-art online time. Our shuffle PIR protocol achieves extremely fast online computation, taking 7–220 ms for the different databases we considered. Thus our online throughput improves the throughput of SimplePIR by 5–6$\times$. When we also process online sub-queries in batches, we see that the per-query throughput improves over SimplePIR by 20–25$\times$. Both Hintless and Spiral query times are slower than SimplePIR, 2–10$\times$ and 10–20$\times$ respectively. Our total computation time (online+offline) is always faster than Spiral, faster than HintlessPIR for most parameters, and comparable to SimplePIR for large numbers of users.

Both SimplePIR and Spiral require significant amount of setup communication between each client and

Input $\times$ Field	Input size (KiB)	# Clients	Agg. Protocol with RLWE-based Masks				Construction 1		
			RLWE degree N	# Shares	Upload size (KiB)	Agg. time (s)	Dimension d	# Shares	Agg. time (s)
$2^{20} \times \mathbb{F}_{65537}$	2048	1000	2^{10}	23	3460	0.75	2^{10}	29	296
		10000	2^{11}	16	3591	8.01	2^{11}	21	2107
		100000	2^{11}	11	3848	75.59	2^{11}	13	13022
$2^{20} \times \mathbb{F}_{4294967311}$	4096	1000	2^{11}	27	5515	0.98	2^{11}	30	356
		10000	2^{11}	16	5643	8.92	2^{11}	18	2132
		100000	2^{11}	11	5900	84.57	2^{11}	13	15413
$2^{23} \times \mathbb{F}_{65537}$	16384	1000	2^{10}	23	27652	5.04	2^{10}	30	2447
		10000	2^{11}	16	28679	51.42	2^{11}	22	17635
		100000	2^{11}	11	30728	516.79	2^{11}	14	113380
$2^{23} \times \mathbb{F}_{4294967311}$	32768	1000	2^{11}	27	44043	4.94	2^{11}	31	2947
		10000	2^{11}	16	45067	52.39	2^{11}	19	17987
		100000	2^{11}	11	47116	545.63	2^{11}	14	133203

Table 5: Parameters and benchmark results of the aggregation protocol of Construction 2 when using private-key RLWE encryption as an KAMHE scheme, where we rely on the conjectured MDSD hardness to set parameters for the underlying shuffle-model aggregation protocol for aggregating RLWE secrets. We also include the aggregation protocol of Construction 1 with MDSD-based security, where the input vector is divided into sub-vectors of dimension d; hence we run multiple instances of Construction 1 and we set parameters according to the conjectured MDSD security on dimension d. For both protocols, we target 128-bit security level, and we report the time for the server to aggregate all client inputs and recover the aggregation result. For the former protocol, server time includes expanding AES-128 seeds into shares of RLWE secrets, aggregating all such shares and then decrypting the aggregated RLWE ciphertexts. For the latter protocol, the server time includes the time to expand AES-128 seeds for all instances and aggregate shares from all clients.

the server. In the case of SimplePIR, such a preprocessing is database dependent and needs to be updated when the database changes. For the databases we considered in experiments, it takes from 45s to 35mins to preprocess the entire database. SimplePIR requires each client to store the preprocessed information, which results in significant setup bandwidth and storage requirement for the user device. Spiral requires that the server hold state per client resulting from the preprocessing, which is infeasible in many settings, especially if we want to achieve anonymity of queries. This makes it incompatible with the techniques for supporting variable database size records relying on shuffling. Our online communication cost is close to that of Spiral, which achieves the optimal upload/download communication by compressing the query (and this is the reason for its high computation cost). It is 3–13$\times$ less than the online communication of SimplePIR for different parameter settings. The total online and offline communication of our shuffle PIR is better than that for SimplePIR when each client makes a small number of queries; in the setting with a large number of clients, our total communication is better than SimplePIR when each client makes hundreds of queries. HintlessPIR has only online communication which is tens of times more than our online communication and, for a large number of clients, a higher total communication.

In Table 8, we present alternative parameters achieving 128 bits of security with different tradeoffs between the number of real sub-queries ($k = w + 1$) and the total number of dummy sub-queries ($\lambda = m - c \cdot w = c \cdot z$). Specifically, we consider $z = 10$ and 25 dummy sub-queries per client. As before, we group the database rows n_1 into blocks of d rows, and hence we rely on security of $\mathsf{MDSD}_{d,m,c,w}$ problem. Although w becomes smaller for the parameters in Table 8 when fixing d and c, to achieve the same 128-bit security level as in Table 6, the combined number of real and dummy sub-queries per client increases. This results in increases in the client request sizes and server computation costs.

In Table 9, we present parameters of our PIR protocol targeting a lower, 100-bit security level based on the conjectured MDSD hardness. We consider the same configurations of databases and clients as in Table 6, and we also set the number of dummy sub-queries per client to 1. We can lower the number of sub-queries to achieve a lower security level, resulting in smaller request sizes and lower server computation costs.

Database	# Clients	Our PIR Protocol in Construction 3					SimplePIR		HintlessPIR	Spiral	
		n_1	d	# Sub-queries	Offline Up/Down (KiB)	Online Up/Down (KiB)	Setup (KiB)	Up/Down (KiB)	Up/Down (KiB)	Setup (KiB)	Up/Down (KiB)
$2^{20} \times 256$ bytes	1K	2^{15}	2^{14}	65	2 / 1024	4 / 16	90112	66 / 21	442 / 788	7936	16 / 21
	10K	2^{15}	2^{14}	32	1 / 496	4 / 16					
	100K	2^{15}	2^{15}	23	1 / 176	4 / 8					
	1M	2^{16}	2^{16}	18	1 / 68	8 / 4					
$2^{30} \times 1$ bytes	1K	2^{17}	2^{14}	67	8 / 4224	16 / 64	180224	131 / 41	506 / 1576	7936	16 / 21
	10K	2^{17}	2^{14}	33	4 / 2048	16 / 64					
	100K	2^{18}	2^{16}	26	2 / 400	32 / 16					
	1M	2^{18}	2^{16}	19	1 / 289	32 / 16					
$2^{30} \times 8$ bytes	1K	2^{18}	2^{13}	51	25 / 51200	32 / 1024	509751	371 / 104	740 / 4470	10240	16 / 61
	10K	2^{18}	2^{14}	33	8 / 16384	32 / 512					
	100K	2^{18}	2^{16}	26	2 / 3200	32 / 128					
	1M	2^{18}	2^{16}	19	1 / 2304	32 / 128					
$2^{18} \times 32$ KB	1K	2^{18}	2^{14}	67	17 / 33792	32 / 512	180224	1024 / 288	1402 / 1824	10624	16 / 61
	10K	2^{18}	2^{14}	33	8 / 16384	32 / 512					
	100K	2^{18}	2^{15}	24	3 / 5888	32 / 256					
	1M	2^{18}	2^{17}	20	1 / 1216	32 / 64					

Table 6: Parameters (based on conjectured MDSD hardness targeting 128-bit security level) and communication costs of our PIR protocol in the primitive shuffle model. We arrange a database into a rectangle matrix of dimension $n_1 \times n_2$, and each PIR request contains just one dummy sub-query. Furthermore, we split the n_1 rows into blocks of length d, and thus we base security on the $\mathsf{MDSD}_{d,m,c,w}$ problem where $w+2$ is the number of sub-queries per PIR request, and $m = c(w+1)$. The offline part of a request consists of $w+1$ AES-128 seeds, and the online part is a length n_1 binary vector. When running our protocol without the offline phase, the communication cost is simply the sum of offline and online parts. For comparison, we include the communication costs of SimplePIR, HintlessPIR, and Spiral, where SimplePIR and Spiral require a one-time setup.

Database	# Clients	Our PIR Protocol in Construction 3				SimplePIR		HintlessPIR (ms)	Spiral (ms)
		Offline latency (ms)	Online latency (ms)	Online throughput (GiB/s)	Online throughput (batched, GiB/s)	Latency (ms)	Throughput (GiB/s)		
$2^{20} \times 256$ bytes	1K	132	7	36.64	125.05	45	5.56	575	794
	10K	63							
	100K	47							
	1M	33	9	28.64	129.59				
$2^{30} \times 1$ bytes	1K	581	30	32.95	117.13	183	5.46	1033	2576
	10K	270							
	100K	205	35	28.17	126.97				
	1M	139							
$2^{30} \times 8$ bytes	1K	5245	220	36.29	97.59	1442	5.55	3472	12875
	10K	2964							
	100K	2171							
	1M	1317							
$2^{18} \times 32$ KB	1K	8690	220	36.29	97.59	1483	5.39	2333	32050
	10K	3200							
	100K	1857							
	1M	1430							

Table 7: Server running times of our PIR protocol in the primitive shuffle model, for the conjectured parameters of Table 6. The server processes sub-queries in batches, where the batch size is at most the number of sub-queries of a single query. Offline latency measures the time taken to process a batch of offline sub-queries. Online latency measures the time to process a single online sub-query without batching. We also report the server running times of related works where their latency numbers measure the time for processing an individual query.

Database	# Clients	Our PIR w/ 10 Dummy Sub-Queries per Client						Our PIR w/ 25 Dummy Sub-Queries Per Client					
		n_1	d	w	Offline Up/Down (KiB)	Online Up/Down (KiB)	Latency Off/Online (ms)	n_1	d	w	Offline Up/Down (KiB)	Online Up/Down (KiB)	Latency Off/Online (ms)
$2^{20} \times 256$ bytes	1K	2^{15}	2^{14}	60	2 / 1120	4 / 16	143 / 7	2^{15}	2^{14}	55	3 / 1280	4 / 16	164 / 7
	10K	2^{15}	2^{14}	28	1 / 608	4 / 16	80 / 7	2^{15}	2^{14}	26	2 / 816	4 / 16	105 / 7
	100K	2^{15}	2^{15}	20	1 / 240	4 / 8	62 / 7	2^{15}	2^{15}	18	1 / 344	4 / 8	88 / 7
	1M	2^{16}	2^{16}	15	1 / 100	8 / 4	50 / 9	2^{16}	2^{16}	14	1 / 156	8 / 4	83 / 9
$2^{30} \times 1$ bytes	1K	2^{17}	2^{14}	61	9 / 4544	16 / 8	600 / 30	2^{17}	2^{14}	56	10 / 5184	16 / 8	700 / 30
	10K	2^{17}	2^{14}	29	5 / 2496	16 / 8	320 / 30	2^{17}	2^{14}	27	7 / 3328	16 / 8	438 / 30
	100K	2^{18}	2^{16}	23	2 / 528	32 / 18	285 / 35	2^{18}	2^{16}	21	3 / 736	32 / 18	394 / 35
	1M	2^{18}	2^{16}	16	2 / 416	32 / 16	207 / 35	2^{18}	2^{16}	15	3 / 640	32 / 16	341 / 35
$2^{30} \times 8$ bytes	1K	2^{18}	2^{13}	48	29 / 59392	32 / 1024	7463 / 220	2^{18}	2^{13}	44	35 / 70656	32 / 1024	9634 / 220
	10K	2^{18}	2^{14}	30	10 / 20480	32 / 512	3996 / 220	2^{18}	2^{14}	27	13 / 26624	32 / 512	5811 / 220
	100K	2^{18}	2^{16}	23	2 / 4224	32 / 128	3212 / 220	2^{18}	2^{16}	21	3 / 5888	32 / 128	4842 / 220
	1M	2^{18}	2^{16}	16	2 / 3328	32 / 128	2261 / 220	2^{18}	2^{16}	15	3 / 5120	32 / 128	4110 / 220
$2^{18} \times 32$ KB	1K	2^{18}	2^{14}	62	18 / 36864	32 / 512	10312 / 220	2^{18}	2^{14}	57	21 / 41984	32 / 512	11417 / 220
	10K	2^{18}	2^{14}	29	10 / 19968	32 / 512	4163 / 220	2^{18}	2^{14}	27	13 / 26624	32 / 512	5841 / 220
	100K	2^{18}	2^{15}	21	4 / 7936	32 / 256	3012 / 220	2^{18}	2^{15}	19	5 / 11264	32 / 256	4887 / 220
	1M	2^{18}	2^{17}	17	1 / 1728	32 / 64	2431 / 220	2^{18}	2^{17}	16	1 / 2624	32 / 64	4183 / 220

Table 8: Alternative parameters (based on the conjectured MDSD hardness targeting 128-bit security level) and communication costs of our PIR protocol in the primitive shuffle model, where each client contribute either $z = 10$ or $z = 25$ dummy sub-queries for a total of $\lambda = c \cdot z$ dummy sub-queries. We arrange a database into a rectangle matrix of dimension $n_1 \times n_2$, and we split the n_1 rows into blocks of length d; hence we base security on the $\mathsf{MDSD}_{d,m,c,w}$ problem where $w + z + 1$ is the number of sub-queries per client, and $m = cw + \lambda$. The offline part of a request consists of $w + z$ AES-128 seeds, and the online part is a length n_1 binary vector. When running our protocol without the offline phase, the communication cost is simply the sum of offline and online parts.

Database	# Clients	Our PIR Protocol w/ 100-bit Security					
		n_1	d	w	Offline Up/Down (KiB)	Online Up/Down (KiB)	Latency Off/Online (ms)
$2^{20} \times 256$ bytes	1K	2^{15}	2^{14}	55	2 / 896	4 / 16	113 / 7
	10K	2^{15}	2^{14}	25	1 / 416	4 / 16	53 / 7
	100K	2^{15}	2^{15}	17	1 / 144	4 / 8	39 / 7
	1M	2^{16}	2^{16}	13	1 / 56	8 / 4	28 / 7
$2^{30} \times 1$ bytes	1K	2^{17}	2^{14}	56	7 / 3648	16 / 64	483 / 30
	10K	2^{17}	2^{14}	26	3 / 1728	16 / 64	228 / 30
	100K	2^{18}	2^{16}	20	1 / 336	32 / 16	159 / 30
	1M	2^{18}	2^{16}	14	1 / 241	32 / 16	125 / 30
$2^{30} \times 8$ bytes	1K	2^{18}	2^{13}	44	23 / 46080	32 / 1024	4740 / 220
	10K	2^{18}	2^{14}	26	7 / 13824	32 / 512	2413 / 220
	100K	2^{18}	2^{16}	20	1 / 2688	32 / 128	1598 / 220
	1M	2^{18}	2^{16}	14	1 / 1920	32 / 128	1106 / 220
$2^{18} \times 32$ KB	1K	2^{18}	2^{14}	57	15 / 29696	32 / 512	7221 / 220
	10K	2^{18}	2^{14}	26	7 / 13824	32 / 512	2528 / 220
	100K	2^{18}	2^{15}	18	2 / 4864	32 / 256	1458 / 220
	1M	2^{18}	2^{17}	15	1 / 1024	32 / 64	1211 / 220

Table 9: Parameters (based on the conjectured MDSD hardness) and communication costs of our PIR protocol in the primitive shuffle model to achieve 100-bit security, where each client contribute one dummy sub-queries. We arrange a database into a rectangle matrix of dimension $n_1 \times n_2$, and we split the n_1 rows into blocks of length d; hence we base security on the $\mathsf{MDSD}_{d,m,c,w}$ problem where $w + 2$ is the number of sub-queries per client, and $m = c(w + 1)$. The offline part of a request consists of $w + 1$ AES-128 seeds, and the online part is a length n_1 binary vector. When running our protocol without the offline phase, the communication cost is simply the sum of offline and online parts.

Finally, in Table 10 we present benchmark results of our PIR protocol where the parameters are chosen according to the conservative security reduction from the SD problem. Since the reduction in Lemma 6 relies on the hardness of $\mathsf{SD}_{d,w+\lambda,w}$ where $\lambda = c \cdot z$ is the total number of dummy sub-queries, in order to set w relatively small, we must have λ sufficiently large. When there are $c = 10000$ clients, we can set $z \approx 30$; and when $c = 100000$, it is sufficient to set $z \approx 20$.

Database	# Clients	Conservative Parameters							
		n_1	d	w	z	Offline Up / Down (KiB)	Online Up / Down (KiB)	Offline latency (ms)	Online latency (ms)
$2^{20} \times 256$ bytes	10000	2^{14}	2^{14}	32	32	2 / 1024	2 / 16	204	7
	100000	2^{15}	2^{15}	25	24	2 / 392	4 / 8	101	7
$2^{30} \times 1$ bytes	10000	2^{17}	2^{14}	35	34	17 / 4416	16 / 64	570	30
	100000	2^{17}	2^{15}	25	28	7 / 1696	16 / 32	459	30
$2^{30} \times 8$ bytes	10000	2^{18}	2^{15}	52	25	77 / 19712	128 / 256	5455	224
	100000	2^{18}	2^{17}	41	17	14 / 3648	128 / 64	4049	224
$2^{18} \times 32$ KB	10000	2^{18}	2^{15}	51	25	19 / 19456	32 / 256	7552	224
	100000	2^{18}	2^{17}	39	17	4 / 3584	32 / 64	5492	224

Table 10: Communication and computation cost of our PIR protocol in the primitive shuffle model, where we estimate security using the conservative reduction from the SD problem.

7 PIR with Variable-Sized Records

In this section, we discuss an orthogonal use case of the split-and-mix paradigm which enables PIR on databases with variable-sized records.

7.1 Motivation and Goals

Real-world databases typically contain records in a wide variety of sizes—a platform like Youtube, for instance, houses both 30-second shorts along with 5 hour long documentaries. In many situations, knowing merely the size of the record can help to identify the record itself (or at least substantially reduce the possibility space). Consequently, for PIR protocols to be implemented for such databases in practice, it is necessary to also prevent leakage of the size of the record accessed.

Unfortunately, the existing literature on PIR typically considers databases where each record is of the same size—and for good reason. In particular, in the standard PIR setting, the only way to hide the size of the retrieved record is to *pad* each record to the length of the largest record (or some upper bound), or pack smaller records in the size of the largest record [GCM+16, ASA+21]. Even though this leaks nothing to the server about the size of record being queried (except that it is no bigger than the padding bound), a client who only wishes to access small records needs to pay the high communication cost of retrieving the largest record. This is highly sub-optimal especially because in practice, we expect most clients to query for average-sized records, while only a few clients query for very large ones.

Contrary to the standard model, we show that the shuffle model enables better communication-security tradeoffs compared to padding every record to the maximum length. The ideal security goal here is to leak only (an upper bound on) the total size of the records retrieved by *all* clients. In a bit more detail, consider multiple clients querying records of different sizes simultaneously. If we were in the standard model (every record is padded to an upper bound L), then the server sees homogeneous PIR queries all for size L; in other words, the queried sizes form a histogram $(L, L, \ldots, L)$. In the shuffle model, if the PIR queries were heterogeneous for different sizes (e.g., the server can distinguish between a PIR query for size-1 record and a PIR query for size-2 record), then the server would learn a histogram, e.g., $(2, L, 3, \ldots, 1)$, that indicates the queried sizes of all clients.

While revealing the histogram does not immediately lead to severe attacks, recent works [GSB+17, ZKP16] show that in certain applications the histogram reveals sensitive information. We wish to further hide the histogram up to only leaking the sum, or in a slight relaxation a very rough shape of the histogram.

A simple yet effective solution. We now describe a simple solution that achieves the above security goal without any padding. Suppose the database stores the concatenation of all the records without padding and assume that it contains n bits in total. Now, a client wanting to retrieve a record of length ℓ makes ℓ

PIR queries retrieving 1 bit each in the ℓ-length record. (More realistically, 1 bit can be replaced by a larger unit, depending on the maximal record size.) Observe that when the queries from all clients are mixed, the server only sees homogeneous queries, the only thing it learns is the total number of such queries—the total length of records queried by all clients. This simple solution has the communication per client proportional only to ℓ but not the maximum record size L (which can be much larger); in contrast the typical padding approach results in communication proportional to L even for clients querying small records.

Given this simple solution, we want to explore the following questions. First, can we achieve better per-client communication by retrieving more bits using one PIR query? In other words, the client splits record length ℓ into sub-lengths that may be larger than 1. When records are split into different sub-lengths, the PIR queries are no longer homogeneous (the server can tell between PIR queries for length-2 record and for length-1 record); so how much information of the queried lengths does the *multi-set* of the sub-lengths reveal? Note that the above simple solution leaks only the total length because the *multi-set* of the sub-lengths is identical given a particular sum of record lengths. The same is not true if the records are split in other ways (or not split at all), e.g., using the powers of 2 corresponding to the binary representation; there are many cases where a certain multi-set directly or indirectly reveals information about the original lengths configurations.

On leaking the total size of records. So far, we are able to hide the histogram that is generated under a given sum S and the number of clients C, with the restriction of length at most L. However, for certain regimes of (C, S, L), the record length of every client can be directly inferred from the sum. As a toy example, consider a database x with two entries: x_1 of size 1, and x_2 of size 1000, and two clients doing split-and-mix with splitting down to size 1. If the server knows the total size is 1001, then certainly there is one client querying x_1 and another querying x_2. Formally, given the information of number of clients C, maximum length L, and the total length S, the number of possible C-sized partitions of S with restricted to L is immediately known. Hiding such information is beyond the scope of cryptography, analogous to hiding the message size in encryption. If leaking this information is concerning in applications, one can increase the possibility space of the partitions by increasing C, or increasing S with randomized padding.

Here we consider the following quesiton: Given C, S, L, how can we randomly split recoreds so that, for any two query histograms in the partition space, the distributions of multi-sets of sub-lengths are statistically close? Note that this is a much stronger notion than differential privacy. Our next task is to formally define this problem and quantify the distinguishing advantage.

7.2 Problem Definition

We will define a probabilistic leakage function to capture how much information is leaked by the multi-set generated from a given query-length configuration. In other words, the server should be able simulate its view in the actual protocol given the output of the leakage function.

As a warm-up, let us first consider the following examples. In the simple case where each record is split into 1 bit, the leakage is only the sum of all the query-length. However, once we split a record into sub-records larger than 1 bit (to reduce the number of PIR queries), the corresponding multi-set reveals more information than just the sum: a server receiving a multi-set $(2, 1, 1, \ldots, 1)$ will immediately know that not all the clients are querying records of length 1, while this fact cannot be learned using the solution of splitting records down to size 1. On the positive side though, this intuitively leaks much less information than directly revealing the original query-length.

Definition 12 formally describes our security goal for record-length splitting using the language of leakage function. In the next section we discuss how record-splitting can be compiled with any single-server PIR protocols in a black box way to (partially) hide the query-length information.

Definition 12 (Leakage function for record splitting). *Let $L \in \mathbb{N}^+$ (the total length), and let $U = \{1, \ldots, L\}$. Define probabilistic leakage function $\mathcal{L}_U$ that takes in a multi-set $H \subset U$ and outputs a distribution over U.*

For any two equal-sized multi-sets $H, H' \subset U$ with the same sum of elements, we say that the leakage function $\mathcal{L}_U$ is ϵ-secure if

$$\mathsf{s.d.}(\mathcal{L}_U(H), \mathcal{L}_U(H')) \le \epsilon.$$

For correctness of PIR, the output of the leakage function must be restricted by the input H: we require every possible output of $\mathcal{L}_U$ must be partitioned into subsets, where there exists a surjective mapping from the subsets to H, such that the sum of elements in each subset equals to an element in H. For example, if $H = \{1, 3, 4\}$, then $\mathcal{L}_U(H) = \{1, 1, 2, 2, 2\}$ is a valid output because the partition $\{\{1\}, \{1, 2\}, \{2, 2\}\}$ gives H, while $\mathcal{L}_U(H) = \{2, 3, 3\}$ is not valid because none of its partitions gives H.

We next give the definition of the length-splitting algorithm run by each client. The corresponding multi-set of sub-lengths is just generated by multiple clients running the splitting algorithm.

Definition 13 (Splitting algorithm)**.** *A splitting algorithm* $\mathsf{Split}(\ell; L)$*, parameterized by a maximum record length* L*, takes in a record length* $\ell \in [L]$*, and outputs a tuple* $(b_j)_{j\in[h]}$ *of values where* $h = h(\ell, L)$ *is a (possibly randomized) function of* ℓ, L*. We define the following properties of* Split*.*

Correctness. *For every* $\ell \in [L]$*, letting* $(b_1, \ldots, b_h) \leftarrow\!\$\, \mathsf{Split}(\ell; L)$*, it holds that* $\sum_{j=1}^{h} b_j \geq \ell$*.*

Efficiency. *We want to capture the efficiency of algorithm* Split *by measuring the portion of bits the client wants to retrieve in all the bits the client actually retrieves. We say that* Split *has bit efficiency* μ*, if for every* $L \in \mathbb{N}$*, and all* $\ell \in [L]$*, let* $(w_1, \ldots, w_h) \leftarrow\!\$\, \mathsf{Split}(\ell; L)$*, it holds that* $(\sum_{j=1}^{h} w_j)/\ell \leq 1/\mu$*. Note that* $1/L \leq \mu \leq 1$*. When* $\mu = 1$*, there is no waste on bits retrieved by the client; when* $\mu = 1/L$*, it is the same as padding.*

Another efficiency metric we consider is message complexity, which we measure as $\mathbb{E}[h(\ell; L)]$*, where the randomness comes from the* Split *algorithm.*

Now consider $C > 1$ clients running Split algorithm, where their inputs are $\ell_1, \ldots, \ell_C \in [L]$. Let H be the multi-set $\{\ell_1, \ldots, \ell_C\}$, and define $\mathcal{L}_{[L]}(H)$ to be the joint outputs of $\mathsf{Split}(\ell_i; L)$ for all $i \in [C]$; now, given any algorithm Split, one can compute the leakage ϵ of $\mathcal{L}_{[L]}$. Furthermore, if we know (an upper bound of) the distance between H and H' (say L_1 distance between the histogram of H and H'), then one can write ϵ as a function of the distance. Intuitively, if the distance of two record-length histograms increases, then the corresponding ϵ will also increase. For example, consider the following three cases:

- L clients all querying length 1,
- Only one client querying length L,
- $L - 2$ clients querying length 1, one client querying length 2.

In general, the ϵ we can achieve between case 1 and case 2 is larger than that between case 1 and case 3. We give a concrete instantiation of Split algorithm and analyze the leakage in Section 7.3.

7.3 Construction

High-level idea. We now give the high-level idea of our Split algorithm that splits a given length ℓ to roughly $\log \ell$ sub-lengths. Assuming L is power of 2 w.l.o.g. (If not, we can increase L to the nearest power of 2.) Each client builds $1 + \log L$ levels (indexed from 0), where the j^{th} level represents length 2^j for $j = 0, \ldots, \log L$. The Split algorithm can be viewed as placing balls at the $1 + \log L$ levels. The client starts with an initial configuration where ℓ is split into powers of 2; this corresponds to placing exactly one ball at the corresponding levels; and the rest levels are empty. For instance, if the record length is 40, the client splits it to 32 and 8, which corresponds to placing one ball at level 5 and one ball at level 3. Then starting from the highest level (the $(\log L)$-th level), for each ball at the current level j, with probability 1/2 the client split it into two balls and place them at level $j - 1$; with probability 1/2 the client leaves this ball at the current level. In other words, the client recursively splits the balls until it reaches the level indexed by 0. The eventual balls-to-bins configuration is the output of Split algorithm.

In our final construction, there is a parameter ρ that specifies the depth of the levels until which all balls are split with probability 1 (i.e., fully split). We can view ρ as a trade-off factor between efficiency and leakage: when $\rho = \log L$, we have $\epsilon = 0$ but the message complexity equals the length of the record; when $\rho = 0$, the record is not split at all so the message complexity is 1, but $\epsilon = 1$. The construction is described in Construction 4.

Construction 4 (Recursive splitting)**.** *Let L be the maximum length of database records (w.l.o.g. assume L is power of 2). Let ρ be a parameter indicating the depth of full-split levels where $\rho \in [\log L]$.* $\mathsf{Split}(\ell, L) \to (b_1, \ldots, b_h)$ *is defined as follows.*

1. *Let $B_0, \ldots, B_{\log L} \in \{0,1\}$ be the binary decomposition of ℓ, i.e., $\ell = \sum_{j=0}^{\log L} B_j \cdot 2^j$.*
2. *For $j = \log L, \ldots, \log L - \rho + 1$:* *// fully split*
   (a) *Set $B_{j-1} \leftarrow 2B_j$.*
3. *For $j = \log L - \rho, \ldots, 1$:* *// for each sub-record, split with probability 1/2*
   (a) *Set $S_j := 0$.*
   (b) *Repeat B_j times the following:*
   Sample $r \leftarrow_\$ \{0,1\}$, if $r = 0$, then set $S_j \leftarrow S_j + 1$.
   (c) *Set $B_j \leftarrow B_j - S_j$ and set $B_{j-1} \leftarrow B_{j-1} + 2S_j$.*
4. *Output $(\underbrace{2^0, \ldots, 2^0}_{B_0}, \underbrace{2^1, \ldots, 2^1}_{B_1}, \ldots, \underbrace{2^{\log L}, \ldots, 2^{\log L}}_{B_{\log L}})$.*

Compilation with PIR. So far we only describe how records are split into sub-records, now we describe how the client retrieve from the server the sub-records privately. As before, all the records are concatenated together to an n-bit database (no padding). Then the server (virtually) prepares $\log n$ databases, each of n bits; the j-th database is partitioned into entries of size of 2^j for $j = 0, \ldots, \log n$. The server also sets up a helper database that stores, for each item, the range of indices in the original database (i.e., concatenation) that the record resides. For example, the 10-th entry in the helper database stores the information: the 10-th record consists of the 100 bits from index 200 to 300 in the n-bit database.

To retrieve an item, the client first makes a PIR query to the helper database to obtain the range of the indices to which the record corresponds. Suppose the record is of length ℓ. Now the client runs the algorithm Split that takes in ℓ and splits it to h sub-lengths. Then client retrieves h sub-records from the $\log n$ logical databases; these h sub-records should cover the entire range that the client wants to query. The client can label the query with the level number so that the server knows how to partition the database when answering queries.

A concrete example is as follows. Suppose the client wants to retrieve the record that resides in range $[11, 16] \subset [n]$, and the Split algorithm outputs lengths 2, 4. The client will issue a query of the form ($\mathsf{PIR.Query}(6)$, "size 2") and ($\mathsf{PIR.Query}(4)$, "size 4"). The former tuple queries the 6th size-2 sub-record, i.e., $[11, 12]$, and the latter tuple is for the 4th size-4 sub-record, i.e., $[13, 16]$. The server answers to the former tuple by viewing the n-bit database as 2 bits per entry, and answers to the latter tuple by viewing the database as 4 bits per entry.

Concrete security. We next give data points to show how our Split algorithm achieves a better balance between query communication cost and leakage of record sizes, compared to the naive approach (splitting records down to 1 bit) or the non-private approach. Specifically, we focus on regime where $C = L$, and choose $L = 2^{20}$. This choice captures the parameters in applications; for example, the maximum size of a Google doc is 50 MB and if a unit is 32 bytes, then we have L being roughly 2^{20}. If C is smaller than L, we can increase the unit size. Since in real-world settings most clients will query records close to the average length, we further assume that, for any two query histogram with the same total length, the C clients query records of at most $\sqrt{C}$ *distinct* lengths.

Figure 1 shows the distinguishing advantage ϵ and efficiency for different solutions. Note that even $\varepsilon = 0.13$ gives a significant level of uncertainty, leaking a very rough shape of the histogram; in contrast, the non-private solution has $\epsilon = 1$ where the server learns the full shape of the histogram. In terms of efficiency, we use two metrics: the total number of messages that all clients send to the server, and the total number of bits the clients retrieve from the server. The latter is essentially the size of all PIR answers. Our recursive

schemes	sum S	total retrieved	total #msg	ε
trivial pad	N/A	137.44 GB	$1.05 \cdot 10^6$	0
no pad, no split	2^{25}	4.19 MB	$1.05 \cdot 10^6$	1
	2^{30}	134.22 MB		
	2^{35}	4.29 GB		
no pad, split to 1	2^{25}	4.19 MB	$3.36 \cdot 10^7$	0
	2^{30}	134.22 MB	$1.07 \cdot 10^9$	
	2^{35}	4.29 GB	$3.44 \cdot 10^{10}$	
no pad, recur. split	2^{25}	4.19 MB	$< 1.05 \cdot 10^6$	< 1.00
			$< 4.19 \cdot 10^6$	< 0.50
	2^{30}	134.22 MB	$< 3.36 \cdot 10^7$	< 0.18
			$< 1.34 \cdot 10^8$	< 0.09
	2^{35}	4.29 GB	$< 1.07 \cdot 10^9$	< 0.03
			$< 4.29 \cdot 10^9$	< 0.02

Figure 1: Efficiency comparison between our construction and baselines. We assume maximum record size is $L = 2^{20}$ and there are $C = 2^{20}$ clients who query records of total length S, where we additionally assume there are most $\sqrt{C}$ distinct lengths. The parameter ϵ measures how much the server learns about the shape of the histogram. Our recursive splitting algorithm (Construction 4) offers a trade-off between ϵ and message complexity. Here we provide two data points per S.

splitting approach has the optimal number of retrieved bits (hence has bit efficiency 1); this matches the download size of the non-private solution. We also reduce the total number of messages by $32\times$ with leaking only a very rough shape of the histogram ($\epsilon = 0.13$). In this case, a client querying record of length, e.g., 16, will only need to send 4 queries in expectation instead of 16 queries. Also, when S is larger, we get better trade-off between ϵ and the message complexity.

Finally, an interesting future direction is to get better parameters with *randomized padding* and splitting; note that our Split algorithm above does not pad any record. For example, one can pad the record length to $L = 2^k$ (up to the maximum length) with probability 2^{-k}. We leave the analysis to future work.

Proof and analysis. To calculate the ε for our Split algorithm in Construction 4, one can rely on the two lemmas below.

Lemma 7 (Smoothing Lemma). *Given k, r such that $0 < r < \sqrt{k}$, consider placing k balls at level j v.s. placing $k-r$ balls at level j and place $2r$ balls at level $j-1$. Let $\mathcal{D}_0$ be the distribution of balls after randomized splitting starting with the first configuration; similarly $\mathcal{D}_r$ for the second configuration. Then there exists a constant $c > 0$ such that $\mathsf{s.d.}(\mathcal{D}_0, \mathcal{D}_r) \leq c \cdot r/\sqrt{k}$.*

Proof. We start with the base case where $r = 1$. Observe that we can bound the statistical distance simply by considering the splitting from level j to level $j-1$. Specifically, we can sum up, for each i, the difference in probabilities between the two scenarios that level $j-1$ contains exactly i balls after the splitting of balls in level j. We get:

$$\begin{aligned}\mathsf{s.d.}(\mathcal{D}_0, \mathcal{D}_1) \leq\ & \binom{k}{0}(\frac{1}{2})^k + \sum_{i=1}^{k} \left| \binom{k}{i}(\frac{1}{2})^k - \binom{k-1}{i-1}(\frac{1}{2})^{k-1} \right| \\ \leq\ & 2 \cdot \binom{k}{k/2}(\frac{1}{2})^k = \Theta(\frac{1}{\sqrt{k}}) \qquad \text{. . . by Stirling's approximation.}\end{aligned}$$

The same idea applies to any $r < \sqrt{k}$. We have

$$\begin{aligned}
\mathsf{s.d.}(\mathcal{D}_0, \mathcal{D}_r) &\leq (\frac{1}{2})^k \cdot (\binom{k}{0} + \ldots + \binom{k}{r-1}) + \sum_{i=r}^{k} \left| \binom{k}{i} (\frac{1}{2})^k - \binom{k-r}{i-r} (\frac{1}{2})^{k-r} \right| \\
&\leq 2r \cdot \binom{k}{k/2} (\frac{1}{2})^k \\
&\leq c \cdot \frac{2r}{\sqrt{k}} \qquad \text{for some constant } c > 0.
\end{aligned}$$

□

Lemma 8 (Bounding the differences at every level)**.** *Consider any two configurations with equal sum. Then, at every level, the difference in the number of balls between the two configurations is at most C.*

Proof. Let the sum be S and the number of clients be C. Now consider the balls-to-bins configuration of all the clients.

The maximum number of balls above the k-th level is $C(2^{k-1} + \ldots 2^0) = C(2^k - 1)$, as for each client, each level has at most one ball. So in this case, the number of balls at the k-th level is $\frac{S - C(2^k-1)}{2^k} = \frac{S}{2^k} - C + \frac{C}{2^k}$. In the other case, the minimum number of balls below the k-th level is 0; then in this case the k-th level has the number of balls $S/2^k$. Therefore, the difference is $C - \frac{C}{2^k}$, which is smaller than C. □

8 Conclusion and Open Questions

We presented a new computational analysis for the "split-and-mix" approach that shuffles additive shares of multiple inputs in a way that reveals only the sum of the inputs. In the process, we introduced the new computational MDSD problem and provided security analysis. We showed how to use the split-and-mix technique to construct new secure aggregation and PIR protocols in the shuffle model, which improve on the state of the art. In particular, our secure aggregation protocol improves the communication rate by orders of magnitudes over the near optimal information-theoretic analysis, and our PIR protocol achieves best online performance overall, and is also more efficient than most existing protocols without requiring an expensive setup. We leave it as an open problem to further study the MDSD problem and to explore its applications.

Another open problem is to obtain a more complete understanding of the achievable trade-offs between efficiency and security for PIR with variable-size records in the shuffle model. This includes solutions that may slightly increase the total size of the records retrieved by the clients, combining splitting and padding.

Acknowledgement

This project was supported by a Google faculty research award. Y. Ishai was supported by ERC grant NTSC (742754), BSF grant 2022370, ISF grant 2774/20, and ISF-NSFC grant 3127/23.

References

[AFS03] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A fast provably secure cryptographic hash function. Cryptology ePrint Archive, Paper 2003/230, 2003. `https://eprint.iacr.org/2003/230`.

[AGJ$^+$22] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In *SCN*, pages 516–539, 2022.

[AIK07] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 92–110, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

[AIVG22] Kinan Dak Albab, Rawane Issa, Mayank Varia, and Kalman Graffi. Batched differentially private information retrieval. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 3327–3344. USENIX Association, 2022.

[App] Apple. icloud private relay overview. `https://www.apple.com/icloud/docs/iCloud_Private_Relay_Overview_Dec2021.pdf`.

[ASA$^+$21] Ishtiyaque Ahmad, Laboni Sarker, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Coeus: A system for oblivious document ranking and retrieval. In *SOSP*, pages 672–690. ACM, 2021.

[BBCG$^+$21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.

[BBG$^+$20] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In *CCS*, pages 1253–1269. ACM, 2020.

[BBGN19] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. The privacy blanket of the shuffle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 638–667, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[BBGN20] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. Private summation in the multi-message shuffle model. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 657–676, Virtual Event, USA, November 9–13, 2020. ACM Press.

[BCG$^+$20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st Annual Symposium on Foundations of Computer Science*, pages 1069–1080, Durham, NC, USA, November 16–19, 2020. IEEE Computer Society Press.

[BEM$^+$17] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, 2017.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

[BGL$^+$23] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. ACORN: Input validation for secure aggregation. In *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[BIK$^+$17] Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, pages 1175–1191. ACM, 2017.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 55–73, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012.

[BM18] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In *PQCrypto*, volume 10786 of *Lecture Notes in Computer Science*, pages 25–46. Springer, 2018.

[BMvT78] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3), 1978.

[BØ23] Pierre Briaud and Morten Øygarden. A new algebraic approach to the regular syndrome decoding problem and implications for PCG constructions. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 391–422, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

[CGB17] Henry Corrigan-Gibbs and Dan Boneh. Prio: private, robust, and scalable computation of aggregate statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, 2017.

[CGKS95] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, 1995.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2), 1981.

[Cha88] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptol.*, 1(1), 1988.

[Che21] Albert Cheu. Differential privacy in the shuffle model: A survey of separations. *CoRR*, abs/2107.11839, 2021.

[Chr] Google Chrome. Ip protection. `https://github.com/GoogleChrome/ip-protection`.

[CL05] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 169–187, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[Clo] Cloudflare. Privacy gateway. `https://blog.cloudflare.com/building-privacy-into-internet-standards-and-how-to-make-your-app-more-private-today`.

[CSU$^+$19] Albert Cheu, Adam D. Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 375–403, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[CZ22] Albert Cheu and Maxim Zhilyaev. Differentially private histograms in the shuffle model from fake users. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 440–457. IEEE, 2022.

[DK99] Yevgeniy Dodis and Sanjeev Khanna. Space time tradeoffs for graph properties. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 291–300. Springer, 1999.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04. USENIX Association, 2004.

[DPC23] Alex Davidson, Gonçalo Pestana, and Sofía Celi. Frodopir: Simple, scalable, single-server private information retrieval. *Proc. Priv. Enhancing Technol.*, 2023(1):365–383, 2023.

[Dum91] Ilya Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52. Moscow, 1991.

[EB22] Andre Esser and Emanuele Bellini. Syndrome decoding estimator. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 112–141. Springer, 2022.

[EFM$^+$19] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: from local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, 2019.

[EVZB23] Andre Esser, Javier Verbel, Floyd Zweydinger, and Emanuele Bellini. `CryptographicEstimators`: a software library for cryptographic hardness estimation. Cryptology ePrint Archive, Paper 2023/589, 2023. `https://eprint.iacr.org/2023/589`.

[FM02] Michael J. Freedman and Robert Morris. Tarzan: a peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, 2002.

[GCM$^+$16] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath T. V. Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with popcorn. In *NSDI*, pages 91–107. USENIX Association, 2016.

[GGM98] Yael Gertner, Shafi Goldwasser, and Tal Malkin. A random server model for private information retrieval or how to achieve information theoretic PIR avoiding database replication. In *RANDOM*, volume 1518 of *Lecture Notes in Computer Science*, pages 200–217. Springer, 1998.

[GJ04] Philippe Golle and Ari Juels. Dining cryptographers revisited. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 456–473, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[GMPV20] Badih Ghazi, Pasin Manurangsi, Rasmus Pagh, and Ameya Velingker. Private aggregation from fewer anonymous messages. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 798–827, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[GRS99] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Commun. ACM*, 42(2), 1999.

[GSB$^+$17] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 655–672. IEEE Computer Society, 2017.

[GT96] Ceki Gulcu and Gene Tsudik. Mixing email with babel. In *Proceedings of the 1996 Symposium on Network and Distributed System Security (SNDSS '96)*, SNDSS '96, 1996.

[HHC$^+$23] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, 2023.

[HIKR23] Shai Halevi, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. Additive randomized encodings and their applications. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 203–235. Springer, 2023.

[HOSS22] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. *Journal of Cryptology*, 35(2):13, April 2022.

[IKLM] Yuval Ishai, Mahimna Kelkar, Daniel Lee, and Yiping Ma. Single-server private information retrieval in the shuffle model. Personal communication, `https://nycryptoday.wordpress.com/2022/09/`.

[IKOS06] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from anonymity. In *47th Annual Symposium on Foundations of Computer Science*, pages 239–248, Berkeley, CA, USA, October 21–24, 2006. IEEE Computer Society Press.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science*, pages 364–373, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[KSS10] Jonathan Katz, Ji Sun Shin, and Adam Smith. Parallel and concurrent security of the HB and HB+ protocols. *Journal of Cryptology*, 23(3):402–421, July 2010.

[LB88] Pil Joong Lee and Ernest F. Brickell. An observation on the security of mceliece's public-key cryptosystem. In *Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 1988.

[LLPT23] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. LERNA: secure single-server aggregation via key-homomorphic masking. In *ASIACRYPT (1)*, volume 14438 of *Lecture Notes in Computer Science*, pages 302–334. Springer, 2023.

[LMRSW23] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz-Wu. Hintless single-server private information retrieval. Cryptology ePrint Archive, Paper 2023/1733, 2023. `https://eprint.iacr.org/2023/1733`.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In *STOC*, pages 595–608. ACM, 2023.

[LWYY22] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. The hardness of LPN over any integer ring and field for PCG applications. *IACR Cryptol. ePrint Arch.*, page 712, 2022.

[McE78] Robert J. McEliece. A public key cryptosystem based on algebraic coding theory. In *The Deep Space Network Progress Report*, 1978.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 465–484, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

[MMR$^+$17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, 2017.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228. Springer, 2015.

[MW18] Daniele Micciancio and Michael Walter. On the bit security of cryptographic primitives. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 3–28, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[MW22] Samir Jordan Menon and David J. Wu. Spiral: Fast, high-rate single-server pir via fhe composition. In *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[MWA$^+$23] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 477–496. IEEE, 2023.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, 2001.

[Pie12] Krzysztof Pietrzak. Cryptography from learning parity with noise. In *SOFSEM*, volume 7147 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2012.

[PPY18] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Private stateful information retrieval. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1002–1019, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory*, 8(5):5–9, 1962.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[Reg06] Oded Regev. Lattice-based cryptography (invited talk). In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 131–141, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany.

[Sen11] Nicolas Sendrier. Decoding one out of many. In *PQCrypto*, volume 7071 of *Lecture Notes in Computer Science*, pages 51–67. Springer, 2011.

[Ste88] Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.

[TDG16] Raphael R. Toledo, George Danezis, and Ian Goldberg. Lower-cost $\in$-private information retrieval. *Proc. Priv. Enhancing Technol.*, 2016(4):184–201, 2016.

[TS16] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In *PQCrypto*, volume 9606 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2016.

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 707–720. USENIX Association, 2016.

[ZMA22] Ke Zhong, Yiping Ma, and Sebastian Angel. Ibex: Privacy-preserving ad conversion tracking and bidding. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 3223–3237, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[ZPSZ23] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server pir with sublinear server computation. Cryptology ePrint Archive, Paper 2023/452, 2023. `https://eprint.iacr.org/2023/452`.

A Optimizations to the PIR Protocol

We now introduce several optimizations for improving the concrete efficiency of our base construction from Section 5.2; collectively, these result in a highly practical protocol.

A.1 Stream Processing for Batched Queries

The first optimization is a system-level one which can process multiple queries with just one sequential pass over the database. For this optimization we assume the PIR protocol is instantiated over $\mathbb{F}$ with characteristic 2.

We illustrate the key idea via the following scenario. Assume that the database is large (can only be stored in e.g., disk) and each database entry is much larger than one bit (e.g., KB), and suppose the client's

Batch size	1	2	4	8	16	32
128 bytes record	4.68	6.96	10.78	14.03	16.48	16.51
256 bytes record	8.14	14.09	23.11	29.76	33.27	35.97

Figure 2: Throughput (GB/s) of batched XOR computation on random records of size 128 and 256 bytes.

query can be fit into memory or cache. Consider the server computation for processing a single client's query for database of size n: the client splits its selection vector (a unit vector of length n) into w shares (each is a binary vector of length n). Recall that the PIR response to each share is the inner product between the database vector and the sharing vector; alternatively, it is the XOR of those database entries that the corresponding selection bit is 1.

To answer all the w shares from a client, naively the server needs to read the database w times, each for computing one inner product. However, reading the database from disk is costly. We instead let the server *sequentially* read the database *once* to process a batch of w sharing vectors to get w inner products. Concretely, the database is divided into equally-sized rows (where w many rows can be fit into the cache); the server then sequentially reads each row of the database, and computes the inner product between the row and a batch of w sub-vectors.

This optimization essentially means the server can process one query consisting of w share, or even multiple queries, by making a single pass over the database. We can compare the concrete efficiency with the recent state-of-the-art PIR schemes (HintlessPIR [LMRSW23] SimplePIR [HHC+23]) in an analytical way and in an experimental way. Note that both the recent schemes have linear-time server preprocessing to generate hints that needs to be re-computed once the database changes. In contrast, our protocol does not have such hints and the server can update the database without extra cost.

Analytical comparison. Assuming atomic operations in CPU are 32-bit integer addition/multiplication, and 32-bit XOR. Note that different atomic operations may have different run time. Suppose the database consists of n elements, each of size 32ℓ bits. To answer one query, the server in SimplePIR computes $n\ell$ 32-bit integer multiplications and $n\ell$ 32-bit integer additions. The computation for HintlessPIR is more expensive. In our protocol, on the same database, the server computes on average $(n/2) \cdot \ell w$ 32-bit XORs, where w is the number of shares per client.

Experimental results. Table 2 shows the throughput of XORing random records in batches of difference sizes. When the batch size increases from 1 to 32, the throughput increases by roughly a factor of 4. The throughput reaches a plateau at batch size 16, so in the PIR protocol we can let the server batch process 16 sharing vectors.

A.2 Splitting the Database

Recall in Section 5.2, we give our base construction for PIR in the shuffle model. This does not directly give us practical PIR: concretely, to guarantee over 100 bits of security for a database of size 2^{10}, if we split the query into 10 shares, we need 2^{20} dummies. Next we present an optimization that reduces the total number of dummies without increasing server or client computation.

The idea of this optimization comes from a simple observation on the standard syndrome decoding problem (Section 2): given (m, n, τ) for an SD problem, and fix m and τ, the smaller n is, the harder the SD problem is. In the corresponding PIR setting, this observation implies that when the database size decreases, the required number of dummies also decreases. So our technique is to split the database into smaller equal-sized sub-databases, and have the client also split the selection vector (a unit vector) into the same equal-sized sub-vectors. For a target security level, the database-splitting approach requires less number of dummies compared to the base solution.

Concrete security. On one hand, splitting the database into smaller sub-databases will make the corresponding MDSD instance harder; on the other hand, it will generate multiple MDSD instances and we need to guarantee no adversary can break any of these MDSD instances (can be analyzed by union bound).

#splits L	512	1024	2048
dimension of sub-databases n/L	2048	1024	512
#real shares per client $(w+1)$	25	25	25
#dummy shares per client	4	3	2
#total shares $(m+c)$	2893216	2762144	2631073
bit security of $\mathsf{MDSD}_{n/L,m,w}$	150	157	157
PIR security w/ database splitting	132	137	135

Figure 3: Examples of splitting database to reduce dummies. Suppose the database size $n = 2^{20}$ and there are c =100K clients. The total number of shares include real and dummy sub-queries.

Table 3 shows example parameters when we split the database into sub-databases of size 2048, 1024 and 512. For example, if we use sub-databases of size 1024, then each client needs to add 3 dummies for a query. The bit security of the corresponding MDSD problem (a single instance) is 157, but combining 1024 instances together gives us the overall bit security 137. If the database is split into smaller dimension such as 512, we get same level of security with fewer dummies per MDSD instance.

Theorem 5 (Asymptotic reduced number of dummies)**.** *Given database size $n \in \mathbb{N}$ and security parameter κ. Given construction 3, assume the underlying assumption is $\mathsf{MDSD}_{n,cw+\lambda,c,w}$ with κ bits of security. Applying the splitting optimization with parameter L, then under the same number of shares w per query and same level of security κ, we can reduce the number of dummies from λ to λ^* where*

$$\log \lambda^* = (1 + \frac{2\log L}{\kappa + 2\log c}) \log \lambda - \frac{\kappa \log L}{\kappa + 2\log c}.$$

Proof. Let κ^+ be the security parameter for a single instance of $\mathsf{MDSD}_{n/L,cw+\lambda,c,w}$. By results from [MW18], we have $\kappa = \kappa^+ - 2\log L$. Using the concrete security result in Section 3.2, for the optimized version we have

$$w^* = \frac{\kappa^+ + 2\log c}{\log \lambda^* - \log(n/L)}$$

number of shares. On the other hand, for the non-optimized version we have

$$w = \frac{\kappa + 2\log c}{\log \lambda - \log n}.$$

Let $w = w^*$, we can solve the relation between λ^* and λ as the theorem states. □

A.3 Finding Basis Across Multiple Queries

The complementary approach. Recall that to answer each query share, the server computes the inner product of the database with the (binary) share vector. By viewing the binary share as a selection vector for which database entries to XOR, this results in $n/2$ XORs (for a database of size n) on expectation. When the server needs to compute more than $n/2$ XORs, a simple optimization is to XOR the remaining entries instead along with the pre-computed XOR of all entries. For large n, the number of XORs needed after this optimization is approximated by the normal distribution $N(n/2, n/4)$, and the number of XORs under the complementary approach follows the half-normal distribution, with the expectation being $n/2 - 0.4\sqrt{n}$. Therefore it saves roughly $0.4\sqrt{n}$ XORs compared to the naive approach. One can generalize the above approach to storing the XOR of subsets of entries; and prior works [BIM00, DK99] show that the server can save a factor of d for computation with 2^d extra storage. In our experiments, we will use the non-generalized version as it incurs extra storage only of one database entry and is friendly for database updates.

Fast matrix-vector multiplications. Naively, multiplying an $n \times n$ matrix with an n-length vector has computational cost $O(n^2)$. Using Method of the Four Russians (M4RM) and Strassen-Winograd algorithm, we can improve the cost to $O(n^{(\log_2 7-1)})$. We leave the experimental cost to future work.

A.4 Composing with Single-Server PIR protocol in the Standard Model

The easiest approach to reduce the database dimension for the shuffle PIR is to organize the database as a matrix and have the client retrieve an entire row (instead of an entry), which increases per-client communication. One can further reduce the per-client communication additionally using a standard PIR scheme to retrieve the entry from the row, as specified below.

Take any standard single-server PIR scheme stdPIR and denote the shuffle PIR construction as ShPIR. The server organizes the size-n database as an $d \times (n/d)$ matrix where d is a constant. The key idea here is to use stdPIR to retrieve a column and ShPIR to retrieve a row. The server treats each column as a database in ShPIR and runs $\mathsf{ShPIR.Setup}$ on it. The server stores the preprocessed results as lookup tables (hence n/d tables in total).

Suppose a client wants to retrieve the entry at r-th row and c-th column. The client runs the query algorithm of ShPIR on index $r \in [d]$ and generates k sub-queries. Then the client sends k messages anonymously, where the j-th message consists of the j-th sub-query of ShPIR and a stdPIR query for index $c \in [n/d]$. On receiving each message, the server first processes the sub-query of ShPIR (essentially n/d table lookup operations), which results in n/d elements; then the server processes the stdPIR query on these n/d elements.

Compared to running stdPIR on a size-n database, this technique reduces server computation by a factor of d. And the ShPIR database size is d, which neither requires too many clients nor incurs high anonymity cost. The tradeoff is that a client sends k messages in the stdPIR-ShPIR combination instead of one message when using stdPIR only.

Later in Appendix B, we can further reduce the black-box standard single-server PIR by extending the functionality of the shuffler.

B Extensions to the Shuffle Model

In Section 5.2, we presented shuffle PIR constructions which assume that all the queries are shuffled together before the server receives them. In order to satisfy this assumption in practice, we need to employ an additional party who can perform the shuffling. Once we have such an additional party, it is an interesting question whether we can leverage it to further improve the efficiency of the constructions. Of course, there are two-server PIR constructions that already present solutions [GGM98] where the second party does more than shuffling. However, those require that both parties have the database and their computation uses it. This imposes challenges when the database is frequently changing (which is the case in real world) since the two parties need to synchronize the changes, which incurs heavy communication cost.

In this section, we explore the idea how we can leverage more efficiently an intermediary party such as the shuffler to reduce the overall overhead for the PIR constructions, but in a way that still preserves the property that the intermediary, which we will call a *proxy*, does not need to know anything about the database. We further assume that the intermediary and the server have much more computational resources than the clients, thus shifting work from the clients to any of these parties will lead to an improvement. Similarly, in the real-world setting, communication between the server and the intermediary is cheaper than the communication to the client (e.g., the server and intermediary have more bandwidth and they have stable networks), hence, another goal will be to reduce the message size going to the client even if it comes with larger messages between the intermediary and server.

One obvious way to shift client's work in our previous construction (Section 5.2) to the intermediary is to have the intermediary insert the dummies in addition to shuffling the queries that come from the clients.

We can further leverage the intermediary to reduce the communication back to the client. In order to reduce the length of query vectors, we arranged the database by two dimensions and use the shuffle PIR to retrieve entries from the first dimension, and hence the PIR answer is of length of the second dimension.

One option to reduce the answer size was to compose the shuffle PIR with a single-server PIR scheme to select the query item across the second dimension (Appendix A.4), but this incurs heavy computational cost for the server since existing single-server PIR schemes are expensive. A different idea is to enable the proxy to select the appropriate record from the items in the second dimension. To do this without revealing any information about the query to the intermediary, the client can send a random permutation to the server that specifies how the server should permute the items selected with the shuffle PIR step before sending them to the intermediary. The client would also provide the proxy with the appropriate location of the queried record in the random permutation, thus the proxy can select the appropriate record from ones it receives from the server and forward it to the client. We present the details of this approach in Construction 5.

In a different idea we explore whether we can outsource the query generation for shuffle PIR to the proxy beyond just adding the dummy items before the shuffle. In the ideal setting the client will be sending constant-sized message for a query index. One idea to achieve this is to have client generate an encryption of its index i under additive homomorphic encryption which has plaintext space $\mathbb{Z}_{\mathsf{n}}$, and send the encrypted index $\mathsf{HE.Enc}(i)$ to the proxy. The proxy will generate a shuffle PIR query as follows: it generates shares of the index vector selecting position one; for each share it selects a random number $\mathsf{r} \in \mathbb{Z}_{\mathsf{n}}$. It cyclically shifts (towards right) the share vector by r positions and additionally computes $\mathsf{HE.Enc}(i - r - 1)$, which is paired with the rotated share. The complete client query consists of all share vectors together with their corresponding encrypted indices. Dummy items now will be generated also as a random share vector plus random encrypted index. The final shuffle PIR queries are the shuffle across the newly defined client query shares and the dummies.

The server processes each query by decrypting the index sent with each share, rotating the share in increasing direction as many positions as the decrypted index and then computing an inner product with the database. Each client recovers its output as before. Note that this approach is composable with the one above that reduces the length of the response sent from the intermediary to the client.

We could also aim to use the proxy to reconstruct the answer that should be sent to the client from the encrypted shares of the answer sent by the server. As a reminder, those shares will need to be encrypted under different keys so that they are not linkable by the server. To achieve both goal we will leverage homomorphic encryption that supports both message and key homomorphism. This constructions enables the proxy to generate different encryption keys that add up to the key chosen by the client and sent encrypted to the proxy. As a result adding the additive shares of the output encrypted under all of these keys results in the reconstructed output encrypted under the key of the client. The details of this approach are presented in Construction 15.

B.1 Extended Definition

In the previous sections we used a definition for shuffle PIR which implicitly assumes the shuffling of the clients' inputs without explicitly defining a functionality for the shuffler. Since in the proxy constructions the intermediary party will perform operations beyond shuffling, we extend the definition with functionalities that correspond to the proxy's action when submitting the query and sending the answer back to the client. We present the details of this extended definition next.

Definition 14 (PIR in the proxy model (Reduced Client Download))**.** *Let Σ be a finite alphabet and $n \in \mathbb{N}$. A (single-server) PIR protocol in the proxy model on database Σ^n is a tuple of algorithms* $\mathsf{ShPIR} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{ProxyQ}, \mathsf{ProxyPerm}, \mathsf{Answer}, \mathsf{ProxyRRecon})$.

- $\mathsf{Setup}(x) \to P_x$*: a deterministic algorithm executed by the server that takes in an n-entry database $\mathbf{x} \in \Sigma^n$ and outputs its encoding P_x.*

- $\mathsf{Query}(i; n) \xrightarrow{\$} \sigma_i$*: a randomized algorithm (parameterized by n) executed by the client that takes in an index $i \in [n]$, and outputs an encoded query σ_i.*

- $\mathsf{ProxyQ}(\sigma_i) \xrightarrow{\$} (q_1, \ldots, q_k, \mathsf{st}_{\mathsf{C}})$*: a randomized algorithm executed by the proxy that takes an encoded query σ_i generated by a client C and outputs sub-queries $q_1, \ldots, q_k$, and state st_{C}.*

- $\mathsf{ProxyPerm}(q_1, \ldots, q_{m\cdot k})$*: it takes sub-queries from m clients and outputs a new set of sub-queries and a state* $\mathsf{st_P}$*.*

- $\mathsf{Answer}(P_x, q_\ell) \to a_\ell$*: a deterministic algorithm executed by the server that takes in the encoding P_x and a sub-query q_ℓ, and outputs an answer a_ℓ.*

- $\mathsf{ProxyR}((a_1, \ldots, a_{m\cdot k}), \mathsf{st}_{\mathsf{C},1}, \ldots, \mathsf{st}_{\mathsf{C},m}, \mathsf{st_P}) \to (\tau_1, \ldots, \tau_m)$*: a deterministic algorithm executed by the proxy that takes in answers $a_1, \ldots, a_{m\cdot k}$, where for all $j \in [m], \ell \in [k]$, $a_{k\cdot(j-1)+\ell}$ is the answer to the j'th client sub-query q_ℓ, and $\mathsf{st}_{\mathsf{C},j}$ is the state corresponding to the j'th client, while $\mathsf{st_P}$ is the proxy state across all clients. It outputs encoded output τ_j for client j.*

- $\mathsf{Recon}(\tau_i) \to \mathbf{x}[i]$*: a deterministic algorithm executed by the client that takes an encoded output τ_i from the proxy; and outputs $\mathbf{x}[i] \in \Sigma$.*

B.2 Proxy PIR with Reduced Client Download

The next construction realized the idea that the proxy can filter only the relevant element for the client from a shuffle PIR response that includes multiple database elements.

Construction 5 (Proxy PIR Protocol with Reduced Client Download)**.** *Let $\mathbb{F}$ be a field and $n \in \mathbb{N}$, and let $\mathbf{x} \in \mathbb{F}^n$ be a database of n elements. Our Proxy PIR protocol is parameterized with positive integers k (the number of shares per query), z (the number of dummy sub-queries per query), and c (the number of queries). Furthermore, let $\lambda = c \cdot z$ be the total number of dummy sub-queries. The PIR protocol consists of the following algorithms.*

- $\mathsf{Setup}(\mathbf{x}) \to \mathbf{M} \in \mathbb{F}^n$*: Rearrange the database as database of size m_1 where each entry consists of m_2 consecutive entries from $\mathbf{x}$, $\mathbf{M} \in (\mathbb{F}^{m_2})^{m_1}$.*

- $\mathsf{Query}(j; m_1, m_2) \overset{\$}{\to} \sigma_i$*: The query algorithm first splits the indicator vector $\mathbf{u}_j \in \mathbb{F}^{m_1}$ into additive secret shares $\mathbf{s}_1, \ldots, \mathbf{s}_k \in \mathbb{F}^{m_1}$ such that $\mathbf{s}_i$ are uniformly random conditioned on $\sum_{i=1}^k \mathbf{s}_i = \mathbf{u}_j \in \mathbb{F}^{m_1}$. For each $1 \le i \le k$, sample a uniform permutation $\pi_i : [1, m_2] \to [1, m_2]$ and set $t_i = \pi_i(j \mod m_1)$, and set the the i-th share to be $(\mathbf{s}_i, \pi_i, t_i)$. The encoded query is then the tuple of vectors $\sigma_i \leftarrow \Big((\mathbf{s}_1,\ \pi_1,\ t_1), \ldots, (\mathbf{s}_k,\ \pi_k,\ t_k)\Big)$. Note that the values $\mathbf{s}_i, \pi_i$ are encrypted with the public key for the server and t_i is encrypted for the proxy.*

- $\mathsf{ProxyQ}(\sigma_i) \to \sigma_i$*.*

- $\mathsf{ProxyPerm}(\sigma_1, \ldots, \sigma_c)$*: The proxy generates λ dummy items by sampling random vectors and permutations $(\mathbf{r}_h, \tau_h)$ for all $1 \le h \le \lambda$. It outputs for the server a permutation α of the sub-queries $\mathbf{q}_i = (\mathbf{s}_i, \pi_i)$ in all $\sigma_1, \ldots, \sigma_c$ and the dummy queries $\mathbf{q}_{m\cdot k+h} = (\mathbf{r}_h, \tau_h)$ for all $1 \le h \le \lambda$. It keeps as state $t_1, \ldots t_{ck}$ and α.*

- $\mathsf{Answer}(\mathbf{M}, \mathbf{q} = (\mathbf{s}, \pi))$*: The server computes $\mathbf{y} = \langle \mathbf{M}, \mathbf{s} \rangle$ which results in a vector of items of length m_2. It returns $\pi(\mathbf{y})$.*

- $\mathsf{ProxyR}((\mathbf{y}_1, \ldots \mathbf{y}_k, \mathbf{d}_1, \ldots, \mathbf{d}_z, \mathsf{st_C} = (t_1, \ldots, t_k)$*: The proxy returns $(\mathbf{y}_1[t_1], \ldots, \mathbf{y}_k[t_k])$. The answers $\mathbf{d}_1, \ldots, \mathbf{d}_z$ for the dummy queries are discarded.*

- $\mathsf{Recon}(a_1, \ldots, a_k) \to \sum_{i=1}^k a_i$*: To reconstruct the requested database element, the client computes the sum of answers $a_1, \ldots, a_k$ received from the proxy.*

The security of the above construction follows from the security of the shuffle PIR protocol and the fact that the indices t_i that the proxy receives are completely random and thus it does not learn any information about the query.

B.3 Proxy PIR with Reduced Client Upload

The next construction show how the proxy can generate the shuffle query on the behalf of the client using an encrypted query index.

Definition 15 (PIR in the proxy model (Reduced Client Download))**.** *Let Σ be a finite alphabet and $n \in \mathbb{N}$. A (single-server) PIR protocol in the proxy model on database Σ^n is a tuple of algorithms* $\mathsf{ProxyPIR} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{ProxyQ}, \mathsf{Answer}, \mathsf{ProxyRRecon})$.

- $\mathsf{Setup}(\mathbf{x} \in \mathbb{F}^n) \xrightarrow{\$} (\mathbf{x}, \mathsf{pk_Q}, \mathsf{sk_Q}, \mathsf{pk_K}, \mathsf{sk_K})$*: The setup algorithm, as usual, treats the database as a vector of length n over $\mathbb{F}$. It also generates key pairs $(\mathsf{pk_Q}, \mathsf{sk_Q})$ and $(\mathsf{pk_K}, \mathsf{sk_K})$ for a public-key additive homomorphic encryption scheme $\mathcal{E}$, where the message space for the first instantiation is $\mathbb{Z}_n$ and the message space for the second instantiation coincides with the key space $\mathcal{K}$ for a symmetric-key KMAHE scheme $\mathcal{E}'$. The server sends $\mathsf{pk_Q}$ and $\mathsf{pk_K}$ to the clients and the proxy.*

- $\mathsf{Query}(q_j; n) \xrightarrow{\$} (\mathsf{ct_Q}, \mathsf{ct_K})$*: The client first encrypts $\mathsf{ct_Q} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk_Q}, q_j)$. It samples a key for the KMAHE scheme: $\mathcal{E}'$ $\mathsf{sk_C} \leftarrow_\$ \mathcal{K}$, and encrypts it as $\mathsf{ct_K} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk_K}, \mathsf{sk_C})$.*

- $\mathsf{ProxyQ}(\mathsf{ct_Q}, \mathsf{ct_K}) \xrightarrow{\$} (\{\mathbf{s'}_i, \mathsf{ct}_i, \mathsf{ct}'_i\}_{i=0}^{w})$*: The proxy generates additive secret shares $(\mathbf{s}_0, \ldots, \mathbf{s}_w) \leftarrow \mathsf{Share}(\mathbf{1} \in \mathbb{F}^n, w+1)$, where $\mathbf{1}$ is the selection vector with 1 in first position and 0's in all other positions. It samples $w+1$ symmetric keys $\mathsf{R}_i \leftarrow_\$ \mathcal{K}$ such that $\sum_{i=0}^{w} \mathsf{R}_i = 0$.*

  For each $0 \le i \le w$, the proxy generates a random shift $0 \le k_i \le n-1$, computes $\mathsf{ct}_i \leftarrow \mathsf{ct_Q} + \mathcal{E}.\mathsf{Enc}(\mathsf{pk_Q}, k_i)$ and $\mathbf{s'}_i$ which is obtained from $\mathbf{s}_i$ by cyclically rotating k_i positions backwards, i.e. $\mathbf{s'}_i[j] = \mathbf{s}_i[j - k_i \mod n]$ for all j. It also computes $\mathsf{ct}'_i \leftarrow \mathsf{ct_K} + \mathcal{E}.\mathsf{Enc}(\mathsf{pk_K}, \mathsf{R}_i)$.

- $\mathsf{ProxyPerm}(\{\mathbf{s'}_i, \mathsf{ct}_i, \mathsf{ct}'_i\}_{i=0}^{w \cdot m}) \xrightarrow{\$} (\{\mathbf{s'}_i, \mathsf{ct}_i, \mathsf{ct}'_i\}_{i=0}^{w \cdot m + \lambda})$*: The proxy generates λ dummy items by sampling random vectors, encryptions of random numbers and random encryption keys $(\mathbf{r}_j, \mathsf{ct}_j, \mathsf{ct}'_j)$. It outputs the permuted real items from m clients and λ dummy items. It keeps as state the applied permutation π.*

- $\mathsf{Answer}(\mathbf{s}, \mathsf{ct}, \mathsf{ct}') \xrightarrow{\$} \mathsf{ct}_a$*: The server decrypts $k = \mathcal{E}.\mathsf{Dec}(\mathsf{sk_Q}, \mathsf{ct})$. It then cyclically shifts $\mathbf{s}$ forward with k positions to obtain $\mathbf{s}'$, i.e. $\mathbf{s}'[j] = \mathbf{s}[j + k \mod n]$. It computes a as the inner product of $\mathbf{s}'$ and the database $\mathbf{x}$.*

  The server decrypts $\mathsf{sk_C} = \mathcal{E}.\mathsf{Dec}(\mathsf{sk_K}, \mathsf{ct}')$ and outputs $\mathsf{ct}_a \leftarrow \mathcal{E}'.\mathsf{Enc}(\mathsf{sk_C}, a)$.

- $\mathsf{ProxyR}\big(\pi(\mathsf{ct}_{a_1}, \ldots, \mathsf{ct}_{a_{w \cdot m}}, \mathsf{ct}_{d_1}, \ldots, ct_{d_\lambda})\big) \to (\mathsf{ct}_{q_1}, \ldots, \mathsf{ct}_{q_m})$*: the proxy reverses the permutation and filters out the answers of the dummy items. For each client $j \in [c]$, it computes $\mathsf{ct}_{q_j} = \sum_{i=(j-1)\cdot w+1}^{j \cdot w} \mathsf{ct}_i$, which is equivalent to $\mathcal{E}'.\mathsf{Enc}(w \cdot \mathsf{sk_C} + \sum_{i=1}^{w} \mathsf{R}_i, \sum_{i=1}^{w} a_i) \equiv \mathcal{E}'.\mathsf{Enc}(w \cdot \mathsf{sk}_C, \mathbf{x}[q_i])$.*

- $\mathsf{Recon}(\mathsf{ct}_{q_i}) \to \mathbf{x}[q_i]$*: The client decrypts using the decryption key for the corresponding query $\mathbf{x}[q_i] \leftarrow \mathcal{E}'.\mathsf{Dec}(w \cdot \mathsf{sk_C}, \mathsf{ct}_{q_i})$.*

Security Sketch. The view of the server is the proxy construction is the same as in the shuffle PIR construction. The server sees random shuffled shares of the query selection vectors, which are represented as the shifted share vector plus the shift. The encryption keys are also indistinguishable from independently sampled keys.

The semi-honest proxy sees the query bit encrypted and the response shares are also encrypted under keys that it does not know.

Database	Field	# Clients	Our PIR Protocol in Proxy Model						
			n_1	d	# Sub-queries	Client Up/Down (KiB)	Proxy Up/Down (KiB)	Proxy time (ms)	Server time (ms)
$2^{20} \times 256$ bytes	$\mathbb{F}_{65537}$	10000	2^{15}	2^{14}	32	32 / 26	1752 / 1628	41	74
		100000	2^{15}	2^{15}	23	32 / 26	622 / 289	15	47
$2^{23} \times 1$ bytes	$\mathbb{F}_{65537}$	10000	2^{17}	2^{14}	33	32 / 26	7233 / 26880	172	306
		100000	2^{18}	2^{16}	26	32 / 13	2825 / 2625	67	218

Table 11: Parameters and communication costs of our PIR protocol in the proxy model as in Appendix B.3.