Querying for Queries
Indexes of Queries for Efficient and Expressive IT-PIR

Syed Mahbub Hafiz
Indiana University
Computer Science Department
Bloomington, IN 47405, USA
shafiz@indiana.edu

Ryan Henry
Indiana University
Computer Science Department
Bloomington, IN 47405, USA
henry@indiana.edu

ABSTRACT
We propose *indexes of queries*, a novel mechanism for supporting efficient, expressive, and information-theoretically private single-round queries over multi-server PIR databases. Our approach decouples the way that users construct their requests for data from the physical layout of the remote data store, thereby enabling users to fetch data using "contextual" queries that specify *which* data they seek, as opposed to "positional" queries that specify *where* those data happen to reside. For example, an open-access eprint repository could employ indexes of queries to let researchers fetch academic articles via PIR queries such as for "this year's 5 most cited papers about PIR" or "the 3 most recently posted papers about PIR". Our basic approach is compatible with any PIR protocol in the ubiquitous "vector-matrix" model for PIR, though the most sophisticated and useful of our constructions rely on some nice algebraic properties of Goldberg's IT-PIR protocol (Oakland 2007). We have implemented our techniques as an extension to Percy++, an open-source implementation of Goldberg's IT-PIR protocol. Our experiments indicate that the new techniques can greatly improve not only utility for private information retrievers but also efficiency for private information retrievers and servers alike.

KEYWORDS
Private information retrieval; expressive queries; batch codes; batch queries; ramp schemes; efficiency

1 INTRODUCTION
Private information retrieval (PIR) is a cryptographic technique that enables users to fetch records from untrusted and remote database servers without revealing to those servers which particular records are being fetched. This paper proposes a new technique for conducting *efficient*, *expressive*, and *information-theoretically private* PIR queries over structured or semi-structured (i.e., tagged) data. Conceptually, the new approach involves building a layer of indirection (realized using a special kind of sparse "database" we call an *index of queries*) atop existing PIR protocols.

With only a few exceptions, existing PIR constructions require users to indicate which records they wish to fetch via the indices of those records—that is, via the *physical locations* of those records relative to others in the data store. Our indexes of queries decouple the way that users construct their requests for data from the physical layout of the remote data store, thereby enabling users to fetch data using "contextual" PIR queries that specify *which* data they seek, as opposed to "positional" PIR queries that specify *where* in the database those data happen to reside.

Database operators can construct many distinct indexes of queries for a given data set, thus providing many distinct views through which users can interact with the underlying data. Abstractly, each index of queries facilitates requests for "the best k matches for z", where the precise meaning of 'best', an upper bound on the number of matches to return k, and a domain of possible search criteria z are all determined by the database operator and fixed for the particular index of queries under consideration. Queries of the above form arise naturally in a plethora of online and mobile applications. In many such applications, the query term z reveals a great deal of identifiable and potentially sensitive information about the habits, interests, and affiliations of the querier [20]. The index-of-queries approach we propose herein provides significant improvements to both the *efficiency* and *expressiveness* of the most performant and well studied PIR techniques in the literature, exposing intuitive APIs through which applications can safely, easily, and efficiently interact with the underlying PIR. We therefore believe (and certainly hope) that indexes of queries will prove to be a useful building block in the construction of efficient, privacy-preserving alternatives to many widely deployed products and services.

Relationship with prior work. The research literature on PIR is vast; for over two decades, the cryptography, privacy, and theory research communities have studied PIR intensively and from a variety of perspectives. However, this considerable attention notwithstanding, apart from a few notable exceptions, existing work focuses exclusively on an oversimplified model in which users request fixed-length blocks—or even *individual bits!*—of data by specifying the physical locations of those data within the database.

A small body of existing work constructs PIR queries whose expressiveness extends beyond the ability to fetch records by index, including techniques that enable keyword-based [3–5, 7] and simple SQL-based [21, 22, 27] PIR queries. Although our techniques bare a superficial resemblance to these prior efforts, the precise problem we solve and the technical machinery we use to solve it are fundamentally new. Indeed, our approach offers several distinct advantages (and a few limitations) compared with existing techniques, and we therefore view indexes of queries as being complementary to—as opposed to an alternative to—existing techniques for expressive PIR. A later section compares and contrasts indexes of queries with competing approaches.

Motivation. The primary objective of this paper is to introduce and analyze indexes of queries as a new PIR technique in the cryptographic engineers' toolkit, rather than to explore the nuances of any particular system that one might use indexes of queries to build. Nevertheless, to both motivate and ground our proposal, we briefly consider three natural use cases that showcase the immediate applicability of indexes of queries to the construction of privacy-respecting technologies. We reiterate that these use cases are merely intended to illustrate a high-level idea; indeed, it is beyond the scope of this paper to present the full architecture of—let alone to treat all the minutiae of implementing full, workable systems from—any of these motivating examples.

Use case 1: Maps and location-based recommendation systems. A mapping service like Google Maps or a recommendation service like Yelp could instantiate indexes of queries over a *Points of Interest* (POI) database to satisfy PIR requests such as for "the 10 cafés nearest my current location" or "the 5 highest rated Italian restaurants in Manhattan".

Use case 2: Social networks and microblogging platforms. A Twitter-like microblogging service could instantiate indexes of queries over its database of tweets to satisfy PIR requests such as for "the 10 most recent tweets by @realDonaldTrump" or "the 15 top trending tweets for hashtag #ccs17".

Use case 3: Streaming audio and video services. Streaming media services like Youtube or Spotify could instantiate indexes of queries over their respective media catalogs to satisfy PIR requests such as for "the most recent episode of Last Week Tonight with John Oliver" or "the 10 songs topping the latest Billboard Hot 100".

Countless use cases beyond those just listed are possible; e.g., throughout our technical discussions we will use the running example of privately fetching emails from a remote inbox. One could use this idea to, say, hide users' email access patterns from a web mail service like Gmail or, more interestingly, to build a next-generation Pynchon Gate [24] for pseudonymous mail retrieval.

Outline. The remainder of the paper is structured as follows. Section 2 describes the abstract PIR framework in which all of the indexes-of-queries constructions reside. Section 3 introduces *simple indexes of queries*, the most basic (and least interesting) form of our construction, while Section 4 describes a more sophisticated construction for *batch indexes of queries*, which leverage ideas from coding theory to reduce costs and improve privacy compared to simple indexes of queries. Section 5 further extends this idea to construct *indexes of batch queries*, which allow users to fetch a batch of several related blocks using a single, fixed-length query. (The latter 'indexes of batch queries' are what is needed to realize the queries arising in the above motivating examples.) Section 6 reviews prior work on expressive PIR queries—SQL- and keyword-based PIR queries and PIR from function secret sharing—and comments on the synergistic relationship between our work and those techniques. We present some findings from our proof-of-concept implementation in Section 7 before concluding in Section 8.

2 THE "VECTOR-MATRIX" PIR MODEL

Our constructions are in the ubiquitous *vector-matrix model* for PIR. Vector-matrix PIR is a special case of *linear PIR* where the database is represented as an $r \times s$ matrix $\mathbf{D}$ over a finite field $\mathbb{F}$ in which each of the r rows is a fetchable unit of data (called a *block* in typical PIR parlance). Users encode requests for blocks as vectors from the so-called standard basis for $\mathbb{F}^r$: a user desiring the ith block (i.e., the ith row of $\mathbf{D}$) represents its request with the length-r vector $\vec{e}_i$ having a 1 in its ith coordinate and 0s elsewhere. The response to request $\vec{e}_i$ is defined as the vector-matrix product $\vec{e}_i \cdot \mathbf{D}$, which is easily seen to equal the desired ith row of $\mathbf{D}$. We refer to such vector-based requests as *positional queries* in order to highlight the fact that they require queriers to know the physical positions (i.e., the row numbers) within $\mathbf{D}$ of whatever blocks they seek to fetch.

PIR protocols in the literature obtain privacy in the vector-matrix model through a variety of different means. Of particular interest to us is the information-theoretically private (IT-PIR) approach based on linear secret sharing. Here, the user "shares" its query vector $\vec{e}_i$ component-wise using a linear secret sharing scheme, and then it sends each of the resulting vectors of shares to a different server from a pool of (non-colluding, but otherwise untrusted) servers who each hold a replica of $\mathbf{D}$. Upon receiving a share vector from the user, each server independently computes and returns to the user the product with $\mathbf{D}$ of the share vector it just received. As an immediate consequence of linearity, the servers' responses are each component-wise secret sharings of the vector-matrix product $\vec{e}_i \cdot \mathbf{D}$. Thus, to recover its requested block, the user performs a component-wise secret reconstruction over the responses it collects from the various servers.

Goldberg's IT-PIR protocol. One natural and attractive choice for the secret sharing scheme, the use of which for vector-matrix IT-PIR was first advocated by Goldberg [13], is Shamir's $(t+1, \ell)$-threshold scheme [25]. To share a basis vector $\vec{e}_i$ with Shamir's $(t+1, \ell)$-threshold scheme, the user selects pairwise distinct scalars $x_1, \ldots, x_\ell \in \mathbb{F} \setminus \{0\}$ and a uniform random vector of polynomials $\vec{\mathbf{F}} \in (\mathbb{F}[x])^r$, subject to the conditions that (i) each polynomial in $\vec{\mathbf{F}}$ has degree at most t, and (ii) a component-wise evaluation of $\vec{\mathbf{F}}$ at $x = 0$ gives $\vec{e}_i$. The jth server receives $(x_j, \vec{\mathbf{Q}}_j)$, where $\vec{\mathbf{Q}}_j = \vec{\mathbf{F}}(x_j)$ is a component-wise evaluation of $\vec{\mathbf{F}}$ at $x_j = j$. We refer to a sequence $(x_1, \vec{\mathbf{Q}}_1), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell)$ of $\ell > t$ such ordered pairs (computed from a common $\vec{\mathbf{F}}$ and pairwise distinct x_i) as a *component-wise* $(t+1, \ell)$*-threshold sharing of* $\vec{e}_i$.

Shamir's threshold scheme provides the requisite linearity and a useful Byzantine robustness property, owing to its relationship with (indeed, equivalence to) Reed-Solomon codes [23] and related multiple-polynomial error-correcting codes [8]. The protocol obtained by using Shamir's $(t+1, \ell)$-threshold scheme in the vector-matrix model realizes t*-private* (m, ℓ)*-server IT-PIR* for any $m \geq t+1$: the user retrieves its desired block provided $m \geq t+1$ out of ℓ servers respond, yet no coalition of t or fewer malicious servers can use the share vectors its members receive to learn any information about which blocks the user has requested. (It is also v*-Byzantine robust* for any $v \leq m-t-2$: the user retrieves its desired block even if up to $m-t-2$ servers return incorrect responses [10].)

$$\mathbf{D} \coloneqq \begin{array}{c} \\ \left(\begin{array}{lllll} \texttt{subject} & \texttt{sender} & \texttt{date} & \texttt{size} & \texttt{body} \\ \text{Re: ccs2017 submission} & \text{Bob} & \text{2017-02-17} & \text{7.7KB} & \texttt{0x2ff1 e1a9...} \\ \text{definitely not a virus} & \text{Dave} & \text{2017-02-1f1} & \text{13.0KB} & \texttt{0xb05f d7a1...} \\ \text{UK-LOTTO sweepstake!} & \text{Alice} & \text{2017-02-04} & \text{336KB} & \texttt{0x0365 ce00...} \\ \text{Fwd: Re: Fwd: roflmao} & \text{Carol} & \text{2017-01-07} & \text{2.5KB} & \texttt{0x7e7a 36b7...} \\ \text{cash4gold!!!1} & \text{Edward} & \text{2016-12-23} & \text{4.0KB} & \texttt{0xd96d faff...} \end{array}\right) \end{array}$$

Figure 1: Toy example of an email inbox database comprising five emails and associated metadata.

The above intuitive notion of *t-privacy* is formalized by requiring statistical independence between the pair (I, Q) of random variables respectively describing (i) which particular block the querier is requesting, and (ii) the joint view of any coalition of up to t servers involved in the request. We refer the reader to Henry [16, end of §2] for a detailed formal definition for *t-private k-batch (m, ℓ)-server IT-PIR* in the vector-matrix model. Our discussion of the Shamir-based PIR protocol up to this point corresponds to Henry's definition with a fixed batching parameter of $k = 1$; looking forward, the construction we present in Section 5 returns several related blocks for each query and, therefore, corresponds to Henry's definition for some $k > 1$.

In their most basic form, our indexes of queries are compatible with any PIR protocol in the vector-matrix model, although our exposition assumes—and our more sophisticated and useful indexes-of-queries constructions rely on some nice algebraic properties of—(a scheme [16] that builds upon a scheme [18] that builds upon) Goldberg's Shamir-based IT-PIR; thus, although we have attempted to make our exposition of the new constructions as self-contained as possible, readers unfamiliar with the various building blocks may wish to peruse Goldberg's paper [13]—and the follow up papers by Henry [16] and by Henry, Huang, and Goldberg [18]—for an initial 'lay of the land'.

3 QUERYING FOR QUERIES

At the heart of our approach is a simple observation regarding the use of $(0, 1)$-matrices as PIR databases. We begin with the most simplistic possible version of our idea, restricting our attention to $r \times r$ permutation matrices and building up to more general and interesting cases as the paper progresses.

Recall that an *$r \times r$ permutation matrix* is just an $r \times r$ matrix having exactly one 1 in each row and each column, and 0s elsewhere (equivalently, it is a matrix obtained by permuting the rows of an $r \times r$ identity matrix). Each such matrix represents a specific permutation on r elements: given a length-r vector $\vec{v}$ and an $r \times r$ permutation matrix Π, the vector-matrix product $\vec{v} \cdot \Pi$ yields a length-r vector with the same components as $\vec{v}$, but in a permuted order.

For example, given $\vec{v} = \langle a \;\; b \;\; c \rangle$ and a permutation matrix

$$\Pi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

it is easy to check that $\vec{v} \cdot \Pi = \langle a \; c \; b \rangle$; i.e., Π permutes $\vec{v}$ by transposing its second and third components.

The following observation is exceedingly obvious, and yet it is sufficiently central to our approach as to nonetheless warrant formal explication.

Observation 3.1. If $\vec{e} \in \mathbb{F}^r$ is a standard basis vector and $\Pi \in \mathbb{F}^{r \times r}$ is a permutation matrix, then $\vec{e} \cdot \Pi$ is a (possibly different) standard basis vector.

For example, given the above-defined 3×3 permutation matrix Π and the standard basis $\{\vec{e}_1, \vec{e}_2, \vec{e}_3\}$ for $\mathbb{F}^3$, we have that $\vec{e}_1 \cdot \Pi = \vec{e}_1$, that $\vec{e}_2 \cdot \Pi = \vec{e}_3$, and that $\vec{e}_3 \cdot \Pi = \vec{e}_2$. In the context of IT-PIR, we are actually interested in the following immediate corollary to Observation 3.1.

Corollary 3.2. Let $\vec{e} \in \mathbb{F}^r$ be a standard basic vector and let $\Pi \in \mathbb{F}^{r \times r}$ be a permutation matrix. If $(x_1, \vec{Q}_1), \ldots, (x_\ell, \vec{Q}_\ell)$ is a component-wise $(t + 1, \ell)$-threshold sharing of $\vec{e}$, then $(x_1, \vec{Q}_1 \cdot \Pi), \ldots, (x_\ell, \vec{Q}_\ell \cdot \Pi)$ is a component-wise $(t + 1, \ell)$-threshold sharing of a (possibly different) standard basis vector $\vec{e}\,' \in \mathbb{F}^r$; namely, of $\vec{e}\,' = \vec{e} \cdot \Pi$.

Colloquially, one can think of Corollary 3.2 as stating that a t-private IT-PIR query issued against a permutation matrix yields another t-private IT-PIR query (possibly for some other block) or, put another way, that a permutation matrix is, in a sense, just a "database of positional PIR queries".

Despite the naïvety of our discussion up to this point, we are already well positioned to demonstrate a novel application of permutation matrices to PIR queries.

3.1 Example application: Private queries over a remote email inbox

Consider the toy example of an email inbox depicted in Figure 1. The inbox $\mathbf{D}$ in the figure contains five emails, which are physically stored, naturally, in the same order that they were received.

Each row of $\mathbf{D}$ represents one email and is structured around a schema that includes—in addition to the body of the email—fields for metadata about the email including its `subject`, its `sender`, the `date` it was received, and its `size`. Of course, the schema for a real email inbox would include several additional fields.

Suppose we wish to set up a PIR protocol to facilitate retrieval of emails from the inbox $\mathbf{D}$. In a typical PIR setting, the user would fetch an email from $\mathbf{D}$ using a positional PIR query. Doing so would require the user to know (or, at least, to learn) quite a lot about the *physical layout* of $\mathbf{D}$, as the row number of the desired email corresponds to its chronological order among all of the other emails. By contrast, a typical non-private email client would provide a convenient interface to help the user locate emails of interest, for instance by imposing a user-selected *logical order* on the emails and allowing the user to browse through them in this sorted order. As a concrete example, the email client might present the user with a view of the inbox in which emails are sorted numerically by `size`, or lexicographically by `subject` or `sender`, among other possibilities.

Observation 3.3. Each of the above-mentioned views of $\mathbf{D}$ (i.e., sorted by `size`, by `subject`, or by `sender`) corresponds to a particular 5×5 permutation matrix.

For example, referring back to $\mathbf{D}$, we observe that the permutation matrices

$$\Pi_{\mathsf{sender}} := \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \Pi_{\mathsf{size}} := \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

respectively map a query encoding the ith standard basis vector of $\mathbb{F}^5$ to a query for the ith email in a lexicographic ordering of the inbox by `sender` or a numerical (decreasing) ordering of the inbox by `size`. Thus, the user could request, say, the *largest* email in $\mathbf{D}$ by sending a vector of shares of the basis vector $\vec{e}_1 \in \mathbb{F}^5$, along with the hint "`size`", to each of the PIR servers hosting $\mathbf{D}$. We emphasize that the user can construct this query knowing only the total number of rows in Π_{size}; in particular, the user need not know anything about which emails occupy which rows of $\mathbf{D}$.

Upon receiving the share vector $\vec{\mathbf{Q}}_j$ and hint "`size`" from the user, server j first permutes the components of $\vec{\mathbf{Q}}_j$ via multiplication with Π_{size} to get $\vec{\mathbf{Q}}_j^{\mathsf{size}} := \vec{\mathbf{Q}}_j \cdot \Pi_{\mathsf{size}}$, after which it computes and returns the response $\vec{\mathbf{R}}_j := \vec{\mathbf{Q}}_j^{\mathsf{size}} \cdot \mathbf{D}$ as usual. It is easy to verify—and the reader should take a moment to do so, since going forward we will repeatedly use this simple idea, but in increasingly sophisticated ways—that, upon reconstructing the servers' responses, the user indeed learns the largest email in $\mathbf{D}$, just as it sought to do. To see why, simply note that $\vec{e}_1 \cdot \Pi_{\mathsf{size}} = \vec{e}_3$, and that $\vec{e}_3 \cdot \mathbf{D}$ yields the 336 KB email, which has the largest `size` among emails in the inbox.

Before we move on, a few remarks about this simple example are in order. First, we note that the example highlights a potential application of permutations in PIR, but it reveals no obvious advantage to thinking about such permutations in terms of multiplication by a permutation matrix (as opposed to using some other representation of a permutation). Nevertheless, in the sequel we will see increasingly sophisticated variations of this idea which *do* rely inherently on the idea of "permuting queries" by way of matrix multiplications.[1] Second, we reiterate that the user in this example *does not require any specific knowledge* about $\mathbf{D}$ or Π_{size}, beyond the height and semantic meaning of Π_{size}. In fact, even upon reconstructing the servers' responses, the user still learns nothing about the physical layout of $\mathbf{D}$—not even the row number of the email it just fetched! Finally, although the permuted query vectors arising in our example provide the exact same t-privacy guarantee as regular queries in the underlying PIR protocol,[2] the additional hint "`size`" does reveal some meta-information about which emails the user is after. This meta-information can have implications for privacy; for instance, in our email fetching example, the servers may infer that a user requesting emails by `size` is interested in emails residing in the tails of the `size` distribution (i.e., very small or large emails) as opposed to those near the middle. Thus, in applications that wish to leverage permutations in this way, special care must be taken to identify and quantify if and how such leakage might betray the users' privacy. We emphasize that (i) such leakage is application-dependent and cannot be meaningfully quantified outside the context of a specific application, and (ii) in many (if not most) applications, business logic already betrays similar meta-information.

3.2 From permutation matrices to "simple indexes of queries"

One can think of the permutation matrices Π_{sender} and Π_{size} from the preceding subsection as being two "indexes" through which users can fetch the blocks comprising $\mathbf{D}$. Indeed, both indexes are themselves just special databases whose blocks are all (non-private) positional *queries* for blocks in $\mathbf{D}$; in other words, Π_{sender} and Π_{size} are two very simple examples of what we call "indexes of queries".

Note that such indexes of queries need not take the form of permutation matrices—permutation matrices merely capture the special case in which the index of queries presents a sorted view of all blocks in $\mathbf{D}$. Indeed, one could just as well consider an index of queries Π that (i) is not square, (ii) has some columns containing no 1s (meaning that certain blocks from $\mathbf{D}$ are not accessible via Π), and/or (iii) has some columns containing multiple 1s (meaning that certain blocks from $\mathbf{D}$ are accessible in multiple ways via Π). One even consider indexes of *aggregate* queries, in which some *rows* may contain several arbitrary non-zero entries. Such indexes of queries would map standard basis vectors to requests for *linear combinations* of blocks from $\mathbf{D}$ and may be useful for solving simple statistical queries. However, we leave the development of this idea to future work and, for the time being, cast the following definition for "simple indexes of queries", which captures all but the last possibility just mentioned.

Definition 3.4. A *simple index of queries* for a database $\mathbf{D} \in \mathbb{F}^{r \times s}$ is a $(0,1)$-matrix $\Pi \in \mathbb{F}^{p \times r}$ in which each row contains exactly one 1 entry.

An equivalent definition states that $\Pi \in \mathbb{F}^{p \times r}$ is a *simple index of queries* for $\mathbf{D} \in \mathbb{F}^{r \times s}$ if it maps each standard basis vector from $\mathbb{F}^p$ to a standard basis vector from $\mathbb{F}^r$.

3.3 Leakage: It's not a bug, *it's a feature*

In the epilogue to Section 3.1, we remarked that the mere act of fetching blocks through a given index of queries can implicitly leak meta-information about which blocks the user seeks. Furthermore,

[1]We point out, moreover, that even for this very simple example, representing the requisite permutations as matrix multiplications is not unreasonable, as the special structure of permutation matrices (specifically, their sparsity and the restriction of their components to $\{0, 1\}$) allows the servers to store and compute with them very efficiently, a fact that will later prove crucial.

[2]Indeed, Shamir's $(t+1, \ell)$-threshold scheme perfectly hides the secret from coalitions of up to t shareholders; thus, no amount of post-processing—including, of course, multiplication by a permutation matrix—will allow coalition members to extract any information about the user's query.

Definition 3.4 explicitly permits indexes of queries through which (owing to the presence of all-0 columns) it is impossible to access certain blocks from $\mathbf{D}$, thus potentially making the information leakage *explicit*. In this subsection, we briefly revisit this information leakage—seemingly a weakness of simple indexes of queries—and spin it as a potentially useful feature.

In particular, in use cases where certain implicit leakage is tolerable (or even inevitable), it is possible to reduce the cost of PIR queries by *explicitly leaking the exact same information*. Trading off some (limited and controlled) information leakage in exchange for more efficient and expressive PIR queries is not without precedence; for example, both of Olumofin and Goldberg's [21] and Wang, Yun, Goldwasser, Vaikuntanathan, and Zaharia's [27] SQL-based PIR queries leak the "shape" of a query while hiding its sensitive constants. Learning the shape of an SQL query may betray information about the possible (and likely) constants in a way analogous to indexes of queries; however, as is the case with our indexes of queries, quantifying precisely how much information is leaked (and how troubling this leakage is) remains highly application-dependent. To make our explicit-leakage proposal more concrete, we return to the earlier example of requesting emails by `size` and suppose that, due to the context in which indexes of queries are being employed, the servers can immediately deduce that any email requested by `size` resides in the "large" tail of the `size` distribution (and yet, for the sake of the example, that such leakage is deemed acceptable). In this case, it is possible to support queries by `size` much more efficiently if we replace Π_{size} with a matrix through which only emails in the "large" tail are actually accessible.

This involves deleting each row of Π_{size} that corresponds to an email not in the "large" tail of the `size` distribution, resulting in a rectangular *pseudo-permutation matrix*; that is, in a $p \times r$ matrix that has *at most* one 1 in each row and each column and 0s elsewhere. Thus, we end up with a short-and-fat $(0,1)$-matrix having *full rank* (i.e., rank p). For instance, the three `largest` emails in $\mathbf{D}$ could be accessed via

$$\Pi_{\mathsf{largest}} := \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The following analog of Corollary 3.2 applies.

Observation 3.5. *Let $\vec{\boldsymbol{e}} \in \mathbb{F}^p$ be a standard basic vector and let $\Pi \in \mathbb{F}^{p\times r}$ be a pseudo-permutation matrix with rank p. If $(x_1, \vec{\mathbf{Q}}_1), \dots, (x_\ell, \vec{\mathbf{Q}}_\ell)$ is a component-wise $(t+1,\ell)$-threshold sharing of $\vec{\boldsymbol{e}}$, then $(x_1, \vec{\mathbf{Q}}_1 \cdot \Pi), \dots, (x_\ell, \vec{\mathbf{Q}}_\ell \cdot \Pi)$ is a component-wise $(t+1,\ell)$- and $(1,\ell)$-threshold sharing of a standard basis vector $\vec{\boldsymbol{e}}\,' \in \mathbb{F}^r$; namely, of $\vec{\boldsymbol{e}}\,' = \vec{\boldsymbol{e}} \cdot \Pi$.*[3]

Intuitively, Observation 3.5 implies that a t-private IT-PIR query through a short-and-fat pseudo-permutation matrix yields a t-private IT-PIR query *over a non-hidden subset of a larger database*. Specifically, such a matrix $\Pi \in \mathbb{F}^{p\times r}$ necessarily contains $r-p$ all-0 columns; consequently, every pseudo-permuted share vector $(x_j, \vec{\mathbf{Q}}_j \cdot \Pi)$ has $r-p$ corresponding 0 entries, which means queries through Π cannot fetch blocks corresponding to the all-0 columns in Π. Note that anyone can deduce the set of unfetchable blocks by inspecting Π (or a pseudo-permuted query vector $\vec{\mathbf{Q}}_j \cdot \Pi$), and it is in this sense that Π explicitly leaks information: it explicitly leaks that the request is for a block "indexed by some query" in Π.

[3]Specifically, the $r-p$ entries corresponding to all-0 columns in Π are $(1,\ell)$-threshold shares of 0; the remaining p entries are each $(t+1,\ell)$-threshold shares of either 0 or 1.

The upshot of explicitly leaking this information is twofold. First, the query vectors become shorter (their lengths correspond to the number of queries p in Π, rather than to the number of blocks r in $\mathbf{D}$); thus, each request incurs strictly lower upstream communication cost (p group elements) compared to a positional query over $\mathbf{D}$ (r group elements). Second, because each pseudo-permuted query vector $\vec{\mathbf{Q}}_j' := \vec{\mathbf{Q}}_j \cdot \Pi$ has support of size p, the vector-matrix product $\vec{\mathbf{Q}}_j' \cdot \mathbf{D}$ incurs strictly lower computation cost ($\approx 2ps$ field operations) compared to a positional query over $\mathbf{D}$ ($\approx 2rs$ field operations). We also stress that whatever information does leak is known *a priori* to the user; i.e., although queries leak some information, they do so transparently—there are no surprises.

3.4 Privacy in the face of implicit and explicit information leakage

In the preceding subsection, we claimed that a t-private query through a simple index of queries Π is, at least in some sense, still t-private. Formally proving that this is indeed the case necessitates a slight (though natural) modification to the standard definition of t-privacy. In particular, a direct application of the standard definition would require, for every coalition $S \subseteq [1..\ell]$ of at most t servers and for every record index $i \in [1..r]$, that

$$\Pr[I = i \mid Q_S = (\Pi; \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})] = \Pr[I = i], \tag{1}$$

where I and Q_S denote the random variables respectively describing the block index the user requests and the joint distribution of share vectors it sends to servers in S (including the "hint" that the query should go through Π).

However, it is evident that Equation (1) need not hold, for example, when the block in row i of $\mathbf{D}$ is not accessible through Π. It would not suffice to merely restrict the quantifier so that I ranges only over the subset of block indices which are accessible through Π; indeed, there may be several distinct indexes of queries, each inducing its own *conditional distribution* for I. In other words, a correct definition must account for the fact that curious PIR servers will inevitably—upon learning that a given request is through a particular index of queries Π—leverage this information to update their priors. The following modified t-privacy definition captures this idea.

Definition 3.6. Let $\mathbf{D} \in \mathbb{F}^{r\times s}$ and let each $\Pi_1, \dots, \Pi_n$ be an index of queries[4] for $\mathbf{D}$. Requests are *t-private with respect to* $\Pi_1, \dots, \Pi_n$ if, for every coalition $S \subseteq [1..\ell]$ of at most t servers, for every record index $i \in [1..r]$, and for every index of queries $\Pi \in \{\Pi_1, \dots, \Pi_n\}$,

$$\Pr[I = i \mid Q_S = (\Pi; \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})] = \Pr[I = i \mid E_\Pi],$$

where I and Q_S denote the random variables respectively describing the block index the user requests and the joint distribution of query vectors it sends to servers in S (including the "hint" that the query should go through Π), and where E_Π is the event that the request is through Π.

[4]Our omission of the word "simple" here is intentional: each Π_j can either be simple indexes of queries or one of the more sophisticated types we introduce in the sequel. In particular, by allowing some or all of the Π_j to be different kinds of indexes of queries, we can use Definition 3.6 to define privacy for all constructions in this paper.

Observe that a t-private query through a simple index of queries Π is functionally equivalent to—ergo, provides the exact same privacy guarantee as—a t-private positional query over the database $\mathbf{D}_\Pi := \Pi \cdot \mathbf{D}$. Consequently, Definition 3.6 reduces to the usual t-privacy definition when $\Pi \in \mathbb{F}^{r \times r}$ is the identity matrix.

The next theorem follows from the above observation and the t-privacy of Goldberg's IT-PIR [13, 16].

Theorem 3.7. *Let $\mathbf{D} \in \mathbb{F}^{r\times s}$ and let each $\Pi_1, \dots, \Pi_n$ be a simple index of queries for $\mathbf{D}$. If $\Pi \in \{\Pi_1, \dots, \Pi_n\}$ with $\Pi \in \mathbb{F}^{p\times r}$ and if $(x_1, \vec{\mathbf{Q}}_1), \dots, (x_\ell, \vec{\mathbf{Q}}_\ell)$ is a component-wise $(t+1, \ell)$-threshold sharing of a standard basis vector $\vec{\boldsymbol{e}} \in \mathbb{F}^p$, then $(\Pi, x_1, \vec{\mathbf{Q}}_1), \dots, (\Pi, x_\ell, \vec{\mathbf{Q}}_\ell)$ is t-private with respect to $\Pi_1, \dots, \Pi_n$.*

Proof. Consider a coalition S comprising t servers. Fix $i \in [1..r]$ and $\Pi \in \{\Pi_1, \dots, \Pi_n\}$ with $\Pi \in \mathbb{F}^{p\times r}$, and let I, J, and Q_S respectively denote the random variables describing the index (within $\mathbf{D}$) of the block the user requests, the index of the standard basis vector the user actually encodes in its query, and the joint distribution of share vectors it sends to servers in S (including the "hint" that the query should go through the index Π).

As per Definition 3.6, we need to show that $\Pr[I = i \mid Q_S = (\Pi; \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})] = \Pr[I = i \mid E_\Pi]$, where E_Π denotes the event that the user's request is through Π. The key observation underlying the proof is that $\Pr[I = i \mid E_\Pi] = \Pr[\vec{\boldsymbol{e}}_J \cdot \Pi = \vec{\boldsymbol{e}}_i \mid E_\Pi]$ and $\Pr[I = i \mid Q_S = (\Pi; \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})] = \Pr[\vec{\boldsymbol{e}}_J \cdot \Pi = \vec{\boldsymbol{e}}_i \mid Q_S = (\Pi; \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})]$.

Hence, we have

$$\begin{aligned}\Pr\big[I = i \mid E_\Pi\big] &= \Pr\big[\vec{\boldsymbol{e}}_J \cdot \Pi = \vec{\boldsymbol{e}}_i \mid E_\Pi\big]\\ &= \Pr\big[\vec{\boldsymbol{e}}_J \cdot \Pi = \vec{\boldsymbol{e}}_i \mid Q_S = (\Pi; \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})\big]\\ &= \Pr\big[I = i \mid Q_S = (\Pi; \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})\big],\end{aligned}$$

as desired. Note that the second line of the above derivation follows immediately from the t-privacy of $(x_1, \vec{\mathbf{Q}}_1), \dots, (x_\ell, \vec{\mathbf{Q}}_\ell)$. □

4 BATCH INDEXES OF QUERIES

In the preceding section, we discussed how requests through a simple index of queries can leak meta-information about which blocks the user seeks. We now turn our attention to our first non-trivial indexes of queries, called *batch indexes of queries*, which improve on simple indexes of queries by leveraging ideas from coding theory to decrease this information leakage.

Suppose we wish to leverage simple indexes of queries for an application in which the servers will hold multiple indexes of queries intended to facilitate different kinds of requests, yet in which all requests always will go through one of these indexes of queries. In this case, knowing only that a given request passed through *some* index of queries yields no information for an attacker: for information about a request to leak, the attacker would have to learn through *which* index of queries that request passed. Thus, concealing through which index of queries each request passes would effectively eliminate this source of information leakage while maintaining the utility that indexes of queries provide.

For example, suppose that a server believes *a priori* that a given request will pass through the index Π_1 with probability p and that it will pass through the index Π_2 with probability $1-p$, so that

$$\Pr\big[I = i\big] = \Pr\big[I = i \mid E_{\Pi_1}\big] \cdot p + \Pr\big[I = i \mid E_{\Pi_2}\big] \cdot (1-p).$$

In this case, if the server receives the hint "1", then it can immediately update its priors to conclude that $\Pr\big[I = i\big] = \Pr\big[I = i \mid E_{\Pi_1}\big]$; thus, the hint "1" in this example is leaking information about which block the client is fetching. On the other hand, if the client were somehow able to route its request through Π_1 without revealing through which of Π_1 or Π_2 its request is passing, then the server would be unable to update its priors and the request would leak no new information.

Even in cases where, say, some queries through an index and others are positional, and hence bypass the indexes of queries altogether, hiding through which index a given non-positional request passes would still serve to reduce the quantity of information that leaks. Batch indexes of queries provide one such way to hide through which out of several simple indexes of queries a given request passes.

The batch-indexes-of-queries construction we present here is specific to Goldberg's IT-PIR, leveraging the so-called "u-ary family of codes" [16, §3]. One could of course consider analogous instantiations for other PIR protocols or based on other codes; however, we leave exploration of this idea to future work. Before proceeding, we briefly review u-ary codes and how they are used to construct efficient IT-PIR protocols.

4.1 IT-PIR from u-ary codes

Recall that, in the vector-matrix model for PIR (as expounded in Section 2), each server typically holds a complete, plaintext replica of the database $\mathbf{D}$. Several recent IT-PIR constructions [2, 6, 12, 16] have instead considered a generalization of the vector-matrix model wherein each server holds an encoded *bucket* that is merely derived from—and typically much smaller than—the actual database $\mathbf{D}$. The benefits of this bucketized vector-matrix approach echo the benefits of explicit leakage described in the previous section: smaller buckets directly translate into lower upstream communication, lower per-server computation costs, and lower per-server storage costs.

One recently proposed construction in the bucketized vector-matrix model modifies Goldberg's IT-PIR protocol to utilize what is called the *u-ary family of codes* [16, §3 and §5]. In this scheme, each bucket is a matrix of (0-private) "shares" obtained using a "rampified" variant of Shamir's $(1, \ell)$-threshold scheme: given $u \in \mathbb{N}$, the u-ary code encodes $\mathbf{D} \in \mathbb{F}^{r\times s}$ by (i) partitioning the r blocks comprising $\mathbf{D}$ into r/u many u-tuples,[5] (ii) interpolating component-wise through each u-tuple at some predefined x-coordinates to obtain r/u many length-s vectors of degree-$(u-1)$ polynomials from $\mathbb{F}[x]$, and then (iii) placing a single component-wise evaluation of each vector of polynomials into each of the $\ell > u$ buckets. Thus, the bucket held by each of the ℓ servers resides in $\mathbb{F}^{(r/u)\times s}$ and, in particular, is a factor u smaller than $\mathbf{D}$.

Despite no server actually holding $\mathbf{D}$, users can still fetch blocks of $\mathbf{D}$ using slightly modified t-private positional queries over the buckets. Specifically, a user desiring the ith block from $\mathbf{D}$ simply needs to determine (i) which of the r/u bucket rows holds evaluations of the polynomial vector passing through the desired

[5]For ease of exposition, here and throughout we make the simplifying assumption that $u \mid r$. Eliminating this assumption is trivial, but doing so would only serve to introduce unnecessary clutter and exceptional cases to our notation and the descriptions of our constructions.

block, and (ii) at which x-coordinate that polynomial vector passes through the desired block. It then constructs a length-(r/u) vector of $(t+1,\ell)$-threshold shares encoding a positional query for the above bucket row *at the above x-coordinate* (in contrast to always encoding the positional query at $x=0$, as it would typically do in Goldberg's protocol). The rest of the protocol is exactly as in the standard vector-matrix model, except that, in the secret reconstruction step, the user interpolates the servers' responses to the same x-coordinate it used to encode its request. The result is a t-private v-Byzantine-robust (m,ℓ)-server IT-PIR protocol for any $m \geq t+u$ [16, Theorem 1] and $v \leq m-t-u-1$ [16, Theorem 2]. In particular, note that the number of servers, the privacy threshold, and the downstream communication cost are each identical to in Goldberg's protocol, whereas the upstream communication cost, the storage cost, and the server-side computation cost are all a factor u lower. (The tradeoff for the latter improvements is a reduction by u in the protocol's robustness to non-responding and Byzantine servers.) For additional details and proofs, we refer the reader to the original paper [16].

There are at least two ways to improve on our simple indexes of queries using u-ary codes. The most obvious way is to encode a simple index of queries into u-ary buckets, thereby reducing the upstream communication, and possibly the computation cost, associated with queries through that index. This slightly improves efficiency (though, as we will see in Section 7, indexes of queries are already plenty fast), but it does nothing to address information leakage. The remainder of this section deals with a more interesting approach that combines multiple disparate indexes of queries into a single batch index, thereby reducing information leakage by letting each user query $\mathbf{D}$ through the index of its choice *without revealing which particular index of queries it uses.* The idea is to merge all the indexes of queries into a single matrix of polynomials using component-wise polynomial interpolation (á la the above u-ary codes), so that each server holds only a single bucket obtained via component-wise evaluation of the resulting polynomial matrix. Users can then formulate requests through any of the constituent indexes of queries using appropriately crafted queries over the buckets, all the while concealing through which of the underlying simple indexes of queries their requests pass. Before formalizing this idea in Section 4.3, we walk through the process of merging the simple indexes of queries Π_{sender} and Π_{size} from Section 3.1.

4.2 Batching two indexes of queries

Recall that Π_{sender} and Π_{size} are the permutation matrices that respectively map a request encoding the ith standard basis vector of $\mathbb{F}^5$ to a positional query for the ith email in a lexicographic ordering of the email inbox $\mathbf{D}$ depicted in Figure 1 by `sender` or a numerical (decreasing) ordering of $\mathbf{D}$ by `size`. They are defined as

$$\Pi_{\text{sender}} := \begin{pmatrix} 0&0&1&0&0\\ 1&0&0&0&0\\ 0&0&0&1&0\\ 0&1&0&0&0\\ 0&0&0&0&1 \end{pmatrix} \text{ and } \Pi_{\text{size}} := \begin{pmatrix} 0&0&1&0&0\\ 0&1&0&0&0\\ 1&0&0&0&0\\ 0&0&0&0&1\\ 0&0&0&1&0 \end{pmatrix}.$$

The merging process is simple. We first fix some x-coordinates, say $x=0$ and $x=1$, which serve as identifiers for Π_{sender} and Π_{size}. Then, for each entry in Π_{sender}, we interpolate through that entry (at $x=0$) and the corresponding entry of Π_{size} (at $x=1$) to obtain a linear polynomial in $\mathbb{F}[x]$. As both Π_{sender} and Π_{size} are $(0,1)$-matrices, only four polynomials can arise in this step (corresponding to the pairs $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$); i.e., every interpolation yields one of $f_{00}(x)=0$, $f_{01}(x)=x$, $f_{10}(x)=1-x$, or $f_{11}(x)=1$.

Carrying out this process for Π_{sender} and Π_{size} yields

$$\Pi_{\text{sender,size}}(x) := \begin{pmatrix} 0&0&1&0&0\\ 1-x&x&0&0&0\\ x&0&0&1-x&0\\ 0&1-x&0&0&x\\ 0&0&0&x&1-x \end{pmatrix} \in \big(\mathbb{F}[x]\big)^{5\times 5}.$$

One can verify that evaluating $\Pi_{\text{sender,size}}(x)$ component-wise at $x=0$ and $x=1$ recovers Π_{sender} and Π_{size}, respectively; indeed, computing the vector-matrix product of $\Pi_{\text{sender,size}}(x)$ with $\vec{\boldsymbol{e}}_i \in \mathbb{F}^5$ and then evaluating the result at $x=0$ and $x=1$ yields the ith rows from Π_{size} and Π_{sender}, respectively. For example, $\vec{\boldsymbol{e}}_3 \cdot \Pi_{\text{sender,size}}(x) = \langle x\ 0\ 0\ 1{-}x\ 0 \rangle$, which evaluates to $\vec{\boldsymbol{e}}_3 \in \mathbb{F}^5$ and $\vec{\boldsymbol{e}}_1 \in \mathbb{F}^5$ at $x=0$ and $x=1$.

Let $x_1,\ldots,x_\ell \in \mathbb{F}\setminus\{0,1\}$ be arbitrary, pairwise distinct scalars. The bucket held by each server j will be obtained via component-wise evaluation of $\Pi_{\text{sender,size}}(x)$ at $x=x_j$. Thus, to fetch the ith email in a lexicographic ordering of $\mathbf{D}$ by `sender`, the user will "encode at $x=0$" the standard basis vector $\vec{\boldsymbol{e}}_i \in \mathbb{F}^5$ into ℓ vectors, $(x_1,\vec{\mathbf{Q}}_1^{(0)}),\ldots,(x_\ell,\vec{\mathbf{Q}}_\ell^{(0)})$, of $(t+1,\ell)$-threshold shares; specifically, it will select a length-5 vector of degree-t polynomials from $\mathbb{F}[x]$ uniformly at random, subject only to the requirement that this vector passes component-wise through $\vec{\boldsymbol{e}}_i$ at $x=0$, and then it will send to each server j the component-wise evaluation $\vec{\mathbf{Q}}_j^{(0)}$ of this vector at $x=x_j$. Likewise, to fetch the ith email in a numerical (decreasing) ordering of $\mathbf{D}$ by `size`, the user will "encode at $x=1$" the same standard basis vector $\vec{\boldsymbol{e}}_i \in \mathbb{F}^5$ into ℓ vectors $(x_1,\vec{\mathbf{Q}}_1^{(1)}),\ldots,(x_\ell,\vec{\mathbf{Q}}_\ell^{(1)})$ of $(t+1,\ell)$-threshold shares.

Notice that the only difference between how the user constructs the above two requests is the x-coordinate at which it encodes the standard basis vector $\vec{\boldsymbol{e}}_i$; thus, the x-coordinate here serves the same purpose that the hint, "`sender`" or "`size`", served back in Section 3.1, allowing the user to specify through which of the two indexes of queries its request is intended to pass. However, in contrast to with the hints that the user explicitly revealed in Section 3.1, from the perspective of any coalition of up to t servers, requests encoded at $x=0$ are perfectly indistinguishable from those encoded at $x=1$; that is, it is impossible for such a coalition to infer (based on the shares its members receive) through which of the two indexes of queries a given request passes [17].

Before we move on, a few brief remarks about this simple example are in order. First, we note that the resulting buckets are still quite sparse, having at most 2 non-zero entries in each row and in each column. Second, we observe that the t-privacy of requests through the buckets is still an immediate consequence of [16, Theorem 1]; indeed, the buckets are nothing more than 2-ary buckets of a database obtained by appropriately splicing together the indexes of queries Π_{sender} and Π_{size}. Finally, we point out that, owing to the fact that each entry of $\Pi_{\text{sender,size}}$ is an (at-most-)linear polynomial, reconstructing the servers' responses now requires one additional response. Hence, the protocol just described implements t-private (m,ℓ)-server IT-PIR for any $m > t+1$.

4.3 Batching u indexes of queries

Definition 4.1 formalizes a generalization of the construction from Section 4.2, which allows combining for arbitrarily many simple indexes of queries.

Definition 4.1. Fix $u > 1$ and let $x_1, \dots, x_\ell \in \mathbb{F} \setminus \{0, \dots, u-1\}$ be pairwise distinct scalars. A sequence $\Pi_1, \dots, \Pi_\ell \in \mathbb{F}^{p \times r}$ of matrices is a *u-batch index of queries* for Goldberg's IT-PIR with bucket coordinates $x_1, \dots, x_\ell$ if (i) $\Pi_{i_1} \neq \Pi_{i_2}$ for some $i_1, i_2 \in [1..\ell]$, and (ii) for each $j = 0, \dots, u-1$,

$$\pi_j := \sum_{i=1}^{\ell} \Pi_i \cdot \Big(\frac{j - x_1}{x_i - x_1}\Big) \cdots \Big(\frac{j - x_{i-1}}{x_i - x_{i-1}}\Big)\Big(\frac{j - x_{i+1}}{x_i - x_{i+1}}\Big) \cdots \Big(\frac{j - x_\ell}{x_i - x_\ell}\Big)$$

is a simple index of queries.

The first requirement of Definition 4.1, which insists that $\Pi_{i_1} \neq \Pi_{i_2}$ for some $i_1, i_2 \in [1..\ell]$, is a *non-triviality* requirement included merely to prevent simple indexes of queries from qualifying.[6] The second requirement is what captures the key property we intuitively desire from batch indexes of queries. The expression arising in that second requirement is just the familiar Lagrange interpolation formula. Intuitively, the definition says that the sequence of buckets $\Pi_1, \dots, \Pi_\ell$ is a u-batch index of queries if interpolating component-wise through the Π_i at each $x = 0, \dots, u-1$ yields a length-u sequence of simple indexes of queries. The restriction that $x_1, \dots, x_\ell$ be elements of $\mathbb{F} \setminus \{0, \dots, u-1\}$ is necessary to guarantee that users can actually request blocks through the constituent simple indexes of queries without betraying the privacy of their requests (see [16, proof of Theorem 1]).

The next theorem follows easily from [16, Theorems 1&2].

Theorem 4.2. Fix $u > 1$ and $j \in [0..u-1]$, and let $\Pi = (\Pi_1, \dots, \Pi_\ell) \in (\mathbb{F}^{p \times r})^\ell$ be buckets of a u-batch index of queries with bucket coordinates $x_1, \dots, x_\ell \in \mathbb{F} \setminus \{0, \dots, u-1\}$. If $(x_1, \vec{\mathbf{Q}}_{j_1}), \dots, (x_\ell, \vec{\mathbf{Q}}_{j_\ell})$ is a sequence of component-wise $(t+1, \ell)$-threshold shares of a standard basis vector $\vec{\boldsymbol{e}} \in \mathbb{F}^p$ encoded at $x = j$, then $(\Pi, x_1, \vec{\mathbf{Q}}_{j_1}), \dots, (\Pi, x_\ell, \vec{\mathbf{Q}}_{j_\ell})$ is t-private with respect to Π.

Proof. The proof of this theorem is nearly identical to that of Theorem 3.7. Consider a coalition S comprising t servers. Fix $i \in [1..r]$ and let I, J, K, and Q_S respectively denote the random variables describing the index (within $\mathbf{D}$) of the block the user requests, the index of the standard basis vector the user actually encodes in its query, the x-coordinate at which it encodes that standard basis vector, and the joint distribution of share vectors it sends to servers in S (including the "hint" that the query should go through the u-batch index of queries Π).

As per Definition 3.6, we need to show that $\Pr[I = i \mid Q_S = (\Pi;\, \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})] = \Pr[I = i \mid E_\Pi]$, where E_Π denotes the event that the user's request is through Π. The key observation is that $\Pr[I = i \mid E_\Pi] = \sum_{k=0}^{u-1} \Pr[\vec{\boldsymbol{e}}_J \cdot \pi_k = \vec{\boldsymbol{e}}_i \mid K = k, E_\Pi] \cdot \Pr[K = k \mid E_\Pi]$ and $\Pr[I = i \mid Q_S = (\Pi;\, \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})] = \sum_{k=0}^{u-1} \Pr[\vec{\boldsymbol{e}}_J \cdot \pi_k = \vec{\boldsymbol{e}}_i \mid K = k, Q_S = (\Pi;\, \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})] \cdot \Pr[K = k \mid E_\Pi]$. Hence, we have

$$\begin{aligned}\Pr[I = i \mid E_\Pi] &= \sum_{k=0}^{u-1} \Pr[\vec{\boldsymbol{e}}_J \cdot \pi_k = \vec{\boldsymbol{e}}_i \mid K = k, E_\Pi] \cdot \Pr[K = k \mid E_\Pi] \\ &= \sum_{k=0}^{u-1} \Pr[\vec{\boldsymbol{e}}_J \cdot \pi_k = \vec{\boldsymbol{e}}_i \mid K = k, Q_S = (\Pi;\, \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})] \\ &\qquad \cdot \Pr[K = k \mid E_\Pi] \\ &= \Pr[I = i \mid Q_S = (\Pi;\, \vec{\mathbf{Q}}_{j_1}, \dots, \vec{\mathbf{Q}}_{j_t})],\end{aligned}$$

as desired. Note that the second line of the above derivation follows immediately from the t-privacy of $(x_1, \vec{\mathbf{Q}}_1), \dots, (x_\ell, \vec{\mathbf{Q}}_\ell)$. □

Corollary 4.3. The construction just described implements t-private v-Byzantine-robust (m, ℓ)-server IT-PIR for any $m \geq t + u$.

In each of the following results, when we speak of a "u-batch" index of queries, we are implicitly assuming that u is the largest value for which Definition 4.1 is satisfied—i.e., that

$$\pi_u := \sum_{i=1}^{\ell} \Pi_i \cdot \Big(\frac{u - x_1}{x_i - x_1}\Big) \cdots \Big(\frac{u - x_{i-1}}{x_i - x_{i-1}}\Big)\Big(\frac{u - x_{i+1}}{x_i - x_{i+1}}\Big) \cdots \Big(\frac{u - x_\ell}{x_i - x_\ell}\Big)$$

is not another simple index of queries—and that the buckets have minimal degree in this regard. More precisely, we assume that interpolating through the buckets (at the indeterminate x) yields a matrix of polynomials each having degree at most $u-1$. We also point out that the results all hold for $u = 1$, provided we treat "1-batch index of queries" as synonymous with "simple index of queries". The first observation regards the sparsity of u-batch indexes of queries, while the second regards the possible values that their non-zero entries can take on.

Observation 4.4. Fix $u > 1$. If $\Pi_i \in \mathbb{F}^{p \times r}$ is a bucket of a u-batch index of queries, then the rows and columns of Π_i each contain at most u non-zero entries; hence, the total number of non-zero entries in Π_i is at most $\min(p, r) \cdot u$.

Observation 4.5. Fix $u > 1$. If $\Pi_i \in \mathbb{F}^{p \times r}$ is a bucket of a u-batch index of queries, then there exists a set S comprising at most $2^u - 1$ scalars from $\mathbb{F}$ such that every non-zero element in Π_i is an element of S.

Both observations are trivial to prove: it suffices to note that all entries in a bucket are y-coordinates of points on polynomials obtained via interpolating through the u values that reside in corresponding coordinates of the u constituent pseudo-permutation matrices. When all u components are 0, interpolation yields the zero polynomial (Observation 4.4); in all cases, every polynomial corresponds to a particular non-zero u-bit binary string.

5 INDEXES OF BATCH QUERIES

In the previous section, we proposed batch indexes of queries as a way to obtain all the benefits of simple indexes of queries but with improved privacy guarantees. We now turn our attention to a special kind of batch indexes of queries, called *indexes of batch queries*, which improve on the earlier batch indexes of queries by enabling users to fetch several related blocks (i.e., a batch of related blocks) with a single request.

Suppose we wish to leverage indexes of queries for an application in which typical requests seek the best k matches for some

[6] Omitting the non-triviality requirement would mean that whenever $\Pi \in \mathbb{F}^{p \times r}$ is a simple index of queries, the sequence of buckets $\Pi, \Pi, \dots, \Pi \in \mathbb{F}^{p \times r}$ is a u-batch index of queries for every $u \geq 1$. Clearly, this fails to jibe with what we intuitively mean by "u-batch" index of queries.

search term z. An obvious straw man construction would involve creating, for each possible search term z, a simple index of queries $\Pi_z \in \mathbb{F}^{k\times r}$ whose k rows are positional queries for the best k matches for that z. Unfortunately, this trivial solution offers little privacy: knowing which simple index of queries a user's requests go through immediately reveals precisely which blocks those requests are for. In theory, merging all of the simple indexes of queries into a batch index of queries would eliminate this leakage, but this approach does not scale; indeed, several of the motivating uses cases from Section 1 require best k queries involving *millions of possible search terms*, which would require millions of buckets held by millions of non-colluding servers! Indexes of batch queries provide an alternative construction that facilitates such requests supporting many—perhaps *millions* of—search terms much more efficiently and without requiring a large number of servers.

5.1 IT-PIR with k-batch queries

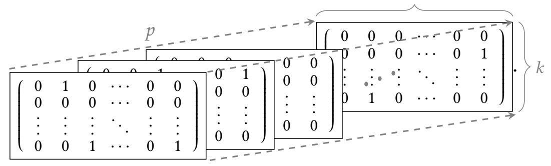
Recall that in the vector-matrix model for PIR, a typical request takes the form of a positional query represented by a standard basis vector. In the case of Goldberg's IT-PIR, the querier encodes this vector component-wise into ℓ vectors of $(t+1,\ell)$-threshold shares, and then it sends one such vector of shares to each of ℓ servers; thus, a user seeking the blocks referenced by the k rows of one of the simple indexes of queries $\Pi_z \in \mathbb{F}^{k\times r}$ from our straw man construction would need to make k separate requests, respectively encoding the standard basis vectors $\vec{e}_1,\dots,\vec{e}_k \in \mathbb{F}^k$. Of course, as we already noted, such a user should not expect any privacy.

Henry, Huang, and Goldberg [17] proposed *k-batch queries* as a more efficient way to request k blocks at once. Their k-batch queries are based on the same idea as u-ary codes: instead of encoding each basis vector $\vec{e}_1,\dots,\vec{e}_k$ in a separate request, a k-batch query encodes them all in a single request using $(t+1,\ell)$-threshold ramp shares, much like we saw in Section 4. Specifically, the user selects a length-k vector of degree-$(t+k-1)$ polynomials uniformly at random, subject to the requirement that, for each $i=1,\dots,k$, the vector passes component-wise through $\vec{e}_i$ at $x=i-1$. Nothing changes from the perspective of the servers[7] and yet a little algebra establishes that, if such a request passes through the simple index of queries $\Pi_z \in \mathbb{F}^{k\times r}$ to a database $\mathbf{D} \in \mathbb{F}^{r\times s}$, then the servers' responses reconstruct to $\vec{e}_1 \cdot \Pi_z \cdot \mathbf{D}$ at $x=0$, to $\vec{e}_2 \cdot \Pi_z \cdot \mathbf{D}$ at $x=1$, and so on up to $\vec{e}_k \cdot \Pi_z \cdot \mathbf{D}$ at $x=k-1$. Of course, the user should still not expect any privacy; we have only succeeded in making the non-private solution more efficient.

Whereas k-batch queries commingle effortlessly with simple indexes of queries, some technicalities interfere when one attempts to naïvely perform k-batch queries through batch indexes of queries (cf. [16, §5]), due to the way batch indexes of queries associate their constituent simple indexes of queries with specific x-coordinates.

5.2 k-batch queryable batch indexes of queries

Our indexes of batch queries are essentially just k-batch indexes of queries that have been constructed so as to map specific k-batch queries into other, meaningful k-batch queries over $\mathbf{D}$. Conceptually, we "transpose" the impractical straw man construction that began this section in a way that makes the best k queries for each search term z occupy a single row of a k-batch index of queries, at k pairwise distinct x-coordinates. To see how this works, it is helpful to think of the buckets comprising a k-batch index of queries as 2-dimensional projections of a particular 3-dimensional matrix; for instance, if there are p possible search terms z, then the p-batch index of queries arising from the straw man construction would be projections of a matrix Π residing in $\mathbb{F}^{k\times r\times p}$, say

$\Pi =$ 

Viewed in this way, it becomes apparent that we should transpose Π with respect to its height (k) and depth (p) axes. Doing so yields a matrix $\Pi' \in \mathbb{F}^{p\times r\times k}$ wherein, for each $i=1,\dots,p$ and $j=1,\dots,k$, the ith "plane" corresponds to a specific search term z_i in which the vector intersecting the "layer" at depth j holds a positional query for the jth-best matching block for z_i in $\mathbf{D}$. Each server will then hold one bucket from a "layer-wise" k-ary encoding of Π'. To fetch the best k matches for search term z_i, the user will simply encode k copies of the standard basis vector $\vec{e}_i$ in a t-private k-batch query, at $x=0$, at $x=1$, and so on up to $x=k-1$. We emphasize that the user here encodes *the same basis vector* at each of $x=0,\dots,k-1$. In a typical k-batch query, encoding multiple copies of the same basis vector would provide no benefit to the user and would only unnecessarily reduce the query's Byzantine robustness. It is also worth noting that the user can choose to encode just $m<k$ copies of $\vec{e}_i$ at $x=0,\dots,m-1$ in order to fetch only the best m matches for search term z_i.

The following definition formalizes the above construction, while the theorem that proceeds it addresses the parameters of IT-PIR protocol queries through such an index.

Definition 5.1. Fix $k>1$ and let $x_1,\dots,x_\ell \in \mathbb{F}\setminus\{0,\dots,k-1\}$ be pairwise distinct scalars. A sequence $\Pi_1,\dots,\Pi_\ell \in \mathbb{F}^{p\times r}$ of matrices is an *index of k-batch queries* for Goldberg's IT-PIR with bucket coordinates $x_1,\dots,x_\ell$ if, for each $i=1,\dots,p$, the matrix

$$\begin{pmatrix} \pi_{i0} \\ \vdots \\ \pi_{i(k-1)} \end{pmatrix}$$

is a pseudo-permutation matrix, where, for each $j \in [0\,..\,k-1]$,

$$\pi_{ij} \coloneqq \vec{e}_i \cdot \sum_{n=1}^{\ell} \Pi_n \cdot \big(\tfrac{j-x_1}{x_n-x_1}\big)\cdots\big(\tfrac{j-x_{n-1}}{x_n-x_{n-1}}\big)\big(\tfrac{j-x_{n+1}}{x_n-x_{n+1}}\big)\cdots\big(\tfrac{j-x_\ell}{x_n-x_\ell}\big).$$

We emphasize that indexes of k-batch queries are a special case of k-batch indexes of queries; hence, Theorem 4.2 implies that t-private queries through an index of k-batch queries Π are t-private with respect to Π. In light of this, Theorem 5.2 is just a restatement of Corollary 4.3.

Theorem 5.2. The construction just described implements t-private v-Byzantine-robust (m,ℓ)-server IT-PIR for any $m \geq t+2k-1$ and $v \leq m-t-2k+1$.

[7] In fact, coalitions of up to t servers cannot distinguish k-batch from non-batch queries [18].

6 RELATED WORK

This section discusses the small body of existing literature on expressive PIR queries, describing how our new techniques relate to and differ from those prior works.

Keyword-based PIR queries. A technical report by Chor, Gilboa, and Naor [7] proposed a mechanism through which users can fetch blocks privately by specifying *keywords* of interest. Similar to our indexes of queries, they accomplish this by augmenting the database with an auxiliary data structure (a binary search tree, a trie, or a minimal perfect hash function) intended to help users translate keyword-based requests into positional PIR queries, which are ultimately handled by the underlying PIR protocol. Specifically, the user employs positional queries to obliviously traverse the auxiliary data structure (which, for tree-based data structures, may require many iterative queries) in order to learn the physical location within the database of some record of interest, which it eventually fetches using a final positional PIR query over the actual data.

In contrast to keyword-based PIR, indexes of queries let users fetch data in a single round of interaction and they do not reveal any information about the structure and layout of the underlying data set. Indeed, the communication and computation costs incurred while fetching records via an index of queries are decoupled from the number of blocks in the database and are, in fact, *upper bounded* by the cost of a positional PIR query over the database (such as the one occurring in the last step of Chor et al.'s scheme).

SQL-based PIR queries. Olumofin and Goldberg [21] extended Chor et al.'s approach to a scheme enabling users to fetch blocks privately using simple *SQL queries* filtered by WHERE or HAVING clauses. Similar to indexes of batch queries, Olumofin and Goldberg accomplish this by having the database operator prepare (perhaps several) inverted indexes that map sensitive search criteria to the physical locations of associated blocks in the database. Also similar to our approach, their technique may leak some information about which blocks a user seeks, as it hides the sensitive search terms that appear in a query but not the overall "shape" of the query.

Of course, because Olumofin and Goldberg's construction directly build on keyword-based PIR, the differences we highlighted above also differentiate our approach from theirs. Moreover, although a single SQL query in their model may return a batch consisting of several records, this comes at a cost of requiring the user to perform multiple positional queries against the underlying database (indeed, the user must always perform a number of queries corresponding to the maximum possible size of a response, so as to avoid leaking information about the actual size of the response); indexes of batch queries, by contrast, can return such batches in a single response using only a single query (and without leaking the size of the response).

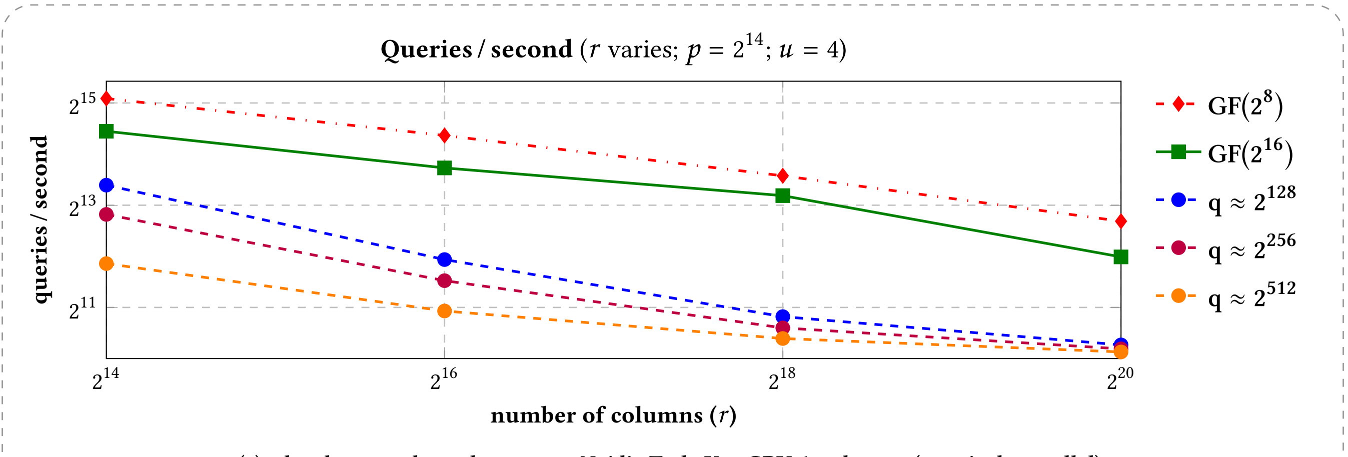
PIR from function secret sharing. In terms of functionality, our proposal is most directly comparable to the recent PIR protocols based on Boyle, Gilboa, and Ishai's *function secret sharing* (FSS) [3–5]. FSS provides a way for clients to split certain functions f into pairs of "function shares", which are themselves functions that can be evaluated at an input x to produce additive shares of $f(x)$. This enables the construction of expressive 2-server protocols with which users can fetch records privately using any search criteria expressible as a polynomial sized *branching program.* (FSS constructions that split functions into ℓ-tuples for $\ell > 2$ have also been proposed, thus yielding analogous ℓ-server PIR protocols, but these constructions are dramatically less efficient and require stronger computational assumptions compared to the 2-party construction.)

In contrast to our index-of-queries approach, FSS permits keyword searches without any need for the server to prepare auxiliary data structures. However, this added flexibility comes at a cost of stronger security assumptions and a (potentially) higher computation cost. Specifically, unlike the information-theoretic PIR underlying our approach, existing PIR protocols based on (2-party) FSS schemes (i) require a comparatively stronger non-collusion assumption (i.e., that there exists a pair of servers who may not collude), (ii) provide only computational security even when this maximally strong non-collusion assumption holds, and (iii) necessarily incur computational cost comparable to the *upper bound* on that of our index-of-queries approach.

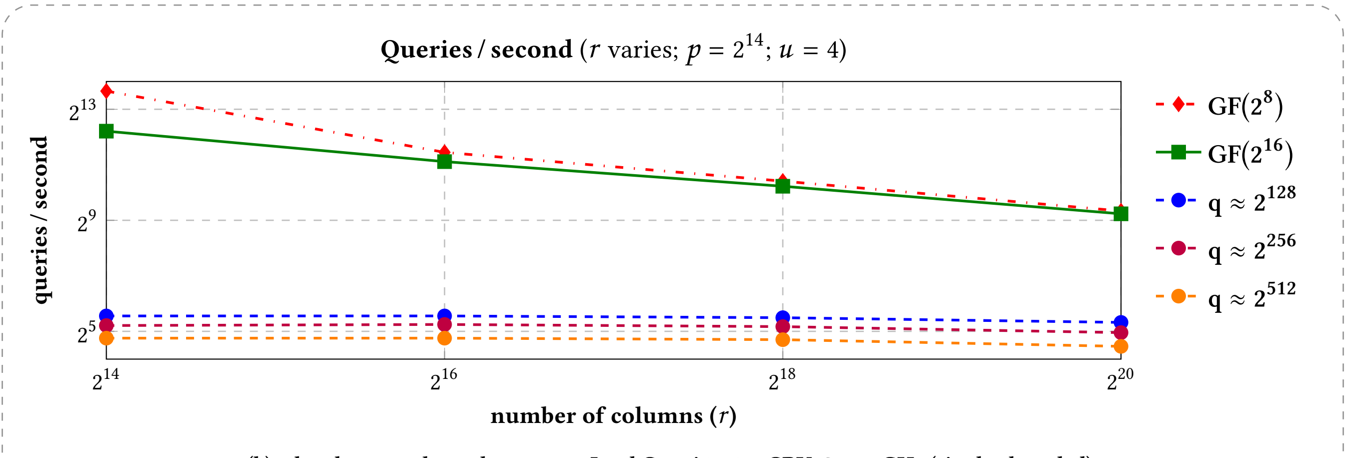
SQL-based PIR queries from FSS. A recent paper of Wang, Yun, Goldwasser, Vaikuntanathan, and Zaharia [27] proposed *Splinter*, a system that employs function secret sharing to support a range of queries comparable to those supported by Olumofin and Goldberg's SQL-based approach. Splinter provides both the best and worst of both worlds: on one hand, Splinter supports a similar set of queries as SQL-based PIR with improved performance (by replacing many recursive PIR-by-keyword queries with single-round FSS queries); on the other hand, it leaks the shape of queries (á la SQL-based PIR) and requires both computational assumptions and rigid non-collusion assumptions (á la FSS-based PIR).[8]

Despite the above benefits of indexes of queries over existing keyword-, and SQL-, and function secret sharing-based PIR approaches, there exist use cases in which the latter are more useful—each approach facilitates fundamentally different classes of interactions. Indeed, it is not obvious how to realize efficient keyword-based queries using indexes of queries alone, as this would require users to somehow learn which rows in the index correspond to which keywords. For instance, returning to our running private-inbox-queries example, we note that while many casual interactions with an email client leverage only the views naturally supportable with indexes of queries, users can and do frequently rely on keyword-based searches to locate emails of interest. Thus, an actual private email client would benefit from simultaneous support for both indexes of queries and keyword- or SQL-based queries. Fortunately, because none of the three approaches require any modification to the underlying database, no technical challenges prevent the servers from supporting all of them at the same time.

[8]Wang et al. assert that Olumofin and Goldberg's SQL-based PIR "requires all the providers to be honest" [27, §8.2]; however, this claim is false. Indeed, Olumofin and Goldberg provide the same degree of flexibility as our indexes of queries in choosing security assumptions: the default instantiation is unconditionally private provided at most t out of ℓ servers collude, for *any* choice of $t \leq \ell$ including, e.g., $\ell = t - 1$. (By contrast, existing FSS schemes, including those used by Splinter, *only* support $t = \ell - 1$ and, even then, only provide computational privacy against smaller coalitions.) Moreover, one can employ either computational or hybrid PIR in Olumofin and Golberg's framework, thus providing computational privacy when all servers collude and, indeed, even allowing the protocol to run with a single server. This can be accomplished under a wide variety of computational assumptions, including Paillier's decisional composite residuosity assumption (DCRA) or standard lattice assumption. Finally, setting $t < \ell - 1$ allows the former scheme to provide some level of Byzantine-robustness, which equates to better liveness and potential mitigation of active attacks by small coalitions of servers.



(a) 4-batch query throughput on an Nvidia Tesla K20 GPU Accelerator (massively parallel)



(b) 4-batch query throughput on an Intel Core i5-2500 CPU @ 3.30GHz (single-threaded)

Figure 2: Number of 4-batch index of queries requests our implementation can process per second. Figure 2(a) depicts the counts for a massively parallel implementation on an Nvidia Tesla K20 GPU Accelerator; Figure 2(b) depicts the same counts for a single-threaded implementation on an Intel Core i5-2500 CPU. Each experiment was repeated for 100 trials; we report here the mean number of requests per second. Error bars are omitted due to their small size (all standard deviations were below 2% of the mean).

7 IMPLEMENTATION AND EVALUATION

All three variants of indexes of queries introduced in this paper yield sparse matrices; thus, querying through an index of queries is an instance of sparse matrix-vector (SpMV) multiplication, an embarrassingly parallelizable workload that is particularly well suited to implementation on a massively parallel compute platform, such as a general-purpose GPU device.

In order to empirically gauge the practicality of our indexes-of-queries approach, we implemented finite-field SpMV multiplication and ran a series of experiments both on an Nvidia Tesla K20 GPU Accelerator [9] and on an Intel Core i5-2500 CPU. Both implementations support SpMV multiplication in the binary fields **GF**(2^8) and **GF**(2^{16}) and in $\mathbb{Z}_q$ for arbitrary multi-precision prime moduli q.[9] We use lookup tables and exclusive-ORs for fast binary field arithmetic; for prime-order field arithmetic, our GPU code uses a hand-optimized PTX implementation of “schoolbook” multiplication/addition together with Barrett reduction [1], while our CPU implementation outsources arithmetic to NTL [26] and GMP [11]. Our implementations are licensed under version 2 of the GNU

[9]Numerous efficient CUDA-based SpMV multiplication implementations already exist, yet essentially all implementations we found assume that the entries are floating-point numbers. Modifying any of these implementations to do integer arithmetic modulo a 32-bit word-size prime would be relatively straightforward; however, in order to obtain good PIR performance, we need support for SpMV multiplication over small binary fields and/or over prime-order fields with multiple-precision prime moduli. Indeed, benchmarks we ran on Percy++, an open-source implementation of Goldberg’s PIR protocols, indicate that the PIR over small binary fields is fastest, followed by PIR over prime order fields with moduli $\geq$ 128 bits long. For instance, we observe about a $3.5\times$ speedup switching from a 32-bit modulus to a 1024-bit modulus.

General Public License (GPLv2), and we are presently working to integrate both versions into Percy++ [14], an open-source implementation of Goldberg's IT-PIR protocol.

We conducted two sets of experiments. The first set of experiments consists of microbenchmarks designed to measure the latency imposed by routing PIR queries through an index of queries prior to conducting a positional query against the actual database. The second set of experiments evaluates the feasibility of deploying our techniques over a real-world dataset; specifically, we constructed several batch indexes of queries through which users can fetch academic articles posted to the *IACR Cryptology ePrint archive* [19].

7.1 SpMV microbenchmarks

For the first set of experiments, we generated a large number of random u-batch indexes of queries for various choices of u, index dimensions, and finite fields, and then we measured the number of SpMV operations we could evaluate per second, either as a massively parallel computation on our Nvidia K20 GPU Accelerator or as a single-threaded computation on our Intel Core i5-2500 CPU. The results of this experiment are unsurprising—our SpMV multiplications consistently run extremely fast, even when the indexes of queries have quite large dimensions.

In line with expectations, we observed that varying the height of the index (p) and the batching parameter (u) had very little impact on throughput for our GPU implementation,[10] whereas the throughput decreased linearly with pu for our CPU implementation.

Figure 2 plots the measurements we obtained from one arbitrary-yet-representative set of parameters; specifically, it shows the results for a sequence of indexes of 4-batch queries having $p = 2^{14}$ rows and mapping to databases $\mathbf{D}$ having between $r = 2^{14}$ and $r = 2^{20}$ blocks. In all cases, our GPU implementation was able to process well over a thousand requests per second (indeed, we found that memory bandwidth to and from the GPU was consistently the bottleneck); our CPU implementation was able to process between a few hundred (for $r = 2^{20}$) and a few thousand (for $r = 2^{14}$) requests per second in the binary fields and on the order of a few dozen requests per second (for all r) in large prime-order fields. In all cases, increasing r yielded a roughly linear decrease in throughput, with a slope inversely proportional to the cost of a single field operation.

For comparison, we found that it took just over 1.4 second per GiB of database (using a single thread) to process a single positional query using fast arithmetic in $\mathbf{GF}(2^8)$, with every other field we measured taking notably longer. Thus, we conclude that, even in the worst conceivable cases, indexes of queries introduce no significant latency to PIR requests (and, when $p \ll r$, they may significantly speed up the subsequent PIR processing by producing positional queries with small support).

[10]We ran experiments for various choices of $u \in [1\,..\,16]$ and power-of-two heights and widths, with dimensions ranging from extremely short-and-fat to perfectly square (but never tall-and-skinny); hence, our indexes were consistently extremely sparse (at most about 0.1% of entries were nonzero), causing most GPU threads to sit idle most of the time, regardless of how we set u and p.

7.2 IACR Cryptology ePrint Archive

For the second set of experiments, we created a dataset by scraping the *IACR Cryptology ePrint Archive* [19], an open-access repository that provides rapid access to recent research in cryptology. In particular, we scraped metadata (paper id, paper title, author list, submission date, keywords, and file size) for 10,181 papers (which was the entire dataset as of midday on February 10, 2017, excluding 60 papers that our scraper skipped over because of inconsistently formatted metadata). We also scraped citation counts for each paper in the dataset from *Google Scholar* [15].

Using this data, we constructed a "synthetic ePrint" database, in which the ith row holds a random bitstring whose length equals the file size of the ith paper in the actual ePrint dataset (padded with 0s to the length of the largest paper).[11] The largest paper in the dataset was 19.3 MiB, but only 56 out of the 10,181 papers exceeded 4.69 MiB; therefore, we pruned those 56 papers to obtain a dataset comprising 10,125 papers (the discarded manuscripts were predominantly older PostScript files). This resulted in a 46.35 GiB database (including the 0-padding) of chronologically sorted "synthetic ePrint papers" that user can fetch using IT-PIR queries.

We also constructed histograms to determine (i) the total number of papers associated with each keyword, and (ii) the total number of papers by each author. We identified 1,005 unique keywords that were associated with five or more distinct papers each, and 1,750 unique authors that were each associated with four or more distinct papers each, within the pruned dataset. From here, we constructed four different indexes of 4-batch queries over $\mathbf{GF}(2^8)$; namely, we created indexes of 4-batch queries supporting requests for the "4 most highly cited" and the "4 most recently posted" ePrint papers for each keyword (associated with at least 5 papers) and for each author (associated with at least 4 papers).

We performed two kinds of experiments for each of the four indexes of queries. Table 1 summarizes the results of these experiments, as well as some statistics about the time required to generate, and the storage requirements for, each index of queries. First, we measured the total number of requests through each of the four indexes of queries that both our Nvidia Tesla K20 GPU Accelerator and our Intel Core i5-2550 CPU could process per second; given their small dimensions and the choice of working over $\mathbf{GF}(2^8)$, in all cases we managed a whopping 49,000+ queries per second on the GPU and over 20,000 queries per second on a single core of the CPU. Second, we measured the total time required to retrieve a random paper from the dataset using a positional query output by each of the four indexes of queries. Because each of these indexes of queries contains a relatively large number of all-0 columns, the cost of the latter PIR step was substantially lower than that of a standard positional query. In particular, queries by keyword took around 19 seconds, on average, whereas queries by author took around 33 seconds, on average; by contrast, positional PIR queries over the entire database took nearly 70 seconds, on average. These measurements suggest that indexes of queries can indeed be a useful building block in the construction of practical PIR-based systems for datasets on the order of tens of GiB.

[11]We refrained from downloading all ePrint papers and instead opted for random data purely to avoid unduly burdening ePrint with a high volume of unnecessary downloads.

Table 1: Experimental results obtained for the IACR *Cryptology ePrint Archive* [19] dataset. The dataset consists of 10,181 academic papers and associated metadata. All timing experiments were repeated for 100 trials to obtain a standard deviation to one significant figure, and are reported to that precision (± the standard deviation).

Search criteria	Sort criteria	Simple index generation	Bucket generation (interp. + eval.)	Bucket size	# of nonempty columns	GPU index throughput (queries/sec)	CPU index throughput (queries/sec)	PIR throughput (secs/query)
Keyword ($p = 1005$)	Recency	1.5 ± 0.1 s	54 ± 9 ms	71.76 KiB	2692	49100 ± 100	32800 ± 400	19.1 ± 0.7 s
	Citations	1.3 ± 0.1 s	52 ± 10 ms	70.17 KiB	2645	49100 ± 100	30700 ± 600	18.8 ± 0.6 s
Author ($p = 1750$)	Recency	3.1 ± 0.1 s	63 ± 6 ms	92.38 KiB	4548	49100 ± 100	22700 ± 400	32.6 ± 0.8 s
	Citations	3.1 ± 0.2 s	63 ± 8 ms	91.69 KiB	4546	49000 ± 100	20100 ± 300	32.4 ± 0.9 s

8 CONCLUSION AND FUTURE WORK

We proposed indexes of queries, a novel mechanism for supporting efficient and expressive, single-round queries over multi-server PIR databases. Our approach decouples the way users construct their queries from the physical layout of the database, thereby enabling users to retrieve information using contextual queries that specify *which* data they seek, as opposed to position-based queries that specify *where* in the database those data happen to reside. We demonstrated the feasibility of at least one promising applications of our indexes-of-queries approach, and proposed several other compelling possibilities, which we believe present several exciting opportunities for future work.

Another potential avenue for future work is to explore the index of queries approach as it applies to other vector-matrix PIR protocols, which may lead to additional useful instantiations (e.g., eliminating non-collusion assumptions and compressing queries by settling for computational privacy). Likewise, it would be interesting to explore how other families of batch codes might yield alternative constructions for batch indexes of queries and indexes of batch queries, which may offer different tradeoffs or compatability with a wider range of PIR protocols.

Acknowledgements. This material is based upon work supported in part by the National Science Foundation under Grant No. 1718475, by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute, and by the Indiana METACyt Initiative. The Indiana METACyt Initiative at IU is also supported in part by Lilly Endowment, Inc.

REFERENCES

[1] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology: Proceedings of CRYPTO 1986*, volume 263 of *LNCS*, pages 311–323, Santa Barbara, CA, USA (August 1987).
[2] Simon Blackburn and Tuvi Etzion. PIR array codes with optimal PIR rate. *arXiv:CoRR*, abs/1607.00235 (July 2016).
[3] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Advances in Cryptology: Proceedings of EUROCRYPT 2015 (Part II)*, volume 9057 of *LNCS*, pages 337–367, Sofia, Bulgaria (April 2015).
[4] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *Advances in Cryptology: Proceedings of CRYPTO 2016 (Part I)*, volume 9814 of *LNCS*, pages 509–539, Santa Barbara, CA, USA (August 2016).
[5] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of CCS 2016*, pages 1292–1303, Vienna, Austria (October 2016).
[6] Terence H. Chan, Siu-Wai Ho, and Hirosuke Yamamoto. Private information retrieval for coded storage. In *Proceedings of ISIT 2015*, pages 2842–2846, Hong Kong (June 2015).
[7] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Technical Report CS 0917, Technion-Israel Institute of Technology, Haifa, Israel (February 1997).
[8] Henry Cohn and Nadia Heninger. Approximate common divisors via lattices. In *Proceedings of ANTS X (2012)*, volume 1, number 1 of *The Open Book Series*, pages 271–293, San Diego, CA, USA (July 2012).
[9] Nvidia Corporation. Tesla® Kepler$^{\text{tm}}$ GPU Accelerators. http://www.nvidia.com/content/tesla/pdf/Tesla-KSeries-Overview-LR.pdf. *(Accessed: February 16, 2017).*
[10] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally robust private information retrieval. In *Proceedings of USENIX Security 2012*, pages 269–283, Bellevue, WA, USA (August 2012).
[11] Torbjörn Granlund et al. GNU multiple precision arithmetic library; version 6.1.2 [computer software]. Available from: https://gmplib.org/ (December 2016).
[12] Arman Fazeli, Alexander Vardy, and Eitan Yaakobi. Codes for distributed PIR with low storage overhead. In *Proceedings of ISIT 2015*, pages 2852–2856 (June 2015).
[13] Ian Goldberg. Improving the robustness of private information retrieval. In *Proceedings of IEEE S&P 2007*, pages 131–148, Oakland, CA, USA (May 2007).
[14] Ian Goldberg, Casey Devet, Wouter Lueks, Ann Yang, Paul Hendry, and Ryan Henry. Percy++ / PIR in C++; version 1.0 [computer software]. Available from: git://git-crysp.uwaterloo.ca/percy (October 2014).
[15] Google Scholar. https://scholar.google.com/. *(Accessed: February 16, 2017).*
[16] Ryan Henry. Polynomial batch codes for efficient IT-PIR. In *Proceedings on Privacy Enhancing Technologies (PoPETS)*, volume 2016(4), pages 202–218, Darmstadt, Germany (July 2016).
[17] Ryan Henry, Kevin Henry, and Ian Goldberg. Making a Nymbler Nymble using VERBS. In *Proceedings of PETS 2010*, volume 6205 of *LNCS*, pages 111–129, Berlin, Germany (July 2010).
[18] Ryan Henry, Yizhou Huang, and Ian Goldberg. One (block) size fits all: PIR and SPIR with variable-length records via multi-block queries. In *Proceedings of NDSS 2013*, San Diego, CA, USA (February 2013).
[19] IACR Cryptology ePrint Archive. https://eprint.iacr.org/. *(Accessed: February 10, 2017).*
[20] Arvind Narayanan and Vitaly Shmatikov. Myths and fallacies of "personally identifiable information". *Communications of the ACM (CACM)*, 53(6):24–26 (June 2010).
[21] Femi G. Olumofin and Ian Goldberg. Privacy-preserving queries over relational databases. In *Proceedings of PETS 2010*, volume 6205 of *LNCS*, pages 75–92, Berlin, Germany (July 2010).
[22] Joel Reardon, Jeffrey Pound, and Ian Goldberg. Relational-complete private information retrieval. Technical Report CACR 2007-34, University of Waterloo, Waterloo, ON, Canada (December 2007).
[23] Irving S. Reed and Gustav Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, 8(2):300–304 (June 1960).
[24] Len Sassaman, Bram Cohen, and Nick Mathewson. The Pynchon Gate: A secure method of pseudonymous mail retrieval. In *Proceedings of WPES 2005*, pages 1–9, Alexandria, VA, USA (November 2005).
[25] Adi Shamir. How to share a secret. *Communications of the ACM (CACM)*, 22(11):612–613 (November 1979).
[26] Victor Shoup. NTL: A library for doing number theory; version 10.5.0 [computer software]. Available from: http://www.shoup.net/ntl (July 2017).
[27] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *Proceedings of NSDI 2017*, pages 299–313, Boston, MA, USA (March 2017).