Ryan Henry

Polynomial Batch Codes for Efficient IT-PIR

Abstract: Private information retrieval (PIR) is a way for clients to query a remote database without the database holder learning the clients' query terms or the responses they generate. Compelling applications for PIR are abound in the cryptographic and privacy research literature, yet existing PIR techniques are notoriously inefficient. Consequently, no such PIR-based application to date has seen real-world at-scale deployment. This paper proposes new "batch coding" techniques to help address PIR's efficiency problem. The new techniques exploit the connection between *ramp secret sharing schemes* and efficient *information-theoretically secure* PIR (IT-PIR) protocols. This connection was previously observed by Henry, Huang, and Goldberg (NDSS 2013), who used ramp schemes to construct efficient "batch queries" with which clients can fetch several database records for the same cost as fetching a single record using a standard, non-batch query. The new techniques in this paper generalize and extend those of Henry et al. to construct "batch codes" with which clients can fetch several records for only *a fraction* the cost of fetching a single record using a standard non-batch query over an unencoded database. The batch codes are highly tuneable, providing a means to trade off (i) lower server-side computation cost, (ii) lower server-side storage cost, and/or (iii) lower uni- or bi-directional communication cost, in exchange for a comparatively modest decrease in resilience to Byzantine database servers.

Keywords: Private information retrieval, batch codes, batch queries, ramp schemes, efficiency

DOI 10.1515/popets-2016-0036
Received 2016-02-29; revised 2016-06-02; accepted 2016-06-02.

1 Introduction

Private information retrieval (PIR) is a cryptographic primitive that solves the seemingly impossible problem of letting clients query a remote database without letting the database holder learn the clients' query terms or the responses they generate. PIR has received considerable attention from the cryptographic and privacy research communities since its introduction by Chor, Goldreich, Kushilevitz, and Sudan in 1995 [16], and compelling applications for PIR are abound in the cryptographic and privacy research literature. Alas, despite a series of significant advances over the past two decades, existing PIR techniques are notoriously inefficient [50]. Consequently, to date not one of the myriad PIR-based applications in the research literature has been deployed at-scale to protect the privacy of users "in the wild".

This paper proposes new techniques that considerably increase the practicality of certain multi-server *information-theoretic private information retrieval* (IT-PIR) protocols. The new techniques are based on *ramp schemes* [7] constructed from polynomials over finite fields. Jumping ahead, we find that using ramp schemes to encode not only requests for data (as in prior work [34, 39]), but also the data themselves, can dramatically reduce the server-side computation cost, per-server storage cost, and (upstream, downstream, and/or bidirectional) communication costs of IT-PIR. Specifically, we propose a way to encode a database into ramp shares so that each share encodes (a portion of) multiple records, and so that clients can fetch several records at once by combining multiple requests into a single ramp-encoded query. Given a fixed pool of database servers, we obtain much-improved performance with zero impact to privacy, in exchange for a comparatively modest decrease in robustness to Byzantine database servers. Alternatively, allowing the number of servers and/or collusion threshold to vary—thereby resulting in IT-PIR schemes with *incomparable* security guarantees—yields novel IT-PIR constructions with attractive asymptotic characteristics. We view these findings as a generalization and extension of Henry, Huang, and Goldberg's multi-block IT-PIR queries [34], which are likewise constructed from ramp schemes.

2 Preliminaries

Our new approach builds on several earlier techniques from the PIR literature. This section introduces those techniques, beginning with a brief description of the basic mathematical framework in which they all operate.

The PIR database is structured as an $r \times s$ matrix $\mathbf{D}$ over a finite field $\mathbb{F}$ in which each of the r rows represents one s-word *block* of data. We write $\mathbf{D}_{(i,j)}$ to denote the word in position (i, j) of $\mathbf{D}$. A block (i.e., row) is the basic unit of data that a client may fetch from $\mathbf{D}$, and the goal of PIR is to let clients fetch arbitrary blocks of their choosing without letting the servers learn *which* particular blocks they fetch. Formalizations of this intuitive privacy notion follow easily by considering either (i) the advantages of malicious PIR servers

Ryan Henry: Indiana University, E-mail: henry@indiana.edu

in an indistinguishability game [14], or (ii) indistinguishability among the ensembles of random variables that describe the views of PIR servers in "real" versus "simulated" protocol executions [10]. The most basic form of PIR only supports fetching blocks by their index (i.e., by their row number within $\mathbf{D}$); however, prior work shows how one can implement queries that are more expressive, such as retrieval-by-keyword [15] or simple SQL queries [42, 45], atop this basic primitive.

A simple if inefficient way to realize the above-described functionality with *perfect* privacy is to employ a "trivial download" strategy in which each client downloads a complete copy of $\mathbf{D}$ and then looks up any desired blocks locally. This approach is highly impractical when $\mathbf{D}$ is large; thus, another goal of PIR is to transmit strictly less data than the trivial download strategy (both *asymptotically* and *concretely* for "realistic" database and block sizes), while still concealing from the database servers which particular blocks the clients fetch. Formally, this property (called *non-triviality*) requires that the communication cost to fetch a block from $\mathbf{D} \in \mathbb{F}^{r \times s}$ scales as $o(rs)$ field elements. A trivial lower bound on the downstream communication cost of any PIR protocol is $s + O(1)$ field elements, the cost of a non-private fetch. A protocol that meets this lower bound is said to have *optimal downstream rate*. Much prior work [2, 9, 24, 27, 40, 52] has focused on optimizing the downstream rate of PIR (often to the detriment of the upstream communication and computation costs). The culmination of these efforts includes two recent PIR protocols—one *computationally private* [37] and the other *information-theoretically private* [47]—that achieve optimal downstream rate.

Non-private fetches from basic linear algebra

One *non-private* way to fetch a block from $\mathbf{D}$ with $r + s$ total communication cost is to have the client construct and send to the database holder a suitable vector from the "standard" orthonormal basis for $\mathbb{F}^r$. In particular, to fetch the ith block from $\mathbf{D}$, the client sends the length-r row vector $\vec{e}_i \in \mathbb{F}^r$ containing unity in its ith coordinate and zero elsewhere. (We herein refer to such a vector $\vec{e}_i$ simply as the "ith standard basis vector" in $\mathbb{F}^r$.) The server responds with the length-s row vector $\vec{\mathbf{D}}_i$ given by the vector-matrix product $\vec{e}_i \cdot \mathbf{D}$. A simple calculation confirms that $\vec{\mathbf{D}}_i$ is indeed equal to the ith row of $\mathbf{D}$:

$$\vec{e}_i \cdot \mathbf{D} = \langle 0 \;\; 0 \cdots 1 \cdots 0 \rangle \cdot \begin{pmatrix} \mathbf{D}_{(0,0)} & \mathbf{D}_{(0,1)} & \cdots & \mathbf{D}_{(0,s\text{-}1)} \\ \mathbf{D}_{(1,0)} & \mathbf{D}_{(1,1)} & \cdots & \mathbf{D}_{(1,s\text{-}1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{D}_{(i,0)} & \mathbf{D}_{(i,1)} & \cdots & \mathbf{D}_{(i,s\text{-}1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{D}_{(r\text{-}1,0)} & \mathbf{D}_{(r\text{-}1,1)} & \cdots & \mathbf{D}_{(r\text{-}1,s\text{-}1)} \end{pmatrix}$$
$$= \langle \mathbf{D}_{(i,0)} \;\; \mathbf{D}_{(i,1)} \;\; \cdots \;\; \mathbf{D}_{(i,s\text{-}1)} \rangle.$$

The PIR literature considers two basic approaches for turning such vector-based fetches into private queries.

Private fetches from homomorphic encryption

The first approach for making the above vector-based fetches private uses partially homomorphic encryption. In this approach, the client encrypts the vector $\vec{e}_i$ component-wise using a semantically secure, additively homomorphic encryption scheme [11] (for instance, Paillier [44] or Damgård and Jurik's generalization thereof [19]),[1] and then it sends the encrypted vector to the database server. Since the vector-matrix product $\vec{e}_i \cdot \mathbf{D}$ is nothing more than a sequence of linear combinations with coefficients from $\mathbf{D}$, the database server (holding a *plaintext* copy of $\mathbf{D}$) can use the additive homomorphism of the encryption scheme to obliviously compute and return to the client a component-wise encryption of $\vec{\mathbf{D}}_i$.

The privacy guarantees of such encryption-based PIR follow directly from the IND-CPA security of the underlying encryption scheme against computationally bounded attackers; thus, such protocols are called *computationally secure PIR* (CPIR) protocols. Unfortunately, this encryption-based approach imposes very high computational overhead [50], thus limiting its practicality for all but the smallest of databases.

Private fetches from secret sharing

The second approach for making the above vector-based fetches private uses secret sharing. In this approach, the client encodes the vector $\vec{e}_i$ component-wise using a (linear) secret sharing scheme [17, 29] (for instance, the basic (ℓ, ℓ)-additive scheme or Shamir's $(t+1, \ell)$-threshold scheme [48]), and then it sends share vectors to ℓ different database servers (who each hold a local replica of $\mathbf{D}$). Linearity of the secret sharing scheme enables each server to compute and return to the client a vector of secret shares that, together with the responses of the other servers, reconstructs component-wise to $\vec{\mathbf{D}}_i$.

The privacy guarantees of such sharing-based PIR follow directly from the security of the secret sharing scheme, which holds *unconditionally* under an appropriate non-collusion assumption; thus, such protocols are called *information-theoretically secure PIR* (IT-PIR) protocols. The most common non-collusion assumptions require that at most t-out-of-ℓ servers collude, in which case the protocol is called a *t-private ℓ-server IT-PIR* protocol.

IT-PIR is comparatively fast—two or more orders to magnitude faster than the most performant CPIR [43]—and it does not rely on unproven computational assumptions; however, unlike CPIR, it requires multiple *non-colluding* (but otherwise un-

1 The first such scheme [38] predates Paillier and instead uses Goldwasser and Micali's XOR-homomorphic encryption [31], which one can view as being additively homomorphic over a field of characteristic 2. Some recent proposals [1, 8] use lattice-based cryptosystems that are more efficient than those based on the hardness of number-theoretic problems.

Let $\mathbb{F}$ be a finite field and let $x_1, \ldots, x_\ell$ be a sequence of ℓ pairwise distinct, non-zero scalars from $\mathbb{F}$. To share a *secret* $\mathcal{S} \in \mathbb{F}$ among a set of ℓ *shareholders* so that any coalition of $t+1$ or more shareholders can efficiently reconstruct $\mathcal{S}$, yet no coalition of just t or fewer shareholders can derive any information about $\mathcal{S}$: (i) choose a polynomial $f \in \mathbb{F}[x]$ uniformly at random subject to $\deg f \leq t$ and $f(0) = \mathcal{S}$, and then (ii) for each $i = 1, \ldots, \ell$, send $(x_i, f(x_i))$ to the ith shareholder. The tuple $(x_i, f(x_i))$ is called the *ith share* of $\mathcal{S}$.

Any coalition of $k > t$ shareholders can reconstruct $\mathcal{S}$ from their collective shares (say, $(x_1, y_1), \ldots, (x_k, y_k)$) using polynomial interpolation. The standard approach for secret reconstruction is to use Lagrange interpolation:

$$\mathcal{S} = \sum_{j=1}^{k} y_j \left(\prod_{\substack{i=1 \\ i \neq j}}^{k} x_i (x_i - x_j)^{-1} \right).$$

By contrast, given just $k \leq t$ of the shares, all possibilities for the secret $\mathcal{S} \in \mathbb{F}$ are equally likely. Note that the Lagrange interpolation formula is just a specific linear combination of the y_i; going forward, we refer to the coefficients $\prod_{i=1, i\neq j}^{k} x_i (x_i - x_j)^{-1}$ in this linear combination as *Lagrange coefficients*.

In the case of Goldberg's IT-PIR protocol, the secrets to be shared are components of $\vec{e}_i$, the shareholders are the IT-PIR servers, and the security assumption is that no such coalition of $k > t$ shareholders will engage in secret reconstruction.

Fig. 1. Shamir's $(t+1, \ell)$-threshold scheme [48]

trusted) database servers. While non-collusion is a very strong assumption, many successful privacy-preserving designs, including secure multiparty computation [6], Tor [23], mix networks [12, 20], and cryptographic voting protocols [13, 46], also base their security on non-collusion assumptions.

In the sequel, we focus exclusively on IT-PIR protocols constructed in (a generalization of) the above secret-sharing model and instantiated with (again, a generalization of) Shamir's $(t+1, \ell)$-threshold scheme [48]. The basic form of this Shamir-based protocol was proposed by Goldberg [29] and was subsequently extended and improved in several papers [22, 34, 35, 41]. A C++ implementation of Goldberg's basic protocol, enhanced with several of these improvements, is available through the open-source Percy++ project [30]. For completeness, we describe Shamir's threshold scheme in Figure 1; interested readers should consult Shamir's [48] and Goldberg's [29] papers for additional details on the threshold scheme and how it is used to realize IT-PIR.

We treat Goldberg's protocol as the "baseline" against which to compare our new techniques, measuring the performance of our new IT-PIR protocols relative to that of Goldberg's basic protocol. Observation 1, which follows easily by inspection, characterizes the efficiency of this baseline.

Observation 1. The costs to fetch a block from $\mathbf{D} \in \mathbb{F}^{r \times s}$ using Goldberg's protocol are as follows:

- *Query preparation:* The client performs tr multiplications and $(t-1)r + 1$ additions in $\mathbb{F}$, per server (preparing share vectors, using Horner's method to evaluate polynomials);
- *Query transmission:* The client sends r field elements (a share vector in $\mathbb{F}^r$) to each of ℓ servers;
- *Query processing:* Each server performs rs multiplications and $(r-1)s$ additions in $\mathbb{F}$ (a vector-matrix multiplication);
- *Response transmission:* Each server sends s field elements (a share vector in $\mathbb{F}^s$) to the client; and
- *Reconstruction:* The client performs $s\ell$ multiplications and $(s-1)\ell$ additions in $\mathbb{F}$ to interpolate the responses (assuming the Lagrange coefficients have been precomputed).

Robust ℓ-server IT-PIR from noisy interpolation

An important practical consideration for ℓ-server IT-PIR schemes is how to ensure that fetches still succeed if one or more database servers are *Byzantine*—that is, if one or more servers respond either incorrectly or not at all. Beimel and Stahl [4, 5] initiated the study of t-private ℓ-server IT-PIR protocols that are resilient to such Byzantine behaviour. Specifically, they studied protocols that can guarantee successful fetches provided k-out-of-ℓ servers respond, even if up to v-out-of-k of the responses contain arbitrary errors (whether inadvertent or maliciously crafted). A protocol with this property is called a *t-private v-Byzantine-robust (k, ℓ)-server IT-PIR* protocol. As originally proposed, Goldberg's IT-PIR [29] is a t-private v-Byzantine-robust (k, ℓ)-server IT-PIR protocol for any $k > t$ and $v < k - \lfloor \sqrt{kt} \rfloor$. The latter inequality follows from the list decoding radius for Reed-Solomon codes.

Followup work by Devet, Goldberg, and Heninger [22] modifies the client in Goldberg's protocol to increase the robustness bound from $v < k - \lfloor \sqrt{kt} \rfloor$ to $v < k - t - 1$ Byzantine servers, which is the theoretically maximum possible value. Their improvement follows by using a variant of Cohn-Heninger multi-polynomial decoding [18] to decode *multiple queries* simultaneously. In the worst case, such decoding becomes possible only after the client issues up to v queries (where v is the number of servers giving Byzantine responses). They point out, however, that in many practical deployment scenarios the client will naturally issue v or more queries *irrespective of Byzantine behaviour by the servers*. In such cases, the algorithmic advances in the decoding algorithm yield improved Byzantine robustness "for free".

Of course, one hopes that $k = \ell$ servers will respond to every query, and that $v = 0$ will do so Byzantinely; nonetheless, it is common practice in the IT-PIR literature to provision some "extra" servers in order to provide resilience in the face of some malicious or malfunctioning servers. The next construction we discuss takes advantage of such over-provisioning of servers to increase the throughput of Goldberg's protocol.

Let $\mathbb{F}$ be a finite field and let $x_1, \ldots, x_\ell$ be a sequence of ℓ pairwise distinct scalars from $\mathbb{F} \setminus \{0, \ldots, q-1\}$. To share a size-$q$ set of *secrets* $\mathcal{S}_0, \ldots, \mathcal{S}_{q-1} \in \mathbb{F}$ among a set of ℓ *shareholders* so that any coalition of $t+q$ or more shareholders can reconstruct the q secrets but no coalition of just t or fewer shareholders can derive any information about them: (i) choose t scalars $y_q, \ldots, y_{q+t-1} \in \mathbb{F}$ uniformly at random, (ii) use polynomial interpolation to solve for the unique polynomial $f \in \mathbb{F}[x]$ of degree (at most) $t+q-1$ that passes through the $t+q$ points

$$(0, \mathcal{S}_0), (1, \mathcal{S}_1), \ldots, (q-1, \mathcal{S}_{q-1}), (q, y_q), (q+1, y_{q+1}), \ldots, (q+t-1, y_{q+t-1}),$$

corresponding to the q secrets and t random scalars, and then (iii) for each $i = 1, \ldots, \ell$, send $(x_i, f(x_i))$ to the ith shareholder. The tuple $(x_i, f(x_i))$ is called the *ith ramp share* of $\mathcal{S}_0, \ldots, \mathcal{S}_{q-1}$.

Any coalition of $k \geq t+q$ shareholders can reconstruct $\mathcal{S}_0, \ldots, \mathcal{S}_{q-1}$ from their collective ramp shares (say, $(x_1, y_1), \ldots, (x_k, y_k)$) using polynomial interpolation. Similar to in Shamir's $(t+1, \ell)$-threshold scheme, the standard approach is to use Lagrange interpolation to reconstruct $\mathcal{S}_h$ for each $h = 0, \ldots, q-1$:

$$\mathcal{S}_h = \sum_{j=1}^{k} y_j \prod_{\substack{i=1 \\ i \neq j}}^{k} (h - x_i)(x_j - x_i)^{-1}.$$

As in the $(t+1, \ell)$-threshold scheme, given just $k \leq t$ shares, all possibilities for the $\mathcal{S}_0 \ldots, \mathcal{S}_{q-1} \in \mathbb{F}$ are equally likely; given $t < k < t+q$ shares, each $\mathcal{S}_h$ is equally likely, but the joint distribution of $(\mathcal{S}_0, \ldots, \mathcal{S}_{q-1})$ has only $t+q-k-1$ degrees of freedom.

Fig. 2. $(t+1, q, \ell)$-ramp scheme variant of Shamir's threshold scheme [34]

q-batch queries from ramp schemes

Henry, Huang, and Goldberg [34] proposed a generalization of Goldberg's IT-PIR that allows clients to fetch q blocks at a time. The naïve way for a client to fetch q blocks from $\mathbf{D}$ would be for it to sequentially issue q standard, single-block queries. This approach requires the client to send qr field elements to and receive qs field elements from each server, and requires each server to perform about $2qrs$ field operations to process the q queries. A modest optimization is to have the client send all q share vectors in parallel and have the servers treat the q share vectors as rows of a $q \times r$ matrix. The servers then process all q queries simultaneously using fast matrix multiplication [3, 41]. This reduces the server-side computation cost—particularly so when q is large—and has no effect on the Byzantine robustness, communication cost, client-side computation cost, or server-side storage cost of the protocol.

Henry et al.'s idea is to instead replace Shamir's $(t+1, \ell)$-*threshold scheme* with a variant that encodes the q secrets concurrently, called a $(t+1, q, \ell)$-*ramp scheme*. Briefly, whereas Shamir's $(t+1, \ell)$-threshold scheme encodes one secret in a degree-t polynomial, the $(t+1, q, \ell)$-ramp scheme encodes q secrets in a degree-$(t+q-1)$ polynomial, as described in Figure 2. Batch queries leverage this idea as follows:

> *Rather than encoding $q = 1$ basis vector (and thereby fetching $q = 1$ block) per query, the client encodes $q > 1$ basis vectors (and thereby fetches $q > 1$ blocks) per query.*

The resulting queries—called *q-batch queries*—fetch q blocks at once with *the same communication and server-side computation cost as fetching a single block using a standard query*, thus offering a q-fold improvement in throughput.[2] Switching from a $(t+1, \ell)$-threshold scheme to a $(t+1, q, \ell)$-ramp scheme does not affect privacy; indeed, the entire protocol view from the perspective of any coalition of up to t servers is *identical* to that coalition's view in one single-block query using Goldberg's basic protocol. Rather, the tradeoff for achieving such a q-fold improvement in throughput is a comparatively modest reduction in the Byzantine robustness bound, from $v \leq k-t-1$ to $v \leq k-t-q$ Byzantine servers [34].

As in Goldberg's scheme, no coalition of t or fewer servers can derive any information about which blocks a client is requesting; however, now the client requires $t+q$ or more valid responses before polynomial interpolation becomes possible. (Note that coalitions of more than t but fewer than $t+q$ servers may learn some "partial" information about which blocks the client is requesting.) The decreased robustness bound for q-batch queries follows by applying Devet et al.'s variant of Cohn-Heninger multi-polynomial decoding [22, Algorithm 1] to polynomials whose degree is now increased from t to $t+q-1$. The resulting protocol implements *t-private v-Byzantine-robust q-batch (k, ℓ)-server IT-PIR* for any $q \leq |\mathbb{F}| - \ell$, $k \geq t+q$, and $v < k-t-q$. Its communication and server-side computation costs are exactly as in Observation 1, but now these costs are amortized over q (concurrent) fetches. Note that fixing $q = 1$ yields Goldberg's basic protocol.

Security model

We prove the security of our new constructions in the same model considered by Henry et al., which is a natural "batching-aware" extension of the standard security model for IT-PIR that was introduced in the seminal paper of Chor et al. [17, §2]. Although this model has been used implicitly in several recent papers [21, 34, 39, 51], a rigorous security definition is lacking from the literature; for completeness, we therefore provide such a definition. Specifically, we formally define a *t-private q-batch (k, ℓ)-server IT-PIR protocol*.

2 The client-side computation cost for a q-batch query is nominally higher than that for one single-block query, due to the increased degrees of the polynomials involved; it is, however, much lower than the client-side cost for q single-block queries. In any case, the computation cost incurred by the client in preparing share vectors and decoding the responses is typically very small relative to the cost incurred by each server when computing the product of the share vector it receives with $\mathbf{D}$.

An IT-PIR protocol is a special kind of multi-server information retrieval (IR) protocol; thus, we first define a *q-batch ℓ-server IR protocol* as a multi-party protocol, executed between a client and ℓ servers $S_1, \ldots, S_\ell$, that consists of four PPT algorithms: (i) a setup algorithm ($\mathcal{S}$), (ii) a query algorithm ($\mathcal{Q}$), (iii) an answer algorithm ($\mathcal{A}$), and (iv) a reconstruction algorithm ($\mathcal{R}$). The *setup* algorithm is executed once to set up the system. It takes as input the database $\mathbf{D} \in \mathbb{F}^{r\times s}$ and the identities of the ℓ servers, and it outputs an $(\ell+1)$-tuple $(\Gamma, \mathbf{D}(1), \ldots, \mathbf{D}(\ell)) \leftarrow \mathcal{S}(\mathbf{D}, S_1, \ldots, S_\ell)$ in which Γ is a string containing system parameters and each $\mathbf{D}(j)$ is an internal state to be held by S_j.[3] The system parameters—which comprise all metadata needed to carry out the protocol, such as the dimensions $r \times s$ of $\mathbf{D}$ and the identities/addresses of the servers—are an implicit input to each of the other algorithms. (We assume that $|\Gamma| \geq (r+s)\ell$ so that $\mathcal{Q}$ and $\mathcal{R}$ may run in time polynomial in $r\ell$ and $s\ell$, respectively.) The *query* algorithm is executed by the client to fetch (up to) q blocks, say $\vec{\mathbf{D}}_{i_1}, \ldots, \vec{\mathbf{D}}_{i_q}$, from the servers. It takes as input a q-tuple of block indices $(i_1, \ldots, i_q) \in [r-1]^q$, and it outputs an ℓ-tuple of requests $(Q_1, \ldots, Q_\ell) \leftarrow \mathcal{Q}(i_1, \ldots, i_q)$, one for each server. The *answer* algorithm is executed by S_j upon receiving a request Q_j from the client. It takes as input the incoming request Q_j and S_j's internal state $\mathbf{D}(j)$, and it outputs a response $A_j \leftarrow \mathcal{A}(\mathbf{D}(j), Q_j)$. Finally, the *reconstruction* algorithm is executed by the client to recover the blocks $\vec{\mathbf{D}}_{i_1}, \ldots, \vec{\mathbf{D}}_{i_q}$ from the servers' responses. It takes as input k (or more) responses $A_{j_1}, \ldots, A_{j_k}$ from the servers, and it outputs either the requested database blocks $(\vec{\mathbf{D}}_{i_1}, \ldots, \vec{\mathbf{D}}_{i_q}) \leftarrow \mathcal{R}(A_{j_1}, \ldots, A_{j_k})$ or $\perp$ to indicate failure. To qualify as t-private q-batch (k,ℓ)-server IT-PIR, these four algorithms must satisfy three special criteria.

Definition (t-private q-batch (k,ℓ)-server IT-PIR). A q-batch ℓ-server IR protocol $(\mathcal{S}, \mathcal{Q}, \mathcal{A}, \mathcal{R})$ with servers $S_1, \ldots, S_\ell$ implements t-private q-batch (k,ℓ)-server IT-PIR if it is:

i) k-correct: For every batch of q block indices $(i_1, \ldots, i_q) \in [r-1]^q$ and for every size-k subset $\{S_{j_1}, \ldots, S_{j_k}\}$ of the servers, if $(Q_1, \ldots, Q_\ell) \leftarrow \mathcal{Q}(i_1, \ldots, i_q)$ and $A_j \leftarrow \mathcal{A}(\mathbf{D}(j), Q_j)$ for each $j = 1, \ldots, \ell$, then

$$\Pr[(\vec{\mathbf{D}}_{i_1}, \ldots, \vec{\mathbf{D}}_{i_q}) \leftarrow \mathcal{R}(A_{j_1}, \ldots, A_{j_k})] = 1;$$

ii) t-private: Let X be a random variable denoting the block indices the client requests. For every q-tuple of block indices $(i_1, \ldots, i_q) \in [r-1]^q$ and for every size-t coalition $T = \{S_{j_1}, \ldots, S_{j_t}\}$ of the servers,

$$\Pr[X = (i_1, \ldots, i_q) \mid Q_T = (Q_{j_1}, \ldots, Q_{j_t})] = \Pr[X = (i_1, \ldots, i_q)],$$

where Q_T is a random variable denoting the joint distribution of requests sent to the servers in T; and

iii) non-trivial: For every $c > 0$, there exists some positive integer N such that, for all $r, s > N$, the following holds: For every database $\mathbf{D} \in \mathbb{F}^{r\times s}$ and for every batch of q indices $(i_1, \ldots, i_q) \in [r-1]^q$, if

- $(\Gamma, \mathbf{D}(1), \ldots, \mathbf{D}(\ell)) \leftarrow \mathcal{S}(\mathbf{D}, S_1, \ldots, S_\ell)$,
- $(Q_1, \ldots, Q_\ell) \leftarrow \mathcal{Q}(i_1, \ldots, i_q)$, and
- $A_j \leftarrow \mathcal{A}(\mathbf{D}(j), Q_j)$ for each $j = 1, \ldots, \ell$,

then $|\Gamma| + |Q_1| + \cdots + |Q_\ell| + |A_1| + \cdots + |A_\ell| < c|\mathbf{D}|$.

Note that a protocol can be both t-private and k-correct only if $t < k$; indeed, as mentioned previously, it is customary in the IT-PIR literature to set $t < k-1$ so that the protocol can provide some robustness to Byzantine servers. In the remainder of this paper, we follow Henry et al. in studying ways to trade off some of this robustness for better performance. After having introduced our new techniques in Sections 3–5, we briefly revisit this assumption in Section 7, where we discuss the alternative approach of allowing the parameters ℓ, k, and t to vary.

3 Ramp-coded databases

The q-batch queries of Henry et al. encode several requests together in a single query using $(t+1, q, \ell)$-ramp shares, thereby enabling clients to fetch q blocks with the same communication and server-side computation cost as fetching just one block using a standard, non-batch query. In this section, we flip that idea on its head by using ramp shares to encode not requests for data, but the data themselves.

Encoding the data in ramp shares yields efficiency-robustness tradeoffs complementing those offered by q-batch queries. Specifically, encoding several blocks in a single vector of ramp shares allows each server to store a matrix comprising *strictly fewer rows* than the unencoded database. Consequently, the upstream communication and server-side computation costs for a single-block query over the encoded database are both strictly lower than in the baseline protocol.

Encoding the database

We now describe how to encode $\mathbf{D} \in \mathbb{F}^{r\times s}$ using $(1, u, \ell)$-ramp shares. At a high level, our approach is to represent indices i of the blocks in $\mathbf{D}$ in the form $i = ui_Q + i_R$ with $0 \leq i_R < u$, and then to encode each of the u blocks associated with a given quotient i_Q together in a vector of $(1, u, \ell)$-ramp shares over $\mathbb{F}$. We refer to this encoding as a *u-ary encoding* of $\mathbf{D}$ and to u as the *arity* of the encoding. For ease of exposition, we assume throughout that $u \mid r$; however, we remark that the case where $u \nmid r$ presents no technical difficulties.

3 Typically, the setup algorithm just returns $\mathbf{D}(1) = \mathbf{D}(2) = \cdots = \mathbf{D}(\ell) = \mathbf{D}$ so that each server holds an exact replica of $\mathbf{D}$; however, in our constructions, the setup procedure will be somewhat more elaborate.

$$\mathbf{D} = \begin{pmatrix} \mathbf{D}_{(0,0)} & \mathbf{D}_{(0,1)} \\ \mathbf{D}_{(1,0)} & \mathbf{D}_{(1,1)} \\ \mathbf{D}_{(2,0)} & \mathbf{D}_{(2,1)} \\ \mathbf{D}_{(3,0)} & \mathbf{D}_{(3,1)} \end{pmatrix} \quad \overset{2\text{-ary}}{\Longrightarrow} \quad \mathbf{D}^{(2)} = \begin{pmatrix} \vec{d}_0(x) \\ \vec{d}_1(x) \end{pmatrix} = \begin{pmatrix} (\mathbf{D}_{(1,0)} - \mathbf{D}_{(0,0)})x + \mathbf{D}_{(0,0)} & (\mathbf{D}_{(1,1)} - \mathbf{D}_{(0,1)})x + \mathbf{D}_{(0,1)} \\ (\mathbf{D}_{(3,0)} - \mathbf{D}_{(2,0)})x + \mathbf{D}_{(2,0)} & (\mathbf{D}_{(3,1)} - \mathbf{D}_{(2,1)})x + \mathbf{D}_{(2,1)} \end{pmatrix}$$

$$\text{2nd bucket: } \begin{pmatrix} 2\cdot\mathbf{D}_{(1,0)} - 1\cdot\mathbf{D}_{(0,0)} & 2\cdot\mathbf{D}_{(1,1)} - 1\cdot\mathbf{D}_{(0,1)} \\ 2\cdot\mathbf{D}_{(3,0)} - 1\cdot\mathbf{D}_{(2,0)} & 2\cdot\mathbf{D}_{(3,1)} - 1\cdot\mathbf{D}_{(2,1)} \end{pmatrix} \quad \text{4th bucket: } \begin{pmatrix} 4\cdot\mathbf{D}_{(1,0)} - 3\cdot\mathbf{D}_{(0,0)} & 4\cdot\mathbf{D}_{(1,1)} - 3\cdot\mathbf{D}_{(0,1)} \\ 4\cdot\mathbf{D}_{(3,0)} - 3\cdot\mathbf{D}_{(2,0)} & 4\cdot\mathbf{D}_{(3,1)} - 3\cdot\mathbf{D}_{(2,1)} \end{pmatrix}$$

$$\text{3rd bucket: } \begin{pmatrix} 3\cdot\mathbf{D}_{(1,0)} - 2\cdot\mathbf{D}_{(0,0)} & 3\cdot\mathbf{D}_{(1,1)} - 2\cdot\mathbf{D}_{(0,1)} \\ 3\cdot\mathbf{D}_{(3,0)} - 2\cdot\mathbf{D}_{(2,0)} & 3\cdot\mathbf{D}_{(3,1)} - 2\cdot\mathbf{D}_{(2,1)} \end{pmatrix} \quad \text{5th bucket: } \begin{pmatrix} 5\cdot\mathbf{D}_{(1,0)} - 4\cdot\mathbf{D}_{(0,0)} & 5\cdot\mathbf{D}_{(1,1)} - 4\cdot\mathbf{D}_{(0,1)} \\ 5\cdot\mathbf{D}_{(3,0)} - 4\cdot\mathbf{D}_{(2,0)} & 5\cdot\mathbf{D}_{(3,1)} - 4\cdot\mathbf{D}_{(2,1)} \end{pmatrix}$$

Fig. 3. The 2-ary encoding procedure for a 4×2 database into 4 buckets

The new ramp-based encoding is closely related to the optional τ-independence [28] feature of Goldberg's basic protocol [29], which encodes the database component-wise using Shamir's $(\tau+1, \ell)$-threshold scheme. This prevents coalitions of up to τ database servers from learning the contents of the database they hold. Indeed, simply swapping in a $(\tau+1, u, \ell)$-ramp scheme in our construction yields a τ-independent u-ary encoding that gives rise to IT-PIR protocols with closely related parameters. In the following, we assume that $\tau = 0$.

The encoding procedure is simple. Given a database $\mathbf{D} \in \mathbb{F}^{r\times s}$, the first step is to rewrite $\mathbf{D}$ as

$$\mathbf{D} = \begin{pmatrix} \tilde{\mathbf{D}}_0 \\ \tilde{\mathbf{D}}_1 \\ \vdots \\ \tilde{\mathbf{D}}_{\lceil r/u \rceil - 1} \end{pmatrix} \in \left(\mathbb{F}^{u\times s}\right)^{\lceil r/u \rceil \times 1},$$

with each submatrix $\tilde{\mathbf{D}}_i$ residing in $\mathbb{F}^{u\times s}$. The next step is to encode each submatrix $\tilde{\mathbf{D}}_i$ column-wise using the $(1, u, \ell)$-ramp scheme variant of Shamir's threshold scheme (see Figure 2). That is, for each $i = 0, \ldots, \lceil r/u \rceil - 1$, use polynomial interpolation to find the (unique) length-s vector of degree-$(u-1)$ polynomials $\vec{d}_i = \langle d_{i0}(x), d_{i1}(x), \cdots, d_{i(s-1)}(x)\rangle \in (\mathbb{F}[x])^s$ such that $d_{ik}(h)$ is equal to the component in position (h, k) of $\tilde{\mathbf{D}}_i$ for every $k = 0, \ldots, s-1$ and every $h = 0, \ldots, u-1$.

Let $x_1, \ldots, x_\ell$ be an arbitrary sequence of ℓ pairwise distinct scalars from $\mathbb{F} \setminus \{0, \ldots, u-1\}$. The setup algorithm provides each of the ℓ servers with a matrix obtained by evaluating the above ramp-share matrix component-wise at one of the x_i. In particular, for each $j = 1, \ldots, \ell$, server S_j holds the $\lceil r/u \rceil \times s$ matrix of ramp shares

$$\mathbf{D}^{(u)}(x_j) := \begin{pmatrix} \vec{d}_0(x_j) \\ \vec{d}_1(x_j) \\ \vdots \\ \vec{d}_{\lceil r/u \rceil - 1}(x_j) \end{pmatrix} = \begin{pmatrix} d_{00}(x_j) & \cdots & d_{0(s-1)}(x_j) \\ d_{10}(x_j) & \cdots & d_{1(s-1)}(x_j) \\ \vdots & \ddots & \vdots \\ d_{(\lceil r/u \rceil - 1)0}(x_j) & \cdots & d_{(\lceil r/u \rceil - 1)(s-1)}(x_j) \end{pmatrix},$$

called the *x_jth u-ary bucket* of $\mathbf{D}$.

Observe that the 1-ary encoding of $\mathbf{D}$ encodes each element in a *constant* polynomial so that all 1-ary buckets are equal to $\mathbf{D}$. For $u > 1$, each u-ary bucket is an element of $\mathbb{F}^{\lceil r/u \rceil \times s}$ and is therefore a factor u smaller than the 1-ary (i.e., unencoded) database, which itself resides in $\mathbb{F}^{r\times s}$.

Recalling that we have fixed the independence threshold as $\tau = 0$, note that (i) the x_jth u-ary bucket of $\mathbf{D}$ is uniquely determined by the triple $(u, x_j, \mathbf{D})$, and (ii) one can trivially recover $\mathbf{D}$ from any size-u set of distinct u-ary buckets using polynomial interpolation. Also note that, although reconstructing $\mathbf{D}$ from fewer than u buckets is not generally possible, the u-ary encoding provides no formal guarantees regarding the privacy of data stored in the database. Figure 3 illustrates the 2-ary encoding procedure for a 4×2 database into $\ell = 4$ buckets (indexed respectively by $x_1 = 2$, $x_2 = 3$, $x_3 = 4$, and $x_4 = 5$). When $u = 2$, as in the figure, the $(1, u, \ell)$-ramp shares are just points on lines in $\mathbb{F}[x]$ and so it is particularly easy to see how one can reconstruct $\mathbf{D}$ from any pair of u-ary buckets; when $u > 2$, the visualization is harder, but the underlying idea is exactly the same.

Querying the encoded database

We now describe how the client constructs a t-private query to fetch the block $\vec{\mathbf{D}}_i$ from a u-ary encoding of $\mathbf{D}$. The query construction is a direct modification of that for t-private queries in Goldberg's basic protocol, with the key differences being the vector length and the x-coordinate at which the client encodes a basis vector in the Shamir threshold shares. Answering a query involves multiplying y-coordinates of points on degree-t polynomials by those on degree-$(u-1)$ polynomials; thus, the servers respond to such queries with vectors whose components are y-coordinates on polynomials of degree $t + u - 1$. The resulting protocol will therefore be a t-private (k, ℓ)-server protocol for any $k \geq t + u$; that is, the client will require at least $k \geq t + u$ valid responses for polynomial interpolation to succeed.

Write $i = ui_Q + i_R$ with $0 \le i_R < u$ using the Division Algorithm (that is, let $i_Q = \lfloor i/u \rfloor$ be the quotient and $i_R = i \bmod u$ the remainder upon dividing the desired block index i by the arity of the encoding u.) The following observation follows immediately by inspection of the u-ary encoding procedure.

Observation 2. The block $\vec{\mathbf{D}}_i$ can be recovered via component-wise interpolation of the i_Qth rows from any size-u subset of u-ary buckets of $\mathbf{D}$ at the input $x = i_R$. Specifically, if $\vec{\mathbf{D}}^{(u)}_{i_Q}(x_1), \ldots, \vec{\mathbf{D}}^{(u)}_{i_Q}(x_k)$ denote the i_Qth rows of $k > u$ distinct u-ary buckets of $\mathbf{D}$, then

$$\vec{\mathbf{D}}_i = \sum_{j=1}^{k} \vec{\mathbf{D}}^{(u)}_{i_Q}(x_j) \prod_{\substack{i=1 \\ i \neq j}}^{k} (i_R - x_i)(x_j - x_i)^{-1}.$$

Given the above observation, the query algorithm is straightforward. Let $\vec{e}_{i_Q}$ denote the i_Qth standard basis vector from $\mathbb{F}^{\lceil r/u \rceil}$. To query for block $\vec{\mathbf{D}}_i$, the client chooses a length-$\lceil r/u \rceil$ vector of degree-t polynomials $\vec{f} \in (\mathbb{F}[x])^{\lceil r/u \rceil}$ uniformly at random subject to $\vec{f}(i_R) = \vec{e}_{i_Q}$, where $\vec{f}(i_R)$ denotes the component-wise evaluation of the polynomials comprising $\vec{f}$ at the input $x = i_R$. For each server $S_1, \ldots, S_\ell$, respectively holding u-ary buckets $\mathbf{D}^{(u)}(x_1), \ldots, \mathbf{D}^{(u)}(x_\ell)$ of $\mathbf{D}$, the client sends the length-$\lceil r/u \rceil$ vector of shares $\vec{f}(x_j) \in \mathbb{F}^{\lceil r/u \rceil}$ to S_j.

The answer algorithm is exactly as in Goldberg's basic scheme; that is, each server S_j simply responds with the vector-matrix product $\vec{R}(x_j) := \vec{f}(x_j) \cdot \mathbf{D}^{(u)}(x_j)$. The reconstruction algorithm recovers block $\vec{\mathbf{D}}_i$ from any $k \ge t + u$ responses via polynomial interpolation at $x = i_R$:

$$\begin{aligned}
&\sum_{j=1}^{k} \vec{R}(x_j) \prod_{\substack{h=1 \\ h \neq j}}^{k} (i_R - x_h)(x_j - x_h)^{-1} \\
&= \sum_{j=1}^{k} \vec{f}(x_j) \cdot \mathbf{D}^{(u)}(x_j) \prod_{\substack{h=1 \\ h \neq j}}^{k} (i_R - x_h)(x_j - x_h)^{-1} \\
&= \sum_{j=1}^{k} \Big(\sum_{n=1}^{\lceil r/u \rceil} f_n(x_j) \cdot \vec{\mathbf{D}}^{(u)}_n(x_j) \prod_{\substack{h=1 \\ h \neq j}}^{k} (i_R - x_h)(x_j - x_h)^{-1} \Big) \\
&= \sum_{n=1}^{\lceil r/u \rceil} \Big(\sum_{j=1}^{k} f_n(x_j) \cdot \vec{\mathbf{D}}^{(u)}_n(x_j) \prod_{\substack{h=1 \\ h \neq j}}^{k} (i_R - x_h)(x_j - x_h)^{-1} \Big) \\
&= \sum_{n=1}^{\lceil r/u \rceil} f_n(i_R) \cdot \vec{\mathbf{D}}^{(u)}_n(i_R) \\
&= \vec{e}_{i_Q} \cdot \mathbf{D}^{(u)}(i_R) \\
&= \vec{\mathbf{D}}^{(u)}_{i_Q}(i_R) \\
&= \vec{\mathbf{D}}_i.
\end{aligned}$$

Note that the polynomials in $\vec{f}$ are of degree t and the polynomials in each $\vec{\mathbf{D}}^{(u)}_i$ are of degree $u - 1$ so that $k \ge t + u$ responses indeed suffice to perform the above interpolation. Also note that when $u = 1$, the above queries are identical to t-private queries in Goldberg's basic protocol.

Analysis

Theorem 1 characterizes the security of IT-PIR queries over a u-ary database, as described in the preceding two subsections. It follows almost immediately from the properties of Shamir's threshold scheme, although one technical detail warrants special consideration: the x-coordinates at which clients encode the basis vectors in their queries.

It is clear that modifying Shamir's threshold scheme to encode secrets at an x-coordinate other than $x = 0$ has no impact on its security in "traditional" secret sharing contexts (provided, of course, that the shareholders' indices are selected so as not to coincide with any x-coordinate that may encode the secret); however, there is an important, subtle difference between sharing a secret and querying a u-ary database. Specifically, in the latter case, *learning the x-coordinate at which the share vector encodes the secret basis vector reveals nontrivial information about the index i of the requested block*. Indeed, if the client seeks block $\vec{\mathbf{D}}_{ui_Q + i_R}$, then $\vec{f}$ encodes the basis vector $\vec{e}_{i_Q}$ at the input $x = i_R$. Therefore, learning the x-coordinate at which the secret is stored is equivalent to learning the modulo-u congruence class of the requested block index. Fortunately, it is easy to argue that the set of share vectors held by any coalition of up to t servers contains no information about this x-coordinate (nor any other information about the block index i). Indeed, the restriction that $x_1, \ldots, x_\ell \notin \{0, \ldots, u - 1\}$ is sufficient to ensure that no information is leaked by encoding the basis vector $\vec{e}_{i_Q}$ at $x = i_R$.

Theorem 1. *In the construction for IT-PIR queries over a u-ary encoded database $\mathbf{D}$ described in the preceding two subsections, no coalition of up to t database servers, holding buckets $\mathbf{D}^{(u)}(x_{j_1}), \ldots, \mathbf{D}^{(u)}(x_{j_t})$ with $x_{j_1}, \ldots, x_{j_t} \notin \{0, \ldots, u - 1\}$, can deduce any information about the requested block index i; that is, the queries are t-private.*

Proof (sketch). Given any coalition T of t database servers holding share vectors $\vec{f}(x_{j_1}), \ldots, \vec{f}(x_{j_t})$ and any guess for the block index, say $i^* = ui^*_Q + i^*_R$ with $0 \le i^*_R < u$, there is a *unique* vector of degree-t polynomials consistent with these t share vectors and passing component-wise through $\vec{e}_{i^*_Q}$ at $x = i^*_R$.

Now, since $\vec{f} = \langle f_1, \ldots, f_{\lceil r/u \rceil} \rangle$ is uniform random, subject only to $\vec{f}(i_R) = \vec{e}_{i_q}$ and each of the f_i having degree at most t, the one-one correspondence noted above implies that

$$\Pr[Q_T = (\vec{f}(x_{j_1}), \ldots, \vec{f}(x_{j_t})) \mid \vec{f}(i^*_R) = \vec{e}_{i^*_Q}] = |\mathbb{F}|^{-t},$$

where Q_T is the random variable denoting the joint distribution of the share vectors sent to the servers in T. From here, Bayes' Theorem and some easy algebraic manipulations yields

$$\Pr[\vec{f}(i^*_R) = \vec{e}_{i^*_Q}] = \Pr[\vec{f}(i^*_R) = \vec{e}_{i^*_Q} \mid Q_T = (\vec{f}(x_{j_1}), \ldots \vec{f}(x_{j_t}))]$$

so that the query is t-private, as desired. □

Theorem 2 characterizes the Byzantine robustness of t-private IT-PIR queries over a u-ary database. It follows from the decoding bound for Devet et al.'s variant of Cohn-Heninger multi-polynomial decoding [22, Algorithm 1] and our earlier observation that the client must interpolate polynomials of degree $t + u - 1$ in order to reconstruct a block $\vec{\mathbf{D}}_i$ from the servers' responses to a t-private query.

Note that Cohn-Heninger decoding requires the syndrome in each noisy codeword to be *random* and *independent* [18]. As we make no *a priori* assumptions on the types of errors that Byzantine servers may introduce, the client must enforce this randomness and independence in the way it constructs its query. To this end, Devet et al. suggest the following simple modification to the query generation procedure [22, Section 4.1]: the client constructs share vectors as usual, but before sending them to the database servers, the client "blinds" each share vector by multiplying it with a uniform random, non-zero scalar. (Note that the client selects a fresh, uniform random scalar for each server it queries.) Prior to interpolating, the client "unblinds" each response by multiplying it with the appropriate inverse scalar. With this modification in place, the following theorem is immediate.[4]

Theorem 2. *In the construction for IT-PIR queries over a u-ary database described in the preceding two subsections (with the above randomization procedure in place), suppose that k-out-of-ℓ servers respond to a client's query for block $\vec{\mathbf{D}}_i$, and that $v < k - t - u$ of these servers are Byzantine. Then the client can, with high probability, correctly reconstruct $\vec{\mathbf{D}}_i$ after issuing at most $m = \lceil \frac{v}{k-v-t-u} \rceil \leq v$ requests.*

Note that $m \leq v$ requests is sufficient (but perhaps not necessary) to ensure that the Cohn-Heninger decoder receives the requisite number of noisy codewords having *uncorrelated* error syndromes. (Although the client receives s codewords in its response from each server, the syndromes *within* a given response may be correlated, even when the client employs Devet et al.'s randomization procedure.) Fortunately, the m requests can be arbitrary and so a client that already seeks m or more blocks obtains the robustness bound quoted in Definition 2 without any overhead from the decoder. A client seeking fewer than m blocks could minimize the overhead, for instance, by crafting $m - 1$ "robustness" queries in which all but the first component of the share vector is set to zero.

Theorems 1 and 2 together imply that the u-ary IT-PIR construction described in the preceding two subsections is a *t-private v-Byzantine-robust u-ary (k, ℓ)-server IT-PIR* protocol for any $u \leq |\mathbb{F}| - \ell$, $k \geq t + u$ and $v < k - t - u$. Observation 3 characterizes the communication and computation costs of the u-ary IT-PIR construction.

Observation 3. The costs to fetch a block using a t-private query over the u-ary encoding of $\mathbf{D} \in \mathbb{F}^{r \times s}$ are as follows:

- *Query preparation:* The client performs $(t + 1)\lceil r/u \rceil$ multiplications and $(t - 1)\lceil r/u \rceil + 1$ additions in $\mathbb{F}$ per server (preparing randomized share vectors, using Horner's method to evaluate the secret sharing polynomials);
- *Query transmission:* The client sends $\lceil r/u \rceil$ field elements (a share vector in $\mathbb{F}^{\lceil r/u \rceil}$) to each of ℓ servers;
- *Query processing:* Each responding server performs $\lceil r/u \rceil s$ multiplications and $(\lceil r/u \rceil - 1)s$ additions in $\mathbb{F}$ (a vector-matrix multiplication);
- *Response transmission:* Each responding server sends s field elements (a share vector in $\mathbb{F}^s$) to the client; and
- *Reconstruction:* The client performs $2sk$ multiplications and $(s - 1)k$ additions in $\mathbb{F}$ to derandomize and interpolate through the responses (assuming the appropriate Lagrange coefficients have been precomputed).

Notably, the upstream communication cost and server-side computation cost to query a u-ary encoding of $\mathbf{D}$ are both about a factor u lower than in the baseline protocol.

Finally, we note that fixing $u = 1$ yields Goldberg's basic protocol and that, for any $u > 1$, the server-side storage cost for a u-ary bucket of $\mathbf{D}$ is a factor u lower than that of storing $\mathbf{D}$ itself, as each server must do in the baseline protocol.

4 Ramp scheme constructions as batch codes

The q-batch queries and u-ary encodings respectively discussed in Sections 2 and 3 are both closely related to *batch codes*, which were introduced by Ishai, Kushilevitz, Ostrovsky, and Sahai [36] as a way to reduce the server-side computation cost of PIR in cases where clients wish to fetch several blocks at once. This section explores that connection in detail.

Batch codes

We first recall the definition of a batch code. The definition we state below differs from the original one given by Ishai et al. in three respects: (i) we have adapted the notation to match the notation we use in our IT-PIR constructions, (ii) our definition

4 If the client does *not* randomize queries as described above, then a coalition of Byzantine servers can introduce correlated error terms to foil the Cohn-Heninger decoder. In this case, clients may still use a list decoding algorithm, such as Guruswami-Sudan [33], to correctly decode when up to $v < k - \sqrt{k(t + u - 1)}$ servers are Byzantine.

refers to database *blocks* rather than the individual *words* (or *symbols*) that comprise the database, and (iii) we speak of *querying* buckets (using IT-PIR queries) rather than *probing* (or "reading from") buckets. The latter distinction is important, as it allows us to consider batch codes that incorporate some level of robustness to handle cases in which some of the servers are Byzantine, returning faulty responses to queries over the buckets they hold.

Definition (Batch codes; Ishai et al [36, Definition 2.1]). An $(r, \mathcal{R}, q, n, m)$-*batch code* over $\mathbb{F}$ encodes a database $\mathbf{D} \in \mathbb{F}^{r \times s}$ into an n-tuple of *buckets* in $\mathbb{F}^{\lceil \mathcal{R}/n \rceil \times s}$ in such a way that a client can obtain any subset of q blocks from $\mathbf{D}$ by querying each bucket (at most) m times.[5]

As each bucket in an $(r, \mathcal{R}, q, n, m)$-batch code comprises $\lceil \mathcal{R}/n \rceil \times s$ field elements, the aggregate storage cost for the n buckets is $n\lceil \mathcal{R}/n \rceil s \approx \mathcal{R}s$ field elements, in contrast to the rs field elements required to store the "default" plaintext encoding of $\mathbf{D}$. The ratio $r/\mathcal{R}$ is called the *information rate* of the batch code. A simple information-theoretic argument establishes that the information rate of any batch code can be at most 1. Viewing r, q, and n as given (fixed) parameters, batch codes seek to minimize $\mathcal{R}$ and m—thereby maximizing the information rate and minimizing the upstream communication cost and computation cost that must be incurred for a client to fetch a batch of q blocks by issuing PIR queries over the buckets.

Existing batch codes bucketize the database "vertically"; that is, when using the batch code to encode a database over which to perform IT-PIR, each server needs to hold a replica of all n buckets, and clients must query each server at all n buckets in order to fetch a batch of q blocks. Employing such *vertical batch codes* in Goldberg's protocol therefore reduces upstream communication cost and server-side computation cost (relative to having the client sequentially issue q single-block queries) if and only if $m\mathcal{R} < qr$, and it reduces the downstream communication cost if and only if $nm < q$. A review of batch codes in the literature reveals that all existing codes have $q < n$ and $m = 1$; thus, while running Goldberg's IT-PIR atop existing batch codes may lead to reduced upstream communication cost and server-side computation costs, it does so at the expense of increasing the downstream communication and server-side storage costs.

5 More generally, each bucket can have a different number of rows (in which case we require that the number of rows across all buckets merely sums to $\mathcal{R}$); however, all constructions of which we are aware (including the ones presented herein) have uniform-sized buckets that each comprise $\lceil \mathcal{R}/n \rceil$ rows, as in the above definition.

Ramp constructions as 'degenerate' batch codes

One can view q-batch queries and the u-ary encoding respectively as $(r, r\ell, q, \ell, 1)$- and $(r, \lceil r\ell/u \rceil, 1, \ell, 1)$-batch codes. Both of these codes are in some sense "degenerate". Specifically, q-batch queries are degenerate as a batch code because no actual "coding" happens to the *database*—each bucket is simply a replica of $\mathbf{D}$ and, therefore, the information rate $rs/(rs\ell) = 1/\ell$ is inversely proportional to the number of buckets. Although this information rate is essentially pessimal, q-batch queries bucketize the database "horizontally" so that each server holds just one of the ℓ buckets; hence, the *aggregate storage cost among all servers matches the theoretical lower bound* for the storage cost of any vertical batch code. Likewise, the u-ary encoding is degenerate as a batch code because (essentially) no coding happens to the *queries*—each query is just a slightly modified standard, non-batch query with which clients can only fetch batches of size $q = 1$. Nonetheless, as with q-batch queries, the u-ary encoding bucketizes the database horizontally, which yields an information rate of $rs/\lceil r\ell/u \rceil s \approx u/\ell$; hence, the aggregate storage cost across all servers is a factor u smaller than the theoretical lower bound for the storage cost of any vertical batch code.

Ramp constructions as 'robust' batch codes

In the preceding subsection, we observed that one can view q-batch queries and the u-ary encoding respectively as $(r, r\ell, q, \ell, 1)$- and $(r, \lceil r\ell/u \rceil, 1, \ell, 1)$-batch codes. An interesting property of these batch codes is that they provide for some degree of Byzantine robustness (the level of which is determined by the list decoding bound for Reed-Solomon codes). If the client in these two codes instead queries each bucket $m_q = \ell - t - q$ and $m_u = \ell - t - u$ times, respectively, and then uses Cohn-Heninger multi-polynomial decoding to interpret the responses, we obtain $(r, r\ell, qm_q, \ell, m_q)$- and $(r, \lceil r\ell/u \rceil, m_u, \ell, m_u)$-batch codes providing the theoretically optimal level of robustness.

5 Non-degenerate batch codes from ramp schemes

In this section, we turn our attention to the final construction of the paper: a new, horizontally bucketized $(r, \lceil r\ell/u \rceil, q, \ell, 1)$-batch code constructed from the Shamir ramp scheme. The new construction amalgamates (slightly modified versions of) q-batch queries and the u-ary encoding to obtain a novel $(r, \lceil r\ell/u \rceil, q, \ell, 1)$-batch code with which clients can fetch $q \geq 1$ blocks at a factor $1/u$ of the cost of fetching a *single* block using a non-batch query over an unencoded database.

Note that a "direct" composition of q-batch queries with the u-ary encoding fails to produce the desired batch code, as such a composition only permits clients to fetch a subset of the possible size-q batches. The reason for this is that the u-ary encoding requires clients to encode basis vectors *at very specific x-coordinates* within the ramp scheme polynomials. In particular, if two blocks are encoded at the same x-coordinate in the u-ary buckets—that is, if their block indices are congruent modulo u—then constructing a q-batch query to fetch any batch that contains these two blocks is not possible.

Example. Suppose $q = 2$ and consider how the client might encode a q-batch query to fetch a pair of blocks, say $\vec{\mathbf{D}}_{i_1}$ and $\vec{\mathbf{D}}_{i_2}$ with $i_1 = ui_{Q_1} + i_{R_1}$ and $i_2 = ui_{Q_2} + i_{R_2}$, from a u-ary encoding of $\mathbf{D}$. On one hand, if $i_1 \not\equiv i_2 \bmod u$ so that $i_{R_1} \neq i_{R_2}$, then encoding a 2-batch query for the desired blocks is straightforward and the client's query will succeed. (Indeed, to construct its query vector, the client merely interpolates through $\vec{\boldsymbol{e}}_{i_{Q_1}}$ at $x = i_{R_1}$, through $\vec{\boldsymbol{e}}_{i_{Q_2}}$ at $x = i_{R_2}$, and through t uniform random vectors at t additional x-coordinates.) On the other hand, if $i_1 \equiv i_2 \bmod u$ so that $i_{R_1} = i_{R_2}$, then the client finds itself in the impossible situation of having to interpolate through two distinct basis vectors, $\vec{\boldsymbol{e}}_{i_{Q_1}}$ and $\vec{\boldsymbol{e}}_{i_{Q_2}}$, at the same input $x = i_{R_1} = i_{R_2}$ within the ramp-scheme polynomials that comprise its query vector. Thus, *it is impossible to retrieve a fraction $1/u$ of all size-2 batches using a single 2-batch query*.

For the general case of $q > 1$ and $u > 1$ (or even $u + \tau > 1$), an analogous argument shows that it is not possible to encode a q-batch query for any $\vec{\mathbf{D}}_{i_1}, \ldots \vec{\mathbf{D}}_{i_q}$ in which $i_j \equiv i_k \bmod u$ for some pair $j \neq k$.[6] Fortunately, a simple modification to q-batch queries and the u-ary encoding allows them to be composed in a way that *does* allow clients to issue arbitrary q-batch queries over a u-ary database, for any $q \geq 1$ and $u \geq 1$.[7] Briefly, the fix is to modify the u-ary encoding to encode all blocks $\vec{\mathbf{D}}_i$ at pairwise distinct x-coordinates within the ramp scheme polynomials (i.e., to encode each block at an x-coordinate that is not used to encode any of the $r - 1$ other blocks in any of the $\lceil r/u \rceil$ polynomials). Encoding each block at a distinct x-coordinate overcomes the problem described above by ensuring that, no matter which batch the client seeks, no two blocks in that batch will require the client to interpolate its query vector through two different basis vectors at a single x-coordinate; as a tradeoff, this fix necessitates a somewhat larger field size, a subtle point to which we return at the end of this section.

6 One consequence of this fact, which was apparently overlooked by Henry et al., is that it is not possible to issue any q-batch query over a τ-independent database in Goldberg's basic protocol, since in this case $u = 1$ and all blocks are encoded in $(\tau + 1, \ell)$-threshold shares at $x = 0$.

7 Indeed, the proposed modification enables clients to issue arbitrary q-batch queries over τ-independent u-ary databases, for any $\tau \geq 0$ and $u \geq 1$.

Encoding the database

We now describe how to encode $\mathbf{D} \in \mathbb{F}^{r \times s}$ using $(1, u, \ell)$-ramp shares in such a way that clients can issue arbitrary q-batch queries over the encoded database. We refer to such an encoding of $\mathbf{D}$ as a *u-ary batch encoding* of $\mathbf{D}$ and to u as the *arity* of the batch encoding. As before, we assume (purely for ease of exposition) that $u \mid r$.

The u-ary batch encoding is nearly identical to the u-ary encoding from Section 3, with the only difference being the x-coordinate at which each block gets encoded in the ramp shares. In particular, as the goal of the encoding is to retain the benefits of the u-ary encoding while ensuring that no size-q batch of blocks "collides" at an x-coordinate, the encoding is modified to ensure that *every block* in $\mathbf{D}$ is encoded at a *distinct* x-coordinate in the $(1, u, \ell)$-ramp shares that represent $\mathbf{D}$. The most natural way to do this is to encode the components of each block $\vec{\mathbf{D}}_i$ in the appropriate ramp-scheme polynomial at the input $x = i$. Thus the u-ary batch encoding procedure for a database $\mathbf{D} \in \mathbb{F}^{r \times s}$ is as follows. First, rewrite $\mathbf{D}$ as

$$\mathbf{D} = \begin{pmatrix} \tilde{\mathbf{D}}_0 \\ \hdashline \tilde{\mathbf{D}}_1 \\ \hdashline \vdots \\ \hdashline \tilde{\mathbf{D}}_{\lceil r/u \rceil - 1} \end{pmatrix} \in \left(\mathbb{F}^{u \times s}\right)^{\lceil r/u \rceil \times 1},$$

with each submatrix $\tilde{\mathbf{D}}_i$ residing in $\mathbb{F}^{u \times s}$. Next, for each $i = 0, \ldots, \lceil r/u \rceil - 1$, use polynomial interpolation to find the (unique) length-s vector of degree-$(u - 1)$ polynomials $\vec{\boldsymbol{d}}_i = \langle d_{i0}(x), d_{i1}(x), \cdots, d_{i(s-1)}(x) \rangle \in (\mathbb{F}[x])^s$ such that $d_{ik}(ui + h)$ is equal to the component in position (h, k) of $\tilde{\mathbf{D}}_i$ for every $k = 0, \ldots, s - 1$ and every $h = 0, \ldots, u - 1$.

Let $x_1, \ldots, x_\ell$ be an arbitrary sequence of ℓ pairwise distinct scalars from $\mathbb{F} \backslash \{0, \ldots, r-1\}$. The setup algorithm provides each of the ℓ database servers with a bucket obtained by evaluating the above ramp-share vectors component-wise at one of the x_i. In particular, server S_j holds the $\lceil r/u \rceil \times s$ matrix of ramp shares

$$\mathbf{D}^{(u)}(x_j) \coloneqq \begin{pmatrix} \vec{\boldsymbol{d}}_0(x_j) \\ \vec{\boldsymbol{d}}_1(x_j) \\ \vdots \\ \vec{\boldsymbol{d}}_{\lceil r/u \rceil - 1}(x_j) \end{pmatrix} = \begin{pmatrix} d_{00}(x_j) & \cdots & d_{(0s-1)}(x_j) \\ d_{10}(x_j) & \cdots & d_{1(s-1)}(x_j) \\ \vdots & \ddots & \vdots \\ d_{(\lceil r/u \rceil - 1)0}(x_j) & \cdots & d_{(\lceil r/u \rceil - 1)(s-1)}(x_j) \end{pmatrix},$$

called the *x_jth u-ary batch bucket* of $\mathbf{D}$.

Encoding a q-batch query

We now describe how the client constructs a t-private q-batch query to fetch blocks $\vec{\mathbf{D}}_{i_1}, \ldots, \vec{\mathbf{D}}_{i_q}$ from a u-ary batch encoding of $\mathbf{D}$. The query construction is a direct modification of Henry et al.'s q-batch queries, with the key differences being the vector length and the x-coordinates at which the client encodes the basis vectors in the ramp shares.

The query algorithm for a size-q batch of (pairwise distinct) blocks $\vec{\mathbf{D}}_{i_1}, \ldots, \vec{\mathbf{D}}_{i_q}$ works as follows. For each $k = 1, \ldots, q$, let $i_{Q_k} = \lfloor i_k / u \rfloor$ denote the quotient obtained upon dividing i_k by u and let $\vec{e}_{i_{Q_k}}$ denote the i_{Q_k} th standard basis vector from $\mathbb{F}^{\lceil r/u \rceil}$. The client chooses a length-$\lceil r/u \rceil$ vector of degree-$(t+q-1)$ polynomials $\vec{f} \in (\mathbb{F}[x])^{\lceil r/u \rceil}$ uniformly at random subject to $\vec{f}(i_k) = \vec{e}_{i_{Q_k}}$ for each $k = 1, \ldots, q$, where $\vec{f}(i_k)$ denotes the component-wise evaluation of the polynomials comprising $\vec{f}$ at the input $x = i_k$. For each server $S_1, \ldots, S_\ell$, respectively holding u-ary batch buckets $\mathbf{D}^{(u)}(x_1), \ldots, \mathbf{D}^{(u)}(x_\ell)$ of $\mathbf{D}$, the client sends the length-$\lceil r/u \rceil$ vector of shares $\vec{f}(x_j) \in \mathbb{F}^{\lceil r/u \rceil}$ to server S_j.

The answer algorithm does not change: each server S_j simply responds with the vector-matrix product $\vec{R}(x_j) := \vec{f}(x_j) \cdot \mathbf{D}^{(u)}(x_j)$. The reconstruction algorithm recovers the blocks $\vec{\mathbf{D}}_{i_1}, \ldots, \vec{\mathbf{D}}_{i_q}$ from any $k \geq t+q+u-1$ responses via polynomial interpolation at the sequence of inputs $x = i_1, \ldots, i_q$. Specifically, to obtain block $\vec{\mathbf{D}}_{i_m}$ it computes:

$$
\begin{aligned}
&\sum_{j=1}^{k} \vec{R}(x_j) \prod_{\substack{h=1 \\ h \neq j}}^{k} (i_m - x_h)(x_j - x_h)^{-1} \\
&= \sum_{j=1}^{k} \vec{f}(x_j) \cdot \mathbf{D}^{(u)}(x_j) \prod_{\substack{h=1 \\ h \neq j}}^{k} (i_m - x_h)(x_j - x_h)^{-1} \\
&= \sum_{j=1}^{k} \Big(\sum_{n=1}^{\lceil r/u \rceil} f_n(x_j) \cdot \vec{\mathbf{D}}_n^{(u)}(x_j) \prod_{\substack{h=1 \\ h \neq j}}^{k} (i_m - x_h)(x_j - x_h)^{-1} \Big) \\
&= \sum_{n=1}^{\lceil r/u \rceil} \Big(\sum_{j=1}^{k} f_n(x_j) \cdot \vec{\mathbf{D}}_n^{(u)}(x_j) \prod_{\substack{h=1 \\ h \neq j}}^{k} (i_m - x_h)(x_j - x_h)^{-1} \Big) \\
&= \sum_{n=1}^{\lceil r/u \rceil} f_n(i_m) \cdot \vec{\mathbf{D}}_n^{(u)}(i_m) \\
&= \vec{e}_{i_{Q_m}} \cdot \vec{\mathbf{D}}^{(u)}(i_m) \\
&= \vec{\mathbf{D}}_{i_{Q_m}}^{(u)}(i_m) \\
&= \vec{\mathbf{D}}_{i_m}.
\end{aligned}
$$

Note that the servers process a query by multiplying y-coordinates of points on degree-$(t+q-1)$ polynomials by those on degree-$(u-1)$ polynomials; thus, the responses are vectors whose components are y-coordinates on polynomials of degree $t+q+u-2$ so that $k \geq t+q+u-1$ responses indeed suffice to perform the above interpolation. The resulting protocol is therefore a (k, ℓ)-server IT-PIR protocol for any $k \geq t+q+u-1$.

Analysis

Theorem 3 characterizes the security of the construction for q-batch IT-PIR queries over a u-ary batch database, as described in the preceding two subsections. It parallels Theorem 1 for the u-ary encoding, following from the restriction that $x_1, \ldots, x_\ell \notin \{0, \ldots, r-1\}$. Its proof is almost identical to the proof of Theorem 1 and is therefore omitted.

Theorem 3. *In the construction for q-batch IT-PIR queries over a u-ary batch database $\mathbf{D}$ described in the preceding two subsections, no coalition of up to t database servers, holding buckets $\mathbf{D}^{(u)}(x_{j_1}), \ldots, \mathbf{D}^{(u)}(x_{j_t})$ with $x_{j_1}, \ldots, x_{j_t} \notin \{0, \ldots, r-1\}$, can deduce any information about the requested block indices $i_1, \ldots, i_q$; that is, the q-batch queries are t-private.*

Theorem 4 characterizes the Byzantine robustness of t-private q-batch queries over a u-ary batch database. Like Theorem 2, it follows immediately from the decoding bound for Devet et al.'s variant of Cohn-Heninger multi-polynomial decoding [22, Algorithm 1] and the observation that the clients must interpolate polynomials of degree $t+q+u-2$ in order to reconstruct a batch of blocks $\vec{\mathbf{D}}_{i_1}, \ldots, \vec{\mathbf{D}}_{i_q}$. As with Theorem 2, it assumes that the client employs Devet et al.'s query randomization strategy to force randomness and independence in the error terms returned by Byzantine servers.

Theorem 4. *In the construction for q-batch IT-PIR queries over a u-ary batch database described in the preceding two subsections (with the randomization procedure described in Section 3 in place), suppose that the servers holding k-out-of-ℓ buckets respond to a client's query for blocks $\vec{\mathbf{D}}_{i_1}, \ldots, \vec{\mathbf{D}}_{i_q}$, and that $v < k - t - q - u + 1$ of these servers are Byzantine. Then the client can, with high probability, correctly reconstruct $\vec{\mathbf{D}}_{i_1}, \ldots, \vec{\mathbf{D}}_{i_q}$ after issuing at most $m = \lceil \frac{v}{k-v-t-q-u+1} \rceil \leq v$ queries to each bucket.*

Theorems 3 and 4 together imply that the new construction for q-batch IT-PIR queries over a u-ary batch database yields a *t-private v-Byzantine-robust u-ary q-batch (k, ℓ)-server IT-PIR* protocol for any $u \geq 1$, $q \geq 1$, $k \geq t+u+q-1$ and $v < k-t-q-u+1$. Observation 4 characterizes the communication and computation costs of this q-batch u-ary IT-PIR construction.

Observation 4. The costs to fetch a block using a t-private q-batch query over the u-ary batch encoding of $\mathbf{D} \in \mathbb{F}^{r \times s}$ are as follows:

- *Query preparation:* The client performs $(t+q)\lceil r/u \rceil$ multiplications and $(t+q-2)\lceil r/u \rceil + 1$ additions in $\mathbb{F}$ per server (preparing randomized share vectors, using Horner's method to evaluate the ramp scheme polynomials);

- *Query transmission:* The client sends $\lceil r/u \rceil$ field elements (a share vector in $\mathbb{F}^{\lceil r/u \rceil}$) to each of ℓ servers;
- *Query processing:* Each responding server performs $\lceil r/u \rceil s$ multiplications and $(\lceil r/u \rceil - 1)s$ additions in $\mathbb{F}$ (a vector-matrix multiplication);
- *Response transmission:* Each responding server sends s field elements (a share vector in $\mathbb{F}^s$) to the client; and
- *Reconstruction:* The client performs $(q+1)sk$ multiplications and $q(s-1)k$ additions in $\mathbb{F}$ to derandomize and interpolate through the responses (assuming the appropriate Lagrange coefficients have been precomputed).

Notably, the upstream communication and server-side computation costs to fetch q blocks from a u-ary batch encoding of $\mathbf{D}$ are both a factor qu lower, and the downstream communication cost is a factor q lower, than in the baseline protocol.

Finally, we note that fixing $u = q = 1$ yields Goldberg's basic protocol and that, as with the u-ary encoding, the server-side storage cost for a u-ary batch bucket of $\mathbf{D}$ is a factor u lower than that of storing $\mathbf{D}$ itself, as each server must do in the baseline protocol.

A note on the field size

One consequence of the new q-batch u-ary IT-PIR approach warrants some additional discussion; namely, that the liberal use of x-coordinates at which clients must encode basis vectors necessitates working in a field $\mathbb{F}$ of size at least $r + \ell$. Such a dependence of the field size on the number of database blocks is not a common feature of IT-PIR protocols (the dependence on ℓ is, of course, common to all schemes based on polynomial interpolation). In our case, the need to have $|\mathbb{F}| \geq r + \ell$ follows because a request for block $\vec{\mathbf{D}}_i$ must be encoded in a polynomial at the input $x = i$ (thus "exhausting" the x-coordinates $x = 0, \ldots, r-1$) and because each of the servers must hold a distinct u-ary batch bucket indexed by some $x \notin \{0, \ldots, r-1\}$.

Although the requirement to work in a field of (relatively) large order is not a major restriction, it does have some ramifications for the computational efficiency of the protocol. Specifically, we note that finite field arithmetic is fastest in fields with small order. Moreover, Percy++ [30]—the open-source implementation of Goldberg's IT-PIR protocol into which we have incorporated our constructions—includes very fast, hand-tuned implementations for arithmetic in two binary fields, $\mathbf{GF}(2^8)$ and $\mathbf{GF}(2^{16})$. Arithmetic in these small binary fields is incredibly fast, owing to the fact that additions and multiplications have been implemented as simple XOR operations and fast lookups in hard-coded multiplication tables.[8] Unfortunately, $\mathbf{GF}(2^8)$ and $\mathbf{GF}(2^{16})$ have just 256 and 65 536 elements, respectively, and can therefore support q-batch queries only over u-ary databases with a very small number of blocks.[9] Our experiments indicate that Percy++ can process PIR queries over $\mathbf{GF}(2^8)$ and $\mathbf{GF}(2^{16})$ at over 450× and 120× the rates at which it can process queries in the almost-identically-sized fields of integers modulo 257 and 65537. Arithmetic in $\mathbf{GF}(2^8)$ in particular is, far and away, the fastest mode of operation for Percy++, requiring just over 900 ms (of single-threaded CPU time) per GB of database on our test server (see Section 6).

The slowdown from moving to a field of large prime order is nowhere near as dramatic as the 450× slowdown observed when moving from $\mathbf{GF}(2^8)$ to the field of integers modulo 257. Indeed, increasing the field size simultaneously decreases the number of words needed to represent each database block; thus, while the computation cost per arithmetic operation increases, the total number of arithmetic operations each server must perform decreases. In our experiments, we observed that the computation cost to process a query continues to decline as the field size increases up to around 1024 bits, after which performance begins to slowly degrade (presumably due to limitations in the architecture). Still, the slowdown we observe switching from $\mathbf{GF}(2^8)$ to a 1024-bit prime was about 120×.

Of course, the performance comparisons we have discussed so far are not apples-to-apples: we are comparing heavily optimized $\mathbf{GF}(2^8)$ arithmetic with arithmetic performed by a general-purpose implementation (specifically, Victor Shoup's NTL library [49] backed by the GNU Multi-Precision library [26]). As part of prior work [35], we wrote a hand-tuned implementation for arithmetic in a particular finite field with 160-bit prime order, and we incorporated this implementation into version 0.8 of Percy++.[10] We feel confident in asserting that a 160-bit upper bound on $r + \ell$ is not at all restrictive, even for the largest conceivable databases. Our experiments indicate that the rate at which Percy++ can process PIR queries over this 160-bit field is only about a factor 1.36× slower than over $\mathbf{GF}(2^8)$ and is, in fact, a factor 1.5× faster than over $\mathbf{GF}(2^{16})$; hence, even for $u = 2$, we can expect notable speedups relative to (1-ary) queries over $\mathbf{GF}(2^8)$, and our experiments in the next section bare out this prediction.

8 In fact, recent x86 CPUs have instructions for $\mathbf{GF}(2^x)$ arithmetic through the Carryless Multiplication (CLMUL) instruction set, though Percy++ does not currently make use of these.

9 Indeed, the restriction $r + \ell \leq 256$ seems to preclude the use of $\mathbf{GF}(2^8)$ for most interesting applications. The restriction $r + \ell \leq 65\,536$ is much less severe and may be acceptable for some compelling applications, such as speeding up a recently proposed Netflix-inspired private video streaming service [32] that needs only support up to a few tens of thousands of titles.

10 This functionality was deprecated in version 1.0, the most recent stable release of Percy++. The choice of 160-bit prime order was intended to strike a balance between performance and the hardness of certain number-theoretic problems; however, 160 bits no longer provides adequate security for such computational problems and the fast arithmetic code therefore fell into a state of disrepair. As we do not require any computational hardness assumptions, we have revived this old code for our experiments in order to take advantage of its ample size and superior performance relative to other prime moduli.

6 Implementation and performance evaluation

The u-ary encoding and u-ary batch encodings introduced in this paper have been implemented in the development branch of Percy++ [30], an open-source implementation of Goldberg's PIR protocol in C++, and will be available in the next major Percy++ release (tentatively slated for late 2016).

We ran a series of experiments to see how well the empirical performance improvements due to the new encodings agree with our theoretical predictions.

Experimental setup

The client-side measurements were performed on the author's desktop in a VirtualBox VM running Ubuntu 16.04 (on a Windows 8.1 host). That machine is equipped with a quad-core i7-4770 CPU @ 3.4 GHz and 16 GB of RAM. The server-side measurements were performed on a remote server running Red Hat Enterprise 6.7. That machine is equipped with an older quad-core Xeon X5570 CPU @ 2.96 GHz and 48 GB of RAM. For all experiments, we disabled multithreading in Percy++ (both on the client and on the server); thus, the timings reported herein are wall-clock times for a single thread of execution. We emphasize that all of the workloads we measure are embarrassingly parallelizable so that the numbers we report should be viewed as a (very pessimistic) upper bound on the actual running times.[11]

We performed 100 trials of each experiment and report here the average over those 100 trials. We consistently observed standard deviations around 1%–2% of the mean, except in the server-side experiments using a database whose size exceeded available RAM, in which case the standard deviation was around 5%–6% of the mean. Due to this high degree of consistency, we felt that including error bars in the graphs added little information at the expense of much clutter; thus, we chose to omit them.

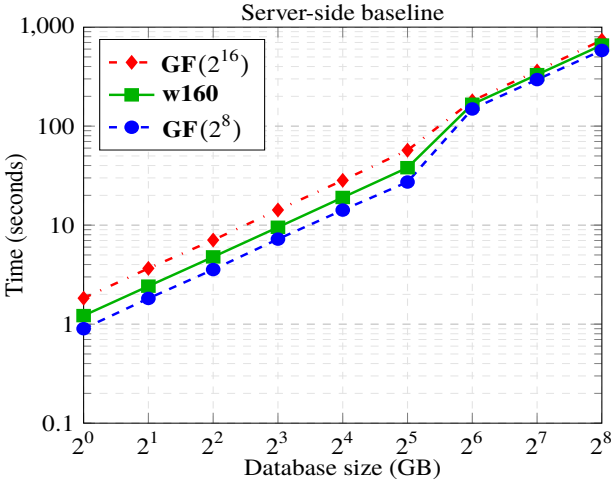
For server-side measurements, we ran each experiment using the fewest number of servers the parameters allowed; that is, for a given value of u, we set $t = 1$ and ran the experiment with $\ell = u+1$ servers.[12] Prior to running the main experiments, we ran preliminary experiments to confirm that t, q, and ℓ have no impact on the per-server cost; indeed, per-server running times in our experiments using $u+1$ versus $2(u+1)$ servers were almost identical (with correlation $R^2 = 0.9999997$).

11 Note that running the experiments with multithreading enabled would reduce the actual times we report, but not the trends.

12 Of course, we could not set $t = 0$ as then the query vectors would be non-uniform, which we observed to have a significant impact on the running times.

Experiment 1: Server-side baseline

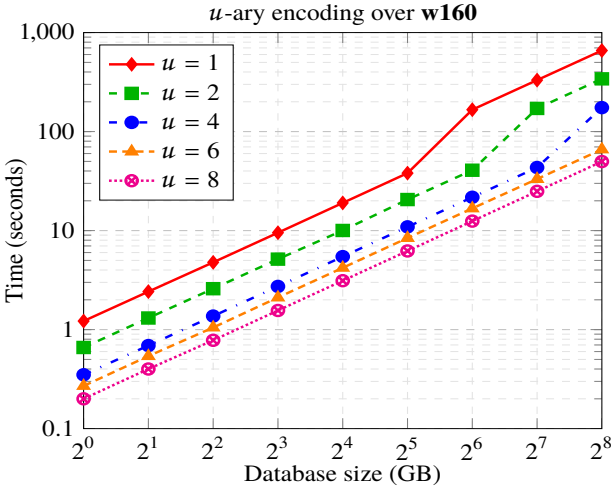
Our first experiment measures the server-side computation cost of the baseline protocol for various database sizes up to 256 GB. Note that the running time at the server depends mostly on the total size of the database, not its dimensions; thus, we set $r \approx s$ so that the database is approximately square. In the plot, **w160** denotes the aforementioned 160-bit prime-order field.



The plot uses a log-log scale so that the results can be easily compared with Experiment 2, where some data points differ by over an order of magnitude. The abrupt steepening of the slope between 32 GB and 64 GB occurs because a 32 GB database fits in RAM, while a 64 GB database does not.

Experiment 2: Server-side encoded

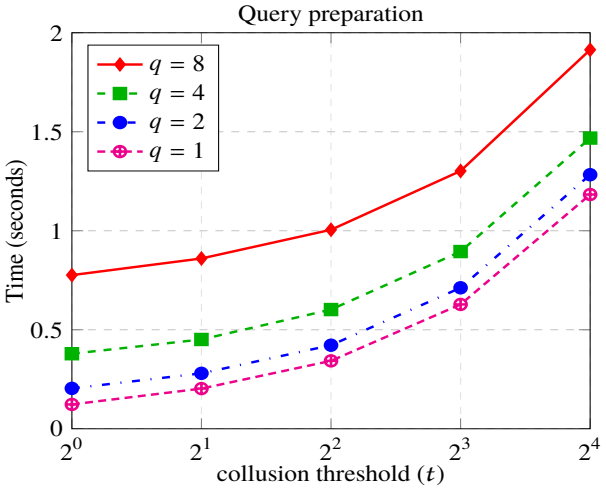
Our second experiment is similar to the first, except each server holds a u-ary bucket of the original database. We ran the experiment in **w160** for each $u \in \{1, 2, 4, 6, 8\}$.



For small databases that fit in RAM, the theoretically predicted factor-u speedup holds almost perfectly. When the unencoded database exceeds available RAM but a u-ary bucket does not, the speedup is more pronounced; for example, the 8-ary encoding for a 256 GB database gave a 13.5× speedup instead of the theoretically predicted 8× speedup.

Experiment 3: Query preparation

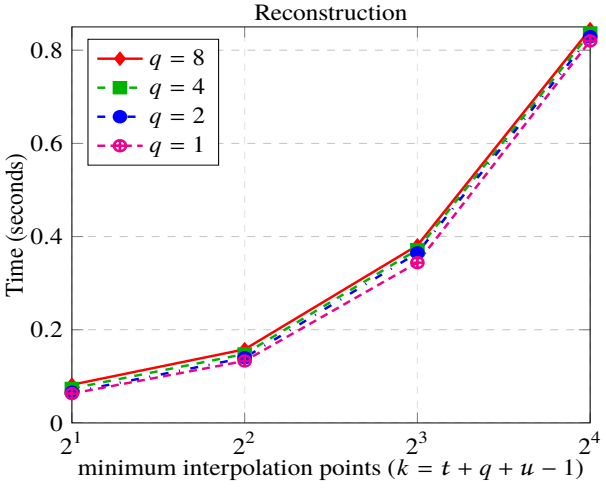
Our third experiment measures the cost for the client to construct its query. The cost to generate and evaluate each ramp-scheme polynomial depends on t and q; the number of such polynomials is $\lceil r/u \rceil$; and each of the polynomials must be evaluated at ℓ points. The plot displays measurements for constructing a query in **w160** with $u = 1$, $r = 16\,000$ and $\ell = 32$.



We note that (i) setting $u > 1$ would decrease the above costs (by a factor u), (ii) Percy++'s query construction code does not currently leverage preprocessing or fast **w160** arithmetic, (iii) this workload is highly parallelizable, and (iv) $\ell = 32$ seems like a large number of servers in practice; thus, the data in the plot are best viewed as *extremely* pessimistic upper bounds.

Experiment 4: Query reconstruction

Our fourth experiment measures the cost for the client to reconstruct its requested blocks. The cost to interpolate each ramp-scheme polynomial depends on the degree $t + q + u - 2$; and the number of interpolations to reconstruct q blocks is $s \cdot q$. The plot displays measurements for reconstructing a q-batch query in **w160** with $\ell = 32$ and $s = 200\,000$.



Similar to the case of query preparation, we note that this workload is highly parallelizable and that the reconstruction code in Percy++ does not currently leverage fast **w160** arithmetic (it does, however, precompute the Lagrange coefficients); thus, as with Experiment 3, the data in the plot is really a pessimistic upper bound.

Meta-analysis

The dimensions $r = 16\,000$ and $s = 200\,000$ from Experiments 2 and 3 correspond to a 64 GB database of 4 MB blocks over **w160**. Our "pessimistic" experiments indicate that the combined client-side compute time over a large selection of reasonable parameter settings for such a database never exceeds about 2.75 s, which we believe is well within the realm of feasibility for real-world deployment.

Indeed, setting $u = q = 8$ and $t = 16$ yields a 32-server 1-Byzantine-robust IT-PIR protocol (secure assuming an honest majority of servers) with which the client can fetch $8 \times 4 = 32$ MB of data from a 64 GB database in about $0.8 + 1.3/8 + 12.5 = 13.5$ total core seconds.

7 Discussion

Updating an encoded database

Most prior work in the IT-PIR literature does not explicitly address the cost of updates to the database. Indeed, in protocols that have each server hold a complete replica of **D**, the update process is rather simple and uninteresting. (Nonetheless, a fascinating line of recent work studies how clients can deal with cases where some replicas are out of sync due to failed or incomplete updates [25]. We could presumably adapt these techniques to work with u-ary encoded databases, although we do not explore that possibility here.) With u-ary encoded databases, updates may imply (at least partial) re-encoding and can therefore incur a much higher cost. The simplest and lowest cost updates for an encoded database include modifying blocks *in situ* and appending (or truncating) blocks at the end of the database. These operations are trivial to implement and need only incur overhead proportional to the size of the update. Inserting or deleting a block at an arbitrary location in the database can be much costlier, as doing so shifts the indices—and, therefore, x-coordinates—of all subsequent blocks. In the worst case, this could imply re-encoding every block. For applications where such insert/delete updates occur frequently, a better strategy might involve modifying the u-ary encoding in a way that decouples x-coordinates from block indices, and then to provide clients with an alternative means to discover the x-coordinate at which their desired blocks are encoded.

Varying the numbers of servers

Our analyses in this paper follows Henry et al. in assuming a fixed number of servers ℓ and collusion threshold t so that increasing q or u trades off Byzantine robustness in exchange for improved performance *without any impact to privacy*. An alternative to trading off robustness would be to allow ℓ and/or t to vary. By allowing the number of servers to grow, it is possible to increase q and u arbitrarily; however, we are reluctant to recommend this approach, as its implications for the privacy guarantees of the protocol are difficult to quantify. Indeed, even for natural assumptions like "honest majority", it appears hopeless to directly compare the relative privacy of IT-PIR protocol involving different numbers of servers—if protocol A is a 4-private 10-server protocol and protocol B is a 49-private 100-server protocol, which provides "stronger" privacy protection? Nevertheless, it is interesting to note that the flexibility to add additional servers enables parameter choices that, for example, can asymptotically decrease any given cost of interest to mere poly-logarithmic levels. This is because the u-ary encodings provide a way to directly map an *additive* increase in the difference $\ell - t$ to a *multiplicative* decrease in one or more protocol costs.

More efficient fields

The performance gap between queries over the 160-bit prime-order field and $\mathbf{GF}(2^8)$ is surprisingly narrow. Nonetheless, the choice of a 160-bit modulus was made for an old version of Percy++ to facilitate public key operations in an elliptic curve group, rather than to yield the fastest IT-PIR. We re-purposed the old modular arithmetic implementation in our experiments because it is, in fact, quite fast and because the existence of this code made the choice of a 160-bit prime-order field very convenient; however, switching to a similarly hand-tuned implementation of arithmetic in a field whose order is a prime slightly smaller than (perhaps a small multiple of) 64 bits (i.e., the CPU word size) would presumably result in somewhat faster IT-PIR on 64-bit x86 machines, while still providing ample indices to support q-batch queries over u-ary databases comprising—for all practical purposes—an unbounded number of blocks.

7.1 Summary

We have proposed new batch coding techniques that exploit the connection between ramp schemes and IT-PIR to allow new tradeoffs between the efficiency and robustness of Shamir-based IT-PIR. The new techniques allow clients to fetch several records for only a fraction the cost of fetching just one record using a standard query over an unencoded database. The batch codes are highly tuneable, providing a means to trade off (i) lower server-side computation cost, (ii) lower server-side storage cost, and/or (iii) lower uni- or bi-directional communication cost, in exchange for a comparatively modest decrease in resilience to Byzantine database servers. We implemented the new encodings in the open-source Percy++ library, and found that the performance improvements they yield agree well with our theoretical predictions.

Acknowledgements

The author thanks Yizhou Huang and Ian Goldberg for their collaboration on the prior work out of which these ideas grew, as well as the anonymous PETS reviewers for their many suggestions on how to improve the paper. Particular thanks are due to my shepherd, Tanja Lange, both for the especially detailed and valuable feedback she provided and for the impressive (and embarrassing) quantity of typos she located.

References

[1] Carlos Aguilar-Melchor and Philippe Gaborit. A fast private information retrieval protocol. In *Proceedings of ISIT 2008*, pages 1848–1852, Toronto, ON, Canada (July 2008).

[2] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-François Raymond. Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *Proceedings of FOCS 2002*, pages 261–270, Vancouver, BC, Canada (November 2002).

[3] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *Journal of Cryptology*, 17(2):125–151 (March 2004).

[4] Amos Beimel and Yoav Stahl. Robust information-theoretic private information retrieval. In *Proceedings of SCN 2002*, volume 2576 of *LNCS*, pages 326–341, Amalfi, Italy (September 2002).

[5] Amos Beimel and Yoav Stahl. Robust information-theoretic private information retrieval. *Journal of Cryptology*, 20(3):295–321 (July 2007).

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (Extended abstract). In *Proceedings of STOC 1988*, pages 1–10, Chicago, IL, USA (May 1988).

[7] George R. Blakley and Catherine Meadows. Security of ramp schemes. In *Advances in Cryptology: Proceedings of CRYPTO 1984*, volume 196 of *LNCS*, pages 242–268, Santa Barbara, CA, USA (August 1984).

[8] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. Private database queries using somewhat homomorphic encryption. In *Proceedings of ACNS 2013*, volume 7954 of *LNCS*, pages 102–118, Banff, AB, Canada (June 2013).

[9] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology: Proceedings of EUROCRYPT 1999*,

volume 1592 of *LNCS*, pages 402–414, Prague, Czech Republic (May 1999).

[10] Jan Camenisch, Gregory Neven, and abhi shelat. Simulatable adaptive oblivious transfer. In *Advances in Cryptology: Proceedings of EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 573–590, Barcelona, Spain (May 2007).

[11] Yan-Cheng Chang. Single database private information retrieval with logarithmic communication. In *Proceedings of ACISP 2004*, volume 3108 of *LNCS*, pages 50–61, Sydney, Australia (July 2004).

[12] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM (CACM)*, 24(2):84–88 (February 1981).

[13] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II: End-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE Transactions on Forensics and Security (TIFS)*, 4(4):611–627 (December 2009).

[14] Benny Chor and Niv Gilboa. Computationally private information retrieval (Extended abstract). In *Proceedings of STOC 1997*, pages 304–313, El Paso, TX, USA (May 1997).

[15] 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).

[16] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of FOCS 1995*, pages 41–50, Milwaukee, WI, USA (October 1995).

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

[18] 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).

[19] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Proceedings of PKC 2001*, volume 1992 of *LNCS*, pages 119–136, Jeju Island, South Korea (February 2001).

[20] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of IEEE S&P 2003*, pages 2–15, Oakland, CA, USA (May 2003).

[21] Danniel Demmler, Amir Herzberg, and Thomas Schneider. RAID-PIR: Practical multi-server PIR. In *Proceedings of CCSW 2014*, pages 45–56, Scottsdale, AZ, USA (November 2014).

[22] 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).

[23] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *Proceedings of USENIX Security 2004*, San Diego, CA, USA (August 2004).

[24] Zeev Dvir and Sivakanth Gopi. 2-server PIR with sub-polynomial communication. In *Proceedings of STOC 2015*, pages 577–584, Portland, OR, USA (June 2015).

[25] Giulia C. Fanti and Kannan Ramchandran. Efficient private information retrieval over unsynchronized databases. *IEEE Journal of Selected Topics in Signal Processing (J-STSP)*, 9(7):1229–1239 (October 2015).

[26] Free Software Foundation. GNU multiple precision arithmetic library; version 6.1.0 [computer software]. Available from: http://www.shoup.net/ntl/ (November 2015).

[27] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *Proceedings of ICALP 2005*, volume 3580 of *LNCS*, pages 803–815, Lisbon, Portugal (July 2005).

[28] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Science (JCSS)*, 60(3):592–629 (June 2000).

[29] Ian Goldberg. Improving the robustness of private information retrieval. In *Proceedings of IEEE S&P 2007*, pages 131–148, Oakland, CA, USA (May 2007).

[30] 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).

[31] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of STOC 1982*, pages 365–377, San Francisco, CA, USA (May 1982).

[32] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath T. V. Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with Popcorn. In *Proceedings of NSDI 2016*, pages 91–107, Santa Clara, CA, USA (March 2016).

[33] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. In *Proceedings of FOCS 1998*, pages 28–39, Palo Alto, CA, USA (November 1998).

[34] 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).

[35] Ryan Henry, Femi Olumofin, and Ian Goldberg. Practical PIR for electronic commerce. In *Proceedings of CCS 2011*, pages 677–690, Chicago, IL, USA (October 2011).

[36] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of STOC 2004*, pages 262–271, Chicago, IL, USA (June 2004).

[37] Aggelos Kiayias, Nikos Leonardos, Helger Lipmaa, Kateryna Pavlyk, and Qiang Tang. Optimal rate private information retrieval from homomorphic encryption. In *Proceedings of PETS 2015*, volume 2, pages 222–243, Philadelphia, PA, USA (June–July 2015).

[38] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of FOCS 1997*, pages 364–373, Miami Beach, FL, USA (October 1997).

[39] Lichun Li, Michael Militzer, and Anwitaman Datta. rPIR: Ramp secret sharing based communication efficient private information retrieval. *IACR Cryptology ePrint Archive*, Report 2014/044 (January 2014).

[40] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *Proceedings of ISC 2005*, volume 3650 of *LNCS*, pages 314–328, Singapore (September 2005).

[41] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In *Proceedings of FC 2015*, volume 8975 of *LNCS*, pages 168–186, San Juan, Puerto Rico (January 2015).

[42] 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).

[43] Femi G. Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *Proceedings of FC 2011*, volume 7035 of *LNCS*, pages 158–172, Gros Islet, St. Lucia (February 2011).

[44] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology: Proceedings of EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 223–238, Prague, Czech Republic (May 1999).

[45] 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).

[46] Peter Y. A. Ryan and Steve A. Schneider. Prêt à voter with re-encryption mixes. In *Proceedings of ESORICS 2006*, volume 4189 of *LNCS*, pages 313–326, Hamburg, Germany (September 2006).

[47] Nihar B. Shah, K. V. Rashmi, and Kannan Ramchandran. One extra bit of download ensures perfectly private information retrieval. In *Proceedings of ISIT 2014*, pages 856–860, Honolulu, HI, USA (June–July 2014).

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

[49] Victor Shoup. NTL, a library for doing number theory; version 9.8.1 [computer software]. Available from: http://www.shoup.net/ntl/ (April 2016).

[50] Radu Sion and Bogdan Carbunar. On the practicality of private information retrieval. In *Proceedings of NDSS 2007*, San Diego, CA, USA (March 2007).

[51] Luqin Wang, Trishank Karthik Kuppusamy, Yong Liu, and Justin Cappos. A fast multi-server, multi-block private information retrieval protocol. In *Proceedings of GLOBECOM 2015*, pages 1–6, San Diego, CA, USA (December 2015).

[52] Sergey Yekhanin. New locally decodable codes and private information retrieval schemes. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(127) (October 2006).