# Practical PIR for Electronic Commerce*

Ryan Henry
Cheriton School of Computer Science
University of Waterloo
Waterloo ON Canada N2L 3G1
rhenry@cs.uwaterloo.ca

Femi Olumofin
Cheriton School of Computer Science
University of Waterloo
Waterloo ON Canada N2L 3G1
fgolumof@cs.uwaterloo.ca

Ian Goldberg
Cheriton School of Computer Science
University of Waterloo
Waterloo ON Canada N2L 3G1
iang@cs.uwaterloo.ca

## ABSTRACT

We extend Goldberg's multi-server information-theoretic private information retrieval (PIR) with a suite of protocols for privacy-preserving e-commerce. Our first protocol adds support for single-payee tiered pricing, wherein users purchase database records without revealing the indices or prices of those records. Tiered pricing lets the seller set prices based on each user's status within the system; e.g., non-members may pay full price while members may receive a discounted rate. We then extend tiered pricing to support group-based access control lists with record-level granularity; this allows the servers to set access rights based on users' price tiers. Next, we show how to do some basic bookkeeping to implement a novel top-$K$ replication strategy that enables the servers to construct bestsellers lists, which facilitate faster retrieval for these most popular records. Finally, we build on our bookkeeping functionality to support multiple payees, thus enabling several sellers to offer their digital goods through a common database while enabling the database servers to determine to what portion of revenues each seller is entitled. Our protocols maintain user anonymity in addition to query privacy; that is, queries do not leak information about the index or price of the record a user purchases, the price tier according to which the user pays, the user's remaining balance, or even whether the user has ever queried the database before. No other priced PIR or oblivious transfer protocol supports tiered pricing, access control lists, multiple payees, or top-$K$ replication, whereas ours supports all of these features while preserving PIR's sublinear communication complexity. We have implemented our protocols as an add-on to Percy++, an open source implementation of Goldberg's PIR scheme. Measurements indicate that our protocols are practical for deployment in real-world e-commerce applications.

## General Terms

Algorithms, Design, Security

## Categories and Subject Descriptors

K.4.4 [**Computers and Society**]: Electronic Commerce—*security, payment schemes*; H.2.0 [**General**]: Security, integrity, and protection; H.2.4 [**Database Management**]: Systems—*distributed databases, query processing.*

*An extended version of this paper is available [36].


*CCS'11,* October 17–21, 2011, Chicago, Illinois, USA.


## Keywords

Private information retrieval, PIR, e-commerce, access control, zero-knowledge proofs, privacy-enhancing technologies, PETs.

## 1. INTRODUCTION

Private information retrieval (PIR) provides a means of querying a database without the database being able to learn any information about the query [22]. In multi-server PIR, $\ell$ database servers each possess a replica of the database and a user submits his query to some size-$k$ (or larger) subset of these servers in such a way that no server (or coalition of servers up to some threshold $t$) can learn the user's query. One can view the database $X$ as consisting of $n$ bits organized into $r$ records, each of size $b = n/r$ bits. We follow the usual convention of specifying a PIR query by the index $i$ of interest. Thus, in a PIR query, the user retrieves the record at index $i$ without the servers learning any information about $i$. We note, however, that existing approaches allow one to build queries that are more expressive on top of this basic setup; for example, keyword-based lookups [21] or simple SQL queries [48]. Using these techniques in conjunction with the ideas in this paper is straightforward.

Existing multi-server PIR schemes offer information-theoretic privacy protection for the user's query, but they allow a dishonest user to obtain additional information, such as the record at index $j \neq i$, or the exclusive-or of some subset of records in the database [31]. However, for many real-world applications, protecting database privacy by preventing dishonest users from learning extra information about the database is advantageous. Examples abound in online sales of digital goods, such as a pay-per-download music store [1] where users must pay for each song they download, a pay-per-retrieval DNA database [17], a stock-information database [31], or a patent database [2]. In all of these practical situations, it is necessary to guarantee the seller of these digital goods that users learn exactly the database record of interest and nothing more. In some scenarios it may even be desirable to sell database records according to a *tiered pricing plan* whereby different users pay different prices for each record depending on, e.g., their membership status or geographic location.

Symmetric private information retrieval (SPIR) [31] adds an additional restriction to PIR that prevents the user from learning information about any records except for the one he requested, thus addressing the need for simultaneous user and database privacy; however, no existing SPIR scheme supports both (tiered) record-level pricing and access control lists. Some oblivious transfer (OT) schemes [1, 17–19] offer one or the other of these functions, but no scheme in the literature provides them both. Moreover, OT schemes generally have no requirement for sublinear communication complexity, which renders them useless for online sales of

some types of digital goods, such as multimedia data, where the bandwidth requirement is high. Some schemes even require the user to download an encrypted copy of the *entire database* (e.g., [17–19]) and later purchase decryption keys for individual encrypted records. This allows one to amortize the cost of many transactions, but renders the scheme unsuitable for applications in which the contents of the database change frequently. Storing the database in an encrypted format also limits the usefulness of the database for other applications that need ready access to the cleartext data. Other OT-based schemes [1] require the database servers to store state information, such as the number of purchases made by a user, or his remaining balance, which might leak information about the user's queries or enable the server to link his purchases.

In this paper, we present a protocol that extends the open-source PIR scheme by Goldberg [33] to a priced symmetric private information retrieval (PSPIR) scheme offering tiered pricing with record-level granularity. Our initial PSPIR construction is a simple 'single payee' scheme wherein a single *content provider* (CP) sells digital goods through a distributed database and collects all proceeds from these sales. We then extend this simple scheme in three important ways. First, we introduce a slight modification to the protocol that enables the database servers to control access to individual records by implementing group-centric access control lists. Next, we propose a novel top-$K$ replication strategy that makes it possible for the database servers to periodically identify and replicate the $K$ most popular records to a smaller database — i.e., to construct a bestsellers list — thus facilitating more efficient retrieval for these most popular items. Finally, we show how to adapt the single-payee scheme to scenarios in which multiple (possibly competing) CPs sell their own digital goods through a common database and, using a distributed bookkeeping protocol, determine to what portion of the proceeds from these sales each seller is entitled. These enhancements provide a stronger and more realistic model of private information retrieval that enables e-commerce to coexist happily with strong privacy protection.

In our model, users belong to different pricing tiers and pay (perhaps different amounts) for each record; moreover, the database may require users to have explicit authorization to access some or all of the records in the database. In particular, tiered pricing logically groups users into different price tiers and allows the database to set the price and availability of each record with respect to each tier (a price tier is then roughly analogous with a *group* in the context of access control). Our approach enforces these constraints without revealing the user's price tier to the servers during a protocol run. Thus, when combined with an anonymous communications channel, our protocols maintain user anonymity in addition to query privacy; that is, the database servers do not learn any information about the identity nor the query of the user. More specifically, queries do not leak information about the index or price of the purchased record, the price tier according to which the user pays, the user's remaining balance, or even whether the user has ever queried the database before.

### *Outline.*

We organize the remainder of this paper as follows: §2 introduces our system model, including our design goals and threat model, and an example use case for our scheme. §3 presents our notation and the basic building blocks we use in our protocols. Our main contribution follows in §4, where we describe each of our constructions in depth. In §5, we discuss our implementation and the results of some empirical performance evaluations we ran on it. We then proceed to differentiate our approach from related solutions in §6, and finally conclude the paper with a summary in §7.

## 2. SYSTEM MODEL

Our basic scenario consists of three parties: the *user* interested in purchasing digital goods, the *server* having a database containing potentially tens or hundreds of gigabytes of data that is divided into $r$ records (or files), and the *bank*, an independent issuer of digital *wallets* (see below). We build our scheme from multi-server information-theoretic PIR; thus, the server is actually comprised of $\ell$ independent PIR servers that each hosts a complete replica of the database. Users submit their queries to any subset of at least $k > \ell/2$ servers of their choosing. We associate one or more price lists $\vec{p}_1, \ldots, \vec{p}_T$ with the database, where there are $T$ tiers and each price list specifies a price for each of the $r$ individual records. For simplicity, we represent price lists by length-$r$ vectors of nonnegative integers (or $\perp$ to indicate that a record is unavailable in this price plan), although representations that are more efficient are typically possible and using one of these representations changes the protocols only superficially. Users' wallets are kept with any non-rerandomizable (one-show) anonymous credential scheme, such as that of Brands [10] or that of Au *et al.* [3]; the wallet encodes as attributes a *balance* and the index $\pi$ of the price list according to which the owner of that wallet must pay, called that user's *price tier* or simply his *tier*. (For example, one price tier might apply to members while another price tier might apply to non-members; in general, any number of tiers may exist, although a large number of tiers might adversely affect system performance.) The tier $\pi$ is encoded in the credential in a special way: each wallet encodes a collection of $T$ attributes $x_1, \ldots, x_T$ such that $x_i = 1$ if $i = \pi$ and $x_i = 0$ otherwise. The bank initially issues each user with a wallet encoding the balance 0; users may charge their wallets at any time using, e.g., a prepaid credit card obtained via cash transaction from a grocery store. We make no assumptions regarding noncollusion between the bank and the database servers; indeed, it is not even required that the bank and database servers be different entities, although synchronization challenges may emerge in the case of a distributed bank. We do not discuss the full semantics of the bank, since this is not our focus in this paper and such details depend on the chosen credential system.

To query for the record at index $\beta$, the user must first prove that his wallet encodes sufficient funds to purchase that particular record according to his tier. To do so, the user must send his current wallet to each of at least $k$ servers, which makes the task of detecting double spending particularly easy (via the pigeonhole principle) since the wallet is not rerandomizable and $k > \ell/2$. Along with his query response, the user receives a cryptographically signed *receipt* encoding the price paid for the query and the wallet used to make the payment. The user then uses this receipt to *refresh* his wallet with the bank; i.e., to obtain a new wallet (which is unlinkable to his old wallet) encoding his new remaining balance. This refreshing step does not reveal any information to the bank about the user's (old or new) balance, his tier, or the price encoded in the receipt.

Before discussing our constructions in further detail, we first present our high-level design goals and our threat model, as well as some motivation by way of a simple example use case that uses our full suite of protocols.

### 2.1 Design goals

We are interested in enhancing Goldberg's PIR protocol to yield a scheme with the following properties.

#### *Utility.*

In addition to PIR's standard functionality, we seek to provide the database servers with the following capabilities.

**Tiered pricing:** Users pay predetermined amounts for each retrieved record. The system assigns each user to a price tier and the prices they pay depend on both this tier and the particular records they purchase.

**Access control:** The database servers may set the availability of each record with respect to users of each price tier.

**Replication:** The database servers can dynamically learn which records are most popular without revealing information about individual users' query patterns. This allows popular records to be accessed at a lower computation and communication cost than their less popular counterparts.

**Bookkeeping:** A common database may sell records from multiple CPs while ensuring that each CP receives the correct share of profits based on sales of its own records.

*Security and privacy.*

Traditional databases can already offer all of the above functionality, and more. What makes our situation unique is that we wish to provide this functionality while offering strong privacy protection, both for users and for CPs.

**Correctness:** Users with sufficient funds and privileges can always retrieve a consistent copy of their desired record.

**Query privacy:** The database servers and bank learn no non-trivial information about the records accessed by a user.

**User anonymity:** The database servers and the bank learn only that 'some user with sufficient funds and privileges retrieved some record'; i.e., they learn no other information about a user's identity (including whether or not this user has previously queried the database), the price he pays for a record, or the past or present balance encoded in his wallet.

**Database privacy:** Dishonest users cannot learn any extra information about database records that they do not purchase.

*Practicality.*

**Computational cost:** Any increase in the computational cost of the underlying PIR scheme should be small and scale sublinearly in $n$ and at most linearly in $r$.

**Query size:** The size of users' queries should increase by no more than a small multiplicative factor as compared to the underlying PIR scheme. Furthermore, the size of the query response should increase by no more than a small additive constant.

**Round complexity:** The protocols should add at most one additional round of interaction to the PIR protocol, per query.

## 2.2 Threat model

We consider a threat model in which users of the system are potentially malicious, while database servers and CPs (as well as the bank) are honest-but-curious (i.e., semi-honest); however, Goldberg's PIR scheme — and by extension, our own proposed scheme — is robust against some threshold of malicious database servers as well. Users of our system have obvious incentives for being malicious; for example, they may wish to learn about records that they cannot afford (or for which they simply do not want to pay), or to retrieve records for which they do not have authorization. Moreover, in e-commerce situations, unscrupulous competitors may try to subvert the bookkeeping and replication functionality by acting as users and submitting specially crafted, malformed queries. The system must provide the database servers with strong security guarantees against all such attacks.

Honest-but-curious database servers may collude among themselves (and the bank) to try to reveal the identity of a user, the price tier or balance encoded in his wallet, or the content of his queries. The system should be secure against attacks on user anonymity regardless of who may be colluding with whom, and should be secure against attacks on query privacy provided an honest majority exists among the database servers.[1] Malicious database servers may also try to compromise the integrity of the system by refusing to respond to user queries or by returning incorrect results in an effort to prevent a user from obtaining his desired record or a valid receipt. The system should be robust to some number of malicious database servers and should allow affected users to learn the identity of whichever servers misbehave. We do not consider the case of an actively malicious bank, but we do note that a fair exchange protocol such as the one proposed by Camenisch *et al.* [18] can also mitigate threats associated with a malicious bank.

We argue that our assumption of semi-honest database servers is realistic for many practical e-commerce scenarios, particularly in the multiple-payee variant of our protocol where the CPs themselves may host some or all of the database servers. In this setting, several distinct and possibly competing CPs cooperate to provide a value-added service to their customers (i.e., a privacy-preserving way to purchase their digital goods). On the one hand, the CPs have a vested interest in cooperating, since providing this service to their customers would otherwise be infeasible. On the other hand, competing CPs have an incentive not to divulge additional information about customer spending to one another, lest this information help the other CPs gain a competitive advantage over them.

## 2.3 A hypothetical use case

As a hypothetical use case to motivate our protocols, we consider an online seller of e-books akin to Amazon's Kindle Store. One can easily envision similar use cases for other online retailers (e.g., Google's Android Market); we consider an e-book store because the size of these increasingly popular digital goods makes them ideal candidates for distribution using PIR. In light of the Amazon's 2010 lawsuit against the state of North Carolina seeking to prevent the disclosure of customer purchase records [44], we also feel it is especially fitting to demonstrate how our platform could replace Amazon's current e-book sales model to provide a more privacy-friendly experience for customers, and to mitigate the risk of such egregious attacks on privacy in the future.

Suppose a number of independent publishers wish to team up to form a privacy-preserving alternative to the Kindle Store, wherein users can purchase electronic copies of these publishers' books without revealing their identities or facilitating the construction of privacy-invasive dossiers detailing their purchasing habits.[2]

Each publisher hosts a database server containing a replica of the entire e-book catalogue and users purchase e-books from this data-

[1] Of course, if sufficiently many database servers collude to reveal the content of a user's query, they may learn some information about that user's identity; for example, by noting if the user retrieves a record that is only available to certain tiers of users, or by making inferences based on external knowledge. We cannot protect against such attacks, so in these cases we aim only to minimize the information made available to the adversary.

[2] If the publishers desire an Amazon-style recommendation system, existing approaches to privacy-preserving targeted advertising [34, 38, 56] may apply; however, we leave further investigation of this idea for future work.

base using PSPIR. Periodically, (for example, weekly) the publishers cooperate to learn to what portion of the profits each is entitled. They also use this opportunity to determine the top-$K$ best sellers, which they subsequently replicate to a smaller database to facilitate faster purchases of these books. Much like Amazon does with the Kindle Store [26], the publishers can sell the same e-book to users that are registered in different geographical locations for different prices, thus enabling them to recoup costs associated with, e.g., service fees incurred by offering customers free 3G service to purchase their books from a mobile device.

# 3. BUILDING BLOCKS

This section introduces our notation and the cryptographic primitives that we use in our construction.

For notational convenience, we use $\delta_{ij}$ to denote the well-known Kronecker delta function; that is, $\delta_{ij} = 1$ if $i = j$, and $\delta_{ij} = 0$ otherwise. We also define the complementary Kronecker delta function, $\bar{\delta}_{ij} = (1 - \delta_{ij})$. We use $\mathbb{Z}_m$ to denote the ring of integers modulo $m$ (or the finite field of order $m$ when $m$ is prime); we will represent elements of $\mathbb{Z}_m$ by elements of $\{0, \ldots, m-1\}$. $\mathbb{Z}_m[x]$ denotes the ring of polynomials with coefficients in $\mathbb{Z}_m$, and $(\mathbb{Z}_m)^r$ the set of length-$r$ vectors over $\mathbb{Z}_m$ (and similarly for $(\mathbb{Z}_m[x])^r$). We write $a \in_R \mathbb{Z}_m$ to mean that $a$ is selected uniformly at random from $\{0, \ldots, m-1\}$. The notation $a \| b$ denotes concatenation (as strings) of values $a$ and $b$. $\kappa \in \mathbb{N}$ is a parameter used to tune the soundness versus performance of certain zero-knowledge proofs.

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be cyclic groups of prime order $q$ (which we shall express multiplicatively). We assume throughout the existence of a bilinear pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$; we also assume that $g, h \in \mathbb{G}_1$, $\hat{g} \in \mathbb{G}_2$ and $g_{\mathrm{T}}, h_{\mathrm{T}} \in \mathbb{G}_T$ are known generators of their respective groups, where $g_{\mathrm{T}} = e(g, \hat{g})$ and $h_{\mathrm{T}} = e(h, \hat{g})$. The crucial property of $e$ is that of bilinearity: $e(g^a, \hat{g}^b) = e(g, \hat{g})^{ab}$ for all $a, b \in \mathbb{Z}_q$. If $\mathbb{G}_1 = \mathbb{G}_2$, the pairing is called *symmetric*; otherwise it is *asymmetric*. Elements of $\mathbb{G}_1$ in asymmetric pairings are shorter than in symmetric pairings. The pairing $e$ we assume in this work is asymmetric.

## 3.1 Shamir secret sharing

We make extensive use of Shamir's polynomial secret sharing scheme [54] to share field elements among the servers. An element $a \in \mathbb{Z}_q$ is shared by choosing a polynomial $f_a(x) = a_t x^t + \cdots + a_1 x + a \in \mathbb{Z}_q[x]$ with each non-constant coefficient $a_i \in_R \mathbb{Z}_q$ and the constant term equal to the shared value; server $j$'s share of $a$ is then $f_a(j) \in \mathbb{Z}_q$. Any subset of at least $t+1$ servers can cooperate to reconstruct $a$ using Lagrange interpolation [45, Ch. 12]; however, $t$ or fewer colluding servers cannot deduce any non-trivial information about $a$. Such a scheme is called a $(t+1, \ell)$-threshold secret sharing scheme, since a threshold of at least $t+1$ out of $\ell$ servers must cooperate to recover the secret value. In general, any choice of $0 < t < \ell$ will suffice, however our top-$K$ replication protocol requires that $t \leq \lfloor (\ell - 1)/2 \rfloor$. We recommend $t = \lfloor (\ell - 1)/2 \rfloor$, which ensures that the protocols are secure whenever an honest majority exists among the servers. We write $[a]_q$ to denote a Shamir secret sharing of $a \in \mathbb{Z}_q$ among the $\ell$ servers; that is, $[a]_q = \langle f_a(1), \ldots, f_a(\ell) \rangle$ where the $j^{\text{th}}$ component of this vector is known only to server $j$.

### *Computing with Shamir secret shares.*

Suppose $[a]_q$ and $[b]_q$ are two shared secrets and $c \in \mathbb{Z}_q$ is a public scalar. We write $[a]_q \oplus [b]_q$, $[a]_q \ominus [b]_q$ and $[a]_q \odot [b]_q$ to denote the component-wise sum, difference and product, respectively, of $[a]_q$ and $[b]_q$ (and similarly for $[a]_q \oplus c$, $[a]_q \ominus c$ and $[a]_q \odot c$). Observe that $[a]_q \oplus [b]_q = [a+b]_q$ and $[a]_q \ominus [b]_q = [a-b]_q$ (and similarly for $[a]_q \oplus c$ and $[a]_q \ominus c$), and that $c \odot [a]_q = [c \cdot a]_q$. Moreover, the product $[a]_q \odot [b]_q$ yields a $(2t+1, \ell)$-threshold sharing of $a \cdot b$; thus, when $t \leq \lfloor (\ell - 1)/2 \rfloor$ as we require above, the servers can still interpolate to recover this product.

It is possible to construct algorithms for more complex operations using the above facts; e.g., distributed pseudorandom number generation [4], testing equality [25], or evaluating order predicates [46]. Indeed, we implicitly use these more complex operations for top-$K$ replication, but do not discuss them in depth. The interested reader can consult Nishide and Ohta's paper [46] for further details on how to implement them.

## 3.2 Goldberg's PIR scheme

Goldberg's PIR scheme [33] is a multi-server information-theoretic scheme with good support for query robustness against colluding servers. It provides a $t$-private $v$-Byzantine-robust $k$-out-of-$\ell$ scheme for $0 < t < k \leq \ell$ and $v < k - \lfloor \sqrt{kt} \rfloor$ protection. In other words, users submit their queries to at least $k$ out of the $\ell$ servers, and the system can tolerate up to $v$ servers being Byzantine (i.e., responding incorrectly) without inhibiting the ability of users to retrieve the correct record, and $t$ servers colluding without compromising users' query privacy. The scheme also optionally supports $\tau$-independence [30], a property that prevents the database servers from learning the contents of the database with information-theoretic protection against coalitions of up to $\tau$ servers.

The scheme structures the $n$-bit database $X$ as an $r \times s$ matrix $D$ over $\mathbb{Z}_q$, where $r$ is the number of records, $b$ is the size of each record (in bits), $w = \lfloor \lg q \rfloor$ is the word size, and $s = b/w$ is the number of words per record. For minimal communication, $b = \sqrt{wn}$. The user's query is a standard basis vector $\vec{1}_\beta \in (\mathbb{Z}_q)^r$, which has all elements 0 except for index $\beta$ where it is 1. The scheme uses Shamir secret sharing to split $\vec{1}_\beta$ into $k$ parts $\vec{\rho}_1, \ldots, \vec{\rho}_k$, which the user sends to the respective PIR servers.

A user queries for the record at index $\beta$ by choosing a vector of $r$ polynomials, $\vec{f} = \langle f_1, \ldots, f_r \rangle$, each of degree (at most) $t$, with uniformly random coefficients from $\mathbb{Z}_q$ for the non-constant terms. The constant term of $f_i$ is $\delta_{i\beta}$. In addition, the user chooses $k$ distinct server indices $I_1, \ldots, I_k$ and forms $k$ vectors of $\mathbb{Z}_q$ elements by evaluating $\vec{f}$ component-wise at the $k$ respective indices; that is $\vec{\rho}_j = \langle f_1(I_j), \ldots, f_r(I_j) \rangle$. The user forwards $\vec{\rho}_j$ to server $I_j$, while each server $I_j$ computes an $s$-element vector $R_j = \vec{\rho}_j \cdot D$ and returns it back to the user. Finally, the user computes the record at index $\beta$ from the $R_j$ by using Lagrange interpolation (and also Guruswami-Sudan list decoding [35] if some servers are Byzantine or malicious).

## 3.3 Threshold BLS signatures

The BLS signature scheme [8] is a 'short' signature scheme that uses a pairing function for signature verification. The signer's private signing key is a random integer $x \in \mathbb{Z}_q$, and the corresponding public verification key is $(\hat{g}, \hat{g}^x)$ (recall that $\hat{g}$ is a generator of $\mathbb{G}_2$). Given the signing key $x$ and a message $m$, the signature is computed via $\sigma = h^x$ where $h = \mathrm{hash}(m)$ is a cryptographic hash of $m$; the verification equation is $e(\sigma, \hat{g}) \stackrel{?}{=} e(h, \hat{g}^x)$. We use the $(k, \ell)$-threshold variant (and also the $(k, k)$-threshold variant) of BLS signatures; in both cases, the signing keys are evaluations of a polynomial of degree $k-1$ and the master secret is the constant term of this polynomial. The user recombines signature shares via Lagrange interpolation in the exponent. Note that by publishing the individual 'verification key shares' of each signer, threshold BLS signatures provide some level of robustness against Byzantine signers since each signature share can also be verified independently by using the signer's public verification key share.

## 3.4 Polynomial commitments

Polynomial commitments [40] allow a prover to form constant-sized commitments to polynomials in such a way that a verifier can later use these commitments to confirm evaluations of the committed polynomials without revealing any additional information about them. We use the $\texttt{PolyCommit}_{\text{DL}}$ construction of Kate *et al.* [40], which provides unconditional hiding if the commitment is opened to at most $t-1$ evaluations (for a degree-$t$ polynomial) and computational hiding under the discrete log (DL) assumption if the polynomial is opened at a $t^{\text{th}}$ point ($t+1$ or more openings is sufficient to interpolate and thus recover the committed polynomial), as well as their $\texttt{PolyCommit}_{\text{Ped}}$ construction, which offers unconditional hiding even when $t$ evaluations are revealed. Their constructions are based on the polynomial remainder theorem: if $f$ is a polynomial, then the remainder obtained by dividing $f(x)$ by $x-r$ equals $f(r)$; in other words, $x-r$ divides $f(x)-f(r)$. We describe how $\texttt{PolyCommit}_{\text{DL}}$ works, and refer the reader to [40] for details on the similar $\texttt{PolyCommit}_{\text{Ped}}$ construction. A commitment to the polynomial $f(x) = a_t x^t + \cdots + a_1 x + a_0$ in $\texttt{PolyCommit}_{\text{DL}}$ has the form $\mathcal{C}_f = (g^{\alpha^t})^{a_t} \cdots (g^{\alpha})^{a_1} g^{a_0} = g^{f(\alpha)}$, where $\alpha$ is secret, $g \in \mathbb{G}_1$ is a generator, and all bases (as well as $\hat{g}$ and $\hat{g}^{\alpha}$) are part of the commitment scheme's public key. If $\texttt{PolyCommit}_{\text{Ped}}$ commitments are used, then the public key includes the additional values $h^{\alpha^t}, \ldots, h^{\alpha}, h$, where $h \in \mathbb{G}_1$ is a generator whose discrete logarithm with respect to $g$ is unknown. To open an evaluation of $f$ at $x = r$, the prover invokes $\texttt{CreateWitness}(f, r)$, which outputs a polynomial commitment $w$ to the quotient obtained upon division of $f(x)-f(r)$ by $x-r$; the commitment $w$ is called a *witness*. The verifier can confirm the claimed evaluation by checking if $\texttt{Ver}(\mathcal{C}_f, r, f(r), w) = \left[e(\mathcal{C}_f, \hat{g}) \stackrel{?}{=} e(w, \hat{g}^{\alpha}/\hat{g}^{r}) \cdot e(g, \hat{g})^{f(r)}\right]$ is `true`. Note that in [40], polynomial commitments are constructed over a symmetric pairing, whereas in this work we construct our polynomial commitments over an asymmetric pairing, since we wish to reuse this pairing for *short* threshold BLS signatures.

Much like traditional discrete logarithm commitments [27] and Pedersen commitments [52], polynomial commitments are additively homomorphic and scalar multiplication of committed values can be computed by exponentiating the commitment by the scalar. We exploit both of these facts extensively in our protocols.

## 3.5 Zero-knowledge proofs

Our protocols employ several standard zero-knowledge proofs (ZKPs) from the literature: proofs of knowledge of a committed value [53], range proofs [9] to prove that a committed value is non-negative, proofs of knowledge of a discrete log representation of a number [11], and proofs that a commitment opens to the product of the openings of two other commitments [20]. We refer the interested reader to the respective papers for more details on each of these proofs, or to [16] for a self-contained treatment of all of the aforementioned proofs. We also use some efficient batch proof techniques [6, 7] to achieve practicality in our protocols; the rest of this section describes these batch proofs.

### 3.5.1 Proving equality of 1-out-of-r discrete logs.

We combine the batch verification techniques of Bellare *et al.* [6, 7] with Cramer *et al.*'s [24] technique for proving the disjunction of two or more propositions to yield efficiently verifiable proofs of equality of 1-out-of-$r$ discrete logarithms. That is, given bases $g$ and $h$ and two sets of inputs $g_1, g_2, \ldots, g_r$ and $h_1, h_2, \ldots, h_r$, to prove the predicate $\bigvee_{i=1}^{r} \left(\log_g(g_i) = \log_h(h_i)\right)$ without revealing which particular statements are true and which are false.

A detailed description of how to implement this proof is provided in Appendix A, and it is proved secure by Bellare *et al.* [7, Theorem 2.2].

### 3.5.2 Proving that a vector of commitments opens to a standard basis vector.

We introduce a new proof that allows one to efficiently prove that a vector of $r$ commitments opens to an $r$-dimensional standard basis vector (i.e., a length-$r$ vector containing a single 1 and the rest 0). Our proof uses a special case of the batch proof of equality of 1-out-of-$r$ discrete logarithms from the previous section as a subroutine. In particular, we use the special case in which $g_1 = g_2 = \cdots = g_r$ and the $h_i$ are all different, but $\log_{h^{\gamma}}(h_i) = a_i$ is known to the verifier, where $\gamma$ is randomly chosen by the prover and unknown to the verifier. In our protocol, the prover actually wishes to prove to the verifier that the *vector of polynomials* committed to by a vector of polynomial commitments evaluate to a standard basis vector at $x = 0$. However, modifying our approach as described here to handle other types of commitments (e.g., Pedersen commitments) is straightforward and modifying it to handle different evaluation points is trivial.

Let $\vec{a} = \langle a_1, \ldots, a_r \rangle \in_R (\mathbb{Z}_{2^{\kappa}})^r$. The key observation behind our approach is as follows: if $\vec{v}$ is a standard basis vector, then $\vec{v} \cdot \vec{a} = a_i$ for some $1 \leq i \leq r$; conversely, if $\vec{v}$ is not a standard basis vector, then with high probability $\vec{v} \cdot \vec{a} \neq a_i$ for any $1 \leq i \leq r$.

A detailed description of how to implement this proof is provided in Appendix B, and it is proved secure in the extended version of this paper [36, Appendix A].

### 3.5.3 Batch verification of evaluations of polynomial commitments at a common point.

In [40], Kate *et al.* show how to open *a single polynomial commitment* to *a set of evaluations* at the same time with a single witness element, a technique they call *batch opening*. We flip this proof around and show how to verify the evaluations of *a set of polynomial commitments* at *a single point*, a technique we call *batch verification*. Batch verification can be either *cooperative* or *non-cooperative*. The cooperative form of the protocol is interactive (though it can be made noninteractive using the Fiat-Shamir heuristic [28]), and uses only a single witness element, while the non-cooperative form is noninteractive and uses one witness element per commitment. As the name implies, the noncooperative form of batch verification does not require the prover's cooperation; i.e., only the verifier changes. In particular, the verifier combines all of the witnesses (and commitments) into a single witness (and commitment) at verification time to significantly reduce verification time at the cost of a negligible decrease in soundness.

A detailed description of how to implement cooperative batch verification is provided in Appendix C, and it is proved secure in the extended version of this paper [36, Appendix B].

# 4. CONSTRUCTIONS

We now describe the full details of our constructions. We develop our scheme incrementally in three steps. First, we describe how to convert Goldberg's multi-server PIR into SPIR. We then describe the basic single-payee PSPIR construction and show how to extend it to support access control lists. Finally, we discuss our approach to bookkeeping and use this to add support for top-$K$ replication and to construct multiple-payee PSPIR.

## 4.1 Symmetric PIR construction

The first step in our construction is to convert Goldberg's PIR scheme into SPIR; that is, we augment the scheme to enforce the

additional property that no query will ever reveal information about more than a single record, under some mild computational assumptions. This property implies that no coalition of users can use knowledge obtained from one or more PIR queries to learn any information about a record that they did not purchase in one of those queries. We accomplish this with the aforementioned proof that a vector of commitments opens to a standard basis vector. In particular, the user (querying servers with indices $I_1, \ldots, I_k$ for the record at index $\beta$ using his current wallet, `wallet`) does the following:

1. chooses $\vec{f} = \langle f_1, \ldots, f_r \rangle \in_R (\mathbb{Z}_q[x])^r$ such that $\deg(f_i) \leq t$ and $f_i(0) = \delta_{i\beta}$,
2. computes a vector $\vec{C}$ of component-wise $\texttt{PolyCommit}_{\text{DL}}$ commitments to the polynomials in $\vec{f}$,
3. computes $k$ vectors $\vec{\rho}_j = \langle f_1(I_j), \ldots, f_r(I_j) \rangle$ of evaluations of the polynomials in $\vec{f}$, and $k$ witnesses $w_j$ that attest to the fact that the $r$ evaluations in $\vec{\rho}_j$ are correct using cooperative batch verification (made noninteractive via Fiat-Shamir), for $1 \leq j \leq k$,
4. computes the set $S$ of commitment values from the proof that the polynomials committed to in $\vec{C}$ open to a standard basis vector at $x = 0$, and
5. sends $(\vec{C}, S, \texttt{wallet}, \vec{\rho}_j, w_j)$, to server $I_j$ for $1 \leq j \leq k$.

Note that each server receives different vectors of evaluation points and witnesses, but the same wallet and sets of commitments. Upon receiving these values, each server $I_j$

6. ensures that it has not seen `wallet` in an earlier query,
7. verifies that the evaluations in $\vec{\rho}_j$ are correct using cooperative batch verification (with witness $w_j$),
8. computes a $(k, \ell)$-threshold BLS signature share $\sigma_j$ on the value $\vec{C} \| S \| \texttt{wallet}$, and
9. sends $\sigma_j$ to the user.

After receiving signature shares from each server, the user

10. combines $\sigma_1, \ldots, \sigma_k$ into a signature $\sigma$ on $\vec{C} \| S \| \texttt{wallet}$,
11. computes the challenge $c = \text{hash}(\sigma)$ and uses this challenge to compute the set $V$ of responses to complete the aforementioned proof that the polynomials committed to in $\vec{C}$ open to a standard basis vector at $x = 0$, and
12. sends $(\sigma, V)$ to server $I_j$ for $1 \leq j \leq k$.

Upon receipt of this response, each server $I_j$

13. verifies that $\sigma$ is a valid signature on $\vec{C} \| S \| \texttt{wallet}$, and
14. computes $c = \text{hash}(\sigma)$ and checks if the responses in $V$ are valid responses for this challenge.

Recall that in Goldberg's PIR scheme, the user recovers the record by Lagrange interpolation at the point $x = 0$. It is apparent that the above proof convinces the database servers that the query only reveals information about a single record *when the responses are interpolated at the point* $x = 0$, but we must also consider a clever user that chooses the polynomials in his query non-randomly. In this case, the polynomials might be chosen such that interpolating at some other point $x = a$ reveals information about some other database record. This is unsurprising, since it is known that information-theoretic SPIR is impossible to achieve without some interaction between the servers, or a shared secret among them [31]. We thus introduce a shared secret key $sk$, known to all the servers but unknown to the users. (Note that the servers must already share a copy of the database, so requiring them to share an additional secret key is reasonable.) To prevent the above attack, server $I_j$

15. seeds a pseudorandom generator (PRG) with $F_{sk}(\vec{C})$ for some pseudorandom function family $F$,
16. uses the PRG to generate a common pseudorandom nonce and appends it to the database as an ephemeral $(r+1)^{\text{th}}$ database record for this query,[3]
17. uses the PRG to generate $t - 1$ random $\mathbb{Z}_q$ elements from which it forms a common pseudorandom polynomial $g \in \mathbb{Z}_q[x]$ of degree (at most) $t$ with $g(0) = 0$,
18. computes and appends $g(I_j)$ to $\vec{\rho}_j$, and
19. encodes the query response exactly as in Goldberg's original construction. Note that when encoding the response, the servers include the ephemeral $(r+1)^{\text{th}}$ record in the database, and also include their respective evaluations of the pseudorandom polynomial $g$ in the query as if the user had submitted it as part of his original query.

This last set of steps effectively *rerandomizes* the user's query. The user decodes the responses to his rerandomized query in the usual way (see §3.2).

Note that this SPIR construction preserves the $t$-privacy and $v$-Byzantine-robustness properties of the underlying PIR protocol; however, our approach to rerandomizing user queries prevents us from inheriting the scheme's optional $\tau$-independence property.

**Theorem 1.** *The above modifications convert Goldberg's multi-server information-theoretic PIR into multi-server SPIR. Query privacy is provided information theoretically against up to $t - 1$, and computationally against $t$ (under the DL assumption), colluding servers; the database's privacy is protected computationally (under the $t$-SDH assumption [40]).*

The proof of this theorem is in the extended version of this paper [36, Appendix C].

## 4.2 Single-payee PSPIR

Next, we extend the above SPIR construction to single-payee PSPIR. To do this, we augment the protocol as follows. First, we have the user compute a commitment, called a *receipt*, that encodes the price of the requested record under the price tier encoded in his wallet. The user proves in zero-knowledge that the receipt is well-formed (i.e., that it encodes the correct price) and that the balance in his wallet is sufficient to purchase the record at that price; once convinced by this proof, the database servers issue a threshold BLS signature on the user's receipt and wallet. The user can later exchange his wallet and this signed receipt with the bank to retrieve a new wallet for use in a future transaction. We also discuss how a user can recharge the balance in his wallet, and then point out a simple trick that enables the servers to enforce access control lists with only a slight modification to the PSPIR protocol.

[3] It is important that no other values are used to seed the PRG, since the user might otherwise replay $\vec{C}$ to retrieve a different nonce and potentially leak some information about other database records.

*Proving sufficient funds and computing the receipt.*

To compute the receipt, the user and each database server independently compute a commitment to the price of the record encoded in the user's query for each price tier. This is done by taking advantage of the homomorphic properties of polynomial commitments: each party computes the $T$ polynomial commitments $P_i = \prod_{j=1}^{r} C_j^{p_{ij}}$ for $1 \leq i \leq T$, where $p_{ij}$ is the $j^{\text{th}}$ component of $\vec{p}_i$. (Recall that $\vec{p}_i$ is the tier $i$ price list.) Note that $P_i$ is a commitment to a polynomial $f_{P_i} = \vec{f} \cdot \vec{p}_i$ whose constant coefficient is equal to the price of record $\beta$ in $\vec{p}_i$ (i.e., $f_{P_i}(0) = p_{\pi i}$). Next, the user

1. chooses $\gamma_0, \gamma_1 \in_R \mathbb{Z}_q$ and computes the $\texttt{PolyCommit}_{\text{Ped}}$ commitment $\mathcal{C}_P = P_\pi(h^\alpha)^{\gamma_1} h^{\gamma_0}$, where $\pi$ is the price tier encoded in `wallet`,

2. computes a Pedersen commitment $\texttt{Receipt}_P = g_{\text{T}}^{p_{\pi\beta}} h_{\text{T}}^{\gamma_0}$ and the witness $w_P = g^{\phi(\alpha)} h^{\gamma_1}$ where $\phi(x)$ is the quotient upon division of $f_{P_\pi}(x) - f_{P_\pi}(0)$ by $x - 0$,

3. computes $\Pi_P$, a ZKP of knowledge of $(x_1, \ldots, x_T, b)$ and $(\gamma_0, \gamma_1, p)$ such that $\mathcal{C}_P = P_1^{x_1} \cdots P_T^{x_T} (h^\alpha)^{\gamma_1} h^{\gamma_0}$ (recall that $x_i = \delta_{i\pi}$), $\texttt{Receipt}_P = g_{\text{T}}^{p} h_{\text{T}}^{\gamma_0}$, $b - p \geq 0$, and `wallet` encodes attributes $x_1, \ldots, x_T$ and balance $b$, and

4. sends the tuple $(\mathcal{C}_P, w_P, \texttt{Receipt}_P, \Pi_P)$ to server $I_j$ for $1 \leq j \leq k$.

Upon receiving these values, each server $I_j$

5. verifies that the proof $\Pi_P$ is correct,

6. checks if $e(\mathcal{C}_P, \hat{g}) \stackrel{?}{=} e(w_P, \hat{g}^\alpha) \cdot \texttt{Receipt}_P$,

7. computes a $(k, \ell)$-threshold BLS signature share $\varsigma_j$ on the value $\texttt{wallet} \| \texttt{Receipt}_P$, and

8. sends $\varsigma_j$ to the user.

If any verification step above fails, then the servers abort immediately; otherwise, the servers proceeds to process the user's query as usual. The user recombines the signature shares $\varsigma_j$, $1 \leq j \leq k$, to recover the signature $\varsigma$ on $\texttt{wallet} \| \texttt{Receipt}_P$.

*Remark* 1. To improve performance, our implementation of the above protocols (as described in [36, Appendix D]) diverges somewhat from the above descriptions. In particular, each of the ZKPs used in the SPIR construction are computed noninteractively using Fiat-Shamir, and the commitments and responses from this proof are transmitted to each server as early as possible, thus allowing the servers to begin verification before the user completes the proof. Then, instead of computing $\texttt{Receipt}_P$ noninteractively, each server issues a threshold BLS signature share on `wallet`, all common values from this proof, and all commitment values from the proof that $\texttt{Receipt}_P$ is valid; the user recombines these signature shares to produce a challenge value for this latter proof, then transmits the recombined signature and his responses to each server. This convinces the servers that they each saw the same wallet and query in the earlier SPIR proof.[4]

[4]The reason we describe the protocols as above is to make the SPIR construction secure *on its own*. In our implementation, the security of the SPIR relies on successful verification of the subsequent receipt proof (which is acceptable, since the servers do not respond to the query until they have verified both proofs).

*Refreshing the wallet.*

Before performing subsequent queries, the user must refresh his wallet with the bank. To do so, the user sends the tuple of values $(\varsigma, \texttt{Receipt}_P, \texttt{wallet})$ to the bank, who verifies that $\varsigma$ is a valid signature on $\texttt{wallet} \| \texttt{Receipt}_P$. If so, the user and the bank run the credential issuing protocol for the credential system (see §2) that represents the wallet. At the end of this protocol, the user has a new unlinkable (even to the bank) wallet $\texttt{wallet}'$ encoding the same tier as `wallet` and a balance equal to the price committed to in $\texttt{Receipt}_P$ subtracted from the balance encoded in `wallet`. The user may similarly *recharge* his wallet with additional funds by first 'purchasing' a receipt that encodes a negative price using, for example, a prepaid credit card. Note that in this procedure, the bank does not learn the balance in the new or the old wallet, or the price encoded in the receipt; in fact, the bank cannot even distinguish between a user that is refreshing his wallet and one who is recharging it.

*Supporting access control lists.*

We now describe a simple modification to implement access control lists atop our PSPIR construction. The idea is to impose a maximum balance $b_{\max}$ on users' wallets, and then require all users to prove that their new balance does not exceed $b_{\max}$ each time they refresh or recharge their wallets with the bank. The bank will refuse to issue any wallet without such a proof, thus ensuring that no user's balance ever exceeds $b_{\max}$. The remainder of the protocol remains unchanged, except that a price of $\perp$ in $\vec{p}_\pi$ is treated as a price of $b_{\max} + 1$, which, by our restriction above, no user can afford. Thus, this simple modification effectively prevents users in price tier $\pi$ from purchasing any record marked as $\perp$ in $\vec{p}_\pi$.

## 4.3 Bookkeeping

This section discusses our approach to bookkeeping. At a high level, our idea uses the additive and multiplicative homomorphic properties of Shamir secret shares to maintain and compute on shares of aggregate counts of the number of times each database record is retrieved (and at what price). In Goldberg's original PIR construction, the database servers do not maintain any state information — the only information they store is the actual database contents. We augment the database with two additional columns of state information; i.e., we require the database servers to store two $w$-bit words of state (Shamir secret shares) per database record, which are the length-$r$ vectors of shares $\vec{\mathbf{c}}_j = \langle [c_1]_q, \ldots, [c_r]_q \rangle$ and $\vec{\mathbf{d}}_j = \langle [d_1]_q, \ldots, [d_r]_q \rangle$.

Our initial idea was for servers to maintain a running sum of all queries they witness between successive bookkeeping operations. Unfortunately, this naive solution requires all servers to be involved in all queries, since if server $j$ aggregates a query but server $j'$ does not, their shares will be inconsistent and interpolate to an unpredictable value that does not reflect the actual number of queries per record. However, requiring all servers to participate in all queries hurts availability since the failure of a single database server would render the system inoperable. (It also hurts efficiency by requiring additional bandwidth and computational power be devoted to each query.) Moreover, even if all servers aggregate all queries, this only enables them to track the number of times each record is retrieved, but not the prices paid for them. We solve the first problem by having the user reveal which subset $Q$ of database servers are involved in each query, and then use this knowledge to convert the Shamir secret shares into additive shares; therefore, all servers must be online to compute on the shared bookkeeping data, but they do not need to be online during every query. To solve the second problem, we have the user (querying for record $\beta$ under price tier $\pi$)

include the additional vector of shares $\vec{\varrho}_j$ of $(p_{\pi\beta} \cdot \vec{1}_\beta)$ along with his query.[5] The servers convert both vectors into additive shares and aggregate them into $\vec{\mathbf{c}}_j$ and $\vec{\mathbf{d}}_j$, which are then vectors of shares of the number of times each record was retrieved, and the total price paid for those retrievals, respectively.

After each bookkeeping operation, database server $j$ reinitializes the auxiliary vectors $\vec{\mathbf{c}}_j$ and $\vec{\mathbf{d}}_j$ back to the length-$r$ zero vector, chooses a new private signing key $x_j$ for BLS signature generation, and publishes the corresponding verification key $(\hat{g}, \hat{g}^{x_j})$. By having each server choose its signing key independently, every subset $Q$ of $k$ servers has a unique public verification key $pk_Q$ for $(k,k)$-threshold BLS signatures; this key is easily computable using $Q$ via the expression $pk_Q = \prod_{I_j \in Q} \left(\hat{g}^{x_{I_j}}\right)^{\lambda_{Q,j}} \bmod q$ where $\lambda_{Q,j} = \prod_{I_i \in Q - \{I_j\}} I_j \cdot (I_j - I_i)^{-1} \bmod q$. Users then reveal $Q$ during each query, and the servers append an unambiguous string representation of $Q$ to the message resulting in the signature $\sigma$. The servers accept the signature as valid if and only if verification succeeds using verification key $pk_Q$, where $Q$ is the set of servers encoded in the message. Similarly, the servers encode $Q$ into the signature $\varsigma$ on the user's receipt, and the user transmits $Q$ along with the receipt to the bank.

This ensures that each server involved in a query sees a consistent set $Q$ of other servers; however, a malicious user may still disrupt the bookkeeping process by neglecting to send the tuple $(\sigma, V)$ to any nontrivial subset of $Q$ on Step 12 of the SPIR protocol. The servers that do receive $(\sigma, V)$ will aggregate the user's query into $\vec{\mathbf{c}}_j$ and $\vec{\mathbf{d}}_j$, while those that do not will not (thus resulting in inconsistent shares among the servers in $Q$). We therefore rely on the bank to facilitate atomicity to the query aggregation process. Instead of returning the regular query response $\vec{\rho}_j \cdot D$, the servers use the PRG (seeded with their shared secret and the user's wallet) to produce a common $\mathbb{Z}_q$ element for blinding, $\Gamma$, and return $\vec{\rho}_j \cdot D + \Gamma$. When the user sends $(\varsigma, Q, \texttt{wallet}, \texttt{Receipt}_P)$ to the bank, the bank 1) verifies that $\varsigma$ is a valid signature on $\texttt{wallet} \| \texttt{Receipt}_P \| Q$ using $pk_Q$, 2) computes and sends $\Gamma$ to the user, and 3) notifies each server in $Q$ that the transaction involving wallet is complete. At this point, all servers can safely update their aggregate shares.

### *Top-$K$ replication.*

Given the above modifications, supporting top-$K$ replication is straightforward. When a query $(\vec{C}, S, \texttt{wallet}, \vec{\rho}_j, w_j, Q)$ arrives at server $I_j$, it temporarily stores $\vec{\rho}_j$. Upon notification of the query's success from the bank, server $I_j$ accumulates the query by computing $\vec{\mathbf{c}}_j = \vec{\mathbf{c}}_j + \lambda_{Q,j} \cdot \vec{\rho}_j$. Computing the top-$K$ records from these shares is then a straightforward application of Burkhart and Dimitropoulos' [14] *privacy-preserving top-$K$* (PPTK) algorithm. The algorithm outputs the top-$K$ largest shares in $\vec{\mathbf{c}}_j$ without revealing any additional information about the value of any share. After the top $K$ are revealed, the servers replicate these to a smaller database and reinitialize $\vec{\mathbf{c}}_j$ to the length-$r$ zero vector.

### *Multiple-payee PSPIR.*

Supporting multiple payees in the tiered pricing model is slightly more involved than is supporting top-$K$ replication. Due to space constraints, we only address the simpler case of a single-tiered pricing scheme, and then briefly outline how to extend this approach to a system with multiple price tiers (and access control lists). Full details of the more general construction are available in the extended version of this paper [36].

The single-tier case simplifies the construction in two important ways: first, the user need not send or prove statements about the additional vector of shares $\vec{\varrho}_j$ as above and, second, the servers only need to store a single additional column of auxiliary information. Consider the above PSPIR construction with a single price list $\vec{p} = \langle p_1, \ldots, p_r \rangle$ for all users, and a set of $m$ CPs $\mathbf{CP} = \{\text{CP}_1, \ldots, \text{CP}_m\}$. For ease of presentation, we define $r_1 = 0$ and $r_{m+1} = r$, and assume that $\text{CP}_i$ $(1 \le i \le m)$ owns the records at indices $r_i + 1$ through $r_{i+1}$. The amount payable to $\text{CP}_i$ is then computed by summing the additive shares $\sum_{i=r_i+1}^{r_{i+1}} p_i \cdot [c_i]_q$.

With tiered pricing, the computation is conceptually similar but requires cooperation from the user. The user sends the vector of shares $\vec{\varrho}_j$ described above along with his query. The server chooses a random length-$r$ vector of challenges $\vec{c} \in (\mathbb{Z}_q)^r$ (via Fiat-Shamir) and computes the $T$ dot products $[e_i]_q = (\vec{\varrho}_j - \vec{p}_i \boxdot \vec{\rho}_j) \cdot \vec{c}$, where $\boxdot$ denotes component-wise multiplication of two vectors, for $1 \le i \le T$. If $\vec{\varrho}_j$ is well-formed, then $[e_i]_q = [0]_q$ for $i = \pi$; otherwise, $[e_i]_q \neq [0]_q$ for any $1 \le i \le T$ with overwhelming probability. The user then proves that $[e_i]_q = [0]_q$ for $i = \pi$ using a proof similar to the existing proof of correctness for the receipt. Once convinced, server $j$ is convinced that $\vec{\varrho}_j$ is a vector of commitments to the correct price, and can safely aggregate $\vec{\varrho}_j$ into $\vec{\mathbf{d}}_j$.

### *Bookkeeping frequency.*

Bookkeeping necessarily leaks some information about user queries. In the extreme case, where only a single user queries the database between bookkeeping operations, bookkeeping may completely reveal that user's query. At the other end of the spectrum, when every record is accessed hundreds or thousands of times between bookkeeping operations, the information leakage is minimal and likely not at all invasive to any user's privacy. However, prolonging the period between bookkeeping limits its usefulness in the case of top-K replication, and may be economically unacceptable in the case of multiple-payee PSPIR. Thus, a great deal of discretion is necessary on the part of the database servers in determining how often to run the bookkeeping protocols. For databases with consistently high usage, a simple bookkeeping schedule such as once per week or once per month may suffice, whereas those databases with lower usage may need to wait until the servers answer some threshold number of queries. In general, the bookkeeping policy is highly dependent both on the characteristics of the database and the business logic of its operators. We leave an in-depth investigation of this privacy-utility tradeoff as an important area for future work.

## 5. IMPLEMENTATION & EVALUATION

We implemented the protocols described in this paper using Ben Lynn's Pairing-Based Cryptography (PBC) [43] library with Aniket Kate's PBCWrapper [39] package for C++ Wrapper classes, Victor Shoup's NTL [55] with the GNU Multi-Precision Arithmetic Library [29] for multi-precision arithmetic, and OpenSSL [50] for hash functions (we use SHA-256). All experiments use a value of $\kappa = 40$ for the soundness parameter. Our PSPIR implementation is built atop Ian Goldberg's implementation of his PIR protocols, Percy++ [32]. For our evaluation, we implemented the protocol as a standalone add-on to Percy++, but we will later integrate it with the Percy++ library. We used the BigInteger-based version of Martin Burkhart's SEPIA library [13] and his PPTK [14] protocol for our top-$K$ replication benchmarks. All measurements were taken in Ubuntu Linux 10.04.1 LTS running on a machine with Dual Intel

[5] Note that $\vec{\varrho}_j \neq p_{\pi\beta} \cdot \vec{\rho}_j \bmod q$, which would reveal the price $p_{\pi p}$ to the servers; rather, $\vec{\varrho}_j$ is an independently chosen vector of shares. Of course, $\vec{\varrho}_j - p_{\pi\beta} \cdot \vec{\rho}_j$ yields shares of the length-$r$ zero vector, which is the property we exploit to let the user prove the well-formedness of $\vec{\varrho}_j$.

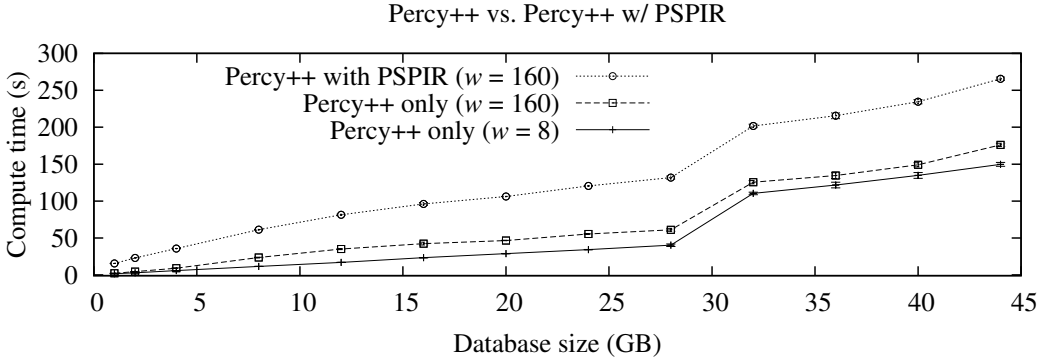


**Figure 1: Query execution time for Percy++ with and without PSPIR ($k = 4$, $t = 2$). The percent compute time attributable to the PSPIR enhancements decreases monotonically from $\approx 86\%$ for a 1 GB database down to $\approx 33\%$ for a 44 GB database. Percy++ starts carrying the extra overhead of disk reads after a 28 GB database, which exceeds available RAM. The $w = 8$ plot shows the execution time for Percy++ using its performance-optimal parameter choices, whereas the $w = 160$ plot shows Percy++ with parameters needed to ensure the security of PSPIR. The Percy++ with PSPIR plot shows the combined cost of Percy++ with $w = 160$ and the PSPIR enhancements. Error bars are plotted for all data points, but are small and may therefore be difficult to see. For comparison, downloading a 44 GB database in OT-based schemes takes over 11 hours at 9 Mbps, which is the average Internet bandwidth in Canada and the US [49].**

Xeon E5420 2.50 GHz CPUs and 32 GB memory. The value of $q$ (the order of the pairing groups and the modulus for the polynomial operations) was 160 bits long.

## 5.1 Experiments

We measured the performance of the PSPIR and top-$K$ replication protocols for various values of the PIR parameters $n$ (the size of the database), $b$ (the size of each record in the database), $k$ (the number of servers participating in each query), and $t$ (the number of servers that can collude without affecting query privacy).

**Experiment 1.** Our first experiment measures the computational overhead added to Percy++ by the PSPIR enhancements. We generated databases of sizes ranging from 1 GB to 44 GB containing random data and took measurements for both Percy++ and the Percy++ with PSPIR.

In Figure 1, we plot the results for parameters $k = 4$, $t = 2$, and $b = \sqrt{160n}$, which is the communication-optimal record size for this PIR scheme. We observe that PSPIR results in a moderate increase in compute times, with the percent compute time attributable to the PSPIR enhancements decreasing monotonically from about 86% for a 1 GB database down to just 33% for a 44 GB database. The upward bump just before 30 GB marks the point after which the database no longer fits in available memory. From that point on, every query bears more overhead from disk reads. In terms of communication overhead, PSPIR increases Percy++'s query size, which is itself just $k$ times the size of the retrieved record, by a multiplicative factor of about 5. However, it increases each server's response by only 46 bytes, which corresponds to two $\mathbb{G}_1$ elements (BLS signature shares).

The PSPIR compute time scales linearly with $k$ and $r = n/b$. In our implementation, the cost is independent of $t$ except for a one-time preprocessing step at the client. (The size of the long-term polynomial commitment public key and Percy++'s client-side compute time are also linear in $t$. This latter cost consistently accounted for less than one percent of overall compute time in our experiments.) The bottleneck operation for the client is computing commitments, while for the server it is processing the actual PIR query. In both cases, the bottleneck operation increases with $r$. Top-$K$ replication or bucketization [47] can enable one to trade some privacy to support larger values of $r$ with little to no additional compute time or communication overhead.

**Experiment 2.** Our next experiment evaluates the impact of top-$K$ replication; i.e., it studies the performance gains for the users when all queries for the $K$ most popular records go to the smaller replicated database. In this experiment, we assume that all records are physically replicated to a second set of database servers, thus increasing the maximum database size for which both the top-$K$ and non-top-$K$ records fit in physical memory. Alternatively, the database servers could simply publish a list of indices for the top-$K$ records and allow users to perform PIR on just this subset of the database; this would result in identical performance when the entire database fits in physical memory and somewhat lower performance otherwise. All trials of the top-$K$ experiment used a query distribution that we generated at random using a bounded Pareto distribution that satisfies the 80/20 rule; i.e., about 80% of queries are for just 20% of the database records, with the number of queries per record bounded between 0 and 10,000. As such, we use $K = \lfloor r/5 \rfloor$. We chose an 80/20 distribution because such distributions are commonly observed in the wild, but we emphasize that the actual performance gains that a database can expect from top-$K$ replication is highly dependent on the underlying query distribution. Figure 2 plots the mean query execution time for top-$K$ and non-top-$K$ queries, as well as the average (amortized) cost per query when 80% of queries are top-$K$ queries and 20% are not. Note that the average query execution time is well below that of Percy++ without PSPIR functionality (cf. Figure 1).

We also measured the cost to the servers of actually computing the top-$K$ using the SEPIA library. We found that using 160-bit secret shares results in poor performance compared to the benchmarks presented by Burkhart *et al.* [15]. Fortunately, in our case we can assert that all shared secrets are much smaller than $q/2$, which enables us to eliminate two-thirds of the computation in the bottleneck 'less than' computation. Furthermore, about 86% of the remaining computation time is spent generating random bit-wise secret shares modulo $q$. These random shares can be precomputed between top-$K$ computations, resulting in a respectable 57 less-than operations per second in our tests. This still leads to significant computation times to isolate the top-$K$ using Burkhart *et al.*'s PPTK algorithm [14] when the database size is large; thus, we fur-

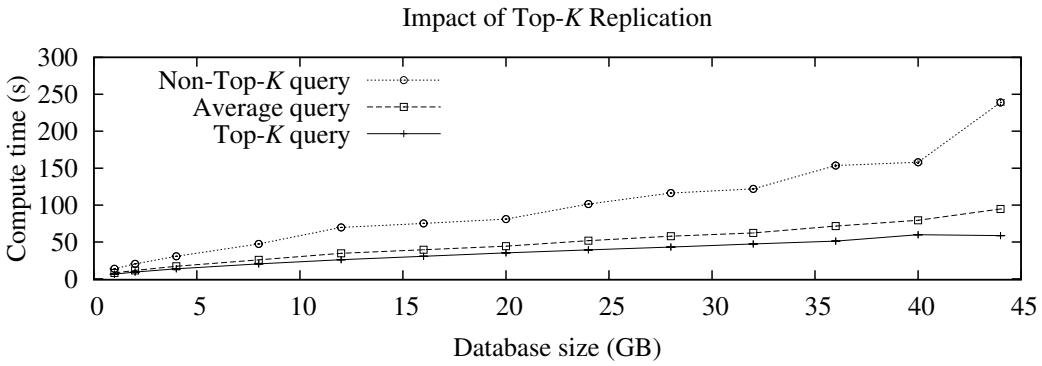


**Figure 2: Query execution time for Percy++ with PSPIR and top-$K$ replication ($k = 4$, $t = 2$, $K = \lfloor r/5 \rfloor$). Queries follow a bounded Pareto distribution that satisfies the 80/20 rule; thus, 80% of queries are for the top-$K$ entries and 20% are for the remaining $r - K$ entries (labelled 'Non-Top-$K$' on the plot). The average query cost for the top-$K$ replicated database ranges from just 51% that of an equivalent non-replicated database for a 1 GB database down to 36% for a 44 GB database. Error bars are plotted for all data points, but are small and may therefore be difficult to see.**

ther optimize the algorithm by relaxing PPTK to a 'top-$K$-ish' algorithm. This reduces computation times quite significantly. For example, with a 1 GB database ($r = 7328$ records) and a bounded 80/20 Pareto distribution our modified PPTK takes an average of about 9700 comparisons ($\approx 2.8$ minutes) to find the top 19–21% of records, whereas standard PPTK takes about 14600 comparisons ($\approx 4.2$ minutes); for a 20 GB database ($r = 32768$ records) this figure is about 33500 comparisons ($\approx 11.0$ minutes) for our relaxed PPTK versus 137500 comparisons ($\approx 40.2$ minutes) for standard PPTK, and for a 44 GB database ($r = 48603$ records) it is about 48605 comparisons ($\approx 14.2$ minutes) for our relaxed PPTK versus 197500 comparisons ($\approx 57.7$ minutes) for standard PPTK. Moreover, a large fraction of trials with standard PPTK had no solution (i.e., no unique top-$K$) and the algorithm therefore returned only the top-$K$-ish anyhow. Note that we generated our test sets using a bounded 80/20 Pareto distribution and then used the CDF for this distribution as an 'initial guess' in the PPTK algorithm. In practice, such an accurate guess will typically not be available and the actual number of comparisons will be greater than our predictions. Nonetheless, we feel confident in concluding that even for large databases with imperfect knowledge of the underlying query distribution, the cost of computing the top-$K$ will be quite reasonable.

# 6. RELATED WORK

The related bodies of work are symmetric private information retrieval (SPIR), oblivious transfer (OT), OT with access control (OTAC), and priced OT (POT).

OT schemes allow a database $X$ consisting of two records and a user holding an index $i \in \{0, 1\}$ to run a protocol that results in the user learning the $i^{\text{th}}$ record and no information the $(1 - i)^{\text{th}}$ record, while the database learns nothing about $i$. Unlike PIR and SPIR, however, OT schemes have no sublinear communication requirements. Brassard *et al.* [12] considered the more general notion of 1-out-of-$n$ OT, where the database holds $n$ records and the user learns the record at index $i$, and learns nothing about the remaining $n - 1$ records [51]; the database still learns nothing about $i$.

SPIR schemes [31] address the honest-user assumption of PIR by additionally preserving database privacy so that dishonest users cannot learn any information about other database records beyond the record retrieved. All existing communication-efficient 1-out-of-$n$ OT schemes are essentially single-server SPIR, whereas all existing communication-efficient distributed OT schemes [31] (i.e., two or more servers) of 1-out-of-$n$ OT schemes are essentially multi-server SPIR. The first work on preserving database privacy against dishonest users in a multi-server PIR setting was by Gertner *et al.* [31]. They propose a single-round $\ell$-server SPIR scheme with communication complexity $O(\log n \cdot n^{1/(2\ell-1)})$ for $\ell \geq 2$ and a $O(\log n)$-server scheme with communication complexity $O(\log^2 n \cdot \log \log n)$. Kushilevitz and Ostrovsky [42] briefly discuss how to convert their single-server PIR into SPIR using general zero-knowledge proof techniques, however they propose no concrete constructions. No existing SPIR scheme simultaneously provides support for both access control and tiered pricing.

Several OTAC schemes [17, 23, 57] were recently proposed. As with our approach, these schemes typically consist of three parties: user, database, and issuer. The issuer provides users needing access to the database with credentials encoding the access rights of users as an attribute set. The database encrypts its content under an access policy specific to each record and makes the encrypted contents available to users for download. A user with a valid credential can run the OTAC protocol with the database to obtain a decryption key for a particular record. After the protocol, the database learns that a user with a valid credential has obtained a key, but learns nothing about the user's credential or the decryption key issued. User's download the entire encrypted database and use the key obtained to decrypt the desired record. Zhang *et al.* [57] used attribute-based encryption to specify record-level access policies in disjunctive form without requiring duplication of database records. However, these schemes do not consider an economic model where users pay for each record and their high communication overhead makes them considerably more costly than SPIR.

POT schemes [1, 18] were originally introduced by Aiello *et al.* [1] to explore the difference between physical goods requiring close monitoring of inventory level and digital goods that are essentially of unlimited supply. In their model, users first deposit some money with the database and then proceed to buy multiple digital goods from the database, such that the total price of purchased goods does not exceed the user's deposit/balance. The database does not learn which digital goods the user has purchased. However, since the database tracks the users' accounts, all queries by a single user are linkable; thus, the approach lacks the anonymity properties that we seek. This enables the database server to deduce the number of digital goods a particular user has purchased, the average price of those purchases, and the user's spending pattern [18]. Furthermore, the scheme provides no way for users to

recharge their balance, which means that when a user's balance becomes lower than the price of any record, the remaining balance is rendered useless. Camenisch *et al.* [18] address these problems by encoding users' wallets in an anonymous credential so that the database is no longer required to maintain user-specific state information; as a result, user purchases become unlinkable. They also lay out an extension that makes use of a trusted third party to facilitate a *fair purchase* protocol; i.e., an optimistic fair exchange protocol to prevent the database server from cheating by not sending the correct decryption key (or wallet) to the user.

All of the above priced and access-control-capable OT and SPIR schemes lack some ingredients necessary for deployment in a practical setting. The foremost missing ingredient is the right combination of functionalities for access control, tiered pricing, support for multiple payees, sublinear communication complexity, and availability of practical implementations. The SPIR schemes [31, 42] provide no pricing or access control functions. OT schemes (i.e., 1-out-of-$n$) have prohibitively expensive communication costs and require a static encrypted database, which potentially breaks other applications using the same database. In particular, existing OTAC schemes [17, 23, 57] do not provide pricing functions, while the POT schemes [1, 18], on the other hand, provide no access control functions. Note that one cannot simply adopt our approach of setting the price of a record higher than the maximum wallet balance, since all users in these schemes pay according to the same price list (and thus would automatically have the same access privileges). Moreover, no existing POT scheme supports multiple payees selling goods through a common database.

Much like our top-$K$ replication strategy, a few research efforts have also focused on increasing the practicality of PIR by finding ways around PIR's linear computational requirements. Beimel *et al.* [5] propose preprocessing, which enables PIR servers to answer queries with only sublinear computation by precomputing and storing some extra information (the size of which is polynomial in the database size $n$). Ishai *et al.* [37] propose a different approach called batch coding that — while still requiring linear computation — enables the servers to process several PIR queries by the same user simultaneously, thus providing an amortized cost per query that is strictly smaller than $n$. Nearest to our own work, Olumofin and Goldberg [47] recently proposed an approach to indexing and partitioning large databases into highly diverse bucket portions that users can query independently. This approach makes querying such large databases with PIR practical, and simplifies the tradeoff between privacy and runtime; however, it does not include any way for the database servers to dynamically learn about and exploit the relative popularities of individual records to improve performance, as does the top-$K$ approach taken in this work.

# 7. CONCLUSION

We have extended Goldberg's multi-server information-theoretic PIR with a suite of protocols for privacy-preserving e-commerce. Our protocols add support for tiered pricing with multiple payees, group-based access control lists with record-level granularity, and dynamic top-$K$ replication, while preserving the sublinear communication complexity of PIR; no other scheme for priced retrieval using PIR or OT supports tiered pricing, multiple payees, access control, or dynamic replication. We have implemented the single-payee variant of our PSPIR protocol atop Percy++, an open-source implementation of Goldberg's PIR scheme, and evaluated its performance empirically. We also evaluated the cost and impact of top-$K$ replication. Our measurements indicate that this combination of protocols results in performance that is acceptable for deployment in real-world e-commerce applications. Furthermore, the extensive functionality of our protocols, SPIR's sublinear communication costs, and the ability to operate on an unencrypted database, makes our approach more practical than competing OT-based approaches. For future work, we intend to optimize our implementation and add full support for multiple payees (which we do not expect to significantly alter the runtime), and to incorporate our protocols into Percy++.

**Acknowledgements.** We are extremely thankful to Martin Burkhart for his assistance with SEPIA. This research is supported by NSERC, OGS, Mprime, and a Cheriton Graduate Scholarship.

# REFERENCES

[1] W. Aiello, Y. Ishai, and O. Reingold. Priced Oblivious Transfer: How to Sell Digital Goods. In *Proceedings of EUROCRYPT 2001*, Innsbruck, Austria, May 2001.

[2] D. Asonov. *Querying Databases Privately: A New Approach To Private Information Retrieval*, volume 3128 of *LNCS*. Springer, 2004.

[3] M. H. Au, W. Susilo, and Y. Mu. Constant-Size Dynamic $k$-TAA. In *Proceedings of SCN 2006*, Maiori, Italy, September 2006.

[4] J. Bar-Ilan and D. Beaver. Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction. In *Proceedings of PODC 1989*, Edmonton, AB, August 1989.

[5] A. Beimel, Y. Ishai, and T. Malkin. Reducing the Servers' Computation in Private Information Retrieval: PIR with Preprocessing. In *Proceedings of CRYPTO 2000*, Santa Barbara, CA, August 2000.

[6] M. Bellare, J. A. Garay, and T. Rabin. Batch Verification with Applications to Cryptography and Checking. In *Proceedings of LATIN 1998*, Campinas, Brazil, April 1998.

[7] M. Bellare, J. A. Garay, and T. Rabin. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In *Proceedings of EUROCRYPT 1998*, Espoo, Finland, May 1998.

[8] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. *Journal of Cryptology*, 17(4):297–319, January 2004.

[9] F. Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In *Proceedings of EUROCRYPT 2000*, Bruges, Belgium, May 2000.

[10] S. Brands. Restrictive Blinding of Secret-Key Certificates. In *Proceedings of EUROCRYPT 1995*, Saint-Malo, France, May 1995.

[11] S. A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000.

[12] G. Brassard, C. Crépeau, and J.-M. Robert. All-or-Nothing Disclosure of Secrets. In *Proceedings of CRYPTO 1986*, Santa Barbara, CA, 1986.

[13] M. Burkhart. SEPIA: Security through Private Information Aggregation. Version 0.8.2.

[14] M. Burkhart and X. Dimitropoulos. Fast Privacy-Preserving Top-$k$ Queries using Secret Sharing. In *Proceedings of ICCCN 2010*, Zurich, Switzerland, August 2010.

[15] M. Burkhart, M. Strasser, D. Many, and X. A. Dimitropoulos. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *Proceedings of USENIX Security 2010*, Washington, DC, August 2010.

[16] J. Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zurich, 1998. Reprint as vol. 2 of *ETH Series in Information Security and Cryptography*, Hartung-Gorre Verlag, Konstanz, 1998.

[17] J. Camenisch, M. Dubovitskaya, and G. Neven. Oblivious Transfer with Access Control. In *Proceedings of ACM CCS 2009*, Chicago, IL, November 2009.

[18] J. Camenisch, M. Dubovitskaya, and G. Neven. Unlinkable Priced Oblivious Transfer with Rechargeable Wallets. In *Proceedings of FC 2010*, Tenerife, Canary Islands, January 2010.

[19] J. Camenisch, M. Dubovitskaya, G. Neven, and G. M. Zaverucha. Oblivious Transfer with Hidden Access Control Lists. In *Proceedings of PKC 2011*, Taormina, Italy, March 2011.

[20] J. Camenisch and M. Michels. Proving in Zero-Knowledge that a Number Is the Product of Two Safe Primes. In *Proceedings of EUROCRYPT 1999*, Prague, Czech Republic, May 1999.

[21] B. Chor, N. Gilboa, and M. Naor. Private Information Retrieval by Keywords. Cryptology ePrint Archive, Report 1998/003, 1998. `http://eprint.iacr.org/`.

[22] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. In *Proceedings of FOCS 1995*, Milwaukee, WI, October 1995.

[23] S. E. Coull, M. Green, and S. Hohenberger. Controlling Access to an Oblivious Database Using Stateful Anonymous Credentials. In *Proceedings of PKC 2009*, Irvine, CA, March 2009.

[24] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Proceedings of CRYPTO 1994*, Santa Barbara, CA, August 1994.

[25] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation. In *Proceedings of TCC 2006*, New York, NY, March 2006.

[26] B. Doe. The Kindle in Australia, October 2009.

[27] P. Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *Proceedings of FOCS 1987*, Los Angeles, CA, October 1987.

[28] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Proceedings of CRYPTO 1986*, Santa Barbara, CA, 1986.

[29] Free Software Foundation. The GNU Multiple Precision (GMP) Arithmetic Library. Version 5.0.1.

[30] Y. Gertner, S. Goldwasser, and T. Malkin. A Random Server Model for Private Information Retrieval. In *Proceedings of RANDOM 1998*, Barcelona, Spain, October 1998.

[31] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting Data Privacy in Private Information Retrieval Schemes. In *Proceedings of STOC 1998*, Dallas, TX, May 1998.

[32] I. Goldberg. Percy++ / PIR in C++. Version 0.7.1.

[33] I. Goldberg. Improving the Robustness of Private Information Retrieval. In *Proceedings of IEEE S&P 2007*, Oakland, CA, May 2007.

[34] S. Guha, B. Cheng, and P. Francis. Privad: Practical Privacy in Online Advertising. In *Proceedings of NSDI 2011*, Boston, MA, March 2011.

[35] V. Guruswami and M. Sudan. Improved Decoding of Reed-Solomon and Algebraic-Geometric Codes. In *Proceedings of FOCS 1998*, Palo Alto, CA, November 1998.

[36] R. Henry, F. Olumofin, and I. Goldberg. Practical PIR for Electronic Commerce. Tech. Report CACR 2011-04, University of Waterloo, 2011.

[37] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Batch Codes and Their Applications. In *Proceedings of STOC 2004*, Chicago, IL, June 2004.

[38] A. Juels. Targeted Advertising... And Privacy Too. In *CT-RSA*, San Francisco, CA, April 2001.

[39] A. Kate. PBCWrapper: C++ Wrapper Classes for the Pairing-Based Cryptography Library. Version 0.8.0.

[40] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-Size Commitments to Polynomials and Their Applications. In *Proceedings of ASIACRYPT 2010*, Singapore, December 2010.

[41] A. Kate, G. M. Zaverucha, and I. Goldberg. Polynomial Commitments. Tech. Report CACR 2010-10, University of Waterloo, 2010.

[42] E. Kushilevitz and R. Ostrovsky. Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval. In *Proceedings of FOCS 1997*, Miami Beach, FL, October 1997.

[43] B. Lynn. PBC Library: The Pairing-Based Cryptography Library. Version 0.5.11.

[44] D. McCullagh. Amazon Fights Demand for Customer Records. In *CNET News*. April 2010. `http://news.cnet.com/8301-13578_3-20002870-38.html`.

[45] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[46] T. Nishide and K. Ohta. Constant-Round Multiparty Computation for Interval Test, Equality Test, and Comparison. *IEICE Transactions*, 90-A(5):960–968, 2007.

[47] F. Olumofin and I. Goldberg. Preserving Access Privacy Over Large Databases. Tech. Report CACR 2010-33, University of Waterloo, 2010.

[48] F. G. Olumofin and I. Goldberg. Privacy-Preserving Queries over Relational Databases. In *Privacy Enhancing Technologies*, Berlin, Germany, July 2010.

[49] Ookla Net Metrics. Canada and US Source Data. `http://www.netindex.com/source-data/`.

[50] OpenSSL Project. OpenSSL: The Open Source toolkit for SSL/TLS. Version 1.0.0.

[51] R. Ostrovsky and W. E. Skeith III. A Survey of Single-Database Private Information Retrieval: Techniques and Applications. In *Proceedings of PKC 2007*, Beijing, China, April 2007.

[52] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Proceedings of CRYPTO 1991*, Santa Barbara, CA, August 1991.

[53] C.-P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Proceedings of CRYPTO 1989*, Santa Barbara, CA, August 1989.

[54] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[55] V. Shoup. NTL: A Library for doing Number Theory. Version 5.5.2.

[56] V. Toubiana, H. Nissenbaum, A. Narayanan, S. Barocas, and D. Boneh. Adnostic: Privacy Preserving Targeted Advertising. In *Proceedings of NDSS 2010*, San Diego, CA, February 2010.

[57] Y. Zhang, M. H. Au, D. S. Wong, Q. Huang, N. Mamoulis, D. W. Cheung, and S.-M. Yiu. Oblivious Transfer with Access Control : Realizing Disjunction without Duplication. In *Proceedings of Pairing 2010*, Yamanaka Hot Spring, Japan, December 2010.

# APPENDIX

## A. PROVING EQUALITY OF 1-OUT-OF-$r$ DISCRETE LOGARITHMS

Let $g$ and $h$ be (known) generators of a group $\mathbb{G}$ (of order $q$) with $\log_g(h)$ unknown to the verifier, and let $g_1, g_2, \ldots, g_r$ and $h_1, h_2, \ldots, h_r$ be given. The proof works as follows:

**Prover knows**: $x = \log_g g_j = \log_h h_j$ and index $j$

**Verifier learns**: that $\log_g g_{j'} = \log_h h_{j'}$ for at least one $j'$

1. The prover chooses $\gamma_1, \ldots, \gamma_r \in_R \mathbb{Z}_q$ and $c'_1, \ldots, c'_r \in \mathbb{Z}_{2^\kappa}$, then computes and sends the commitments $\eta_i = g^{\gamma_i} g_i^{c'_i \bar{\delta}_{ij}}$ and $\zeta_i = h^{\gamma_i} h_i^{c'_i \bar{\delta}_{ij}}$ to the verifier, for $1 \leq i \leq r$.

2. The verifier chooses and sends $c \in_R \mathbb{Z}_{2^\kappa}$ to the prover.

3. The prover sets $c_i = c'_i$ and $\upsilon_i = \gamma_i$ for $i \in [1, r] - \{j\}$ and computes $c_j = c - \sum_{i=1}^{r} c'_i \bar{\delta}_{ij} \bmod 2^\kappa$ and $\upsilon_j = \gamma_j - c_j x \bmod q$, then sends the pair $(\vec{c}, \vec{\upsilon})$ to the verifier, where $\vec{c} = \langle c_1, \ldots, c_r \rangle$ and $\vec{\upsilon} = \langle \upsilon_1, \ldots, \upsilon_r \rangle$.

4. The verifier chooses $\vec{b} = \langle b_1, \ldots, b_r \rangle \in_R (\mathbb{Z}_{2^\kappa})^r$ and computes $\upsilon = \vec{\upsilon} \cdot \vec{b}$. The verifier accepts if and only if $\prod_{i=1}^{r} \eta_i^{b_i} \stackrel{?}{=}$

$g^{\upsilon}\left(\prod_{i=1}^{r} g_i^{b_i c_i}\right)$, $\prod_{i=1}^{r} \zeta_i^{b_i} \stackrel{?}{=} h^{\upsilon}\left(\prod_{i=1}^{r} h_i^{b_i c_i}\right)$ and $c \stackrel{?}{\equiv} \sum_{i=1}^{r} c_i \bmod 2^{\kappa}$ all hold.

Note that the above batch verification equation is more efficient than checking each of the $r$ verification equations independently, since both $b_i$ and $c_i$ are *short* exponents; moreover, in our own application (see §3.5.2), we take advantage of some properties of the special case we are proving to further reduce verification costs.

# B. PROVING THAT A VECTOR OF COMMITMENTS OPENS TO A STANDARD BASIS VECTOR

**Prover knows**: a length-$r$ vector of polynomials $\vec{f} \in (\mathbb{Z}_q[x])^r$
**Verifier learns:** a length-$r$ vector $\vec{C}$ of component-wise commitments to polynomials in $\vec{f}$ and that $\vec{f}$ evaluates component-wise to a standard basis vector at $x = 0$

1. The prover computes and sends $\vec{C}$ to the verifier.
2. The verifier chooses a vector of challenges $\vec{a} \in_R (\mathbb{Z}_{2^\kappa})^r$ and sends it to the prover; meanwhile, the verifier computes $\mathcal{C}_a = \prod_{i=1}^{r} C_i^{a_i}$, where $C_i$ and $a_i$ are the $i^{\text{th}}$ components of $\vec{C}$ and $\vec{a}$, respectively. Note that $\mathcal{C}_a$ is a commitment to the dot product $f_a = \vec{f} \cdot \vec{a}$.
3. The prover computes the dot product $f_a = \vec{f} \cdot \vec{a}$ and engages in a zero-knowledge proof of knowledge of the evaluation of $f_a$ at $x = 0$ with the verifier, such as by using the technique described in [41, Appendix D].
4. Let $Y = g_{\mathrm{T}}^{\gamma \cdot y}$ be the (blinded) commitment to $y = f_a(0)$ from this last proof of knowledge. The prover sends $\nu = h^{\gamma}$ together with proof that $\gamma$ is the same randomness used to blind $Y$, and engages in a batch proof of equality of 1-out-of-$r$ discrete logarithms to prove $\bigvee_{i=1}^{r} \left(\log_{g_{\mathrm{T}}} Y = \log_h \nu^{a_i}\right)$.

*Remark* 2. Because we are dealing with the special case of the batch proof of equality of 1-out-of-$r$ discrete logarithms in which $g_1 = g_2 = \cdots = g_r = Y$ and $\log_{\nu}(h_i) = a_i$ is known to the verifier, the following optimizations apply: instead of checking

$$\prod_{i=1}^{r} \eta_i^{d_i} \stackrel{?}{=} g^{\upsilon}\left(\prod_{i=1}^{r} g_i^{c_i d_i}\right) \text{ and}$$

$$\prod_{i=1}^{r} \zeta_i^{d_i} \stackrel{?}{=} h^{\upsilon}\left(\prod_{i=1}^{r} h_i^{c_i d_i}\right)$$

in the verification equation, the verifier computes $w_1 = \vec{c} \cdot \vec{d} \bmod q$ and $w_2 = \sum_{i=1}^{r} a_i c_i d_i \bmod q$ and checks if

$$\prod_{i=1}^{r} \eta_i^{d_i} \stackrel{?}{=} g^{\upsilon} Y^{w_1} \text{ and}$$
$$\prod_{i=1}^{r} \zeta_i^{d_i} \stackrel{?}{=} h^{\upsilon} \nu^{w_2}.$$

This reduces the cost of verification from 2 full length exponentiations and $6r$ 'short' exponentiations (i.e., exponentiations with $\kappa$-bit exponents) to 4 full length exponentiations and $2r$ short exponentiations.

# C. VERIFYING EVALUATIONS OF POLYNOMIAL COMMITMENTS AT A COMMON POINT

**Prover knows**: a length-$r$ vector of polynomials $\vec{f} \in (\mathbb{Z}_q[x])^r$
**Verifier learns:** a length-$r$ vector $\vec{C}$ of component-wise commitments to polynomials in $\vec{f}$, a component-wise evaluation $\vec{\rho}$ of $\vec{f}$ at $x = x_0$, and the evaluation point $x_0$

1. The prover computes and sends $\vec{C}$ and $\vec{\rho}$ to the verifier.
2. The verifier chooses $\vec{a} = \langle a_1, \ldots, a_r \rangle \in_R (\mathbb{Z}_{2^\kappa})^r$ and sends it to the prover; meanwhile, the verifier computes the dot product $\rho_a = \vec{\rho} \cdot \vec{a}$ and the commitment $\mathcal{C}_a = \prod_{i=1}^{r} C_i^{a_i}$.
3. The prover computes the dot product $f_a = \vec{f} \cdot \vec{a}$ and the witness $w_a = \texttt{CreateWitness}\,(f_a, x_0)$, then sends $w_a$ to the verifier.
4. The verifier checks if $\texttt{Ver}\,(\mathcal{C}_a, x_0, \rho_a, w_a) \stackrel{?}{=} \texttt{true}$.

The noninteractive form of batch verification works similarly, except the prover computes and sends a vector of witnesses to the verifier (one for each polynomial commitment), and *the verifier* combines the witnesses locally by computing $w_a = \prod_{i=1}^{r} w_i^{a_i}$; in particular, the vector $\vec{a}$ is local to the verifier and the prover never sees it.