Efficient Computationally Private Information Retrieval From Anonymity or Trapdoor Groups

Jonathan Trostle and Andy Parrish
Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723

Abstract

A Private Information Retrieval (PIR) protocol allows a database user, or client, to obtain information from a data- base in a manner that prevents the database from knowing which data was retrieved. Although substantial progress has been made in the discovery of computationally PIR (cPIR) protocols with reduced communication complexity, there has been relatively little work in reducing the computational complexity of cPIR protocols. In particular, Sion [18] argues that existing cPIR protocols are slower than the trivial PIR protocol (in overall performance). In this paper, we present a new family of cPIR protocols with a variety of security and performance properties. Our protocols enable much lower CPU overhead for the database server. When the database is viewed as a bit sequence, only addition operations are performed by the database server. We can view our protocol as a middle ground between the trivial protocol (fastest possible computational complexity and slowest possible communication complexity) and protocols such as Gentry-Ramzan [6] (fast communication complexity but slower computational complexity). This middle ground enjoys a much better overall performance. The security of the general version of our protocol depends on either a trapdoor group assumption or sender anonymity [14], and we present two specialized versions, the first of which depends on the trapdoor group assumption, and the second which depends on the sender anonymity assumption. We may view both Gentry-Ramzan and our cPIR protocol as instances of a more general new construct: the *trapdoor group*. In a trapdoor group, knowledge of the trapdoor allows efficient computation of an inversion problem, such as computing discrete logarithms. Without the trapdoor, it is computationally hard to solve the inversion problem. For our protocol, we assume, roughly speaking, that given only the elements $be_1, \ldots, be_t$ in the group $\mathbb{Z}_m$, where $e_i < m/t$ and t is small, it is hard to compute low order bits of the group order m. One version of our cPIR protocol depends only on sender anonymity, which to our knowledge, is the first cPIR protocol to depend only on an anonymity assumption. Our prototype implementation shows that our performance compares favorably with existing cPIR protocols.

1 Introduction

Private Information Retrieval (PIR) is any protocol that allows a database user, or client, to obtain information from a database in a manner that prevents the database from knowing which data was retrieved. Typically, the database is modeled as an n-bit string and the user wants to obtain bit i in the string, such that i remains unknown to the database. More generally, the database is modeled as a sequence of blocks, and the client will retrieve the ith block, where i remains unknown to the database. The trivial protocol consists of downloading the entire database, which clearly preserves privacy. The goal of PIR protocols is to obtain better performance (both computational and communication costs) than the trivial protocol, yet maintain privacy.

There is a wealth of information in a user's database access patterns. In many cases, these patterns give information about a user's or organization's future plans. For example, consider a pharmaceutical corporation's plans for future patent applications and potential database searches in connection with those plans.

In this work, we consider applications where the privacy of the database is not protected. In particular we consider private information retrieval from either a public database or a database with a group of subscribers. Although clients can download the entire database, this takes too long for a large database. Thus PIR that protects only the user is desirable in this scenario.

1.1 Prior work

Chor [4] introduced PIR. Their paper presents information-theoretic PIR protocols where the database is replicated and the replicas are not allowed to communicate; they show that single database information-theoretic PIR with no leakage does not exist. In general, the security of information-theoretic PIR protocols requires that the replicas not communicate.

Kushilevitz and Ostrovsky [10] presented the first single database *computational* PIR (cPIR) scheme where the security is established against a computationally bounded adversary. Their scheme is based on the quadratic residuosity assumption. More generally cPIR schemes can be constructed based on homomorphic public key cryptosystems. [7] showed how to protect database privacy as well. The work in [3, 15, 12, 6] demonstrated schemes with polylogarithmic communication complexity. On the other hand, there has been little work aimed at improving computational costs on the server (possibly since the server cannot have better than $\Omega(n)$ computational complexity). Sion and Carbunar [18] presented performance results claiming that all of the existing schemes are slower than the trivial protocol, mainly due to the bottleneck caused by the server's computational load. Ishai et. al. [9] explored using anonymity in order to create more efficient cPIR protocols; their protocol depends on both sender anonymity and the noisy curve reconstruction problem (whereas our anonymity based cPIR protocol depends only on sender anonymity). Aguilar-Melchor and Gaborit [1] give a fast cPIR protocol that is based on coding theory and lattice assumptions.

1.2 Our results

First we will present an anonymity based cPIR protocol that depends only on sender anonymity. We then present the trapdoor group based cPIR protocol where security depends on the hidden modular group assumption (explained below). Finally, we will present the generalized cPIR protocol which includes the first two as special cases; here security assumes either sender anonymity or the hidden modular group assumption.

1.2.1 Anonymity Based cPIR Protocol

Given sender anonymity, we can apply the "split and mix" technique from [9]. The user will split their query into a small number of subqueries and mix them with other subqueries before sending to the database server. These subqueries will be mixed with other user cPIR subqueries. By mixing the subqueries, the adversary must guess the right subqueries to add back together to obtain the original query. This problem becomes difficult as the number of users and queries increases. As the number of users grows, the amount of noise added by each user (the number of subqueries w that each query will be split into) can be reduced.

An overview of the anonymity based cPIR protocol (for the bit vector case) is as follows:

1. We view the database as a $\sqrt{n}$ by $\sqrt{n}$ bit matrix, with n total entries. For example, each row could be viewed as a block of data (set of bits).

2. The user will obtain a single block by sending a PIR query. The user creates a bit vector B with a one bit in the position corresponding to the row the user will query, and zero bits in the other positions. Of course, sending this vector would not protect the user's privacy. Therefore, the user adds (over the group $\mathbb{Z}_2$) a uniformly generated bit vector U to B to obtain the bit vector R. R will be sent to $\mathcal{DB}$ as a subquery along with other "noise" subqueries. These subqueries will be a set of vectors $D_1, \ldots, D_w$ such that $\sum_{i=1}^{w} D_i = U$.

3. $\mathcal{DB}$ receives the subqueries. Of course, if these are the only subqueries received by $\mathcal{DB}$, it can sum them and obtain B. We obtain security via sender anonymity and assuming that many subqueries are sent by the original user and other users corresponding to many queries. The subqueries are all uniformly distributed and indistinguishable to $\mathcal{DB}$. Thus it becomes computationally infeasible for $\mathcal{DB}$ to pick the right queries to add together to obtain B.

4. For each subquery, $\mathcal{DB}$ will sum the elements (mod 2) in each column corresponding to the one bits in the received subquery. The resulting bit vector is returned to the user. Note: $\mathcal{DB}$ could sum the elements as integers and allow the user to mod the result.

5. The user will take the correct set of returned bit vectors and sum them (mod 2) in order to obtain the desired row of $\mathcal{DB}$.

Example: Here we give a toy example for the protocol. We consider the database as a 4x4 bit matrix $\mathcal{DB}$ (or as consisting of four row elements each with 4 bits).

$$DB = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

Let $B_1 = [0\ 0\ 1\ 0]$ which indicates that the user desires to obtain the 3rd row (record), and let $B_2 = [0\ 0\ 0\ 1]$ which indicates that the same or a 2nd user desires to obtain the 4th row in the $\mathcal{DB}$. The first user generates uniformly distributed vectors $U_1 = [1\ 1\ 0\ 0]$, $D_1 = [0\ 1\ 0\ 0]$, and $D_2 = [0\ 0\ 1\ 1]$. Then $D_3 = [1\ 0\ 1\ 1]$ is selected such that $\sum_i D_i = U_1$. The first query B_1 is split into subqueries $R_1 = B_1 \bigoplus U_1$, D_1, D_2, and D_3. Similarly, the second user generates uniformly distributed vectors $U_2 = [0\ 1\ 0\ 0]$, $E_1 = [1\ 1\ 0\ 0]$, and $E_2 = [0\ 1\ 1\ 1]$. Also, $E_3 = [1\ 0\ 1\ 1] \bigoplus [0\ 1\ 0\ 0] = [1\ 1\ 1\ 1]$ is selected such that $\sum_i E_i = U_2$. The second query B_2 is split into subqueries $R_2 = B_2 \bigoplus U_2$, E_1, E_2, and E_3.

Suppose the subqueries from both queries are mixed in the following order:

$$D_1, R_1, E_2, D_3, R_2, E_1, E_3, D_2.$$

Then due to sender anonymity the adversary cannot determine which subquery was sent by which user. Also, each of the subquery vectors is uniformly distributed so they are indistinguishable to the adversary. The adversary can learn which records are requested only by adding the correct subqueries together. As the number of queries and subqueries increases, this task becomes more difficult.

We now consider correctness. The $\mathcal{DB}$ will take each subquery vector and compute the inner product of it and the *ith* column to get the *ith* element of the return vector. Thus each subquery vector will generate a response vector. Each user will receive a response vector corresponding to each of their subqueries. The response vector for the first subquery D_1 is [0 1 0 1]. Similarly the response vectors for R_1, D_3, and D_2 are [0 1 1 0], [1 1 1 0], and [0 1 0 0]. The first user will sum (mod 2) these response vectors to obtain [1 0 0 1], which is the 3rd row of the $\mathcal{DB}$ as desired. Similarly, the 2nd user sums the response vectors it receives to obtain the 4th row of the $\mathcal{DB}$ as desired.

1.2.2 Trapdoor Group cPIR Protocol

We now overview the trapdoor group cPIR protocol. In the Gentry-Ramzan PIR protocol, the group $\mathbb{Z}_n^*$ is used, where $\Phi(n)$ is unknown to the server (an extension of the Φ-hiding assumption is used). Using its knowledge of $\Phi(n)$, the client is able to compute discrete logs in order to obtain the requested blocks from the server. Thus $\Phi(n)$ acts as a trapdoor. Similarly, in our trapdoor group cPIR protocol, the group order, m, is unknown to the server whereas the client uses this value to compute discrete logs to obtain the results of the PIR query.

The natural generalization is a *trapdoor group*. In a trapdoor group, an inversion problem such as computing discrete logarithms may be computed efficiently assuming knowledge of the trapdoor. Without the trapdoor, solving the inversion problem is computationally hard.

The trapdoor group version of our cPIR protocol works as follows:

1. The user selects a large secret number m (the group order), depending on the number of entries in the database, n, and the number of rows requested, r. We view the database as a $\sqrt{n}$ by $\sqrt{n}$ matrix, with n total entries, each with $\log_2(N)$ bits. (A common case would be $N = 2$ in which case the database is viewed as a square bit array).

2. The user randomly selects a secret value $b \in \mathbb{Z}_m^*$, where $\gcd(b, m) = 1$, and $t = \sqrt{n}$ secret coefficients $\{e_i\}$, with the restriction that (a) $e_i < m/(t(N-1))$ for all i (b) the coefficients for unrequested rows are multiples of N^r, and (c) the coefficients for requested rows have the form $N^l + a_l N^r$, for some choice of $l < r$ and a_l. (If $r = 1$, then the user is requesting one row, and all the e_i values are even except for the index j corresponding to this one row; e_j is odd).

3. The user sends $be_i \bmod m,\ 1 \leq i \leq t$ to the database.

4. For each column, the database multiplies each be_i by the corresponding database entry and adds up all of the results (as integers since m is unknown to the database), and sends the total back to the user.

5. The user multiplies each of the results by $b^{-1} \mod m$, and finds the requested values in the base N digits of these quotients.

The security of our trapdoor group PIR protocol follows from the following hardness assumption: let $t = \sqrt{n}$, where the adversary is given only $be_1, \ldots, be_t$ in the group $\mathbb{Z}_m$ (as values less than m), where $e_i < m/(t(N-1)), 1 \leq i \leq t$. Let j be the requested index. e_j is selected randomly from the odd values using the uniform distribution. Then e_i, where $i \neq j$, is selected uniformly randomly from the even values. Then it is computationally infeasible for a probabilistic polynomial time algorithm to compute the group order m. Typical values are $n = 2^{30}$, $N = 2$, and $m = 2^{200}$. We discuss this further below.

We give a reduction that shows that an adversary who can break the security of our trapdoor group cPIR protocol with non-negligible probability, can also determine the value of m with non-negligible probablity, thus breaking the above assumption.

We demonstrate lower bounds on m that ensure that the adversary cannot determine m by treating the PIR request values as random opaque values without structure (given enough of such values, then the adversary could mount a guessing attack on m.) In other words, for the practical sizes of databases that we consider, the number of request elements t is small relative to the size of m so such attacks are not feasible.

We also show a strong security property for our trapdoor group cPIR protocol. In particular, we show that any attack that only considers a small subset of c PIR request elements cannot succeed. The number c will depend on the modulus m. The idea here is that a small set of PIR request elements can also represent a different PIR request with a different base value b'. As c increases, this becomes more unlikely.

Hidden Modular Group Order Hardness Assumption: Here we discuss the hidden modular group order hardness assumption further. Our trapdoor group cPIR protocol has some similarities to the Merkle-Hellman public key cryptosystem [13]. The Merkle-Hellman protocol works as follows: a user creates a private key $(m, b, e_1, \ldots, e_n)$ where m is the modulus, $\gcd(b, m) = 1$, the sequence $e_1, \ldots, e_n$ is superincreasing ($e_i > \sum_{j=1}^{i-1} e_j$), and $\sum_{j=1}^{n} e_j < m$. Then the user computes

$$a_i = be_i \bmod m, \ 1 \leq i \leq n.$$

The public key is the set of elements $a_1, \ldots, a_n$. To encrypt a bit string $x_1, \ldots, x_n$, a user would compute $s = \sum x_i a_i$.

Shamir, building upon the work of others, was able to break the Merkle-Hellman scheme [16]. (Also, [2] describes the other work in this area). In particular, he gave an algorithm which allows decryption of a message without knowledge of the private key. Subsequently, lattice reduction attacks [11] were also used with success to break similar cryptosystems.

The above attacks did not reveal the secret modulus m. Furthermore, Shamir and Zippel [17] had previously shown that knowledge of m was sufficient to cryptanalyze the Merkle-Hellman algorithm. Given that no attacks were found that revealed m gives us additional confidence in our hardness assumption. (The above attacks do suggest that the PIR response may be inadequate to protect confidential server data from network attackers. In the PIR model, the adversary is the database server, and the server already has access to the data. Thus the attacks do not affect PIR security).

1.2.3 General cPIR Protocol

Our general cPIR protocol is the trapdoor group cPIR protocol together with the split and mix construction using anonymity as described above for the anonymity based cPIR protocol.

Remark: The anonymity only version of our cPIR protocol corresponds to the special case of the general cPIR protocol parameters where m is public, $b = 1$, $a_i = 0$, and $r = \log(m)$.

1.3 Performance

Our cPIR protocol is a significantly faster, single database cPIR protocol. We gain performance over many existing cPIR protocols since the database performs additions instead of modular multiplications.

We give some performance results from our prototype implementation later in the paper. For example, our tests indicate that a PIR query and response to obtain one file can be processed in approximately 8 minutes, with likely improvement to around 4-5 minutes by fully leveraging an optimized implementation on a dual core processor given a database of 1000 2 MB files. Our implementation is unoptimized, so further improvements are likely (on the other hand, we have not integrated our algorithm into a database in a manner that allows keyword search queries vs. a request for an index per the usual PIR model).

1.4 Organization

The paper is organized as follows: we present the anonymity based cPIR protocol in Section 2. We then present the trapdoor group cPIR protocol in Section 3. Section 4 discusses the hidden modular group order hardness assumption and security for the trapdoor group cPIR protocol. Here we show that the number of points in a PIR request doesn't leak too much information about m, we give the reduction argument, and cover the strong security property. We cover our implementation's performance in Section 5. We summarize and discuss future work in Section 6. Appendix A analyzes the upper bound on performance based on the methodology from [18].

2 Anonymity Based cPIR Protocol

In this section, we present the anonymity based cPIR protocol. It depends on sender anonymity (see below).

At the end we discuss the general cPIR protocol where the security depends on either the existing hidden modular group order assumption (see below) or sender anonymity. In other words, an adversary must break both assumptions in order to break the security of the cPIR protocol.

Ishai et. al. [9] used the anonymity assumption in combination with a noisy curve reconstruction assumption to show security for cPIR protocols. We apply their "split and mix" technique. To our knowledge, our anonymity based cPIR protocol that we present in this section is the first PIR protocol to rely solely on an anonymity assumption.

Definition 2.1 (Sender Anonymity Assumption) *The PIR system satisfies the sender anonymity [14] assumption when a particular message is not linkable to any sender, and that to a particular sender, no message is linkable. (In other words, the adversary gains no information from the run of the system that would enable linking messages and senders better than from her a-priori knowledge.)*

Definition 2.2 *As above, we view the database as a $\sqrt{n}$ by $\sqrt{n}$ bit matrix, say $\mathcal{DB} = (x_{i,j})$. Let $t = \sqrt{n}$. We select a public group order $m = 2^i$. Let B be the random variable corresponding to the cPIR request that takes on zero values for non-requested indices, and takes on the values $1, 2, \ldots, i$ for the requested indices, respectively. Let $R = B - U \ mod(m)$ where U is uniform over length i bit strings and let $U = \sum_{i=1}^{w} D_i$ where $D_1, \ldots, D_{w-1}$ are uniform over length i bit strings.*

The user sends $w+1$ PIR requests where the elements are generated from $R, D_1, D_2, \ldots D_w$ but $R, D_1, D_2, \ldots D_w$ are permuted into a random sending order (we will see later that they will also be mixed with other queries as well). We will show that R is uniform; therefore $R, D_1, D_2, \ldots D_{w-1}$ are indistinguishable.

$\mathcal{DB}$ processes each received subquery by computing

$$C_{j,k} = \sum_{i=1}^{t} x_{i,j} v_{i,k}, mod(m)$$

given request subquery $v_k = (v_{1,k}, \ldots, v_{t,k})$. All the $C_{j,k}$ values are returned to the respective users. Each returned column vector $C_{j,k}$ is processed by the user; the user sums all of the $C_{j,k}$ vectors corresponding to a query: $\sum_k C_{j,k} = \sum_k \sum_i x_{i,j} v_{i,k} = \sum_i (x_{i,j} \sum_k v_{i,k}) = \delta_{i,j} mod(m)$ where $\delta_{i,j} = x_{i,j}$ if i is a desired index, 0 otherwise. Thus we see that the user obtains the rows of the $\mathcal{DB}$ corresponding to the requested indices.

We will prove the following theorem in a more general setting that includes both the anonymous cPIR protocol with parameters from the preceding definition and the general cPIR protocol. In particular, m is the group order as above, b is the base value ($b = 1$ above), r and t are as described for the trapdoor group protocol in Section 1 and 3. The e have the form $e = aN^r$ for non-requested indices and $e = N^l + aN^r$ for requested indices, where either (1) a is zero as described above for the anonymity based cPIR protocol or (2) a is uniformly randomly generated where $a < \frac{m}{\sqrt{n}(N-1)N^r}$.

Theorem 2.3 *$R = B - U$ is uniform; therefore $R, D_1, D_2, \ldots D_{w-1}$ are indistinguishable.*

Proof: Let $\beta = \lfloor m/t \rfloor$. We have:

$$\alpha_1 = Pr[B = y] = \frac{2^r}{\beta} \left(\frac{t-r}{t} \right)$$

where $y = be$ and $e = aN^r$ for some a.

$$\alpha_2 = Pr[B = y] = \frac{2^r}{\beta} \left(\frac{1}{t} \right)$$

where $y = be$ and $e = 1 + aN^r$ for some a.

$$\ldots$$

$$\alpha_{r+1} = Pr[B = y] = \frac{2^r}{\beta} \left(\frac{1}{t} \right)$$

where $y = be$ and $e = N^{r-1} + aN^r$ for some a.

Then

$$Pr(R = z) = \sum_{\beta/2^r} \alpha_1(1/m) + \sum_{\beta/2^r} \alpha_2(1/m) + \ldots + \sum_{\beta/2^r} \alpha_{r+1}(1/m)$$
$$= (1/m)[(t-r)/t + r(1/t)] = 1/m$$

▮

The database adversary, given a sequence of PIR requests, can determine the B values only by adding the correct PIR request values together. Thus we obtain security via the split and mix technique; the user splits their PIR query into a small number of subqueries and the additional PIR subqueries corresponding to other PIR requests (including from the same user) serve to make it difficult for the Adversary to add the correct requests together to obtain B. In other words, with sufficiently many queries and subqueries, the Adversary's task of picking the right collection of queries to add together becomes computationally infeasible.

2.1 Anonymity Based cPIR Protocol - Analysis

Suppose we select $m = 2$ as in the example in Section 1. The distribution B consists of a single "1" bit for the requested index and "0" bits for the non-requested indices. The advantage in selecting small values of m is that the user will send many smaller queries to retrieve the same amount of information as if sending a few large queries (this advantage holds if the speedup is linear as $\log(m)$ decreases.) But sending more queries is an advantage since w (the number of subqueries that a query is split into) will be smaller. We discuss this further in Section 5.

The distributions $U, D_1, \ldots, D_{w-1}$ are identical independent uniform distributions. Therefore D_w is uniform as well. Any subset of $D_1, \ldots, D_w$ gives no information about B. Similarly R and any subset of $w-1$ elements of $D_1, \ldots, D_w$ gives no information about the missing D_i distribution. The adversary is given a sequence of subquery requests where the subqueries belong to different queries. Thus the adversary must make a random guess at selecting $R, D_1, \ldots, D_w$ as the subqueries belonging to a single query. The computational complexity increases as the number of queries and w increases. Also, as the number of queries per unit time increases, then w can be decreased to obtain the same amount of security. For high query rates, the performance is similar to the trapdoor group cPIR protocol discussed above, since each client can add less noise (we may shrink w).

Let q be the total number of queries sent by all users during the time interval (so $q(w+1)$ subqueries are sent during the interval, since each query is subdivided into $w+1$ subqueries.) The probability of success for an adversary guess is $q/\binom{q(w+1)}{(w+1)}$. Thus for example, suppose $w = 7$, B is a bit vector, the database consists of 1024 2MB files, and the user will retrieve a single file. If $t = \sqrt{n} = 2^{17}$, then the user will need to send 128 queries ($2^7 2^{17} = 2^{24}$) and each query will be subdivided into 8 (since $w = 7$) subqueries. If there are 8 such meta-queries (from multiple users) arriving per unit time, then the probability of a correct adversary guess is approximately

$$q/\binom{q(w+1)}{(w+1)} = 1024/\binom{8196}{8} \approx 2^{-78.7}$$

2.2 General cPIR Protocol

This protocol combines the sender anonymity property as described above and the trapdoor group parameters (see above and Section 1). Since we split the queries, the performance of this protocol is not as good as the performance of the trapdoor cPIR protocol. Since we use the trapdoor group parameters, including the larger (secret) modulus, the performance is less than the anonymity based cPIR protocol. Thus we trade performance for additional security (we now depend on either sender anonymity or the hidden group order assumption).

3 Trapdoor Group cPIR Protocol

We present our protocol in a more general group setting than $\mathbb{Z}_m$.

Let $\mathcal{DB} = (x_{i,j})$ be a database, viewed as a $\sqrt{n} \times \sqrt{n}$ table of elements of $\mathbb{Z}_N$. Let $G = \langle g \rangle$ be a group of order m relatively prime to N, in which discrete log can be efficiently computed.

A user wants to query $\mathcal{DB}$ to learn the values of various entries, all the values from rows $i_0, \ldots, i_{r-1}$ in such a way that $\mathcal{DB}$ cannot determine which entries the user is querying.

The user picks a random $y \in \mathbb{Z}_m^*$, and computes his secret key $b = g^y$. Since y and m are relatively prime, b generates G. He then randomly picks secret exponents $e_1, e_2, \ldots, e_{\sqrt{n}} < \frac{m}{\sqrt{n}(N-1)}$ satisfying the following:

1. If i_l is one of the queried rows, then $e_{i_l} = N^l + a_{i_l} N^r$ for some a_{i_l}.

2. Otherwise, if i is an unqueried row, then $e_i = a_i N^r$ for some a_i.

Note that the restriction on the e_i's requires that each a_i is bounded above by $\frac{m}{\sqrt{n}(N-1)N^r}$. Each a_i is randomly generated from the uniform distribution.

The user then calculates $b_i = b^{e_i}$. He sends $\{b_i\}$ to $\mathcal{DB}$.

For each column j, $\mathcal{DB}$ computes

$$C_j = \prod_{i=1}^{\sqrt{n}} b_i^{x_{i,j}}$$

and sends these values to the user.

The user computes $h_j \equiv \log_b(C_j) \mod m$, and writes the result as

$$(z_{|m|,j} \cdots z_{1,j} z_{0,j})_N$$

in base N. The user concludes that

$$x_{i_0,j} = z_{0,j}$$
$$x_{i_1,j} = z_{1,j}$$
$$\vdots$$
$$x_{i_{r-1},j} = z_{r-1,j}$$

Remark: Let $t = \sqrt{n}$. Note that r is bounded by $(\log(m) - \log(t(N-1)) - \log(t))/(\log(N))$. This fact follows from $a_i < \frac{m}{\sqrt{n}(N-1)N^r}$ which implies $t < \frac{m}{\sqrt{n}(N-1)N^r}$.

3.1 Correctness

Fix a column j. We omit the corresponding subscripts for simplicity. We show that the protocol is correct for this column. Let $t = \sqrt{n}$.

$$\begin{aligned} h = \log_b C = \log_b \left(\prod_{i=1}^{t} b_i^{x_i} \right) &= \log_b \left(\prod_{i=1}^{t} b^{x_i e_i} \right) \\ \equiv \sum_{i=1}^{t} \log_b b^{x_i e_i} &\equiv \sum_{i=1}^{t} x_i e_i \bmod m \end{aligned}$$

Because each $x_i \leq N-1$, and $e_i < \frac{m}{t(N-1)}$, we see that

$$\sum_{i=1}^{t} x_i e_i < \sum_{i=1}^{t} (N-1) \frac{m}{t(N-1)} = m$$

This inequality tells us that the residue modulo m is the number itself, meaning

$$h = \sum_{i=1}^{t} x_i e_i.$$

Now, the user writes $h = (z_m \cdots z_1 z_0)_N$. Consider the base N representation of each e_i. For $0 \leq l < r$, e_{i_l} has a 1 in its (N^l)s place. It follows that $x_{i_l} e_{i_l}$ has a x_{i_l} in its (N^l)s place. Because e_{i_l} is the unique exponent with a non-zero digit in this position, we see that adding the $x_i e_i$'s causes no "carries." From here, we see that the value of z_l is precisely the value of x_{i_l}.

3.2 Trapdoor Group cPIR Protocol Example

Here we give a toy example for the protocol. We consider the database as a 3x3 bit matrix DB (or as consisting of three row elements each with 3 bits).

$$DB = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Suppose the database client desires to obtain the 3rd row of the database. First the client selects the group order, which is not required to be prime. In this case, we select (arbitrarily for the purposes of the example) $\mathbb{Z}_p$ where $p = 83$. The client also selects a random secret element b, where b has a multiplicative inverse. Suppose the client selects $b = 33$. The client then selects random values e_1, e_2, and e_3 where $e_i < p/3$. Also, the client desires to obtain the 3rd row, so e_1 and e_2 are even, whereas e_3 is odd. We suppose that the client sets

$$e_1 = 22,\ e_2 = 12,\ e_3 = 23.$$

The client computes $be_i \mod p$ for $1 \leq i \leq 3$:

$$\begin{aligned} 33(22) &\equiv 62 \mod 83 \\ 33(12) &\equiv 64 \mod 83 \\ 33(23) &\equiv 12 \mod 83 \end{aligned}$$

and sends the values to the server. The database server sets

$$\vec{v} = [62\ 64\ 12\ ]$$

and computes the matrix product (note that since we view the database as a bit matrix, the product only consists of addition operations):

$$v \cdot DB = [62\ 64\ 12\ ] \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

and obtains the vector

$$\vec{v} = [74\ 76\ 126\ ]$$

which is returned to the client. (Recall the server does not know the group order.) The client then computes the values mod(83), and divides by $b = 33$. The Euclidean algorithm gives us that $b^{-1} = 78$.

Thus the client obtains

$$\begin{aligned} 78(74) &\equiv 45 \mod 83 \\ 78(76) &\equiv 35 \mod 83 \\ 78(43) &\equiv 34 \mod 83 \end{aligned}$$

The parity of these elements gives the 3rd row of the database:

$$\begin{aligned} 45 &\mod 2 = 1 \\ 35 &\mod 2 = 1 \\ 34 &\mod 2 = 0 \end{aligned}$$

4 Trapdoor Group cPIR Protocol Security

In this section, we present the hardness assumption, and then bound the leakage of the value of m due to the $\sqrt{n}$ points in the PIR request. We also give the reduction from the hardness assumption to the security of our PIR protocol. Finally, we show the strong security property which ensures the security of our PIR protocol against attacks that only use a small number of elements. We let $negl(k)$ denote a negligible function; $negl(k) < k^{-c}$ for sufficiently large k values given any c.

Here we will define security for our PIR protocol. We first define a PIR instance:

Definition 4.1 *A PIR instance is*

$$I = (m, b, N, r, n, e_1, \ldots, e_c),$$

where $e_i < \frac{m}{t(N-1)}$, $1 \leq i \leq c$. *We assume* $G = \mathbb{Z}_m$, *and* $\gcd(b, m) = 1$. *The other parameters are as described above. Let* k *be the security parameter. The probabilistic polytime sampling algorithm* $Gen(1^k) = (m, b, N, r, n, e_1, \ldots, e_c)$ *maps the security parameter into PIR instances. Given a PIR instance* $I = (m, b, N, r, n, e_1, \ldots, e_c)$, *select an integer representative* v_i, $1 \leq i \leq c$, *where* $v_i \equiv be_i \bmod m$. *Then* $Q(I) = (v_1, \ldots, v_c)$ *is the PIR request corresponding to the instance. Usually, unless stated otherwise, we assume* $N = 2$.

We now define security. Basically, our protocol is secure if a computationally bounded adversary is unable to leak information about the low order bits of the exponents (e_i values).

Definition 4.2 *Let* k *be the security parameter. We define* $LSB(e, r)$ *to be the* r *low order bits of* e. *Let*

$$Pr[LSB(e_i, r) = s] = \epsilon_{i,s}, \forall i, s,$$

where the probability is over e_i *in* I. *We define* $\mathcal{A}(Q(I)) = (i, s)$ *if* $\mathcal{A}$ *predicts* s *for* $LSB(e_i, r)$. *The above PIR protocol is secure if for any PPT adversary* $\mathcal{A}$,

$$Pr[\mathcal{A}(Q(I)) = (i, s)] \leq \epsilon_{i,s} + negl(k), \forall i, s.$$

4.1 Hardness Assumption and Information-Theoretic Bound

We formally present our hardness assumption in this section. The basic idea behind the assumption is that a computationally bounded adversary will be unable to obtain an advantage in computing the group order m.

Definition 4.3 *(Hidden Modular Group Order Assumption) Let* $t = \sqrt{n}$. *Given the PIR request* $Q(I) = (v_1, \ldots, v_t)$, *where* $e_i < m/(t(N-1)), 1 \leq i \leq t$, $e_i = a_i N^r$ *for non-requested indices* i, *and* $e_{i_l} = N^l + a_{i_l} N^r$ *for requested indices* i_l. *The* a_i *values are uniform random elements where* $a_i < \frac{m}{t(N-1)N^r}$. *For any probabilistic polynomial time (PPT) adversary* $\mathcal{A}$, $Pr[\mathcal{A}(Q(I)) = m] \leq negl(k)$.

Remark: An adversary that obtains the low order bits of the PIR exponents, can be used as a subroutine in a reduction that obtains m (see Section 4.2).

As discussed above, our assumption is similar to the assumption that the modulus m cannot be obtained in the Merkle-Hellman public key protocol [13, 17]. The Merkle-Hellman protocol was subsequently cryptanalyzed [16], but we are unaware of any successful attacks aimed at discovering m in Merkle-Hellman.

Given $t = \sqrt{n}$ points that are spread between 0 and m, then we expect, on average, that the largest point is approximately equal to $m - m/t$. Since m/t has approximately $\log(t)$ fewer bits than m, the subtraction leaves the largest $\log(t)$ bits of m intact. Thus we can estimate the largest $\log(t)$ bits of m from the largest point in the PIR request.

We now give a bound for the leakage of the high order bits of m from the PIR request, when the PIR points are treated as unstructured values. This bound does not establish security for our protocol, rather it gives a lower bound for the size of the group order m.

The min-entropy of a random variable X is

$$H_\infty(X) = -\log(max_x Pr[X = x]).$$

Definition 4.4 *Following [5], we define the average min-entropy*

$$\tilde{H}_\infty(X|Y) = -\log\left(E_{y\leftarrow Y}[2^{-H_\infty(X|Y=y)}]\right).$$

The average min-entropy gives the remaining entropy in the random variable X, given knowledge of Y. For us, X will be the group order m, and Y will be the set of PIR request elements. We will bound the remaining uncertainty about m given the PIR request elements and assuming the request elements are treated as unstructured values.

Theorem 4.5 *Given the PIR request elements, which we treat as unstructured values. Let M be the maximum possible value for the group order (which is selected randomly using the uniform distribution). Given the PIR request, the remaining uncertainty about the group order m is at least*

$$-\log\left(2^{-\log(M)+\log(t)+2U(M,t)-1} + \frac{2^{-(\log(M)-2\log(t)-1)}}{1-e^{-2^{-(1/2\log(M)-2\log(t)-U(M,t))}}}\right)$$

where we will set $U(M,t) = \log(M)/8$. In particular, if $\log(t) \leq 20$ and $\log(M) \geq 200$, then the remaining uncertainty is at least $(5/8)\log(M) - 1$ bits.

Proof: We will establish a lower bound on $\tilde{H}_\infty(X|Y)$, where X is the random variable that takes on the group order and Y is the random variable for the set of values in the PIR request. We let $\max\{y\}$ be the largest PIR request value. Let $a(y) = \max\{y\}+1$, and let $b(j) = \lfloor j/\sqrt{n}\rfloor$. Then

$$\begin{aligned}
\tilde{H}_\infty(X|Y) &= -\log(E_{y\leftarrow Y}(max_x Pr[X=x|Y=y])) \\
&= -\log(E_{y\leftarrow Y}(max_x \frac{Pr[X=x \ and \ Y=y]}{Pr(Y=y)})) \\
&= -\log(E_{y\leftarrow Y}\frac{Pr[X=a(y) \ and \ Y=y]}{Pr(Y=y)}) \\
&= -\log(E_{y\leftarrow Y}\frac{Pr[X=a(y) \ and \ Y=y]}{Pr(\bigcup_{i=a(y)}^M [Y=y \ and \ X=i])}) \\
&= -\log(E_{y\leftarrow Y}\frac{Pr[X=a(y) \ and \ Y=y]}{\sum_{i=a(y)}^M Pr[Y=y \ and \ X=i]}) \\
&= -\log\left(E_{y\leftarrow Y}\frac{1/\binom{b(a(y))}{\sqrt{n}}}{\sum_{i=a(y)}^M 1/\binom{b(i)}{\sqrt{n}}}\right) \\
&= -\log\left(E_{y\leftarrow Y}\frac{(b(a(y))-\sqrt{n})!/b(a(y))!}{\sum_{i=a(y)}^M (b(i)-\sqrt{n})!/b(i)!}\right) \\
&= -\log\left(E_{y\leftarrow Y}\frac{1/w_{a(y)}}{\sum_{i=a(y)}^M 1/w_i}\right)
\end{aligned}$$

where $w_i = (b(i)-\sqrt{n})!/(b(i))!$. We set $c(y) = \max\{w_{i+1}/w_i\}$. The maximum occurs at the smallest i such that $b(i)$ increases by 1 over $b(i-1)$. We may write $a(y) = q(y)\sqrt{n}+r(y)$ where $0 \leq r(y) < \sqrt{n}$. Then

$$c(y) = \frac{(b(a(y)+\sqrt{n}-r))!(b(\max\{y\}-r))!}{(b(\max\{y\}+\sqrt{n}-r))!(b(a(y)+r))!}$$

If we select $\bar{y}$ such that $b(a(\bar{y}) + \sqrt{n} - r) = 2^{M/2}/t$, then

$$c(\bar{y}) = \frac{2^{\log(M) - \log t - 1}}{2^{\log(M) - \log t - 1} - 2^{\log(t)}}$$

Continuing from above, we have:

$$\frac{1/w_{a(y)}}{\sum_{i=a(y)}^{M} 1/w_i} =$$

$$\frac{1/w_{a(y)}}{(1/w_{a(y)})(1 + w_{a(y)}/w_{a(y)+1} + \ldots + w_{a(y)}/w_M)}$$

$$\leq \frac{1}{1 + 1/c(y) + 1/c(y)^2 + \ldots + 1/c(y)^{M-a(y)}}$$

$$= \frac{1 - 1/c(y)}{1 - (1/c(y))^{M-a(y)+1}}$$

When $u(y) = M - \max\{y\}$ satisfies $u(y) > 2^{5/8 \log(M)}$, we have

$$c^{u(y)} \geq$$

$$\left(\frac{1}{1 - 2^{\log(t)}/2^{\log(M) - \log(t)}}\right)^{2^{\log(M) - \log(t)} 2^{-(3/8 \log(M) - \log(t))}}$$

$$\approx e^{2^{\log(t)} 2^{-(3/8 \log(M) - \log(t))}}$$

and

$$(1/c)^{u(y)} \leq e^{-2^{\log(t)} 2^{-(3/8 \log(M) - \log(t))}} = e^{-2^{(-3/8 \log(M) + 2\log(t))}}.$$

Thus,

$$\tilde{H}_\infty(X|Y) = -\log(E_{y \leftarrow Y}(max_x Pr[X = x | Y = y]))$$

$$\geq -\log\left(\sum_{u(y) \leq 2^{(5/8)\log(M)}} Pr[Y = y]\left(\frac{1 - 1/c(y)}{1 - (1/c(y))^{u(y)}}\right)\right.$$

$$\left. + \sum_{u(y) > 2^{(5/8)\log(M)}} Pr[Y = y]\left(\frac{1 - 1/c(y)}{1 - (1/c(y))^{u(y)}}\right)\right)$$

$$\geq -\log\Big((1/2) Pr[\max\{y\} \geq M - 2^{(5/8)\log(M)}]$$

$$\left. + \frac{2^{-(\log(M) - 2\log(t) - 1)}}{1 - \left((2^{\log(M) - \log(t)} - 2^{\log(t)})/2^{\log(M) - \log(t)}\right)^{2^{(5/8)\log(M)}}}\right)$$

$$\approx -\log\left(2^{-(3/4)\log(M) + \log(t) - 1} + \frac{2^{-(\log(M) - 2\log(t) - 1)}}{1 - e^{-2^{-((3/8)\log(M) - 2\log(t))}}}\right)$$

The first term in the last inequality follows since $M - \max\{y\} \leq 2^{(5/8)\log(M)}$ which implies that the group order m satisfies $m \geq M - 2^{(5/8)\log(M)}$ (with probability equal to $2^{-(3/8)\log(M)}$) and $\max\{y\} \geq M - 2^{(5/8)\log(M)}$, which occurs with probability equal to $2^{-(3/8)\log(M) + \log(t)}$.

Since $e^x \approxeq 1 + x$ when x is small, we have

$$\frac{2^{-(\log(M)-2\log(t)-1)}}{1 - e^{-2^{-((3/8)\log(M)-2\log(t))}}} = \frac{2(M^{3/8} + t^2)}{M} = \frac{2}{m^{5/8}} + \frac{2t^2}{M}$$

The large term is $2/m^{5/8}$ which is also larger than

$$2^{-(3/4)\log(M)+\log(t)-1}$$

from above. Thus the remaining uncertainty is at least $(5/8)\log(M) - 1$ bits. ▮

Remark: It is possible to optimize the remaining uncertainty in the proof, by solving the resulting cubic equation. The result will be close to $U(M,t) = \log(2M^{1/6}/t^{1/3})$ which we will approximate by $U(M,t) = \log(M)/8$. The approximation will be much simpler and accurate enough for our purposes.

4.2 Proof of Security via Reduction

In this section, we show that a PIR adversary who can break the security of our PIR protocol can also obtain the group order m, thus breaking our security assumption.

We will prove the $r = 1$ case; the proof can be generalized to $r > 1$. We will demonstrate an algorithm B which uses the PIR adversary as an oracle in order to determine m. The basic idea of the reduction is to reduce even exponents (which we learn via the oracle) when they are associated with even remainders which we are given as part of a PIR instance; exponent e and remainder r are related by the equation $be = km + r$. We generate new even exponents and remainders by combining like types (e.g., add an odd exponent and odd remainder with another odd exponent and odd remainder). When we obtain small exponents, we may guess some exponent values and solve for m.

We say that such an oracle is reliable when it determines the parity of the e_i exponents with negligible error, and faulty otherwise. We will assume a reliable oracle initially, and then indicate how the faulty oracle can be used to obtain information about the low order bits of the group order m.

4.2.1 PIR Reduction Algorithm

The reduction algorithm works as follows. Algorithm B is given a PIR instance $Q = (v_1, \ldots, v_t)$ where each v_i satisfies $0 < v_i < m$. B selects an initial set of points from the PIR instance and begins the following algorithm:

B also associates a rational number q_i with each element in the PIR instance; initially, these values are all equal to 1. B submits a PIR instance to the oracle PPT adversary $\mathcal{A}$; $\mathcal{A}$ predicts the parity of some request element exponent.

1. If $\mathcal{A}$ predicts an even exponent e_i, and v_i is even, then B replaces the element v_i with $v_i/2$ to obtain a new PIR instance $\acute{Q}$. B also replaces q_i with $q_i/2$.

2. If one of the other three cases occurs, B will withdraw the element v_i from the PIR instance to obtain a new instance $\acute{Q}$. The other three cases (the odd types) are:

(a) v_i even and e_i is predicted to be odd.

(b) v_i odd and e_i is predicted to be odd.

(c) v_i odd and e_i is predicted to be even.

B maintains the withdrawn elements in three separate lists (according to type), along with their associated q_i values. Periodically B will take like types from each of the withdrawn lists, compute the midpoints of various pairs of these elements and compute midpoints for the associated q_i values, and then add some of the new midpointed elements back into a new PIR instance.

4.2.2 Analysis of Reduction Algorithm

The above algorithm, as long as it continues to iterate and create new elements, will exhibit some elements that have substantially reduced exponents and remainders. Algebraically, the set and the operations above constitute a partial magma (a partial magma is a set with a binary nonassociative operation where the operation is not defined for some pairs of elements). Our goal is to show that the sub-partial magma generated by the PIR instance is not too small.
Remark: We note that the reduction algorithm above is unlikely to obtain very small exponents or remainders. Given the group size m, then the set $\{r : be = km + r \text{ for some exponent } e < K\}$ is unlikely to have elements much smaller than m/K. Given the size restriction on the exponents, we would expect the smallest exponents to be around $2^{\log(m)/2-\log(t)/2}$ and the smallest remainders to be around $2^{\log(m)/2+\log(t)/2}$.

We will show that the reduction algorithm above is likely to obtain exponents and remainders as small as possible, given the preceding remark.

Theorem 4.6 *Given the group order m and c initial elements from a PIR request. The above reduction algorithm will run for at least*

$$T(m,c) = \frac{\log(m) - 4\log(c)}{-\log(v(c)0.85)}$$

iterations, where $v(c)$ is the shrinkage factor for the interval due to the combining operations in the algorithm. (If after $i-1$ reduction rounds the difference between the largest and smallest points is L, then $v(c)L$ is a lower bound on the interval size after i reduction rounds.) Also, $T(m,c)$ iterations gives average reduced remainder values around

$$2^{-T(m,c)/4}2^{\log(m)-1},$$

and corresponding exponent values around

$$2^{-T(m,c)/4}2^{\log(m)-\log(t)-1}.$$

Furthermore, information on the lower order bits of the exponents will be obtained during the algorithm, and this can be used to obtain information on the lower order bits of m.

Sketch of Proof: We will sketch the proof for the reliable oracle case. For the reduction algorithm, without loss of generality, we may assume that the algorithm proceeds by reducing all possible even-even types (cutting them in half), in a given round of operations. Then the algorithm combines all possible like odd types (obtaining their midpoints) in a subsequent round of operations. The algorithm then repeats this procedure. Each pair of halfing and midpoint operations is considered a single iteration.

The algorithm may terminate if no new points can be created. Thus we must show that the number of collisions is bounded relative to the set of new points that are created during each iteration, for up to $T(m, c)$ iterations as described above. We will keep at least $c + 1$ points after each set of midpoint operations, and at least c points after each set of halving operations. (We may keep an additional side set of highly reduced points that are potentially maintained across multiple iterations.) The length of the ith interval is the difference between the largest and smallest points, which we denote by L and S, after the first $i - 1$ iterations. Consider the start of the ith iteration. Then the series

$$\begin{aligned} 3/4 + 3/16(1/2) + 3/64(1/4) + \ldots = \\ 192/256 + 24/256 + 3/256 + 3/256(1/8) + \ldots \\ = 219/256 + 3/1792 < 220/256 = 55/64 \end{aligned}$$

converges to a value between 219/256 and 55/64, and represents the expected reduction factor. Then $(219/256)L$ is a lower bound on the new expected largest point after the halving operations. Also, $(55/64)S$ is a upper bound on the new expected smallest point after the halving operations. Thus $(219/256)L - (55/64)S$ is an expected lower bound on the size of the interval after the halving operations. We approximate this value as $(.85)(L - S)$.

Given expected values for points in the interval after the first $i - 1$ iterations, we compute a lower bound L on the expected value for the largest point, and an upper bound S on the expected value of the smallest point. During the midpoint operations, we create roughly $c^2/2$ new points, and thus we are able to maintain points that are uniformly spread through the new interval. Thus if we have a and b as the smallest and largest points after the first $i - 1$ iterations, then the set of expected points is $a, a + (b - a)/(c - 1), a + (2/(c - 1))(b - a), \ldots, a + ((c - 2)/(c - 1))(b - a), b$. The lower bound for the new largest point will be

$$\begin{aligned} L = (1/3)\frac{a + ((c-2)/(c-1))(b-a) + b}{2} + \\ (2/9)(\frac{a + ((c-3)/(c-1))(b-a) + b}{2}) + \\ (4/27)(\frac{a + ((c-4)/(c-1))(b-a) + b}{2}) \end{aligned}$$

and this last values simplifies to

$$L = (11/18)(a/(c-1)) + ((38c - 71)/54)(b/(c-1))$$

which is a lower bound on the expected largest point. Similarly, we may write the upper bound S as

$$S = \frac{65}{54(c-1)}b + \frac{54c - 119}{54(c-1)}a$$

and the lower bound on the expected size of the interval as:

$$L - S = \frac{38c - 136}{54}\frac{b}{c-1} - \frac{54c - 152}{54}\frac{a}{c-1}$$

When $c = 256$, this simplifies to approximately $(0.696)b - (0.9927)a$. We may overestimate the shrinkage as $(.696)(b - a)$. In this case, $v(c)$ can be replaced with 0.696. (We can obtain a similar value with $c = 32$).

Thus we have lower bounds on the expected shrinkage of the interval size as the algorithm iterates. Next, we obtain bounds on the expected number of collisions. We can upper the collisions due to the halving operations as:

$$(3/16)(c+1)\frac{c+1}{b-a} + (3/64)\frac{(c+1)^2}{(b-a)} + \dots$$

Thus $(c + 1) - (1/4)(c + 1)^2/(b - a)$ is the expected number of new points left after the halving operations.

We also bound the number of collisions due to the midpoint operations. We can overestimate this as $(c^2/3)(c^2/(b - a) = c^4/(3(b - a))$. The collisions due to the midpoint operations are the significant number here; we may ignore the collisions due to the halving operations. Given this collision bound, plus the values above, we obtain the first statement of the theorem.

The second statement follows since, on average, each element or its descendant will be reduced by half every fourth time. When the most reduced points combine with the lesser reduced points, the lesser reduced points will maintain their amount of reduction. Also, more highly reduced points are also created over the 4 iterations.

Remark: The theorem states the average amount of reduction for points, but some points will be more reduced (sampling with replacement which yields a binomial distribution.) We can reduce exponents further by going to the limit in the algorithm described above, and then subtracting remainders to obtain 2^c new lists. One of these lists will have negative remainders and all positive exponents. This list will allow reduction to even smaller exponents. We may guess e_1, e_2, e_3, and e_4, or guess e_1 and derive other exponents from it and the reduction operations. We may write $be_1 = k_1m + r_1$, and $be_2 = k_2m + r_2$, where our reduction algorithm has given us e_1 and e_2 candidates. With high probability, e_1 and e_2 are relatively prime so we can write $y_1e_1 + y_2e_2 = 1$ for integers y_1 and y_2.

$$y_1r_1 + y_2r_2 = b - u_1m$$

We may also obtain another pair e_3 and e_4 and repeat to obtain

$$y_3r_3 + y_4r_4 = b - u_2m$$

Then with high probability, $u_1 \neq u_2$, and thus we can subtract to obtain a value $(u_1 - u_2)m$. We can then guess m from the small number of possibilities, and we can confirm our guess using additional values from the reduction. The probabilistic case can be handled using standard techniques to increase reliability.

We note that an adversary that obtains m can potentially break the security of our PIR protocol, unless much larger sized groups are used. Using m, the adversary can mount a lattice reduction attack [11], using the lattice spanned by the following vectors, where the be_{i_j} are unqueried indices:

$$\begin{aligned}
\mathbf{v}_0 &= (b^{e_{i_1}}, b^{e_{i_2}}, \ldots, b^{e_{i_c}}) \\
\mathbf{v}_1 &= (m, 0, \ldots, 0) \\
\mathbf{v}_2 &= (0, m, \ldots, 0) \\
&\vdots \\
\mathbf{v}_c &= (0, 0, \ldots, m)
\end{aligned}$$

In some cases, a short lattice vector obtained from the reduction algorithm can reveal b and the exponents.

4.3 Strong Security Property

Our security assumption ensures the security of our PIR protocol against a computationally bounded adversary. In this section, we show that any attack which breaks the security assumption and results in a successful attack against our PIR protocol cannot use c or fewer of the PIR instance elements in the attack. In other words, a successful attack must leverage more than a small number of PIR elements, and this minimal number increases with the size of m. Another way to view this is that as m increases relative to c, then a given PIR request can be represented as many different base elements each raised to a different bounded exponent set.

In the following theorem, we assume m is prime, but the theorem can be generalized to composite m.

Theorem 4.7 *Given a PIR instance $Q(I) = (v_1, \ldots, v_c)$ with n, N, m, b, and r as defined above, where $e_i \le m/(N-1)t$, $1 \le i \le c$. Suppose $i_0, i_1, \ldots, i_{r-1}$ are the requested indices. Given indices $j_0, j_1, \ldots, j_{r-1}$ corresponding to an arbitrary PIR request as described above.*

1. *Let $m - 1 > ((N-1)^c + 1)^2(\sqrt{n}(N-1))^c$. With probability*

$$P \ge 1 - \frac{((N-1)^c + 1)^2(\sqrt{n})^c}{(m-1)(N-1)^c}$$

   there exists

$$\frac{m-1}{((N-1)\sqrt{n})^c((N-1)^c + 1)}$$

   other PIR instances $\bar{I}$ with the same n, N, m, and r such that $Q(I) = Q(\bar{I})$. (Here we say I and $\bar{I}$ are compatible).

2. *Let $m - 1 > (((N-1)^c + 1)^2 N^{rc}((N-1)\sqrt{n})^c)/r!$. Then with probability*

$$1 - \frac{N^{rc}(\sqrt{n})^c((N-1)^c + 1)^2}{r!(N-1)^c(m-1)}$$

   there exist

$$\frac{(m-1)r!}{((N-1)\sqrt{n})^c N^{rc}((N-1)^c + 1)}$$

   PIR instances compatible with the PIR request corresponding to $j_0, j_1, \ldots, j_{r-1}$.

We will make use of the following corollary to Chebyshev's inequality [8]:

Lemma 4.8 *Let X_i be independent, identically distributed random variables with mean μ and variance σ^2. Let $S_n = X_1 + \ldots + X_n$. For any $\epsilon > 0$, we have*

$$P\left(\left|\frac{S_n}{n} - \mu\right| \geq \epsilon\right) \leq \frac{\sigma^2}{n\epsilon^2}.$$

We will make use of the max norm: $\|x\|_\infty = \max_i\{|x_i|\}$. We now give the proof of the theorem:

Proof: (1) The vector $\vec{e} = (e_1, \ldots, e_c)$ generates a cyclic submodule L of $\mathbb{Z}_m^c$ over the ring $\mathbb{Z}_m$. We may view te_i as t varies over $\mathbb{Z}_m^*$ as a random permutation of $\mathbb{Z}_m^*$. Thus $t\vec{e}$ is a random permutation over the cross product of the individual coordinate subsets. Thus we may assume that

$$\begin{aligned} Pr[\|x\|_\infty < m/((N-1)\sqrt{n}) \textit{ given } x \in L] = Pr[\|x\|_\infty \\ < m/((N-1)\sqrt{n})] \end{aligned}$$

Also,

$$Pr[\|x\|_\infty < m/((N-1)\sqrt{n})] = \frac{1}{(N-1)\sqrt{n})^c}$$

Let $\epsilon = 1/(((N-1)^c + 1)(\sqrt{n})^c)$. By the lemma above, we have

$$Pr[|S_{m-1}/(m-1) - \mu| \geq \epsilon] < \frac{\sigma^2}{(m-1)\epsilon^2}$$

where $X_i = 1$ if the ith vector in L satisfies

$$\|x\|_\infty < m/((N-1)\sqrt{n}),$$

$S_i = X_1 + X_2 + \ldots + X_i$, with μ as the mean of X_i, and σ^2 is the variance. $\sigma^2 = \mu(1-\mu) < \mu$ since the X_i are Bernoulli random variables.

$\mu = Pr[X_1 = 1] = (1/((N-1)\sqrt{n}))^c$. Thus

$$\frac{\sigma^2}{(m-1)\epsilon^2} < \frac{\mu}{(m-1)\epsilon^2} = \frac{((N-1)^c + 1)^2(\sqrt{n})^c}{(m-1)(N-1)^c}$$

Thus with probability P where

$$P \geq 1 - \frac{((N-1)^c + 1)^2(\sqrt{n})^c}{(m-1)(N-1)^c}$$

we have

$$\begin{aligned} \frac{S_{m-1}}{m-1} \geq \mu - \epsilon = \frac{1}{((N-1)\sqrt{n})^c} - \frac{1}{((N-1)^c + 1)(\sqrt{n})^c} = \\ \frac{1}{((N-1)^c + 1)(\sqrt{n}(N-1))^c} \end{aligned}$$

It remains to show that the S_{m-1} short vectors in L correspond to other PIR instances. For each such vector s, we have $t_s e_i \equiv s_i \mod m$, $1 \leq i \leq c$, for some t_s where $1 \leq t_s \leq m-1$. Let

$f \equiv b(t_s)^{-1} \mod (m)$. Then $fs_i \equiv be_i \mod (m)$, $1 \leq i \leq c$. Thus $f, s_1, \ldots, s_c$ are part of a PIR instance $\bar{I}$, such that $Q(I) = Q(\bar{I})$.

(2) Now we prove claim (2) above. Let $Y_i = 1$, if the ith vector in L is short and compatible with $j_0, \ldots, j_{r-1}$, and let $Y_i = 0$ otherwise, where compatible is defined as a vector resulting in the same requested PIR indices. Let $T_i = Y_1 + Y_2 + \ldots + Y_i$. Let

$$\epsilon_2 = \frac{r!}{N^{rc}(\sqrt{n})^c((N-1)^c + 1)}.$$

From the lemma, we have

$$Pr[|T_{m-1}/(m-1) - \mu| \geq \epsilon_2] < \frac{\sigma^2}{(m-1)\epsilon_2^2}$$

$\mu = Pr[Y_1 = 1] = (1/((N-1)\sqrt{n}))^c(r!/N^{rc})$. Thus

$$\frac{\sigma^2}{(m-1)\epsilon_2^2} < \frac{N_{rc}(\sqrt{n})^c((N-1)^c+1)^2}{r!(N-1)^c(m-1)}$$

Thus the number of compatible PIR instances is at least

$$\frac{(m-1)r!}{((N-1)(\sqrt{n}))^c N^{rc}((N-1)^c + 1)}$$

∎

Table 1 gives values of m that ensure that attacks will not be successful unless they make use of more than c PIR request values (derived from Theorem 4.7(2), $\log(m) = 16c + 42 > 16c + 2$ so $m - 1$ satisfies the hypothesis of Theorem 4.7(2)).

c	$\log(m) = 16c + 42$
8	170
16	298
24	426
32	554

Table 1: $N = 2$, $r = 1$, $n = 2^{30}$; each row's $\log(m)$ value ensures that attacks must use more than c elements (c in the same row), with probability $> 1 - 2^{-40}$

5 Implementation Performance

In this section we present performance results. We have an initial, unoptimized, prototype implementation, and we include performance numbers in order to give a lower bound on performance.

Length of m	200	300	400	500
Time for $n = 2^{24}$	0.673	0.71	0.946	1.08
Time for $n = 2^{26}$	1.746	2.12	2.49	2.765
Time for $n = 2^{28}$	6.29	6.638	7.684	8.73
Time for $n = 2^{30}$	24.08	25.1	26.47	31.46
Time for $n = 2^{32}$	98.3	114.65	122.04	131.92
Time for $n = 2^{34}$	412.2	461.2	489.8	510.6

Table 2: Execution times (sec.) for our protocol based on unoptimized prototype implementation

5.1 Prototype Implementation

To test the performance of our trapdoor group protocol, we ran it with several parameter sets and timed each part of the executions. The protocol was run on an Intel Core2 Duo T7500, with 2.2 GHz clock speed, 2 GB of RAM, and 4 MB of L2 cache. In each test, we used $N = 2$ and $r = 1$. All times are given in seconds (see Table 2). Increasing r has minimal impact on performance. Increasing the size of m only results in small decreases in performance; the reason appears to be the parallelism that our system uses in adding large integers. Based on these tests, we estimate that the anonymity based protocol is anywhere between 20-50% as fast as the trapdoor group protocol (since larger numbers of queries gives smaller w and less impact), but a full understanding requires testing with large numbers of users, queries, and a real-time anonymity system.

Using the example from Aguilar-Melchor and Gaborit [1], we roughly compare the performance of the trapdoor group protocol and anonymity based cPIR protocols against the Aguilar-Melchor and Gaborit, Gentry and Ramzan, and Lipmaa protocols. Here the user retrieves a 3 MB file from a database with 1000 files.

The database size is approximately $2^{34.55}$ bits, which requires approximately 1.5 times as long as the 2^{34} case entry in Table 2. For the trapdoor group case, using $\log(m) = 400$, we obtain roughly 12 minutes for client and server processing times. The network speeds from the example are 1 Mbit upload and 20 Mbit download speeds. We add 60 seconds for initial network transfer time to obtain 13 minutes for the trapdoor group protocol. The CPU clockrate in our experiments is 2.2 Ghz vs. the 4 GHz 64 bit dual core machine from [1]. Thus the times for the trapdoor group and the Aguilar-Melchor and Gaborit protocols are comparable.

The anonymity based protocol becomes more efficient than existing cPIR protocols for databases with many requests per unit time interval. Given the results from Table 2 (based on tests on the 2.2 Ghz machine), we roughly estimate that $m = 2^{160}$ requires 450 seconds per query for the 3 MB file retrieval example above. We use 160 since $160 \times 2^{17.27} \approx 3MB$. (We recall we assume a database with 1000 3 MB files.) If each user subdivides their query into 10 subqueries, then 4500 seconds, or 75 minutes is needed to serve the user request. The security obtained is:

$$q/\binom{q(w+1)}{(w+1)} = 200/\binom{2000}{10} \approx 2^{-80.2}$$

where we assume 200 user queries per time interval (each user query is subdivided into 10 subqueries). In this case the time interval is 75 minutes. The total processing power required to handle the 200 user requests must be at least 200 times as large as the 2.2 Ghz processing power. The 75 minute time compares favorably to the existing cPIR protocols (roughly 14 times as fast as the

Gentry Ramzan protocol); we can reduce this time further if we assume a higher request rate per unit time interval.

cPIR Protocol	**Query Plus Download Time**
Limpaa	33h
Gentry & Ramzan	17h
Aguilar-Melchor & Gaborit	10.32m
Trapdoor Group	13m
Anonymity Based	75m

Table 3: Performance Comparison for PIR Schemes (first 3 entries from [1]); last two entries based on 2.2 Ghz machine tests vs. faster machine for first 3 entries

6 Summary and Future Work

In this paper, we presented new cPIR protocols that have much improved computational complexity. When the database is viewed as a bit sequence, only modular addition operations are performed by the database server. We can view our protocols as a middle ground between the trivial protocol (fastest possible computational complexity and slowest possible communication complexity) and protocols such as Gentry-Ramzan [6] (fast communication complexity but slower computational complexity). This middle ground enjoys a substantially better overall performance. To our knowledge, our anonymity based protocol is the first cPIR protocol that depends only on sender anonymity.

For our trapdoor group cPIR protocol, we assume that given only the elements $be_1, \ldots, be_t$ in the group $\mathbb{Z}_m$, where $e_i < m/t$, the low order r bits of e_i encode the requested indices, and t is small, it is hard to compute the group order m. Given this assumption, we have proved that our protocol is secure against a polynomial-time bounded adversary. Our prototype implementation shows that our protocol's performance compares favorably with existing cPIR protocols.

Our future work is aimed at formalizing trapdoor groups. Our initial thoughts are that a trapdoor group consists of a group G, an image group $\bar{G}$, an invertible map $\alpha : G \rightarrow \bar{G}$, (possibly α is the identity), the trapdoor secret s, and an inversion problem P consisting of a set of elements in G. We also plan to protect the privacy of the database.

7 Acknowledgments

We thank Bill Gasarch for helpful comments.

References

[1] C. Aguilar-Melchor and P. Gaborit. A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol. http://eprint.iacr.org/2007/446.pdf

[2] E. Brickell, A. Odlyzko. Cryptanalysis: A Survey of Recent Results. *Contemporary Cryptology,* G. J. Simmons (ed.), IEEE Press (1991), pp. 501-540.

[3] C. Cachin, S. Micali, and M. Stadler. Private Information Retrieval with Polylogarithmic Communication. In *Proceedings of Eurocrypt*, pages 402-414. Springer-Verlag, 1999.

[4] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of the 36th Annual IEEE Symp. on Foundations of Computer Science*, pp. 41-51, 1995. Journal version: J. of the ACM 45:965-981, 1998.

[5] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy Extractors: How to generate strong keys from biometrics and other noisy data. In *Proceedings of Eurocrypt*, pages 523-540, Springer-Verlag, 2004.

[6] C. Gentry and Z. Ramzan. Single Database Private Information Retrieval with Constant Communication Rate. ICALP 2005, LNCS 3580.

[7] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *STOC 98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 151-160, New York, NY, USA, 1998. ACM Press.

[8] P. Hoel, S. Port, and C. Stone. *Introduction to Probability Theory.* Houghton Miflin Company, 1971.

[9] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography from Anonymity. In *Proceedings FOCS* 2006.

[10] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings of FOCS.* IEEE Computer Society, 1997.

[11] A. K. Lenstra, H.W Lenstra Jr., and L. Lovasz. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen 261,* 1982, pp. 515-534,

[12] H. Lipmaa. An oblivious transfer protocol with log-squared communication. Cryptology ePrint Archive, 2004.

[13] R. C. Merkle and M. E. Hellman. Hiding Information and Signatures in Trapdoor Knapsacks. *IEEE Trans. Information Theory,* vol. 24, 1978, pp. 525-530.

[14] A. Pfitzmann and M. Kohntopp. Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology. *Anonymity 2000,* LNCS 2009, pp. 1-9, 2001.

[15] J. P. Stern. A New Efficient All-Or-Nothing Disclosure of Secrets Protocol. In K. Ohta, D. Pei, editors: ASIACRYPT. Volume 1514 of Lecture Notes in Computer Science, Springer (1998) pp. 357-371.

[16] A. Shamir. A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem. *IEEE Trans. Information Theory,* vol. IT-30, 1984, pp. 699-704.

[17] A. Shamir and R. Zippel. On the Security of the Merkle-Hellman Cryptographic Scheme. *IEEE Trans. Information Theory,* vol. 26, no.3, May 1980, pp. 339-340.

[18] R. Sion and B. Carbunar. On the Computational Practicality of Private Information Retrieval. In *Network and Distributed System Security Symposium NDSS 2007*

A Complexity and Performance Modeling

In this section, we use the (Sion-Carbunar) methodology developed in [18] to analyze the performance of our PIR protocol. Their methodology makes assumptions in favor of increased performance. Thus we regard the resulting performance numbers as an upper bound on the performance. The other point here is that their methodology when applied to existing cPIR protocols led to their conclusions regarding very slow cPIR performance. Their methodology applied here results in a very fast predicted performance. We have an initial, unoptimized, prototype implementation, and those performance numbers are discussed above in Section 5. We consider those results to be a lower bound on performance.

A.1 Complexity

There are two complexity issues here — communication and computation.

Let $\mathcal{DB}$ be a database with n entries, each in $\mathbb{Z}_N$. This, as stated earlier, is viewed as a $\sqrt{n} \times \sqrt{n}$ table of values. Because we are interested in a fast protocol, we give the complexity when $G = (\mathbb{Z}_m, +)$, where discrete log is efficiently computable (as required), and where computations are very fast. Similar asymptotic bounds hold for other groups.

Communication — The user sends $\sqrt{n}$ group elements to $\mathcal{DB}$, each of length $\log_2 m$, and $\mathcal{DB}$ sends as many group elements to the user. These add up to a total of $2\sqrt{n}\log_2 m$ bits. This complexity can be improved to $O(n^\epsilon)$ by moving from a 2-dimensional database to a d-dimensional one. However, because our goal is a fast protocol, we note that increasing the dimension also increases $\mathcal{DB}$'s computation time, which is the bottleneck [18].

Computation — The user picks $\sqrt{n}$ random exponents, and computes as many group exponentiations. In $(\mathbb{Z}_m, +)$, an exponentiation is merely multiplication modulo m. $\mathcal{DB}$ performs $\sqrt{n}$ group operations (additions modulo m or integer additions) for each of $\sqrt{n}$ columns, totalling n additions. When the entries of $\mathcal{DB}$ are bits, this is all of the computation done by $\mathcal{DB}$. When they are elements of $\mathbb{Z}_N$, then it also must perform n multiplications of group element by $\mathcal{DB}$ entry. After receiving the output from $\mathcal{DB}$, the user now processes only a small number of group elements returned. He performs one discrete log (division modulo m) on each, converts to base N and extracts the queried data, which takes about $\log_N m$ time — negligible relative to $\sqrt{n}$. $\mathcal{DB}$ is destined to be stuck with $\Omega(n)$ computation in any PIR scheme, so we cannot hope for any better.

A.2 Performance analysis

The goal of this protocol is not to push down the bounds on complexity of cPIR, which has already been done in previous research. Instead, the goal is a cPIR protocol which has improved performance on some common database sizes and which is faster than the trivial PIR protocol — transferring the entire database in question.

In Sion and Carbunar [18], it was claimed that the current best cPIR protocols are orders of magnitude slower than sending an entire database to the user (the trivial protocol). Their results explained that, because of the large cost of $O(n)$ modular multiplications in $\mathbb{Z}_m^*$, and because data transfer speeds are close enough to processor speeds, it is unlikely that any cPIR protocol could perform more quickly than sending the database.

For end user links, our PIR protocol is substantially faster than the trivial protocol (which itself is at least an order of magnitude faster than existing cPIR protocols). We use the numbers and formulas put together in [18] to upper bound the performance of our protocol, and compare it to that of the trivial protocol.

A.3 Formulas and estimates

First, we define the following variables for any computer:

- B — bandwidth of network connection, in bits per second
- M — the average time required for the CPU to perform a single instruction, taken from the MIPS rating (Millions of Instructions Per Second)
- d — the bit-size of a digit in the CPU's architecture
- $t_{add}(m)$ — time needed for the CPU to perform an addition in $\mathbb{Z}_m$
- $t_{mul}(m, N)$ — time needed for the CPU to perform a multiplication of a number in $\mathbb{Z}_m$ by a number less than N, modulo m
- t_d — time needed by the CPU to perform a single digit multiplication operation
- t_a — time needed by CPU to perform a single digit addition operation (most modern processors have multiple ALU's and can perform more than one addition per cycle).
- t_t — time needed to transfer a bit over the network

From the verified assumption in [18], we estimate that $t_d \approx \frac{1}{M}$, $t_a \approx \frac{1}{2M}$, and $d = 5$. Note that $|m|$ denotes the bit-length of m. The analysis in [18] also shows that

$$t_{add}(m) \approx \frac{|m|}{d} \times t_a.$$

Thus

$$t_{add}(m) \approx \frac{|m|}{2M \times d}$$

and

$$t_{mul}(m, N) \approx \frac{|m| \times |N|}{M \times d^2}.$$

Assuming throughput actually matches bandwidth — and therefore overestimating transfer speeds and favoring the trivial protocol — we estimate that $t_t \approx \frac{1}{B}$.

A.4 Analysis

Let $\mathcal{DB}$ be a database with n entries, each a member of $\mathbb{Z}_2$ (thus $N = 2$). Let k be the security parameter, and m be the modulus for the protocol.

Let T_{pir}, T_{trans} be the times needed to complete the cPIR protocol, and to transfer the entire database over a network, respectively.

Trivally, we see that $T_{trans} = n \cdot t_t \cdot \log_2 N$, since we must transfer n entries, each with $\log_2 N$ bits. From Section A, we get

$$T_{pir} = n \cdot t_{add}(m) + n \cdot t_{mul}(m, N) + \sqrt{n} \cdot t_{mul}(m, m) + 2\sqrt{n} \cdot t_t \cdot \log_2 m$$

The first two terms of this formula are the additions and multiplications performed by $\mathcal{DB}$, the third is multiplications by the user, and the final term is communication between the two parties. The second term will be zero, since we are assuming $N = 2$; the database performs no multiplications.

In [18], Sion and Carbunar distinguished three classes of networks: end-user internet connections, high-end intersite connections, and Ethernet LAN connections. In 2006, he estimated that the average bandwidths for these were 6 Mbps, 1.5 Gbps, and 10 Gbps respectively. Additionally, he estimated the average MIPS rating as 25000.

Using these numbers and formulas, we see the following execution times, based on databases of various sizes, with entries viewed as bits. Table 4 gives the estimated times, T_{pir}, for our PIR protocol when using the Sion-Carbunar performance model as discussed above. We assume $N = 2$, and $r = 128$. Thus we are viewing the database as a $\sqrt{n} \times \sqrt{n}$ bit array and we retrieve $32\sqrt{n}$ bits per each PIR request (we retrieve r rows during each request.) For the results in the table, we set $\log(m) = 200$. Column 3 of the table gives the performance times for our protocol. The latter columns give the time required for the trivial protocol, over various sized networks. For T_{pir}, we assume the end-user bandwidth for all data transfers.

DB Size	returned bits: $32\sqrt{n}$	T_{pir}	End-user	Inter-site	LAN
2 MB	2^{17} bits	0.2867 s	2.796 s	0.01118 s	0.0016778 s
128 MB	2^{20} bits	3.045 s	357.914 s	1.4316 s	0.21474 s
0.5 GB	2^{21} bits	7.8 s	23.86 mins	5.7264 s	0.8589 s
2 GB	2^{22} bits	22.47 s	95.44 mins	22.9 s	3.4356 s
8 GB	2^{23} bits	72.47 s	6.363 hours	91.6 s	13.74 s

Table 4: Execution times for our protocol and full database transfer, for various database sizes

From this table, we see that execution time of our protocol is much faster than database transfer for end-user connections, regardless of how the database is viewed. For the high-end intersite networks, our PIR protocol is slightly faster than the trivial protocol for the larger DB sizes. Summarizing, for the slower connections, our PIR protocol is faster than the trivial protocol, but for faster connections, the trivial protocol is faster. In [18], the authors show that the trivial protocol is significantly faster, on all types of networks, than all previous cPIR protocols.