A Fast Protocol for Computationally Private Information Retrieval

Andy Parrish and Jonathan Trostle
Johns Hopkins University Applied Physics Laboratory

August 20, 2007

Abstract

We present a new private information retrieval (PIR) protocol. The protocol is based on a single private, non-shared key cryptosystem; the security of this cryptosystem is based on a new hardness (secret base) assumption. We prove security for the secret base assumption in an extended generic group model. We also show parameters that ensure security against a lattice-based attack. We measure performance using the methodology in [10]; our scheme is orders of magnitude faster than any existing scheme and faster than the trivial protocol for the home user scenario.

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. 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.

1.1 Prior work

Chor [3] introduced information-theoretic PIR where the database is replicated and the replicas are not allowed to communicate; they show that single database information-theoretic PIR does not exist. Kushilevitz and Ostrovsky [6] 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. [5] showed how to protect database privacy as well. The work in [1, 2, 7, 4] demonstrated schemes with polylogarithmic communication complexity. On the other hand, there has been little work aimed at improving communication costs on the server (possibly since the server cannot have better than $\Omega(n)$ computational complexity). Sion [10] presented performance results claiming that all of the existing schemes are slower than the trivial protocol.

1.2 Our results

Various cPIR protocols have been presented, all relying on known public key cryptosystems. The entire field of cPIR has been criticized for being too slow *in principle*, not just in practice [10]. We aim to counter such claims by providing a significantly faster cPIR protocol. We do this by allowing several speedups:

- Users may query several database entries simultaneously.
- Database entries may be viewed as numbers rather than bits.
- The protocol is secure in $(\mathbb{Z}_m, +)$, allowing fast computation.

Although existing cPIR schemes are based on public key cryptosystems, the public key is not used as such. In the existing schemes, the user encrypts with their public key and decrypts the returned data with their private key. Thus, the public key does not need to be known by outside parties. In our scheme, we use a single key that is kept private by the user; it is not shared with any other party.

A simplified, high level description of the idea behind our protocol is as follows: the user picks some random secret exponents in a bounded range,

and picks a secret base number b, raising b to each of the exponents. These values are sent to the database which then takes the product of the values corresponding to the "1" bits in the database. The product is returned to the user who then computes the discrete log (the group is selected as one in which discrete logs can be efficiently computed). The sum of the exponents corresponding to the "1" bits can be calculated from the discrete log since the exponents are in a bounded range. Whether certain factors are present in the sum then detemines the values of queried indices. If the group is $\mathbb{Z}_m$, then the product corresponds to addition operations which gives very fast computational performance on the server.

While the protocol will work in any group where discrete log can be computed, we briefly describe it here in $\mathbb{Z}_m$.

1. The user selects a large number m, depending on the number of entries in the database, n, the number of entries requested, r, and the maximum possible value for entries in the database, N.

2. The user randomly selects a secret value $b \in \mathbb{Z}_m^*$, and $\sqrt{n}$ secret coefficients $\{e_i\}$, with the restriction that (a) $\sum e_i < m$, (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.

3. The user sends $\{e_i b\}$ to the database.

4. For each column, the database multiplies each $e_i b$ by the corresponding database entry and adds up all of the results (both modulo m), and sends the total back to the user.

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

How large should the "large" number m be? For a given parameters m, n, r, N as above, it takes $\sqrt{\frac{m}{\sqrt{n}(N-1)N^r}}$ operations to break security. Thus, for a security parameter of k, we should have

$$m \approx 2^{2k}\sqrt{n}(N-1)N^r$$

However, to remain secure from lattice attacks, we need to increase m further. Say we have a database of 2^{40} 32-bit integers, a security parameter of 80, and

would like to request one entry. This would require m to be around 1293 bits. To query more entries simultaneously, we need to increase m further – see Table 1.

The paper is organized as follows: we present our protocol and establish its correctness in Section 2. We give a definition for security and show a possible attack in Section 3. In Section 4, we present an extension of the generic group model, and we show that our protocol requires $\sqrt{p}$ operations in this model to obtain the secret base b, where p is the order of the group. We describe how our protocol reduces to the problem of determining the secret base. We also demonstrate a lattice attack against our protocol and give lower bounds on the values for the group order in order to prevent this attack. The basic idea here is that by increasing the group order, we increase the number of vectors that can be returned by the LLL (lattice basis reduction) algorithm. Therefore the adversary has a small probability of obtaining a vector with the secret exponent values. In Section 5, we discuss communication and computational complexity. We cover performance in Section 6. We conclude in Section 7.

2 Our 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 from rows $i_0, \ldots, i_{r-1}$ in such a way that neither an eavesdropper nor $\mathcal{DB}$ itself can 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 $\frac{m}{\sqrt{n}(N-1)N^r}$.

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

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

2.1 Correctness

Fix a column j. We omit the corresponding subscripts for simplicity. We show that the protocol is correct for this column.

$$h = \log_b C = \log_b \left(\prod_{i=1}^{k} b_i^{x_i} \right) = \log_b \left(\prod_{i=1}^{k} b^{x_i e_i} \right) \equiv \sum_{i=1}^{k} \log_b b^{x_i e_i} \equiv \sum_{i=1}^{k} x_i e_i \bmod n$$

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

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

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

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

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 Security

Fix $r, n, N \in \mathbb{N}$ — the parameters for a user to query information from r rows of a $\sqrt{n} \times \sqrt{n}$ database whose entries lie in $\mathbb{Z}_N$. Fix a group $G = \langle g \rangle$ of order m, where $(m, N) = 1$.

We define security as a game. An adversary provides two sets of r indices each: $i_0, \ldots, i_{r-1} \leq \sqrt{n}$ and $j_0, \ldots, j_{r-1} \leq \sqrt{n}$.

The game picks a random $y \in \mathbb{Z}_m^*$ and calculates its random generator $b = g^y$. The game then picks random exponents $e_1, \ldots, e_{\sqrt{n}} < \frac{m}{\sqrt{n}(N-1)}$, each a multiple of N^r, and computes $b_i = b^{e_i}$, placing the results into an array Q. Now, for each i_l sent by the adversary, the game picks a random exponent $f_l < \frac{m}{\sqrt{n}(N-1)}$ of the form $N^l + a_l N^r$ for some a_l, and computes $c_l = b^{f_l}$. An array Q_0 is constructed which differs from Q only in the chosen indices $i_0, \ldots, i_{r-1}$, where the i_l^{th} entry is replaced by c_l. Another array Q_1 is constructed in the same way, using indices $j_0, \ldots, j_{r-1}$. A random bit $z \in \{0, 1\}$ is chosen, and Q_z is sent to the adversary.

The adversary wins the game if it outputs the value of z, correctly distinguishing between the two arrays.

The cPIR protocol is secure with the above parameters r, n, N, G when a probabilistic polynomial time adversary can win this game with probability no more than $\frac{1}{2} + \epsilon$, where ϵ is a negligible function of $|G|$ for any fixed r, n, N. Probability is taken over the random choices made by the game as well as the adversary.

To see why we require $(m, N) = 1$, suppose we allowed m and N to share a nontrivial common divisor d. From here, we see that security can be easily broken by calculating $(b_i)^{\frac{m}{d}}$. Let i be an unqueried row, and $j = i_0$ be the exponent for the first queried row. Then

$$e_i = a_i N^r = a_i' d \text{ and } e_j = 1 + a_j N^r = 1 + a_j' d$$

for some a_i, a_i', a_j, a_j'. Then we see that

$$(b_i)^{\frac{m}{d}} = (b^{a_i' d})^{\frac{m}{d}} = (b^m)^{a_i'} = 1$$

$$(b_j)^{\frac{m}{d}} = (b^{1 + a_j' d})^{\frac{m}{d}} = (b^{\frac{m}{d}})(b^m)^{a_j'} = b^{\frac{m}{d}} \neq 1$$

This immediately identifies one of the queried bits, so the adversary can win the security game.

3.1 A potential attack

Although the security of this protocol explicitly depends on determining whether an exponent is a multiple of N^r, a natural attack is to attempt to find the secret base, b. An adversary who knows b can compute $\log_b b_i$ to reveal each of the exponents, thus identifying the queried indices. One approach to finding b is to pick a random $x \in \mathbb{Z}_m$ and any index i, and compute $c = (b_i)^x$. The special form of the e_i's allows us to easily rule out most any $c \neq b$ by taking discrete logs with respect to c. Of course, each attempt has a probability of $\frac{1}{m}$ for finding b in this manner. We show how to increase these chances to approximately $\frac{\sqrt{n}(N-1)N^r}{m}$.

As before, the adversary picks a random $x \in \mathbb{Z}_m$ and any index i, and compute $(b_i)^x$. He hopes to find $b^{s^{-1}}$ for some $s \in \{1, 2, \ldots, \sqrt{n}(N-1)N^r\}$.

Suppose he succeeds, and finds $b^{s^{-1}}$ for some unknown s. By taking the discrete log of $b_i = b^{e_i}$ with respect to $b^{s^{-1}}$, the adversary finds numbers of the form $\frac{e_i}{s^{-1}} \equiv e_i s \mod m$. He then calculates $d_i \equiv (e_i s)N^{-r} \mod m$. Assuming i is an unqueried index, we can write $e_i = a_i N^r$, and thus $d_i \equiv a_i s$. If i is a queried index, then d_i has an effectively random value.

Note that, because $e_i = a_i N^r < \frac{m}{\sqrt{n}(N-1)}$, we see that $a_i < \frac{m}{\sqrt{n}(N-1)N^r}$. Together with the bound on s, we get $a_i s < m$, meaning that d_i and $a_i s$ are equal, rather than just equivalent modulo m. The adversary can now take several of these d_i's and calculate their GCD. If the indices he uses for this are unqueried, then we will get

$$\gcd(d_1, d_2, \ldots) = \gcd(a_1 s, a_2 s, \ldots) = s \cdot \gcd(a_1, a_2, \ldots)$$

With high probability, the remaining GCD will be 1 meaning the adversary learns s. From here, he computes $(b^{s^{-1}})^s = b$, and learns every exponent.

Note that, in the game definition of security, the adversary can easily pick unqueried indices to compute this GCD. In practice, only a small portion of the rows will be queried, so an adversary who discovers $b^{s^{-1}}$ can find s — and b — with high probability.

This attack leads to a good heuristic for picking m. There are $\sqrt{n}(N-1)N^r$ possibilities for s. Not all of these will necessarily be invertible modulo m, but we ignore this to err on the side of caution.

To reduce the probability of this attack succeeding, we want $\frac{m}{\sqrt{n}(N-1)N^r}$ — the average number of attempts before successfully finding a $b^{s^{-1}}$ — to be prohibitively large.

4 An extended generic group model

We now prove the security of our protocol in the case of a prime order group. To emphasize this, we write $m = p$. We prove this in two steps. First we prove that it is difficult to discover the secret base. We then show that any algorithm to break the security of the protocol can be used to find the base.

In the first step, we use a generic group model (GGM) with access to a generic discrete log oracle. The oracle is generic in the sense that it returns random strings corresponding to the answer, capturing the notion that taking the discrete log of an group elements with bounded random exponents will give random numbers. The adversary is allowed to do ordinary arithmetic operations on the exponents and use them to exponentiate group elements.

4.1 Modified Protocol

Here we present a strictly stronger variation of the original protocol.

In the original protocol, the user computes $b_i = b^{e_i}$, and sends these values to $\mathcal{DB}$. In this variation, the user picks a random secret $t \in \mathbb{Z}_n$, and computes $b_i = b^{t+e_i}$ and sends these instead. For each row j, $\mathcal{DB}$ computes C_j just as before, but also returns

$$s_j = \sum_{i=1}^{k} x_{i,j}$$

When processing a row, the user now computes

$$\log_b C_j = \log_b \left(\prod_{i=1}^{k} (b_i)^{x_{i,j}} \right) = \log_b \left(\prod_{i=1}^{k} b^{x_{i,j}(e_i+t)} \right) = \sum_{i=1}^{k} x_{i,j}(e_i+t) = h_j + s_j t$$

Here, h_j is defined as in Section 2. From this point, the user knows s_j and t, so he can find h_j and learn the queried values from $\mathcal{DB}$.

Note that this variation eliminates the possibility of the GCD attack given in Section 3.1. Our security analysis will be carried out on this more general protocol (the original protocol corresponds to the $t = 0$ case).

4.2 Security in the Extended Generic Group Model

Theorem 4.1 *Let p be a large prime number, and G be a group of order p with random non-identity element b. Let n, N, r be as in the protocol, and $B = \lfloor \frac{p}{\sqrt{n}(N-1)N^r} \rfloor$ be the upper bound on the values for the a_i's used in the protocol. If A is any algorithm that makes at most l oracle queries, then the probability that the adversary ever produces any group element equal to b is $O(\frac{l^2}{p})$.*

It follows from this theorem that, if the adversary is to find the secret base b with probability bounded away from zero, then he must make $\Omega(\sqrt{p})$ oracle queries.

In order to prove this, we need the following fact from [8]:

Lemma 4.2 *If $f \in \mathbb{Z}_p[X_1, X_2, \ldots, X_k]$ is any polynomial of degree d, and each X_i is restricted to at most B values, then the probability that a random assignment of the X_i's is a root of f is at most $\frac{d}{B}$.*

Proof: Let $i_0, \ldots i_{r-1}$ be the queried indices. Let $X_1, \ldots, X_{\sqrt{n}}$ be indeterminants in $\mathbb{Z}_p$, referred to simultaneously as $\mathbf{X}$. Let Z be an additional indeterminate.

The game maintains four lists:

- A list of rational functions $E_1, \ldots, E_t$ over $\mathbb{Z}_p$, representing the values exponents of b. The game begins with $t = \sqrt{n}$, so that, when i_l is a queried index, $E_{i_l}(\mathbf{X}) = N^l + N^r X_{i_l} + Z$. Likewise, when i is an unqueried index, $E_i(\mathbf{X}) = N^r X_i + Z$. Note that, for all $i \leq \sqrt{n}$, b^{E_i} corresponds exactly to b_i in the protocol.

- A list of random strings $\sigma_1, \ldots, \sigma_t \in \{0,1\}^*$, satisfying $\sigma_i = \sigma_j$ precisely when $E_i = E_j$. Each σ_i corresponds to b^{E_i}.

- A list of rational functions $h_1, \ldots, h_s$ over $\mathbb{Z}_p$, initially empty. Each h_i represents the answer to a discrete log and other arithmetic query.

- A list of random strings $\tau_1, \ldots, \tau_s \in \{0,1\}^*$, satisfying $\tau_i = \tau_j$ precisely when $h_i = h_j$. Each τ_i corresponds to h_i.

At the start of the game, the adversary receives $\sigma_1, \ldots, \sigma_{\sqrt{n}}$. The following queries are allowed:

- Group operation — The adversary gives two indices i and j, and a bit z. The game computes:

  $$E_{t+1} = E_i + (-1)^z E_j$$

  and stores it on the list. If $E_{t+1} = E_k$ for some $k \leq t$, then it sets $\sigma_{t+1} = \sigma_k$. Otherwise, it generates a new random string σ_{t+1} distinct from $\sigma_1, \ldots, \sigma_t$. Output σ_{t+1} to the adversary.

- Discrete log — The adversary gives two indices i and j. The game sets $h_{s+1} = \log_{b^{E_i}}(b^{E_j}) \equiv \frac{E_j}{E_i} \pmod{m}$. If $h_{s+1} = h_k$ for some $k \leq s$, then it sets $\tau_{s+1} = \tau_k$. Otherwise, it generates a new random string τ_{s+1} distinct from $\tau_1, \ldots, \tau_s$. Output τ_{s+1} to the adversary.

- Exponentiate — The adversary gives two indices i and j. The game computes $E_{t+1} = h_j \cdot E_i$. This is the exponent of $(b^{E_i})^{h_j}$. If $E_{t+1} = E_k$ for some $k \leq t$, then it sets $\sigma_{t+1} = \sigma_k$. Otherwise, it generates a new random string σ_{t+1} distinct from $\sigma_1, \ldots, \sigma_t$. Output σ_{t+1} to the adversary.

- Arithmetic — The adversary gives two indices i and j, and an operation $\diamond \in \{+, -, \times, \div\}$ The game computes $h_{s+1} = h_i \diamond h_j$ If $h_{s+1} = h_k$ for some $k \leq s$, then it sets $\tau_{s+1} = \tau_k$. Otherwise, it generates a new random string τ_{s+1} distinct from $\tau_1, \ldots, \tau_s$. Output τ_{s+1} to the adversary.

If any of the operations involve dividing by 0, the game returns $\perp$ and does not store anything from that query.

When the adversary's algorithm terminates, the game randomly selects $x_1, \ldots, x_{\sqrt{n}}$, and $z \in \mathbb{Z}_p$, instantiations of $X_1, \ldots, X_{\sqrt{n}}$, and Z. The adversary wins if any of these conditions are met:

- There is an index i such that $E_i(\mathbf{x}, z) = 1$. This means that the adversary found b.

- There are indices i, j such that $E_i \neq E_j$, but

  $$E_i(\mathbf{x}, z) = E_j(\mathbf{x}, z)$$

  In this case, the game incorrectly told the adversary that two group elements were different when they were actually the same.

- There are indices i, j such that $h_i \neq h_j$, but

$$h_i(\mathbf{x}, z) = h_j(\mathbf{x}, z)$$

  In this case, the game incorrectly told the adversary that the results of two discrete log or arithmetic operations were different when they were actually the same.

- For some $i \leq s$, $h_i(\mathbf{x}, z)$ is undefined. This happens when the adversary asked the game to take a discrete log with respect to the identity, or to otherwise divide by zero, but the game returned an answer.

At termination, the elements on the group exponent list are of two types: $E(\mathbf{x}, z) = p(\mathbf{x}, z)$ where p is a linear polynomial, and $E(\mathbf{x}, z) = p(\mathbf{x}, z)R(\mathbf{x}, z)$ where p is a linear polynomial and R is a rational function.

Consider this second class of elements where $E(\mathbf{x}, z)$ is nonzero. Multiplication by any of the rational functions (as a result of exponentiation operations) creates a new random element on the interval $[0, n)$, except the number of possible values is lower bounded by B. All of these elements are distinct, so all are distributed throughout the interval, on a set S of points where $|S| \geq B$. We can show this last claim using a simple inductive argument. Thus the probability that any of these exponents in the second class is equal to 1 is bounded by $1/B$. We now consider the exponents in the first class. Here we apply the lemma from [8] (Lemma 4.2) and see that the probability that any of these exponents equals 1 is bounded by $1/B$. So the probability that all exponents on the group list are different from 1 is $(1 - 1/B)^r$.

We now consider the case where $E_i(\mathbf{x}, z) = E_j(\mathbf{x}, z)$ for some i and j where $i \neq j$. If E_i and E_j are exponents in the first class, then Lemma 4.2 again applies, so the probability of this event is bounded by $1/B$. Now suppose $E_i = p$ is in the first class, and $E_j = f/g$ is in the second class. Clearly not all terms include every x_i, otherwise we wouldn't have a root. Let x_1 not be included in some term, but present in others. Then we can view $h(\mathbf{x}, z) = pg - f$ as a polynomial in a single variable x_1, with nonzero constant term, and root x_1. We can show that the constant term takes on at least B uniformly distributed different values (factor out one x_i and set the other x_i's and z to arbitrary nonzero values). Therefore $Prob(h(\mathbf{x}, z) = 0) \leq 1/B$.

Now consider the case where E_i and E_j are both exponents in the second class. The same argument applies as in the preceding case. There are $\binom{r}{2} \approx \frac{r^2}{2}$

such pairings, so the probability that all values are distinct is $\left(1-\frac{1}{B}\right)^{\frac{r^2}{2}}$.

The case where $h_i(\mathbf{x}, z) = h_j(\mathbf{x}, z)$ is proved in a similar manner; we get $\left(1-\frac{1}{B}\right)^{\frac{s^2}{2}}$ as the probability that all such pairs are distinct.

Also, $h_i(\mathbf{x}, z)$ is always defined since the group has prime order.

Multiplying all of these together, we get roughly

$$P(\text{Adversary wins}) \leq 1 - \left(1-\frac{1}{B}\right)^{r^2+s^2} \approx (r^2+s^2)/B$$

This last value is $O(\frac{l^2}{p})$. ∎

4.3 Reduction of security to finding the secret base

In this section we show that, when $\mathcal{DB}$ is viewed as a table of bits, corresponding to $N = 2$, any attack which can break security can be modified to find the secret base, already proven difficult. A similar reduction would work for any small value for N, so long as it is reasonable to perform $O(N)$ additions.

In this section we show the security of our protocol in the case of $N = 2, r = 1$, meaning the entries of $\mathcal{DB}$ are bits and the user requests a single entry. A successful attack in this case is one which, given b^e for some $e < \frac{m}{\sqrt{n}(N-1)} = \frac{m}{\sqrt{n}}$, can determine whether e is even or odd with a probability $\frac{1}{2} + \frac{1}{f(\log m)}$, where f is a polynomial, and the probability is taken over the choice of b, e, and and coin flips made by the oracle.

We say that such an oracle is *reliable* when it determines even/odd with neglibible error, and *faulty* otherwise.

We will show how to use a reliable even/odd oracle to learn the secret base. Afterward, we show how turn a faulty oracle into one which is reliable for our purposes. Finally, we use the security result from Section 4 to conclude that our protocol is secure.

Theorem 4.3 *An adversary who has access to a reliable even/odd oracle can find the secret base b with high probability, using $O(\log m)$ oracle calls and $O(\log^2 m)$ group operations.*

Proof sketch: Let $m, n, e_1, \ldots, e_{\sqrt{n}}, b, b_1, \ldots, b_{\sqrt{n}}$ all be as defined in the protocol.

Fix an integer c, and pick $2c$ random group elements with even exponents from $\{b_1, \ldots, b_{\sqrt{n}}\}$. Written as $b_i = b^{e_i}$, each of the corresponding exponents is less than $B = \frac{m}{\sqrt{n}}$.

For each selected b_i, we calculate $\sqrt{b_i} = b^{\frac{e_i}{2}}$. Note that, since $\frac{1}{2} \equiv \frac{m+1}{2} \mod m$, each of these calculations takes about $\log m$ group operations using fast exponentiation.

Successively cut each exponent in half until the result is odd. Since each exponent was originally less than B, we can bound each of the exponents above by $2^{-s}B$, where s is the number of times it has been cut in half.

Given b^x, b^y, with x, y both odd and less than $2^{-s}B$, we multiply them and raise the result to the $\frac{1}{2}$ power to get $b^{\frac{x+y}{2}}$, which is still less than $2^{-s}B$. With probability $\frac{1}{2}$, this number will be even, so we make the exponent smaller. We repeat this process until we find $b = b^1$.

After cutting the original exponents in half, we expect about c odd exponents less than $\frac{B}{2}$ (with the others bounded even lower). We would like to maintain this number of distinct exponents at each level. With c exponents, there are $\binom{c}{2} \approx \frac{c^2}{2}$ possible values for $\frac{x+y}{2}$ as above. About half of these will be even, and a quarter can be cut in half more than once. This gives roughly $\frac{c^2}{8}$ smaller odd exponents, though some of them may be equal. To give a much higher chance of getting c distinct smaller exponents, we use $\frac{c^2}{8} > 10c$. This suggests c should be around 80. Note that, as the bounds on the exponents become small, finding distinct exponents becomes impossible, since (for example) there are only 5 odd numbers less than 10, meaning a search for 80 of them would be fruitless. Fortunately, if we run out of exponents for this reason, the last exponents found will still be bounded above by around c. Let b^x be such a group element. Then we can exhaustively test for the secret base, beginning with b^x, then $b^{\frac{x}{3}}, b^{\frac{x}{5}}, \ldots$, until it is found.

In order to get to b, we need to cut the exponents in half $\log B$ times. For each step, we need to do about c^2 oracle calls, and $c^2 \log m$ group operations. We must repeat the process $\log B = \log \frac{m}{\sqrt{n}}$ times to force an exponent to be 1. This gives us the promised $O(\log m)$ oracle calls and $O(\log^2 m)$ group operations. ∎

To complete the reduction, we explain informally how a faulty oracle can be used to create an oracle which is reliable for our purposes. This would be

trivial if even/odd queries were randomly self-reducible, We show that the problem is self-reducible, but not every reduction is possible.

Suppose we have access to a faulty oracle which, given b^e for some $e < B$ determines even/odd with probability $\frac{1}{2} + \frac{1}{f(\log m)}$ for some polynomial f. Then, if e is small enough, the oracle can also accept b^{3e}, b^{5e}, and more, each preserving the parity of e.

Specifically, if $e < 2^{-s}B$, we have 2^{s-1} equivalent exponents to query — $e, 3e, 5e, \ldots, (2^s - 1)e$ — so random reductions among these can be used to bolster confidence in the oracle. Once this 2^{s-1} grows larger than $f(\log m)$, the adversary can decide with high confidence whether a number is even or odd by making queries to a random selection of these.

Thus, after the exponents decrease to below $\frac{B}{f(\log m)}$, the faulty oracle can be used as a reliable one. To get to this point in the reduction, it takes $O(\log f(\log m))$ oracle calls. Since the oracle is accurate with probability $\frac{1}{2} + \frac{1}{f(\log m)}$ we can repeat the entire reduction $2^{O(\log f(\log m))} = O(f(\log m))$ times, so that we can expect at least one of them to correctly perform until the oracle becomes reliable.

All told, each call to the reliable oracle takes $f(\log m)$ calls to the faulty oracle, and the reduction above is repeated $O(f(\log m))$ times, meaning the complete reduction takes $O(f(\log m)^2 \log m)$ oracle calls and $O(f(\log m) \log^2 m)$ group operations, which are both polylogarithmic in m.

So how hard is it to break the security of our protocol? We know that, if m is prime, it takes $\Omega(\sqrt{m})$ group operations for an adversary to find the secret base. Taken together, it takes $\Omega(\frac{\sqrt{m} - f(\log m)\log^2 m}{f(\log m)\log m}) = \Omega(m^{\frac{1}{2}-\epsilon})$ operations to determine even/odd. Because this is exponential in $|m|$, we conclude that our protocol is secure.

4.4 A lattice-based attack

The GGM, and our extension of it, rules out some attacks against our scheme. In particular, it rules out attacks where the group elements are treated in an opaque manner by the adversary. In a lattice attack, the group elements are treated as real numbers, and the resulting vectors are subjected to a distance metric (a lattice is the set of integral linear combinations of a set of basis vectors from R^n). Here we show an attack on our algorithm based on the LLL lattice reduction algorithm. This attack will lead to a necessary lower

bound for the choice of m.

Let $G, m, n, N, r, e_1, \ldots, e_{\sqrt{n}}, b, b_1, \ldots, b_{\sqrt{n}}$ all be as defined in the modified protocol. Define $B = \frac{m}{\sqrt{n}(N-1)N^r}$.

Recall that, for the protocol, it is required that discrete log is efficiently computable in G. Thus, for each i, we can compute

$$x_i = \log_{b_1} b_i = \log_{b_1} b^{e_i+t} \equiv (e_i + t) \log_{b_1} b \mod m$$

Letting $x = \log_{b_1} b$, we see that $x_i \equiv (e_i + t)x \mod m$. Next, to get rid of t, we calculate

$$y_i' \equiv x_i - x_1 \equiv (e_i + t)x - (e_1 + t)x \equiv (e_i - e_1)x \mod m$$

Thus $y_i' = (e_i - e_1)x + k_i m$ for some $k_i \in \mathbb{Z}$. The use of e_1 is of course arbitrary — we use 1 as a convenience, but treat it as an arbitrary unqueried index.

Now, fix a number c with $1 < c \leq \sqrt{n}$, and randomly select $i_1, i_2, \ldots, i_c \leq \sqrt{n}$, unqueried indices from the protocol. Then each e_i is a multiple of N^r, so let

$$y_{i_j} = \frac{y_{i_j}'}{N^r} \equiv (a_{i_j} - a_1)x \mod m$$

where $a_1, a_{i_1}, \ldots, a_{i_c} < B$ are the random numbers chosen by the user.

We now define the following vectors in $\mathbb{Z}^c$:

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

We are interested in the shortest vector in the lattice spanned by these $c + 1$ vectors, in the Euclidean norm.

Let $z \equiv x^{-1} \mod m$. Then

$$z\mathbf{v}_0 = (zy_{i_1}, zy_{i_2}, \ldots, zy_{i_c}) = (a_{i_1} - a_1 + k_1' m, \ldots, a_{i_c} - a_1 + k_c' m)$$

for some constants $k_1', \ldots, k_c' \in \mathbb{Z}$.

Since $\mathbf{v}_1, \ldots, \mathbf{v}_c$ allow us to remove any multiples of m, we see there is a vector $\mathbf{w}$ in the lattice given by

$$\mathbf{w} = (a_{i_1} - a_1, \ldots, a_{i_c} - a_1)$$

Since $0 \leq a_i < B$, we see that $|a_{i_c} - a_1| < B$. Thus,

$$\|\mathbf{w}\| = \left(\sum_{j=1}^{c} (a_{i_j} - a_1)^2 \right)^{\frac{1}{2}} < B\sqrt{c}$$

Of course, if the adversary learns the vector $\mathbf{w}$, she learns the exponents of known group elements. By exponentiating any of these elements, she can easily find b. Also, if the adversary learns a small multiple of $\mathbf{w}$, then she can compute the gcd to find the multiple and therefore find $\mathbf{w}$.

The LLL algorithm, with an input of $c + 1$ vectors, is guaranteed to return a vector $\mathbf{z}$ such that, for every non-zero vector $\mathbf{x}$ in the lattice,

$$\|\mathbf{z}\| \leq \left(\frac{2}{\sqrt{3}} \right)^c \|\mathbf{x}\|$$

Now, looking at the lattice in $(\mathbb{Z}_m)^c$, we see that $\mathbf{v}_0$ is the only basis vector different from $\mathbf{0}$. Thus, every vector has the form $\lambda \mathbf{v}_0$ for some $\lambda \in \mathbb{Z}_m$. Thus, there are only $m - 1$ non-zero vectors in the lattice when reduced modulo m — $\mathbf{v}_0, 2\mathbf{v}_0, \ldots, (m-1)\mathbf{v}_0$. We use $\|\mathbf{u}\|_*$ to refer to the norm of the shortest vector which is congruent to $\mathbf{u}$ modulo m. For any $1 \leq i \leq m - 1$, let $\mathbf{x}_i$ be the shortest vector which is congruent to $i\mathbf{v}_0$ modulo m. The LLL algorithm will return one of the $\mathbf{x}_i$ vectors (if it didn't, the adversary could reduce modulo m to obtain an $\mathbf{x}_i$ vector). (Note: the $\mathbf{x}_i$ vectors form a Z_m module).

We now give lower bounds on the probability P that a vector is in the proper bounded range to be returned by the LLL algorithm ($\|\mathbf{z}\| \leq \left(\frac{2}{\sqrt{3}}\right)^c \|\mathbf{s}\|$ where $\mathbf{s}$ is the shortest vector in the lattice.) If the order of the group, m, is sufficiently large, then the expected number of vectors mP which can be returned by the LLL algorithm will be so large that the probability of returning a small multiple of $\mathbf{w}$ will be very small. Let $\mathbf{s}$ denote the shortest vector in the above lattice. We first prove a lemma:

Lemma 4.4 *$\|\mathbf{w}\| \geq B/2^{\max\{1, -\log(\epsilon_2)/c\}}$ with probability $\geq 1 - \epsilon_2$.*

Proof: $Prob(\|\mathbf{w}\| < B/2^i) < Prob(all\ |w_i| < B/2^i) = (1/2^i)^c = 1/2^{ic}$. Let $ic = -log(\epsilon_2)$. Then $\|\mathbf{w}\| \geq B/2^{\max\{1,-\log(\epsilon_2)/c\}}$ with probability at least equal to $1 - \epsilon_2$. ∎

The above lemma will be used with $\epsilon_2 = 1/2^{80}$. We now obtain lower bounds on the number of candidate vectors that can be returned by the LLL algorithm. We handle this task in two cases: the first case being where the shortest vector in the lattice, $\mathbf{s}$, is equal to $\mathbf{w}$, and the second case being where they are distinct.

case 1: $\mathbf{s} = \mathbf{w}$

Let $h(c) = \left(\frac{2}{\sqrt{3}}\right)^c \|\mathbf{w}\|/\sqrt{c}$. Suppose all coordinates of a vector $\mathbf{v}$ satisfy $|v_i| \leq h(c)$ Then $\|\mathbf{v}\| \leq \left(\frac{2}{\sqrt{3}}\right)^c \|\mathbf{w}\|$ so $\mathbf{v}$ can be returned by the LLL algorithm.

Thus each $|v_i|$ can be either shorter than $h(c)$, or greater than $m - h(c)$ — a total of $2h(c)$ possible values.

Since each a_{i_j} is picked at random from $\{1, \ldots, B\}$, each coordinate of $\mathbf{w}$ is uniformly distributed in $\{1 - a_1, \ldots, B - a_1\}$. Let a be the maximum coordinate (in absolute value) of $\mathbf{w}$. Then for $|\lambda| < \frac{m}{2a}$, we get that

$$\|\lambda\mathbf{w}\|_* = \|\lambda\mathbf{w}\| = |\lambda| \cdot \|\mathbf{w}\|$$

since all coordinates are in $\left(\frac{-m}{2}, \frac{m}{2}\right)$, making them as small as possible. For $|\lambda| \geq \frac{m}{2a}$, this is no longer the case, so some mod m reductions can make the vectors smaller. However, for these values of λ, "wrapping around" makes the coordinates of $\lambda\mathbf{w}$ effectively random. Thus, for a fixed $|\lambda| \geq \frac{m}{2a}$, the probability of all c random coordinates in $\mathbb{Z}_m$ taking only these $2h(c)$ permissible values is given by $(2h(c)/m)^c$. Thus

$$P \geq \left(2\left(\frac{2}{\sqrt{3}}\right)^c \|\mathbf{w}\|/(\sqrt{c}\ m)\right)^c \geq \left(2\left(\frac{2}{\sqrt{3}}\right)^c B/(\sqrt{c}\ m 2^{max\{1,80/c\}})\right)^c =$$
$$\left(2\left(\frac{2}{\sqrt{3}}\right)^c /(\sqrt{c}\ 2^{max\{1,80/c\}}\sqrt{n}(N-1)N^r)\right)^c \quad (1)$$

where we have used Lemma 4.4 above. We note that we have assumed that $2\left(\frac{2}{\sqrt{3}}\right)^c \|\mathbf{w}\|/\sqrt{c} \leq m$.

case 2: $\mathbf{s} \neq \mathbf{w}$

We will first prove some lemmas:

Lemma 4.5 *Let $\alpha < 1$. Consider the set $S(\alpha) = \{\mathbf{v}$ is wrapped with respect to $\mathbf{w}$ ($\mathbf{v} = \lambda\mathbf{w}$ where $\lambda \geq m/2a$) and $|v_i| \leq \alpha\|\mathbf{w}\|$ for $1 \leq i \leq c$. Then either the expected size of $S(\alpha)$ is less than or equal to $2^{-\epsilon}$, or $(2\sqrt{c}\alpha/\sqrt{n}(N-1)N^r)^c \geq 2^{-\epsilon}/m$.*

Proof: Since the coordinates of $\mathbf{v} \in S(\alpha)$ are random, the probability that $|v_i| \leq \alpha\|\mathbf{w}\|$ for $1 \leq i \leq c$, is $(2\alpha\|\mathbf{w}\|/m)^c \leq (2\alpha B\sqrt{c}/m)^c = (2\alpha\sqrt{c}/\sqrt{n}(N-1)N^r)^c$. If this last probability is less than $2^{-\epsilon}/m$, then the expected number of vectors in $S(\alpha)$ is less than $2^{-\epsilon}$. ∎

Lemma 4.6 *If $\mathbf{w}$ satisfies $\|\mathbf{w}\| \leq \left(\frac{2}{\sqrt{3}}\right)^c \|\mathbf{s}\|$ and $\|\mathbf{s}\| = \alpha\|\mathbf{w}\|$, then $1/\alpha \leq \left(\frac{2}{\sqrt{3}}\right)^c$.*

Proof: $\|\mathbf{w}\| \leq \left(\frac{2}{\sqrt{3}}\right)^c \|\mathbf{s}\| = \left(\frac{2}{\sqrt{3}}\right)^c \alpha\|\mathbf{w}\|$, so $1/\alpha \leq \left(\frac{2}{\sqrt{3}}\right)^c$. ∎

The conditions of Lemma 4.6 are trivially satisfied, since if $\mathbf{w}$ is not in LLL range, then there is nothing to prove.

Now since $\mathbf{s} \neq \mathbf{w}$, we have that $\|\mathbf{s}\| = \alpha\|\mathbf{w}\|$ for some $\alpha < 1$ so $|s_i| \leq \alpha\|\mathbf{w}\|$, $1 \leq i \leq c$. Also, $\mathbf{s}$ is wrapped with respect to $\mathbf{w}$. Therefore, $\mathbf{s} \in S(\alpha)$, and by Lemma 4.5 (with high probability), it follows that

$$\begin{aligned}
\left(2\sqrt{c}\alpha/\sqrt{n}(N-1)N^r\right)^c &\geq 2^{-\epsilon}/m \\
c\left[\log(2\sqrt{c}\alpha) - \log(\sqrt{n}(N-1)N^r)\right] &\geq -\epsilon - \log(m) \\
\log(2\sqrt{c}) + \log(\alpha) &\geq \left[-\epsilon - \log(m) + c\log(\sqrt{n}(N-1)N^r)\right]/c \\
\alpha &\geq \frac{1}{2\sqrt{c}}\, 2^{[-\epsilon-\log(m)+c\log(\sqrt{n}(N-1)N^r)]/c} \\
\alpha &\geq \frac{1}{2\sqrt{c}}\, 2^{(-\epsilon-\log(m))/c}\sqrt{n}(N-1)N^r \qquad (2)
\end{aligned}$$

At this point, we have two different lower bounds on α which we will use to establish lower bounds on the number of vectors which can be returned by the LLL algorithm. We will show that the set of vectors which can be returned is large.

Theorem 4.7 *Given a vector z in the lattice. Then the probability that z is in LLL range is greater than or equal to $\left(\frac{2\left(\frac{2}{\sqrt{3}}\right)^c\alpha}{\sqrt{c}\ 2^{max\{1,80/c\}}\beta}\right)^c$ where $\beta = \sqrt{n}(N-1)N^r)$, $\|\mathbf{s}\| = \alpha\|\mathbf{w}\|$, with probability greater than $1 - (1/2^{80})$.*

Proof: If $|z_i| \leq \left(\frac{2}{\sqrt{3}}\right)^c \|\mathbf{s}\|/\sqrt{c}$ for $1 \leq i \leq c$, then $\mathbf{z}$ is in LLL range. Thus

$$P = Prob(z \; is \; in \; LLL \; range) \geq \left(\frac{2\left(\frac{2}{\sqrt{3}}\right)^c \|\mathbf{s}\|}{\sqrt{c}\; m} \right)^c = \left(\frac{2\left(\frac{2}{\sqrt{3}}\right)^c \alpha \|\mathbf{w}\|}{\sqrt{c}\; m} \right)^c$$

The last expression is greater than or equal to

$$\left(\frac{2\left(\frac{2}{\sqrt{3}}\right)^c \alpha B}{\sqrt{c}\; 2^{max\{1,80/c\}} m} \right)^c = \left(\frac{2\left(\frac{2}{\sqrt{3}}\right)^c \alpha}{\sqrt{c}\; 2^{max\{1,80/c\}} \beta} \right)^c = Q$$

with probability greater than $1 - 1/2^{80}$. We have two different lower bounds on α ((2) and Lemma 4.6); we obtain

$$Q \geq \max\{ \left(\frac{2}{\sqrt{c}\; 2^{max\{1,80/c\}} \beta} \right)^c, \left(\frac{\left(\frac{2}{\sqrt{3}}\right)^c}{c\; 2^{max\{1,80/c\}} 2^{(\epsilon + \log(m))/c}} \right)^c \}$$

∎

The first bound decreases as c increases, and the second bound increases as c increases. Let T be the number of non-mod'd multiples of w in the lattice (the number of multiples before the largest value wraps due to modulo m reductions). The following table, Table 1, gives the base 2 log of the group order ($\log(m)$) values such that mP/T is greater than 2^{80}. We note that we set $\epsilon = 80$ to obtain the results. This ensures that (2) holds with high probability.

5 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.

n	N	r	$\log(m)$ for $s \neq w$	$\log(m)$ for $s = w$	$\log(m)$
2^{40}	2	1	1293	812	1293
2^{40}	2	32	2874	3850	3850
2^{30}	2	1	1038	558	1038
2^{30}	2	32	2619	3200	3200
2^{20}	2	1	783	367	783
2^{20}	2	32	2364	2605	2605
2^{35}	2^{32}	1	4353	8763	7516
2^{40}	2^{32}	1	4506	7756	7756

Table 1: Group sizes for our protocol that give 2^{80} or better security against the lattice attack

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. As noted in [11], 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 will turn out to be the bottleneck.

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) 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 $\sqrt{n}$. As noted in [11], $\mathcal{DB}$ is destined to be stuck with $\Omega(m)$ computation in any PIR scheme, so we cannot hope for any better.

6 Performance analysis

The goal of this protocol is not to push down the bounds on complexity of cPIR, which has already been done in [1], [2], and [7]. Instead, the goal is a cPIR protocol which is faster than the trivial PIR protocol — transferring the entire database in question.

In [10], it was shown that the current best cPIR protocols are orders of magnitude slower than sending an entire database to the user. His results explained that, because of the large cost of $O(n)$ modular multiplications in $\mathbb{Z}_m^*$ for [6], 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.

We use the numbers and formulas put together by Sion in [10] to show the performance of our algorithm, and compare it to that of database transfer.

6.1 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 operation
- t_t — time needed to transfer a bit over the network

From the verified assumption in [10], we estimate that $t_d \approx \frac{1}{M}$, and $d = 5$. Additionally, we estimate

$$t_{add}(m) \approx \frac{|m|}{M \times d} \ ^{1}$$

[1] A naive approach to addition modulo m takes 2 additions: first adding, and then (possibly) subtracting m. This would suggest $t_{add}(m) \equiv \frac{2|m|}{M \times d}$. However, in summing

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

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

6.2 Analysis

Let $\mathcal{DB}$ be a database with n entries, each a member of $\mathbb{Z}_2$. Let k be the security parameter, and m be an appropriate modulus for the protocol (see Table 1 for m values), in which we request one or 32 entries.

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 5, 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 multiplcations performed by $\mathcal{DB}$, the third is multiplications by the user, and the final term is communication between the two parties.

Our goal is now to see for which parameters $T_{pir} < T_{trans}$.

In [10], Sion 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 size 1 MB, 1 GB, and 1 TB, with entries viewed as bits. With entries viewed as bits, where a single bit is requested, the protocol mandates m have 783, 1038, and 1293 bits, depending on the database. If 32 bits are requested ($r = 32$), then we need m to have 2605, 3200, and 3850 bits, respectively. Fix the security parameter as $k = 80$. For T_{pir}, we assume the end-user bandwidth for all data transfers, because the time savings are negligible.

---

many terms, we can perform, for example, 100 additions, and then subtract $50m$ — the expected magnitude of the result — and make a one-time adjustment at the end of the summation. Ammortized, this effectively drops the coefficient to 1, as in our estimate.

[2] This is a natural extension of the estimate from [10].

	T_{pir}		T_{trans}		
Size	bits	integers	End-user	Inter-site	LAN
1 MB	0.0563 s	0.20723 s	1.398 s	0.005592 s	0.0008389 s
1 GB	71.52 s	221.45 s	23.87 mins	5.727 s	0.8590 s
1 TB	25.27 hours	75.275 hours	16.97 days	97.73 mins	14.66 mins

Table 2: 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 high-end, intersite networks, transfering the database takes about 7-10% the time as PIR with $N = 2$, but takes 2% compared to PIR with $r = 32$. When operating in a 10 Gbps Ethernet LAN, the trivial protocol enjoys are even larger speed advantage. Summarizing, for the slower connections, our PIR protocol is faster than the trivial protocol, but for faster connections, the trivial protocol is faster. In [10], the authors show that the trivial protocol is significantly faster, on all types of networks, than all previous cPIR protocols.

7 Conclusions and open problems

We have developed a new protocol which offers several significant speedups over previous work in cPIR. We have shown that the underlying hardness assumption, finding the secret base, has a computational lower bound in an extended generic group model. The security of our protocol also reduces to this hardness assumption. We have obtained group sizes that are secure against a LLL lattice based reduction attack. Our protocol is orders of magnitude faster than existing protocols, according to the methodology used in [10].

The security for the case where the group order is composite instead of prime, is an open question. The composite case may be vulnerable to lattice based attacks.

References

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

[2] Y. Chang. Single-Database Private Information Retrieval with Logarithmic Communication. In *Proceedings of the 9th AUstralasian Conference on Information Security and Privacy ACISP*. Springer-Verlag, 2004.

[3] 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.

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

[5] 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.

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

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

[8] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701-717, 1980.

[9] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. *Theory and Application of Cryptographic Techniques*, 1997.

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

[11] A. Yerukhimovich. A General Framework for One Database Private Information Retrieval. Online at http://www.cs.umd.edu/Grad/scholarlypapers/papers/Arkady-pircomp.pdf