# Computationally Inexpensive cPIR

William Gasarch and Arkady Yerukhimovich

Dept. of Computer Science, University of Maryland, College Park, MD, USA
{gasarch,arkady}@cs.umd.edu

**Abstract.** Computational Private Information Retrieval(cPIR) schemes allow a user to retrieve data from a database privately while exchanging only a small amount of data between the database and the user, where the privacy is based on a cryptographic assumption. All previous cPIR protocols require the database to compute a modular multiplication for every bit of the database on each query, making them too computationally expensive to be practical. In this paper, we propose two protocols that achieve the $O(n^\epsilon)$ communication complexity of previous protocols, while significantly reducing the amount of computation that must be done per query.

**Keywords:** Computational Private Information Retrieval, Lattice Theoretic Cryptography

## 1 Introduction

In Private Information Retrieval(PIR) we view the database as an $n$-bit string $x$ and the user wants to retrieve the bit $x_i$ while keeping $i$ private from the database. The traditional, information theoretic, model of PIR was formulated by Chor, et. al. in [4]. The authors show that $n$ bits of communication are necessary to achieve information theoretic privacy if only one database is available. In order to circumvent this bound, the authors propose using multiple non-interacting databases. Much work in this area has produced upper and lower bounds for information theoretic PIR schemes with different numbers of databases. For a list of these results and related topics see [5, 6].

A different model of PIR called Computational Private Information Retrieval(cPIR) was proposed by Kushilevitz and Ostrovsky in [7] to get around this issue of multiple databases. In this model, the authors show an $O(n^\epsilon)$ communication complexity, single server cPIR protocol where the security of the user's query is based on a cryptographic assumption. In a further result [8], Mann showed that such a cPIR scheme can be built using any bitwise encryption scheme with certain homomorphism properties.

All these results deal with optimizing the communication complexity of cPIR protocols. However, a problem that has largely been ignored by all of them is the computational complexity of such algorithms. This is in part due to an obvious lower bound that the server must perform $\Omega(n)$ computation, because if the database does not touch a bit in generating its response it knows the user did not request that bit, thus violating the security. All previous cPIR schemes required the database to compute a modular multiplication for each bit of the database making them very computationally expensive and infeasible in practice [13].

In this paper, we present two different cPIR protocols with $O(n^\epsilon)$ communication complexity whose security is based on the worst-case hardness of some lattice problems which are conjectured to be hard. In particular, both of our protocols accomplish this while requiring the server to perform only $O(n)$ modular additions and no multiplications per query. The protocols we propose use the ideas from [8] to turn the public key cryptosystems of [11, 12] into PIR protocols.

One drawback of our protocols is that, unlike previous cPIR protocols, our protocol has a non-zero probability of error. We show that this error is negligibly small in the size of the database.

This paper is organized as follows. First, in section 2, we briefly review lattices and the problems on which we base the security of out protocol. Then, in section 3 we present two versions of our first protocol. In section 3.1 we present a basic scheme that does not achieve the promised communication complexity, but is used as a building block for the final, recursive, scheme, which we present in 3.2. In section 4 we present two versions of our second protocol, again presenting it in two parts as the first. Finally, in Section 5 we compare the two protocols with the original protocol from [7].

## 2 Lattice Definitions [10]

**Definition 1.** *A lattice in $m$-dimensional Euclidean Space $\mathbb{R}^m$ is the set*

$$\mathcal{L}(\mathbf{b_1},\ldots,\mathbf{b_n}) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b_i} : x_i \in \mathbb{Z} \right\}$$

*of integral combinations of linearly independent vectors $\mathbf{b_1},\ldots,\mathbf{b_n}$ which are called a lattice basis. Equivalently, we can write the lattice as*

$$\mathbf{B} = [\mathbf{b_1},\ldots,\mathbf{b_n}] \in \mathbb{R}^{m\times n}$$
$$\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx} : \mathbf{x} \in \mathbb{Z}^n\}$$

where $\mathbf{b_i}$ is a length $m$ column vector. The integer $m$ is the *dimension* of the lattice and $n$ is the *rank*. For the sake of this paper we will assume $n = m$.

**Definition 2.** *$\gamma$-Approximate Shortest Vector Problem(SVP): Given a basis $\mathbf{B} \in \mathbb{Z}^{m\times n}$, find a nonzero lattice vector $\mathbf{Bx}$ such that $||\mathbf{Bx}|| \leq \gamma \cdot ||\mathbf{By}||$ for any $y \in \mathbb{Z}^n \backslash \{0\}, y \neq x$.*

**Definition 3.** *$n^c$-Unique Shortest Vector Problem(uSVP):Given a lattice basis $\mathbf{B}$ find the shortest nonzero vector $\mathbf{v}$ in $\mathcal{L}(\mathbf{B})$ where you are promised that the shortest vector is unique in the sense that any other vector whose length is at most $n^c||\mathbf{v}||$ is parallel to $\mathbf{v}$ [2].*

**Definition 4.** *$\gamma$-Approximate Shortest Independent Vector Problem(SIVP): Given a lattice basis $\mathbf{B}$ of rank $n$, find linearly independent lattice vectors $\mathbf{s_1},\ldots,\mathbf{s_n}$ such that $\forall i \in [n], ||\mathbf{s_i}|| \leq \gamma \cdot \lambda_n(\mathcal{L}(\mathbf{B}))$ where $\lambda_n(\mathcal{L}(\mathbf{B}))$ is the $n$th successive minimum of $\mathcal{L}(\mathbf{B})$ defined as the radius of the smallest sphere centered at the origin containing $n$ linearly independent lattice vectors.*

Our protocols will be based on the worst-case hardness of the above three problems. The best known polynomial time algorithms for SVP and SIVP achieve only slightly sub-exponential approximation factors. In [9], Micciancio showed that it is $NP$-hard to approximate SVP for any approximation factor up to $\sqrt{2}$. It is therefore, conjectured that there is no classical polynomial time algorithm that approximates SVP and SIVP to within a polynomial factor. In [12], Regev conjectures that this holds for quantum algorithms. The uSVP problem is not as well studied as the other two and while this problem is not believed to be NP-complete it is believed to be hard. In [1] Ajtai conjectures that all three of these problems are hard.

## 3 uSVP Based Protocol

### 3.1 Basic Scheme

**Theorem 1.** *Let $k \in \mathbb{Z}, k > 4$, $n >> k$ is the size of the database. There exists a cPIR protocol with the following properties.*

1. *$O(n^{(k+4)/(2k)})$ communication complexity*
2. *The user does $O(n^{2/k})$ modular multiplications in the preprocessing phase. Then, the user performs $O(n^{(k+4)/(2k)})$ additions and 1 multiplication per query. The database performs $O(n)$ additions per query. All operations are done on $O(n^{2/k})$-bit numbers.*
3. *Probability of error less than $2^{-\omega(\log n)}$*
4. *Security guaranteed by the hardness of $l^{(k+6)/4}\log l$-uSVP on a lattice of dimension $l = n^{1/k}$*

*Proof.* We describe the cPIR protocol and then prove that it has the properties we claim.

**Intuition** The user views the database as a $\sqrt{n}\times\sqrt{n}$ array of bits, he wants bit $x_{i_0,j_0}$. As suggested in [8], the user generates an array of $\sqrt{n}$ numbers placing an encryption of 1 in Regev's cryptosystem at index $j_0$ and encryptions of 0 everywhere else. The database computes the inner product of each row with this array, this sum contains the encryption of 1 iff $x_{i_0,j_0} = 1$. Therefore, the user can determine the value of $x_{i_0,j_0}$ by checking this. The details are presented below.

**Definitions**

1. $n$ - size of the database
2. $k$ - parameter of our choice relating database size to dimension of lattice.
3. $l = n^{1/k}$ - dimension of lattice
4. $m = c_m l^2$ - integer s.t. $c_m > 7$ is the constant from Claim 5.3 in [11].
5. $N = 2^{8l^2}$
6. $\gamma$ - real number s.t. $\gamma(l) = \omega(l^{(k+4)/4}\sqrt{\log l})$
7. $\Psi_\alpha$ - normal distribution with mean 0 and standard deviation $\frac{\alpha}{\sqrt{2\pi}}$ reduced modulo 1.
8. $T_{h,\alpha}$ - distribution with peaks at multiples of $\frac{1}{h}$ with distribution $\Psi_\alpha$ around the peaks.
9. $frc(x) = |x - \lfloor x\rceil|$ - distance from $x \in \mathbb{R}$ to the nearest integer.

**Protocol**

1. The database is a $\sqrt{n}\times\sqrt{n}$ array of bits $\{x_{i,j}\}$. User has $i_0, j_0$ and wants to obtain $x_{i_0,j_0}$.
2. User chooses $h \in \mathbb{R}$ uniformly at random from the set $H = \{h \in [\sqrt{N}, 2\sqrt{N}) : frc(h) < \frac{1}{16m\sqrt{n}}\}$
3. Set $d = \frac{N}{h}$.
4. The user chooses $\alpha \in [\frac{2}{\gamma(l)}, \frac{2\sqrt{2}}{\gamma(l)})$ uniformly at random.
5. The user chooses $\{a_1, ..., a_m\}$ from $T_{h,\alpha}$ as follows.
   (a) For each $a_i$ the user does the following.
      i. User chooses $x_i \in \{0, 1, ..., \lceil h\rceil - 1\}$ uniformly at random, $y_i$ according to $\Psi_\alpha$
      ii. If $\frac{x_i+y_i}{h} < 1$, $a_i = \lfloor N(\frac{x_i+y_i}{h})\rfloor$. Otherwise repeat.
   (b) User finds an $i'$ s.t. $x_{i'}$ is odd. Note, $a_{i'}$ is near an odd multiple of $\frac{N}{h}$.
6. For each $j \in \{[\sqrt{n}] : j \neq j_0\}$

   (a) User chooses a random subset $S \subseteq [m]$
   (b) User computes $b_j = \sum_{i \in S} a_i \pmod N$
7. For $j = j_0$
   (a) User chooses a random subset $S \subseteq [m]$
   (b) User computes $b_j = \lfloor \frac{a_{i'}}{2} \rfloor + \sum_{i \in S} a_i \pmod N$
8. The user sends $b_1, \ldots, b_{\sqrt{n}}$ to the server. Note that $b_j$ is near a multiple of $\frac{N}{h}$ when $j \neq j_0$ and far from a multiple of $\frac{N}{h}$ when $j = j_0$.
9. For each row $i$, the server sends back $s_i = \sum_{j=1}^{\sqrt{n}} b_j x_{i,j} \pmod N$.
10. The user computes $w = s_{i_0}(\frac{h}{N})$.
    – If $frc(w) < \frac{1}{4}$ then $x_{i_0,j_0} = 0$, otherwise $x_{i_0,j_0} = 1$

**Note**: For practical purposes all numbers that are defined as reals should be viewed as a decimal which approximates a real number up to error $2^{-l}$. This is a binary decimal with $O(n^{1/k})$ bits after the decimal point. By doing this we discretize the distribution $T_{h,\alpha}$ with accuracy $2^{-l}$.

**Note**: Steps 2-5 can be done by the user without knowing $i_0$ and $j_0$. Therefore, these steps can be done offline to generate the $a_i$s and then the same $a_i$s can be reused to run the PIR procedure multiple times. Only steps 6-10 must be performed for each bit that the user wants to retrieve.

**Communication Complexity** Each of the $b_i$ and $s_i$ has length $O(\log N) = O(8l^2)$ bits. The user sends $\sqrt{n}$ $b_i$s for a total of $O(8l^2\sqrt{n})$ bits. The database sends back $\sqrt{n}$ $s_i$s for a total of $O(8l^2\sqrt{n})$ bits. So, in total $O(16l^2 n^{1/2})$ are exchanged. If we set $l = n^{1/k}$, this gives a protocol with communication complexity $O(n^{(k+4)/(2k)})$ which is sublinear for any $k > 4$.

**Computation** The computation in this *PIR* protocol can be broken up into four parts: preprocessing, query by user, response by database, interpretation of response by user. The preprocessing part can be reused to retrieve multiple bits, the other three parts must be done for each query. All operations done modulo $N$.

– **Preprocessing (steps 1-5):** The user randomly chooses $h, \alpha$, this takes generating $O(\log \sqrt{N} + 2l)$ random bits. The user samples $m$ integers of length $O(\log \sqrt{N})$ and decimals of length $l$. So, in total the user generates $O(ml^2)$ random bits. The user then calculates the $a_i$, taking $O(m)$ multiplications of $O(l^2)$-bit numbers.
– **Query by User (steps 6-8):** For each query the user chooses $m\sqrt{n}$ random bits, to indicate the membership of each $a_i$ in each $b_i$. Then the user computes $O(m\sqrt{n})$ modular additions of $O(l^2)$-bit numbers to generate the $b_i$.
– **Computing Response by DB (step 9):** The database performs $O(n)$ modular additions of $O(\log N)$-bit numbers.
– **Decoding Response by User (step 10):** The user performs 1 multiplication of $O(l^2)$-bit numbers
– **Totals:** Preprocessing takes $O(n^{2/k})$ multiplications, but can be done once for many PIR executions. The total per query computation that needs to be done is as follows. The user computes $O(n^{(k+4)/(2k)})$ modular additions and 1 multiplication. The database computes $O(n)$ additions. All operations are done on $O(n^{2/k})$-bit numbers.

**Correctness** The probability of retrieving the wrong value of $x_{i_0,j_0}$ is at most $2^{-\omega(\log n)}$.
We show that if $x_{i_0,j_0} = 0$ then $s_{i_0}$ is close to a multiple of $d$ and if $x_{i_0,j_0} = 1$ then $s_{i_0}$ is far from a multiple of $d$. For the case where $x_{i_0,j_0} = 0$ observe that each $a_i$ is close to a multiple of $d$ with some small standard deviation. We show that with high probability even the sum of $m\sqrt{n}$ $a_i$s stays close to a multiple of $d$. The case for $x_{i_0,j_0} = 1$ is similar. For a detailed proof see Appendix A.

**Security** The proof of security can be broken down into 3 steps.

1. By a proof identical to the one in [7] we can show that if the database can with non-negligible probability distinguish between queries for bits $i$ and $i'$ then it can distinguish between encryptions of 0 and 1 of Regev's cryptosystem [11].
2. In Lemma 5.4 of [11] Regev shows that if there exists such a distinguisher then there exists a distinguisher that can distinguish between $U$ and $T_{h,\alpha}$ with non-negligible probability for $h, \alpha$ chosen as in our protocol.
3. In Theorem 4.5 of [11], Regev shows that this implies an efficient algorithm that solves $\sqrt{l} \cdot \gamma(l)$-uSVP. For our settings of $l, \gamma$ this implies an efficient algorithm for $l^{(k+6)/(4)} \log l$-uSVP on a lattice of dimension $l = n^{1/k}$

### 3.2 Recursive Scheme

**Theorem 2.** *Let $k \in \mathbb{Z}, k > 4$, $n >> k$ is the size of the database. There exists a cPIR with the following properties.*
*a) $O(n^{\frac{3\sqrt{k}+2}{k+\sqrt{k}}}) = O(n^{\epsilon})$ communication complexity.*
*b) The user does $O(n^{2/k})$ multiplications in preprocessing. The user then does $\sqrt{k}n^{\frac{(k+2\sqrt{k}+2)}{(k\sqrt{k}+k)}}$ additions and $\sqrt{k}$ multiplications per query, while the database performs $n(k\sqrt{k}+1)$ additions per query. All operations are done on $O(n^{2/k})$-bit numbers.*
*c) Probability of error less than $2^{-\omega(\log n)}$*
*d) Security guaranteed by the hardness of $l^{(k+6)/4} \log l$-uSVP on a lattice of dimension $l = n^{1/k}$.*

*Proof.* We describe the protocol and then prove it has the properties we claim.

**Intuition** The following protocol is a modification of the basic protocol presented earlier. Note that in the basic protocol the database sends to the user $\sqrt{n}$ strings of length $k$ of which the user needs only 1. Instead of sending all these strings, the user and database recursively perform cPIR on them to retrieve the needed bits.

**Definitions**

1. $n, k, l, m$ - defined as in the basic scheme
2. $L$ - integer, depth of the recursion.
3. $t = n^{1/(L+1)}$
4. $K = \log N = 8l^2 = 8n^{2/k}$

**Protocol**

1. Perform steps 1-5 of the original scheme to generate $m$ numbers $a_i$ replacing the $\sqrt{n}$ with $t$ in the definition of the set $H$ in step 2.
2. The user views the DB as an $s$ by $t$ array($s$ rows, $t$ columns)
3. The user performs steps 6-8 from the original scheme replacing the $\sqrt{n}$ with $t$.
4. The DB performs the calculation in step 9 of the original scheme, but does not send back any of the $s$ generated strings. Each string is of length $K$ bits. View these strings instead as $K$ strings of length $s$ where one string has the first bit of all $s$ strings, one has the second, and so on.
5. The user wants one bit out of each of these strings of length $s = \frac{n}{t} = n^{L/(L+1)}$(the bit corresponding to the desired row). He will retrieve this bit by using PIR on each of these strings. Note that since the user wants the same bit out of each of these strings he only needs to generate one query which he can use on all the strings.
   (a) The user and DB recursively repeat steps 3 and 4, $L$ times. In level $j$ of the recursion
      i. The user sends $t$ numbers $b_i$ of length $K$ as in the original scheme.
      ii. The DB uses these numbers to perform $K$ instances of PIR on strings of length $s_j = n^{(L-(j-1))/(L+1)}$, splitting each of these strings into $K$ strings as in step 4.
   (b) At the bottom level of the recursion the DB sends all $K^L$($K$ strings are generated at each level for each string at the previous level) strings that it is holding, of length $s_L = n^{1/(L+1)}$ each, back to the user.
   (c) The user views these strings as $n^{1/(L+1)}K^{(L-1)}$ strings of length $K$ and uses them to reconstruct the answers at each level of the recursion by performing step 10 of the original scheme on the appropriate strings.

**Note:** The user does not require response from the server and can ask all the queries in one round.

**Communication Complexity** At each level of the recursion the user sends $t$ strings of length $K$. Since there are a total of $L$ levels of the recursion this is a total of $n^{1/(L+1)} \cdot (KL)$ bits. The database sends back $K^L$ strings of length $n^{1/(L+1)}$, for a total of $n^{1/(L+1)} \cdot K^L$. Therefore, the total communication complexity is $n^{1/(L+1)} \cdot (K^L + KL)$ bits. Plugging in for $K$ we get

$$CC = n^{1/(L+1)} \cdot (8^L n^{2L/k} + 8n^{2/k}L) = O(n^{\frac{k+2L^2+2L}{kL+k}})$$

If $l, k, L << n$. So, if we set $L = \lfloor \sqrt{k} \rfloor = O(\sqrt{k}) = O(\sqrt{\frac{\log n}{\log K}})$, as in [7]. We get

$$CC = O(n^{\frac{3k+2\sqrt{k}}{k^{3/2}+k}}) = O(n^{\frac{3\sqrt{k}+2}{k+\sqrt{k}}}) = O(n^{\epsilon}) \qquad 0 < \epsilon < 1$$

**Computation** As in the non-recursive scheme, the computation can be broken up into four parts.

- **Preprocessing:** The preprocessing is exactly the same as in the basic scheme since we can reuse the generated $a_i$s.
- **Query by User:** For each level of the recursion the user must generate $t = n^{1/(L+1)}$ $b_i$s. This requires the user to flip $mt$ coins and compute $mt$ modular additions of $O(l^2)$-bit integers.

– **Response by database:** At the top level of the recursion the database computes $n$ additions. Then, at each level $i$ below the top level it computes $ks_i = kn^{\frac{\sqrt{k}-(i-1)}{\sqrt{k}+1}}$ additions. So the total number of additions the database has to do is $n + k\sum_{i=1}^{\sqrt{k}} n^{\frac{\sqrt{k}-(i-1)}{\sqrt{k}+1}} < n(k\sqrt{k}+1)$. All additions are done on $O(l^2)$-bit numbers.

– **Decoding response by user:** The user computes 1 modular multiplication of $O(l^2)$-bit numbers per level of recursion for a total of $\sqrt{k}$ multiplications.

– **Totals:** The preprocessing takes $O(n^{2/k})$ multiplications. Per query, the user computes $\sqrt{k}n^{\frac{(k+2\sqrt{k}+2)}{(k\sqrt{k}+k)}}$ modular additions and $\sqrt{k}$ multiplications and the database computes less than $n(k\sqrt{k}+1)$ additions. All operations are done on $O(n^{2/k})$-bit numbers.

**Correctness** The recursive protocol decodes a bit incorrectly if there is an error at any level of the recursion. Therefore, by the union bound, $Pr[error] \leq \sum_{i=1}^{L} Pr[\text{error on level } i]$. Since, by the same analysis as in the non-recursive scheme, $Pr[\text{error on level } i] < 2^{-\omega(\log n)}$, $Pr[\text{error}] < (\sqrt{k}+1)2^{-\omega(\log n)} < 2^{-\omega(\log n)}$.

**Security** The proof of security of the recursive scheme is the same as in [7], except that it is based on the hardness of distinguishing encryptions of 0 and 1 in Regev's cryptosystem [11] which reduces to solving the uSVP.

## 4 SVP, SIVP Based Protocol

### 4.1 Basic Scheme

**Theorem 3.** *Let $k \in \mathbb{Z}, k > 1$, $n >> k$ is the size of the database. There exists a cPIR protocol with the following properties.*

1. *$\widetilde{O}(n^{(k+1)/(2k)})$ communication complexity*
2. *For $O(n^{1/k})$-bit numbers the user does $\widetilde{O}(n^{1/k})$ multiplications in preprocessing and $\widetilde{O}(n^{1/k})$ additions per query.*
   *For $O(\log n)$-bit numbers the user does $\widetilde{O}(n^{2/k})$ multiplications and $\widetilde{O}(n^{1/k})$ additions in preprocessing and $\widetilde{O}(n^{2/k})$ additions and $\widetilde{O}(n^{1/k})$ products per query.*
   *The database does $O(n)$ additions of $O(n^{1/k})$-bit numbers and $O(n^{(k+1)/k})$ additions of $O(\log n)$-bit numbers per query.*
3. *Probability of error less than $2^{-\omega(\log n)}$*
4. *Security guaranteed by the worst case quantum hardness of the following two problems on lattice $L$ of dimension $l = n^{(1/k)}$*
   – *Find a set of $l$ linearly independent lattice vectors of length $\leq \widetilde{O}(\lambda_l(L) \cdot l^{(k+8)/4})$*
   – *Approximate $\lambda_1(L)$ to within $\widetilde{O}(l^{(k+8)/4})$*

*Proof.* We describe the cPIR protocol and then prove that it has the properties we claim.

**Intuition** This protocol works the same way as the one from Section 3.1. It replaces the cryptosystem from [11] with the one from [12].

**Definitions**

1. $n, k, l, \Psi_\alpha$ - defined as before
2. $p$ - prime integer s.t. $l^{(k/4)+2} \leq p \leq 2l^{(k/4)+2}$
3. $m$ - integer s.t. $m \geq 5(l+1)\log p \geq 5(l+1)(\frac{k}{4}+2)\log l$
4. $\alpha$ - real number s.t.

$$\alpha = o\left(\frac{1}{n^{(1/4)}\sqrt{m}\sqrt{\log l}}\right) = o\left(\frac{1}{n^{(1/4)}\sqrt{l}\log l}\right) \qquad \alpha = \omega\left(\frac{1}{n^{(1/4)}l\sqrt{l}}\right)$$

In practice we can choose $\alpha = \left(\frac{1}{n^{(1/4)}l}\right)$

**Protocol**

1. The database is a $\sqrt{n} \times \sqrt{n}$ array of bits $\{x_{i,j}\}$. User has $i_0, j_0$ and wants to obtain $x_{i_0,j_0}$.
2. The user chooses $l$ dimensional vector $\mathbf{s} \in \mathbb{Z}_p^l$ uniformly at random.
3. The user chooses $m$ vectors $\{\mathbf{a_1}, \ldots, \mathbf{a_m}\} \in \mathbb{Z}_p^l$ uniformly at random.
4. The user chooses $e_1, \ldots, e_m \in \mathbb{R}$ according to rounded distribution $\overline{\Psi_\alpha}$as follows
   (a) Choose $e'_1, \ldots, e'_m \in \mathbb{T}$ according to $\Psi_\alpha$
   (b) Output $e_i = e'_i * p$ rounding to precision $2^l$ (l bits after the decimal point)
5. The user computes $b_i =< \mathbf{a_i}, \mathbf{s} > +e_i$.
6. For each $j \in \{[\sqrt{n}] : j \neq j_0\}$
   (a) User chooses a random subset $S \subseteq [m]$
   (b) User computes $\mathbf{A_j} = \sum_{i \in S} \mathbf{a_i}$, $B_j = \sum_{i \in S} b_i$
7. For $j = j_0$
   (a) User chooses a random subset $S \subseteq [m]$
   (b) User computes $\mathbf{A_j} = \sum_{i \in S} \mathbf{a_i}$, $B_j = \lfloor \frac{p}{2} \rfloor + \sum_{i \in S} b_i$
8. The user sends $((\mathbf{A_1}, B_1), \ldots, (\mathbf{A_{\sqrt{n}}}, B_{\sqrt{n}}))$ to the server.
9. For each row $i$, the server does the following.
   (a) Compute $\alpha_\mathbf{i} = \sum_{j=1}^{\sqrt{n}} (\mathbf{A_j}) x_{i,j}$, $\beta_i = \sum_{j=1}^{\sqrt{n}} (B_j) x_{i,j}$
   (b) Send back $((\alpha_\mathbf{1}, \beta_1), \ldots, (\alpha_{\sqrt{\mathbf{n}}}, \beta_{\sqrt{n}}))$
10. The user computes $w = \beta_{i_0} - < \alpha_{\mathbf{i_0}}, \mathbf{s} >$.
    – If $w$ is closer to 0 than to $\lfloor \frac{p}{2} \rfloor$ modulo $p$, then $x_{i_0,j_0} = 0$, otherwise $x_{i_0,j_0} = 1$

**Communication Complexity** The user sends $\sqrt{n}$ pairs $(A_i, B_i)$. Each $A_i$ consists of $l$ numbers in $\mathbb{Z}_p$, so it takes $l \log p$ bits. Each $B_i$ takes $\log p$ bits to represent the integer part and $l$ bits for the fractional part. So, in total, the user sends $\sqrt{n}((l+1)\log p + l) = n^{(1/2)}((n^{(1/k)}+1)(1+\frac{k+8}{4k}\log n) + n^{(1/k)}) = \widetilde{O}(n^{(k+1)/(2k)}) = \widetilde{O}(n^{(1/2)+\epsilon})$ bits. The database also sends the same number of bits in the answer. So the total communication complexity of this protocol is $\widetilde{O}(n^{(1/2)+\epsilon})$.

**Computation** The computation in this $PIR$ protocol can be broken up into four parts: preprocessing, query by user, response by database, interpretation of response by user. In the analysis below we use $S_l = l + \log p$, $S_s = \log p$. All operations are done modulo $p$.

- **Preprocessing (steps 1-5):** The user generates $(m+1)l \log p$ random bits and samples $m$ $S_l$-bit values from the normal distribution. Next, he computes $m$ multiplications of $S_l$-bit numbers to generate the $e_i$, and $ml$ multiplications and $m$ additions on $S_s$-bit numbers to make the $b_i$s.
- **Query by User (steps 6-8):** The user first chooses $m$ random bits. Then he computes $O(lm)$ additions of $S_s$-bit numbers and $O(m)$ additions of $S_l$-bit numbers.
- **Computing Response by DB (step 9):** The database performs $\sqrt{n}l$ additions of $S_s$-bit numbers and $O(\sqrt{n})$ additions of $S_l$-bit numbers per row, for a total of $O(nl)$ additions of $S_s$-bit numbers and $O(n)$ additions of $S_l$-bit numbers over all the rows.
- **Decoding Response by User (step 10):** The user computes $l$ products of $S_s$-bit numbers and 1 addition of $S_l$-bit numbers.
- **Totals:** In preprocessing, the user generates $\widetilde{O}(n^{2/k})$ random bit, computes $\widetilde{O}(n^{1/k})$ multiplications of $O(l)$-bit numbers and $\widetilde{O}(n^{2/k})$ multiplications and $\widetilde{O}(n^{1/k})$ additions of $O(\log n)$-bit numbers. Per query, the user computes $\widetilde{O}(n^{2/k})$ additions and $\widetilde{O}(n^{1/k})$ products on $O(\log n)$-bit numbers and $\widetilde{O}(n^{1/k})$ additions of $O(n^{1/k})$-bit numbers. The database computes $O(n^{(k+1)/k})$ additions of $O(\log n)$-bit numbers and $O(n)$ additions of $O(n^{1/k})$-bit numbers.

**Correctness** The probability of retrieving the wrong value of $x_{i_0,j_0}$ is at most $2^{-\omega(\log n)}$.
We show that if $x_{i_0,j_0} = 0$ then $w = \sum e_i$ where the $e_i$ are from a normal distribution with small standard deviation such that with high probability the sum of $m\sqrt{n}\ e_i$ is still close to 0. If $x_{i_0,j_0} = 1$ then $w = \sum e_i + \lfloor \frac{p}{2} \rfloor$ is close to $\lfloor \frac{p}{2} \rfloor$ for the same reason. For a detailed proof see Appendix B

**Security** The proof of security can be broken down into 4 steps.

1. By a proof identical to the one in [7] we can show that if the database can with non-negligible probability distinguish between queries for bits $i$ and $i'$ then it can distinguish between encryptions of 0 and 1 of Regev's cryptosystem [12].
2. In Lemma 5.4 of [12] Regev shows that for $m \geq 5(l+1)\log p$, as in our protocol, if there exists a polynomial time algorithm $W$ that distinguishes between encryptions of 0 and 1 in his cryptosystem then there exists a distinguisher $Z$ that distinguishes between $A_{\mathbf{s},\overline{\Psi_\alpha}}$ and $U$ for a non-negligible fraction of all possible $\mathbf{s}$. Here $U$ is the uniform distribution and $A_{\mathbf{s},\overline{\Psi_\alpha}}$ is the distribution over all possible values of $\mathbf{s}$ and all values $e_i \in \overline{\Psi_\alpha}$ as chosen in the protocol.
3. In Section 4, Regev shows that this implies an efficient algorithm that solves $LWE_{p,\Psi_\alpha}$, which is the Learning With Errors Problem as defined by Regev.
4. Finally, in Theorem 3.1, Regev shows that since

$$\alpha p \geq \omega\left(\frac{l^{(k/4)+2}}{l^{(k/4)}l}\right) = \omega(l) > 2\sqrt{l} \qquad \text{for } l > 4$$

   this implies a polynomial time quantum algorithm for solving the following worst-case lattice problems on a lattice of dimension $l$.
   - Find a set of $l$ linearly independent lattice vectors of length $\leq \widetilde{O}\left(\frac{\lambda_l(L)\cdot l}{\alpha}\right) = \widetilde{O}(\lambda_l(L)\cdot l^{(k+8)/4})$
   - Approximate $\lambda_1(L)$ to within $\widetilde{O}(l/\alpha) = \widetilde{O}(l^{(k+8)/4})$

## 4.2 Recursive Scheme

**Theorem 4.** *Let $k \in \mathbb{Z}, k > 1$, $n >> k$ is the size of the database. There exists a cPIR protocol with the following properties.*

1. $\widetilde{O}(n^{\frac{2\sqrt{k}+1}{k+\sqrt{k}}})$ *communication complexity*
2. *For $O(n^{1/k})$-bit numbers the user does $\widetilde{O}(n^{1/k})$ multiplications in preprocessing and $\widetilde{O}(n^{\frac{k+\sqrt{k}+1}{k\sqrt{k}+k}})$ additions per query.*
   *For $O(\log n)$-bit numbers the user does $\widetilde{O}(n^{2/k})$ multiplications and $\widetilde{O}(n^{1/k})$ additions in preprocessing and $\widetilde{O}(n^{\frac{k+2\sqrt{k}+2}{k\sqrt{k}+k}})$ additions and $\widetilde{O}(n^{1/k})$ products per query.*
   *The database does $O(n)$ additions of $O(n^{1/k})$-bit numbers and $O(n^{(k+1)/k})$ additions of $O(\log n)$-bit numbers per query.*
3. *Probability of error less than $2^{-\omega(\log n)}$*
4. *Security guaranteed by the worst case quantum hardness of the following two lattice problems on lattice $L$ of dimension $l = n^{(1/k)}$*
   - *Find a set of $l$ linearly independent lattice vectors of length $\leq \widetilde{O}(\lambda_l(L) \cdot l^{(k+8)/4})$*
   - *Approximate $\lambda_1(L)$ to within $\widetilde{O}(l^{(k+8)/4})$*

*Proof.* We describe the cPIR protocol and then prove that it has the properties we claim.

**Intuition** The following protocol is the protocol from Section 3.2 with the cryptosystem from [12] instead of the one from [11].

### Definitions

1. $L$ - integer depth of the recursion.
2. $t = n^{1/(L+1)}$
3. $K = (l+1)\log p + l$
4. $k, l, m, n$ - same as defined in non-recursive scheme

### Protocol

1. Perform steps 1-5 of the original scheme to generate the $\mathbf{a_i}$ and $b_i$.
2. The user views the DB as an $s$ by $t$ array($s$ rows, $t$ columns)
3. The user performs steps 6-8 of the original scheme replacing the $\sqrt{n}$ with $t$.
4. The DB performs the calculation in step 9 of the original scheme, but does not send back any of the $s$ generated pairs. Note, the DB now has $s$ pairs $(\alpha_{\mathbf{i}}, \beta_i)$. We view these as strings of length $K$ bits. View these instead as $K$ strings of length $s$ where one string has the first bit of all $s$ strings, one has the second, and so on.
5. The user wants one bit out of each of these strings of length $s = \frac{n}{t} = n^{L/(L+1)}$(the bit corresponding to the desired row). He will retrieve this bit by using PIR on each of these strings. Note that since the user wants the same bit out of each of these strings he only needs to generate one query which he can use on all the strings.
   (a) The user and the database recursively repeat steps 3 and 4, $L$ times. In level $j$,
      i. The user sends $t$ pairs $(\mathbf{A_i}, B_i)$ of total length $K$ as in the original scheme.

ii. The DB uses these numbers to perform $K$ instances of PIR on strings of length $s_j = n^{(L-(j-1))/(L+1)}$, splitting each of these strings into $K$ strings as in step 4.

(b) At the bottom level of the recursion the DB sends all $K^L$(at each level each string is split into $K$) strings that it is holding, of length $s_L = n^{1/(L+1)}$ each, back to the user.

(c) The user views these strings as $n^{1/(L+1)}K^{(L-1)}$ pairs $(\mathbf{A_i}, B_i)$ and uses these to reconstruct the answers at each level of the recursion by performing step 10 of the original scheme on the appropriate pairs.

**Communication Complexity** At each level of the recursion, the user sends $t$ strings of length $K$. Since there are $L$ levels of recursion, this amounts to a total of $n^{1/(L+1)} \cdot (KL)$ bits. The database sends back $K^L$ strings of length $n^{1/(L+1)}$, for a total of $n^{1/(L+1)} \cdot K^L$ bits. Therefore, the total communication complexity is $n^{1/(L+1)} \cdot (K^L + KL)$ bits. Plugging in for $K$ we get

$$CC = n^{1/(L+1)} \cdot (((l+1)\log p + l)^L + ((l+1)\log p + l)L) = O(n^{\frac{L^2+L+k}{Lk+k}} \log^L n)$$

If $k, l, L << n$. So, if we set $L = \lfloor \sqrt{k} \rfloor = O(\sqrt{k}) = O(\sqrt{\frac{\log n}{\log K}})$ as in [7], we get

$$CC = O(n^{\frac{2k+\sqrt{k}}{k^{(3/2)}+k}} \log^{\sqrt{k}} n) = \widetilde{O}(n^{\frac{2\sqrt{k}+1}{k+\sqrt{k}}}) = \widetilde{O}(n^{\epsilon}) \qquad 0 < \epsilon < 1$$

**Computation** Just like in the non-recursive scheme, the computation is in four parts. $S_l = l + \log p$, $S_s = \log p$. All operations are done modulo $p$

- **Preprocessing:** The preprocessing stage is the same in as in the non-recursive scheme.
- **Query by User (steps 10-12):** For each level of the recursion, the user generates $t$ pairs $(\mathbf{A_i}, B_i)$ requiring choosing $mt$ random coins and then performing $O(lmt)$ modular additions of $S_s$-bit integers and $O(mt)$ additions of $S_l$-bit numbers. Over all levels of recursion this adds up to $Lmt$ random coins and $Llmt$ additions of $S_s$-bit integers and $Lmt$ additions of $S_l$-bit numbers.
- **Computing Response by DB (step 13):** At the top level of the recursion the DB computes $O(ln)$ additions of $S_s$-bit numbers and $O(n)$ additions of $S_l$-bit numbers. At each level $i$ below this it performs $k$ PIRs for a total of $O(kls_i)$ additions of $S_s$-bit numbers and $O(ks_i)$ additions of $S_l$-bit integers. Over all the levels of recursion this is $O(ln) + kl\sum_{i=1}^{L} O(n^{\frac{\sqrt{k}-(i-1)}{\sqrt{k}+1}}) < O(ln(kL + 1))$ additions of $S_s$-bit numbers and $O(n(kL+1))$ additions of $S_l$-bit numbers.
- **Decoding Response by User (step 14):** For each level of the recursion, the user computes $l$ products of $S_s$-bit numbers and 1 addition of $S_l$-bit numbers. Over all the levels of recursion this adds up to $Ll$ products and $L$ additions.
- **Totals:** The preprocessing is the same as in the non-recursive scheme in Section 4. Per query the user computes $\widetilde{O}(n^{\frac{k+2\sqrt{k}+2}{k\sqrt{k}+k}})$ additions and $O(n^{1/k})$ products of $S_s$-bit integers and $\widetilde{O}(n^{\frac{k+\sqrt{k}+1}{k\sqrt{k}+k}})$ additions of $S_l$-bit numbers. The DB does $O(n^{(k+1)/k})$ additions of $S_s$-bit integers and $O(n)$ additions of $S_l$-bit numbers.

**Correctness** The recursive protocol succeeds only if every level of the recursion succeeds. Therefore, by the union bound, $Pr[error] \leq \sum_{i=0}^{L} Pr[\text{error on level } i] < (\sqrt{k}+1)2^{-\omega(\log n)} = 2^{-\omega(\log n)}$

**Security** The proof of security is a modification of the proof for the non-recursive protocol and is identical to the proof in [7] substituting the hardness of distinguishing between encryptions of 0 and 1 in Regev's cryptosystem [12] instead of the quadratic residue assumption.

## 5 Comparison Of Protocols

We compare the recursive versions of the two protocols with the protocol from [7]. In the protocol from Section 4.2 we only consider the operations on the longer real numbers as these dominate the running time. Thus, all the operations listed below are on $(n^{1/k})$-bit numbers.

| | Section 3.2 | Section 4.2 | Kushilevitz, Ostrovsky [7] |
|---|---|---|---|
| Comm. Complexity | $O(n^{\frac{3\sqrt{k}+2}{k+\sqrt{k}}})$ | $O(n^{\frac{2\sqrt{k}+1}{k+\sqrt{k}}})$ | $O(n^{\frac{2\sqrt{k}+1}{k+\sqrt{k}}})$ |
| Preprocessing | $O(n^{2/k})$ mults | $\widetilde{O}(n^{1/k})$ mults | $\widetilde{O}(1)$ mults |
| User Per Query | $O(1)$ mults<br>$O(n^{\frac{(k+2\sqrt{k}+2)}{(k\sqrt{k}+k)}})$ adds | 0 mults<br>$\widetilde{O}(n^{\frac{k+\sqrt{k}+1}{k\sqrt{k}+k}})$ adds | $\widetilde{O}(n^{1/(\sqrt{k}+1)})$ mults<br>0 adds |
| DB Per Query | $O(n(k\sqrt{k}+1)) = O(n)$ adds | $O(n)$ adds | $O(n)$ mults |
| Security based on | $n^{(k+6)/(4k)}$-uSVP | $\text{SVP}_{n^{(k+8)/(4k)}}$ | QRA |

Our second protocol has the same communication complexity as the original Kushilevitz, Ostrovsky protocol. The work the user and database do per query is significantly decreased by our protocols. Our first protocol has slightly worse complexity as a function of $k$, but still achieves the reduction in the server computation and shows a solution based on a different hardness assumption.

## 6 Conclusion

In this paper, we presented two $O(n^\epsilon)$ communication complexity cPIR protocols based on worst-case hardness of certain lattice problems. Both of these protocols only need the server to perform $O(n)$ modular additions and no multiplication. This is a significant improvement over all previous cPIR protocols, which have required $O(n)$ modular multiplications by the server per query.

The main shortcoming of the protocols in this paper is that the database has to be very large. In order to decrease the communication and computation complexity we must make $k$ quite large. However, the dimension of the lattice on which all the hardness assumptions are based is $n^{1/k}$, so if $k$ is very large, this lattice becomes very low dimensional in which case the hardness assumptions may not be true. So for the dimension of the lattice to be big enough to provide security, the database must be very large.

## 7 Open Problems

One open problem suggested by this paper is to reduce the size of the database without losing the security. To this end it may be possible to restructure the current scheme or to use another cryptosystem, possibly based on other assumptions. Another open problem is to create a computationally efficient *polylog*$(n)$ communication complexity protocol based on the scheme from [3].

## 8 Acknowledgements

We would like to thank Radu Sion for suggesting this problem, Yuval Ishai for suggesting Regev's cryptosystem, and Oded Regev for his helpful communications.

## References

1. Ajtai, M. Generating hard instances of lattice problems (extended abstract). In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (New York, NY, USA, 1996), ACM Press, pp. 99–108.
2. Ajtai, M., and Dwork, C. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (New York, NY, USA, 1997), ACM Press, pp. 284–293.
3. Cachin, C., Micali, S., and Stadler, M. Computationally private information retrieval with polylogarithmic communication. *Lecture Notes in Computer Science 1592* (1999), 402–??
4. Chor, B., Goldreich, O., Kushilevitz, E., and Sudan, M. Private information retrieval. In *FOCS '95: Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)* (Washington, DC, USA, 1995), IEEE Computer Society, p. 41.
5. Gasarch, W. PIR website. http://www.cs.umd.edu/ gasarch/pir/pir.html.
6. Gasarch, W. A survey on private information retrieval, 2004.
7. Kushilevitz, E., and Ostrovsky, R. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *IEEE Symposium on Foundations of Computer Science* (1997), pp. 364–373.
8. Mann, E. Private access to distributed information, 1998.
9. Micciancio, D. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing 30*, 6 (Mar. 2001), 2008–2035. Preliminary version in FOCS 1998.
10. Micciancio, D., and Goldwasser, S. *Complexity of Lattice Problems: a cryptographic perspective*, vol. 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, Mar. 2002.
11. Regev, O. New lattice based cryptographic constructions. *Journal of the ACM 51*, 6 (2004), 899–942. Preliminary version in STOC'03.
12. Regev, O. On lattices, learning with errors, random linear codes, and cryptography. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing* (New York, NY, USA, 2005), ACM Press, pp. 84–93.
13. Sion, R., and Carbunar, B. On the practicality of private information retrieval. In *NDSS '07: Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS'07)* (2007).

## A Correctness of Section 3.1

*Claim.* The probability of retrieving the wrong value of $x_{i_0,j_0}$ using the protocol from Section 3.1 is at most $2^{-\omega(\log n)}$

*Proof.* Consider the case when $x_{i_0,j_0} = 0$. Let $R = \{j : x_{i_0,j} = 1\}$ and let $S_j$ be the subset of indices included in the sum used to generate $b_j$. Then
$w := \sum_{j\in R} b_j \pmod N = \sum_{j\in R}\sum_{i\in S_j} a_i \pmod N$ is the response from the server following our protocol. Note that $v := \sum_{j\in R}\sum_{i\in S_j} a_i \le m\sqrt{n}\cdot N$. We want to show that $w$ is near a multiple of $d$. Let $c$ be such that $cd$ is the nearest multiple of $d$ to $w$, $w = cd + r$. Since $cd \equiv qN + cd \pmod N$ we have that $v = qN + cd + r$ where $q \le m\sqrt{n}$. Clearly $d\lfloor h\rceil$ is the nearest multiple of $d$ to $N$. Let $c_2$ be such that $c_2 d$ is the nearest multiple of $d$ to $v$. Then

$$||w - cd| - |v - c_2 d|| \le q|N - d\lfloor h\rceil| \le m\sqrt{n}|N - d\lfloor h\rceil| = m\sqrt{n}\cdot d\cdot frc(h) < \frac{d}{16}$$

Therefore,

$$|w - cd| < |v - c_2 d| + \frac{d}{16}$$

Note that $frc\left(\frac{w}{d}\right) = \frac{|w-cd|}{d}$. Therefore,

$$frc\left(\frac{w}{d}\right) < \frac{1}{16} + frc\left(\frac{v}{d}\right) < \frac{1}{16} + \frac{m\sqrt{n}}{d} + frc\left(\frac{N}{d}\sum_{j\in R}\sum_{i\in S_j}\left(\frac{x_i+y_i}{h}\right)\right)$$
$$= \frac{1}{16} + \frac{m\sqrt{n}}{d} + frc\left(\sum_{j\in R}\sum_{i\in S_j}(x_i+y_i)\right)$$

The second inequality is true because for any positive real number $x$, $x - \lfloor x\rfloor < 1$. Since, $x_i$ is an integer and so $frc(x_i) = 0$, we can take it out of the sum, getting

$$frc\left(\frac{w}{d}\right) < \frac{1}{16} + \frac{m\sqrt{n}}{d} + frc\left(\sum_{j\in R}\sum_{i\in S_j} y_i\right) < \frac{1}{8} + frc\left(\sum_{j\in R}\sum_{i\in S_j} y_i\right)$$

The last inequality is true because $d = O(\sqrt{N}) = 2^{O(n)} >> m\sqrt{n}$.
With probability exponentially close to 1, $x_i < \lceil h\rceil - 1$. If this is the case, the distribution of $y_i$ is $\Psi_\alpha$ and the distribution of $\sum_{j\in R}\sum_{i\in S_j} y_i$ is $\Psi_{|S|\alpha}$ where $|S|\alpha \leq \sqrt{m}n^{(1/4)}\cdot\alpha = O\left(\frac{\sqrt{m}n^{(1/4)}}{\gamma(l)}\right)$ by the properties of a normal distribution.

Note that we are actually slightly off from this because we rounded the values $y_i$ to values $\overline{y_i}$ of precision $2^{-l}$, but we can show that $|\sum_{j\in R}\sum_{i\in S_j}\overline{y_i} - \sum_{j\in R}\sum_{i\in S_j} y_i| \leq \frac{m\sqrt{n}}{2^l} < 1/32$ for large enough $l$, so it is enough to show that $|\sum_{j\in R}\sum_{i\in S_j} y_i| < 1/16$

By Claim 5.1 from [11], $Pr[frc\left(\sum_{j\in R}\sum_{i\in S_j} y_i\right) > \frac{1}{16}] \leq 2^{-\Omega(\frac{(\gamma(l))^2}{m\sqrt{n}})}$
so with probability greater than $1 - 2^{-\Omega(\frac{(\gamma(l))^2}{m\sqrt{n}})}$, $frc\left(\frac{w}{d}\right) < \frac{1}{8} + \frac{1}{16} + \frac{1}{32} < \frac{1}{4}$.

If $x_{i_0,j_0} = 1$ then $w$ is the same as in the case when $x_{i_0,j_0} = 0$ plus $\lfloor\frac{a_{i'}}{2}\rfloor$. Since, $x_{i'}$ is odd and with probability exponentially close to 1, $frc(y_{i'}) < \frac{1}{16}$ we have $frc\left(\frac{\lfloor a_{i'}/2\rfloor}{d}\right) > \frac{1}{2} - \frac{1}{32} - \frac{1}{d}$. Since, for an encryption of 0, $frc\left(\frac{w}{d}\right) < \frac{1}{8} + \frac{1}{16}$, for an encryption of 1,

$$frc\left(\frac{w}{d}\right) > frc\left(\frac{\lfloor a_{i'}/2\rfloor}{d}\right) - \frac{1}{8} - \frac{1}{16} > \frac{1}{4}$$

So, if we take $\gamma(l) = \omega(l^{(k+4)/4}\sqrt{\log l})$ We get

$$Pr[error] \leq 2^{-\Omega(\frac{(\gamma(l))^2}{m\sqrt{n}})} = 2^{-\omega(\log n)} \quad ■$$

## B Correctness Of Section 4.1

*Claim.* The probability of retrieving the wrong value of $x_{i_0,j_0}$ using the protocol from Section 4.1 is at most $2^{-\omega(\log n)}$

*Proof.* Let $R = \{j : x_{i_0,j} = 1\}$, $S_j$ be the subset of indices included in the sum to calculate $A_j, B_j$. Consider the case when $x_{i_0,j_0} = 0$. Then

$$\alpha_{\mathbf{i}} = \sum_{j\in R}\sum_{i\in S_j} \mathbf{a_i}$$
$$\beta_i = \sum_{j\in R}\sum_{i\in S_j} b_i = \sum_{j\in R}\sum_{i\in S_j} < \mathbf{a_i}, \mathbf{s} > + \sum_{j\in R}\sum_{i\in S_j} e_i$$
$$w = \beta_i - < \alpha_{\mathbf{i}}, \mathbf{s} > = \sum_{j\in R}\sum_{i\in S_j} e_i.$$

We now show that $|e| = |\sum_{j\in R}\sum_{i\in S_j} e_i| < \lfloor \frac{p}{2} \rfloor/2$ with high probability, where $|e|$ denotes the distance of $e$ to 0 modulo $p$.
Each $e_i$ is sampled from $\Psi_\alpha$, multiplied by $p$, and then rounded. Since, when we round, we only change the value by less than $2^{-l}$ we have

$$|\sum_{j\in R}\sum_{i\in S_j} e_i - \sum_{j\in R}\sum_{i\in S_j} px_i| \leq \frac{m\sqrt{n}}{2^l} < p/32$$

for large enough $l$. Hence, it is enough to show that

$$|\sum_{j\in R}\sum_{i\in S_j} px_i| < |\sum_{i=1}^{m\sqrt{n}} px_i| < p/16$$

This is equivalent to the condition that

$$|\sum_{i=1}^{m\sqrt{n}}| < 1/16$$

where the sum is performed modulo 1. Since $\sum_{i=1}^{m\sqrt{n}} x_i$ is distributed as $\Psi_{\sqrt{m}n^{(1/4)}\cdot\alpha}$ and $\sqrt{m}n^{(1/4)} \cdot \alpha = o(1/\sqrt{\log l})$, the probability that $|\sum_{i=1}^{k}| < 1/16$ is $1 - 2^{-\omega(\log l)} = 1 - 2^{-\omega(\log n)}$. The proof for the case when $x_{i_0,j_0} = 1$ is similar.