Bandwidth Efficient PIR from NTRU

Yarkın Doröz[1], Berk Sunar[1] and Ghaith Hammouri[2]

[1] Worcester Polytechnic Institute
[2] Crags Inc.

Abstract. We present a private information retrieval (PIR) scheme based on somewhat homomorphic encryption (SWHE). In particular, we customize an NTRU-based SWHE scheme in order to evaluate a specific class of fixed depth circuits relevant for PIR implementation, thus achieving a more practical implementation. In practice, a SWHE that can evaluate a depth 5 circuit is sufficient to construct a PIR capable of retrieving data from a database containing 4 billion rows. We leverage this property in order to produce a more practical PIR scheme. Compared to previous results, our implementation achieves a significantly lower bandwidth cost (more than 1000 times smaller). The computational cost of our implementation is higher than previous proposals for databases containing a small number of bits in each row. However, this cost is amortized as database rows become wider.

Keywords: Private information retrieval, homomorphic encryption, NTRU.

1 Introduction

The problem of Private Information Retrieval (PIR) is one of the simplest yet most useful concepts in cryptography. Simply put, a PIR scheme allows Alice to store a database D at a remote server (Bob) with the promise that Alice can retrieve $D(i)$ without revealing i or $D(i)$ to Bob. The notion of an information theoretic PIR scheme was first introduced in [22] where the limits on Bob's knowledge of i were based on information theoretic arguments. In such a setting, it can easily be shown that in a PIR scheme with a single server (single copy of the database D) the only way to hide access to D in the information theoretic setting is for Bob to send the entire database D back to Alice. Many solutions were proposed in order to produce a secure information theoretic PIR scheme when Alice can communicate with several servers storing a copy of D [22–24]. While these constructions are interesting from a theoretical point of view they are difficult to achieve in a practical setting. For the remainder of this text we focus only on single database PIRs.

As such Chor and Gilboa [25, 26] introduced the concept of computational PIRs (cPIR). In cPIR, Alice is content to have the difficulty of Bob retrieving i (or information about i) based on computational difficulty. That is Alice would like Bob to face a computationally *difficult* problem in order to extract any significant information about i or $D(i)$. Since the introduction of cPIR many

schemes have been proposed. In [27] Kushilevitz and Ostrovsky presented the first single server PIR scheme based on the computational difficulty of deciding the quadratic residuosity of a number modulo a product of two large primes.

Other cPIR constructions include [28] which is based on the computational difficulty of deciding whether a small prime p divides $\phi(m)$ for any composite integer m of unknown factorization where $\phi()$ denotes Euler's totient function. In [29] another cPIR scheme was presented that generalizes the scheme in [28] while using a slight variation on the security assumption. Most notably, the construction in [29] achieves a communication complexity of $O(k + d)$ where k is the security parameter satisfying $k > \log(N)$, N is the database size, and d is the bit-length of the retrieved data. In [7] Lipmaa presented a different yet quite interesting cPIR scheme that leverages a (length-flexible) additively homomorphic public-key encryption scheme and provides better communication complexity performance. Later in [3], an efficient PIR scheme is produced using a partially homomorphic encryption algorithm. This was later followed by a lattice based cPIR construction by Aguilar-Melchor and Gaborit [30].

In 2007 the computational practicality of PIRs was raised by Sion and Carbunar [20] who concluded that no existing construction is as efficient as the trivial PIR scheme.The authors observe that any computational PIR scheme that requires one or more modular multiplications per database bit cannot be as efficient as the trivial PIR scheme. Later, Olumofin and Goldberg [21] revisited the performance analysis and found that the lattice-based PIR scheme by Aguilar-Melchor and Gaborit [30] to be an order of magnitude more efficient than the trivial PIR in situations that reflect average consumer internet bandwidth.

In all these constructions the challenge has been to find an efficient scheme based on a difficult computational problem. The aforementioned schemes utilize a variety of approaches and a diverse set of tools to construct cPIR schemes. However, it has always been clear that given a fully or somewhat homomorphic encryption (SWHE or FHE) scheme achieving a cPIR construction would be conceptually as simple as carrying out normal information retrieval. Although fully homomorphic encryption schemes have been introduced in 2009 [1] efficiency has been the biggest hindrance preventing any practical implementation. As such, FHE schemes have yet to be shown to be useful in progressing a practical realization of a private information retrieval. However, a number of new FHE schemes [5, 10, 4, 6, 11] and optimizations such as modulus and key switching [8], batching and SIMD optimizations [9] have been introduced in just the last few years which improved the efficiency of FHE implementation roughly by two orders of magnitude per year.

Our Contribution. Motivated by these advances, in this work we take a first step towards using a SWHE scheme along with optimizations developed for leveled SWHE implementation to construct an efficient cPIR. We construct a rather simple implementation of a PIR scheme from a batched leveled SWHE implementation based on the NTRU encryption scheme. Our scheme has excellent bandwidth performance compare to previous implementations (more than 1000 times smaller). The computational cost of our implementation is higher than

previous proposals for databases containing a small number of bits in each row. However, this cost is amortized as the database rows become wider.

2 Homomorphic Encryption Based PIR Schemes

In this section we briefly survey 3 representative cPIR schemes constructed out of homomorphic encryption schemes most relevant to our proposal. We note that this survey is only intended to provide a basis for later comparison.

2.1 Kushilevitz-Ostrovsky PIR

At the essence of the K-O scheme [2] is the use of a secure homomorphic operations and the idea of conceptually storing the database as a matrix. To elaborate, we can think of Bob as having a database D of size 2^h with each location containing a single bit (this can easily be extended for longer strings). Bob then stores D in a matrix M of size $2^{h/2} \times 2^{h/2}$. For any location i in the database D, this process can be done by using the first $h/2$ bits of i to represent the number of the row in M and the last $h/2$ bits of i to represent the number of the column in M where i will be placed. Now for Alice to recover the entry $D(i)$, she will take the first $h/2$ bits of i encode them into a one hot encoding A and carry out the same process for the lower $h/2$ bits of i to produce B. Finally, Alice uses a partially homomorphic encryption scheme E to encrypt each bit in A and B. Thus Alice sends to Bob $(E(A_0) \dots E(A_{h/2-1}), E(B_0) \dots E(B_{h/2-1}))$. With this information Bob can now carryout some homomorphic operations between the encrypted bits sent by Alice and the data stored within the matrix M in order to produce an encrypted output which encodes the bit $D(i)$ and can then be sent to Alice for decryption. The matrix is of size $2^{h/2} \times 2^{h/2} = N$ and therefore the communication complexity becomes $\mathcal{O}(\sqrt{N})$.

2.2 Boneh-Goh-Nissim (BGN) Scheme

The BGN cryptosystem is a partially homomorphic encryption scheme [3] capable of evaluating 2-DNF expressions in encrypted form. For example, given two encryptions of messages, we can obtain an encryption of the sum of the messages without decryption or compromising privacy. Indeed, the cryptosystem remains semantically secure. Being (in part) based on the Paillier cryptosystem, BGN inherits its additive homomorphic properties. Moreover, with the clever introduction of pairings, BGN is capable of homomorphically evaluating *one level of multiplication* operations.

The BGN algorithm constructs a homomorphic encryption scheme using finite groups or composite order that support bilinear maps. The construction outlined in [3] uses groups over elliptic curve where homomorphic additions translate into elliptic curve point addition and homomorphic multiplication translates into a pairing operation. Leveraging the single multiplication afforded by pairing operation Boneh, Goh and Nissim also manage to reduce the communication

complexity in the basic step of the Kushilevitz-Ostrovsky PIR protocol from $\mathcal{O}(\sqrt{N})$ to $\mathcal{O}(\sqrt[3]{N})$. In contrast, the computational efficiency of BGN (for the server side PIR computation) scheme lies the pairing operation. Guillevic [18] developed optimized implementations to support BGN which reveals that parings over composite order elliptic curves are far less efficient than parings over prime order curves and also require significantly larger parameter sizes to reach the same security level.

2.3 Aguilar-Melchor-Gaborit's Lattice Based PIR

Most of the single server cPIR schemes rely on costly algebraic operations with large operands such as modular multiplications [32, 2, 33], or pairing operations on elliptic curves [3] to realize the homomorphic evaluations. In contrast, the PIR scheme by Aguilar-Melchor and Gaborit [30, 31] makes use of a new lattice based construction replacing costly modular operations, with much cheaper vector addition operations in lattices. The security is based on the differential hidden lattice problem, which they relate like in many lattice based construction to NP-complete problems in coding theory. Via this connection the scheme is also related to the NTRU scheme [13].

Very briefly, the PIR schemes works as follows. The scheme utilizes a secret random $[N, 2N]$ matrix M of rank N over a field $\mathbb{Z}_p$ which is used to generate a set of *different matrices* obtained by multiplication by invertible random matrices. These matrices are disturbed by the user by the introduction of noise in half of the matrix columns to obtain softly disturbed matrices (SDMs) and hardly disturbed matrices (HDMs). To retrieve an element from the database the client sends a set of SDMs and one HDM to the PIR server. The PIR server inserts each of its elements in the corresponding matrix with a multiplicative operation OP and sums all the rows of the resulting matrices collapsing the PIR server reply to a single noisy vector over $\mathbb{Z}_p$.

While they proposed full fledged protocol and implementation [30, 31], their analysis was limited to server-side computations on a small database consisting of twelve 3 MByte files. Later Olumofin and Goldberg [21] performed extensive experiments with a broad set databases sizes under realistic network bandwidth settings determining that the lattice based Aguilar-Melchor and Gaborit PIR scheme is one order of magnitude more efficient than a simple PIR.

3 From SWHE to PIR

Consider a database D with $|D| = 2^\ell$ rows. Clearly ℓ index bits are sufficient to address all rows. Assume the data bit contained in row i is denoted by D_i. We may retrieve an element of D with given index $x \in \{0,1\}^\ell$ which holds D_x by computing:

$$f(x) = \sum_{y \in [2^\ell]} (x = y) D_y \pmod 2 \ , \tag{1}$$

where the bitwise comparison $(x = y)$ may be computed as $\prod_{i\in[\ell]}(x_i + y_i + 1)$. Here $[\ell] = \{0, 1, \ldots, \ell - 1\}$ for $\ell > 0$ and $[\ell] = \{\}$ otherwise. The function of the inner loop is to check if each bit of the given x matches the corresponding bit of the y value currently processed. The boolean result is multiplied with the current data value D_y and added to the sum. All of the summed terms except the one where there was a match becomes zero and therefore does not affect the result. Therefore, $f(x) = D_x$.

This arithmetic retrieval formulation allows us to build PIRs and sPIRs. In this case the index value x is in encrypted form. Therefore, the database curator does not know which row is read from the database. We wish the curator to still be able to retrieve and serve the requested row. The data in the row itself can also be in encrypted form in which case the protocol is referred to as a symmetric PIR or sPIR in short. In this setting, if the index x is encrypted using a homomorphic encryption scheme E we may evaluate $f(x)$ homomorphically. From the formulation of $f(x)$ we need E to be able to compute a large number of homomorphic additions (XORs) $O(2^\ell)$ and a small number of multiplications (ANDs) ℓ and $\ell + 1$ if the rows are encrypted[3].

4 Picking the SWHE Scheme

To build a PIR for a database of size 2^ℓ as described in Section 3 we need an efficient SWHE instantiation that can evaluate a circuit of depth $\lceil \log_2(\ell) \rceil$. In practice a depth 5 or 6 circuit will suffice since that will give us an ability to construct a PIR for a database of size 2^{32} and 2^{64}, respectively.

For this we make use of the modified NTRU scheme [13] introduced by Stehlé and Steinfeld [12] with a number of optimizations introduced to this construction by Lopez-Alt, Tromer and Vaikuntanathan [11] to turn Stehlé and Steinfeld's scheme into a full fledged fully homomorphic encryption scheme. Here we only need to support a few levels and therefore the full Lopez-Alt, Tromer and Vaikuntanathan scheme is not needed. Stehlé and Steinfeld [12] formalized the security setting and reduced the security of their NTRU variant to the ring learning with error (RLWE) problem. More specifically, they show that if the secret polynomials are selected from the discrete Gaussian distribution with rejection then the public key is indistinguishable from a uniform sample. Unfortunately, the reduction does not carry over to the fully homomorphic setting since relaxation of parameters e.g. a larger modulus is needed to accommodate the noise growth during homomorphic evaluation as noted in [11]. We next summarize our instantiation of the scheme in [12] in a way that supports restricted homomorphic evaluations but does not require all the machinery of [11].

We require the ability to sample from a probability distribution χ on B-bounded polynomials in $R_q := \mathbb{Z}_q[x]/(x^n+1)$ where a polynomial is "B-bounded" if all of its coefficients lie in $[-B, B]$. For example, we can sample each coefficient

[3] Note that we restricted the database entries D_i to be bits but a w-bit entry can also easily be handled by considering w parallel and independent function evaluations.

from a discrete Gaussian with mean 0 and discard samples outside the desired range. Each AND gate evaluation incurs significant noise growth and therefore we use the modulus reduction technique introduced by Brakerski, Gentry and Vaikuntanathan [10]. We assume we are computing a leveled circuit with gates alternating between XOR and AND and modulus reduction taking place after each AND level. We use a decreasing sequence of odd prime moduli $q_0 > q_1 > \cdots > q_d$ where d is the depth of the PIR circuit. In this way, the key (public and evaluation keys) can become quite large and it remains a practical challenge to manage the size of this data and handle it efficiently. For this implementation we specialize the prime moduli q_i by requiring $q_i | q_{i+1}$ as was proposed in [16]. This allows us to eliminate the need for key switching and to reduce the public key size significantly. Also in this implementation we do not use relinearizations as proposed in [11] since we are in a single user setting and we have a shallow well structured circuit (a perfect binary tree) to evaluate. This will significantly improve the efficiency of implementation since relinearization is an expensive operation [16]. The primitives are as follows:

- KeyGen: We choose a decreasing sequence of primes $q_0 > q_1 > \cdots > q_d$ and a polynomial $\Phi_m(x)$, the m-th cyclotomic polynomial of degree $n = \varphi(m)$. For each i, we sample $u^{(i)}$ and $g^{(i)}$ from distribution χ, set $f^{(i)} = 2u^{(i)} + 1$ and $h^{(i)} = 2g^{(i)} \left(f^{(i)}\right)^{-1}$ in ring $R_{q_i} = \mathbb{Z}_{q_i}[x]/\langle \phi(x) \rangle$. (If $f^{(i)}$ is not invertible in this ring, re-sample.)
- Encrypt: To encrypt a bit $b \in \{0, 1\}$ with a public key $(h^{(0)}, q_0)$, random samples s and e are chosen from χ and compute the ciphertext as $c^{(0)} = h^{(0)}s + 2e + b$, a polynomial in R_{q_0}.
- Decrypt: To decrypt the ciphertext c with the corresponding private key $f^{(i)} = f^{2^i} \in R_{q_i}$, multiply the ciphertext and the private key in R_{q_i} then retrieve the message via modulo two reduction: $m = c^{(i)} f^{(i)} \pmod 2$.
- XOR: For two ciphertexts $c_1^{(0)} = \mathsf{Encrypt}(b_1)$ and $c_2^{(0)} = \mathsf{Encrypt}(b_2)$ then their homomorphic XOR is evaluated by simply adding the ciphertexts $\mathsf{Encrypt}(b_1 + b_2) = c_1^{(0)} + c_2^{(0)}$.
- AND: Polynomial multiplication is realized in two steps. We first compute $\tilde{c}^{(i-1)}(x) = c_1^{(i-1)} \cdot c_2^{(i-1)} \pmod{\phi(x)}$ and then perform a modulus reduction operation as $\tilde{c}^{(i)}(x) = \left\lfloor \frac{q_1}{q_0} \tilde{c}^{(i-1)}(x) \right\rceil_2$ where the subscript 2 on the rounding operator indicates that we round up or down in order to make all coefficients equal modulo 2.

4.1 Concrete Setting

To instantiate the Stehlé Steinfeld variant of NTRU for depth d we need to pick a large enough q_0 value to reduce the modulus d times. For instance, for a selection of $B = 2$ and if we cut by 24 bits in each iteration we need at least 200 bits. For such a q parameter we can then select n based on the Hermite factor. The Hermite factor was introduced by Gama and Nguyen [14] to estimate the

hardness of the shortest vector problem (SVP) in an n-dimensional lattice L and is defined as

$$\gamma^{2n} = \frac{||b||}{\mathsf{vol}(L)^{\frac{1}{2n}}}$$

where $||b||$ is the length of the shortest vector or the length of the vector for which we are searching. The authors also estimate that, for larger dimensional lattices, a factor $\delta^n \leq 1.01^n$ would be the feasibility limit for current lattice reduction algorithms. In [15], Lindner and Peikert gave further experimental results regarding the relation between the Hermite factor and the recovery time as $t(\gamma) := \log(T(\gamma)) = 1.8/\log(\gamma) - 110$. For instance, for $\gamma^n = 1.0066^n$, we need about 2^{80} seconds on an AMD Opteron running at 2.5 Ghz [15]. Since we are using a construction based on NTRU we need to determine the desired Hermite factor for the NTRU lattice. Coppersmith and Shamir in [19] show that an attacker would gain useful information with a lattice vector as close as norm $q/4$ to the original secret key vector. Therefore we take $||b|| = q/4$ and $\mathsf{vol}(L) = q^n$ and compute the Hermite factor for the NTRU lattice as $\gamma = (\sqrt{q}/4)^{1/(2n)}$.

To select parameters we also need to consider the noise growth. Since we no longer use relinearization, the powers of the secret key will grow exponentially through the levels of evaluation. To cope with the growth we use the modulus reduction as described in Section 4. Following the noise analysis of [16] (Section 5) we can express the correctness condition as $||c^{2^i} f^{2^d}||_\infty < q_d/2||$ assuming we are evaluating a depth d circuit. Also note that instantiation we fix χ to choose from $\{-1, 0, 1\}$ with probabilities $\{0.25, 0.5, 0.25\}$, respectively. With modulus reduction rate of $\kappa \approx q_{i+1}/q_i$ the following equation holds $c^{2^d} f^{2^d} = (\ldots((c^2\kappa + p_1)^2\kappa + p_2)^2 \ldots \kappa + p_{2^i})f^{2^d}$. In Table 1 we computed the Hermite factor and supported depth for various sizes of q_0 and n for our scheme.

Table 1. Hermite factor and supported circuit depth (γ, d) for various q and n.

n	$\log_2(q)$				
	512	640	768	1024	1280
2^{13}	$(1.01083, 5)$	$(1.0135, 5)$	$(1.0162, 6)$	$(1.0218, 6)$	$(1.0273, 7)$
2^{14}	$(1.00538, 5)$	$(1.0067, 5)$	$(1.0081, 6)$	$(1.0108, 6)$	$(1.0135, 6)$
2^{15}	$(1.00269, 5)$	$(1.0033, 5)$	$(1.0040, 6)$	$(1.0054, 6)$	$(1.0067, 6)$

5 The NTRU based PIR Protocol

In our encryption scheme we are able to batch additional information to the ciphertext polynomials. This allows us to perform retrieval using two different query mechanisms:

Bundled Query one query is used to to retrieve data stored at different rows of the database (different indicies are queried).

Single Query the query retrieves data from a single row (a single index) but processes more indices at a time during the PIR server computation.

Next we explain an FHE optimization technique named *batching* and show how it gives us the two query methods.

Batching. Batching was introduced by Smart and Vercauteren [8, 9]. It allows us to evaluate a circuit on multiple independent data inputs simultaneously by embedding them into the same ciphertext. The independent data inputs are encoded to form special binary polynomials that are used as message polynomials. Addition and multiplication of these the message polynomials has the effect of evaluating XOR and AND operations on the packed message bits. The encoding is achieved using the Chinese Remainder Theorem. First we set $R_{q_0} = \mathbb{Z}_{q_0}/\langle \Phi_m(x) \rangle$, where $\Phi_m(x)$ is defined as the m^{th} cyclotomic polynomial. The cyclotomic polynomial $\Phi_m(x)$ is factored into equal degree irreducible polynomials over $\mathbb{F}_2$ $\Phi_m(x) = \prod_{i \in [\varepsilon]} F_i(x)$. where $\lambda = \deg(F_i)$ is the smallest integer that satisfies $m|(2^\lambda - 1)$. A message polynomial $m(x)$ in the residue space is represented as $m_i = m(x) \pmod{F_i(x)}$. Therefore; given a message bit vector $\mathbf{m} = \{m_0, m_1, m_2, m_3 \ldots, m_\varepsilon\}$ we may compute the corresponding message polynomial using inverse CRT $m(x) = \mathsf{CRT}^{-1}(\mathbf{m})$. Using these special formed messages, we can perform bit level AND and XOR operations: $m_i \cdot m'_i = m(x) \cdot m'(x) \pmod{F_i(x)}$ and $m_i \oplus m'_i = m(x) + m'(x) \pmod{F_i(x)}$.

Bundled Query. The batching technique allows us to embed multiple indices into a query ciphertext and thereby facilitate retrieval of multiple database entries. First recall our PIR function $\sum_{y \in [2^\ell]} [\prod_{i \in [\ell]} (x_i + y_i + 1)] D_y$, which we will now evaluate on encrypted x and y values. Using the batching technique we may evaluate ε retrievals with indices $\beta[1], \ldots, \beta[\varepsilon]$ simultaneously. First we form their bit representation as:

$$\begin{array}{cccc} \beta[1] = (\beta_{\ell-1}[1] & \beta_{\ell-2}[1] & \ldots & \beta_0[1]) \\ \beta[2] = (\beta_{\ell-1}[2] & \beta_{\ell-2}[2] & \ldots & \beta_0[2]) \\ \vdots & \vdots & \vdots & \vdots \\ \beta[\varepsilon] = (\beta_{\ell-1}[\varepsilon] & \beta_{\ell-2}[\varepsilon] & \ldots & \beta_0[\varepsilon]) \end{array}$$

Using the columns of the bit matrix on the RHS, we can compute the batched polynomial versions of the index bits $\tilde{\beta}_i(x)$ as:

$$\tilde{\beta}_i(x) = \mathrm{CRT}^{-1}(\beta_{\mathrm{i}}[1],\ \beta_{\mathrm{i}}[2],\ \ldots\ ,\ \beta_{\mathrm{i}}[\varepsilon])$$

Later, these polynomials are encrypted as: $\xi_i(x) = h(x)s_i(x) + 2e_i(x) + \tilde{\beta}_i(x)$ for $i \in [\ell]$. The query $Q = [\xi_i(x), \ldots, \xi_{\ell-1}(x)]$ is then send to the PIR server. In order to perform parallel comparisons vector row index bit $\{y_i, y_i, \ldots, y_i\}$ should also converted into a polynomial representation using inverse-CRT. Since we are dealing with bits $y_i = \{0, 1\}$, the inverse-CRT will result in $\{0, 1\}$ polynomials, and thus $y_i(x) = y_i$. This is true for data D_y as well. Then, we can rewrite the PIR equation as: $r(x) = \sum_{y \in [2^\ell]} \left(\prod_{i \in [\ell]} (\xi_i(x) + y_i(x) + 1) \right) D_y(x)$. Given

that $y_i(x)$ has small coefficient, i.e. 1 or 0, the additions are done over the least coefficient term in the polynomial. Furthermore, having $D_y(x) = \{0, 1\}$ we may skip the product evaluations unless $D_y(x) = 1$. Once $r(x)$ is homomorphically evaluated simultaneously over the ℓ ciphertexts, the response (a single ciphertext) $R = r([\xi_0(x), \ldots, \xi_{\ell-1}(x)])$ is sent back to the PIR client. The ciphertext response is first decrypted and the individual data entries are recovered using modular reductions: $D_i = \mathsf{dec}(r(x)) \pmod{F_i(x)}$.

Single Query. In the single query mode we will also perform batching as in the Bundled Query mode. However, here we will place the same index into all index slots. The resulting polynomials are encrypted as before giving us a query $Q = [\xi_i(x), \ldots, \xi_{\ell-1}(x)]$. Though this is similar to the Bundled Query, the PIR server side computation is handled quite differently. For parallel comparisons we batch the row bits of y_i and D_y as well:

$$y_i(x) = \mathrm{CRT}^{-1}\{y_i[1], \ \ldots \ , \ y_i[\varepsilon]\}, \quad D_y(x) = \mathrm{CRT}^{-1}\{D_y[1], \ \ldots \ , \ D_y[\varepsilon]\}.$$

These conversions are done on-the fly and are not precomputed. Working in modulo 2 arithmetic makes the evaluations sufficiently fast and easy such that it only adds a small overhead. Although precomputation is an option, storing converted message polynomials would take extra space. The comparison equation will stay the same with the Bundled Query, but $y_i(x)$ and $D_y(x)$ will now binary polynomials. Therefore, we require polynomial addition inside the product and a polynomial multiplication with $D_y(x)$. Since in each iteration we are comparing ε indecies simultaneously we can process the database ε times faster. This speedup comes at a price where each iteration need to carryout a multiplication by the polynomial representation of the batched D_y.

The response ciphertext is first decrypted and then reduced to recover the evaluation bits as before: $z_i = \mathsf{dec}(r([\xi_0(x), \ldots, \xi_{\ell-1}(x)])) \pmod{F_i(x)}$. In a Single Query each z_i refers to a subsection of the summation therefore to compute the overall result we perform a final bit summation $D_y = \sum_{i \in \varepsilon} z_i \bmod 2$.

6 Performance

We implemented the proposed PIR protocol with both the Single and Bundled Querying modes in C++ where we relied on Shoup's NTL library version 6.0 [17] for the lattice operations. Table 2 shows minimal parameters to support various evaluation depths. Each depth can support upto 2^{2^d} entries, e.g. $d = 5$ can support 4 Billion entries. The parameter ε denotes the number of message slots that we can bundle. The query and response sizes are given in Table 2 *without* normalization by ε. In the Bundled Query mode sizes may be normalized with ε to determine the bandwidth per query. In Table 3, we present the time performance for query processing. The reported times are normalized per row of the database and per query. The time is split into two components: the time required to compare the encrypted index to the index of the currently processed row, and the time required to add the data in the current row to the summation.

Table 2. Polynomial parameters and Query/Response sizes necessary to support various database sizes N.

max N	$(\log q,\ n)$	ε	Query Size (MB)	Response Size (KB)
4 Billion	(512, 16384)	1024	32	784
65536	(250, 8190)	630	3.9	154
256	(160, 4096)	256	0.625	44

While the computation cost in comparison is quite high we should note that we are paying primarily for the index comparison. In the Bundled Query case, once the index comparison is completed we may simply reuse the comparison result and only compute an addition operation for each additional bit in the same database entry. In this sense, our results are similar to the other lattice based PIR construction by Melchor and Gaborit [30]. The index comparison may be considered as a one time overhead to be paid for each row that would be amortized as database rows get wider. Still due to the large vector sizes data aggregation will be rather slow. For instance; in a Bundled Query with $d = 4$ and 1 GBytes of data in a row, the processing time will be about 8 times slower than a Kushilevitz and Ostrovsky implementation as given in [21].

Table 3. Index comparison and data aggregation times per entry in the database for (d, ε) choices of $(5, 1024), (4, 630)$ and $(3, 256)$ on Intel Pentium @ 3.5 Ghz.

	Bundled Query (msec)			Single Query (msec)		
Depth (d)	5	4	3	5	4	3
Index comparison	4.45	0.71	0.31	4.56	2.03	1.29
Data aggregation	0.22	0.09	0.04	37	7.45	3.40

What we loose in computational efficiency, we make up for in terms of bandwidth. In Table 4, we give Complexity and Query size comparisons. As before, N is the size of the database and α is the ciphertext size that differs in each scheme [4]. In the Bundled Query case, for instance, the query is formed by $\ell = 2^d = 32$ ciphertexts each made of Mbytes. By normalizing with ε index retrievals in a single query, per retrieval we are paying about 32 Kbytes. The query size of our scheme is smaller by a factor of 1024, 1200, and 3072 when compared to BGN, Melchor-Gaborit and Kushilevitz-Ostrovsky, respectively.

Finally, we would like to point out that for all practical purposes the size α of the ciphertexts in the query and response can be considered *almost* independent of the database size. Therefore, the size of the ciphertext, i.e. α is very mildly effected when the database size is increased. Indeed, as seen in Table 4 when the

[4] For [30], we used the given size of 37.5 MByte for 20,000 entries since it does not provide a complexity. The size will grow significantly when N goes to 2^{32}.

table size is grown from 256 entrees to 2^{16} entries, the ciphertext size grows only about by 1.26 times in the bundled case.

Table 4. Comparison of query sizes for databases upto 2^{32}, 2^{16} and 2^8 entries. Bandwidth complexity is given in the number of ciphertexts; α denotes the ciphertext size.

	BW	α			Query Size		
	Compl.	$d=5$	$d=4$	$d=3$	$d=5$	$d=4$	$d=3$
Boneh-Goh-Nissim	$\alpha\sqrt{N}$	6144	6144	6144	96 MB	384 KB	24 KB
Kushilevitz-Ostrovsky	$\alpha\sqrt{N}$	2048	2048	2048	32 MB	128 KB	8 KB
Ours (Single)	$\alpha \log N$	8388608	2047500	655360	**32 MB**	**249 KB**	**80 KB**
Ours (Bundled)	$\alpha \log N$	8192	3250	2560	**32 KB**	**406 B**	**320 B**

Acknowledgments

Funding for this research was in part provided by the US National Science Foundation CNS Awards #1117590 and #1319130.

References

1. Gentry, C.: Fully homomorphic encryption using ideal lattices, Symposium on the Theory of Computing (STOC), 2009, pp. 169-178.
2. Kushilevitz, E., Ostrovsky, R.: Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval FOCS '97, 1997.
3. Boneh, D., Goh, E.J., Kobbi, N.: Evaluating 2-DNF formulas on ciphertexts. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 325–341. Springer-Verlag, 2005.
4. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. *Advances in Cryptology - CRYPTO 2012*, 850-8, 2012.
5. Gentry, C., Halevi, S.: Implementing Gentry's fully-homomorphic encryption scheme, *Advances in Cryptology–EUROCRYPT*, pp. 129–148, 2011.
6. Bos, J.W., Lauter, K., Loftus, J., Naehrig, M.: Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme In Lecture Notes in Computer Science PQCrypto 2013. pp. 45–64. Springer, 2013.
7. Lipmaa, H.: An Oblivious Transfer Protocol with Log-Squared Communication. In: The 8th Information Security Conference (ISC'05). Volume 3650 of Lecture Notes in Computer Science., Springer-Verlag (2005) 314–328.
8. Gentry, C., Halevi, S., Smart, N.: Fully homomorphic encryption with polylog overhead. Manuscript, 2011.
9. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Manuscript at http://eprint.iacr.org/2011/133, 2011.
10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. ITCS (2012): 309-325.

11. Lopez-Alt. A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Proceedings of the 44th symposium on Theory of Computing, pp. 1219-1234. ACM, 2012.
12. Stehlé, D., Steinfeld, R.: Making NTRU as secure as worst-case problems over ideal lattices. Advances in Cryptology EUROCRYPT (2011): 27-4
13. Hoffstein, J., Pipher, J., Silverman, J.: NTRU: A ring-based public key cryptosystem. Algorithmic number theory (1998): 267-288.
14. Gama, N., Nguyen, P.: Predicting lattice reduction. Advances in Cryptology-EUROCRYPT (2008): 31-5
15. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. Topics in Cryptology CT-RSA (2011): 319-339
16. Doröz, Y., Hu, Y., Sunar, B.: Homomorphic AES Evaluation using NTRU, IACR ePrint Archive. Technical Report 2014/039 January 2014. URL: http://eprint.iacr.org/2014/039.pdf
17. NTL: A Library for doing Number Theory, http://www.shoup.net/ntl
18. Guillevic, A.: Comparing the Pairing Efficiency over Composite-Order and Prime-Order Elliptic Curves Applied Cryptography and Network Security LNCS Volume 7954, 2013, pp 357–372.
19. Coppersmith, D., Shamir, A.: Lattice Attacks on NTRU Advances in Cryptology - EUROCRYPT '97 LNCS Vol. 1233, 1997, pp 52–61.
20. Sion, R., Carbunar, B.: On the Computational Practicality of Private Information Retrieval. In: NDSS'07 (2007).
21. Olumofin, F., Goldberg, I.: Revisiting the Computational Practicality of Private Information Retrieval Financial Cryptography and Data Security, LNCS 7035, 2012, pp 158–172.
22. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private Information Retrieval. In: FOCS 95: Proceedings of the 36th Annual Symposium on the Foundations of Computer Science. pp. 41–50 (Oct 1995)
23. A. Ambainis.: Upper bound on the communication complexity of private information retrieval. In Proc. of the 24th ICALP, 1997.
24. Ishai, Y., Kushilevitz, E.: Improved upper bounds on information-theoretic private information retrieval. In Proc. of the 31th ACM Sym. on TC, 1999.
25. Chor, B., Gilboa, N.: Computationally Private Information Retrieval. In 29th STOC, pp. 304–313, 1997.
26. Ostrovsky, R., Shoup, V.: Private Information Storage. In 29th STOC, pp. 294-303, 1997.
27. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: single database, computationally-private information retrieval. In: FOCS 97. p. 364 (1997)
28. Cachin, C., Micali, S., Stadler, M.: Computationally Private Information Retrieval with Polylogarithmic Communication. In: EUROCRYPT 99. pp. 402–414 (1999)
29. Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: ICALP: Annual International Colloquium on Automata, Languages and Programming. (2005) 803–815
30. Aguilar-Melchor, C., Gaborit, P.: A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol. In: WEWORC 2007 (July 2007)
31. Aguilar Melchor, C., Crespin, B., Gaborit, P., Jolivet, V., Rousseau, P.: High-Speed PIR Computation on GPU. In: SECURWARE'08. pp. 263–272 (2008)
32. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences 28 (2): 270–299
33. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. EUROCRYPT. Springer. (1999). pp. 223–238.