Bandwidth-Optimized Parallel Private Information Retrieval[*]

Ecem Ünal
Sabancı University
ecemunal@sabanciuniv.edu

Erkay Savaş
Sabancı University
erkays@sabanciuniv.edu

ABSTRACT

We present improved and parallel versions of Lipmaa's computationally-private information retrieval (CPIR) protocol based on a additively-homomorphic cryptosystem. Lipmaa's original CPIR utilizes binary decision diagrams, in which non-sink nodes have two children nodes and the data items to be retrieved are placed in the sink nodes. In our scheme, we employ, instead, quadratic and octal trees, where non-sink nodes have four and eight child nodes, respectively. Using other tree forms, which does not change the asymptotic complexity, results in shallow trees by which we can obtain an implementation that is an order of magnitude faster than the original scheme. We also present a non-trivial parallel algorithm that takes advantage of shared-memory multi-core architectures. Finally, our scheme proves to be highly efficient in terms of bandwidth requirement, the amount of data being exchanged in a run of the CPIR protocol.

1. INTRODUCTION

A private information retrieval (PIR) scheme, is a cryptographic protocol that allows a user to access any data item, f_i, in a remotely stored database $\mathcal{F}$ (i.e. $f_i \in \mathcal{F}$), without revealing to the database server which data item he is accessing; namely neither i nor f_i is revealed to the server. The concept for the protocol was first introduced in [5] and has recently gained considerably high attention as a result of the raised awareness in security and privacy concerns pertinent in outsourcing and cloud computing practices. Naturally, a cloud computing user wants to, not only protect the secrecy and integrity of his data, but also hide what he does with it; namely when and how frequently a data item is accessed.

[*]Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org. SIN '14, September 09 - 11 2014, Glasgow, Scotland UK. Copyright 2014 ACM 978-1-4503-3033-6/14/09... $15.00.

The concept of computational PIR (CPIR), introduced in [6], provides the assurance that the difficulty of the server finding out i or f_i can be reduced to a computationally difficult problem. Lipmaa's computationally-private information retrieval (CPIR) protocol [12] suggests using additively-homomorphic encryption algorithm by Damgård and Jurik [7], whose security depends on the well-known decisional composite residuosity assumption while other schemes in the literature depend on relatively less studied lattice problems as in [1,2]. There are also other more recent schemes based on fully homomorphic encryption techniques such as the one in [8]. The Lipmaa's scheme, which uses binary decision diagrams (hence, the scheme being known as BddCpir), is known to offer superior bandwidth performance due to its logarithmic asymptotic complexity.

A trivial solution for PIR is that the user downloads all the database and selects the requested data item, which is possible since the user can see other data items in PIR, which is not the case with the oblivious transfer protocols [16]; a close relative of PIR in the cryptographic literature. Therefore, the essential requirement for an efficient PIR is that the amount of data exchanged between the user and the server must be sublinear to the size of the database. Many schemes [1,2,8] provide very efficient techniques to accelerate the server-side computations, but fail to achieve a meaningful bandwidth performance. On the other hand, BddCpir scheme is not one of the best schemes in the literature in terms of computational complexity.

Our contribution. Firstly, we provide new, improved versions of the original BddCpir [12] using quadratic and octal trees, to the computational complexity without adversely affecting the bandwidth performance. Secondly, we propose a non-trivial parallel algorithm for server-side computations. Lastly, we give a comparison for the bandwidth requirement of the proposed technique and those of two other techniques, and show that the proposed technique is superior.

2. BACKGROUND

The proposed PIR protocol in this work is based on Lipmaa's (n, 1) - CPIR protocol, BddCpir [12], which uses binary decision diagrams and the additively-homomorphic public-key cryptosystem [7]. In this section, we first provide a brief introduction to binary decision diagrams (*BDD*) utilized in BddCpir. Then, we explain the basics of the Damgård-Jurik cryptosystem [7] utilized for encryption and decryption in BddCpir.

2.1 Binary Decision Diagrams

A binary decision diagram is a directed acyclic graph, where each node can have at most two children as in binary tree. The underlying graphs of the decision diagrams that we use in our protocol always have tree properties, therefore in this context BDDs can also be thought as decision trees.

Properties of a BDD. In a binary decision diagram, non-sink (also called non-terminal) nodes are labeled as $R_{i,j}$ where i denotes the level in the tree and j denotes the position of the node in a level. Also, the two outgoing edges of the internal nodes are labeled as 0 or 1, respectively. The sink nodes, however, are data items whose indices are m-bit strings, where m is the depth of the tree since these strings represent the route taken from the root node to that sink node; in other words it is the concatenation of the labels of the edges that are visited while reaching the sink node from the root node. In Figure 1, a binary decision tree with depth two is illustrated.

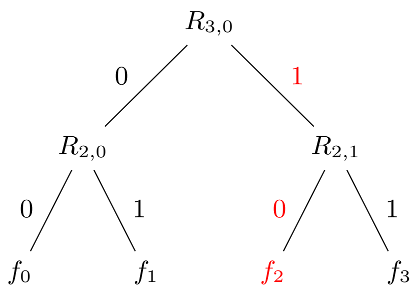

Figure 1: An example BDD constructed by server, shows the case where the client queries the database with binary input $x = 10$, to reach file f_2.

In the CPIR protocol, BddCpir, the sink nodes represent the data items, which are privately retrieved on user input. Thus, the labels of the sink nodes are used to identify the indexes of data items. If the client queries the server with a binary input x of length m, the server returns the data item f_x, stored in the sink node with the label x. As shown in the following, the index of a data item is encrypted using an additively-homomorphic public-key cryptosystem before sending it to the server.

An additively-homomorphic public key cryptography algorithm satisfies the following important homomorphic properties over encryption operation

$$E(x_1) \cdot E(x_2) = E(x_1 + x_2) \text{ and } E(x_1)^c \quad = E(c \cdot x_1),$$

where x_1 and x_2 are plaintext messages, and c is a constant.

Quadratic and Octal Trees. For performance reasons, instead of the binary decision diagrams, we propose using four-child trees (Quadratic trees or simply *quadtree*) and eight-child trees (Octal trees or simply *octree*) in our protocol. These new types of trees, essentially have the same properties as the binary trees, except that each node has four and eight children, in quadratic and in octal trees, respectively. In a quadratic tree, edges of the internal nodes are labeled by two-bit strings, namely $\{00, 01, 10, 11\}$ and hence the labels of the sink nodes have $2m$-bit strings where m again represents the depth of the tree. For the octal tree case, the set of the edge labels consist of three bit strings containing all eight possibilities $\{000, 001, 010, \ldots, 111\}$, therefore the sink nodes should be labeled by strings of $3m$-bit long.

2.2 $(n, 1)$ - CPIR

In this section, we first explain the (2,1) CPIR sub-protocol, which is the base of the $(n,\ 1)$ CPIR scheme in [12]. Then, we show how it is extended to database with n items.

(2, 1) CPIR. In 1-out-of-2 protocol, there are only two data items stored in the server's database, namely (f_0, f_1); therefore the client's input x is either 0 or 1 since it can only request one of $\{f_0, f_1\}$. The properties of PIR requires that the server send f_x to the client without knowing or learning x (i.e., f_x). Lipmaa's protocol [12], $(2, 1)$, CPIR works in three steps: 1) The client sets secret and public keys (sk, pk), computes $c = E_{pk}(x)$ and sends (pk, c) to the server, 2) the server computes $R = E_{pk}(f_0) \cdot c^{f_1 - f_0}$ and sends R to the client, and 3) the client computes $D_{sk}(R)$ to find f_x.

Since the cryptosystem used for encryption and decryption is additively homomorphic we can prove that the client will get f_x at the end of the protocol as

$$\begin{aligned} R &= E_{pk}\left(f_0\right) \cdot c^{\,f_1 - f_0} = E_{pk}\left(f_0\right) \cdot E_{pk}\left(x\right)^{\,f_1 - f_0} \\ &= E_{pk}\left(f_0 + x\left(f_1 - f_0\right)\right) = E_{pk}\left(f_x\right). \end{aligned}$$

Extending (2, 1)-CPIR to (n, 1)-CPIR. The (2, 1)-CPIR protocol is used as the primitive for deeper binary trees to realize $(n,\ 1)$ - CPIR protocol. The protocol starts with the sink nodes, continues in a bottom-up manner, and stops at the root node. While going up, the data items are encrypted repeatedly, resulting in the requested data item, which is encrypted as many times as the depth of the tree. For instance, the server computation of the (4, 1)-CPIR protocol for data items $\{f_0, f_1, f_2, f_3\}$ is implemented for the user input $x = (x_1, x_0)$ in two steps as follows. In the first step, we calculate

$$R_{2,0} = E_{pk}\left(f_0\right) \cdot c_0^{f_1 - f_0} \quad \text{ and } R_{2,1} = E_{pk}\left(f_2\right) \cdot c_0^{f_3 - f_2},$$

where $c_0 = E_{pk}(x_0)$. In the second step, we work with ciphertexts obtained from the previous step as

$$\begin{aligned} R_{3,0} &= E_{pk}\left(R_{2,0}\right) \cdot c_1^{R_{2,1} - R_{2,0}} \\ &= E_{pk}\left(R_{2,0} + c_1 \cdot \left(R_{2,1} - R_{2,0}\right)\right) \\ &= E_{pk}\left(E_{pk}\left(f_{0x_0}\right) + c_1 \cdot \left(E_{pk}\left(f_{1x_0}\right) - E_{pk}\left(f_{0x_0}\right)\right)\right). \end{aligned}$$

Therefore, we obtain the double encryption of f_x, namely $E_{pk}^{(2)}(f_x)$, which is sent to the user. Note that $c_1 = E_{pk}^{(2)}(x_1)$. In the general case, the client receives $E_{pk}^{(m+1)}(f_x)$, where m is the depth of the binary tree. Note also that $c_i = E_{pk}^{(i+1)}(x_i)$.

2.3 Damgård - Jurik Cryptosystem

An additively-homomorphic public key encryption algorithm such as Paillier's probabilistic public key cryptosystem [15] can be used in (2,1)-CPIR, BddCpir. However, the Paillier encryption algorithm leads to message expansion, where the ciphertext will be longer than the plaintext. Therefore, a multiple encryption is not possible with the Paillier public key algorithm, which prevents to extend the (2, 1)-CPIR scheme to general case of (n, 1)-CPIR. The Damgård-Jurik public key cryptosystem [7], which is a generalization of the Paillier scheme, is used in the proposed scheme.

Damgård - Jurik cryptosystem uses the RSA setting, where we employ modulo arithmetic, with a modulus N, which is the product of two sufficiently large prime numbers, p and q. Unlike RSA, which is based on the difficulty of factorization of large integers, the security of the Damgård Jurik cryptosystem relies on the decisional composite residuosity assumption [15], which is also used in the original Paillier cryptosystem. The key generation, encryption and decryption algorithms are briefly described in the following.

Key Generation. The public keys N and g are generated first. The modulus N is an RSA modulus of length k bits, where $N = pq$. For the other component of the public key g, referred as the base, we use the simplified version $g = N+1$ as suggested in [7]. For the private key d, we first compute the least common multiple of $p-1$ and $q-1$, $\lambda = \mathrm{lcm}(p-1, q-1)$. We then choose the private key d such that

$$d = 1 \bmod N^s \text{ and } d = 0 \bmod \lambda.$$

using the Chinese Remainder Theorem (CRT).

Encryption. Given a plaintext $m \in Z_{N^s}$, we choose a random number $r \in_R Z^*_{N^{s+1}}$ and compute the ciphertext as $E(m, r) = g^m \, r^{N^s} \bmod N^{s+1}$.

Decryption. For $g = N+1$ the decryption operation results in $c^d = (1+N)^m \bmod N^{s+1}$. Then, using recursive Paillier decryption algorithm, we can obtain the plaintext m. For more information about the decryption operation refer to [7].

The natural number s in encryption plays an important role in the complexity of the protocol, as we go up in the binary tree it increments in each level. In other words, s denotes the number of multiple encryptions during the computations. The first encryptions in the sink nodes are performed with $s = 1$ while those in the second level will be done with $s = 2$. For instance, for a tree with eight data items in its sink nodes, the encrypted index values are formed as $c_0 = g^{x_0} \, r_0^N \bmod N^2$, $c_1 = g^{x_1} \, r_1^{N^2} \bmod N^3$, and $c_2 = g^{x_2} \, r_2^{N^3} \bmod N^4$, where $r_0 \in_R Z^*_{N^2}$, $r_1 \in_R Z^*_{N^3}$, and $r_2 \in_R Z^*_{N^3}$. Considering the quadratic complexity of Damgård-Jurik encryption operation, the time complexity of the CPIR scheme will be prohibitively high even for databases with moderately high number of data items. The continuous message expansion with multiple encryptions hinders the scalability of the CPIR scheme.

3. PROBLEM STATEMENT

PIR protocols, by definition, reduce the communication cost compared to the trivial solution that involves sending the entire database to the user. This differentiates the PIR protocols from oblivious transfer protocols [16,18] requiring much higher bandwidth, in which user is allowed to retrieve at most one of the database items. PIR protocols result in more bandwidth efficient solutions by removing this additional privacy requirement. In summary, an efficient PIR protocol satisfies two performance requirements:

- **Computational Efficiency and Scalability** PIR protocols involve generally costly cryptographic operations. Computational efficiency is expressed usually as the number of data items or database size processed in a unit time from throughput perspective. The latency, however, is also important since users tolerate waiting only a limited amount of time. Scalability requires that the scheme remain applicable as the number of data items and/or database size increase. The schemes that allow parallel implementation will be advantageous for scalability. In this work, we explore the schemes that benefit parallel implementations.
- **Bandwidth Efficiency** The query and response sizes must be incomparably smaller than the database size. While many solutions minimize the query size sent from the user to the server, others focus on decreasing response size returned by the server to the user. We aim to optimize both query and response sizes.

In the next section, we outline our approach that outperforms the original BddCpir scheme in terms of both computational and bandwidth efficiency.

4. OUR APPROACH

We utilize two techniques to improve computational and bandwidth efficiency of the CPIR scheme. The first technique involves using quadratic and octal trees, in which each non-sink node has four and eight children, respectively. The second technique is a parallel algorithm that takes advantage of shared-memory multi-core processors.

4.1 (n, 1) CPIR with Quadratic Trees

In a quadratic tree, each non-sink node has four children nodes as shown in Figure 2, where a depth-2 quadtree is depicted for 16 data items, namely f_0 through f_{15}. In the binary tree, same number of data items would require the depth of four, that would result in higher overhead in computation and bandwidth requirements as will be shown in subsequent sections. The quadtree scheme increases the number of indexes that are computed and sent by the user for each level. For instance, in the binary tree the user has to compute and send $c_i = E^{(s)}_{pk}(x_i)$ for each level in the tree. On the other hand, in addition to c_i and c_{i+1}, the user has to compute and send $c_{i,i+1} = E^{(s)}_{pk}(x_i \cdot x_{i+1})$, where s denotes the current level of the tree. Although the number of encrypted indexes used in quadtree implementation is now more than those of binary tree implementation, we achieve an improvement for the overall bandwidth requirement with the new method as shown in subsequent sections.

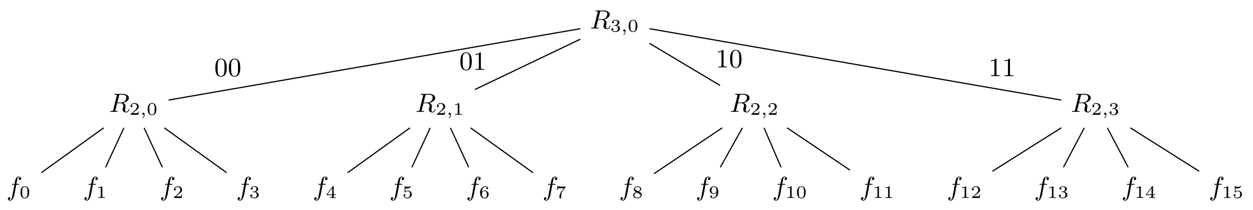
$R_{3,0}$
00
01
10
11
$R_{2,0}$
$R_{2,1}$
$R_{2,2}$
$R_{2,3}$
f_0 f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8 f_9 f_{10} f_{11} f_{12} f_{13} f_{14} f_{15}

Figure 2: A depth-2 quadratic tree implementing (16,1)-CPIR

Assuming that the number of data items n is a power of 4, $n = 4^m$, the protocol is executed as follows:

1. Client sets the secret and public keys (sk, pk) and computes $c_{2i} = E_{pk}^{(i+1)}(x_{2i})$, $c_{2i+1} = E_{pk}^{(i+1)}(x_{2i+1})$, $c_{2i,2i+1} = E_{pk}^{(i+1)}(x_{2i} \cdot x_{2i+1})$ for $i = 0 \ldots m-1$ and sends them and pk to the server,

2. Server computes

   - for $j = 0, 1, \ldots, 4^{m-1} - 1$
   $$R_{2,j} = E_{pk}(f_{4j}) \cdot c_0^{f_{4j+1}-f_{4j}} \cdot c_1^{f_{4j+2}-f_{4j}} \cdot c_{0,1}^{f_{4j+3}-f_{4j+2}-f_{4j+1}+f_{4j}}$$
   - for $k = 2, \ldots m$ and $j = 0, 1, \ldots 4^{m-k} - 1$
   $$R_{k+1,j} = E_{pk}(R_{k,4j}) \cdot c_{2k-2}^{R_{k,4j+1}-R_{k,4j}} \cdot c_{2k-1}^{R_{k,4j+2}-R_{k,4j}} \cdot c_{2k-2,2k-1}^{R_{k,4j+3}-R_{k,4j+2}-R_{k,4j+1}+R_{k,4j}}$$

   and sends $R_{m+1,0}$ to the client.

3. Client computes $D_{sk}(R_{m+1,0})$ to retrieve f_x.

EXAMPLE 1. *For a quadtree with four sink nodes (i.e., data items), the client sends* $c_0 = E_{pk}(x_0)$, $c_1 = E_{pk}(x_1)$, *and* $c_{0,1} = E_{pk}^{(2)}(x_0 \cdot x_1)$ *to the server, who computes the following:* $R_{2,0} = E_{pk}(f_0) \cdot c_0^{f_1-f_0} \cdot c_1^{f_2-f_0} \cdot c_{0,1}^{f_3-f_2-f_1+f_0}$.

4.2 $(n, 1)$ CPIR with Octal Trees

Octal tree, in which each non-sink node has eight children nodes, decreases the depth further, which helps improve the complexity of the overall system; particularly the complexity of cryptographic operations when the number of data items is high. Similar to the quadratic tree solution, the number of indexes is increased, without adversely affecting the bandwidth requirements.

Assuming that the number of nodes is a power of 8, namely $n = 8^m$, the client sets secret and public keys (sk, pk) and computes

$$c_{3i} = E_{pk}^{i+1}(x_{3i}), c_{3i+1} = E_{pk}^{i+1}(x_{3i+1}), c_{3i+2} = E_{pk}^{i+1}(x_{3i+2}),$$
$$c_{3i,3i+1} = E_{pk}^{i+1}(x_{3i} \cdot x_{3i+1}), c_{3i,3i+2} = E_{pk}^{i+1}(x_{3i} \cdot x_{3i+2}),$$
$$c_{3i+1,3i+2} = E_{pk}^{i+1}(x_{3i+1} \cdot x_{3i+2}),$$
$$c_{3i,3i+1,3i+2} = E_{pk}^{i+1}(x_{3i} \cdot x_{3i+1} \cdot x_{3i+2})$$

for $i = 0, \ldots, m-1$ and sends them and pk to the server. The server computation is explained in Figure 3. The server finally obtains $R_{m+1,0}$ and sends it to the client. The client performs the decryption $D_{sk}(R)$ to retrieve f_x.

4.3 A Parallel Algorithm for Server-Side Computation of $(n, 1)$ CPIR Scheme

The construction of encrypted selection bits at the client side is a trivially parallel process, thus the parallel algorithm is straightforward. To exploit the parallelism at the server side, however, takes slightly more effort since computations that start at the sink nodes proceed to the nodes in the upper levels in a sequential manner. However, the operations in a level in the decision tree are independent from each other and can be performed in parallel. In addition, the homomorphic encryption operation (i.e., $E_{pk}(f_{4j})$ in the quadratic tree case) in each level of the tree consists of two modular exponentiation operations (i.e., $g^m \bmod N^{s+1}$ and $r^{N^s} \bmod N^{s+1}$) that can also be calculated in parallel. Therefore, a two-level parallel algorithm is devised, whose description is given in Algorithm 1. Simply speaking, in the algorithm, all calculations (homomorphic encryptions, modular exponentiations & multiplications) are distributed to the available cores, which perform their part of the computations in parallel.

Algorithm 1 Server computation for binary tree based (n,1)-CPIR with two-level parallelization

```
Require: $c_i$, $i = 0, \ldots, m-1$
Ensure: $R_{m+1,0}$
  for $i \leftarrow 1$ to $m$ do
      for $j \leftarrow 0$ to $R_{i+1}.size$ in parallel do
          $f_0 \leftarrow R_{i,2j}$
          $f_1 \leftarrow R_{i,2j+1}$
          in parallel do
              $temp_0 \leftarrow E_{pk}^{(i+1)} f_0$
              $temp_1 \leftarrow c_i^{f_1-f_0}$
          sync
          $R_{i+1,j} \leftarrow temp_1 \times temp_0$
      end parallel for
  end for
  return $R_{m+1,0}$
```

4.4 Analysis of Computational Complexity

In this section, we explain why the quadratic and octal tree implementations are better than the binary tree implementation in terms of the efficiency of server-side computations. We provide a theoretical analysis showing that we should expect a speedup in server-side computations. On the other

```
Server Computation
Input: $c_{3i}$, $c_{3i+1}$, $c_{3i+2}$, $c_{3i,3i+1}$, $c_{3i,3i+2}$, $c_{3i+1,3i+2}$, $c_{3i,3i+1,3i+2}$, $i = 0, \ldots, m-1$
Output: $R_{m+1,0}$

Step 1: Do the following
  for $j = 0, 1, \ldots, 4^{m-1} - 1$
      $R_{2,j} = E_{pk}(f_{8j}) \cdot c_0^{f_{8j+1}-f_{8j}} \cdot c_1^{f_{8j+2}-f_{8j}} \cdot c_2^{f_{8j+4}-f_{8j}} \cdot c_{0,1}^{f_{8j+3}-f_{8j+2}-f_{8j+1}+f_{8j}} \cdot c_{0,2}^{f_{8j+5}-f_{8j+4}-f_{8j+1}+f_{8j}}$
               $\cdot c_{1,2}^{f_{8j+6}-f_{8j+2}-f_{8j+4}+f_{8j}} \cdot c_{0,1,2}^{f_{8j+7}-f_{8j+6}-f_{8j+5}-f_{8j+3}-f_{8j}+f_{8j+4}+f_{8j+2}+f_{8j+1}}$
Step 2: Do the following
  for $k = 2, \ldots m$
    for $j = 0, 1, \ldots 4^{m-k} - 1$
      $R_{k+1,j} = E_{pk}(R_{k,8j}) \cdot c_{3k-3}^{R_{k,8j+1}-R_{k,8j}} \cdot c_{3k-2}^{R_{k,8j+2}-R_{k,8j}} \cdot c_{3k-1}^{R_{k,8j+4}-R_{k,8j}} \cdot c_{3k-3,3k-2}^{R_{k,8j+3}-R_{k,8j+2}-R_{k,8j+1}+R_{k,8j}}$
               $\cdot c_{3k-3,3k-1}^{R_{k,8j+5}-R_{k,8j+4}-R_{k,8j+1}+R_{k,8j}} \cdot c_{3k-2,3k-1}^{R_{k,8j+6}-R_{k,8j+2}-R_{k,8j+4}+R_{k,8j}}$
               $\cdot c_{3k-3,3k-2,3k-1}^{R_{k,8j+7}-R_{k,8j+6}-R_{k,8j+5}-R_{k,8j+3}-R_{k,8j}+R_{k,8j+4}+R_{k,8j+2}+R_{k,8j+1}}$
```

Figure 3: Server computation for octal tree-based (n,1)-CPIR scheme

hand, the theoretical analysis fails to give an exact value for the actual speedup, for which we provide the actual implementation results in Section 5.

The most fundamental operation of the Damgård-Jurik encryption, on which an overwhelming proportion of server-side computations is spent, is modular exponentiation operation, which has quadratic complexity. Suppose that a 1024-bit modular exponentiation takes τ seconds (i.e., N is a 1024-bit number). The first exponentiations performed for the lowest level non-sink nodes ($R_{2,j}$) then are expected to take $\tau_2 = 4\tau$ seconds each since we work with modulo N^2. And the cost of exponentiation increases as we go up in the tree.

For every node of the binary tree, three exponentiations are performed. In quadratic and octal trees, we need five and nine exponentiations, respectively, for a node. For a node in the i^{th} level, we can adopt the following formula for the computation complexity, $t_i^b = 3 \cdot \tau_i$, $t_i^q = 5 \cdot \tau_i$, and $t_i^o = 9 \cdot \tau_i$, respectively for binary, quadratic and octal trees. Then, the overall time complexity of binary, quadratic, and octal trees can be estimated using the following formula $T = \sum_{i=2}^{m+1} r^{m+1-i} t_i$ for $m \geq 1$ where $r \in \{2, 4, 8\}$, $m \in \{m_b, m_q, m_o\}$, and m_b, m_q, and m_o are the number of levels in binary, quadratic, and octal trees, respectively. Employing the assumptions on the quadratic complexity of modular exponentiation operation with respect to bit length of the modulus in homomorphic encryption, we can compute an expected speedup values between different tree implementations. For instance, for $n = 512$, the octal tree implementation is expected to achieve a speedup of about 5.32 over a binary tree implementation. As we will show in Section 5, the actual speedup for this case is over 10. There are two reasons for this discrepancy. Firstly, we use asymptotic complexity of modular exponentiations which does not exactly give the actual execution time of the modular exponentiation for a specific operand length. Secondly, the big integer libraries employ specific optimization techniques for low bit sizes. As the bit size increases, it becomes difficult to use the same optimization techniques.

4.5 Analysis of Communication Complexity

A practical PIR scheme should be more efficient than the user downloading all the database (*the trivial solution*) in terms of the amount of information exchanged between the user and the server. Formally speaking, the bandwidth requirements of a PIR scheme must be sublinear to the size of the database. The bandwidth of the original (n, 1)-CPIR scheme based on binary decision trees has a logarithmic complexity. The proposed schemes based on quadratic and octal trees also have logarithmic complexities. However, the actual implementations of these three CPIR schemes have different bandwidth requirements, which are important in practice.

In PIR protocol, the client sends encrypted selection bits to the server in the first stage and receives the encrypted data item in the second stage. In binary decision tree, the number of selection bits is $\log_2 n$, where n is the number of data items in the database. Assuming $f_i < N$ for all data items and $|N|$ is the size of the modulus N, the size of the selection bit for the lowest level of the tree, $c_0 = E_{pk}(x_0)$, is $2|N|$-bit due to message expansion property of the Damgård-Jurik encryption. The selection bit for the second level $c_1 = E_{pk}^{(2)}(x_1)$, therefore, will be $3|N|$-bit long. In more general case, the selection bit for the i^{th}-level, $c_i = E_{pk}^{(i+1)}(x_1)$ will be $(i+1)|N|$-bit long.

The proposed CPIR schemes based on quadratic and octal trees require 3 and 7 selection bits for each level of the tree, respectively. This is less efficient than BddCpir, which requires only a single bit for one level. On the other hand, quadratic and octal trees are more shallow than binary trees; thus it is not immediately clear as to which scheme offers the best bandwidth efficiency. This calls for a more detailed inspection of bandwidth requirements of each scheme.

The binary, quadratic and octal trees have $\log_2 n$, $\log_4 n$, and $\log_8 n$ levels. The bandwidth requirements for the encrypted selection bits are given as in Table 1.

The size of the response, which contains the requested data

	Client → Server (# of bits)
Binary Tree	$[2+3+\ldots+(log_2 n+1)]\cdot\vert N\vert$
Quadtree	$[3\cdot(2+3+\ldots+(log_4 n+1))]\cdot\vert N\vert$
Octree	$[7\cdot(2+3+\ldots+(log_8 n+1))]\cdot\vert N\vert$

Table 1: The bandwidth requirements of the selection bits in different tree implementations

item in encrypted form, is also important since this is a part of the exchanged messages. The bandwidth requirements of the response message sent by the server to the user are $[log_2 n + 1]\cdot|N|$, $[log_4 n + 1]\cdot|N|$, and $[log_8 n + 1]\cdot|N|$ for binary, quadratic, and octal trees, respectively.

The overall communication cost sums up the number of bits exchanged for the selection bits and the response, which is tabulated in Table 2 for different database sizes. The quadratic tree always results in the minimum bandwidth requirements. The binary case is slightly better than octal tree for database sizes given in Table 2. However, the octal tree will eventually be better than the binary tree as the database size increases. For instance, for a database with $n = 4096$ data items, where each data item is 1 Kbit in length, the number of bits exchanged will be the same, namely 105472 bits, for both cases. The octal tree implementation will result in a better communication complexity for a database of more than $n = 4096$ data items.

n	Database size	binary	quadratic	octal
2	2048	4096	-	-
4	4096	8192	7168	-
8	8192	13312	-	16384
16	16384	19456	12288	-
32	32768	26624	-	-
64	65536	34816	31744	38912
128	131072	44032	-	-
256	262144	54272	48128	-
512	524288	65536	-	68608

Table 2: Actual costs of overall communication for different database sizes (in number of bits)

5. IMPLEMENTATION RESULTS

We implemented both the serial and the parallel versions of all CPIR schemes based on binary, quadratic, and octal trees using C++ with GMP library optimized for big number arithmetic. For parallel implementations we used OpenMP API that allow shared-memory multiprocessing programming. We used four parallel threads in our implementations and the platform is a computer featuring four cores, with hyper-threading support running Ubuntu 12.04 64 bit. Each core is an Intel i7 processor operating at 3.07 GHz. Finally, we used a 1024-bit modulus, providing 80-bit equivalent security, which is sufficient for PIR applications.

5.1 Client-Side Computations

The client performs encryption operations for building the secure indexes (i.e., encrypted selection bits) and one decryption operation to retrieve the requested data item. Encryptions are parallelized while the decryption, which is relatively simple operation, is performed in serial. For the three cases, the results are given in Table 3. As can be observed, the CPIR implementations based on quadratic and octal trees offer an obvious advantage over the binary tree implementation as far as the client side computation is concerned.

No. of Items	Client Encryption (ms)			Client Decryption (ms)		
	binary	quad	oct	binary	quad	oct
2	3	-	-	2	-	-
4	14	8	-	8	2	-
8	27	-	10	17	-	2
16	51	40	-	30	8	-
32	87	-	-	48	-	-
64	134	120	30	71	17	8
128	191	-	-	100	-	-
256	275	268	-	135	30	-
512	384	-	82	177	-	17

Table 3: Timings of client's selection bit encryptions and the decryption of the final result

5.2 Server-Side Computations

The server-side computations constitute the most time- and resource-consuming part of all CPIR schemes since all data items have to be processed before the requested one is selected out. Therefore, the computation complexity is directly a function of the database size. On the other hand, some of the involved operations are often independent and therefore, can be performed in parallel. In what follows, we present the timing results both for serial and parallel implementations and demonstrate that the proposed CPIR schemes take advantage of parallel processing.

5.2.1 Serial Case

In serial implementation, a single core is used to implement server side of the three CPIR schemes and the results are enumerated in Table 4. The table shows that we can achieve a speedup of up to $\frac{28000}{2550} = 10.98$ for a database with 512 data items. As the number of data items increases one should expect an increase in the speedup values as well.

Number of Items	Server Computation (ms)		
	binary	quadratic	octal
2	8	-	-
4	49	13	-
8	176	-	28
16	502	107	-
32	1,258	-	-
64	2,900	560	292
128	6,364	-	-
256	13,500	2,489	-
512	28,000	-	2,550

Table 4: Timings of server computation - sequential

5.2.2 Parallel Case

We developed two versions of parallel implementations. In the first implementation, we did not parallelize the two exponentiations in the Damgård-Jurik encryption operation; namely we performed the two operations $g^m \bmod N^{s+1}$ and

$r^{N^s} \bmod N^{s+1}$ in serial. However, these two exponents are independent and can be performed in parallel. In the second version of the parallel implementations we performed them in parallel and demonstrated that the second version is faster. In both versions, we used the four cores available in our platform for the implementations. The results for the first and the second versions are tabulated in Tables 5 and 6. In the first version, we achieved a speedup of $\frac{8323}{825} = 10.09$ over the binary tree implementation for a database of 512 data items over(*cf.* the last row of Table 5).

Number of Items	Server Computation (ms)		
	binary	quadratic	octal
2	8	-	
4	42	13	-
8	131	-	28
16	290	70	-
32	600	-	-
64	1,174	243	154
128	2,260	-	-
256	4,328	820	-
512	8,323	-	825

Table 5: Timings of server computation - parallel v1

The second version takes a better advantage of the parallelism in the server-side computations. Consequently, it provides a better timing results and an improved speedup values in comparison with those of the first version, as can be observed in Table 6. For a database with 512 data items, the second version is $\frac{825}{716} = 1.15$ times faster than the first version. For the same database, the speedup over the binary tree implementation that we achieve is $\frac{7654}{716} = 10.69$.

Number of Items	Server Computation (ms)		
	binary	quadratic	octal
2	5	-	-
4	33	7	-
8	100	-	13
16	240	49	-
32	502	-	-
64	1,003	199	97
128	1,992	-	-
256	3,885	740	-
512	7,654	-	716

Table 6: Timings of server computation - parallel v2

Obviously, parallel computation on shared-memory multicore computing platforms benefits all CPIR schemes and the benefit is more pronounced when the number of data items is high. For instance, with 512 data items, we can achieve a speedup of $\frac{2550}{716} = 3.56$ for octal tree when the speedup for binary tree implementation is $\frac{28000}{7654} = 3.65$. These results show that using octal tree in the CPIR scheme does not negatively affect the parallelism in the server-side computation in any significant way. Also, with CPIR schemes we cannot achieve the ideal speedup, which is equal to the number of cores in the computing platform, since the parallelism becomes weaker in the topmost levels of the decision tree, where the encryption operation is the hardest.

Finally, from the binary tree serial implementation to octal tree parallel implementation the achieved speedup is $\frac{28000}{716} =$ 39.11. This is an important improvement that enables the practical use of CPIR schemes.

Our preliminary theoretical analysis shows that the proposed schemes show weak scalability in parallel implementations. Namely, using more computational power (i.e., higher number of processor cores) benefits larger databases with more data items. On the other hand, higher number of cores can also be beneficial for databases with moderately small sizes. For instance, using eight cores is expected to accelerate further the server-side computations for a database with 512 data items. Since processors with more cores are not common, we leave the verification of our claims about scalability of the proposed schemes as future work.

6. LITERATURE ON PIR SCHEMES AND COMPARISON

There is a relatively high academic interest in efficient PIR schemes [1–6,8–11,14,17]. We compare the proposed schemes against two more recent schemes in the literature [1, 2, 8], both of which utilize lattice-based cryptography. The former lattice-based scheme introduced in [1, 2], claim computational efficiency while the latter [8], which utilizes fully homomorphic encryption (FHE), claims superior bandwidth performance over the former while accepting the former is computationally much more efficient. We demonstrate that our proposed scheme is always superior so far as the bandwidth efficiency is concerned while computational efficiency of our scheme is comparable to or better than that in [8], but worse than that in [1, 2]. However, we also show that the scheme in [1, 2] can have such a poor bandwidth performance that it is sometimes better to download the entire database in many circumstances, as also pointed out in [13].

CPIR schemes based on decisional trees use the Damgård-Jurik cryptosystem that is based on the decisional composite residuosity assumption [15], which is a relatively well studied classical problem in comparison with those security arguments used in lattice-based solutions, especially the one in [1, 2].

We compare the bandwidth requirements of the proposed octal tree based CPIR and two other techniques when $n = 512$, and tabulate the results in Table 7, which lists the ratio of exchanged information in a run of the scheme to the database size in each scheme. As can be observed in the table, the proposed method always results in superior bandwidth performance. The lattice-based scheme in [1, 2] requires the transmission of fewer number of bits than the database size only after the size of the database reaches 128 Mbit. The scheme based on FHE never offers better performance than transmitting the entire database. The FHE-based scheme bandwidth requirements will be acceptable only for databases with many data items. For instance, for a database with 2^{16} items where each data item is 1024-bit, the ratio of exchanged data to database size in the FHE-based PIR scheme is 0.53, while it is only 0.03 in the proposed scheme for the same setting. For server-side computations, the lattice based scheme [1, 2] is reported to offer 230 Mbit/s for a database with only 12 data items, each of which is 3 MB. The proposed method offers 715 Kbit/s for a database with 512 data items. FHE-based PIR scheme reports two time performance metrics: i) throughput when

Data item size (# of bits)	Database size (# of bits)	[1,2]	[8]	Proposed method
1 K	512 K	224	67.21	0.135
32 K	8 M	14.01	7.96	0.016
128 K	64 M	1.76	4.42	0.009
256 K	128 M	0.88	4.16	0.008
2 M	1 G	0.11	3.94	0.008

Table 7: Ratio of exchanged information to database in different PIR schemes

multiple requests are bundled into a single query, hence the *bundled* case, and ii) latency when a request is sent alone (*single* case). In the bundled case for data items of 1024-bit long each, the time spent for processing a data item is given as 0.89 ms while it is 1.4 ms in our scheme. On the other hand, for the latency metric indicating the waiting time for a user, (which is what matters most for the user) the time spent for processing a data item is 16.93 ms.

7. CONCLUSION

We proposed and implemented improved and parallel versions of CPIR protocol by Lipmaa (BddCpir). We offered the utilization of quadratic and octal trees and demonstrated that the new version is about 10 times faster than the original BddCpir protocol in terms of server-side computations. In addition, we also provided a parallel algorithm for the server-side computations, that takes advantage of shared-memory multi-core processors and demonstrated that we can achieve a speedup of 3.56 with four cores. Our implementations show that the overall speedup of the new scheme with four cores for a database size of 512 Kbit is 39.11 over the original scheme with a straightforward serial implementation. The gain with the parallel algorithm is likely to be higher if more cores are used for larger databases.

We compared the proposed scheme with the schemes in the literature in terms of bandwidth requirements and found out that the new scheme provides bandwidth efficiencies, which are better from than those of the other schemes by one to three orders of magnitude. Also, the adopted security assumption in our scheme is well studied in comparison with the alternative schemes; another reason for further interest in the proposed scheme.

8. ACKNOWLEDGMENTS

This work is supported by Turk Telekom under Grant Number 3014-07.

9. REFERENCES

[1] Aguilar-Melchor, C., Gaborit, P. "A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol", In *WEWORC 2007*, July 2007.

[2] Aguilar-Melchor, C., Crespin, B., Gaborit, P., Jolivet, V., Rousseau, P. "High-Speed PIR Computation on GPU", In *SECURWARE'08*, pp. 263-272, 2008.

[3] Ambainis, A., "Upper bound on the communication complexity of private information retrieval", In *Proc. of the 24th ICALP*, 1997.

[4] Cachin, C., Micali, S., Stadler, M., "Computationally Private Information Retrieval with Polylogarithmic Communication", In *EUROCRYPT 99*, pp. 402-414, 1999.

[5] 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, 1995.

[6] Chor, B., Gilboa, N., "Computationally Private Information Retrieval", In *29th STOC*, pp. 304-313, 1997.

[7] Damgård, I., and Jurik, M., "A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System", In *Public Key Cryptography*, pp. 119-136. Springer Berlin Heidelberg, 2001.

[8] Doröz, Y., Sunar, B., and Hammouri, G., "Bandwidth Efficient PIR from NTRU", In *Workshop on Applied Homomorphic Cryptography and Encrypted Computing, WHAC'14*, 2014.

[9] Gentry, C., Ramzan, Z., "Single-Database Private Information Retrieval with Constant Communication Rate", In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, pp. 803-815, 2005.

[10] Ishai, Y., Kushilevitz, E., "Improved upper bounds on information-theoretic private information retrieval", In *Proc. of the 31th ACM Sym. on TC*, 1999.

[11] Kushilevitz, E., Ostrovsky, R., "Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval", *FOCS '97*, 1997.

[12] Lipmaa, H., "First CPIR protocol with data-dependent computation", In *Information, Security and Cryptology ICISC 2009*, pp. 193-210. Springer Berlin Heidelberg, 2010.

[13] Olumofin, F., and Goldberg, I., "Revisiting the computational practicality of private information retrieval", In *Proceedings of the 15th international conference on Financial Cryptography and Data Security*, pp. 158 - 172, 2012.

[14] Ostrovsky, R., Shoup, V., "Private Information Storage", In *29th STOC*, pp. 294-303, 1997.

[15] Paillier, P., "Public-key cryptosystems based on composite degree residuosity classes", In *Advances in cryptology, EUROCRYPT'99*, pp. 223-238. Springer Berlin Heidelberg, 1999.

[16] Rabin, M. O., "How to exchange secrets by oblivious transfer", *Technical Report TR-81*, Aiken Computation Laboratory, Harvard University, 1981. available at `http://eprint.iacr.org/2005/187`.

[17] Sion, R., Carbunar, B., "On the Computational Practicality of Private Information Retrieval", In *NDSS07*, 2007.

[18] Wiesner, S., "Conjugate coding", *Sigact News*, vol. 15, no. 1, pp. 78 - 88, 1983.