International Journal of Soft Computing 12 (5-6): 294-302, 2017
ISSN: 1816-9503

Private Information Retrieval in Fuzzy Search Environments

Eric Bagalwa and Okuthe P. Kogeda
Department of Computer Science, Tshwane University of Technology,
Private Bag x680, Pretoria, South Africa

Abstract: Decades of research have resulted in privacy-preserving theories to carry out any computational tasks but there is still a wide gap between theory and practice. This study presents a method for privately retrieving data from an unsecure server. The method used is based on a Private Information Retrieval (PIR) scheme that utilizes Euler's Phi Hiding Assumption. The method is optimized to offer fuzzy search abilities while retaining user's queries privacy. A pre-processing phase is added at the server to reorganize data into buckets. A locality sensitive hashing function is used to create clusters based on string approximation. Similar data are organized and stored into buckets. The PIR query is no longer the target keyword as in typical PIR algorithms but rather the bucket containing the targeted keyword. The method is tested using a prototype written in PHP and C++ to measure performance and accuracy. The result is a system that successfully and privately retrieves a cluster of approximate data matches rather than exact matches. The algorithm is computationally more efficient than the original scheme in the sense that bucketization reduce the number of items to be retrieved which makes the process much faster.

Key words: Private Information Retrieval (PIR), fuzzy search, data bucketization, locality sensitive hashing, efficient, unsecure server

INTRODUCTION

With the development of publicly available databases that are easily accessible throughout the internet and that are being managed by third party servers, several concerns about user's privacy have been raised. One of these concerns is related to confidentiality of queries that are posted when users request information from databases (Chor *et al*., 1998).

Database operators could easily track and analyze user's queries and infer confidential information about their intents, businesses or personal lives. Users are therefore increasingly anxious that their search activities might reveal confidential information about themselves (Pang *et al*., 2010).

PIR protocols aim to retrieve information from public or private databases, without revealing to the database servers which record has been retrieved (Beimel and Stahl, 2002).

The PIR problem was first formulated by Chor *et al*. (1998). In the past 15 years, several solutions to the PIR problem both theoretical and computational have been documented in literature. Starting with a trivial solution where a user makes a local copy of the entire database and only queries the local copy (Chor *et al*., 1998) to more robust solutions relying on hard mathematical problems. In a typical PIR system, a server holds a database B. This database is regarded as an aggregation of single bits, stacked one after the other. When a user wants to retrieve a bit B_i from the database, the user sends the index i to a query generator algorithm that encrypts i; generate a query q and a private key p_k. Then, the user sends the query q to the server that uses it with a database algorithm that runs against all entries to generate the response r as output. The response is sent back to the client and is decrypted using a private key p_k.

In practical settings, however, users are expected to retrieve more than a single bit for each request; records are not referenced by index number but rather by string tags (Young and Moti, 2004); users are not expected to know the list of all the tags that are available for search in the database and it is possible to misspell keywords. A good information retrieval system should leave room for data exploration. Instead of exact keyword match to database entries, retrieval system should compute and propose a list of similar, approximate and available query terms that are likely to be relevant to the requested query.

A solution that both protects the privacy of queries and offers approximate data matching is of importance for a variety of reasons. Be it to protect a cautious individual

Corresponding Author: Eric Bagalwa, Department of Computer Science, Tshwane University of Technology, Private Bag x680, Pretoria, South Africa

who wants total privacy when surfing on the net or to protect a company concerned about the confidentiality of projects they are currently searching information for such solution contributes in building a safe online society.

We present an experimental implementation of a computational private information retrieval scheme based on the Euler's Phi Hiding Assumption that offers Fuzzy search capabilities. Our solution is computationally secure as opposed to theoretical PIR that are theoretically secure.

Theoretically, secure solutions require more than one copy of a database to be hosted in more than one server. Information theoretically secure solutions are unbreakable under any circumstances while computationally secure solutions are only secure under certain computational intractability assumptions. Computational PIR are based on hard mathematical problems. They remain secure for as long as the mathematical assumption behind them remain unsolved or hard to solve (Young and Moti, 2004).

The system offers fuzzy search capabilities by reorganizing data in the server into buckets prior to the data retrieval process. Each bucket contains similar index search terms. A Locality Sensitive Hashing (LSH) function is used to group similar search strings together. LSH are special functions in which similar items are more likely to be hashed to the same bucket than dissimilar items (Chakrabarti, 2003).

The implementation returns, for every input string, an integer value that corresponds to the input string. Two similar strings would hash into two close values which make it simple to group into the same or close buckets.

Importance of the study: Decades of research in cryptography and security have resulted in an elegant theory that shows in principle, how to perform almost any computational task in a privacy-preserving manner but there is a wide gap between theory and practice: currently, most people and organizations have no straightforward way to safeguard the privacy of their sensitive data in realistic networked environments. In the interest of bridging this gap, we design and implement a PIR protocols to enhance user's queries privacy.

Our PIR implementation can find application in a variety of situations. It can be used to search a patent database without revealing to its owner the intellectual property of interest; it can allow an investor to query a stock-market database for the value of a certain stock without revealing the identity of the stock; it can be used to prevent database administrator from profiling their users based on their search queries with the purpose of selling that information to certain companies.

Literature review: Personal information about an individual that queries leak is phenomenal. For example, it is natural for individuals to do some research about their health, especially when they have been diagnosed with a condition. They do so by querying specific keywords related to their situation. On the other hand, health information of customers is a valuable asset in the decision-making of insurance companies. Thus, if an information broker collects and sells these confidential information, it is definitely an infringement of user privacy (Tsan-Sheng *et al*., 2002).

Customer's behavior and queries could be used for commercial profiling or governmental surveillance purposes. It is common practice for online search engines to profile their users by storing and analyzing past searches (Roca *et al*., 2009). These user's queries are kept for several months or years. Google, for example, keeps user's search logs for about 18-24 months (Ye *et al*., 2009).

Protection of user's privacy against unsecure servers is not a recent problem. Pang *et al*. (2010) proposed a method that embellishes text search queries with decoy terms to confuse the server in knowing exactly what the user is looking for. The method generates a set of terms that are similar in form but not in meaning and uses them as decoy. TrackMeNot (Marx, 2003), a light weight web service based on Marx's research study, protects user queries from search engines by hiding them among randomly generated 'ghost' queries. The method attempts to protect the user's privacy by issuing computer-generated random queries. The objective is to make it hard for the search engine to distinguish real user queries from the computer generated "cover traffic". Users can be identified by IP address but what they search for is obscured to some extent by noise. Anonymity Network Tor protects user's privacy by using a network of anonymizing proxies. The source of the connection to the search engine is rotated periodically among the routers in Tor. Also, connections are routed through several Tor routers and encrypted in such a way it is impossible to determine the true source.

One of the most effective solutions that guarantee user's queries privacy has been proposed by Chor *et al*. (1998) back in 1995. They were the first to articulate the PIR problem. In its most basic form, the PIR problem is for one party to retrieve the entry in a database that is maintained by one or more other parties without revealing which entry is sought. Over the years, several solutions to the PIR problem have been proposed in literature; starting with the trivial solution proposed by Chor *et al*. (1998) where the entire database is locally duplicated to the client. While this trivial solution raises communication complexity concerns and is considered to be unpractical due to the amount of data needed to be communicated

between server and clients, it presents the advantage of achieving maximum privacy with a linear communication complexity.

To reduce communication complexity, Beimel and Ishai (2001), Ishai and Kushilevitz (1999) proposed information theoretic solutions that let users access replicated copies of the same database. In their implementations, several copies of the same database are distributed over multiple independent servers that do not interact with each other. Information theoretic solutions are able to achieve a communication complexity of $O(n^{1/3} \log l)$ where l is the number of duplicated servers and n the number of records the database has Ishai and Kushilevitz (1999) achieved a communication complexity of $O(n^{1/3})$ with 2 servers.

Due to the impracticability of maintaining several copies of the same database on several independent servers, single server PIR algorithms relying on mathematical problems have been proposed starting with Eyal and Ostrovsky's solution (Kushilevitz and Ostrovsky, 1997).

One such mathematical hard problem is Euler's Phi Hiding assumption. Euler's Phi Hiding Assumption states that if p_1 and p_2 are primes exactly one of which divides $\varphi(n)$ where n is a number whose factorization is unknown and φ is Euler's totient function, then there is no polynomial-time algorithm to distinguish which of the primes p_1 and p_2 divides $\varphi(n)$ evenly with a probability significantly greater than 1/2 (Schridde and Freisleben, 2008). An implementation based on this assumption has been experimented by Young and Moti (2004) with polylogarithmic communication complexity.

Kushilevitz and Ostrovsky (1997) presented the first single database computational PIR (cPIR) scheme where the security is established against a computationally bounded adversary. Their scheme is based on the quadratic residuosity assumption. Using the quadratic residuosity problem (Kushilevitz and Ostrovsky, 1997), they were able to implement a single-database computational PIR scheme with a $O(n^c)$ communication complexity (c>0).

Early PIR algorithms considered the database to be a simple aggregation of single bits associated with an integer index i. In practical settings, however, a user would want to retrieve a record, not a single bit. Furthermore, users are more interested in querying by keyword rather than by an index number. Young and Moti (2004) presented a Tagged Private Information Retrieval (TPIR) scheme that instead of retrieving data bit by bit, retrieves records associated to a tag keyword. The Scheme uses Euler's Phi Hiding assumption to hide a prime tailored after the tag keyword. The scheme retrieves data in fixed-sized block rather than single bits.

Approximation string matching for PIR is a relatively new concept. Related works close to this concept include the work by Ghinita *et al*. (2008) that designed a scheme to privately retrieve data for location based services. The aim was to retrieve location sensitive information from an unsecure server while protecting the location of the user. The scheme uses cryptographic PIR techniques and partitions space into regions. The user finds the region that contains him and utilizes PIR to request all points within the region. The result is a system that hides user's location from the server. Olumofin *et al*. (2010) proposed a similar PIR scheme for retrieving location based information on a cellphone without revealing the location of the client. Although, these schemes are not exactly doing string approximation, they are similar to our implementation because they return approximate location information while hiding the location of interest.

The particularity of our implementation is that we propose to provide search string approximation along with the protection of user's intents. We have set an environment for which the PIR problem needs to be solved. We want our solution to be adequate for fuzzy search environments. We want to provide suggestions along with exact string match. The solution that we propose should allow users to browse the content of the database while hiding their intents from the database owner.

MATERIALS AND METHODS

System design and architecture: Figure 1 portrays the major component blocks that constitute our solution. At a very high level, our implementation is made of three operational blocks: a query generator, a database

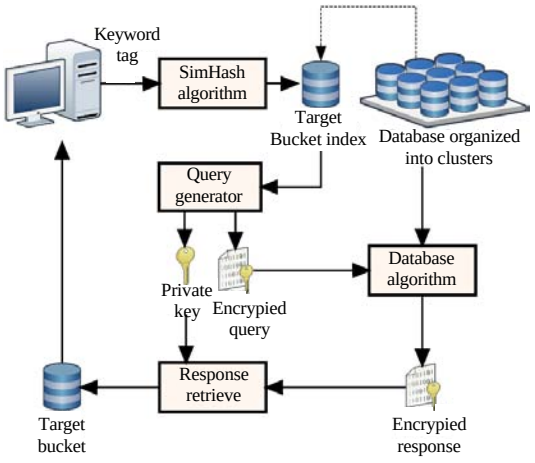

Fig. 1: System architecture

algorithm and a response retriever. Both query generator and response retriever runs on client side while the database algorithm runs on the server.

A short pre-processing phase is needed at the server to reorganize similar terms into buckets. Bucketization is a common technique for fast search in information retrieval systems.

The query generator uses a SimHash algorithm which is designed to hash approximate strings into similar buckets. The user input a string S to the SimHash algorithm the algorithm runs on the input string S and produces an integer output i. Similar input strings produce output integers that are close to each other and that can be grouped in buckets using a simple integer division.

The query generator uses the bucket number n that supposedly contains the searched item as its new query. It then encrypts the query n in such a way that the bucket b_n can be privately retrieved from the server. The output of this algorithm is both a concealed query q that hides n from the server and a secret private key p_k needed to decrypt server's response.

The database algorithm takes query q as an input and runs against all entries in the database. The result of this computation is a response r that is sent back to the client.

The client uses its secret key p_k along with the response r as an input to its response retriever algorithm to extract and decrypt the response to query q. The response is the entire bucket that contains the desired item. Bucket items are used as alternate suggestions to the initial search term and thus providing fuzzy search.

Implementation

PIR based on the Phi-hiding: One of the main issues that any PIR algorithm has to deal with is the amount of computation that each query requires when retrieving a single item from the database. To preserve the privacy of queries, the server has to perform a certain amount of computation on every record the database has. The complexity of these computations determines how fast it would be to retrieve items from the database. A PIR algorithm that takes 15 min to retrieve a single element will be of no good value.

At the heart of our solution, a computational PIR algorithm based on Euler Phi-Hiding problem has been utilized. The implementation is based on a Tagged Private Information Retrieval scheme proposed by Young and Moti (2004). The solution works in two-rounds and achieves both polylogarithmic communication complexity and polylogarithmic user computational complexity (Chor and Gilboa, 1997).

Communication complexity refers to the amount of data that get sent from the server to the client each time a query is executed. In the trivial solution, communication complexity is always equal to O(n) where n is the number of bits inside the database. A PIR algorithm is said to have a polylogarithmic communication algorithm if its communication complexity is in $\log_n$ order of magnitude.

Computational complexity refers to how much computation is required to execute a single query. Users are required to execute a query generator and response retriever algorithm that both execute in time $\log_n$ proportional to the number of items in the database.

Our implementation improves computational complexity due to the fact that items are reorganized into buckets. This reduces the number of elements n to retrieve. We can't say the same for communication complexity. Even though n is decreased, there are more elements stacked in each bucket which tend to increase communication complexity. Our solution is based on Euler Phi-Hiding assumption that is defined here.

Definition: We define the Euler totient function $\varphi(m)$ (where $m \in N^*$) as a function that returns the number of natural integer $x \in N^*$ where $x<m$ and is coprime to m. We say that two numbers are coprime to each other if and only if their gcd (great common divisor) is one. In other terms, if they only share one as common divisor. For example, 4 and 7 are said to be coprime because they only have 1 as common divisor. $\varphi(6) = 2$ because only 1 and 5 are coprime to 6. Similarly, $\varphi(35) = 24$ because 35 is coprime with 1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 26, 27, 29, 31, 32, 33 and 34.

We say that a composite integer m φ-hides a prime p if p divides $\varphi(m)$ evenly. The Phi-hiding assumption states that it is computationally intractable to decide whether a given small prime $p>2$ divides $\varphi(m)$ where m is a composite integer of unknown factorization.

The intuition behind the scheme is as follows: an algorithm is given that allows a sufficiently large prime number p_j to be constructed deterministically for each database entry j. Hence, database elements 1, 2, 3, …, n in the database would correspond to primes $p_1, p_2, p_3, \ldots, p_n$. With overwhelming probability these primes would be distinct. When a user want to retrieve a bit B_i from a database B without revealing i, the algorithm constructs a composite m which is tailored after the prime p_i that was deterministically generated for element i. The composite m is crafted in such a way that p_i divides $\varphi(m)$ evenly, yet m is difficult to factor even when p_i is known. A random integer x_0 is then selected from Z^*_m such that x_0 does not have p_i^{th} roots modulo m. A number that does not have p_i^{th} roots modulo is interpreted as a binary zero.

The database algorithm is supplied with x_0 as well as the composite m. The administrator uses the database algorithm to compute primes $p_1, p_2, p_3, \ldots, p_n$. To retrieve the desired item, the administrator computes $x_j = x_{j-1}^{p_j^{B_j}} \bmod m$ and returns x_n to the user (where B_j is the database bit

at location j; p_j is the prime that correspond to element j). The response retriever algorithm examines the response x_n. x_n will not have p_i^{th} roots modulo m unless $B_i = 1$.

PIR implementation: The PIR algorithm has three components: a query generator, a database algorithm and a response retriever.

QueryGenerator (n, i, 1^k):

Input: integer n (number of bits in the database B)
integer i satisfying $1 \le i \le n$
Output: a query q = (m, x, y) where:
m is a composite that is k^f bits in length
x contained in Z_m^*
y which is a k-bit string
secret s which is the factorization of m
1. Generate a random k-bit string y
2. Set I to be the (log n)-bit representation of i
3. Compute $p_i = RandPrime_k(y||I)$
p_i is a prime number of length k that is deterministically generated using y and I. $RandPrime_k$ is the function that produces p_i
4. Compute $(Q_1, Q_2) = PhiHide(f, p_i, 1^k)$
5. Compute $m = Q_1Q_2$
6. Choose x randomly from Z_m^*
7. Output q = (m, x, y) and s and halt s is the secret which is the factorization of m:s (Q_1, Q_2)

The algorithm PhiHide is used to ϕ-hide p_i within the composite m.

Phi-Hide (f, p_i, 1^k):

Input: integer f, k-bit prime p_i
Output: the factorization s = (Q_1, Q_2) of a composite $m = Q_1Q_2$ where m is k^f bits in length
1. Repeatedly choose a random (k^f-k)-bit integer q_1 until
$Q1 = p_i q_1+1$ is prime
2. Choose a random k^f-bit prime Q_2
3. Output s = (Q_1, Q_2) and halt

Database algorithm (B, q, 1^k):

Input: database $B = b_1b_2, ..., b_n$ consisting of n bits
Query q = (m, x, y)
Output: a response r contained in Z_m^*
1. Set n = |B| (n = number of bits or elements in B)
2. Set $x_0 = x$
3. For j = 1 to n do:
4. Set J to be (log n)-bit representation of j
5. Compute pj = Rand $Prime_k$ (y||J)
6. Compute $e_j = p_j^{b_j}$
7. Compute $x_j = x_{j-1}^{e_j} \bmod m$
8. Output $r = x_n$ and halt

Response retriever (n, i, (q, s), r, 1^k):

Input: integer n (number of bits in the database B)
integer i satisfying $1 \le i \le n$
query q = (m, x, y)
secret factorization s = (Q_1, Q_2)
response $r \in Z_m^*$ obtained from DatabaseAlgorithm
Output: a bit b such that $b = b_i$ with overwhelming probability
1. Set I to be the (log n)-bit representation of i
2. Compute $p_i = RandPrime_k(y||I)$
3. Compute $t = (Q_1 - 1)(Q_2 - 1)/p_i$
4. Compute $w = r^t \bmod m$
5. Set b = 0
6. if w = 1 then set b = 1
7. Output b and halt

To illustrate the usage of the above algorithms, we are going to use a dummy example. Let us say we have a database with 4 bits: B = 1011; in case 1, the user wants to retrieve i = 3 and in case 2, he wants to retrieve i = 2. Let us have the set of primes {7, 5, 3, 11} corresponding to the four elements in the database B, the security parameter k and f are set to 1.

Case 1:

QueryGenerator (4, 3)
i = 3; n = 4
$m = Q_1*Q_2$
$p_3 = 3$
$Q_1 = p_3*q_1+1$ (must be a prime)
Q_2 must be a prime
Lets select q_1 to be 2 which means $Q_1 = 7$ and let Q_2 be 5
$m = Q_1*Q_2 = 35$
s $(Q_1, Q_2) = (7, 5)$
Test: $\varphi(m) \bmod p_3$ must be equal to zero
$\varphi(35) \bmod 3$ is equal to 24 mod 3 which is equal to zero
Finally, lets select x contained in Z_{35}^* to be 6
Our query is q (35, 6)

Database algorithm (B(1011), q(35, 6)):

m = 35; n = 4; $x_0 = 6$
for j = 1 to 4
$r = x_n$ which means $r = x_4 = 6$
Response Retriever (4, 3, (q (35, 6), s (7, 5)), 6):
r = 6; n = 4; i = 3; $Q_1 = 7$; $Q_2 = 5$; $p_i = p_3 = 3$

$$t = \frac{(Q_1-1)(Q_2-1)}{p_i} = \frac{(7-1)(5-1)}{3} = 8$$

w is equal to 1 which means that b_3 (Element of the database at position 3) is bit 1

Case 2:

QueryGenerator (4, 2)
i = 2; n = 4
$m = Q_1*Q_2$
$p_2 = 5$
$Q_1 = p_2*q_1+1$ (must be a prime)
Q_2 must be a prime
Let's select $q_1 = 2$ which means $Q_1 = 11$ and $Q_2 = 5$
$m = Q_1*Q_2 = 55$
s $(Q_1, Q_2) = (11, 5)$
Test: $\varphi(m)$ must be equal to zero
$\varphi(55) \bmod 5$ is equal to 40 mod 5 which is equal to zero
Finally, let's select x contained in Z_{55}^* to be 3
Our query is q (55, 3)

Database algorithm (B(1011), q (55, 3)):

m = 55; n = 4; $x_0 = 3$
for j = 1 to 4
$r = x_n$ which means $r = x_4 = 47$
Response Retriever (4, 2, (q(55,3), s(11,5)), 47):
r = 47; n = 4; i = 2; $Q_1 = 11$; $Q_2 = 5$; $p_i = p_2 = 5$

$$t = \frac{(Q_1-1)(Q_2-1)}{p_i} = \frac{(11-1)(5-1)}{5} = 8$$

$w = r^t \bmod m = 47^8 \bmod 55 = 16$
w is not equal to 1 which means that b_2 (Element of the database at position 2) is bit 0

Table 1: Database algorithm runs

j = 1	j = 2	j = 3	j = 4
$p_1 = 7$	$p_2 = 5$	$p_3 = 3$	$p_4 = 11$
$e_1 = 7^1$	$e_2 = 5^0$	$e_3 = 3^1$	$e_4 = 11^1$
$x_1 = x_0^{e1} \bmod m$	$x_2 = x_1^{e2} \bmod m$	$x_3 = x_2^{e3} \bmod m$	$x_4 = x_3^{e4} \bmod m$
$x_1 = 6^7 \bmod 35$	$x_2 = 6^1 \bmod 35$	$x_3 = 6^3 \bmod 35$	$x_4 = 6^{11} \bmod 35$
$x_1 = 6$	$x_2 = 6$	$x_3 = 6$	$x_4 = 6$

Table 2: Database algorithm runs

j = 1	j = 2	j = 3	j = 4
$P_1 = 7$	$P_2 = 5$	$P_3 = 3$	$P_4 = 11$
$e_1 = 7^1$	$e_2 = 5^0$	$e_3 = 3^1$	$e_4 = 11^1$
$x_1 = x_0^{e1} \bmod m$	$x_2 = x_1^{e2} \bmod m$	$x_3 = x_2^{e3} \bmod m$	$x_4 = x_3^{e4} \bmod m$
$x_1 = 3^7 \bmod 55$	$x_2 = 42^1 \bmod 55$	$x_3 = 42^3 \bmod 55$	$x_4 = 3^{11} \bmod 55$
$x_1 = 42$	$x_2 = 42$	$x_3 = 3$	$x_4 = 47$

In practical applications primes that are selected are big numbers in the order of 128-256 bits for security reasons (Table 1 and 2).

SimHash (locally sensitive hashing): In computer science, approximate string matching (often colloquially referred to as fuzzy string searching) is the technique of finding strings that match a pattern approximately (rather than exactly). The problem of approximate string matching is typically divided into two sub-problems: finding approximate substring matches inside a given string and finding dictionary strings that match the pattern approximately (Brin, 1995).

SimHash or similarity hashing is a technique of hashing strings such that similar strings end up close to each other. SimHash was developed by Charikar (2002) and is useful when clustering similar data together. The simhash of a string is calculated as follows:

SimHash (y):

Input: string y to be hashed
Output: a numeric value x that represent the hashed version of the input string

1. pick a hash size, let's say 32 bits
2. let V be an array of size 32 initialized with 0s
3. break the input string into small features
4. hash each feature using a normal 32-bit hash algorithm
5. for each hash:
   a) If bit_i of hash is set then add 1 to V[i]
   b) If bit_i of hash is not set then take 1 from V[i]
6. simhash bit_i is 1 if V[i]>0 and 0 otherwise

We have used the above algorithm to cluster a sample of close strings. Here are the results after sorting by the simhash column. The cluster number is obtained by dividing and rounding the hash value by 100,000,000. From Table 3, we can see that similar words end in the same bucket.

Phi-hiding PIR for fuzzy search environments: In this study, we provide a description of our implementation. The application has two components: a client component and a server component. Both components are divided into three layers: storage which is the bottom layer, a backend application written in C++ and a front end application designed to provide web access (Fig. 2 and 3).

Table 3: SimHash clustering

Input strings	SimHash	Cluster
Annimation	245833192	2
Piedestrian	278036687	3
Pedestran	335817807	3
Pdestrian	336825423	3
Pedestrian	345418975	3
Pedestrian	345418975	3
Overflow	1431863692	14
Flowering	1442611213	14
Overfloing	1465424141	15

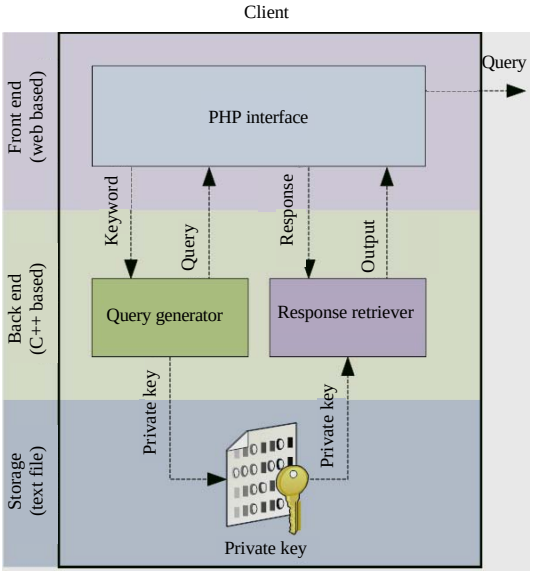

Fig. 2: Client component

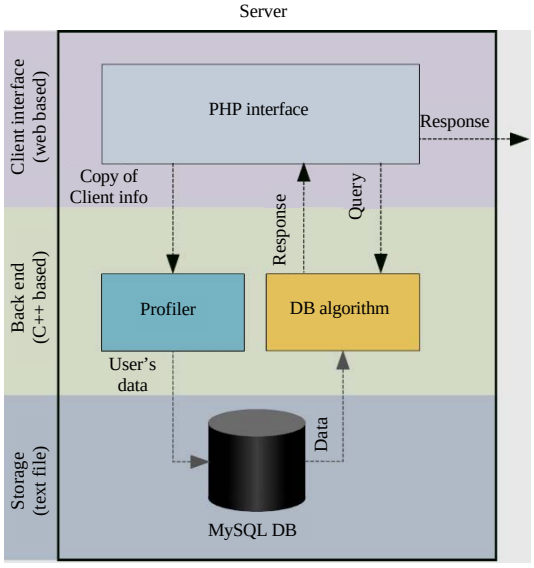

Fig. 3: Server component

The client component has two algorithms: query generator and response retriever. The query generator has been written in C++ to take advantage of the bignumber library that allowed us to perform computation on very large numbers. The query generator receives the searched keyword from the top layer which is designed in html/PHP. The query generator computes the desired bucket number by simhashing the keyword. It then generates query elements using the Phi-hiding PIR function. It, at the same time, outputs a secret key that is stored in the bottom layer. The top layer receives the query and posts it to the server (Fig. 2).

The server receives the query through its PHP interface, it then send the query to the backend DB Algorithm C++ application which uses it along with the MySQL database at the bottom layer to compute a response. We have modified the database algorithm to retrieve, not a single bit as in the implementation discussed in this study. But multiple bytes at a time. We have incorporated a profiler component at the backend layer to make copies of the entire request coming from users. The response is sent back to the client via the top web layer (Fig. 3).

The client receives the response via its web interface and uses the Response retriever algorithm to decipher it and to send the result to the top layer. The top layer interprets the result, extract suggestions and present it to the requester.

RESULTS AND DISCUSSION

The result of our implementation is a system that can privately and reliably retrieve items from an unsecure database. The system provides suggestions of relevant terms while searching for matchs. The time it takes to retrieve an item and the size of the database are important variables used to measure the performances of the solution. We also compared the performances of our implementation against the original Phi-hiding PIR algorithm. Two computers were used to experiment with the solution. We ran our server on a computer that uses a Core(TM) i5-2400 Intel Processor clocked at 3.10 GHz. The client was ran on a laptop that is powered by a Core(TM) i5-3210M Intel Processor clocked at 2.50 GHz.

For our data, we used a MySQL database hosted in an apache server. Our sample database was made of several tables with different set size of items.

Data bucketization: We first measured the performance of our Data Bucketization. It is done by a php script that accesses a MySQL table and reorganizes items into buckets using our simHash function. The number of buckets and bucket sizes depend on a divider d used to normalize hash values into bucket indexes. When we set d = 10000000, we get the following results (Fig. 4).

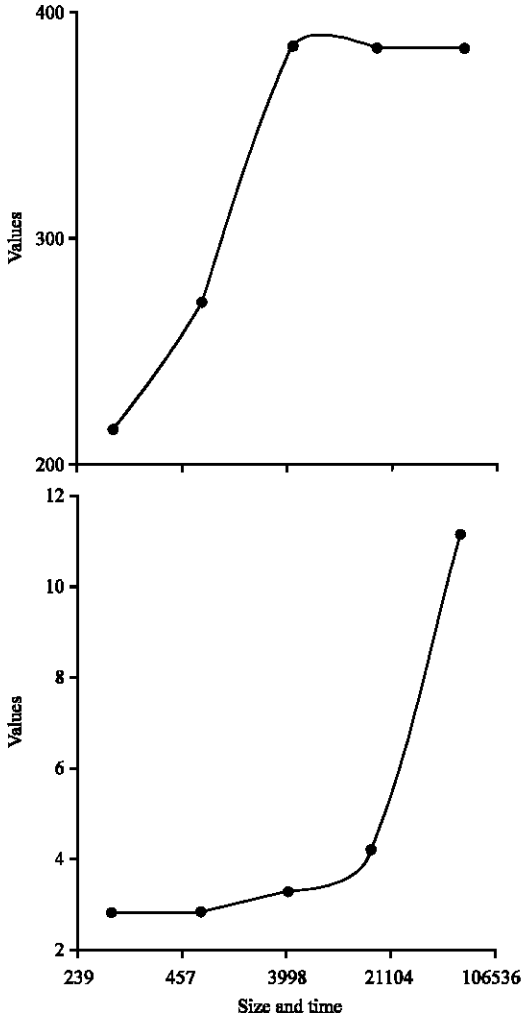

Fig. 4: Bucketizationa

PIR algorithm: We have compared the execution time of our approach to the original Phi-hidding PIR approach (Fig. 5). Our approach improves drastically the execution time due to the fact that the number of clusters remains relativaly low, even when the number of items in the database increases. In the original PIR algorithm, the packet size is always constant and depend on the set i = maxByteP erRecord which is the maximum number of bytes that is retrieved for each query execution. In our implementation, the packet sizes increases as more items are filled into buckets. An optimal compromise between

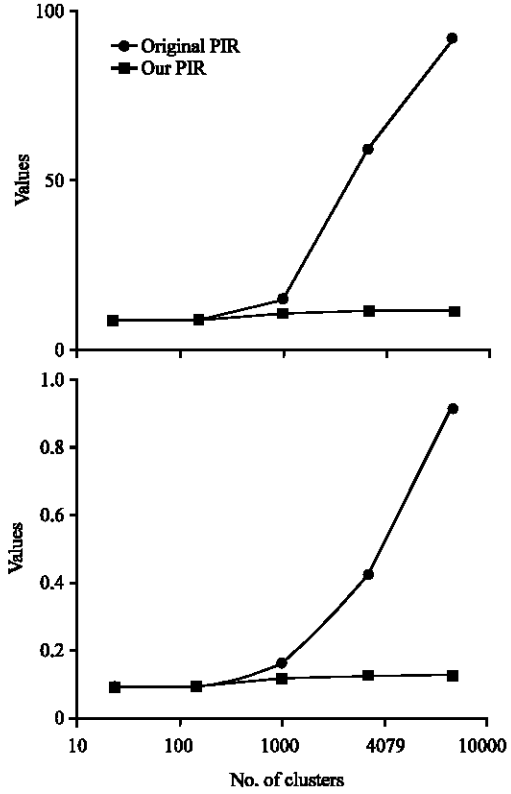

Fig. 5: PIR algorithm

execution time and packet size can however be obtained by tuning divider parameter d which effectively controls the size and the number of buckets.

CONCLUSION

In this research paper, we have proposed an algorithm that achieves a usable compromise between privately retrieving information from an unsecure server and providing benefit of a fuzzy search system. The efficiency of the system in terms of time required to retrieve items from the database is proportional to the size of the database. The system is usable for small to medium sized databases and becomes inadequate on very large databases. Further work should be devoted in optimizing the three PIR algorithms in order to reduce execution time to a minimum.

ACKNOWLEDGEMENT

Researchers would like to thank the anonymous Wireless Personal communications reviewers for helping improve this work.

REFERENCES

Beimel, A. and Y. Ishai, 2001. Information-theoretic private information retrieval: A unified construction. Proceedings of the International Colloquium on Automata, Languages and Programming, July 8-12, 2001, Springer, Berlin, Germany, pp: 912-926.

Beimel, A. and Y. Stahl, 2002. Robust information-theoretic private information retrieval. Proceedings of the 3rd International Conference on SCN, September 11-13, 2002, Springer, Amalfi, Italy, pp: 326-341.

Brin, S., 1995. Near Neighbor Search in Large Metric Spaces. Stanford University, Stanford, California, Pages: 584.

Chakrabarti, S., 2003. Mining the Web: Discovering Knowledge from Hypertext Data. Part 2. Morgan Kaufmann Publishers, California, ISBN: 9781558607545, Pages: 345.

Charikar, M.S., 2002. Similarity estimation techniques from rounding algorithms. Proceedings of the 34th annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montreal, Canada, pp: 380-388.

Chor, B. and N. Gilboa, 1997. Computationally private information retrieval. Proceedings of the 29th Annual ACM Symposium on Theory of Computing, May 04-06, 1997, ACM, New York, USA., ISBN: 0-89791-888-6, pp: 304-313.

Chor, B., E. Kushilevitz, O. Goldreich and M. Sudan, 1998. Private information retrieval. J. ACM, 45: 965-981.

Ghinita, G., P. Kalnis, A. Khoshgozaran, C. Shahabi and K.L. Tan, 2008. Private queries in location based services: Anonymizers are not necessary. Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, June 09-12, 2008, ACM, New York, USA., ISBN:978-1-60558-102-6, pp: 121-132.

Ishai, Y. and E. Kushilevitz, 1999. Improved upper bounds on information-theoretic private information retrieval. Proceedings of the 31th Annual ACM Symposium on Theory of Computing, May 01-04, 1999, ACM, New York, USA., ISBN:1-58113-067-8, pp: 79-88.

Kushilevitz, E. and R. Ostrovsky, 1997. Replication is not needed: Single database, computationally-private information retrieval. Proceedings of the 38th Annual Symposium on Foundations of Computer Science, October 20-22, 1997, IEEE, Miami Beach, Florida, ISBN:0-8186-8197-7, pp: 364-373.

Marx, G.T., 2003. A tack in the shoe: Neutralizing and resisting the new surveillance. J. Soc. Issues, 59: 369-390.

Olumofin, F., P. Tysowski, I. Goldberg and U. Hengartner, 2010. Achieving efficient query privacy for location based services. Proceedings of the 10th International Symposium on Privacy Enhancing Technologies, July 21-23, 2010, Springer, Berlin, Germany, pp: 93-110.

Pang, H., X. Ding and X. Xiao, 2010. Embellishing text search queries to protect user privacy. Proc. VLDB. Endowment, 3: 598-607.

Roca, C.J., A. Viejo and J.J. Herrera, 2009. Preserving user's privacy in web search engines. Comput. Commun., 32: 1541-1551.

Schridde, C. and B. Freisleben, 2008. On the validity of the ϕ-Hiding assumption in cryptographic protocols. Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology-ASIACRYPT, December 7-11, 2008, Springer, Melbourne, Australia, pp: 344-354.

Tsan-Sheng, H., J.L. Churn, D.W. Wei and J. Chen, 2002. Quantifying privacy leakage through answering database queries. Proceedings of the 5th International Conference on Information Security, September 30-October-2, 2002, Springer, Sao Paulo, Brazil, pp: 162-176.

Ye, S., F. Wu, R. Pandey and H. Chen, 2009. Noise injection for search privacy protection. Proceedings of the International Conference on Computational Science and Engineering Vol. 3, August 29-31, 2009, IEEE, Vancouver,BritishColumbia,ISBN:978-1-4244-5334-4, pp: 1-8.

Young, A. and Y. Moti, 2004. Malicious Cryptography: Exposing Cryp-Tovirology. Wiley, Hoboken, New Jersey, ISBN:0-7645-4975-8, Pages: 387.