A report on:

Private Information Retrieval

Sander Mikelsaar

Supervised by Vitaly Skachek

University of Tartu, Estonia

June 12, 2018

Abstract

Openly accessible databases are an indispensable resource in modern times, but it might be necessary for the user to hide their intentions. If the user wants to hide what information they read from the database, they could always download the entire database, but this could result in a massive communication complexity proportional to the size of the database.

In this report, efficient information-theoretically secure methods of information retrieval are studied, ranging from traditional schemes using multiple replicated non-colluding servers, to schemes which use side information to reduce the communication complexity even in the single server case.

1 Introduction

This report, which is based on the original PIR paper [1] and a more recent paper [2], covers the schemes and the communication rates achieved by these schemes for information-theoretically secure means of private information retrieval (PIR). While there are ways of information storage that result in a better storage overhead, for example using erasure codes, the schemes covered in this report are based on multiple servers that store replicated versions of the database, and are based on the assumption of non-collusion between the servers.

The necessity of such schemes is well illustrated in the original PIR paper [1] using the following example. Consider the case when an investor wants to query a database that holds current stock-market values. It is reasonable to assume that the people hosting the database could be interested in seeing which stocks some successful investors are querying from their servers. To protect their business interests, it might be necessary for the investor to hide their demand from the server.

The obvious solution that provides privacy would be to download the entire database, as this would leak no information to the curious host, but could be rather costly as databases can be large in size. This is why more efficient schemes of PIR need to be explored.

2 Schemes

In this section, multiple private information retrieval schemes are presented, including classical schemes using $k \geq 2$ non-colluding servers with replicated data, as well as schemes which use side information in both single- and multiple-server cases.

For the sake of simplicity, the database shall be represented as a binary string $x = x_1 \dots x_n$ of length n and the queries will be made on a single bit, which is represented as the user's demand index w, the bit the user is interested in querying from the binary string x.

2.1 A Basic Two-Server Scheme

The simplest PIR scheme has its data replicated on $k = 2$ servers, denoted as SRV_1 and SRV_2. The user will construct an uniformly random subset of indices S_1 from the set of all possible indices $[n] = \{1, 2, \dots, n\}$. This means that each bit in the server $x_1 \dots x_n$ has a $\frac{1}{2}$ probability of being chosen. The user constructs another set $S_2 = S_1 \cup \{w\}$, if $w \notin S_1$ or $S_2 = S_1 \setminus \{w\}$, if $w \in S_1$ where w is the "demand index" or, in other words, the index of the bit that they are interested in. The user then sends S_1 to SRV_1 and S_2 to SRV_2. The servers SRV_1 and SRV_2 each respond with a single bit constructed by exclusive-or ($\oplus$, XOR) of all the bits it received indices for. More formally, each response is: $rsp_i = \bigoplus_{j \in S_i} x_j$, where i is the set of index of the server. It is clear that all the user needs to do is XOR the responses to get the value of the desired bit: $x_w = rsp_1 \oplus rsp_2$.

While this scheme does provide perfect secrecy, as to each server the set of indices looks uniformly random, it does not reduce the rate of communication between the parties in the case where the database consists of a binary string $x = x_1 \dots x_n$, when compared to downloading all of $x_1 \dots x_n$. The scheme could also be used in the case when messages are much larger than one bit in size, in which case, even this simple scheme would improve the rate of communication drastically. For example, in the case where the database consists of a binary string $x = x_1 \dots x_{10}$ and the user randomly selects sets S_1 and S_2 such that the sizes of the sets $|S_1| = 5$ and $|S_2| = 6$, the user will need to send 11 indices to the servers in total, and the servers would both respond with a single bit. The least amount of bits needed to send the indices is 4 per index, resulting in a total communication of 46 bits between the user and the server, while downloading the entire database would result in a total communication of only 10 bits. In contrast, if the messages were much greater in size, for example 10MB each, then the user would still only need to send the 11 indices, a total of 44 bits, to the servers and both servers would reply with a single 10MB response. This means that, instead of downloading the entire 100MB database, the total communication is reduced to 20MB plus the 44 bits used to the indices.

2.2 Multi-Server Scheme

This scheme can use any number of servers $k \geq 2$ and results in a much lower communication rate, especially when combined with the covering codes method that will be presented in the next section.

The scheme uses $k = 2^d$ servers for any $d \geq 1$ and requires a total communication cost of $2^d(dn^{\frac{1}{d}} + 1)$. The idea is to represent the messages $[n] = \{1, 2, \dots, n\}$ in the database as a d-dimensional cube $[l]^d$, such that $n = l^d$, and to construct subcubes in a similar fashion to what was done in the basic scheme covered in Section 2.1, which would allow the user to reconstruct the bit corresponding to their demand index.

We represent the demand index w as a d-tuple $w = (w_1 \dots w_d)$, such that the tuple represents the position of w within the d-dimensional cube. Also, we name the k servers with a binary string $\{0,1\}^d$ – for example, with $d = 3$, we will use $k = 2^3 = 8$ servers, which we shall name $SRV_{000}, SRV_{001}, \dots, SRV_{111}$.

Firstly, the user selects d subsets uniformly and randomly, similarly to what was done in section 2.1: $S_1^0, S_2^0, \dots, S_d^0 \subseteq [l]$. Then, similarly to the two-server scheme, the user constructs another d sets $S_1^1 \dots S_d^1$ and for each set, if $w_i \in S_i^0$, then $S_i^1 = S_i^0 \setminus \{w_i\}$ and if $w_i \notin S_i^0$, then $S_i^1 = S_i^0 \cup \{w_i\}$. Then, the user pairs the subsets, creating pairs $(S_1^0, S_1^1), \dots, (S_d^0, S_d^1)$ and sends to each server a single subset from each pair, such that the sets chosen from each pair correspond to the name of the server. For example, as shown previously for the case $d = 3$, SRV_{000} gets the sets S_1^0, S_2^0, S_3^0, SRV_{001} receives the sets S_1^0, S_2^0, S_3^1 and so on.

Each server then responds with the exclusive-or (XOR) of all the bits defined by the indices it received from the user. More formally, for each server SRV_σ, the response is constructed as:

$$rsp_i = \bigoplus_{j_1 \in S_1^{\sigma_1} \dots j_d \in S_d^{\sigma_d}} x_{j_1 \dots j_d}$$

The user then XOR-s all the bits it receives from the servers, which will result in the value of the desired bit:

$$x_w = \bigoplus_{i=1 \dots k} rsp_i$$

The scheme works because the demand index w, or more specifically, the demand index position $(w_1 \dots w_d)$ only appears in a single subcube, as exactly one of each sets per each subset pair (S_i^0, S_i^1) holds the value w_i. Each of the other positions $(j_1 \dots j_d)$ appear in an even number of subcubes and thus, when XOR-ing the responses from the servers, only the bit w remains.

From the perspective of each server, each of the d subsets of $[l]$ are uniformly random and thus no information about the demand index w is revealed.

As to each of the k servers, a sequence of d subsets of $[l]$ is sent and the response is only a single bit, the total communication complexity is equal to $k(dl + 1) = 2^d(d\sqrt[d]{n} + 1)$.

2.3 Covering Codes Scheme

The covering codes scheme is derived from the multi-server scheme described in Section 2.2. Instead of requiring $k = 2^d$ servers, it makes use of covering codes to let servers emulate multiple other servers. This drastically reduces the number of servers needed to make the previously described multi-server scheme work.

The scheme works by using a covering code of radius 1 for $\{0,1\}^d$ and taking the amount of codewords needed as the new k – the amount of servers needed. For example, for $d = 3$, a covering code $\{(0,0,0),(1,1,1)\}$ can be used, reducing the number of servers from 8 to only 2. For $d = 4$, the code $\{(0,0,0,0),(1,1,1,1),(1,0,0,0),(0,1,1,1)\}$ could be used, which means that instead of 16 servers, only 4 are required.

The main idea behind covering codes is to select a set of codewords such that all possible codewords in the message space are within a certain distance r of the nearest codeword of the chosen set of the covering code. The amount of codewords necessary for the construction of a covering code depends on the size of the message space and the chosen radius r.

The covering codes scheme requires servers to emulate other servers. Each server that is used needs to emulate every other server which has a name that is at most a Hamming distance of 1 away from it. For example with $d = 3$, the server SRV_{000} would, in addition to responding to the request made to the server, calculate all the possible responses from the servers SRV_{001}, SRV_{010} and SRV_{100}. Similarly, SRV_{111} would emulate SRV_{110}, SRV_{101} and SRV_{011}.

The emulation can be done because for any server that is being emulated, all but one set of indices is known. For example, with $d = 3$, SRV_{000} obtains the sets S_1^0, S_2^0, S_3^0. To emulate server SRV_{001}, the first two sets, S_1^0, S_2^0 are the same and for the third set S_3^1, there are only $\sqrt[3]{n}$ possible sets. The server SRV_{000} calculates all the possibilities: for all $j \in \{1, 2, \dots, \sqrt[3]{n}\}$, $S_3^1 = S_3^0 \cup \{j\}$ if $j \notin S_3^0$ and $S_3^1 = S_3^0 \setminus \{j\}$ if $j \in S_3^0$. The server then responds with all the possible responses from S_{001} to emulate it. The same is done for all the other servers it has to emulate as well.

The user then selects the "correct" responses corresponding to the servers being emulated and calculates the value of the desired bit as before.

As the construction of the sets remains unchanged, the privacy of the multi-server scheme is preserved. The communication from the user to the server is drastically reduced compared to the multi-server scheme. With the multi-server scheme, the user had to send a sequence of d subsets to each of the k servers, with a rather large k, whereas with the covering codes scheme, the user sends the same amount of d subsets to far fewer servers (in the case of $d = 3$, the user sends the sets to only 2 servers instead of 8). However, the communication from any server to the user equals $k + (2^d - k)n^{\frac{1}{d}}$ bits, which is much greater compared to the single-bit responses of the multi-server scheme. Thus the total communication cost is $(kd + 2^d - k)n^{\frac{1}{d}} + k$ bits. The precise volume of communication depending on n, d and k can be seen from the table:

Dimension d	2^d	# of servers (codewords) k	Volume (lower) bound	Total communication			
				asymptotic	$n = 2^{20}$	$n = 2^{30}$	$n = 2^{40}$
3	8	2	2	$12n^{1/3}$	1,224	12,300	123,864
4	16	4	4	$28n^{1/4}$	924	5,096	28,700
5	32	7	6	$60n^{1/5}$	1,020	3,900	15,420
6	64	12	10	$124n^{1/6}$	1,249	3,968	12,598
7	128	16	16	$224n^{1/7}$	1,792	4,480	11,872
8	256	32	29	$480n^{1/8}$	2,715	6,458	15,360

Table 1: Total communication, source: Private Information Retrieval [1]*

2.4 Partition and Code Schemes

The following scheme can be used to either query a single server or, as with the previous schemes, $k \geq 2$ servers, but requires the user to have some prior side information. Without side information, it can be shown that to achieve information-theoretic privacy, the whole content of the database needs to be downloaded if the data is stored on a single server. However, by using side information and the partition and code scheme, the amount of data downloaded can be greatly reduced.

2.4.1 Single-Server Partition and Code Scheme

Similarly to the previous schemes, assume that the database is a binary string $x = x_1 \dots x_n$ of length n. The user is interested in a single bit x_w and has a set S of side information of size m: $S = s_1, \dots, s_m$, which is a subset of the set of messages in the database.

First, the user creates $g - 1$ empty sets of size $m + 1$ and one set such that the sum of the sizes of all the g sets matches the size n of the database. In the case that $(m + 1)|n$, all the g sets will be of size $m + 1$, otherwise the last set will have a size of $n - (g - 1)(m + 1)$. More formally, the number of sets is $g = \left\lceil \frac{n}{m+1} \right\rceil$. Let these g sets be called $P_1 \dots P_g$. The user assigns probabilities to each of the sets according to their size – for each set P_i, the corresponding probability will be $\frac{|P_i|}{n}$. Then the user selects a random set based on the assigned probabilities.

Once chosen, the demand index w and the indices $s_1 \dots s_m$ of the side information the user possesses are added to the set. In the case that the set P_g is chosen and the size of the set is smaller than $m + 1$, the user randomly selects indices from the set of side information S.

Next, the user sends the sets of indices $P_1 \dots P_g$ to the server in a random order. The server responds with the XOR of all the bits defined by the indices in each set and sends the result of each set back to the user.

The user can then use their existing side information to extract the value of the bit x_w by taking the answer corresponding to the set which contained the demand index w and XOR-ing the answer with all the side information that were in the set.

It should be noted that while the original PIR paper [1] focused on minimizing the rate of total communication, the paper on PIR with side information [2] only focuses on the download rate.

2.4.2 Multi-Server Partition and Code Scheme

The multi-server partition and code scheme is based on a scheme presented in [3], but extends the scheme with the addition of side information.

The scheme assumes that the database consists of n binary string messages $x_1 \dots x_n$ replicated on k servers and that the user has a set S of side information with size m. It is also assumed that the length of each message x_i is $t = k^{n/(m+1)}$.

In the proposed scheme, the user constructs g empty sets, with $g = \frac{n}{m+1}$. Next, the user creates a set $P_1 = S \cup \{w\}$, where w represents the index of the message the user is interested in, or in other words, the user's demand index. The rest of the $g - 1$ sets are filled with indices that were not used in the construction of P_1, more formally each set $P_2 \dots P_g$ will be filled with random indices $i \in \{1, 2, \dots, n\} \backslash P_1$ such that the size of each of the resulting sets P_j, where $j \in \{1, \dots, g\}$, is equal to $m + 1$.

Secondly, the user sends all the sets $P_1 \dots P_g$ to all of the servers in a random order. The servers then construct g "$super\text{-}messages$" utilizing the protocol described in [3] and respond with a set the "$super\text{-}messages$" $\{\hat{x}_1, \dots, \hat{x}_g\}$, where $\hat{x}_i = [x_1, \dots, x_n] \cdot \mathbf{1}_{P_i}$ for $i \in \{1, \dots, g\}$. The characteristic vector $\mathbf{1}_S$ for any subset $S \subset [1, \dots, n]$ is a binary vector of length n, which is equal to 1 in each position j, where $j \in S$.

By using the "$super\text{-}message$" $\hat{x}_1$, the user can then extract the value of x_w, by "substracting" the side information used to construct the set P_1.

While the idea behind constructing the "$super\text{-}messages$" is rather complicated and beyond the scope of this report, the main idea behind it can be shown in a simple example: consider a case with 2 servers SRV_1 and SRV_2, which contain a replicated database of 2 messages X_1 and X_2, each 4 bits long. If the user is interested in downloading X_1, the servers respond with "$super\text{-}messages$", the construction of which can be seen in the following table:

SRV_1	SRV_2
$X_{1,1}$	$X_{1,2}$
$X_{2,1}$	$X_{2,2}$
$X_{1,3} + X_{2,2}$	$X_{1,4} + X_{2,1}$

Table 2: Construction of "super-messages", source: [2]

The indices in the table correspond to the index of the message and the index of bit in the message – for example $X_{1,1}$ is the value of the first bit of the message X_1. The user able to construct the whole message X_1 from the bits received, as the values $X_{1,1}$ and $X_{1,2}$ can be read directly from the messages and the values $X_{1,3}$ and $X_{1,4}$ can be retrieved by subtracting $X_{2,2}$ and $X_{2,1}$, which are also known from the messages, from $(X_{1,3} + X_{2,2})$ and $(X_{1,4} + X_{2,1})$ respectively.

Using the scheme, a download rate of $R = (1 + \frac{1}{k} + \ ... + \frac{1}{k^{\frac{n}{(m+1)}-1}})^{-1}$ can be achieved, as it is shown in [3].

2.5 Maximum Distance Separable Scheme

The following scheme uses a maximum distance separable (MDS) code to achieve information-theoretic privacy for both the user's demand index w, as well as their set of side information S.

The scheme assumes that the database consists of n binary messages $x = \{x_1, \dots, x_n\}$ of length t, with a sufficiently large size $t \geq \log_2(2n - m)$. The user also has a set S of m messages of side information: $S = \{s_1, \dots, s_m\}$.

As $2^t \geq 2n - m$, it is possible to construct a $[2n - m, n]$ MDS code over $\mathbb{F}_{2^t}$. By using this construction, the server is queried to send the $n - m$ parity symbols of the MDS code to the user, who is able to use the parity symbols and the side information S to decode the remaining $n - m$ messages. This means that the user will have knowledge of all the m messages in the database, including the message corresponding to the demand index w.

For example, take the generator matrix of a Reed-Solomon code. The matrix can be changed through a series of linear transformations to a form where the leftmost k by k submatrix is an identity matrix – a zero matrix with the main diagonal of 1-s. The form of these matrices can be seen in the following figure:

$$A = \begin{bmatrix} 1 & \dots & 1 & \dots & 1 \\ a_1 & \dots & a_k & \dots & a_n \\ a_1^2 & \dots & a_k^2 & \dots & a_n^2 \\ \vdots & \dots & \vdots & \dots & \vdots \\ a_1^{k-1} & \dots & a_k^{k-1} & \dots & a_n^{k-1} \end{bmatrix} \rightarrow A' = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & g_{1,k+1} & \dots & g_{1,n} \\ 0 & 1 & 0 & \dots & 0 & g_{2,k+1} & \dots & g_{2,n} \\ 0 & 0 & 1 & \dots & 0 & g_{3,k+1} & \dots & g_{3,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & \dots & 0 & \dots & 1 & g_{k,k+1} & \dots & g_{k,n} \end{bmatrix}$$

Figure 1: Generator matrix for a Reed-Solomon code and its transformation

When the matrix A' is multiplied by a vector of all the messages in the database, the resulting vector $y = (x_1, \dots, x_n) * A'$ will be of the form $(x_1, \dots, x_k \mid y_{k+1}, y_{k+2}, \dots, y_n)$. The symbols $x_1 \dots x_k$ in y are known as information symbols and the symbols $y_{k+1}, \dots, y_n$ are the parity check symbols. Because each y_i is calculated as $y_i = x_1 g_{1,i} + x_2 g_{2,i} + \dots$, the user is able to construct a set of equations from the parity check symbols it receives from the server. Because the matrix A' and a set of messages (side information set S) is known to the user, it is possible to calculate the values of all the messages in the server from the set of equations.

As neither the query nor the response of the server contains information about the demand index w or the contents of the user's set of side information S, it is clear that the privacy requirement is satisfied. The only piece of information revealed to the server is the size m of the set S of the user's side information.

The amount of data downloaded from the server is equal to $(n-m)$ parity symbols for a MDS code over $\mathbb{F}_{2^t}$. This means that each parity symbol has a size of t bits and the resulting total cost of communication is equal to $(n-m)t$ bits.

3 Conclusion

In this report, based on the papers [1] and [2], the problem of private information retrieval is explored. We survey several PIR schemes discussed in the papers. While these schemes rely on different models of PIR, ranging from multi-server schemes with no side information to both single- and multi-server schemes using side information, the goal of each of the schemes is to use clever constructions to reduce the rate of communication. The report explains the methods and constructions used in these schemes, as well as the resulting communication rates in a slightly simplified manner when compared to the original papers. While the schemes covered are somewhat unrealistic in their assumptions, they could be applied to more realistic problems of information retrieval and ensure privacy when it is needed.

4 References

[1] Benny Chor, Oded Goldreich, Eyal Kushilevitz, Madhu Sudan, "Private Information Retrieval*", April 1998

[2] Swanand Kadhe, Brenden Garcia, Anoosheh Heidarzadeh, Salim El Rouayheb, and Alex Sprintson, "Private Information Retrieval with Side Information", September 2017

[3] Hua Sun, Syed A. Jafar,"The Capacity of Private Information Retrieval", February 2017