Report on Codes for Private Information Retrieval

Oliver-Matis Lill
Supervisor: Vitaly Skachek

December 18, 2016

1 Introduction

The purpose of PIR (Private Information Retrieval) is to facilitate retrieval of data without revealing what data was retrieved. In most PIR schemes the data is copied to k servers and the user sends requests to those servers in such a fashion that he can retrieve the information he wants but the servers don't get any information on what he has requested, assuming they don't collaborate with each other.

There are many effective PIR schemes around, some having subpolynomial communication complexity, but they all have one big drawback. They assume the data is copied over k servers, inducing a massive storage overhead. The paper this report is based on introduced a way to alleviate the storage overhead of any linear PIR schemes by utilizing coding.

In this report, Section 2 is dedicated to giving the necessary background information to understand how PIR schemes work. Section 3 will present the theory behind combining some code and some PIR Scheme into Coded PIR with better storage overhead. In Section 4 we will introduce an example of a PIR code construction and section 5 will conclude this report.

2 PIR Schemes

In this report the notation $[n]$ for a positive integer n will refer to the set $\{1, \ldots, n\}$. For the purpose of this report we assume that the data stored in servers is an n bit binary vector $\mathbf{x} \in \mathbb{F}_2^n$ and the user Alice wants to retrieve a single bit x_i where $i \in [n]$.

Definition 2.1. A **k-server PIR scheme** consists of the following:

1. k servers, each storing the data vector $\mathbf{x}$
2. The user (Alice) who wants to retrieve x_i without revealing i.

Definition 2.2. A k**-server PIR protocol** has three algorithms $(\mathcal{Q}, \mathcal{A}, \mathcal{C})$ and consists of the following steps:

1. Alice invokes $\mathcal{Q}(k, n; i)$ to generate k randomized queries $(q_1, \ldots, q_k)$, for querying the index i.
2. For each $j \in [k]$ Alice sends the query q_j to the j-th server. For the purpose of this report q_j is a binary vector of fixed length.
3. For $j \in [k]$ the j-th server responds with $a_j = \mathcal{A}(k, j, \mathbf{x}, q_j)$ which for simplicity will be considered a fixed length binary vector.
4. Alice uses the reconstruction algorithm $\mathcal{C}(k, n; i, a_1, \ldots, a_k)$ to compute the value of x_i.

The protocol needs to satisfy an additional property, in order to guarantee privacy (assuming servers don't communicate): any server j must not be able to learn anything about the requested bit location i given the query q_j. To guarantee that, the probability distribution of q_j must be equal for every bit location i, or in other words: $P(q_j = c | i = i_1) = P(q_j = c | i = i_2)$ for any two $i_1, i_2 \in [n], i_1 \neq i_2$ and for every possible query pick c.

For the coding augmentation presented in this report to be useful, the PIR protocol of our schemes must satisfy another property: linearity.

Definition 2.3. A k-server PIR protocol is a **linear PIR protocol** if its function $\mathcal{A}$ for every k, every $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{F}_2^n$ every $j \in [k]$ and every query q_j satisfies the following property:

$$\mathcal{A}(k, j, \mathbf{x}_1 + \mathbf{x}_2, q_j) = \mathcal{A}(k, j, \mathbf{x}_1, q_j) + \mathcal{A}(k, j, \mathbf{x}_2, q_j)$$

Most PIR schemes satisfy this property, so this is not a very limiting constraint. Next we will present an example of a linear PIR scheme.

Example 2.1. Consider a 2-server PIR scheme where each server stores the n-bit database $\mathbf{x}$ and where Alice wants to read the bit x_i. First Alice picks a vector $\mathbf{a} \in \mathbb{F}_2^n$ uniformly at random. Next Alice sends the following queries to the servers:

$$q_1 = \mathbf{a}$$
$$q_2 = \mathbf{a} + \mathbf{e}_i$$

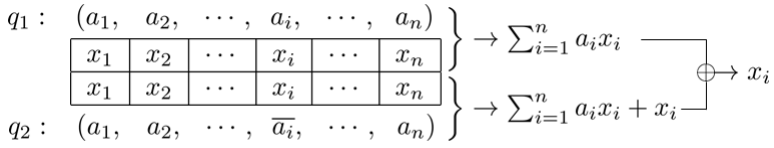
Here $\mathbf{e}_i$ is an unit vector with 1 at position i. After that the servers respond with the following answers:

$$a_1 = q_1 \cdot \mathbf{x} = \mathbf{a} \cdot \mathbf{x}$$
$$a_2 = q_2 \cdot \mathbf{x} = (\mathbf{a} + \mathbf{e}_i) \cdot \mathbf{x}$$

Note that since they compute dot products, they respond with a single bit. With the requested answers, Alice computes the desired bit by just adding the answers together:

$$\mathcal{C}(2, n; i, a_1, a_2) = a_1 + a_2 = \mathbf{a} \cdot \mathbf{x} + (\mathbf{a} + \mathbf{e}_i) \cdot \mathbf{x} = (\mathbf{a} + \mathbf{a} + \mathbf{e}_i) \cdot \mathbf{x} = \mathbf{e}_i \cdot \mathbf{x} = x_i$$

Figure 1. Alice sends the queries $\mathbf{a}$ and $\mathbf{a} + \mathbf{e}_i$ to the servers, servers respond with cross products and Alice sums them to get x_i.



Note that this scheme does preserve the privacy of x_i, since both servers will get some n-bit vector uniformly at random, and won't be able to deduce any information about i only from the query they received. The value of x_i however can be received thanks to the fact that the queries q_1 and q_2 depend on each other. Alice can exploit this dependency to compute x_i, meanwhile as long as the servers don't communicate, they won't be able to see this dependency.

The given example should give some intuition as to how PIR schemes work. Note that Alice has to upload n bits to both servers (but download only a single bit), so the communication complexity is $O(n)$. This makes it a very unpractical scheme, however there are PIR schemes that achieve subpolynomial communication complexity. One big problem with those PIR schemes is that they require the database to be copied to each of the k servers, inducing a storage overhead of k. Next we will present a coding augmentation that will allow us to significantly reduce the storage overhead.

3 Coded PIR

The main idea of our coding augmentation is to divide the data vector $\mathbf{x}$ into s equal parts $\mathbf{x}_1, \ldots, \mathbf{x}_s$ and then have m servers store some linear combinations of them. Formally suppose our scheme consists of the following:

1. The database $\mathbf{x}$ of length n divided into s equal length parts $\mathbf{x}_1, \ldots, \mathbf{x}_s$.
2. m servers containing some linear combinations of the database $\mathbf{c}_1, \ldots, \mathbf{c}_m$.
3. The user Alice, who wants to receive the value of the i-th bit $x_{j,i}$ located in the j-th part of the database $\mathbf{x}_j$

The basic idea is to use coding to simulate a k-server linear PIR protocol with m ($m > k$) servers. The servers store $\frac{n}{s}$ bit linear combinations of the database parts instead of the n bit copies of the database and with sufficiently small m, the storage overhead will be significantly reduced. In more detail, the coded PIR protocol consists of the following steps:

1. Alice invokes $\mathcal{Q}(k, \frac{n}{s}; i)$ to generate the k queries $(q_1, \ldots, q_k)$.
2. She finds k disjoint sets $S_1, \ldots, S_k \subset [m]$ such that for each set u the sum of the data in the servers represented by S_u is $\mathbf{x}_j$. In other words: $\sum_{v \in S_u} \mathbf{c}_v = \mathbf{x}_j$.

3. For each $u \in [k]$ she sends the query q_u and simulated server index u to every server in S_u. To preserve privacy she sends a random (q_u, u) pair to each server not in one of those sets.

4. For every $u \in [k]$ each server $v \in S_u$ responds with $\mathcal{A}(k, u, \mathbf{c}_v, q_u)$. For that set, she sums the answers to get $a_u = \sum_{v \in S_u} \mathcal{A}(k, u, \mathbf{c}_v, q_u) = \mathcal{A}(k, u, \sum_{v \in S_u} \mathbf{c}_v, q_u) = \mathcal{A}(k, u, \mathbf{x}_j, q_u)$

5. Finally she uses the reconstruction algorithm $\mathcal{C}(k, \frac{n}{s}; i, a_1, \ldots, a_k)$ to compute the value of $x_{j,i}$.

Note that in order for us to be able to perform step 2, those k disjoint sets must exist for every database part $\mathbf{x}_j$. As a result, the choice of linear combinations $\mathbf{c}_1, \ldots, \mathbf{c}_m$ determines what is the maximum k such that we can simulate k-server linear PIR protocol on that system. Additionally note that the linearity of our PIR protocol is necessary for performing step 4.

If the sets $S_1, \ldots, S_k$ are picked in the same manner each time, it might reveal some data to the servers on what was queried, not from the query q_u itself but from the index u of the PIR scheme server they should emulate. Fortunately this can be easily resolved by randomly permuting the sets, resulting in each server getting a random emulation index u as well as the query q_u that doesn't reveal anything.

Next we will present an example of a coded PIR scheme:

Example 3.1. Suppose we divide the database into $s = 4$ parts and use a system of 8 servers. Suppose the data stored in servers is the following:

$$\begin{array}{llll} \mathbf{c}_1 = \mathbf{x}_1, & \mathbf{c}_2 = \mathbf{x}_2, & \mathbf{c}_3 = \mathbf{x}_3, & \mathbf{c}_4 = \mathbf{x}_4 \\ \mathbf{c}_5 = \mathbf{x}_1 + \mathbf{x}_2, & \mathbf{c}_6 = \mathbf{x}_2 + \mathbf{x}_3 & \mathbf{c}_7 = \mathbf{x}_3 + \mathbf{x}_4, & \mathbf{c}_8 = \mathbf{x}_4 + \mathbf{x}_1 \end{array}$$

Alternatively it can be represented in the following form:

$$(\mathbf{c}_1, \ldots, \mathbf{c}_8) = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Note how the data stored in servers is completely determined by the matrix. In coding it's called the the generator matrix and we will refer to it as such. Note that the first 4 columns of that matrix form an identity matrix. This is called the standard form and the columns after the identity matrix are called redundancy vectors.

Suppose that Alice wants the i-th bit of x_2 and that we are simulating a 3-server PIR protocol. In that case she performs the following steps:

1. She generates the queries q_1, q_2, q_3.

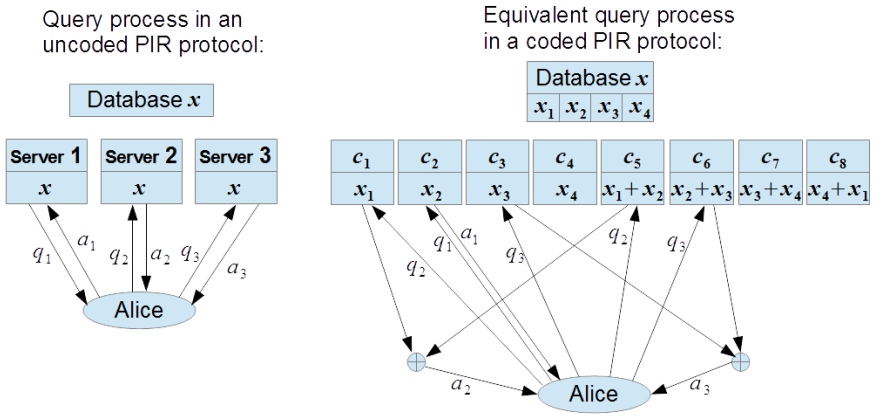
2. She picks the sets ($\{2\}$, $\{1, 5\}$, $\{3, 6\}$) and sends the query index pair $(q_1, 1)$ to server 2, the pair $(q_2, 2)$ to servers 1 and 5 and finally the pair $(q_3, 3)$ to servers 3 and 6. To preserve privacy she happens to randomly send the pair $(q_2, 2)$ to servers 4 and 7 and the pair $(q_1, 1)$ to server 8.

3. The servers respond and she uses the responses to calculate the following:

$$a_1 = \mathcal{A}(3, 1, \mathbf{c}_2, q_1) = \mathcal{A}(3, 1, \mathbf{x}_2, q_1)$$
$$a_2 = \mathcal{A}(3, 2, \mathbf{c}_1, q_2) + \mathcal{A}(3, 2, \mathbf{c}_5, q_2) = \mathcal{A}(3, 2, \mathbf{x}_2, q_2)$$
$$a_3 = \mathcal{A}(3, 3, \mathbf{c}_3, q_3) + \mathcal{A}(3, 3, \mathbf{c}_6, q_3) = \mathcal{A}(3, 3, \mathbf{x}_2, q_3)$$

4. Finally she reconstructs the desired bit $x_{1,i}$ with $\mathcal{C}(3, \frac{n}{3}; i, a_1, a_2, a_3)$

Figure 2. Comparison of this coded PIR scheme and the original uncoded PIR that it simulates.



We can easily see that for every x_u the 3 disjoint sets exist, therefore this coded system can simulate a 3-server PIR protocol. Without coding we would have had to store 3 copies of the database giving us a storage overhead of 3. However in this coded scheme we use 8 servers each storing $\frac{1}{4}$ of the database, giving us a storage overhead of 2, a noticeable improvement.

A coded PIR scheme is primarily characterized by the binary generator matrix G. It describes the data stored in servers with the following formula:

$$(\mathbf{c}_1, \ldots, \mathbf{c}_m) = (\mathbf{x}_1, \ldots, \mathbf{x}_s) \cdot G$$

Now we will define what generator matrices give us useful coded PIR schemes.

Definition 3.1. A binary $s \times m$ matrix G generates a k-server PIR code if for each $i \in [s]$ there exist k disjoint subsets of columns of G with the columns of each subset adding up to a vector $\mathbf{e}_i$, the unit vector with the single 1 in position i. We call such code an $[m, s]$ k-server PIR code.

We can easily see that these subsets of columns correspond to disjoint sets of servers that when summed together give us the desired $\mathbf{x}_j$.

We can see that each server of this PIR scheme stores a $\frac{1}{s}$ fraction of the database. Since we use m servers, this code will give us a storage overhead of $\frac{m}{s}$.

Note that if a PIR protocol $\mathcal{P}$ uploads $\mathcal{U}(\mathcal{P}; n, k)$ bits to a single server and downloads $\mathcal{D}(\mathcal{P}; n, k)$ bits from a single server, then the total number of bits communicated in our coded PIR is $m \cdot (\mathcal{U}(\mathcal{P}; \frac{n}{s}, k) + \mathcal{D}(\mathcal{P}; \frac{n}{s}, k))$. Note that uncoded PIR communicates $k \cdot (U(\mathcal{P}; n, k) + D(\mathcal{P}; n, k))$ bits. If m is significantly larger than k, the communication cost can become larger with coding. Therefore when designing coded PIR schemes we also need to mind that m doesn't become too much larger than k.

In the next section we will introduce a PIR code construction that should demonstrate the usefulness of coding in PIR.

4 PIR Code Constructions

There are many ways to construct effective PIR schemes. We are going to present one that is conceptually simple yet still very effective.

4.1 Cubic Construction

This construction is based on multidimensional cubes. We pick the k meaning that we enable simulation of up to k-server PIR protocols. We also pick a positive integer σ and we divide the database into $s = \sigma^{k-1}$ parts.

We will denote the database parts as $\mathbf{x}_{i_1, i_2, \ldots, i_{k-1}}$ where $i_j \in \sigma$ for each $j \in [k-1]$. Additionally we generate $(k-1)\sigma^{k-2}$ redundancy vectors denoted $p^{(u)}_{i_1, i_2, \ldots, i_{u-1}, iu+1, \ldots, i_{k-1}}$ for each $u \in [k-1]$ where the data stored in the redundancy vectors is given by:

$$p^{(u)}_{i_1, i_2, \ldots, i_{u-1}, i_{u+1}, \ldots, i_{k-1}} = \sum_{i_u=1}^{\sigma} \mathbf{x}_{i_1, i_2, \ldots, i_{k-1}}$$

We can see that for every desired data part $\mathbf{x}_{i_1, \ldots, i_{k-1}}$ the k disjoint sets will be given by:

$$\mathbf{x}_{i_1, \ldots, i_{k-1}}$$

$$p^{(u)}_{i_1, \ldots, i_{u-1}, i_{u+1}, \ldots, i_{k-1}} + \sum_{c \in [k-1] \setminus i_u} \mathbf{x}_{i_1, \ldots, i_{u-1}, c, i_{u+1}, \ldots, i_{k-1}} \text{for each } u \in [k-1]$$

With $s = \sigma^{k-1}$ data parts and $(k-1)\sigma^{k-2}$ redundancy vectors, the number of servers we need is $m = \sigma^{k-1} + (k-1)\sigma^{k-2}$. To better illustrate this construction we will present an example:

Example 4.1. Assume $k = 3$. In that case $s = \sigma^2$ and the number of redundancy vectors is 2σ. The data stored in redundancy vectors is given by:

$$p^{(1)}_i = \sum_{j=1}^{\sigma} \mathbf{x}_{i,j},$$

$$p_i^{(2)} = \sum_{j=1}^{\sigma} \mathbf{x}_{j,i}.$$

Figure 3. Cubic construction for a 3-server PIR code. The different colors denote the different subsets for acquiring $\mathbf{x}_{i,j}$.

$x_{1,1}$	$x_{1,2}$	$\cdots$	$x_{1,j}$	$\cdots$	$x_{1,\sigma}$	$p_1^{(1)}$
$x_{2,1}$	$x_{2,2}$	$\cdots$	$x_{2,j}$	$\cdots$	$x_{2,\sigma}$	$p_2^{(1)}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$x_{i,1}$	$x_{i,2}$	$\cdots$	$x_{i,j}$	$\cdots$	$x_{i,\sigma}$	$p_i^{(1)}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$x_{\sigma,1}$	$x_{\sigma,2}$	$\cdots$	$x_{\sigma,j}$	$\cdots$	$x_{\sigma,\sigma}$	$p_\sigma^{(1)}$
$p_1^{(2)}$	$p_2^{(2)}$	$\cdots$	$p_j^{(2)}$	$\cdots$	$p_\sigma^{(2)}$	

We can see that for cubic construction the storage overhead m can also be written as:

$$m = \sigma^{k-1} + (k-1)\sigma^{k-2} = s + (k-1)s^{\frac{k-2}{k-1}}$$

This means that our storage overhead is $1 + (k-1)s^{-\frac{1}{k-1}} = 1 + O(s^{-\frac{1}{k-1}})$. As a result, as s becomes large, our storage overhead will approach 1 for any fixed k. As we can see this simple construction can already give us a big reduction in storage overhead. There are more complicated constructions with better characteristics.

5 Conclusion

So far the best (and trivial) lower bound on storage overhead is given by $1 + O(\frac{\log s}{s})$. The cubic construction presented here has an overhead of $1+O(s^{-\frac{1}{k-1}})$, which still allows us to get it arbitrarily close to 1. We have shown that coding can easily augment most PIR schemes to deal with the problem of storage overhead. There is a lot of room for creativity in designing useful PIR code constructions and this could be an interesting avenue of future research.

6 References

`https://arxiv.org/pdf/1505.06241.pdf`