Private Information Retrieval (PIR)

Martin Szewieczek, 01627784

194.115 Seminar aus Security

2 July 2022

Contents

1 Introduction

This work is based on the chapter *II. Private Information Retrieval* from the IEEE Journal article on *Private Retrieval, Computing, and Learning: Recent Progress and Future Challenges*[Ulu+22] and the first part of the talk from Dima Kogan at the BIU Winter School on Cryptography. The talk is publicly available on YouTube.

The private information retrieval (PIR) protocol allows a client to retrieve information from a public server database without revealing the information retrieved. This can be very useful in applications were sensitive data is involved, e.g. in the medical or in the banking sector. The privacy requirement can be seen in two different ways:

- information-theoretic: Each server cannot infer any information on the identity of the requested message, even if assuming the server has infinite computation power.
- computational: Each server has only limited computation power, and under such a computational constraint, it is required that the server cannot learn anything on the identity of the requested message.

A naive solution to the PIR problem is that the client is retrieving the whole database and then querying it locally. This solution is yielding to a very high communication cost between client and server and therefore it is not usable in practise.

There are currently two main approaches to solve the private information retrieval (PIR) problem. The first approach is replicating the database between two or more non-colluding servers and is generally referred to as *Multi-server PIR*. The second approach is generally called *Single-server PIR* and is not using multiple servers. It is solving the private information (PIR) problem with the use of cryptographic assumptions.

Figure 1 is showing the idea of a *Multi-server PIR* with two database servers. A client is retrieving information from two non-colluding databases, DB1 and DB2. In the first step, the client is using an algorithm to transform its query into two separate queries q1 and q2. They are then sent to the databases. In the second step, each database is running an algorithm to construct the response and is then sending the response back to the client. Then, in the third step, the client is running an algorithm which is taking the two responses as input and is outputting the desired information. To preserve privacy it must hold that if looking at the individual requests, here q1 and q2, it is not feasible to know which information is being requested.

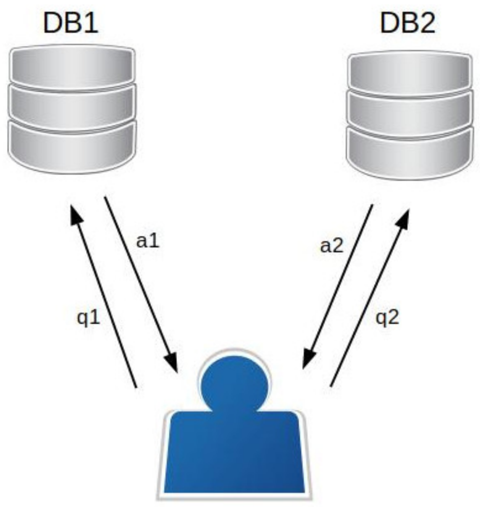


Figure 1: Multi-server PIR

In most of the basic schemes, the database is seen as a two dimensional matrix and the client is querying an element on a specific position in the matrix. Kushilevitz and Ostrovsky[KO97] were introducing back in 1997 a basic scheme for a *Single-server PIR*. In figure 2 the idea of the scheme is shown. The database can be seen as matrix X with i rows and j columns and the client wants to retrieve an element at position ij. Linear Homomorphic Encryption is used to encrypt the query. In step one, the client is encrypting a bit vector with all zeros except for the element at position j. The encrypted query q is then sent to the server. As step two, the server is multiplying the matrix X with the query q. The result is the encrypted response, which is a vector with the desired information at the i-th position. The idea and the security is based on the properties of Linear Homomorphic Encryption.

Both described schemes were querying for one individual element (a single bit per message). This is often referred to as *canonical PIR setting*. In the *canonical PIR setting*, a user wishes to retrieve one of K available messages, from N non-communicating/non-colluding servers, each of which has a copy of these K messages. Most of recent research is being done to reduce the communication cost between the client and server, including both the query communication (upload) cost and the answer communication (download) cost, with respect to N and K. Since the download cost is normally exceeding the upload cost, research is focusing mostly on reducing the download cost. The efficiency is measured by the retrieval rate which is the ratio of the number of retrieved desired message symbols to the number of total downloaded symbols, where the symbols are defined over the same field. The capacity of PIR is the maximum retrieval rate, in other words it is the maximum number of bits of the desired message that can be privately retrieved per one bit of downloaded information.

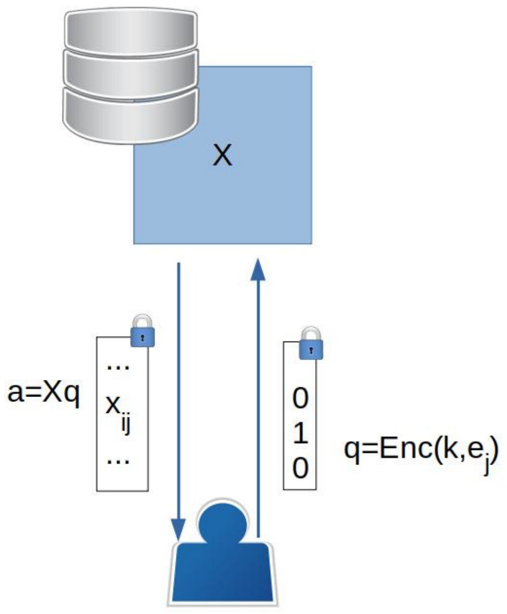


Figure 2: Single-server PIR

For the canonical PIR setting the capacity is

$$C = \left(1 + \frac{1}{N} + ... + \frac{1}{N^{K-1}}\right)^{-1}$$

In the next section various generalisations of the canonical PIR problem are being discussed.

2 Generalisations

2.1 Multi-Message PIR Systems (MPIR)

Having multiple rounds of queries and answers is often preferable. It was shown that the capacity of multiround PIR is in fact the same as single round PIR when there is no constraint placed on the storage cost. This equality is even holding when there are a number of T servers colluding. Note, when the storage is getting constrained, this equality is breaking. But going further, researchers were showing that even in settings with byzantine databases multiround communication is beneficial.

In multi-message PIR, the user is downloading multiple messages privately. Regarding this research area it was shown that downloading multiple messages jointly is more efficient and is beating the sequential use of single-message PIR.

2.1.1 Communication cost

In their work, Banawan and Ulukus[BU18] are presenting various capacity properties for MPIR with N non-colluding and replicated databases.

For $P \geqslant \frac{M}{2}$, meaning the number of desired messages P is at least half of the number of overall stored messages M. The capacity is given by

$$C_{MPIR} = \frac{1}{1 + \frac{M-P}{PN}}$$

If the total number of messages M is an integer multiple of the number of desired messages P (i.e. $\frac{M}{P} \in \mathbb{N}$) then the capacity is given by

$$C_{MPIR} = \left(\frac{1 - \frac{1}{N}}{1 - (\frac{1}{N})^{M/P}} \right) \text{ for } N > 1$$

$$C_{MPIR} = \frac{P}{M} \text{ for } N = 1$$

2.2 Cache or Side Information Aided PIR

Cache aided private information retrieval and side information aided PIR are leading to a reduction in the download costs. This is done through using cached or side information retrieved from databases when querying for a desired message. There are different types:

- the databases are aware or unaware of the side-information at the user
- the user wishes to only keep the message index private from the databases or both the message index and side information
- the type of side information available at the user (e.g. subset of messages or fraction of some/all messages)

In the next subsections a high level solution to a cached aided PIR problem, where the databases are knowing part of the cached bits, is being shown. A more detailed overview and corresponding proofs can be found in [WBU18]

2.2.1 Problem Description

Figure 3 is showing the related system model. A PIR problem with N non-communicating databases is being considered. Each database is storing an identical copy of K statistical independent messages. Each message is L bits long. In the user's local cache memory the

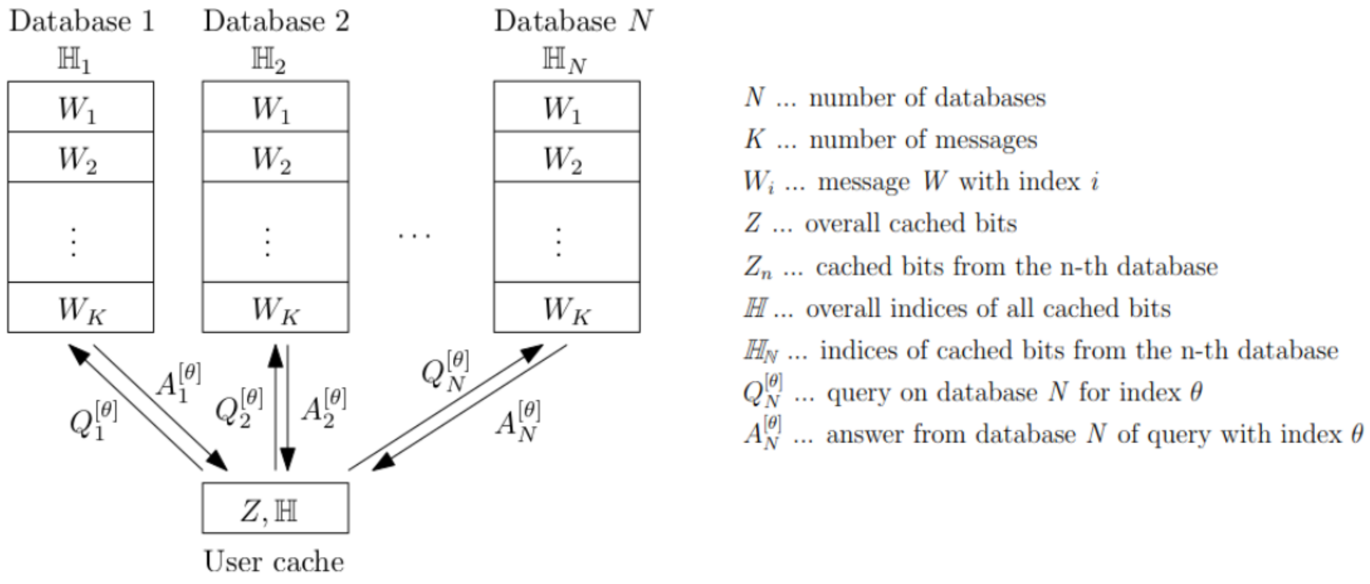

Database 1
$\mathbb{H}_1$
Database 2
$\mathbb{H}_2$
Database N
$\mathbb{H}_N$
W_1
W_2
$\vdots$
W_K
$\cdots$
$Q_1^{[\theta]}$
$A_1^{[\theta]}$
$Q_2^{[\theta]}$
$A_2^{[\theta]}$
$Q_N^{[\theta]}$
$A_N^{[\theta]}$
$Z, \mathbb{H}$
User cache
N ... number of databases
K ... number of messages
W_i ... message W with index i
Z ... overall cached bits
Z_n ... cached bits from the n-th database
$\mathbb{H}$ ... overall indices of all cached bits
$\mathbb{H}_N$ ... indices of cached bits from the n-th database
$Q_N^{[\theta]}$ ... query on database N for index θ
$A_N^{[\theta]}$ ... answer from database N of query with index θ

Figure 3: Cache-Aided PIR Model, taken from [WBU18]

overall cached bits (Z) and indices related to the cached bits are getting stored ($\mathbb{H}$). The local user cache is able to store up to KLr bits. r is the defined caching ratio with $0 \leqslant r \leqslant 1$.

The system is consisting of two phases, the prefetching and the retrieval phase. The prefetching phase is happening before the retrieval phase.

Prefetching Phase:

In the prefetching phase, for each message W_K the user is choosing randomly and independently Lr bits out of L and is writing them to the cache. This means that the bits written to the cache and hence also $\mathbb{H}$ is independent of the messages content. The Lr bits are chosen by prefetching the same amount of bits from each database. More precisely $\frac{KLr}{N}$ bits are getting stored from each database. Since the user caches a subset of the bits from each message, this is called *uncoded prefetching*. It is important to note that a database n is knowing the indices of the cached bits from itself, i.e. $\mathbb{H}_n$, but no other indices.

Retrieval Phase:

Firstly, the user is privately generating an index $\theta \in [K]$. The goal is then to retrieve a message W_θ such that no database is able to identify θ. Due to the fact that θ is unknown during the prefetching phase, the cached bit indices $\mathbb{H}$ are independent of θ. Combining this with $\mathbb{H}$ being independent of the messages content, it is not possible for a database to draw a connection from the cached bits to the private message W_θ.

Then the user is sending N queries $Q_1^{[\theta]}, ..., Q_N^{[\theta]}$ to the N databases, where $Q_n^{[\theta]}$ is the query for message W_θ sent to database n. The queries are being generated according to $\mathbb{H}$, which are independent of the realizations of the K messages. After that, the N databases are sending the answer strings $A_1^{[\theta]}, ..., A_N^{[\theta]}$, where $A_n^{[\theta]}$ is a function of $Q_n^{[\theta]}$ and all the K messages. Next, the user is decoding the received answering strings from all databases into the desired message W_θ.

To ensure private retrieval of message W_θ the following privacy constraint has to hold:

$$\forall n \in [N], \forall \theta \in [K] : (Q_n^{[1]}, A_n^{[1]}, W_1, ..., W_K, \mathbb{H}_n) \sim (Q_n^{[\theta]}, A_n^{[\theta]}, W_1, ..., W_K, \mathbb{H}_n)$$

A $\sim$ B means that A and B are identically distributed. The result of that is that a database view of a query is always the same, even if the queries are referring to different messages.

2.2.2 Example

In this section an example of the problem description from section 2.2.1 with $K = 3$ messages (W_1, W_2, W_3) and $N = 2$ databases is shown. Figure 4 is illustrating the communication between the user and the databases.

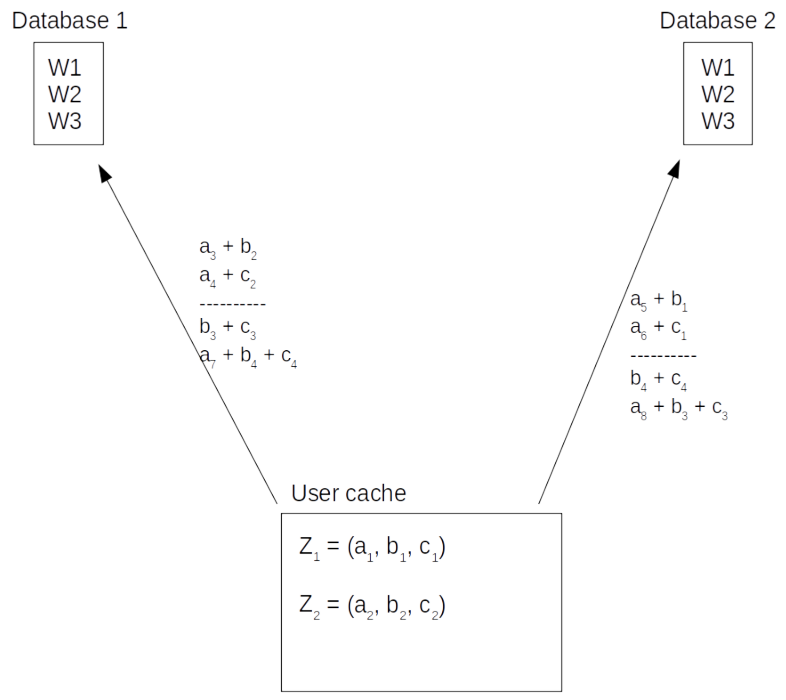


Figure 4: Cache-Aided PIR Example

The bits from all messages W_1, W_2, W_3 are getting permuted randomly and independently. a_i is denoting the bits for W_1, b_i the ones from W_2 and c_i the ones from W_3. In this example the caching ratio r is $\frac{1}{4}$ and the message size is set to 8 bits.

The user wants to retrieve $W_1 = a_1, a_2, ..., a_8$ privately. This means that every bit a_n has to be retrieved either from database 1 or 2.

Prefetching Phase:
Because each message is 8 bits long and the caching ratio is $\frac{1}{4}$ the user is caching 2 bits from each message ($8 * \frac{1}{4}$). Therefore the user is caching 1 bit from each database for each message. One option for the cached bits from database 1 could be $Z_1 = (a_1, b_1, c_1)$ and for database 2 $Z_2 = (a_2, b_2, c_2)$.

Retrieval Phase:
A query is consisting of side information mixed with a desired bit. Side information can be any cached bits or bits retrieved from another database.

First, the user is querying $a_3 + b_2$ and $a_4 + c_2$ from database 1. b_2 and c_2 are the side information. Note that no side information from Z_1 is used because database 1 is knowing those values and could infer a desired bit. To keep message symmetry, the user further is querying $b_3 + c_3$ from database 1. A similar approach is being used for querying $a_5 + b_1, a_6 + c_1$ and $b_4 + c_4$ from database 2. Then the user is exploiting side information $b_4 + c_4$ for querying $a_7 + b_4 + c_4$ from database 1 and $b_3 + c_3$ for querying $a_8 + b_3 + c_3$ from database 2. Here, the retrieved bits are used as side information. After this retrieval no more side information can be used, message symmetry is attained for each database and all desired bits are retrieved.

Next, all desired bits are getting decoded to obtain the whole message W_1. The desired bits are either mixed with cached bits or side information obtained from the databases, therefore they can be easily computed. For example, the user is able to decode a_3 from $a_3 + b_2$ and a_4 from $a_4 + c_2$ since b_2 and c_2 are in the cache. a_7 can be obtained from $a_7 + b_4 + c_4$ since $b_4 + c_4$ is side information from database 2.

Overall the user is downloading 8 bits. This can also be seen in figure 4: A query, i.e. $a_3 + b_2$ or $a_8 + b_3 + c_3$, is going to download one bit. Thus, the normalised download cost is 1 ($\frac{numberOfDownloadedBits}{messageSize} = \frac{8}{8} = 1$).

2.2.3 Communication Cost

Wei, Banawan and Ulukus[WBU18] determined inner and outer bounds for the optimal normalised download cost. Both inner and outer bounds are piece-wise linear functions in r (for fixed N , K) that consist of K line segments. r is the caching ratio, N the number of databases and K the number of messages. The inner bound is being specified by $r \leqslant \frac{1}{N^{K-1}}$ and the outer bound by $r \geqslant \frac{K-2}{N^2-3N+KN}$.

2.3 PIR From Databases With Limited Storage

The assumption that all N databases are storing all K messages can be difficult to implement in practise. Anyhow, the number of redundancy across the different databases is having an impact on the capacity of PIR. On one side, if there exists a high redundancy across the databases, the capacity is also very high. On the other side, if there is no redundancy at all, the only possible solution for PIR is to download all K messages. There have been many different studies, which are exploring the trade-off between the capacity and storage for PIR. *MDS-PIR* is referring to the case when each message is encoded by a maximum distance separable code and stored across the databases. With this approach the problem with colluding databases was turning out to be highly challenging and the capacity is still remaining unknown for general parameters. Another approach is that databases are using an arbitrary linear code for storing data. Other approaches are that databases are only storing a fraction of uncoded messages or the data is only partially replicated according to graph based structures.

In the following subsections a specific approach for private information retrieval from non replicated databases is explained briefly. More details can be found in [BU19]. The paper from Raviv, Tamo, and Yaakobi[RTY20] is closely related to [BU19] and additionally also covers the use case of colluding databases, but is not further addressed in this work.

2.3.1 Problem Description

A PIR problem with N non-replicating and non-colluding databases is being considered. The databases are being denoted by $\mathcal{D} = \{D_1, D_2, ..., D_N\}$. In total K messages are being stored and each message is getting stored across R different databases. In other words R is being the repetition factor of messages. The whole system, which is including all the databases is called the storage system. Each database is storing M different messages locally. The message set is being defined as $\mathcal{W} = \{W_1, W_2, ..., W_K\}$ and each individual message W_k is a vector of length L and is picked independently and identically distributed.

The storage system can be denoted by (K, R, M, N) and in a feasible one it holds that $KR = MN$. This means that the messages stored in total times the repetition factor for every message is equal to the amount of messages an individual database is storing locally times the number of all databases. For simplicity, this work is only considering a storage system with $M = 2$. The storage system is being represented by a R-regular graph. This implies, that each message is getting repeated R times across the storage system. Figure 5 is showing such a graph for a $(6, 3, 2, 9)$ storage system. The regular graph is being defined by (V, E), where $V = \mathcal{W} = \{W_1, W_2, ..., W_K\}$ the set of vertices and $E = \mathcal{D} = \{D_1, D_2, ..., D_N\}$

the set of edges, this means that the vertices are representing the messages and the edges the databases. If an edge D_j is drawn between messages W_m and W_k, this means that the contents of database D_j are $Z_j = \{W_m, W_k\}$. Consequently, database D_1 from example shown in figure 5 is containing the messages $Z_1 = \{W_2, W_1\}$. A full overview of the databases contents can be seen in table 1.

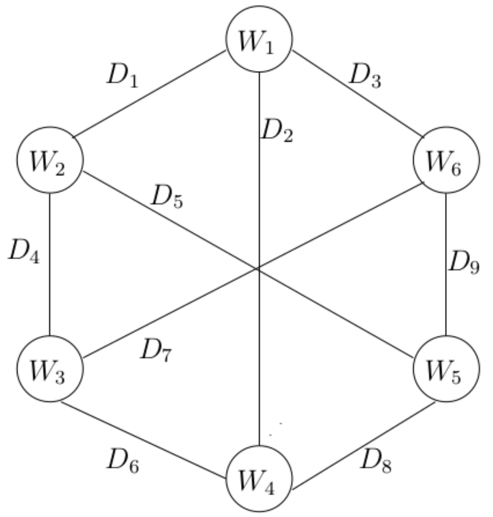


Figure 5: Graph structure for an example (6, 3, 2, 9) storage system. Taken from [BU19].

D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
W_1	W_1	W_1	W_2	W_2	W_3	W_3	W_4	W_5
W_2	W_4	W_6	W_3	W_5	W_4	W_6	W_5	W_6

Table 1: Contents of databases for example (6, 3, 2, 9) storage system specified by graph in figure 5. Taken from [BU19].

The goal of the user is to retrieve a message W_k without leaking any information about the identity of the message to any individual database. In achieving this goal, the user is sending one query to each database, thus in total N queries are being sent. Prior to retrieval, the user has no information about all the stored messages, therefore the queries are independent of the messages, hence a query is not revealing any information about a message. After that, every database is answering with a deterministic function composed by the query sent by the user and the contents stored in the database.

To ensure private retrieval of message W_k, the retrieval strategy of W_k must be indistinguishable to one from another message, e.g. W_{k+1}. Indistinguishability can also be seen as statistical equivalence. Another requirement is, that the user has be able to reconstruct W_k from all the received answers.

2.3.2 Example

For this example, a storage system with $N = 3$ databases is being considered. In total the system is storing $K = 3$ messages, namely W_1, W_2, W_3 and each message is getting replicated across $R = 2$ databases. Thus, each database is storing $M = 2$ messages. The message length is being defined by $L = 12$. The storage system is then denoted by (3, 2, 2, 3). The graph structure for this example is shown in figure 6 and contents of the databases are being presented in table 2.

In this example the user is wanting to retrieve message W_1. First, the user is permuting the indices of all messages (W_1, W_2, W_3) from the databases. The permuted version of W_1 can be described as a vector $(a_1, ..., a_{12})$, W_2 as $b_1, ..., b_{12}$ and W_3 as $c_1, ..., c_{12}$. A straightforward solution to this problem is to apply the scheme defined by Sun and Jafar in [SJ17]. Also a more in depth description regarding the solution scheme can be found there.

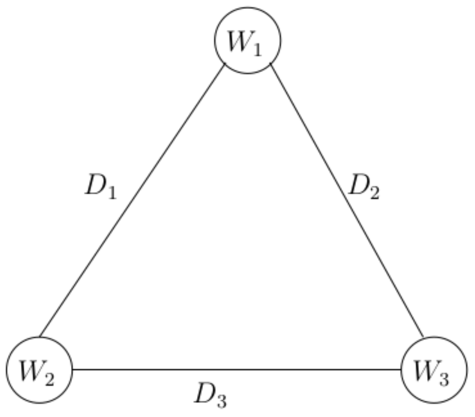


Figure 6: Graph structure for the (3, 2, 2, 3) system. Taken from [BU19].

Database 1 (D_1)	Database 2 (D_2)	Database 3 (D_3)
W_1	W_1	W_2
W_2	W_3	W_3

Table 2: Contents of databases for the (3, 2, 2, 3) system specified by graph in Fig. 6. Taken from [BU19].

Since every database is containing $M = 2$ messages, in repetition 1 (round 1) the user is able to download a single bit from each message from each database. More concretely, the user could download a_1, b_1 from database 1, a_2, c_1 from database 2 and b_2, c_2 from database 3. In the next step, the user could exploit the side information b_2, c_2 by downloading $a_3 + b_2$ from database 1 and $a_4 + c_2$ from database 2. Finally, the user could download $b_3 + c_3$ from database 3. Although the sum $b_3 + c_3$ has nothing to do with the desired message the user is wanting to retrieve ($W_1 = \{a_1, ..., a_{12}\}$), the user has to download these indices to satisfy the privacy constraint. Otherwise database 3 would be able to figure out that the desired message is W_1. Until now, an approach to the first repetition of querying was being described. It can also be seen in figure 7.

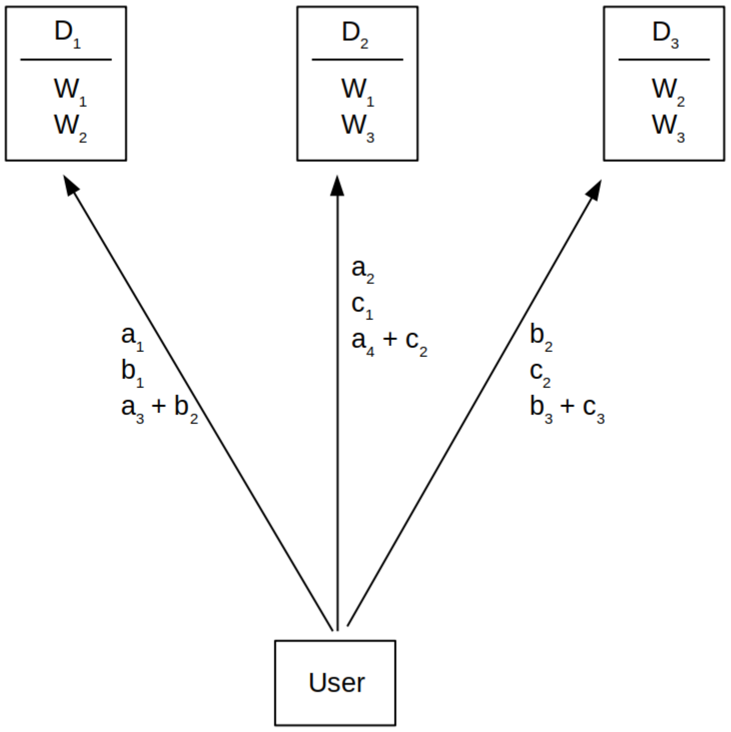


Figure 7: Approach for the first two rounds of querying for the (3, 2, 2, 3) system.

In the first repetition of this scheme the user is downloading 4 bits from the desired message W_1 out of the total 9 downloaded bits. Hence, the retrieval rate is $R_{pir} = \frac{4}{9}$. Despite this good retrieval rate, there is still room for improvements. The main source of inefficiency are the downloads from database 3, as the user is downloading three bits, but is only exploiting two of them (b_2, c_2). An improvement would be to download $b_1 + c_2$ and $b_2 + c_1$ from database 3. Doing this, the user is still able to decode the desired bits. First, the user is decoding c_2 by cancelling b_1 from $b_1 + c_2$ and b_2 by cancelling c_1 from $b_2 + c_1$. After that, it is possible to decode a_3 and a_4 by cancelling b_2 and c_2. A downside to this change is, that the privacy requirement is not holding anymore. Now, the user is downloading two bits from database

3 but three bits from database 1 and 2. Note, that until now only four bits from the desired message are being retrieved, namely $a_1, ..., a_4$, thus this problem can be solved in the next repetitions of the solution scheme. In repetition 2 the user is compressing the downloads from database 2 and downloads $a_7 + c_3$ and $a_8 + c_4$. In this repetition the downloads from database 1 and 3 are not compressed. In the last repetition the queries from database 1 are getting compressed, which results in downloading $a_{10} + b_5$ and $a_{11} + b_6$. Now the amount of queries made are equally distributed throughout the whole scheme. The complete query structure can be seen in table 3.

After the individual repetitions, the user is able to decode the answer strings retrieved from the databases and thus read the desired bits from message W_1. In repetition 1 the bits $a_1, ..., a_4$ can be decoded. In repetition 2, a_5 can be read directly, a_6 is decodable by cancelling b_4 from $a_6 + b_4$ and a_7 by cancelling c_3 from $a_7 + c_3$. Then, c_4 can be decoded from $b_3 + c_4$ and therefore a_8 can be decoded from $a_8 + c_4$. Decoding repetition 3 is following a similar approach.

This scheme is private because the query structure is being symmetric across all databases, and the indices of the bits from each message are getting chosen uniformly, independently and privately. Thus, all queries are equally likely. The achievable rate R_{pir} for the whole example is $\frac{12}{24} = \frac{1}{2}$.

In the paper [BU19], K. Banawan and S. Ulukus are generalising this example for arbitrary K messages and for fully-connected graphs. An in depth discussion about it can be read there.

2.3.3 Communication Cost

In their work[BU19], K. Banawan and S. Ulukus, are presenting a general upper bound for the retrieval rate for storage systems defined by R-regular graphs with $M = 2$ messages and arbitrary (K, R, N)

$$R_{pir} \leqslant min\{\frac{R}{N}, \frac{1}{1 + \frac{\delta}{R}}\}$$

where δ is the spread of the graph. This means that the upper bound of the retrieval rate is also dependent on the structure of the storage system.

K. Banawan and S. Ulukus are also showing that for a cyclic graph storage system, the PIR capacity is given by

$$C_{pir} = \frac{2}{K + 1}$$

	Database 1	Database 2	Database 3
rep. 1	a_1 b_1	a_2 c_1	
	$a_3 + b_2$	$a_4 + c_2$	$b_1 + c_2$ $b_2 + c_1$
rep. 2	a_5 b_3		b_4 c_3
	$a_6 + b_4$	$a_7 + c_3$ $a_8 + c_4$	$b_3 + c_4$
rep. 3		a_9 c_5	b_5 c_6
	$a_{10} + b_5$ $a_{11} + b_6$	$a_{12} + c_6$	$b_6 + c_5$

Table 3: Complete query structure for $K = 3$, $R = 2$, $M = 2$, $N = 3$. Taken from [BU19].

and for a fully-connected graph storage system with $M = 2$, the PIR capacity is given by

$$C_{pir} = \begin{cases} \frac{1}{2}, & K = 2, 3 \\ \frac{2}{K}, & K \geqslant 4 \end{cases}$$

2.4 PIR under Additional Abilities and Constraints for the Databases

Until now, this work was only considering privacy against individual databases, but in practise it could be that some databases are having the ability to collude. In addition to colluding, databases could exhibit Byzantine behaviour, meaning that they may return arbitrarily random or incorrect answers to the queries. Furthermore, databases themselves may require privacy. This means that the user is not learning anything further than the message it wishes to download. The solution to this problem is being called symmetric PIR (SPIR). The capacity of SPIR is proved to being smaller than the PIR capacity, because SPIR is more constrained than PIR. The SPIR scheme is requiring a shared common randomness among the databases. SPIR is often being used as a base in other problems involving symmetric privacy requirements among the participating parties. An example is private set intersection.

Databases may be restricted in their behaviour due to the way of accessing them or the way they are answering. For example, the communication channels which are used by the databases can be unstable. Dealing with that uncertainty, particular channel coding techniques have to be designed. In another setting the rate at which the user is able to download information from the databases can differ, leading to the need of asymmetric user access to the databases and moreover, to the need of asymmetric PIR schemes. Different physical distances between user and the databases can be a reason for this. Another set of practical constrains are arising if the database-to-user channels are being eavesdropped by an unauthorised entity. This is only one part of many security problems regarding communication over a network. Moving away from security issues, messages stored at databases in various lengths are causing constraints regarding the downloading semantics.

In the following sections a PIR problem with colluding servers and eavesdroppers, namely *ETPIR* is being discussed briefly. The content is taken from Wang et. al [WSS19].

2.4.1 Problem Description

The *ETPIR* problem is being comprised of K messages and N servers, where each server is storing all K messages. A user is wanting to retrieve one of the K messages privately. T out of the N servers are colluding and an eavesdropper is existing, which is listening to queries and answers from E servers.

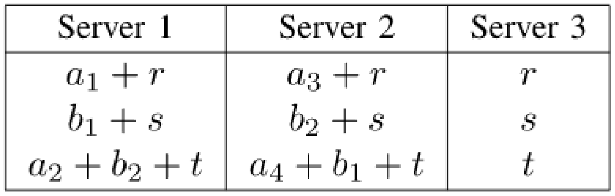
2.4.2 Protocol Design

In this section a high level description of the *ETPIR* protocol design is being shown. As described in the last subsection 2.4.1 an eavesdropper is able to tap on any E answers sent from the databases. Consequently, in *ETPIR* combining any number of E answers is not allowed to contain useful information about the desired message. With using a shared common randomness coded in combination of an (N, E)-MDS matrix and adding that to the answers as the noise, this can be fulfilled. After the noise is being cancelled, the desired symbols must be decodable. The constraint that the desired message can be retrieved privately with at most T servers colluding, namely T-privacy, must also be insured. This can be done in ensuring that every T servers are observing linearly independent queries.

For illustrating the design in a more understandable way, a small and simple example is being presented. In total $K = 2$ messages are getting stored on $N = 3$ servers where any $T = 2$ may collude and an eavesdropper is listening to the communication associated to $E = 1$ server. $\{a_i\}$ is denoting the linear combinations of message W_1 and $\{b_i\}$ the ones from message W_2. The message length is specified by $L = 4$. r, s, t are denoting three uniform independent symbols shared by the servers. These are unknown to the eavesdropper and the user.

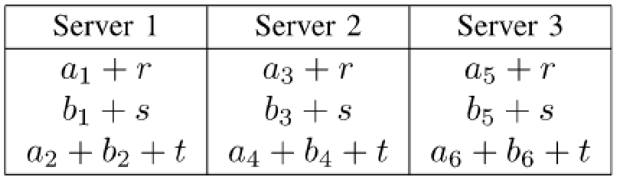
The goal of the user is to retrieve message W_1 privately. To start with, a similar scheme used in the sections before can be applied. The downloading symbols are shown in table 4.

In this example the answers from server 3 are representing pure noise. Nevertheless, this approach is correct because the user is able to cancelling out the noise and side information symbols and thus reading the desired symbols (a_i) is possible. A problem is that this approach is not being 2-private, meaning if two servers are colluding the retrieval of the desired message W_1 is not private anymore. If server 1 and server 2 are colluding they are able to figuring out that the b symbols are having the same indices, but the a symbols not, therefore they could imply that all a symbols are the desired bit.



Server 1	Server 2	Server 3
$a_1 + r$	$a_3 + r$	r
$b_1 + s$	$b_2 + s$	s
$a_2 + b_2 + t$	$a_4 + b_1 + t$	t

Table 4: Query structure approach for ETPIR*. Taken from [WSS19].*



Server 1	Server 2	Server 3
$a_1 + r$	$a_3 + r$	$a_5 + r$
$b_1 + s$	$b_3 + s$	$b_5 + s$
$a_2 + b_2 + t$	$a_4 + b_4 + t$	$a_6 + b_6 + t$

Table 5: Query structure approach for ETPIR *with enhanced privacy. Taken from [WSS19].*

Table 5 is showing a query structure with a privacy enhancing approach. Message symbols are getting added to the noise symbols from server 3, making it 2-private. As stated before each message is consisting of $L = 4$ symbols. These are getting picked from a finite field F_q with $q \geqslant 3$. W_1 and W_2 are denoting the column vectors comprised of the $L = 4$ symbols of each message. The user is privately choosing two $4x4$ matrices U_1 and U_2 independently and uniformly from a set of full-rank nxn matrices $GL_n(F_q)$. *(A matrix is said to having full rank if its rank equals the largest possible for a matrix of the same dimensions, which is the lesser of the number of rows and columns.)* With $a_{(1:4)}$ and $b_{(1:4)}$ the column vectors comprised of $a_1, ..., a_4$ and $b_1, ..., b_4$ are getting denoted and they are being generated by

$$a_{(1:4)} = U_1 W_1$$
$$b_{(1:4)} = U_2 W_2$$

As seen in table 5 there are values a_5, a_6, b_5, b_6, but the message length is gotten specified with $L = 4$. Hence, these symbols have to be constructed with the help of available symbols

$a_1, ..., a_4$ and $b_1, ..., b_4$. Table 6 is showing an approach doing exactly that. For example the symbol a_6 is getting constructed from $a_2 + a_4$ and the symbol b_6 from $2b_4 - b_2$.

Server 1	Server 2	Server 3
$a_1 + r$	$a_3 + r$	$a_1 + a_3 + r$
$b_1 + s$	$b_3 + s$	$b_3 + (b_4 - b_2) + s$
a_2+b_2+t	a_4+b_4+t	$(a_2+a_4)+(b_3-b_1)+(2b_4-b_2)+t$

Table 6: Query structure approach for ETPIR *with enhance privacy and available symbols. Taken from [WSS19].*

Removing the noise symbols is the first step in decoding the query answers. This is done by subtracting the answers of server 1 from the answers from server 2 and 3. Starting with server 2 - server 1 gives us $(a_3 + r) - (a_1 + r) => a_3 - a_1$, $(b_3 + s) - (b_1 + s) => (b_3 - b_1)$ and $(a_4 + b_4 + t) - (a_2 + b_2 + t) => (a_4 - a_2) + (b_4 - b_2)$. Calculating server 3 - serve 1 is done in a similar approach. Table 7 is showing the outcome of this step.

Server 2 $-$ Server 1	Server 3 $-$ Server 1
$(a_3 - a_1)$	a_3
$(b_3 - b_1)$	$(b_3 - b_1) + (b_4 - b_2)$
$(a_4 - a_2) + (b_4 - b_2)$	$a_4 + (b_3 - b_1) + 2(b_4 - b_2)$

Table 7: Query structure approach for ETPIR *after removing noise symbols. Taken from [WSS19].*

At this point correctness, 1-security and 2-privacy can be shown.

Correctness

Looking at table 7, one row of the queries is denoted as a *query row*. From the first *query row*, decoding a_1 and a_3 is possible. Obtaining $(b_3 - b_1)$ and $(b_4 - b_2)$ is possible from *query row* two. With this information cancelling out all the b values from *query row* three is possible. From the remaining symbols $(a_4 - a_2)$ and a_4, the desired values a_4 and a_2 are getting decoded. Message W_1 with $a_1, ..., a_4$ is now fully decoded, thus the scheme is correct.

1-security

1-security means that an eavesdropper is learning no information about the messages from the queries and answers of 1 server. Because *r,s,t* are uniformly distributed random variables independent of the message symbols it holds that *1-security* is holding, meaning from the answer of any server, the eavesdropper is learning nothing about the messages $W1, W2$.

2-privacy

This privacy constraint is guaranteed because the answers of any 2 servers are containing 4 linearly independent random combinations of symbols from each message. For example if server 1 and server 2 are colluding they are observing linearly independent combinations of messages W_1, W_2. This is true also if server 1 and 3, respectively server 2 and 3 are colluding. Because U_1 and U_2 are independently and uniformly chosen from all $4x4$ full-rank matrices, the mapping from W_1 to $a_{(1:4)}$ is independent and identically distributed, as is also the mapping from W_2 to $b_{(1:4)}$. The conclusion is that any 2 servers cannot distinguish W_1 and W_2 and the scheme is private.

In total that scheme is downloading 9 symbols, out of which 4 desired symbols can be decoded by the user. Thus, the achievable rate is $\frac{4}{9}$.

2.4.3 Generalisation

When larger parameters are needed to be handled, the main challenge is designing the linear combination coefficients of the desired and undesired symbols (named *precoding or the precoding matrices*). The following properties must hold:

- Any T servers observing linearly independent combinations of symbols from each message (to guarantee T-privacy)
- After the noise is gotten cancelled, cancelling the interference of undesired symbols can be done as well
- After the noise and the interference of undesired symbols are being cancelled out, the desired symbols are having full rank (to guarantee that the desired message can be recovered).

Wang et. al[WSS19] are also presenting a *ETPIR* protocol for general parameters of K, N, T and E where all above the properties are being satisfied.

2.4.4 Communication Cost

The main result of Wang et. al[WSS19] is a theorem showing the capacity of *ETPIR* as

$$C_{ETPIR} = \begin{cases} (1 - \frac{E}{N}) * (1 + \frac{T-E}{N-E} + (\frac{T-E}{N-E})^2 + ... + (\frac{T-E}{N-E})^{K-1})^{-1}, & \text{when } E < T; \\ 1 - \frac{E}{N}, & \text{when } E \geqslant T \end{cases}$$

where E is the number of servers the eavesdropper is listening to, N the number of servers, K the total number of messages and T the number of servers colluding.

2.5 Relaxed Privacy Notions

In the preceding sections, the goal was always perfect information-theoretic privacy. This is usually at the expense of high downloading costs and is not allowing tuning the PIR efficiency and privacy according to an applications requirements. In practise it could be desirable to trade privacy for communication efficiency, especially if retrieving messages is happening frequently. Selecting a desired leakage level and then designing a leakage constrained retrieval scheme that is guaranteeing such privacy while maximising the download efficiency is often an ideal approach. Repudiative information retrieval was being introduced as a solution to that problem. The repudiative property is being achieved if the probability that the desired message index was i, given the query is non-zero for every index i, meaning there is always some uncertainty at the database about the desired message index. A series of recent works were conducting this problem also for *SPIR*. Relaxing user privacy by allowing bounded mutual information between the queries and the corresponding requested message index is another approach. Recently, the model of latent-variable PIR was getting introduced and studied, where instead of requiring privacy for the message index, one may require privacy of data correlated with the message. Latent-variable PIR (LV-PIR) is aiming at allowing the user to efficiently retrieving one out of K messages, without revealing any information about the latent variable S. In contrast to the other examples shown in this work, only one database is used. In practise this is often the case, but using one database and providing privacy for the whole desired message is only leading to solutions where the whole database has to be downloaded. This is creating a vast amount of communication cost, which is often not acceptable in practise. Samy et. al[Sam+20] are arguing that the content queried by the user is often an intermediate data product that is exploited to infer latent data traits. Providing privacy for these traits is often enough. In the following sections their approach is being discussed briefly.

2.5.1 Problem Description

Considering a PIR problem where a set of $\mathcal{W} = \{W_1, W_2, ..., W_K\}$ of K independent messages, each of size L, are getting stored in one single database. The user is being interested in retrieving message W_θ, where θ is the message index, while hiding a latent variable S, which is taking one of T values from alphabet $S = \{s_1, s_2, ..., s_T\}$. The variable S is depending on the message index θ by the conditional probability $Pr(S = s_t|\theta = k)$ where $t \in [1 : T]$ and $k \in [1 : K]$. In other words, it is showing the probability of a latent variable s_t when a message with index $\theta = k$ is getting retrieved. A TxK matrix is capturing this conditional probability, see figure 8 for an example. Matrix H is fixed and publicly-known to the database and user.

Retrieving message W_θ is done by firstly submitting a query $Q^{(\theta)}$. The database is answering with $A^{(\theta)}$ as a function of the query $Q^{(\theta)}$ and the K messages. An *LV-PIR* scheme must satisfy correctness, meaning the user is able to recover the requested message W_θ from $A^{(\theta)}$. And it must satisfy the latent-variable privacy constraint, meaning the query answer $A^{(\theta)}$ is not leaking any additional information about the latent variable S than what is known by the prior distribution of S (from probability matrix H).

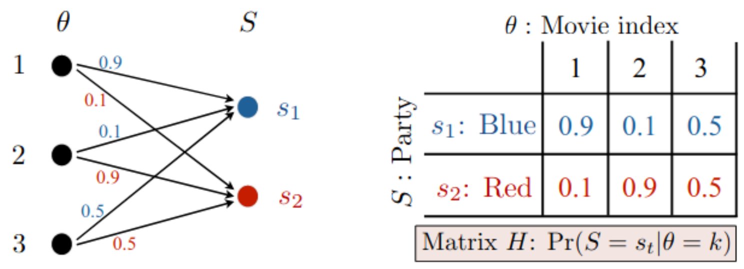
2.5.2 Example

Considering Alice is using a streaming service, which is hosting three movies labeled by index $\theta \in \{1, 2, 3\}$. Access patterns of movies are getting analysed to infer if Alice is leaning towards the blue or the red party. Here, the latent variable S is only taking two values, namely blue and red. The conditional probability distribution $Pr(S|\theta)$ is describing the relationship between content of a movie and affiliation to a party. It is expressed as a matrix H of size $2x3$, with the (t, k) element being $h_{tk} = Pr(S = s_t|\theta = k)$. Figure 8 is showing a sample realisation. For example, if Alice is retrieving the movie with index $\theta = 1$, she is leaning with a probability of 0.9 to the blue party and with a probability of 0.1 to the red party. If Alice is retrieving all movies she is getting associated with each affiliation (blue or red) equiprobably. An advantage to the classical PIR is that with this approach the same privacy constraint is being satisfied with a lower download cost. If Alice is retrieving movie 3, then querying set 3 is sufficient to hide Alice's political affiliation. Although the index θ of the query is getting leaked, it is not revealing any additional information with respect to the latent variable S, which is remaining uniformly distributed. For the case that Alice is wanting to retrieve either movie 1 or movie 2, querying the set $\{1, 2\}$ is yielding a marginal uniform distribution on S, hence is also sufficient to hide Alice's political affiliation. This small example is demonstration two important things:

- by considering the privacy of the latent variable, the downloading cost of PIR can be reduced even for a single database
- the cost is becoming dependent on the desired message k and the conditional distribution $Pr(S|\theta)$ captured by H.

In a formal way the latent-variable privacy constraint can be described as

$$Pr(S = s_t|Q^{(\theta)} = q) = Pr(S = s_t) = 0.5, \forall t, q$$



	1	2	3
s_1: Blue	0.9	0.1	0.5
s_2: Red	0.1	0.9	0.5

Figure 8: Latent variable PIR. Sample realisation of the characteristic matrix H. The random variable θ is representing the message index whereas S is representing the latent variable. Taken from [Sam+20].

which is saying that the probability s_t equals q when executing query $Q^{(\theta)}$, is equal to the probability of just S being a specific value s_t and both probabilities are equal to 0.5. This has to hold for all latent variables indexed by t and all queries q.

Considering again the three messages indexed by $\theta = \{1, 2, 3\}$ from figure 8. These are being related to the latent variable S taking two values $s1, s2$. The conditional probability matrix being defined by H with

$$H = \begin{bmatrix} 0.1 & 0.9 & 0.5 \\ 0.9 & 0.1 & 0.5 \end{bmatrix}$$

Using H, $Pr(S = s_t) = 0.5 \; \forall t \in \{1, 2\}$ can be shown. Retrieving message W_3 is leading to the same distribution over S, which is 0.5. Whenever $\theta = 1$ or $\theta = 2$, downloading both messages (W_1, W_2) is also matching the prior distribution of 0.5. Based on this discussion, the following query structure can be derived for the specified H

$$Q^{(\theta)} = \begin{cases} \{1, 2\}, & \text{when } \theta = 1 \text{ or } \theta = 2 \\ \{3\}, & \text{when } \theta = 0 \end{cases}$$

2.5.3 Communication Cost

Let $D_H(k)$ denoting the number of downloaded bits for a desired message W_k and a characteristic matrix H. Further, $D_H = \frac{1}{K}\sum_{k=1}^{K} D_H(k)$ is describing the average number of downloaded bits. The presented query structure is showing that the downloading cost for each message is $D_H(3) = 1$ and $D_H(1) = D_H(2) = 2$. Consequently the average downloading cost is $D_H = \frac{1}{3} * (2 + 2 + 1) = \frac{5}{3} < 3$. This is lower than the downloading cost of a classical PIR scheme.

2.6 Take-Away

This section should answer why the presented generalisations of PIR are needed. Making PIR usable in a real world scenario is the main reason, this can be largely done in reducing the download costs. Additionally, real world problems like colluding databases and security issues are not included in the canonical PIR scheme. Table8 is listing advantages of the individual generalisations.

Generalisation	**Advantage**
MPIR	Downloading messages one-by-one is not the optimum. Downloading messages jointly is more efficient, even after considering the downloading of undesired bits from the databases which has to be done.
Cache or Side Information Aided PIR	It is leading to reduced download costs, because of the additional information saved locally.
PIR From Databases With Limited Storage	All databases do not have to save all messages anymore. Thus, the storage costs are reduced.
PIR under Additional Abilities and Constraints for the Databases	Constraints like colluding and byzantine databases, enforcing privacy on the database and unstable communication channels are simulating real world problems.
Relaxed Privacy Notions	Streghens practical usage, because when trading privacy for communication efficiency PIR is getting easier to implement.

Table 8: Generalisation Advantages

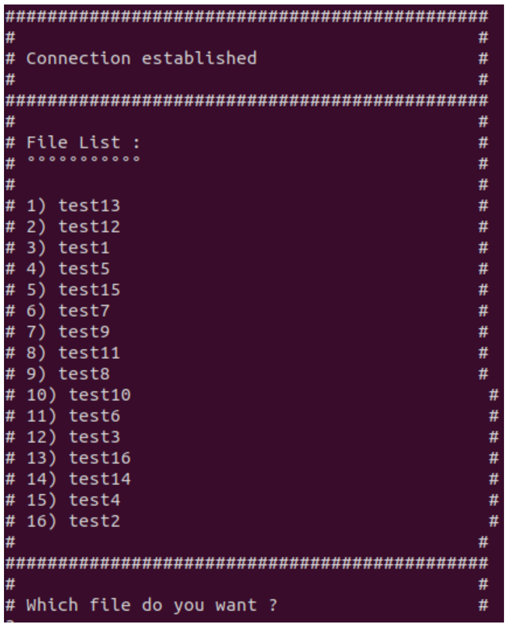
3 PIR in practise

During research for this work no PIR specific libraries in the commonly known programming languages could be found. Also searching for PIR specific functionalities in widely known database and cloud systems was not successful. Most of PIR related implementations were being done for research reasons, where a Paper introducing a new PIR protocol was being used as model. With the help cryptographic libraries it was possible to put some protocols to work. The next sections are consisting of brief descriptions of some PIR implementations.

3.1 XPIR

XPIR[Agu+15] is allowing a user to privately download an element from a database. It should only be used for research purposes and not in production. The PIR setting implemented is working with a single database and is computationally private. For making the retrieval private *Homomorphic Encryption* is used. XPIR is coming with a client and a server, which is making testing on simple examples very convenient. Such a simple test is being shown in this section. Installation instructions and more information can be found on the project's GitHub page[XPI].

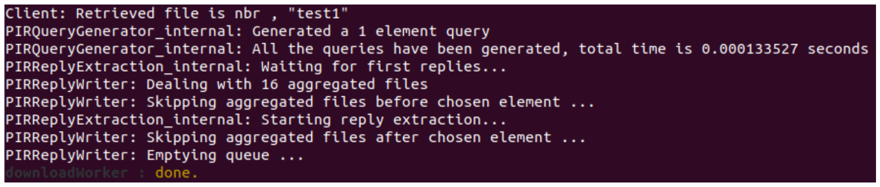
The server's database is simulated by a directory containing multiple files. Each file is representing a database element. The client is automatically connecting to the server and is able to choose one desired file from all the files in the database (see figure 9).



```
##############################################
#                                            #
# Connection established                     #
#                                            #
##############################################
#                                            #
# File List :                                #
# °°°°°°°°°°°                                #
#                                            #
# 1) test13                                  #
# 2) test12                                  #
# 3) test1                                   #
# 4) test5                                   #
# 5) test15                                  #
# 6) test7                                   #
# 7) test9                                   #
# 8) test11                                  #
# 9) test8                                   #
# 10) test10                                  #
# 11) test6                                   #
# 12) test3                                   #
# 13) test16                                  #
# 14) test14                                  #
# 15) test4                                   #
# 16) test2                                   #
#                                            #
##############################################
#                                            #
# Which file do you want ?                   #
```

Figure 9: XPIR console logs: Server asking for file to retrieve

After choosing a file, the client is running an optimiser which is deciding on the best cryptographic and PIR parameters. Next, an encrypted PIR query is getting sent to the server. The server is then computing an encrypted PIR reply and is sending it back to the client. Finally, the client is decrypting the reply and is storing the resulting content into a file. Running the program is not giving away much information, the console outputs are mostly logs listing the current executed steps as can be seen in figure 10.



```
Client: Retrieved file is nbr , "test1"
PIRQueryGenerator_internal: Generated a 1 element query
PIRQueryGenerator_internal: All the queries have been generated, total time is 0.000133527 seconds
PIRReplyExtraction_internal: Waiting for first replies...
PIRReplyWriter: Dealing with 16 aggregated files
PIRReplyWriter: Skipping aggregated files before chosen element ...
PIRReplyExtraction_internal: Starting reply extraction...
PIRReplyWriter: Skipping aggregated files after chosen element ...
PIRReplyWriter: Emptying queue ...
downloadWorker : done.
```

Figure 10: XPIR console logs: Client executing the retrieval.

3.2 Checklist

Checklist[KC21] is a system for private blocklist lookups, meaning a pair of servers is holding a set B of blocklisted strings and a client is holding a private string s. The client is wanting to learn whether s is in the blocklist B without revealing its private string s to the servers. This scheme is introducing an offline/online approach for multiple server PIR. The source code can be found on their GitHub page[Che].

3.3 MuchPIR

MuchPIR[Muc] is an extension for a Postgres database using *Homomorphic Encryption*. It is also including additional privacy use cases:

- **Limited String Search:** searching for small words instead of indexing rows
- **Exact Match Search:** reducing the search space
- **Unique ID Search:** searching data based on unique IDs
- **Accumulation:** aggregating data on columns in a way that is indistinguishable from PIR

3.4 lattice-crypto

lattice-crypto[lat] is a lattice-based cryptographic PIR protocol based on the paper[AG07]. Although it is not the newest implementation, the basic usage of cryptographic functions for PIR systems is being shown.

The PIR scheme is relying on the idea of controlled noise addition. The starting point is a secret random $[N, 2N]$ matrix M of rank N over a field. Using this matrix, a set of different matrices can be generated by multiplication on the left side with invertible random matrices. Next, this matrices are getting disturbed by the user through introducing noise in half of the matrices' columns, obtaining respectively softly disturbed matrices (SDMs) and hardly disturbed matrices (HDMs).

The user is sending a set of SDMs and one HDM, which is associated to the desired element, to the database. After that, the database is inserting each of its elements in the corresponding matrix in using a multiplicative operation and is summing up all the rows of the resulting matrices, resulting in the database response, a single noisy vector. Using the unmodified columns of the matrices sent in the request, the user is able to find the noise associated with the returned noisy vector. If the soft noise multiplied by the total noise factor is much smaller than the hard noise, filtering it out is possible and the user is able to retrieve the information associated to the noise of the HDM matrix, thus the desired element.

The next sections should give an idea of the implementation and how implementing certain parts looks like.

3.4.1 Database generation

Listing 1 is showing the generation of the database. n is the number of elements and l the number of bits per element. To ensure the same length on all elements they are getting padded to the length l. For simplicity the elements are getting generated randomly. The result is a list of n elements.

3.4.2 Request generation

Listing 2 is showing the generation of a query. In line 2 and 3 two random matrices are getting generated with the help of a before generated finite field p and then merged together in line 4. The variable M is then denoting the secret random matrix which is used as starting point. In the for-loop starting with line 9, the SDMs and one HDM are getting generated. The HDM is generated for index $i0$, which is the index of the desired element.

3.4.3 Server-side response

Creating the response is shown in listing 3. The main functionality is implemented in the for-loop starting with line 5. There, each database element is being inserted in the corresponding query matrix using a multiplicative operation. The result is an aggregated version of the matrices created in the for loop, one single vector.

3.4.4 Response extraction

Listing 4 is showing the extraction of the server's response. In lines 4 to 14 the user is recovering the noise included in the vector. This can be done with the help of the unmodified columns. Inside the for-loop starting in line 16 the user is unscrambling and filtering out the noise to obtain the desired information (element).

Listing 1: Database generation

```
def gen_db(n=2, l=6):
    db = []

    for i in range(n):
        element = str(bin(random.getrandbits(l)))[2:]

        while len(element) < l:
            element = "0" + element
        db.append(element)

    return db
```

Listing 2: Request generation

```
def gen_request(n, N, i0, l0, q, p):
    A = gen_rand_matrix(N, N, is_invertible=True, finite_field=p)
    B = gen_rand_matrix(N, N, is_invertible=False, finite_field=p)
    M = np.hstack([A, B])
    scrambler = gen_scrambler(N, p)
    new_columns = column_permutation(2*N)
    request = []

    for i in range(n):
        rand_matrix = gen_rand_matrix(N, N, is_invertible=True,
        finite_field=p)

        M11 = reduce_matrix(np.dot(rand_matrix, M), p)

        if i == i0:
            D = gen_noise(N, hard=True, q=q)
        else:
            D = gen_noise(N, hard=False, q=None)

        noise = reduce_matrix(np.dot(D, scrambler), p, is_noise=True)

        M1 = reduce_matrix(np.hstack([M11[:, :M.shape[1] / 2],
        M11[:, M.shape[1] / 2:] + noise]), p)
        request.append(permute_columns(reduce_matrix(M1, p),
        new_columns))

    return np.array(request), new_columns, A, B, scrambler
```

Listing 3: Server-side response

```
def server_reply(request, db, l0, p):
    N = len(db[0]) / l0
    response = np.zeros([N*len(db), 2*N], dtype=int)

    for i, query in enumerate(request):
        for row in range(N):
            db_substring = int(db[i][row*l0:row*l0 + l0], 2)
            response[i*N + row] = reduce_vector(
            db_substring*request[i][row], p)

    return reduce_vector(np.sum(response, axis=0), p)
```

Listing 4: Response extraction

```
def response_extraction(
    response, column_permutation, N, A, B, p, q, scrambler, l0
):
    disturbed_vector = column_reordering(response, column_permutation)
    undisturbed_half = disturbed_vector[:len(disturbed_vector) / 2]

    A_inverse = matrix_inversion_cofactor(A, p)
    undisturbed_vector = np.concatenate([undisturbed_half,
    reduce_vector(np.dot(np.dot(undisturbed_half, A_inverse), B), p)])
    scrambled_noise = reduce_vector(disturbed_vector - undisturbed_vector
    p)[N:]
    unscrambled_noise = reduce_vector(np.dot(scrambled_noise,
    matrix_inversion_cofactor(scrambler, p)), p)
    response = ""

    for each in unscrambled_noise:
        bitstring = str(bin(recover_bits(each, q, p)))[2:]
        while len(bitstring) < l0:
            bitstring = "0" + bitstring
        response = response + bitstring
    return response
```

References

[AG07] Carlos Aguilar-Melchor and Philippe Gaborit. “A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol” (2007). URL: https://eprint.iacr.org/2007/446.

[Agu+15] C. Aguilar-Melchor et al. “XPIR: Private Information Retrieval for Everyone” (2015). DOI: 10.1515/popets-2016-0010.

[BU18] K. Banawan and S. Ulukus. “Multi-message private information retrieval: Capacity results and near-optimal schemes” (2018). DOI: 10.1109/TIT.2018.2828310.

[BU19] K. Banawan and S. Ulukus. “Private information retrieval from non-replicated databases” (2019). DOI: 10.1109/ISIT.2019.8849399.

[Che] Checklist (). URL: https://github.com/dimakogan/checklist.

[KC21] D. Kogan and H. Corrigan-Gibbs. “Private blocklist lookups with checklist” (2021). URL: https://www.usenix.org/conference/usenixsecurity21/presentation/kogan.

[KO97] E. Kushilevitz and R. Ostrovsky. “Replication is not needed: Single database, computationally-private information retrieval.” (1997). DOI: 10.1109/SFCS.1997.646125.

[lat] lattice-crypto (). URL: https://github.com/adriansoghoian/lattice-crypto.

[Muc] MuchPIR (). URL: https://github.com/ReverseControl/MuchPIR.

[RTY20] N. Raviv, I. Tamo, and E. Yaakobi. “Private information retrieval in graph-based replication systems” (2020). DOI: 10.1109/TIT.2019.2955053.

[Sam+20] I. Samy et al. “Latent-variable Private Information Retrieval” (2020). DOI: 10.1109/ISIT44484.2020.9174451.

[SJ17] H. Sun and S. A. Jafar. “The capacity of private information retrieval.” (2017). DOI: 10.1109/TIT.2017.2689028.

[Ulu+22] S. Ulukus et al. “Private Retrieval, Computing, and Learning: Recent Progress and Future Challenges” (2022). DOI: 10.1109/JSAC.2022.3142358.

[WBU18] Y. Wei, K. Banawan, and S. Ulukus. “Cache-Aided Private Information Retrieval with Partially Known Uncoded Prefetching: Fundamental Limits” (2018). DOI: 10.1109/JSAC.2018.2844940.

[WSS19] Q. Wang, H. Sun, and M. Skoglund. “The capacity of private information retrieval with eavesdroppers.” (2019). DOI: 10.1109/TIT.2018.2884891.

[XPI] XPIR (). URL: https://github.com/XPIR-team/XPIR.