A Random Server Model for Private Information Retrieval
or
How to Achieve Information Theoretic PIR Avoiding Database Replication[⋆]

Yael Gertner[1], Shafi Goldwasser[2], and Tal Malkin[2]

[1] Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA 19104, USA,
`ygertner@saul.cis.upenn.edu`
[2] Laboratory for Computer Science, Massachusetts Institute of Technology,
545 Technology Square, Cambridge, MA 02139, USA
`{shafi,tal}@theory.lcs.mit.edu`

Abstract. Private information retrieval (PIR) schemes enable users to obtain information from databases while keeping their queries secret from the database managers. We propose a new model for PIR, utilizing auxiliary random servers to provide privacy services for database access. In this model, prior to any on-line communication where users request queries, the database engages in an initial preprocessing setup stage with the random servers. Using this model we achieve the first PIR information theoretic solution in which the database does not need to give away its data to be replicated, and with minimal on-line computation cost for the database. This solves privacy and efficiency problems inherent to all previous solutions.
In particular, all previous information theoretic PIR schemes required multiple replications of the database into separate entities which are not allowed to communicate with each other; and in all previous schemes (including ones which do not achieve information theoretic security), the amount of computation performed by the database on-line for every query is at least linear in the size of the database. In contrast, in our solutions the database does not give away its contents to any other entity; and after the initial setup stage which costs at most $O(n \log n)$ in computation, the database needs to perform only O(1) amount of computation to answer questions of users on-line. All the extra on-line computation is done by the auxiliary random servers.

1 Introduction

Private Information Retrieval (PIR) schemes provide a user with information from a database in a private manner. In this model, the database is viewed as an n-bit string x out of which the user retrieves the i-th bit x_i, while giving the database no information about his query i. The notion of PIR was introduced in [10], where it was shown that if there is only one copy of the database available then $\Omega(n)$ bits of communication are needed (for information theoretic user privacy). However, if there are $k \geq 2$ non-communicating copies of the database, then there are solutions with much better (sublinear) communication complexity. *Symmetrically Private Information Retrieval (SPIR)* [11] addresses the database's privacy as well by adding the requirement that the user, on

[⋆]An earlier version of this work appears as MIT technical report MIT-LCS-TR-715. This work was done with the support of DARPA grant DABT63-96-C-0018.

the other hand, cannot obtain any information about the database in a single query except for a single physical value.

Two major problems arise with all existing solutions: firstly, in order to achieve information theoretic privacy all previous solutions call for replicating the database into several non-communicating copies, which constitutes a serious privacy problem (as discussed below); and secondly, even though the communication complexity is sublinear, the amount of computation that the database engages in is *linear* in the size of the database for *every* query of the user. It seems unreasonable to expect a commercial database to distribute copies of its data to non-communicating entities and to perform linear amount of computation per single query solely for the purpose of the user's privacy.

In this paper, we introduce a new model for PIR (or SPIR), which allows us to achieve significant improvements both in terms of security (circumventing the replication problem) and in terms of computational complexity.

The first enhancement to the PIR model is the use of *auxiliary random servers*, whose contents are independent of the contents of the database. This separates the task of information retrieval from the task of providing privacy. We use only a single copy of the original data (the database owner itself), who does not engage in any complex privacy protocols, while all the privacy requirements are achieved utilizing the random servers, who do the work instead of the database. The random servers do not gain any information about the database or the user in the process (this is in contrast to the old model, where a database who wants to hire an agent to do all the privacy work for it must give away all its information to that agent).

The second enhancement to the model, is that we divide the PIR computation into two stages: the setup stage, which takes place ahead of query time and does not involve the user, and the on-line stage, during which the user performs his various queries. The purpose of this split of computation is to allow much of the computation to be done once ahead of time, so that during the on-line stage the database is required to engage in minimal computation and communication.

Using this model, we construct straightforward and efficient protocols for solving the two problems described above. We achieve information theoretic privacy without data replication, and we minimize the on-line computation required from the database.

Below we describe these problems and their solutions in more detail.

1.1 Problems With The Previous PIR Model

Protocols for PIR and SPIR schemes, guaranteeing information theoretic privacy, appeared in [10, 3, 15, 11]. These solutions are based on the idea of using multiple copies of the database that are not allowed to communicate with each other. This allows the user to ask different questions from different copies of the database and combine their responses to get the answer to his query, without revealing his original query to any single database (or a coalition). The recent PIR scheme of [14] uses a single database, but guarantees only computational privacy under the assumption that distinguishing quadratic residues from non-residues modulo composites is intractable. In fact, it was shown in [10] that using a single database makes it impossible to achieve information theoretic privacy with sublinear communication complexity.

Unfortunately, the common paradigm behind all the solutions that guarantee information theoretic privacy — the replication of the database in multiple separated locations — introduces a serious privacy problem to the database, *the data replication problem*. Namely, the database owner is required to distribute its data among multiple foreign entities, each of which could be broken into, or could use the data and sell it to users behind the legitimate owner's back. This is particularly problematic since the database cannot communicate with any of the other copies. Since this replication is used to protect the user's interest, it is doubtful that real world commercial databases would agree to distribute their data to completely separated holders which they cannot communicate with. Viewed from the user's standpoint, it may be doubtful that users interested in privacy of their queries would trust copies of the same database not to communicate with each other.

Secondly, the paradigm used in all existing PIR schemes requires the database to actively participate in a complex protocol in order to achieve privacy. The protocol is complex both in terms of the *computation* necessary for the database to perform in order to answer every question of the user, and in the less quantifiable *lack of simplicity,* compared to the classical lookup-the-query-and-answer approach.

In particular, in all of the existing solutions (whether using a multiple number of databases or a single one), each database performs a computation which is at least *linear* in the size of the database in order to compute the necessary answer for *each* question of the user. This is in contrast to the user's computation and the communication complexity, which are at most sublinear per query. In the single database case (computational privacy) the complexity of the database computation is a function of both the size n of the database and the size of the security parameter underlying the cryptographic assumption made to ensure privacy. Specifically, in the single database solution of [14], the computation takes a linear number of multiplications in a group whose size depends on the security parameter chosen for the quadratic residuosity problem.[1] Again, the overhead in computational complexity and lack of simplicity of existing schemes make it unlikely to be embraced as a solution by databases in practice.

1.2 New Approach: The Random Server Model for PIR

We introduce a new model for PIR, which allows for information theoretic privacy while eliminating the problems discussed above. Since it is not possible to use a single database and achieve sublinear communication complexity information theoretic results ([10]), we must still use a multiple database model. The crucial difference is that the multiple databases are *not* copies of the original database. Rather, they hold auxiliary random strings provided by, say, WWW servers for this purpose. These auxiliary servers contain strings each of which cannot be used on its own[2] to obtain any information about the original data. Thus, an auxiliary server cannot obtain information about the data, sell it to others, or use it in any other way. Instead, they may be viewed as servers who are selling security services to ordinary current day databases.

The database owner, after engaging the services of some servers for the purpose of offering private and secure access to users, performs an initial setup computation with

[1] For example, to achieve communication complexity $O(n^\epsilon)$ the security parameter is of size $O(n^{\epsilon^2})$ and the number of multiplications is $O(\frac{1}{\epsilon}n)$.

[2] or in extended solutions, in coalition with others (the number if which is a parameter determined by the database)

the auxiliary servers. The servers are then ready to assist users in retrieving information from the database owner efficiently and privately during the on-line stage. Periodic re-initialization (setup stage) may be required in some frequency specified by the protocol. Typically this frequency will be once in a large number of queries (e.g. sublinear), or, if no periodic re-setup is required, then only when the database needs to be updated.[3]

We differentiate between two kinds of random servers: universal and tailored. Universal random servers are servers whose contents may be determined in advance, even before the setup stage, without any connection to a particular database. Tailored random servers are those who store some random string specific to the database they are serving, namely those whose content is determined during the setup stage.

One of the parameters of a PIR scheme is how many servers of each kind are required. Clearly, universal servers are preferable, since they can be prepared in advance and therefore are more efficient and more secure. Moreover, since they do not need to store any data specific to the database they are serving, they could potentially be used for multiple databases at the same time. Indeed, our strongest definition of privacy (total independence, below) requires that *all* servers involved are universal.

We define two new kinds of privacy for the database in this setting (formal definitions are in the next section), *independence*, and *total independence*.

Independence informally means that no server can get any information about the original data of the database owner. Thus, the real data is distributed among all the servers in a private way, so that no single one gets any information about it (this can be generalized to t-independence, for any coalition of upto t servers).

Total independence informally means that even *all* the auxiliary servers *jointly* do not contain any information about the original data (namely all servers are universal).

Clearly, total independence implies independence. Indeed the solutions we propose to address the latter are simpler than the ones to address the former.

1.3 Our Results

We provide general reductions, starting from any PIR scheme, to schemes that achieve independence or total independence and low database computation complexity, while maintaining the other privacy properties of the underlying starting scheme (namely user privacy and database privacy). The database computation complexity on-line is reduced to a simple O(1) look-up-the-query computation, or for some of our schemes to no computation at all. Instead, the servers assume responsibility for all computations required for privacy in the starting scheme. The user computation complexity stays the same as in the starting scheme, and (using existing solutions) it is already bounded by the communication complexity (sublinear). Therefore, we concentrate on reducing the database's computation, which in all previous schemes has been at least linear.

Let us describe our results.

Let S be a PIR scheme which requires k copies of the database,[4] and has communication complexity of C_S. We provide two sets of schemes (all terms used below are defined in section 2).

[3] Note that also in the old replication model, reinitialization is required when the database changes.

[4] Note that creating k copies of the database may be viewed as a setup stage of complexity $O(n)$.

Schemes Achieving Independence We state the result both for the interesting special case of $t = 1$ (i.e. independence), and the most general case for any t.

- A scheme achieving independence and maintaining the other privacy properties of S. The scheme uses k tailored and k universal servers, with communication complexity $O(C_S)$, and no database participation in the on-line stage (i.e. no computation).
- A scheme achieving t-independence (for any $t \geq 1$) and maintaining the other privacy properties of S. The scheme uses k tailored and tk universal servers, with communication complexity $(t+1)C_S$, and no database participation in the on-line stage.

Setup stage: The complexity of the setup stage is $O(n)$. The number of tailored servers, who need to obtain some string during setup stage, is k (the same as in the starting scheme S).

Schemes Achieving Total Independence There are two variants here.

- A (basic) scheme achieving total independence and database privacy, and maintaining user privacy up to equality between repeated queries.[5] The scheme uses $\max(k, 2)$ universal servers and the database owner, with at most $O(C_S \log n)$ communication complexity, and $O(1)$ database computation complexity.
- A scheme achieving total independence and maintaining the other privacy properties of S (in particular complete user privacy). The scheme uses $\max(k, 2)$ universal servers and the database owner, with at most $O((m + C_S) \log n)$ communication complexity, where the servers and the database need to engage in a re-setup after every m queries. The database computation is $O(1)$.

Setup stage : The complexity of the setup stage is $O(n \log n)$. Note that all servers are universal, namely they could be prepared ahead of time, and do not change during setup stage.

Tradeoff between the two versions: In the basic version the database can detect repeated queries, but cannot gain any other information about the user's queries. This is dealt with in the final scheme, where total independence is achieved preserving complete user privacy. The price for eliminating detection of repeated queries is that re-setup has to be performed every m queries. The value of m, the frequency of reinitialization, is a parameter chosen to optimally trade off the frequency and the communication complexity. A suitable choice for existing schemes is to choose $m = C_S = n^\epsilon$ the size of the communication complexity for a single query, so that the over all communication complexity does not increase by more than a logarithmic factor, and yet a sublinear number of queries can be made before reinitialization. Choosing between the two versions should depend on the specific application: preventing the database from detecting equal questions, or avoiding reinitialization. Note also that the first version adds database privacy even if the underlying S was not database private.

Main Idea: Note that total independence guarantees that all the auxiliary servers jointly do not contain any information about the data. So how can they assist the database

[5] namely, the only information that the database can compute is whether this query has been made before.

at all? The idea is that during the setup stage, a setup protocol is run amongst the database and the universal random servers, at the end of which the *database* is the one which changes appropriately to ensure the privacy and correctness properties for the on-line stage. During the on-line stage, as before, the user communicates with the random servers and with the database to privately extract the answer to his query.

1.4 Related Work

PIR was originally introduced by [10], who were only concerned with protecting the user's (information theoretic) privacy. In particular, for a constant number k of database copies, [10] with further improvement in [3] (for the case $k > 2$), achieve information theoretic user security with communication complexity of $n^{\frac{1}{2k-1}}$, where n is the length of the data (in bits).

Recently, [11] extended PIR to SPIR (*Symmetrically private information retrieval*), where the privacy of the data (with respect to the user) is considered as well. They use a model where the multiple databases may use some shared randomness, to achieve reductions from PIR to SPIR, paying a multiplicative logarithmic factor in communication complexity.

The work in [9] considers computational privacy for the user, and achieves a 2 database scheme with communication complexity of n^ϵ for any $\epsilon > 0$, based on the existence of one way functions. As mentioned earlier [14] relies on a stronger computational assumption – the quadratic residuosity problem – to achieve a 1-database PIR scheme with computational privacy and communication complexity of n^ϵ for any $\epsilon > 0$.

The work in [15] generalizes PIR for private information storage, where a user can privately read and write into the database. This model differs from ours, since in our model users do not have write access into the database. Still, some connection between the two models can be made, since one might consider a storage model where the first n operations are restricted to be private write (performed by the database owner), and all operations thereafter are restricted to be private reads (by users). This results in a model compatible to our model of independence (although this approach cannot lead to total independence). We note that [15] independently[6] use a basic scheme which is essentially the same as our basic RDB scheme of section 3.1. However, they use the scheme in their modified model (where users have write access), and with a different goal in mind, namely that of allowing users to privately read and write into the databases.

None of the above PIR and SPIR works consider the data replication problem.

Recently, independently from our work, [7] had suggested the commodity based model for cryptographic applications, which relies on servers to provide security, but not to be involved in the client computations. Although this model is related to ours we stress here some important differences. First, their model engages the servers only in the task of sending one message (commodity) to each client, without any interaction. In contrast, our model stresses the interaction of the servers with the clients for the purpose of reducing the computational complexity. Second, our model, unlike theirs, is designed specifically for PIR, and solves problems which were not previously addressed.

[6] our results of section 3 were actually done previously to the publication of [15]

Organization Section 2 introduces the relevant definitions and notation used. In section 3 we describe schemes achieving independence, and in section 4 we describe schemes achieving total independence.

2 Notation and Definitions

The Information Retrieval Model: The *data string* is a string of n bits denoted by $x = x_1, \ldots, x_n$. The user's *query* is an index $i \in \{1, \ldots, n\}$, which is the location of the bit the user is trying to retrieve. The *database* (also referred to as the original database, or the database owner) is denoted by D.

An *information retrieval* scheme is a protocol consisting of a *setup stage* and an *on-line* stage. During the setup stage, the auxiliary random servers are chosen, and possibly some other setup computation is performed. During the on-line stage, a user interacts with the servers and possibly also with the original database in order to obtain his query. At the end of the interaction, the user should have the bit x_i. In all our schemes, the on-line stage consists of a single round.

The Random Servers: There are two kinds of auxiliary servers: *Universal* and *Tailored*. The universal servers contain completely random data that can be prepared ahead of time independently of the particular database in mind. The tailored servers on the other hand are each independent of the database, but their content should be prepared for a particular database (during the setup stage), since the combination of all servers together is dependent on the specific database. One of the parameters for an information retrieval scheme is how many servers of each kind are required.

We require that all servers are separate, in the sense that they are not allowed to communicate with each other. We also address the case where up to t of the servers are faulty and *do* communicate with each other.

Notions of Privacy: We define the following privacy properties for an information retrieval scheme.
User privacy [10]: No single database (or server) can get any information about the user's query i from the on-line stage. That is, all the communication seen by a single database is identically distributed for every possible query. This definition can be extended to *user l-privacy*, where all communication seen by any coalition of up to l databases (servers) is identically distributed for every possible query.
Database privacy [11]: The user cannot get any information about the data string other than its value in a single location. That is, all the communication seen by the user in the on-line stage is dependent on a single physical bit x_i (so it is identically distributed for any string x' s.t. $x_i = x'_i$).
Independence: No auxiliary server has any information about the data string x. That is, the content of the auxiliary server is identically distributed for any data string x. This definition can be extended to *t-independence*, where no coalition of up to t servers has any information about x (thus, independence is the special case of 1-independence).
Total independence: All the auxiliary servers jointly have no information about the data string x, or equivalently all the servers are universal. That is, they are completely independent of the original data, and thus may all be chosen in advance.

In all the above definitions, information theoretic privacy may be relaxed to *computational* privacy, requiring indistinguishablity instead of identical distribution.

Protocols: A *private information retrieval (PIR)* scheme is one that achieves user privacy, and a *symmetrically private information retrieval (SPIR)* scheme is one that achieves user privacy and database privacy. These two protocols were defined in [10, 11], respectively. In this paper we will show how to incorporate the independence or total independence properties into PIR and SPIR schemes.

Complexity: We define the *communication complexity* of an information retrieval protocol to be the total number of bits sent between the user, the database, and the servers. The *computation complexity* (of a user/database/server during setup/on-line stage) is the amount of computation that needs to be performed by the appropriate party before the required communication can be sent. Note that we count sending bits from a specific location towards communication complexity, rather than computation complexity. Communication and computation complexity during the on-line stage refer to the complexity per each (single) query.

3 Achieving Independence: The RDB Scheme

In this section we describe a simple and efficient scheme, which takes advantage of the random server model to achieve t-independence and no database participation in the on-line stage. Specifically, we prove the following theorem.

Theorem 1. *Given any information retrieval scheme S which requires k copies of the database and communication complexity C_S, and for every $t \geq 1$, there exists an information retrieval scheme achieving t-independence and maintaining the other privacy properties (user privacy and data privacy) of S. The t-independent scheme requires $(t+1)C_S$ communication complexity and $(t+1)k$ servers, out of which only k are tailored. The setup complexity is $O(n)$ and the database is not required to participate in the on-line stage.*

An immediate useful corollary follows, setting $t = 1$:

Corollary 1. *Given any information retrieval scheme S which requires k copies of the database, there exists an information retrieval scheme achieving independence and maintaining the other privacy properties of S, which requires a factor of 2 in communication complexity, and uses k tailored servers and k universal ones. The setup complexity is $O(n)$ and the database is not required to participate in the on-line stage.*

The basic version of our reduction (the RDB scheme) is described in section 3.1. In section 3.2 we present another version, possessing some appealing extra properties for security and simplicity. We note however, that the starting point for the second reduction is any information retrieval scheme which has a linear reconstruction function. This is usually the case in existing PIR schemes (cf. [10, 3]). Finally, in section 3.3 we prove that the RDB construction satisfies theorem 1.

Another benefit of our scheme is that it does not require the participation of the database owner D after the setup stage. Instead, the servers deal with all the on-line queries and computations. Even though D is not there for the on-line stage, he is guaranteed that none of the servers who are talking to users on his behalf has any information about his data x.

3.1 The Basic RDB Scheme

In the basic RDB (random data base) scheme, instead of replicating the original database as in the underlying scheme, every copy is replaced by $t+1$ random servers whose contents xor to the contents of the database. The idea behind this replacement is that if these $t+1$ databases are chosen uniformly at random with this property, then any coalition of t of them are simply a random string, independent of the actual original data string. Therefore, t-independence is achieved . We proceed with the details of the basic reduction. The communication complexity and privacy properties of this scheme will be proved in section 3.3.

Let the underlying scheme S be a PIR scheme with k copies of the database.

Setup Stage The database owner D chooses uniformly at random $t+1$ random servers $R_1, \ldots, R_{t+1}$ in $\{0,1\}^n$, such that for every $1 \leq j \leq n$, $R_1(j) \oplus \ldots \oplus R_{t+1}(j) = D(j) = x_j$ i.e., the xor of all the servers is the original data string x. This is done by choosing k universal servers, and computing the content of another tailored server in an appropriate way. A protocol to do that is described in the appendix.

Each of these servers is then replicated k times, for a total of $k(t+1)$ servers.

Thus, at the end of the setup stage, the random servers are

$$R_1^1, \ldots, R_1^k, \; \ldots \; , R_{t+1}^1, \ldots, R_{t+1}^k$$

where $R_s^1 = R_s^2 = \ldots = R_s^k$ for every s, and where $R_1^r \oplus R_2^r \oplus \ldots \oplus R_{t+1}^r = x$ for every r.

On-Line Stage During the on-line stage, the user executes the underlying scheme S $t+1$ times, each time with a different set of k databases. The first execution is with the k copies of R_1, which results in the user obtaining $R_1(i)$. The second execution is with the k copies of R_2, resulting in the retrieval of $R_2(i)$, and so on. Finally, the user xors all the $t+1$ values he retrieved, $R_1(i) \oplus \ldots \oplus R_{t+1}(i) = D(i) = x_i$ in order to obtain his desired value x_i.

Note that the user can perform all these $t+1$ executions of S in parallel. Also, the user may either perform all these parallel executions independently, or simply use exactly the same questions in all of them. Our proofs will cover both these variants, but we prefer the latter since it simplifies the protocol of user-privacy against coalitions. However, in the most general case, if S is a multi round scheme with adaptive questions, we must use the first strategy of independent executions.

Remarks Note that out of the $k(t+1)$ servers, all but k are universal servers which can be prepared ahead of time, whereas the other k (copies of R_{t+1}) are tailored.

Another thing to note is the fact that our scheme uses replication of the random servers. At first glance, this may seem to contradict our goal of solving the data replication problem. However, in contrast to replicating the original database, replicating random servers does not pose any threat to the original data string which we are trying to protect. Thus, we manage to separate the user privacy, which requires replication, from the database privacy, which requires not to replicate the data. Still, in the next section we describe a version in which there is no replication, not even of the auxiliary servers, and which subsequently provides a higher level of privacy, as discussed below.

3.2 The RDB Scheme : Improved Variant

While the basic scheme does achieve t-independence (as no coalition of t servers has any information about x), some of the servers there are replications of each other.

Here, we propose an improvement to the basic scheme, in which a higher level of independence among the random servers is achieved, allowing for more flexibility in choosing the random servers from different providers. Specifically, we achieve t-independence among the servers, namely every combination of t servers are independent of each other (in particular, there is no replication of the servers).[7] Another benefit of this scheme over the basic one is that, while t is still the maximal size of coalition that the database is secure against, it is also secure against many other specific combinations of larger coalitions. This protocol works provided that the underlying PIR scheme has a linear reconstruction function (see 3.3), a quite general requirement that is satisfied by currently known PIR schemes.

Setup Stage Recall that in the basic version, we created $t+1$ servers and replicated each of them k times, thereby creating $t+1$ sets, each of which consist of k identical servers. In this protocol, the k servers in every set will be independent random strings, instead of replications. Specifically, the database owner D chooses *uniformly at random* $k(t+1)$ servers $R_1^1, \ldots, R_{t+1}^1, \ldots, R_1^k, \ldots, R_{t+1}^k$ with the property that $R_1^r \oplus \ldots \oplus R_{t+1}^r = x$ for every $1 \leq r \leq k$.

As in the basic scheme, kt of these servers are universal, and k are tailored. The contents of the tailored servers is computed by D using the same protocol as in the basic scheme (see appendix).

On-Line Stage During the on-line stage, the user sends his queries (according to the underlying S) to each of the servers, where $\{R_1^r, \ldots, R_{t+1}^r\}$ correspond to the r-th copy of the database in the underlying scheme S. After receiving the answers from all the $k(t+1)$ servers, the user xors the answers of $R_1^r, \ldots, R_{t+1}^r$ for each r to obtain the answer of the r-th copy in S, and combines these answers as in S to obtain his final value x_i.

The difference between this version and the basic version, is the following. In the basic scheme, the user first runs S to completion with each of the $t+1$ sets of servers (for example one set is $R_1^1, \ldots R_1^k$) giving the user $t+1$ values that enable him to xor them all together and obtain the value of the primary database. In contrast, here the user first combines answers by xoring values he received (for example from $R_1^1, \ldots R_{t+1}^1$) in the middle of the S protocol, which gives the user the intended answer of each copy of the database, and only then combines the answers as needed in S.

Thus, to succeed in this version, the underlying S must have the following closeness property under xor: *If $f_r(x,q)$ is the function used by the r-th copy of the database in S to answer the user's query q with the data string x, and given $y_1, \ldots, y_m$, then* $f_r(y_1,q) \oplus \ldots \oplus f_r(y_m,q) = f_r(y_1 \oplus \ldots \oplus y_m, q)$. This may be generalized to any underlying scheme with a linear reconstruction function. This requirement is very general,

[7] Moreover, if we assume that the original data x is randomly distributed, then the servers are $2t+1$ independent.

and is usually satisfied by existing PIR protocols (for example, protocols based on xoring subsets of locations in the data string, such as [10, 3], the best PIR schemes known to date).

3.3 Analysis of the RDB Scheme: Proof of Theorem 1

We now analyze the RDB scheme in terms of complexity, correctness, and privacy, to show that it satisfies the bounds given in theorem 1.

The RDB scheme requires a multiplicative factor of $(t+1)$ in communication complexity over the underlying scheme S, since S is simply executed $t+1$ times. Typically, t is a constant $t \geq 1$, which means the communication complexity of RDB is $O(C_S)$, where C_S is the communication complexity of S. The number of tailored servers required is the same as the number of databases required in S, since all the tuples $R_1, \ldots, R_t$ can be prepared in advance, and then they can be xored with the original data to produce R_{t+1}. Thus, one tailored server is needed per one copy of the database in the underlying S.

It is not hard to check that the scheme gives the user the correct value x_i, because of the way the servers were chosen, and from the correctness of S.

User privacy properties carry from S, namely if S was user-l-private (i.e. user private against coalitions of up to l databases), then so is the corresponding RDB scheme (where user privacy is protected from any coalition of l servers). This is clear for coalitions involving servers from the same set $R_s^1, \ldots, R_s^k$ for some s, since the user simply runs S with the set. This argument immediately extends to arbitrary coalitions if the user sends exactly the same questions in all sets (i.e. in every execution of S).[8] In the case of parallel independent executions and a multi round adaptive S, a little more care is needed to show that the view of any coalition is independent of i, using the l-user-privacy of S inside sets, and the independence of the executions across sets.

Database privacy of S also implies database privacy of the corresponding RDB scheme, as follows. If S is database private (SPIR), then in the r-th parallel execution of S the user gets at most one bit, and altogether the user gets at most $(t+1)$ bits. Since these are chosen uniformly at random among all strings that xor to x, it follows that if the $(t+1)$ bits are from the same location i in all servers, they are distributed uniformly over all $(t+1)$-tuples that xor to x_i, and otherwise the $(t+1)$ bits are distributed randomly among all possible tuples. In any case, the user's view depends on at most one physical bit of x, and database privacy is maintained.

Finally, the RDB scheme achieves t-independence since any coalition of up to t servers contains only t or less of the servers in $R_1^r, \ldots, R_{t+1}^r$, and thus (from the way the auxiliary databases were defined), the coalition consists of a string uniformly distributed over all strings of appropriate length, independent of x.

4 Achieving Total Independence: The Oblivious Data Scheme

In this section we present a scheme for total independence PIR (or total independence SPIR), where *all* auxiliary servers are universal, i.e. jointly independent of the database. This scheme also achieves O(1) computation complexity for the database.

[8] This strategy is always possible unless S is a multi round adaptive scheme.

Overview of Results We first describe a basic version of our scheme, which achieves total independence, as well as database privacy, but maintains the user privacy with one exception: in repeated executions of the basic scheme, the database can tell whether the questions in different executions correspond to the same index or not. We prove that no other information about the content of the queries or the relations among them is revealed to the database. We call this *user privacy up to equality between repeated queries*. Thus, we prove the following theorem.

Theorem 2. *Given any PIR scheme S which requires k copies of the database and communication complexity C_S, there exists a total independence SPIR scheme, private for the database and private for the user up to equality between repeated queries, which uses $\max(k, 2)$ universal servers, and requires communication complexity of at most $O(C_S \log n)$. The setup complexity is $O(n \log n)$, and the on-line computation complexity of the database is $O(1)$.*

The scheme is described in section 4.1, and in section 4.2 we prove that it satisfies the theorem.

Since the information of whether users are asking the same question or not may in some applications be an important information that violates the user privacy, we present a generalized version of our scheme in section 4.3, which completely hides all information about the user queries, even after multiple executions. This scheme maintains the privacy properties of the underlying scheme, namely it transforms a PIR scheme into a total independence PIR scheme, and a SPIR scheme into a total independence SPIR scheme. The price we pay for eliminating the equality leakage, is that the setup stage needs to be repeated every m queries, and an additive factor of $m \log n$ is added to the communication complexity, where m is a parameter to the scheme (see 4.3 for how to choose m). Thus, we prove the following theorem.

Theorem 3. *Given any information retrieval scheme S which requires k copies of the database and communication complexity C_S, there exists a total independence information retrieval scheme, maintaining the privacy properties (user privacy and database privacy) of S, which uses $\max(k, 2)$ universal servers, and requires communication complexity of at most $O((m + C_S) \log n)$, where m is the number of queries allowed before the system needs to be reinitialized. The setup complexity is $O(n \log n)$, and the on-line computation complexity of the database is $O(1)$.*

The following corollary is obtained by setting $m = n^\epsilon$ in the above theorem, where n^ϵ is some polynomial equal to the communication complexity of the underlying PIR scheme (it is conjectured in [10] that all information theoretic PIR schemes must have communication complexity of at least $\Omega(n^\epsilon)$ for some ϵ).

Corollary 2. *Given any information retrieval scheme S which requires k copies of the database and has communication complexity $O(n^\epsilon)$, there exists a total independence information retrieval scheme, maintaining the privacy properties of S, which uses $\max(k, 2)$ universal servers, requires communication complexity of $O(n^\epsilon \log n)$, and has to be reinitialized after every $O(n^\epsilon)$ number of queries.*

It is not clear which of the two schemes – the one achieving privacy up to equality (plus database privacy), or the one achieving full privacy but with periodic setups – is better. This depends on the particular needs of the application.

The Main Idea: Oblivious Data Recall that in order to achieve information theoretic PIR a number of multiple servers is required. On the other hand in order to achieve total independence PIR, all auxiliary servers must be (jointly) independent of the data. To accommodate these two seemingly conflicting requirements we use the following idea. During the setup stage, the database and the auxiliary servers create a new "oblivious" string y which depends on the content of all of them. This string must be held by the database D (since all others cannot hold any data dependent on x). Thus, we let the database change during the setup stage, rather than the servers. Later, during the on-line stage, the user interacts with the servers to obtain information about the relation between y and x. Knowing this information the user can simply ask D for the value of y in an appropriate location, whose relation to x he knows from communication with the servers, which enables him to compute x_i. We call y an *oblivious* data string, since it should be related to the data string x, yet in a way which is oblivious to its holder D, so that D cannot relate the user's query in y to any query in x, and therefore learns nothing about the user's interests from the user's query in y. Note that all the database's work is in the setup stage (which amounts to only a logarithmic factor over the work that needs to be done to replicate itself in the old model). During the on-line stage, however, all D needs to do is to reply with a bit from the oblivious string which requires no computation.

4.1 Basic Scheme

Let the underlying scheme S be a PIR scheme with k copies of the database.

Setup Stage The (universal) auxiliary servers are k servers each containing a random string $r \in \{0,1\}^n$, and a random permutation $\pi : [1..n] \to [1..n]$ (represented by $n \log n$ bits in the natural way). D and two of the servers R_1, R_2 engage in a specific multi party computation, described below, at the end of which D obtains the oblivious data string

$$y = \pi(x \oplus r)$$

but no other information about r, π. Each server does not obtain any new information about x.

Naturally, by the general multi-party theorems of [6,8], such setup stage protocol exist, but are very expensive. Instead, we design a special purpose one-round efficient protocol for this purpose.

The multi party computation is done as follows: D chooses uniformly at random two strings x^1 and x^2 such that $x^1 \oplus x^2 = x$. Similarly, R_1 chooses uniformly at random r^1, r^2 such that $r^1 \oplus r^2 = r$. R_2 chooses uniformly at random π^1, π^2 such that $\pi^1 \circ \pi^2 = \pi$, where $\circ$ is the composition operator (that is, $\pi^2(\pi^1(\cdot)) = \pi(\cdot)$). The following information is then sent between the parties on secure channels:

$$\begin{array}{l|l|l} R_2 \to R_1 : \pi^1 & R_1 \to R_2 : r^2 & R_1 \to D : v = \pi^1(r^1 \oplus x^1) \\ D \to R_1 : \ x^1 & D \to R_2 : \ x^2 & R_2 \to D : pi^2, u = \pi(r^2 \oplus x^2) \end{array}$$

D can now compute $y = \pi^2(v) \oplus u = \pi(r^1 \oplus x^1) \oplus \pi(r^2 \oplus x^2) = \pi(r \oplus x)$ ("the oblivious string"). R_1 and R_2 discard all communication sent to them during the setup stage, and need to maintain only their original independent content.

At the end of the setup stage the database D has two strings: x which is the original data string, and also y which is the oblivious data. The auxiliary servers contain the same strings as before the setup stage, and do not change or add to their content.

On-Line Stage In the on-line stage the user first runs S (the underlying PIR scheme) with the servers to retrieve the block $(j := \pi(i)\,,\, r_i)$, as specified below (recall that r_i is the random bit with which the user's desired data bit was masked, and that j is the location of the masked bit in the oblivious data string). Then the user queries D for the value at the j-th location y_j. This is done by simply sending j to D on the clear, and receiving the corresponding bit y_j back. To reconstruct his desired bit, the user computes $y_j \oplus r_i = [\pi(x \oplus r)]_j \oplus r_i = (x \oplus r)_i \oplus r_i = x_i$.

Since S is a PIR scheme for retrieving a single bit, we need to specify how to retrieve the required block. The most naive way is to apply S $\log n+1$ times, each time retrieving a single bit out of the $n(\log n + 1)$ bits. However this way does not necessarily maintain database privacy, and is not the best in terms of communication complexity. This can be improved by noticing that each of the $\log n + 1$ bits required belongs to a different set of n bits. Thus, the online stage can be performed by $\log n + 1$ parallel applications of S for one bit out of n. Further improvements are possible when methods for block retrieval which are more efficient than bit by bit are available (cf. [10]).

Note that the computation complexity for the database here is minimal – $O(1)$. In fact, the only thing required from D is to send to the user a single bit from the specified location.

Remarks Two questions may arise from our setup stage. First, can the setup stage be achieved using only a single server and the database? This would change the required number of servers in the scheme to k instead of $\max(k, 2)$. Second, and more important, note that during our setup stage if R_1 and R_2 collude, together they can find out the data. So, to guarantee total independence, they should discard the communication sent to them during the setup stage.[9] Can we construct a different protocol for the setup stage which avoids this problem? The following lemma helps us answer the above questions, by showing that it is impossible to achieve our setup stage with only two parties.

Before stating the lemma, let us informally describe what it means for a 2-argument function to be privately computable, in the information theoretic model, and with honest players (for formal definitions and treatment see [17, 6, 8, 13]). A function $f([x_1], [x_2]) = ([y_1], [y_2])$ is *privately computable* if there exist a protocol for two players P_1, P_2, as follows. At the beginning of the protocol P_1 holds x_1 and P_2 holds x_2. during the protocol the players may flip coins and alternately send messages to each other. At the end of the protocol, each player P_i $(i = 1, 2)$ can use the communication and his input x_i to reconstruct his output y_i (*correctness*), but cannot obtain any other information about the other party's input, that does not follow from his own input and output (*privacy*).

Lemma 1. *The two-argument function $f([\pi, r], [x]) = ([\emptyset], [\pi(x \oplus r)])$ is* not *privately computable (in the information theoretic model).*

Proof. See appendix. □

From the lemma, it is clear that our setup stage cannot be achieved with a single server and the database, since a multi-party computation is needed (rather than a two-party computation). This is not a real problem though, since if we want information theoretic privacy, we must have $k \geq 2$ in the PIR scheme to begin with, and thus $\max(k, 2) =$

[9] Note that if the servers do not discard the communication, the privacy and independence are not compromised, but the total independence is replaced by simple independence (and may be extended to t-independence).

k is optimal number of servers. If however we are willing to settle for computational privacy, then we can achieve the setup stage with a single server.

As for the second question, note that the lemma implies that there cannot exist a protocol (even with arbitrary number of servers), such that the database obtains $\pi(r \oplus x)$ and the servers *jointly* obtain no information about x. This is because if we consider the information obtained by a coalition of all servers, this is reduced to a two party protocol. Thus, our setup stage cannot be improved in this sense, with respect to our function $\pi(r \oplus x)$.

4.2 Analysis of the Basic Oblivious Scheme: Proof of Theorem 2

We now analyze the oblivious scheme in terms of complexity, privacy, and correctness, to show that it satisfies the bounds given in theorem 2.

It is not hard to verify that the setup stage computation is correct, namely that indeed $y = \pi(x \oplus r)$. Now the **correctness** of the scheme follows from the correctness of the underlying S: since the user uses S to obtain r_i and $j = \pi(i)$, it follows that $y_j = x_i \oplus r_i$ and thus $x_i = y_j \oplus r_i$.

The **communication complexity** of the scheme is at most $(\log n + 1)C_S(1, n) + \log n + 1$, where $C_S(l, n)$ is the communication complexity that the underlying scheme S requires to retrieve a block of l bits out of n bits. This expression is based on a bit by bit retrieval, as discussed above. Alternatively, any other method for retrieving blocks can be used, yielding communication complexity $C_S(\log n+1, n(\log n+1)) + \log n + 1$, which may be lower than the general expression. The **computation complexity** of the database is $O(1)$ during the on-line stage because it needs to send only one bit of information to the user. During the **setup stage** the computation of the database involves linear computation complexity which is similar to the amount of work it needs to do in order to replicate itself in the original PIR model. The communication complexity of the setup stage is $O(n \log n)$, which is a factor of $\log n$ over the $O(n)$ of existing PIR algorithms, where the database has to be replicated.

Total independence is clearly achieved, since the auxiliary servers may all be predetermined in advance, and do not change their content after setup stage.

Database Privacy is also guaranteed by our scheme, even if the underlying S is not database private. This is because, no matter what information the user obtains about π and r, this information is completely independent of the data x. The user gets only a single bit of information which is related to x, and this is the bit y_j at a certain location j of the user's choice. Note that since $y = \pi(x \oplus r)$, the bit y_j depends only on a single physical bit of x.

User Privacy with respect to the servers follows directly from the user privacy of the underlying scheme, and user privacy with respect to D is maintained in a single execution of the scheme, and in multiple executions up to equality between queries, as we prove below. However, if in multiple executions two users are interested in the same query i, the database will receive the same query $j = \pi(i)$, and will thus know that the two queries are the same. This will be dealt with in section 4.3. We proceed in proving user privacy up to equality with respect to the database D.

Consider an arbitrary sequence $(i_1, \ldots, i_m)$ of query indices which are all distinct. We will prove that the distribution of D's view after the setup stage and m execution of the on-line stage with these indices is independent of the values $i_1, \ldots, i_m$.

Lemma 2. *Let V_{setup} be the view of D after performing the setup stage. For every permutation $\hat{\pi} : [1..n] \rightarrow [1..n]$, $Prob[\pi = \hat{\pi} \mid V_{\text{setup}}] = \frac{1}{n!}$ where probability is taken over all the random setup choices π, π^1, r, r^1. In particular, D does not get any information about the permutation π from the setup stage.*

Proof. D's view consists of $V_{\text{setup}} = [x^1, x^2, \pi^2, v = \pi^1(r^1 \oplus x^1), u = \pi(r^2 \oplus x^2)]$. Given this view, every choice for a permutation π fixes the choices of π^1, r, r^1. That is, every $\hat{\pi}$ corresponds to a single choice $(\hat{\pi}, \hat{\pi}^1, \hat{r}, \hat{r}^1)$ which generates the given view. Since all these random choices of the setup stage are done uniformly and independently of each other, each such choice is equally likely. Thus, the probability of a particular $\hat{\pi}$ is $\frac{1}{n!}$. □

Lemma 3. (User Privacy) *Let $(i_1, \ldots, i_m)$ be a tuple of distinct indices in $[1..n]$. Let V_{setup} be the view of D after the setup stage, and $V(i_1, \ldots, i_m)$ be the view of D for m executions of the on-line stage with queries $i_1, \ldots, i_m$. Then for every tuple $(j_1, \ldots, j_m)$ of distinct indices, and for every setup view V_{setup},*

$$Prob[V(i_1, \ldots, i_m) = (j_1, \ldots, j_m) \mid V_{\text{setup}}] = \frac{(n-m)!}{n!}$$

where probability is taken over all the random choices π, π^1, r, r^1. In particular, the view is independent of the user queries.

Proof. Since after the setup stage D did not get any information about π, as proved in proposition 2, every π is equally likely, and thus the given tuple $(j_1, \ldots, j_m)$ may correspond to any original queries tuple $(i_1, \ldots, i_m)$ with equal probability. A formal derivation follows. Denote by $\Pi = \{\pi \mid \pi(i_1, \ldots, i_m) = (j_1, \ldots, j_m)\}$.
$Prob[V(i_1, \ldots, i_m) = (j_1, \ldots, j_m) \mid V_{\text{setup}}] = \sum_{\pi} Prob[\pi \mid V_{\text{setup}}] Prob[V(i_1, \ldots, i_m) = (j_1, \ldots, j_m) \mid V_{\text{setup}}, \pi] = \sum_{\pi \in \Pi} Prob[\pi \mid V_{\text{setup}}] = \sum_{\pi \in \Pi} \frac{1}{n!} = \frac{(n-m)!}{n!}$ □

We proved that any two tuples of distinct queries $(i_1, \ldots, i_m)$ and $(i'_1, \ldots, i'_m)$ induce the same distribution over the communication $(j_1, \ldots, j_m)$ sent to D. Therefore, the basic scheme is user private up to equality.

4.3 Eliminating Detection of Repeated Queries: Proof of Theorem 3

In order for the oblivious database scheme to be complete we need to generalize the basic scheme so that it guarantees user privacy completely and not only up to equality between repeated executions. To extend it to full privacy we need to ensure that no two executions will ever ask for the same location j. To achieve this, we use a buffer of some size m, in which all (question,answer) pairs (j, y_j) that have been queried are recorded. The on-line stage is changed as follows: the user who is interested in index $\hat{i}$ first obtains the corresponding $r_{\hat{i}}, \hat{j}$ from the servers similarly to the basic version. He does that by running S to obtain the bit $r_{\hat{i}}$, and (in parallel) using the most efficient way available to obtain the block $\hat{j}$ (again, a possible way to do it is by running $S(1, n) \log n$ times).[10] Then the user scans the buffer. If the pair $(\hat{j}, y_{\hat{j}})$ is not there, the user asks D for $y_{\hat{j}}$ (as in the basic scheme). If the desired pair is there, the user asks D for y_j in some random location j not appearing in the buffer so far. In any case, the pair (j, y_j) which was asked from D is added to the buffer.

Clearly, a buffer size m results in an additive factor of $m \log n$ in the communication complexity over the basic scheme. On the other hand, after m executions the buffer is full, and the system has to be reinitialized (i.e. the setup stage is repeated, with new r, π). Thus, we want to choose m as big as possible without increasing the communication complexity much. A suitable choice for existing schemes will therefore be $m = n^{\epsilon}$ the

[10] so far we are doing the same as in the basic scheme, except we insist that r_i is retrieved separately from $\hat{j}$. This is done in order to maintain database privacy in case the underlying S is database private, as proved below, and it does not change the communication complexity.

same as the communication complexity of the underlying S. This only increases communication complexity by a constant factor, and still allows for polynomial number of executions before reinitialization is needed. We note that in many practical situations, reinitialization after m steps is likely to be needed anyway, as the database itself changes and needs to be updated.

The database privacy in this case depends on the underlying scheme S: If S is database private (a SPIR scheme), then so is our scheme. This is because, when running S, the user gets only a single physical bit r_i out of r. Now, no matter how much information the user obtains about π or y (either from direct queries or from scanning the buffer), the data x is masked by r (namely $y = \pi(x \oplus r)$), and thus the user may only obtain information depending on a single physical bit of x.

The other privacy and correctness properties can be verified similarly to the basic scheme proofs of section 4.2.

References

1. M. Abadi, J. Feigenbaum, J. Kilian. On Hiding Information from an Oracle. JCSS, 39:1, 1989.
2. N. Adam, J. Wortmann. Security Control Methods for Statistical Databases: a comparative study, ACM Computing Surveys, 21(1989).
3. A. Ambainis. Upper bound on the communication complexity of private information retrieval. ICALP 97.
4. D. Beaver, J. Feigenbaum. Hiding instances in Multioracle Queries. STACS,1990.
5. D. Beaver, J. Feigenbaum, J. Kilian, P. Rogaway. Security with Low Communication Overhead. CRYPTO 1990.
6. M. Ben-Or, S. Goldwasser, A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. STOC 1988.
7. D. Beaver. Commodity Based Cryptography. STOC 1997.
8. D. Chaum, C. Crepeau, I. Damgaard. Multiparty Unconditionally Secure Protocols. STOC 1988.
9. B. Chor, N. Gilboa. Computationally Private Information Retrieval. STOC 1997.
10. B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan. Private Information Retrieval. FOCS 1995.
11. Y. Gertner, Y. Ishai, E. Kushilevitz, T. Malkin. Protecting Data Privacy in Private Information Retrieval Schemes. STOC 1998.
12. O. Goldreich, S. Micali, A. Wigderson. How to Solve any Protocol Problem. STOC 1987.
13. E. Kushilevitz. Privacy and Communication Complexity. FOCS 1989.
14. E. Kushilevitz, R. Ostrovsky. Replication is Not Needed: Single Database, Computationally-Private Information Retrieval. FOCS 1997
15. R. Ostrovksy, V. Shoup. Private Information Storage. STOC 1997.
16. P. Tendick, and N. Natloff. A modified Random Perturbation Method for Database Security. ACM Transactions on Database Systems, 19:1, pp.47-63, 1994.
17. A. C. Yao. Protocols for Secure Computations. FOCS 1982.

A Setup Stage of RDB

For the basic RDB scheme, the database owner D chooses uniformly at random $t + 1$ random servers $R_1, \ldots, R_{t+1}$ in $\{0, 1\}^n$, such that for every $1 \le j \le n$, $R_1(j) \oplus \ldots \oplus R_{t+1}(j) = D(j) = x_j$ i.e., the xor of all the servers is the original data string x. This is done by first choosing $R_1, \ldots, R_t$ containing completely random strings (universal servers), and then using the following protocol which allows D to prepare an appropriate content for the tailored server R_{t+1}. Since we do not allow the servers to gain any information about each other, the result of this computation should only go to the database. One possible way would be to let the database read the content of all universal servers, but this would give the database much more information than it needs, which may be a source for future security problems.[11] Thus, we use a simple multi-party protocol for computing the xor, at the end of which D learns R_{t+1} but no other information, and the servers do not learn any new information.

Computing $R_{t+1} = R_1 \oplus \ldots \oplus R_t \oplus x$ **:** Each of the servers R_s ($1 \leq s \leq t$) first shares its content among all others and D, by choosing uniformly at random $t+1$ shares $a_{s1}, \ldots, a_{st}, a'_s$ that xor to R_s. Each a_{sj} is sent to R_j, and a'_s is sent to D. Next, every server xors all shares sent to it from all other servers, and sends the result to D, who now xors all the messages and x, to obtain the desired content for R_{t+1}.

For the RDB scheme of section 3.2, the same summing protocol is performed k times (computing R^r_{t+1} for every $1 \leq r \leq k$).

B Proof of Lemma 1

Assume towards contradiction that there is a protocol that privately computes the function $f([\pi, r], [x]) = ([\emptyset], [\pi(x \oplus r)])$. That is, before the protocol starts, player P_1 holds (π, r) and player P_2 holds x. During the protocol P_1, P_2 may flip sequences of random coins, denoted c_1, c_2 respectively. Denote the communication exchanged between the players by $\mathtt{comm} = \mathtt{comm}(\pi, r, c_1, x, c_2)$. At the end of the protocol, P_1 gets no information about x; P_2 may apply a reconstruction function $g(\mathtt{comm}, x, c_2) = \pi(r \oplus x)$ to obtain his output; and no other information about (π, r) is revealed to P_2. Now, when r is chosen uniformly $(x, \pi(r \oplus x))$ gives no information about π, and thus P_2 obtains no information about π. In particular, this means that given P_2's view ($x, c_2, \mathtt{comm}$), any permutation could have generated the given communication $\mathtt{comm}$, or more formally,

$$\forall x, c_2, \forall c_1, r, \pi, \pi', \exists r', c'_1 \quad \mathtt{comm}(\pi, r, c_1, x, c_2) = \mathtt{comm}(\pi', r', c'_1, x, c_2) \tag{1}$$

We will show that this implies that P_1 can obtain information about x from the communication, which contradicts the privacy of the protocol. Let P_2 conduct the following mental experiment on his view $(\pi, r, c_1, \mathtt{comm})$. P_2 sets π' to be the identity permutation, and finds r', c'_1 that yield the same communication comm (such r', c'_1 exist by (1)). Now, since $\pi(r \oplus x) = g(\mathtt{comm}(\pi, r, c_1, x, c_2), x, c_2) = g(\mathtt{comm}(\pi', r', c'_1, x, c_2), x, c_2) = \pi'(r' \oplus x) = (r' \oplus x)$, P_2 can find out that x satisfies the equation

$$x = r' \oplus \pi(r \oplus x). \tag{2}$$

Since(2) cannot hold for all x (unless $n = 1$), we have shown that P_2 obtains some information about x, which concludes the proof. □

[11] e.g. in a setting where the same universal random servers may be used by multiple applications.