Private Information Retrieval: A Primer

Amos Beimel
Dept. of Computer Science, Ben-Gurion University,
Beer-Sheva 84105, Israel.

January 14, 2008

Abstract

Private Information Retrieval (PIR) protocols allow a client to retrieve a data item from a database while hiding the identity of the item being retrieved. In *information-theoretic k-server* PIR protocols the database is replicated among k servers, and each server learns absolutely nothing about the item the client retrieves. In *computational* PIR protocols the database is stored on one server, and a computationally-bounded server learns nothing about the item the client retrieves. The important challenge in this area is to design protocols in which the *communication complexity* of retrieving one out of n bits of data is sub-linear. This primer discusses the PIR problem, surveys the known results on PIR, describes the basic construction of information-theoretic and computational PIR protocols, and presents applications of PIR protocols. The primer is intended for readers that do not necessary have background in cryptography. In particular, we supply the background from cryptography, combinatorics, and number theory that is needed to understand the results.

1 Introduction

The complex pattern of activities of a user in the Internet can disclose a lot of information on the user. Thus, it is crucial to supply tools to protect the privacy of the user. A Private Information Retrieval (PIR) is an important protocol towards this goal. It enables a client to retrieve a data item of its choice from a database while preventing the server storing the database from gaining information about the identity of the chosen item. This problem was introduced by Chor, Goldreich, Kushilevitz, and Sudan [20] and has since attracted a considerable amount of attention. Before we continue, we mention a few scenarios which demonstrate why we would want to apply a PIR protocol. First, consider a rich investor that is interested in the price of a stock in the stock market, however the investor does not want to reveal which stock is of interest since this information can affect the price of the stock. Second, consider a company that wants to know what patents are related to a specific subject without disclosing this subject to its competitors. The last scenario, which is currently far from being realized, is to allow clients to access the web without exposing the cites they visit.

In formalizing the PIR problem, it is convenient to model the database by an n-bit string x where the client, holding some *retrieval index* i, wishes to learn the ith data bit x_i. A trivial solution to the PIR problem is to send the entire database x to the client. However, while being perfectly private, the *communication* in this solution for large databases is unreasonable. Indeed, the most significant goal in designing PIR protocols is to minimize their communication complexity. Unfortunately, if the server is not allowed to gain *any* information about the identity of the retrieved bit, then the linear communication complexity of the trivial solution is optimal [20]. There are two ways to get around this problem. The first way, suggested by Chor et al. [20], is that the client access k replicated copies of the database stored at different servers, requiring that each individual server gets absolutely no information about i. PIR in this setting is called *information-theoretic* PIR. The second way is to relax the privacy requirement and consider computational privacy, that is, it is assumed that the running time of the server is polynomial and it is required that such server cannot learn information. PIR in this setting is called *computational* PIR.

We start by mentioning a few, somewhat simplistic, solutions and explaining why they do not work.

- The client can try to "hide" the index it wants in a bigger set of indices, that is, the client picks a moderately small set of indices containing the index it wants, sends this set to the server, and the server sends the contents of the database in the locations in the set. The problem with this solution is that it leaks a lot of information. For example, the server knows that the client is *not* interested in locations outside the set. Another example is when the server has some information on the index the client wants (e.g., the server knows that the index that the client wants is either 1 or 2; in this case when 1 is not in the set sent by the client, the server knows that the client's index is 2). Such information can be learned, for example, from related queries of the client to the server (i.e., learning the profile of the client).

- The client can use an anonymous communication protocol, that is, the client can try to hide its identity from the server. However, this does not solve the PIR problem, as the information that someone is interested in a specific index can be valuable (e.g., this is the location of a "treasure", we want to know this location and we do not care who found it).

- Cryptography offers general protocols for computing functions privately (e.g., [57, 32]). These protocols solve the PIR problem, however the communication in these protocols is at least linear in the size of the database, thus defeating the PIR goal of designing communication efficient protocols.

To conclude, solving the PIR problem is non-trivial, and requires designing new protocols.

1.1 The Models of Private Information Retrieval

Information-Theoretic PIR. Information-theoretic PIR protocols guarantee perfect privacy – even an unbounded server learns no information. However, they require replication of the data among several non-communicating servers. Information-theoretic PIR protocols were introduced and constructed by Chor et al. [20]. In particular, they construct the best known 2-server protocol with communication complexity $O(n^{1/3})$ (where n is the database length). More efficient constructions of k-server protocols for $k > 2$ appear in [2, 38, 6, 56, 8, 59, 40]. The best known 3-server PIR protocol is constructed in a lovely work of Yekhanin [59]; assuming that there are infinitely many Mersenne primes, he constructs a 3-server PIR protocol with communication complexity $n^{O(1/\log\log n)}$ for infinitely many values of n. Specifically, his protocol implies, without any assumptions, a 3-server PIR protocol with communication complexity n^{ϵ} for some $\epsilon < 10^{-7}$. To date, Yekhanin's protocol is the best k-server PIR protocol for every constant k.

One application of information-theoretic PIR is for constructing *locally decodable* error-correcting codes. A locally decodable code encodes a database x into a longer string y, such that, even if a large portion of y is corrupted, each bit of x can still be decoded *with high probability* by probing few bits of y. For example, one would want to encode an entire library such that each page can be read by accessing few bits of the encoding. Note that encoding each page independently does one achieve the goal since even if a small portion of the encoded message is destroyed then the page can be lost. In Section 3.4.1 we discuss the construction of locally decodable codes from PIR protocols as shown by [39]. More details on locally decodable codes can be found in the survey [54, 31]. Another interesting application of information-theoretic PIR, which demonstrates its universality, is for constructing information-theoretic multi-party private protocols for every function. Ishai and Kushilevitz [35] show that, for a constant k, there is an efficient information-theoretic PIR protocol if and only if every function has an *efficient* k-party information-theoretic private protocol.

Computational PIR. In computational PIR protocols a polynomial-time server cannot learn information on the index the client retrieves. In other words, unless the server runs in an unreasonable time, the privacy of the index is guaranteed. The first computational PIR protocol was a multi-server PIR protocol of [18] (assuming that one-way functions exist). Following this work, Kushilevitz and Ostrovsky [43] showed that there is a 1-server computational PIR protocol with sub-linear communication (assuming that the quadratic residuosity problem is hard). Subsequently, more efficient computational protocols, based on various hardness assumptions, were constructed [43, 52, 45, 14, 16, 44, 28]. The best 1-server PIR protocol was constructed by Lipmaa [44]; it is based on the so-called composite residuosity assumption and has communication complexity $O(\log^2 n)$ (ignoring the security parameter). It is important to note that 1-server computational PIR protocols with sublinear communication require some hardness assumptions [7, 23, 36].

Computational PIR protocols are used for constructing efficient protocols for more complex cryptographic tasks. They can be used to construct several cryptographic primitives, e.g., unconditionally hiding commitment [7], oblivious transfer protocols [23, 47], and collision-resistant hash functions [36]. Furthermore, PIR protocols can be used to construct efficient zero-knowledge arguments for a certain class of languages [53]. In Section 4.4, we another application of computational PIR suggested by [49]: using PIR protocols for filtering of encrypted data. That is, we want to construct a public encryption scheme where the encryption contains only information relevant to the client that generates the public key. The twist is that the server performing the encryption should not know the criteria for choosing the relevant information, nevertheless, the length of the encryption should be shorter than the encrypted database. See [49, 12] for more relevant reference on this subject. Other applications of PIR protocols for complexity theory are discussed in the survey [48].

Symmetric PIR. In the above discussion of the PIR problem, we only protect the privacy of the client. While each server is not allowed to learn information about the bit that the client is interested in, the client can learn many bits of the database. This might be problematic in many scenarios. For example, if the server wants to charge the client for each bit it retrieves, then the client gets extra information for free. Gertner, Ishai, Kushilevitz, and Malkin [30] defined symmetric private information retrieval, abbreviated SPIR, where the server does not learn any information and the client only learns the bit that it wants. Such protocols were actually considered before the introduction of PIR protocols and were called oblivious transfer, abbreviated OT, or all-or-nothing disclosure of secrets [50, 25, 13]. The name oblivious transfer, coined by Rabin [50], illustrates the nature of the protocol where the server transfers information to the client, while being oblivious to which information it transfers. However, prior to the PIR literature, the communication complexity of OT protocols was not optimized. In other words, SPIR protocols can be thought of as oblivious transfer protocols with sub-linear communication.

Communication efficient SPIR protocols were first studied, mainly in the information-theoretic, by Gertner et al. [30]. Communication efficient computational SPIR protocols were designed in [43, 52, 47, 16]. Specifically, Naor and Pinkas [47] designed an efficient transformation from PIR protocols to SPIR protocols. The best computational SPIR protocols, which are either designed explicitly [44] or constructed by using the above mentioned transformation from a PIR protocol (e.g., [28]), have the same communication complexity as PIR protocols, namely, $O(\log^2 n)$. SPIR protocols are used as a building block for obtaining sublinear-communication protocols for more general problems of secure computation (e.g., [26, 46, 15]).

1.2 Shortcomings of Current PIR Protocols

The Honest-but-Curious Model. Most of the PIR literature considers the so called "honest-but-curious" model. That is, it is assumed that the client and the server are honest, that is they follow their predefined protocol. However, after the execution of the protocol they try to infer as much information as they could from the transcript of the protocol. This model should be contrasted with the more realistic "malicious" model, where parties can deviate from the protocol to learn more information or to prevent the correct execution of the protocol. There are transformations from protocols secure against honest-but-curious parties to protocols secure against malicious parties [32]. However, in general, these transformations increase the communication complexity, thus, they cannot be used to design communication efficient PIR protocols secure against malicious parties. There are some papers that explicitly design PIR protocols secure against malicious parties (e.g., [52, 47, 16]). However, more research should be conducted in designing PIR and SPIR protocols secure against malicious parties. To conclude, the protocols described in this primer can only be used when it is safe to assume that the parties are indeed honest-but-curious.

Computation in PIR Protocols. One of the problems in implementing PIR protocols is the computation by the server/servers. In the information-theoretic protocols, the total computation of the servers is linear in n – the size of the database. In the computational PIR protocols, the situation is even worse – the computation of the server is $O(n)$ exponentiations in $\mathbb{Z}_p^*$ (or equivalent operations), where the cost of each exponentiation is, roughly, $O((\log p)^3)$ (where, for example, $\log p \approx 1024$). It is easy to see that in single-server PIR protocols, the computation of the server is at least n as if the server does not read some bit of the database while answering a query, then it can deduce that this bit is not the one that the client wants. In [9], it was shown that this is true also for multi-server PIR protocols (information-theoretic and computational) – the expected total computation of the servers has to be at least n. As the size of the database can be large, this poses a major problem in implementing PIR protocols in practice. Indeed, it is claimed by Sion and

Carbunar [51] that in current computational PIR protocols the computation is more expensive than sending the entire database.

Fortunately, there are two possible solutions to overcome this problem. The first solution, suggested in [9], is to allow preprocessing by the servers. That is, before getting queries, each server performs some computation and stores outputs of the computation in its memory. Thereafter, given any query of the client, the server can compute a short answer using sub-linear computation. For example, a k-server protocol with $O(n^{1/k+\epsilon})$ communication and work and $n^{O(1)}$ storage is presented in [9] for every constant $\epsilon > 0$. The second solution, suggested in [9, 37], is amortization. That is, a single query might require linear computation, however, if there are many queries, then the expected computation of the server per query is sub-linear. For example, in [37] a single-server computational PIR protocol is presented, where n^{δ} queries can be answered using $n^{\delta(1+o(1))}$ communication and $n^{1+o(1)}$ computation for every constant $\delta > 0$.

1.3 Aim and Organization

In this primer we discuss the PIR problem, survey the known results on PIR, describe the basic construction of information-theoretic and computational PIR protocols, and mention applications of PIR protocols. We discuss the best known protocols without describing them. For more details, the interested reader is referred to more advanced surveys on PIR [27, 1, 48] and the relevant papers. The primer is intended for readers that do not necessary have background in cryptography. In particular, we supply the results from cryptography, combinatorics, and number theory that are needed to understand the protocols we describe.

Organization. In Section 2 we define information-theoretic and computational PIR protocols and discuss the pros and cons of the two types of protocols. In Section 3 we describe constructions of information-theoretic PIR protocols. Thereafter in Section 4 we present constructions of computational PIR protocols. Finally, in Section 5 we define and show constructions of symmetric PIR protocols.

2 Information-Theoretic vs. Computational PIR

In this section we will define two types of privacy for PIR protocols, information-theoretic and computational and shortly discuss their cons and pros.

We will consider a k-server PIR protocol, where $k \geq 1$ is an integer. In the computational case usually $k = 1$ and in the information-theoretic case usually k is a constant greater than 1. PIR protocols involve k servers $\mathcal{S}_1, \ldots, \mathcal{S}_k$, each server holds the same n-bit string x (the database), and a client $\mathcal{C}$ which wants to retrieve some bit x_i without revealing i. In this primer, we only consider *one-round* PIR protocols. This is justified as the most efficient known protocols use a single round of communication. In a one-round protocol, a client first generates a tuple of k queries based on the index it wants to retrieve. To protect the privacy of the client, this tuple of queries is a *randomized* function of the index. The client sends the queries to the servers, where the server $\mathcal{S}_j$ receives the jth query in the tuple. Each server computes an answer, which is a deterministic function of the query and the database, and sends the answer to the client. Finally, the client, after receiving the k answers, computes x_i. The formal definition of a PIR protocol and its correctness appear in Definition 2.1. The privacy requirements of information-theoretic and computational PIR protocols appear in Definition 2.2 and Definition 2.4 respectively.

Definition 2.1 (PIR) *A PIR protocol $\mathcal{P} = (\mathcal{Q}, \mathcal{A}, \mathcal{C})$ is composed of three algorithms: a query algorithm $\mathcal{Q}$, an answer algorithm $\mathcal{A}$, and a reconstruction algorithm $\mathcal{C}$. At the beginning of the protocol, the client* *has an input i, it picks a random string r with uniformly distribution and it computes a tuple of k queries $(q_1, \ldots, q_k) = \mathcal{Q}(1^n, i, r)$, where n is the length of the database. The client then sends the query q_j to Server $\mathcal{S}_j$ for every $j \in \{1, \ldots, k\}$. Server $\mathcal{S}_j$, where $1 \leq j \leq k$, responds with an answer $a_j = \mathcal{A}(j, q, x)$. Finally, the client computes the bit x_i by applying the reconstruction algorithm $\mathcal{C}(1^n, i, r, a_1, \ldots, a_k)$. We require the following correctness requirement:*

Correctness. *The client always outputs the correct value of x_i. Formally, for every database length n, every database $x \in \{0,1\}^n$, every index $i \in \{1, ..., n\}$, and every random string r, if $(q_1, \ldots, q_k) = \mathcal{Q}(1^n, i, r)$ and $a_j = \mathcal{A}(j, q, x)$ for $j = 1, \ldots, k$, then*

$$\mathcal{C}(1^n, i, r, a_1, \ldots, a_k) = x_i.$$

The communication complexity *of a PIR protocol $\mathcal{P}$ is the* total *number of bits communicated between the client $\mathcal{C}$ and the k servers maximized over all choices of $x \in \{0,1\}^n$, $i \in \{1, \ldots, n\}$, and random inputs. The* query length *of $\mathcal{P}$ is the maximal number of bits sent from $\mathcal{C}$ to* any *single server, and the* answer length *is the maximal number of answer bits sent by* any *server.*

When discussing computational protocols, we want to require that all algorithms run in polynomial time in the database length n. However, the query length and the answer length might be much shorter than n (e.g., polylogarithmic in n). Thus, in the definition of PIR, the input of the query algorithm $\mathcal{Q}$ and the reconstruction algorithm $\mathcal{C}$ contains the length of the database n (in an unary encoding). The input of the answer algorithm $\mathcal{A}$ contains this parameter explicitly as x is an input of $\mathcal{A}$.

Definition 2.2 (Information-Theoretic PIR) *An information-theoretic k-server PIR protocol $\mathcal{P} = (\mathcal{Q}, \mathcal{A}, \mathcal{C})$ is a PIR protocol (as defined in Definition 2.1) satisfying the following privacy requirements:*

Client's Perfect Privacy. *Each server learns no information about i. Formally, for every n, every two indices $i_1, i_2 \in \{1, \ldots, n\}$, every $j \in [k]$, and every query q, the sever $\mathcal{S}_j$ cannot know if the query was generated with index i_1 or with index i_2, formally,*

$$\Pr[\mathcal{Q}_j(1^n, r, i_1) = q] = \Pr[\mathcal{Q}_j(1^n, r, i_2) = q],$$

where $\mathcal{Q}_j$ denotes the jth query in the k-tuple $\mathcal{Q}$ outputs and the probability is taken over the uniform distribution over the random string r of $\mathcal{Q}$.

In Definition 2.2 we require that each server by its own cannot learn information about the secret index. That is, we assume that the servers do not collude and do not communicate between them self. This requirement is called 1-privacy. In this work, we do not consider the more general case of t-private PIR, where the client's privacy holds against collusions of up to t servers. Again, 1-private protocols are the more common protocols in the PIR literature. See more details on t-private PIR protocols in Section 3.3.1.

We next define computational PIR protocols in which the privacy requirement is relaxed compared to information-theoretic PIR protocols. In a computational PIR protocol a polynomial-time server cannot learn information on the index the client retrieves. In other words, unless the server runs in an unreasonable time, the privacy of the index is guaranteed. The formal definition of computational privacy requires some notation and definitions that are standard in cryptography; the primer can be understood without understanding the formal definition. A distribution ensemble $\{X^n\}_{n=1}^{\infty}$ is an infinite sequence of random variables $X^1, X^2, \ldots$. We consider such ensemble since we want to speak about asymptotic behavior.

Definition 2.3 (Indistinguishable Distributions) *Two distribution ensembles* $\{Z^n\}_{n=1}^{\infty}$ *and* $\{Y^n\}_{i=n}^{\infty}$ *are indistinguishable if every probabilistic polynomial-time algorithm* $\mathcal{M}$ *cannot distinguish if it gets an input generated according to* Z^n *or according to* Y^n*, that is, for every integer* $c \geq 1$ *and for every sufficiently large* n

$$\left| \Pr_{z \in Z^n} \left[\mathcal{M}\left(z, 1^n\right) = 1 \right] - \Pr_{y \in Y^n} \left[\mathcal{M}\left(y, 1^n\right) = 1 \right] \right| \leq \frac{1}{n^c}.$$

Definition 2.4 (Computational PIR) *A computational 1-server PIR protocol* $\mathcal{P} = (\mathcal{Q}, \mathcal{A}, \mathcal{C})$ *is a PIR protocol (as defined in Definition 2.1), where* $\mathcal{Q}$*,* $\mathcal{A}$*, and* $\mathcal{C}$ *are polynomial time algorithms and the following privacy requirements is satisfied:*

Client's Computational Privacy. *A polynomial-time server has no information about the bit that the client tries to retrieve: For every two sequences of indices* $\{i_n\}_{n=1}^{\infty}$ *and* $\{j_n\}_{n=1}^{\infty}$*, where* $1 \leq i_n, j_n \leq n$*, the distribution ensembles* $\{\mathcal{Q}(1^n, i_n, r)\}$ *and* $\{\mathcal{Q}(1^n, j_n, r)\}$ *are indistinguishable in polynomial-time, where in the random variables* $\mathcal{Q}(1^n, i_n, r)$ *and* $\mathcal{Q}(1^n, j_n, r)$ *the length* n *and the indices* i_n *and* j_n *are fixed and* r *is chosen at random.*

2.1 Comparison of the Two Types of PIR Protocols

Computational PIR protocols are preferable to information-theoretic ones for obvious reasons. First, the database can be kept on one server and there is no need for replication. Thus, there is no risk that servers will collude and learn information on the retrieved index. Furthermore, the best known single-server computational protocols obtain better asymptotic communication complexity than information-theoretic protocols with a constant number of servers. As a result, computational PIR protocols are used as a building box in several cryptographic protocols, whereas information-theoretic PIR protocols cannot be used in these protocols.

However, for typical real-life parameters the known single-server protocols are less efficient than known multi-server (even 2-server) protocols due to their high computational overhead. That is, in information-theoretic protocols the computation of the server is typically $O(n)$, while in the known computational protocols the server executes $O(n)$ "public key operations" operations (e.g., modular exponentiations); these operations are much more expensive. Furthermore, single-server protocols have some *inherent* limitations which can only be avoided in a multi-server setting. For instance, it is impossible for a (sublinear-communication) single-server PIR protocol to have very short queries (say, $O(\log n)$ bits long) sent from the client to the server, or very short answers (say, one bit long) sent in return. These two extreme types of protocols, which can be realized in the information-theoretic setting, have various applications [22, 9]. Finally, the close relation between information-theoretic PIR and locally decodable codes [39], which does not hold for computational protocols, further motivates the study of information-theoretic PIR.

3 Information-Theoretic Private Information Retrieval

Information-theoretic PIR protocols offer perfect privacy, however, they require replication of the data among several servers. In this section, we present several information-theoretic PIR protocols. In Section 3.1 we describe a 2-server protocol with communication complexity $O(\sqrt{n})$ and in Section 3.2 we present a 2-server protocol with communication complexity $O(\sqrt[3]{n})$. In Section 3.3 we discuss the history of information-theoretic PIR and some extensions of the PIR problem. Finally, in Section 3.4 we describe two applications of information-theoretic PIR: how to construct, using PIR protocols, locally decodable codes and multi-party private protocols for any function.

Servers' Input: a database $x \in \{0,1\}^n$.

Client's Input: an index $i \in \{1, \ldots, n\}$.

1. The client selects a random set $Q^1 \subseteq \{1, \ldots, n\}$ with uniform distribution and computes $Q^2 = Q^1 \bigtriangleup \{i\}$. The client sends Q^j to $\mathcal{S}_j$ for $j \in \{1, 2\}$.

2. Server $\mathcal{S}_j$ (where $j \in \{1, 2\}$) computes

$$a^j \stackrel{\text{def}}{=} \bigoplus_{\ell \in Q^j} x_\ell \,,$$

and sends a^j to the client.

3. The client outputs $a^1 \oplus a^2$.

Protocol 1: A two server protocol with $O(n)$-bit query and 1-bit answer per server.

3.1 A 2-Server Protocol with Complexity $\sqrt{n}$

As mentioned in the introduction, in the naive PIR protocol the server sends the entire database to the client. This is a 1-server protocol whose communication complexity is n. Specifically, the client sends no communication and the server sends n bits to the client. In this section we describe a 2-server protocol with communication complexity $O(\sqrt{n}\,)$. This protocol is inferior to the 2-server protocol that will be presented in Section 3.2 whose communication is $O(\sqrt[3]{n}\,)$. The purpose of describing this inferior protocol is two-fold. First, we want to describe a simple protocol that demonstrates the definitions and shows that saving in communication is possible. Second, this protocol contains ideas that will be used in the computational protocol described in Section 4.

We will describe the protocol with communication $O(\sqrt{n}\,)$ in two stages. In the first stage we will describe a protocol in which the client sends n bits to each server and each server sends one bit back to the client. The total communication of this protocol is $2n + 2$ which is worse than in the naive protocol. However, we will show, in the second stage, how to balance the communication, that is, we will decrease the communication from the client to each server to $\sqrt{n}$ while increasing the communication from each server to the client to $\sqrt{n}$.

Notation. The protocols we describe use the properties of the exclusive-or operator as summarized next: Given two bits a and b, $a \oplus b = 0$ if $a = b$ and $a \oplus b = 1$ if $a \neq b$. In particular, $a \oplus a = 0$; we thus say that the exclusive-or operation has the "canceling" property.

For a set Q and an index i define

$$Q \bigtriangleup \{i\} \stackrel{\text{def}}{=} \begin{cases} Q \cup \{i\} & i \notin Q, \\ Q \setminus \{i\} & i \in Q. \end{cases}$$

First Stage. In Protocol 1 we describe the first stage. In this protocol the client sends a random subset Q^1 of indices to the first server. That is, the server chooses each of the 2^n subsets of $\{1, \ldots, n\}$ with probability $1/2^n$. The client sends the same random subset to the second server with one difference: the desired index

Servers' Input: a database $x \in \{0,1\}^n$.

Client's Input: an index $i \in \{1, \ldots, n\}$.

1. Let $m = \lceil \sqrt{n}\, \rceil$, $i_1 = \lceil i/m \rceil$, and $i_2 = ((i-1) \bmod m) + 1$.
2. The client selects a random set $Q^1 \subseteq \{1, \ldots, m\}$ with uniform distribution and computes $Q^2 = Q^1 \bigtriangleup \{i_2\}$. The client sends Q^j to $\mathcal{S}_j$ for $j \in \{1, 2\}$.
3. Server $\mathcal{S}_j$ computes for every $1 \leq t \leq m$ $a^{j,t} \stackrel{\text{def}}{=} \oplus_{\ell \in Q^j} x_{t,\ell}$. The server sends $a^{j,1}, \ldots, a^{j,1}$ to the client.
4. The client outputs $a^{1,i_1} \oplus a^{2,i_2}$.

Protocol 2: A two server protocol with total communication $O(\sqrt{n}\,)$.

i is in the subset Q^2 sent to the second server if and only if it is *not* in Q^1. Each server responds with the exclusive-or of the bits of the database whose indices are in the set it receives. The client computes the exclusive-or of the two bits that it gets and this is its output. We next explain why this protocol is correct, that is, why the exclusive-or of the two bits is x_i. Informally, this is true by the "canceling" property of the exclusive-or operator, that is, for every index ℓ that is in both Q^1 and Q^2 the bit x_ℓ appears in both answers a^1 and a^2, thus, it is canceled. The only index that appears in exactly one of the sets Q^1 and Q^2 is i, thus, the bit x_i is not canceled and is the output. Formally,

$$a^1 \oplus a^2 = \left(\bigoplus_{\ell \in Q^1} x_\ell \right) \oplus \left(\bigoplus_{\ell \in Q^2} x_\ell \right) = \left(\bigoplus_{\ell \in Q^1 \cap Q^2} x_\ell \oplus x_\ell \right) \oplus x_i = x_i.$$

We argue that this protocol is private. As the set Q^1 is a random set chosen independently of i, Server $\mathcal{S}_1$ gets no information on i. As for $\mathcal{S}_2$, the set Q^2 is also a random set independent of i as explain below. As the probability that $i \in Q^1$ is $1/2$, the probability that $i \in Q^2$ is $1/2$.[1] Clearly, for every $\ell \neq i$ the probability that $\ell \in Q^2$ is $1/2$. Furthermore, all these probabilities are independent. Thus, also Server $\mathcal{S}_2$ gets a random subset of indices independent of i and, thus, gets no information on i.

We next consider the communication complexity of the protocol. The client represents each set Q^j by an n-bit binary string whose ℓth bit is 1 if $\ell \in Q^j$ and 0 otherwise. Thus, the query length per server is n. The answer of each server is one bit. Thus, the total communication is $2n + 2$.

Second Stage. In Protocol 1 the communication is not balanced: while the query length is big, each answer is one bit. We use a general balancing technique of [20] to balance the communication. We view the database as a two-dimensional array. I.e., letting $m = \lceil \sqrt{n}\, \rceil$, the database is viewed as $\langle x_{i_1,i_2} \rangle_{1 \leq i_1, i_2 \leq m}$. We assume that the client wants the index $i = \langle i_1, i_2 \rangle$ in the database, where

- $i_1 = \lceil i/m \rceil$, and
- $i_2 = ((i-1) \bmod m) + 1$.

[1] This is equivalent to flipping a coin at random and then, without looking at the output, negating the output. The negated output is still distributed uniformly.

That is, i_1, i_2 are the unique indices in $\{1, \ldots, m\}$ such that $i = (i_1 - 1)m + i_2$. Now, the client prepares a query as in Protocol 1, where its index is i_2 and the database length is m. The server considers each row of the array as a database and answers as in Protocol 1. That is, its answer is an m-bit string whose ℓth bit is the answer of the server with the ℓth row of x as the database. The client takes the two bits containing the answers for the i_1th row of x and computes the exclusive-or of them. This value is the desired bit x_{i_1,i_2}. The formal description appears in Protocol 2. Its correctness and privacy follow from those of Protocol 1. We next analyze its communication complexity. The client sends a subset of $\{1, \ldots, m\}$, represented by an m-bit string, to each server. Each server sends one bit per row of x, and, as there are m rows in x, each server sends an m-bit string to the client. All together, the communication complexity is $4m = O(\sqrt{n})$. Note that in this protocol the client learns many bits of the database; this does not contradict any requirement of PIR.

3.2 PIR Protocols based on Multivariate Polynomials

In this section we present a 2-server PIR protocol with communication $O(n^{1/3})$.[2] The results of this section are more technical than the other results described in this survey; the rest of the paper can be understood without reading this subsection.

The protocol we present uses multivariate polynomials over $\mathbb{F}_2$ – the finite field with two elements. The elements of $\mathbb{F}_2$ are $\{0, 1\}$, addition is exclusive-or, and multiplication is the usual multiplication of 0 and 1. We recall some elementary facts about polynomials over $\mathbb{F}_2$. First, for a variable Z and an integer $c > 0$, $Z^c = Z$ (since $0^c = 0$ and $1^c = 1$). The degree of a monomial over $\mathbb{F}_2$ is, thus, the number of variables in the monomial. The degree of a polynomial is the maximum of the degrees of the monomials in the polynomial. E.g., the degree of the polynomial $Z_1Z_2 + Z_1Z_6Z_7$ is 3. The term *degree-d* polynomial refers to a multivariate polynomial whose degree is *at most* d. In this section, variables in multivariate polynomials are denoted with capital letters, e.g., Z_h; assignments to these variables are denoted with small letters, e.g., z_h.

We represent the n-bit database x by a multivariate polynomial $P_x(Z_1, \ldots, Z_m)$ over $\mathbb{F}_2$. To represent the database by a polynomial, we will represent each index $i \in \{1, \ldots, n\}$ by an m-bit string $E(i) \in \{0, 1\}^m$ (where m and E will be specified later). The polynomial P_x represent x in the following sense:

$$\forall i \in \{1, \ldots, n\}, \quad P_x(E(i)) = x_i. \tag{1}$$

We consider P_x as a representation of x since x can be reconstructed from P_x. We do not care about the value $P_x(z)$ for assignments z which are not of the form $E(i)$, for some i. The degree d of this polynomial will be a constant and the number of variables is

$$m = dn^{1/d}. \tag{2}$$

Example 3.1 *Let $n = 4$. We represent a database $x = (x_1, x_2, x_3)$ by a polynomial of degree $d = 2$ with $m = 3$ variables. We take $E(1) = (1, 1, 0)$, $E(2) = (1, 0, 1)$, and $E(3) = (1, 1, 0)$. The polynomial P_x we construct is*

$$P_x(Z_1, Z_2, Z_3) = x_1Z_1Z_2 + x_1Z_1Z_3 + x_1Z_2Z_3.$$

Note that the degree of P_x is indeed $d = 2$ as every monomial contains two variables. To see that this polynomials represents x as in (1) consider, for example,

$$P_x(E(2)) = P_x(1, 0, 1) = x_1 \cdot 1 \cdot 0 + x_2 \cdot 1 \cdot 1 + x_3 \cdot 0 \cdot 1 = x_2.$$

[2] This protocol can be generalized to a k-server PIR protocol with communication complexity $O(n^{1/(2k-1)})$.

We next describe the general construction of E and P_x. Let $E(1), \ldots, E(n)$ be n arbitrary distinct binary strings of length m with exactly d ones. Such vectors exist since, by the choice of m in (2), there are enough distinct m-bit strings with exactly d ones, that is, $\binom{m}{d} \geq (m/d)^d = (dn^{1/d}/d)^d = n$. Define

$$P_x(Z_1, \ldots, Z_m) \stackrel{\text{def}}{=} \sum_{i=1}^{n} x_i \prod_{\ell : E(i)_\ell = 1} Z_\ell,$$

where $E(i)_\ell$ is the ℓth bit of $E(i)$. In the above $x_1, \ldots, x_n$ are fixed and $Z_1, \ldots, Z_m$ are variables. Since each $E(i)$ has d ones, the degree of P_x is d. Recall that for $i \neq j$, $E(i)$ and $E(j)$ are two strings with exactly d ones. Thus, there exits at least one index ℓ such that $(E(j))_\ell = 1$ while $(E(i))_\ell = 0$. This implies that each assignment $E(i)$ to the variables $Z_1, \ldots, Z_m$ satisfies exactly one monomial in P_x whose coefficient is x_i; therefore, $P_x(E(i)) = x_i$. For an example of this construction, consider the polynomial in Example 3.1.

Note that P_x is determined by x and is, thus, known to all servers. That is, the coefficients $x_1, \ldots, x_n$ are fixed and known by the servers and the strings $E(1), E(2), \ldots, E(n)$ are known to all servers (as they are part of the specification of the protocol). The client $\mathcal{C}$, on the other hand, does not know x. It has an index i, pointing to the bit from x it is interested in, and it can compute $E(i)$. Hence, the PIR problem is reduced to the problem of evaluating $P_x(E(i))$ while keeping $E(i)$ secret from each server. To this end, $\mathcal{C}$ chooses at random an m-bit string $\vec{y}_1$ and computes

$$\vec{y}_2 = E(i) \oplus \vec{y}_1, \tag{3}$$

where $\oplus$ is a bitwise exclusive-or of the strings. The client sends $\vec{y}_j$ to Server $\mathcal{S}_j$. Since each $\vec{y}_j$ is uniformly and independently distributed, a single server can learn no information about i (this is similar to the fact that each server learns no information in the 2-server protocols presented in Section 3.1). The client's goal is to evaluate $P_x(\vec{y}_1 \oplus \vec{y}_2) = P_x(E(i)) = x_i$. For $j \in \{1, 2\}$, the string $\vec{y}_j$ consists of m bits denoted by $y_{j,1}, \ldots, y_{j,m}$. Equivalently, we can think of each variable Z_h of P_x as the sum of 2 variables: $Z_h = Y_{1,h} + Y_{2,h}$. The value $P_x(E(i))$ is obtained by assigning the value $y_{j,h}$ to each variable $Y_{j,h}$. Let Q_x be the polynomial obtained by viewing P_x as a polynomial in the variables $\{Y_{j,h} : j \in \{1, 2\}, h \in \{1, \ldots, m\}\}$. That is,

$$Q_x(Y_{1,1} \ldots, Y_{1,m}, Y_{2,1} \ldots, Y_{2,m}) \quad \stackrel{\text{def}}{=} \quad P_x(Y_{1,1} + Y_{2,1}, Y_{1,2} + Y_{2,2}, \ldots, Y_{1,m} + Y_{2,m}). \tag{4}$$

This is a degree-d polynomial in $2m$ variables.

Example 3.2 *Consider the polynomial from Example 3.1 with $x = (1, 1, 0)$, that is, $P_{(1,1,0)}(Z_1, Z_2, Z_3) = Z_1 Z_2 + Z_1 Z_3$. We construct the polynomial $Q_{(1,1,0)}$ as follows.*

$$\begin{aligned}
&Q_{(1,1,0)}(Y_{1,1}, Y_{2,1}, Y_{1,2}, Y_{2,2}, Y_{1,3}, Y_{2,3}) \\
&\quad = \quad (Y_{1,1} + Y_{2,1})(Y_{1,2} + Y_{2,2}) + (Y_{1,1} + Y_{2,1})(Y_{1,3} + Y_{2,3}) \\
&\quad = \quad Y_{1,1}Y_{1,2} + Y_{1,1}Y_{2,2} + Y_{2,1}Y_{1,2} + Y_{2,1}Y_{2,2} + Y_{1,1}Y_{1,3} + Y_{1,1}Y_{2,3} + Y_{2,1}Y_{1,3} + Y_{2,1}Y_{2,3}.
\end{aligned}$$

That is, after opening the parenthesis, the polynomial $Q_{(1,1,0)}$ contains 8 monomials. Its degree is 2 as the degree of $P_{(1,1,0)}$.

Suppose, for the moment, that $d = 1$. In this case, Q_x is a polynomial of degree 1 that can written as a sum of two polynomials, one polynomial $Q_{x,1}$ in the variables $Y_{1,1}, \ldots, Y_{1,m}$ and the other polynomial $Q_{x,2}$ in the variables $Y_{2,1}, \ldots, Y_{2,m}$. Using these polynomials, we construct a PIR protocol as described in

Servers' Input: a database $x \in \{0,1\}^n$.

Client's Input: an index $i \in \{1,\dots,n\}$.

1. Let $d = 1$ and $m = n$.
2. $\mathcal{C}$ picks points $\vec{y}_1, \vec{y}_2$ as in (3), and sends $\vec{y}_1$ to server $\mathcal{S}_1$ and $\vec{y}_2$ to server $\mathcal{S}_2$.
3. Each server answers $\mathcal{C}$ with the sum (in $\mathbb{F}_2$) of all monomials assigned to it.
4. $\mathcal{C}$ sums these answers and outputs it.

Protocol 3: A two server protocol with total communication $O(n)$.

Protocol 3. By the above discussion, the sum that the client outputs is equal to x_i, thus, Protocol 3 is correct. The query length of this protocol is $2m+2 = O(n)$ bits while each answer is 1-bit long. Thus, we can use the balancing technique and get a protocol with communication complexity $O(\sqrt{n})$.

We achieve a better communication complexity by taking the degree of P_x (and Q_x) to be 3. Consider a monomial M of the polynomial Q_x; the monomial M depends on at most 3 variables. Since the value of each variable is known to one of the servers (that is, Server $\mathcal{S}_j$ knows the value $y_{j,\ell}$ of the variable $Y_{j,\ell}$), there exists a server that is missing the value at most one of the variables of M; we assign the monomial M to this server. For example, for a monomial $Y_{2,1}Y_{1,2}Y_{2,3}$, Server $\mathcal{S}_2$ knows the values of $Y_{2,1}$ and $Y_{2,3}$ and does not know the value of $Y_{1,2}$. Each server $\mathcal{S}_j$ can therefore substitute the values for all but (at most) one of the variables of each monomial M assigned to it. After substituting these values, the sum of the monomials assigned to $\mathcal{S}_1$ (respectively $\mathcal{S}_2$) can be expressed as a degree-1 polynomial $P_1(Y_{2,1},\dots,Y_{2,m})$ (respectively $P_2(Y_{1,1},\dots,Y_{1,m})$). Note, however, that if the client could learn all polynomials P_1 and P_2, then, by substituting the correct values $y_{j,h}$ for all their variables and summing up the values of the k polynomials, it will get

$$\begin{aligned} P_1(y_{2,1},\dots,y_{2,m}) + P_2(y_{1,1},\dots,y_{1,m}) &= Q_x(\vec{y}_1,\vec{y}_2) \\ &= P_x(\vec{y}_1+\vec{y}_2) \\ &= P_x(E(i)) \;=\; x_i. \end{aligned}$$

The PIR protocol starts as before, but this time each server $\mathcal{S}_j$ sends the $m+1$ *coefficients* (a single bit each) of the degree-1 polynomial P_j. Each query of the server is an m-bit string. The formal description of the polynomial appears in Protocol 4. As the degree of P_1 (and P_2) is 1 and there are m variables, the polynomial P_1 has $m+1$ coefficients, each coefficient is either 0 or 1. The communication complexity of this protocol is, therefore, $O(m) = O(n^{1/3})$ bits. To summarize, we have shown how to obtain a 2-server PIR protocol with the best known communication complexity, namely, $O(\sqrt[3]{n})$.

3.3 Historical Notes and Extensions

Information-theoretic PIR protocols were introduced and constructed by Chor et al. [20]. In particular, they construct the best known 2-server protocol with communication complexity $O(n^{1/3})$ (where n is the database length). They also presented a k-server protocol with communication complexity $O(n^{1/k})$ for every $k > 1$. Subsequently, more efficient constructions of k-server protocols for $k > 2$ were presented

Servers' Input: a database $x \in \{0,1\}^n$.

Client's Input: an index $i \in \{1,\ldots,n\}$.

1. Let $d = 3$ and $m = 3n^{1/3}$.
2. $\mathcal{C}$ picks $\vec{y}_1 \in \{0,1\}^m$ at random with uniform distribution and computes $\vec{y}_2 = E(i) \oplus \vec{y}_1$ (as in (3)). The client $\mathcal{C}$ sends $\vec{y}_1$ to server $\mathcal{S}_1$ and $\vec{y}_2$ to server $\mathcal{S}_2$.
3. Let Q_x be the polynomial defined in (4). Assign to $\mathcal{S}_1$ all monomials of Q_x that contain at most 1 variable $Y_{2,h}$ for some $h \in \{1,\ldots,m\}$. Assign all other monomials in Q_x to $\mathcal{S}_2$.
4. Server $\mathcal{S}_1$ (respectively, $\mathcal{S}_2$) substitutes the value $y_{1,1},\ldots,y_{1,m}$ (respectively, $y_{2,1},\ldots,$ $y_{2,m}$) in the monomials assigned to it and sums the resulting monomials to get a polynomial $P_1(Y_{2,1},\ldots,Y_{2,m})$ (respectively, $P_2(Y_{1,1},\ldots,Y_{1,m})$).
5. Each server $\mathcal{S}_j$ answers $\mathcal{C}$ with the coefficients of the polynomial P_j.
6. $\mathcal{C}$ computes $P_1(\vec{y}_2) + P_2(\vec{y}_1)$ and outputs this value.

Protocol 4: A two server protocol with total communication $O(\sqrt[3]{n})$.

in [2, 38, 6, 56, 8, 59, 40]. The best known 3-server PIR protocol was constructed in a lovely work of Yekhanin [59]; assuming that there are infinitely many Mersenne primes, he constructs a 3-server PIR protocol with communication complexity $n^{O(1/\log\log n)}$ for infinitely many values of n. Specifically, his protocol implies, without any assumptions, a 3-server PIR protocol with communication complexity n^{ϵ} for some $\epsilon < 10^{-7}$. To date, Yekhanin's protocol is the best k-server PIR protocol for every constant k. There are better protocols when the number of servers depends on the length of the database (implicitly in [4, 5] and explicitly in [20, 6, 56]); in particular, there is an $O(\log n)$-server protocol with $O(\log^2 n \log\log n)$ communication complexity.

The PIR problem was studied in many different settings and extensions other than the basic one discussed above in, e.g., [9, 10, 19, 22, 29, 41, 55]. We will describe two such extensions. The reader is referred to the survey by Gasarch [27] for an overview of other extensions of information-theoretic PIR.

3.3.1 t-private Information-Theoretic PIR

In Definition 2.2, the definition of information-theoretic PIR, we require that every individual server does not learn information on the retrieved index. However, two colluding servers might be able to learn this index. Protocols that addresses this concern are t-private PIR protocols in which the client is protected against collusions of up to t servers. t-private PIR protocols were designed in [20, 6, 56, 3]. Specifically, t-private k-server PIR protocols with communication complexity $O(n^{1/\lfloor(2k-1)/t\rfloor})$ were designed by [6, 56]. Furthermore, a transformation from any 1-private k-server PIR protocols to t-private k^t-server PIR protocols with similar communication complexity is given in [3]. Thus, together with the 1-private 3-server PIR protocol of [59], we get for every constant $t > 0$ a t-private 3^t-server PIR protocol with communication complexity $n^{O(1/\log\log n)}$.

The transformation of [3] increases the number of servers from k to k^t. It is open if there exists a

Servers' Input: a database $x \in (\{0,1\}^b)^n$.

Client's Input: an index $i \in \{1,\ldots,n\}$.

Client's Output: the ith block of x (containing b bits).

1. The client prepares a query $(q_1,\ldots,q_k)$ for the index i using a PIR protocol for bits with n as the database length. The client sends q_j to $\mathcal{S}_j$ for every $j \in \{1,\ldots,k\}$.
2. Each server $\mathcal{S}_j$ computes the answers to the query q_j for each of the b databases and sends the b answers to the client.
3. For every $\ell \in \{1,\ldots,b\}$ the client reconstructs the ℓth bit of the block using the answers that it got for the ℓth database.

Protocol 5: A PIR protocol for blocks.

transformation that increases the number of servers to less than k^t. However, such transformation must increase the number of servers to at least kt. By simulation arguments, it can be shown that if there is a t-private kt-server PIR, then there is a 1-private k-server PIR protocol with the same communication complexity. We next mention two additional conclusions of this result. First, by the impossibility result for 1-server PIR [20], the communication complexity of information-theoretic k-private k-server PIR is at least n. Second, without improving the best known 2-server PIR protocol, for $t \geq k/2$ one cannot expect to construct a t-private k-server PIR protocol with communication complexity better than $O(n^{1/3})$.

3.3.2 PIR with Blocks

In many applications, the database is composed of n blocks, each block contains b bits. The client wants to retrieve a block instead of a bit. A trivial solution is to consider the database as containing bn bits and the servers and the client perform b independent PIR executions for each of the b bits of the block. However, there is a more efficient way to construct a protocol for retrieving blocks. We consider the database as b databases of n bits each, where the ℓth database contains the ℓth bit of each block, that is, the ith bit in the ℓth database is the ℓth bit in the ith block in the original database. The client sends a query for the index i using a regular PIR protocol (that is, a protocol with blocks of size 1) with a database of size n. Each server returns b answers to the query, one answer for each of the b databases it holds. The protocol is formally described in Protocol 5. This transformation from regular PIR to PIR for blocks can be applied to every PIR protocol (information-theoretic or computational). The length of the query does not changed in the transformation and the length of each answer is multiplied by b.

3.4 Application of Information-Theoretic PIR

3.4.1 Constructing Locally Decodable Error-Correcting Codes

An interesting application of information-theoretic PIR protocols is for constructing *locally decodable* error-correcting codes. A k-query locally decodable code allows to encode a database $x \in \{0,1\}^n$ into a string y of length m, such that even if a large portion of y is corrupted, each bit of x can still be decoded *with high probability* by probing k, randomly selected, bits of y. That is, if a client needs only one bit of the database, then it does not need to decode the entire database as in classical error correcting codes; it rather can probe few bits of the encoding and decode the desired bit efficiently. The best known locally decodable code is a 3-query code of [59] with encoding size $m = 2^{n^{O(1/\log\log n)}}$ (assuming that infinitely many Mersenne primes exist). The reader can consult [54, 31, 58] for surveys containing formal definitions and results on locally decodable codes, and their applications.

Katz and Trevisan [39] have shown a close relation between locally decodable codes and information-theoretic PIR. In particular, any information-theoretic k-server PIR protocol with a one-bit answer can be transformed into a k-query locally decodable code whose length m is exponential in the length of the PIR queries. We next show how to construct a 4-query locally decodable code whose length is $m = 2^{2\sqrt{n}}$. We start with constructing a 4-server PIR protocol whose query length is $2\sqrt{n}$ and answer length is 1. We then transform this 4-server PIR protocol to the 4-query locally decodable code.

The 4-server PIR protocol we describe is a two-dimensional version of Protocol 1. In Protocol 1, the client sends to each server a subset of $\{1,\ldots,n\}$ such that all indices, except for i, appear either in none of the sets or in both sets, and the index i appears in exactly one of the sets. In the 4-server protocol, we consider x as an $m \times m$ matrix (where $m = \lceil \sqrt{n} \rceil$) and sent each of the four servers a sub-cube of the cube $\{1,\ldots,m\} \times \{1,\ldots,m\}$. The requirement here is:

Requirement 1 *All indices, except for i, appear in an even number of sub-cubes (that is, either in all 4, or in exactly 2, or in none of the sub-cubes), and the index i appears in exactly one of the sub-cubes.*

Each server responds with the exclusive-or of the bits of the database in the sub-cube it gets, and the client computes the exclusive-or of 4 bits it gets. All the bits, except for the bit in location i, cancel them self, thus, the result is the desired bit. The protocol is described in Protocol 6. In this protocol, the client sends a sub-cube $Q_j \times R_j$ to Server $\mathcal{S}_j$ for $1 \leq j \leq 4$. Notice that the index i_1 is in exactly two sets Q_j, the index i_2 is in exactly two sets R_j, and the pair $(i_1, i_2) \in Q_j \times R_j$ for exactly one j. This implies that Requirement 1 holds in Protocol 6, thus as explained above, this protocol is correct. The client sends the description of two subsets of $\{1,\ldots,m\}$ to each server, thus, the query length is $2m = 2\sqrt{n}$. Furthermore, each server responds with a one bit answer.

We next describe the transformation from the PIR protocol described in Protocol 6 to a locally decodable code with encoding length $2^{2\sqrt{n}}$. Given a word x of length n we encode it to a word y of length $2^{2\sqrt{n}}$. Let $m = \lceil \sqrt{n} \rceil$. As in Protocol 6, we consider x as an $m \times m$ matrix. Furthermore, it would be convenient to consider y as a matrix whose columns are rows are indexed by subsets of $\{1,\ldots,m\}$, that is, $y = (y_{Q,R})_{Q,R\subseteq\{1,\ldots,m\}}$. As there are $2^{\sqrt{n}}$ possible values for Q and $2^{\sqrt{n}}$ possible values for R, there are $2^{\sqrt{n}} \cdot 2^{\sqrt{n}} = 2^{2\sqrt{n}}$ coordinates in y. The value of $y_{Q,R}$ is the answer of a server in Protocol 6 to the query Q, R, that is,

$$y_{Q,R} \stackrel{\text{def}}{=} \bigoplus_{t\in Q,\ell\in R} x_{t,\ell}.$$

Now assume that a receiver holds a (possibly corrupted) y and it wants to decode x_{i_1,i_2}. It picks at random a set $Q \subseteq \{1,\ldots,m\}$ and a set $R \subseteq \{1,\ldots,m\}$, queries $y_{Q,R}$, $y_{Q\triangle\{i_1\},R}$, $y_{Q,R\triangle\{i_2\}}$, and $y_{Q\triangle\{i_1\},R\triangle\{i_2\}}$, and reconstructs x_{i_1,i_2} as the exclusive-or of these four bits. If none of these bits is corrupted, then this is exactly how the client reconstructs x_{i_1,i_2} in Protocol 6, thus the reconstruction is correct. Assume that a fraction of at most δ of y is corrupted. In the worse scenario, the corrupted locations are chosen by an adversary which knows the pair (i_1, i_2) describing the index to be decoded, but does not know which locations of y will be queried. This adversary tries to reduce the probability of the correct decoding of x_{i_1,i_2}. That is, it tries to maximize the probability that the client reads a corrupted location. The probability that each one of the 4 locations is corrupted is at most δ, which, by the union bound, implies that the probability that the

Servers' Input: a database $x \in \{0,1\}^n$.

Client's Input: an index $i \in \{1,\ldots,n\}$.

1. Let $m = \lceil \sqrt{n}\, \rceil$, $i_1 = \lceil i/m \rceil$, and $i_2 = ((i-1) \bmod m) + 1$.
2. The client selects a random set $Q^1 \subseteq \{1,\ldots,m\}$ with uniform distribution, computes $Q^3 = Q^1 \vartriangle \{i_1\}$, and defines $Q^2 = Q^1$ and $Q^4 = Q^3$.
3. The client selects a random set $R^1 \subseteq \{1,\ldots,m\}$ with uniform distribution, computes $R^2 = R^1 \vartriangle \{i_2\}$, and defines $R^3 = R^1$ and $R^4 = R^2$.
4. The client sends Q^j and R^j to $\mathcal{S}_j$ for $j \in \{1,2,3,4\}$.
5. Server $\mathcal{S}_j$ computes
$$b^j \stackrel{\text{def}}{=} \bigoplus_{t \in Q^j, \ell \in R^j} x_{t,\ell},$$
and sends b^j to the client.
6. The client outputs $b^1 \oplus b^2 \oplus b^3 \oplus b^4$.

Protocol 6: A four server protocol with query length $2\sqrt{n}$ and answer length 1.

client reads a corrupted location is at most 4δ. Thus, now matter how the adversary chooses the corrupted locations, with probability at least $1 - 4\delta$, the decoding of x_{i_1,i_2} is correct. That is, this is indeed a locally decodable codes.

3.4.2 Construction of Information-Theoretic Multi-Party Private Protocols

Another interesting application of information-theoretic PIR, which demonstrates its universality, is for constructing information-theoretic multi-party private protocols for every function. Assume that k parties, each one holding an n-bit input, want to compute a function f of their inputs. A protocol for f is *t-private* if, after the end of the protocol, each coalition of at most t parties, seeing their inputs and the messages they have gotten during the protocol, do not get any information not implied by their inputs and the output of f. For example, a protocol is 1-private for an election if each party does not learn any information not implied by its vote and the output of the election. In [11, 17] it was shown that every function can be computed $(k-1)/2$-privately in the information-theoretic model; however for most functions their protocols are not efficient. Ishai and Kushilevitz [35] show that, for a constant k, there is an efficient information-theoretic PIR protocol if and only if every function has an *efficient* information-theoretic ck-private protocol for some constant $c < 1/2$.

4 Computational Private Information Retrieval

In this section we describe a computational private information retrieval protocol. For such construction, hardness assumptions are needed. In the construction we present, the hardness assumptions are taken from number theory. We first describe some background from number theory and then describe the PIR protocol.

4.1 Background from Number Theory: Diffie & Hellman (DH) Triplets

Cryptography is largely based on hardness of computation, that is, cryptographic protocols use one-way functions – functions that are easy to compute, however they are hard to invert (based on some hardness assumption). Many such hardness assumptions are based on number theoretic constructions, e.g., assuming that it is hard to factor large numbers or assuming that it is hard to compute the discrete logarithm. Furthermore, in many cryptographic constructions (including the computational PIR protocols we describe) additional structure from the one-way function is needed – it should also contain a "trapdoor". A trapdoor one-way function is a function that is easy to compute and hard to invert, however there is an additional key that enables efficient invention of the function.

We next describe the discrete logarithm function which is believed to be a one-way function, and a construction of Diffie and Hellman [24] that adds a certain type of trapdoor to this function. Let p be a "big" prime. The group $\mathbb{Z}_p^*$ is composed of the set $\{1, \ldots, p-1\}$ together with multiplication operation which is multiplication modulo p. A generator of $\mathbb{Z}_p^*$ is an element g such that for every $y \in \mathbb{Z}_p^*$ there exists $x \in \mathbb{Z}_p^*$ satisfying $y \equiv g^x \pmod p$. In other words, g is a generator if the $\{g^1 \bmod p, g^2 \bmod p, \ldots, g^{p-1} \bmod p\}$ is the set $\{1, 2, \ldots, p-1\}$. For every prime p, such generator exists.

The Discrete Logarithm Problem. Given y find $x \in \mathbb{Z}_p^*$ such that $y \equiv g^x \pmod p$.

This problem is believed to be hard. That is, the function $g^x \pmod p$ is easy to compute and believed to be hard to invert.

A DH triplet is a triplet $A, B, C \in \mathbb{Z}_p^*$ such that $A \equiv g^a \pmod p$, $B \equiv g^b \pmod p$, and $C \equiv g^{ab} \pmod p$ for some $a, b \in \mathbb{Z}_p^*$.[3] Otherwise, the triplet A, B, C is called a non-DH triplet.

The Decisional Diffie & Hellman (DDH) Problem. Given $A, B, C \in \mathbb{Z}_p^*$, determine if they are a DH triplet or a non-DH triplet.

This problem is also believed to be hard. Clearly, if the discrete logarithm problem is easy, then the DDH problem is easy (compute a – the discrete logarithm of A and b – the discrete logarithm of B, and check if $g^{ab} \equiv C \pmod p$).[4]

Given a such that $A \equiv g^a \pmod p$, it is easy to determine if A, B, C is a DH triplet. That is, A, B, C is a DH triplet if and only if $B^a \equiv C \pmod p$ (since $B^a \equiv (g^b)^a \equiv g^{ab} \pmod p$). Thus, a is a "trapdoor" that enables to efficiently solve the DDH problem.

The PIR protocol we describe uses the following multiplication properties of DH and non-DH triplets.

Claim 4.1 *Let A, B_1, C_1 be DH triplet, A, B_2, C_2 be triplet, $B_3 = (B_1 \cdot B_2) \bmod p$, and $C_3 = (C_1 \cdot C_2) \bmod p$. The triplet A, B_2, C_2 is a DH triplet if and only if A, B_3, C_3 is a DH triplet.*

Proof: Let a, b_1, b_2, c_1, c_2 be the discrete logarithms of A, B_1, B_2, C_1, C_2 respectively. Then, $C_1 = g^{ab_1} \pmod p$ since A, B_1, C_1 is a DH triplet. Furthermore,

$$B_3 \equiv g^{b_1} g^{b_2} \equiv g^{b_1+b_2} \pmod p.$$

Now, A, B_2, C_2 is a DH triplet if and only if $C_2 \equiv g^{ab_2} \pmod p$. This implies that A, B_2, C_2 is a DH triplet if and only if

$$C_3 \equiv g^{ab_1} g^{ab_2} \equiv g^{a(b_1+b_2)} \pmod p$$

[3] For security reasons, a and b should be even. We will ignore this issue to simplify presentation.

[4] The reverse direction is not known. That is, consistent with our current knowledge, it is possible that the DDH problem is easy while the discrete logarithm is hard. However, all known attacks on the DDH problem try to solve the discrete logarithm problem.

if and only if $(B_3)^a \equiv (g^{b_1+b_2})^a \equiv C \pmod p$ if and only if A, B_3, C_3 is a DH triplet. $\square$

The multiplication of triplets provides the same properties we needed from the exclusive-or operator when we identify 0 with a DH triplet and 1 with a non-DH triplet. On one hand, multiplying two DH triplets is the same as computing the exclusive-or of two zeros, whose answer is 0. On the other hand, multiplying a DH triplet with a non-DH triplet is the same as computing the exclusive-or of a zero and a one, whose answer is 1.[5]

In the above description we assume that p is a "big" prime. This will ensure that checking if a triplet is a DH triplet modulo p (and, in particular, computing the discrete logarithm modulo p) is an infeasible computation. Based on the best attacks known to date, it is common to require that p is approximately 1024-bit long.

4.2 A Computational Protocol with Short Communication

We present a 1-server computational protocol whose communication complexity is better than n^ϵ for every $\epsilon > 0$. The construction we present is a variant of [45] of the PIR protocol of Kushilevitz and Ostrovsky [43]. As in Section 3.1 we present the protocol in stages. In the first stage we present a protocol with high query length and low answer length and in the second stage we apply the balancing technique and get a 1-server protocol with $O(\sqrt{n}\,)$ communication. In the 1-server setting, we also present a third stage, where we add the idea of recursion and get the more efficient protocol.

4.2.1 First Stage

In Protocol 7 we present a 1-server protocol where the communication of the client is $O(n)$ elements in $\mathbb{Z}_p^*$ and the answer of the server is one element in $\mathbb{Z}_p^*$. In this protocol, the client first "encrypts" the index i using n triplets, where the ith triplet is a non-DH triplet and all other triplets are DH triplets. The client sends this encryption of i to the server. The server responses in a similar way as in the protocol of Section 3.1 (with the above analogy of computing exclusive-OR and multiplying triplets). That is, the server multiplies the triplets corresponding to the indices where $x_j = 1$. On one hand, if $x_i = 0$, then the ith triplet does not participate in the multiplication, thus, the answer is a multiplication of DH triplets, which by Claim 4.1 is a DH triplet. On the other hand, if $x_i = 1$, then the ith triplet participates in the multiplication, thus, the answer is a multiplication of DH triplets and *one* non-DH triplet, which by Claim 4.1 is a non-DH triplet. Therefore, $x_i = 0$ if and only if the answer is a DH triplet, and the protocol is correct. By the assumption that the DDH problem is hard, it is infeasible to distinguish between queries for different indices, thus, the protocol is private.

4.2.2 Second Stage

In Protocol 7 the communication from the server to the client is $2n + 2$ elements in $\mathbb{Z}_p^*$ and the answer of the server is two elements. Applying the same balancing technique as in Section 3.1 we get a 1-server protocol whose communication is $O(\sqrt{n}\,)$ elements in $\mathbb{Z}_p^*$ (an element $\mathbb{Z}_p^*$ is a $\log p$-bit[6] string, where $\log p$ is a security parameter – say $\log p = 1024$).

More generally, we can get a tradeoff between the query length and the answer length. We think of the database as an array with r rows and c columns, where $rc = n$. The client is interested in a bit $i \in \{1, \ldots, n\}$

[5]However, multiplying two non-DH triplets is not the same as computing the exclusive-or of two ones (as with high probability the result is non-DH triplet). This will not cause problems in the PIR protocol we describe.

[6] In this work, as common in theoretical computer science papers, $\log r$ should be read as $\log_2 r$.

Servers' Input: a database $x \in \{0,1\}^n$.

Client's Input: an index $i \in \{1, \ldots, n\}$.

1. The client chooses a large prime number p and a generator g of $\mathbb{Z}_p^*$.
2. The client prepares n triplets such that only the ith triplet is a non-DH triplet:
   - The client chooses at random $a \in \mathbb{Z}_p^*$, $b_1, \ldots, b_n \in \mathbb{Z}_p^*$, and $c_i \in \mathbb{Z}_p^*$ such that $c_i \not\equiv ab_i \pmod{p}$.
   - The client calculates $A = g^a \bmod p$, $B_j = g^{b_j} \bmod p$ for $j = 1, \ldots, n$, $C_j = g^{ab_j} \bmod p$ for $j = 1, \ldots, i-1, i+1, n$, and $C_i = g^{c_i} \bmod p$.
3. The clients sends $p, g, B_1, \ldots, B_n, C_1, \ldots, C_n$ to the server.
4. The server computes $B = \prod_{\{j:x_j=1\}} B_j \bmod p$ and $C = \prod_{\{j:x_j=1\}} C_j \bmod p$ and sends B, C to the client.
5. If A, B, C is a DH triplet (that is, if $B^a \equiv C \pmod{p}$), then the client outputs $x_i = 0$ otherwise the client outputs $x_i = 1$.

Protocol 7: A 1-server protocol with query length $O(n)$ and answer length $O(1)$.

which is represented as i_1, i_2 where $i_1 = \lceil i/c \rceil$ and $i_2 = ((i-1) \bmod c) + 1$. The client executes the PIR protocol with index i_2 where the database length is c. For each of the r rows, the server answers the query treating the row as the database. The server takes the answer corresponding to the i_1th row, and answers according to this answer. In the resulting scheme, the communication from the client to the server is $2c+2$ elements in $\mathbb{Z}_p^*$ and the answer is r elements.

4.2.3 Third Stage

In the protocol described in the second stage, although the answer is long, the client needs only one element from it. However, the server is not allowed to learn which element the client wants. This is exactly the PIR scenario, however with a shorter database. Thus, instead of sending the answer to the client, the client and the server engage in a new PIR protocol with i_1 as the client's index and the server's first answer as the database.

This idea can be carried out recursively, and achieve a PIR protocol with communication smaller than $O(n^\epsilon)$ for every constant $\epsilon > 0$.

Example 4.2 *We exemplify this construction by constructing a protocol with communication $O(n^{1/5} \log p)$. Let p be a large enough prime such that the DDH problem in $\mathbb{Z}_p^*$ is hard. The client and the server use the PIR protocol described in the second stage with $c = n^{1/5} \log p$ and $r = n/c = n^{4/5}/\log p$, where the server computes its answer and keeps it without sending it. The answer of the server is composed of $n^{4/5}/\log p$ blocks, where each block is an $O(\log p)$-bit string. The client and the server apply an additional PIR protocol where in this step the database is the answer. In this step, they use the same $c = n^{1/5} \log p$, however $r = (n^{4/5}/\log p)/c = n^{3/5}/\log^2 p$. This step is repeated, where after 4 steps the answer is* *composed of $n^{1/5}/\log^4 p$ blocks, each block is an $O(\log^4 p)$-bit string. The client sends the answer to the client. Thus, the length of the answer is $n^{1/5}$ and the total communication is $O(n^{1/5}\log p)$.*

Note that the index that the client needs in the second stage (and in all further steps) only depends on the initial index it wants (e.g., $i_1 = \lceil i/(n^{4/5}/\log p)\rceil$). That is, the client can prepare all its queries and send them to the server as one message and the resulting protocol has one round.

By correctly choosing the parameters for the recursion, we get an efficient PIR protocol. Let $\ell = \sqrt{\frac{\log n}{\log\log p}}$. In all recursion steps, we choose $c = n^{1/\ell}$. That is, in the jth recursion step, the database contains $n^{1-(j-1)/\ell}$ blocks, each block is a $(\log p)^{j-1}$-bit string, $r = n^{j/\ell}$, and the answer (which is the database in the next step) contains $n^{1-j/\ell}$ blocks, each block is a $(\log p)^j$-bit string. The query length in each PIR execution is

$$O(c) = O(n^{1/\ell}) = O\left(n^{1/\sqrt{\frac{\log n}{\log\log p}}}\right) = O\left(2^{\log n/\sqrt{\frac{\log n}{\log\log p}}}\right) = O\left(2^{\sqrt{\log n \log\log p}}\right).$$

After ℓ steps, the answer is composed of one block, containing a string whose length is

$$(\log p)^\ell = (\log p)^{\sqrt{\frac{\log n}{\log\log p}}} = 2^{\log\log p\sqrt{\frac{\log n}{\log\log p}}} = 2^{\sqrt{\log n \log\log p}}.$$

The server sends this block to the client. Thus, the total communication complexity of the recursive protocol is

$$O\left(\ell \cdot 2^{\sqrt{\log n \log\log p}}\right) < O\left(\log n \cdot 2^{\sqrt{\log n \log\log p}}\right).$$

When $\log p \ll n$, the above expression is asymptotically smaller than n^ϵ for every constant $\epsilon > 0$.

4.3 Historical Notes and More Efficient Computational PIR Protocols

The first computational PIR protocol was a multi-server PIR protocol of Chor and Gilboa [18]. The first efficient 1-server computational PIR was presented by Kushilevitz and Ostrovsky [43]. Their protocol is similar to the protocol presented above; it was based on the hardness of recognizing quadratic residues modulo a composite. Replacing quadratic residuosity by other constructions (such as DH triplets) was suggested in [45, 52, 16]. The first 1-server PIR with polylogarithmic communication complexity was presented by Cachin, Micali, and Stadler [14]. A protocol with $O(\log^2 n)$ communication complexity was constructed by Lipmaa [44] (the ideas of this construction are discussed below). Finally, a PIR protocol with $O(\log^2 n)$ communication complexity that is more efficient than the protocol of [44] for retrieving large blocks was presented by Gentry and Ramzan [28].

In the rest of the section we discuss how to improve the PIR protocol we presented in Section 4.2. The protocol we presented uses DH triplets. In this protocol, we presented in the first stage a basic protocol, which uses DH triplets, and then showed how to improve it using balancing and recursion. The basic protocol can be based on other hardness assumptions, yielding, after applying balancing and recursion, the same complexity or even PIR protocols with better communication complexity.

We next describe a more general way to view the basic protocol; this view enables to replace the DH triplets with different primitives. It is useful to view a DH triplet as a randomized encryption of 0 and a non-DH triplet as a randomized encryption of 1. Thus, the query of the client with index i is a vector of n randomized encryptions, where $n-1$ of them are encryptions of 0 and the ith encryption is an encryption of 1. The server, with database x, "multiplies" the encryptions corresponding to indices where the database is 1 (that is, it multiplies the encryptions in locations $\{j : x_j = 1\}$). The requirement from the "multiplication" is:

- If the server multiplies two encryptions of 0, then the result is an encryption of 0.
- If the server multiplies an encryptions of 0 and an encryption of 1, then the result is an encryption of 1.

For DH triplets, we proved these properties in Claim 4.1. We can use other randomized encryption schemes that have these properties instead of DH triplets. Specifically, the encryption scheme of [34], based on quadratic-residuosity, was used in the paper of Kushilevitz and Ostrovsky [43] that presented the first efficient 1-server PIR protocol. Other candidates, suggested by [45, 52, 16] include, for example, encryption schemes based on the prime-residuosity assumption and encryption schemes based on lattice problems.

The most impressive case was constructed by Lipmaa [44]; he uses the Damgård-Jurik encryption scheme [21] and obtains a PIR protocol with communication $O(\log^2 n \log p)$. In this protocol there are $\log n$ recursion steps where in each step $c = 2$, that is, the number of blocks in the database is halved. Using stronger homomorphic properties of the Damgård-Jurik encryption scheme and the fact the length of an encryption of a message grows slowly compared to the length of the message, Lipmaa constructs the PIR protocol with communication $O(\log^2 n \log p)$.

4.4 Application of Computational PIR protocols: Filtering of Encrypted Data

To demonstrate the usefulness of PIR protocols, we briefly describe an application that uses these protocols. Suppose that we are in the public key encryption scenario: A client generates and a public encryption key and a private decryption key. The client published the public key and stores its private key securely. Now, every server can encrypt information and send it to the client such that only the client can decrypt it. Here we consider a twist of this scenario, where the server has a large database and the client is only interested in information relevant to it. The client knows in advance, when generating the keys, what type of information it wants (for example, the client might be interested only in documents containing some key words), however, the client does not want to reveal this information. The goal is to produce a public key such that the encrypted message is much smaller than the entire database (assuming that most information in the database is not relevant to the client). Note that the client does not know the size of the database in advance. This scenario was addressed by Ostrovsky and Skeith [49], who show how to solve this problem in the streaming model using PIR protocols and other techniques. Public encryption systems that allow PIR queries are described in [12]. For more relevant papers see, e.g., the references in [49, 12].

We describe a solution, discussed in [48], for the simple case where each database held by a server is composed of n blocks, the client knows n in advance, and knows that it is interested in the ith block of the database. The construction uses a PIR protocol and a public-key encryption scheme. The public key of the client is composed of a public key K of a public-key encryption scheme together with a PIR query q for the index i. Note that by the privacy of the PIR protocol, servers do not know which block the client is interested in. A server, which wants to encrypt the relevant part of the database (without even knowing which parts are relevant), first computes the answer a of the server in the PIR protocol with the query q and the database that it holds, encrypts this answer a using K, and sends the encrypted answer to the client. Since the length of the answer in the PIR protocol is much shorter than the length of the original database, the length of the message sent by the client is short. The client decrypts the answer and reconstructs the ith block using the PIR reconstruction function.

5 Protecting the Server: Symmetric Private Information Retrieval

In the definition of PIR protocols we only protect the privacy of the client. While each server is not allowed to learn information about the bit that the client is interested in, the client can learn many bits of the database. This might be problematic in many scenarios. For example, if the server wants to charge the client for each bit it retrieves, then the client gets extra information for free. In this section we consider symmetric private information retrieval, abbreviated SPIR, where the server learns no information and the client only learns the bit that it wants. As discussed in the introduction, such protocols can be thought of as oblivious transfer protocols with sub-linear communication.

Remark 5.1 1-out-of-2 Oblivious Transfer (OT) protocols are SPIR protocols with $n = 2$; in such protocols a server has two bits and a clients wants to retrieve one of them such that the server does not know which bit the clients retrieves and the client does not learn any additional information. Oblivious transfer, specifically 1-out-of-2 OT, is an important building box in cryptography: 1-out-of-2 OT protocols can be used to construct efficient *secure* protocols for *any* function that can be computed by an efficient algorithm [33, 42]. For example, we will show in Section 5.2 that 1-out-of-2 OT protocols are used to construct SPIR protocols with big databases.

In this section we will consider communication efficient computational SPIR protocols which were studied by, e.g., [43, 52, 47, 16]. We will describe two computational SPIR protocols. First, we will show that a variant of the PIR protocol presented in Section 4.2 is a SPIR protocol. Then, we will show a general transformation from PIR protocols to SPIR protocols. It is worth while to mention an important difference between PIR and SPIR. There is a naive PIR protocol with communication complexity n; the challenge in PIR protocols is to improve its communication complexity. However, sending the entire database violates the server's privacy and it is already a challenge to construct a SPIR protocol with polynomial communication complexity.

We start with a definition of SPIR protocols which is a PIR protocol with an additional server's privacy requirement. Informally, the server's privacy requirement states that if the client queries an index i, then it cannot distinguish between two databases which have the same value of x_i, that is, the client's view – which includes the index i, its random string, and the answer of the server – is indistinguishable given the two databases.[7] The formal definition of the server's privacy is rather technical; the rest of the section can be understood without it.

Definition 5.2 (SPIR) *A computational 1-server* symmetric *private information retrieval (SPIR) protocol $\mathcal{P} = (\mathcal{Q}, \mathcal{A}, \mathcal{C})$, where here the answer algorithm $\mathcal{A}$ is a randomized algorithm, is a protocol satisfying the correctness and client's privacy requirements of computational 1-server PIR protocols as defined in Definition 2.1 and Definition 2.4, and, in addition, satisfying the following requirement:*

Server's Privacy. *The client has no knowledge on the database except for x_i – the value of the database the server holds in the index the client wants to retrieve. Formally, for every sequence of indices $\{i_n\}_{n=1}^{\infty}$, where $1 \leq i_n \leq n$, for every two sequences of databases $\{x^n\}_{n=1}^{\infty}$ and $\{y^n\}_{n=1}^{\infty}$, where $x^n, y^n \in \{0,1\}^n$ and $x^n_{i_n} = y^n_{i_n}$, and for every sequence of random strings of the client $\{r_n\}_{n=1}^{\infty}$, the following ensembles of views of the client are indistinguishable in polynomial-time:*

$$\{(i_n, r_n, \mathcal{A}(1^n, \mathcal{Q}(1^n, i_n, r_n), x^n))\}_{n=1}^{\infty} \quad \textit{and} \quad \{(i_n, r_n, \mathcal{A}(1^n, \mathcal{Q}(1^n, i_n, r_n), y^n))\}_{n=1}^{\infty}.$$

[7] A better definition uses the ideal world vs. the real world paradigm. In this primer we will not elaborate on this paradigm and its advantages.

5.1 A Construction of a SPIR Protocol

In this section we will show that a variant of the PIR protocol presented in Section 4.2 is a SPIR protocol. In Protocol 8 we will describe a SPIR protocol of [13], where we use the DDH problem instead of the quadratic residuosity problem. It is a variant of Protocol 7 and has large query length and small answer length. We will then show that the balancing and recursion techniques presented in Section 4.2.3 preserves the server's privacy requirement. Thus, we get an efficient SPIR protocol.

Servers' Input: a database $x \in \{0,1\}^n$.

Client's Input: an index $i \in \{1,\dots,n\}$.

1. The client chooses a large prime number p and a generator g of $\mathbb{Z}_p^*$.
2. The client prepares n triplets such that the ith triplet is a non-DH triplet and the other triplets are DH triplets:
   - The client chooses at random $a \in \mathbb{Z}_p^*$, $b_0,\dots,b_1 \in \mathbb{Z}_p^*$, and $c_i \in \mathbb{Z}_p^*$ such that $c_i \not\equiv ab_i \pmod{p}$.
   - The client calculates $A = g^a \bmod p$, $B_j = g^{b_j} \bmod p$ for $j = 0,1$, $C_j = g^{ab_j} \bmod p$ for $j \neq i$, and $C_i = g^{c_i} \bmod p$.
3. The clients sends $p, g, A, B_1, \dots, B_n, C_1, \dots, C_n$ to the server.
4. The server picks a random element $r \in \mathbb{Z}_p^*$, computes $B = (g^r \cdot \prod_{\{j:x_j=1\}} B_j) \bmod p$ and $C = (A^r \cdot \prod_{\{j:x_j=1\}} C_j) \bmod p$, and sends B, C to the client.
5. If A, B, C is a DH triplet (that is, if $B^a \equiv C \pmod{p}$), then the client outputs $x_i = 0$ otherwise the client outputs $x_i = 1$.

Protocol 8: A SPIR protocol with $O(n \log p)$ query length and $O(\log p)$ answer length.

The main difference in Protocol 8 compared to Protocol 7 is that the client should not know which B_j's where multiplied to obtain B. This information is not hidden in the original protocol; for example, if the database is the all zero string then $B = 1$. This is solved by the server choosing a random DH triplet and multiplying the answer by this triplet. To enable the server to compute a random DH triplet, the client sends A to the server, the server chooses a random r and computes the DH triplet

$$A, g^r \bmod p, A^r \bmod p. \tag{5}$$

We next analyze the properties of Protocol 8. As in Protocol 7,

$$A, \left(\prod_{\{j:x_j=1\}} B_j \right) \bmod p, \left(\prod_{\{j:x_j=1\}} C_j \right) \bmod p \tag{6}$$

is a DH triplet if and only if $x_i = 0$. The server multiplies the DH triplet it computed in (5) with the triplet in (6). The result of this multiplication are the values A, B, C in Step (5) of Protocol 8. Thus, by Claim 4.1, these values A, B, C are a DH triplet if and only if $x_i = 0$. This implies the correctness of the protocol.

Next, we argue the privacy of the protocol. The client's privacy follows from the same arguments as in Protocol 7, that is, the server cannot distinguish which of the n triplets are the non-DH triplet. The server protects its privacy by multiplying its answer by a random DH triplet. That is, the value B that it answers is uniformly distributed in $\mathbb{Z}_p^*$ independent of the database x. Furthermore, if $x_i = 0$ then $C \equiv B^a \pmod p$, and if $x_i = 1$ then $C \equiv (g^{c_i - ab_i} \cdot B^a) \pmod p$. Thus, in both cases, the client can compute C from the value of x_i and the information that it has, thus, C gives no additional knowledge. To conclude, the client gets no knowledge except for x_i.

Finally, We discuss the communication complexity of the protocol. The clients sends $O(n)$ elements in $\mathbb{Z}_p^*$ to the server while the answer of the server is 2 elements in $\mathbb{Z}_p^*$. Thus, the total communication is $O(n \log p)$ bits. In Section 5.2 we will use this protocol with $n = 2$; in this case the protocol is efficient. However, for large values of n, the communication complexity is too high.

We next show that the balancing and recursion techniques used in Section 4.2.3 preserves the server's privacy requirement. First note that the balancing technique by its self, as described in Section 4.2.2, does not preserve privacy as the client can reconstruct the entire i_2th column of the database. However, if we use the SPIR protocol described in Protocol 8 as the basic protocol, the answer of the server gives no knowledge to the client except for this column. If we apply the recursion technique as in Section 4.2.3, then in the second step of the recursion the database is the answer from the first step viewed as a matrix, which has fewer rows compared to the matrix representing the original database. Thus, the answer of the second step only reveals knowledge on fewer elements of the original database. The server only sends the answer for the last step of the recursion, which is one block that only reveals information on x_i. To conclude, if we start with the SPIR protocol described in Protocol 8 and apply the balancing and recursion techniques we get a SPIR protocol with communication $2^{O(\sqrt{\log n \log\log p})}$. If we start with the protocol of [44] and use $O(\log n)$ steps of recursion, we get a SPIR protocol with communication complexity $O(\log^2 n \log p)$.

In the sequel we will need a SPIR protocol where the server holds blocks instead of bits, and the client wants to retrieve a block. We use the same construction described in Protocol 5 to get an efficient SPIR protocol with blocks.

5.2 A Transformation from PIR Protocols to SPIR Protocols

In this section we describe a transformation of Naor and Pinkas [47] from PIR protocols to SPIR protocols. This transformation uses an encryption scheme, a 1-out-of-2 OT protocol, and a PIR protocol to obtain a communication efficient SPIR protocol. The purpose of describing this transformation, in addition to the construction in Section 5.1, is two-fold. First, we present an alternative construction of SPIR protocols that can be applied to PIR protocols that do not protect the server's privacy (e.g., the protocols of [14, 28]). Second, we demonstrate how 1-out-of-2 OT protocols (that is, SPIR protocols with $n = 2$) are used in cryptography to construct more complex protocols.[8]

To describe the idea of the transformation, first assume that we have a PIR protocol $\mathcal{P}_{\mathrm{PIR}}$ for retrieving large blocks and a SPIR protocol $\mathcal{P}_{\mathrm{SPIR}}$ for retrieving small blocks. We want to construct a SPIR protocol for retrieving large blocks by executing $\mathcal{P}_{\mathrm{PIR}}$ once and executing $\mathcal{P}_{\mathrm{SPIR}}$ once. The construction is described in Protocol 9. The idea of the construction is that the server encrypts each block with an independent key; the client and the server execute a PIR protocol on the encrypted database and a SPIR protocol where the database is the n keys and the client learns only the ith key. The client can learn a lot of information in the

[8] The main motivation in [47] was different; their aim was to construct a 1-out-of-n OT protocol where the computation of the server and client are low. This is true in their construction if we allow $O(n)$ communication. However, if we want a communication efficient SPIR protocol, the computation is relatively high.

Servers' Input: a database $x \in (\{0,1\}^b)^n$.

Client's Input: an index $i \in \{1,\ldots,n\}$.

1. The server chooses n independent keys of an encryption scheme and encrypts the ith block with the ith key for every $i = 1,\ldots,n$.
2. The client and server execute $\mathcal{P}_{\mathrm{PIR}}$ where the database is the encrypted blocks and the client's index is i.
3. The client and server execute $\mathcal{P}_{\mathrm{SPIR}}$, where the database is the n keys and the client's index is i.
4. The client decrypts the block it retrieved in the PIR execution with the key it got in the SPIR execution.

Protocol 9: A simple transformation from PIR to SPIR.

PIR execution about the encrypted blocks, however, since it does not have the appropriate keys, it cannot learn any information on other blocks of the database.

We want to use a similar construction, however by using $\log n$ executions of a 1-out-of-2 OT protocol instead of the SPIR protocol. The idea is to have $2\log n$ keys and encrypt each bit of the database by a subset of size $\log n$ of the keys. The client retrieves a subset of the keys using the OT executions such that it can only decrypt the ith bit of the database. Specifically, for every index j we consider its representation as a $\log n$-bit string $(j_1,\ldots,j_{\log n})$; the keys that are used to encrypt x_j are $K_{1,j_1}, K_{2,j_2},\ldots,K_{\log n, j_{\log n}}$.

The details of the transformation are explained in Protocol 10. The transformation uses a (semantically-secure) symmetric encryption scheme $\mathcal{E},\mathcal{D}$. We denote the encryption of a message m with a key K by $\mathcal{E}_K(m)$, and the decryption of c with a key K by $\mathcal{D}_K(c)$. The correctness of the scheme implies that if we encrypt a message m using a key K and decrypt the result with the same key, then we recover m, that is, $\mathcal{D}_K(\mathcal{E}_K(m)) = m$. The semantic security states that the encryption of m gives no information on m to anybody not holding the key K.

We next analyze the properties of the protocol. The correctness is strait-forward from the description of the protocol, that is, the client decrypts y_i with the keys that encrypted x_i, thus, the client obtains x_i. The client's privacy follows from the fact that the server only gets messages from the OT protocols and the PIR protocol; as the server gets no knowledge from each execution, it does not get knowledge from the entire protocol. The server's privacy follows from the fact that the client gets exactly $\log n$ keys from the OT executions. Therefore, for every index $j \neq i$ it misses at least one key that was used to encrypt the jth block, that is, for every $j \neq i$, there is some ℓ such that $i_\ell \neq j_\ell$ and the client misses K_{ℓ,j_ℓ}, thus cannot learn any information on x_j. This implies that the client does not get any information on any block except for the ith block no matter how much information it gets on the encrypted blocks from the PIR execution.

Next we analyze the communication complexity of the protocol. The protocol is composed of $\log n$ executions of the 1-out-of-2 OT protocol and one execution of the PIR protocol. If we use the SPIR protocol described in Protocol 8 as the 1-out-of-2 OT protocol, then the communication in the OT protocol is $O(\log p)$, thus, the communication in the SPIR protocol is dominated by the communication in the PIR protocol. In other words, we get a SPIR protocol whose communication is basically the same as the communication of the PIR protocol we use.

Servers' Input: a database $x \in \{0,1\}^n$.

Client's Input: an index $i \in \{1,\ldots,n\}$, where $i = (i_1,\ldots,i_{\log n})$.

1. The server chooses at random $2\log n$ keys $K_{1,0}, K_{1,1}, \ldots K_{\log n,0}, K_{\log n,1}$ for an encryption scheme. For every $j \in \{1,\ldots,n\}$ let $j = (j_1,\ldots,j_{\log n})$ be its representation as a $\log n$-bit string. The server computes
$$y_j = \mathcal{E}_{K_{1,j_1}}(\mathcal{E}_{K_{2,j_2}}(\ldots\mathcal{E}_{K_{\log n, j_{\log n}}}(x_j)\ldots)).$$
2. For every $\ell \in \{1,\ldots,\log n\}$ the client and the server execute a 1-out-of 2 OT protocol, where the input of the client is $i_\ell + 1$ and the input of the server is the pair of keys $K_{\ell,0}, K_{\ell,1}$.
3. The client and the server execute a PIR protocol, where the input of the client is i and the input of the server are the n blocks $y_1,\ldots,y_n$.
4. The clients reconstructs y_i from the PIR protocol and computes x_i by decrypting y_i with the keys $K_{\log n, i_{\log n}}, \ldots, K_{1,i_1}$ that it obtained in the executions of the OT protocol.

Protocol 10: A SPIR protocol from a PIR protocol.

Acknowledgment

I would like to thank Eyal Kushilevitz, Yuval Ishai, Tal Malkin, Yoav Stahl, and Enav Weinreb for many discussions on private information retrieval. The write-up in Section 3.2 is based on the write-up in [8] co-authored with Eyal Kushilevitz, Yuval Ishai, and Jean-François Raymond. Finally, I would like to thank Kobbi Nissim for inviting me to write this primer and encouraging me to complete it.

References

[1] C. Aguilar Melchor and Y. Deswarte. Single-database private information retrieval schemes: Overview, performance study, and usage with statistical databases. In J. Domingo-Ferrer and L. Franconi, editors, *Privacy in Statistical Databases*, volume 4302 of *Lecture Notes in Computer Science*, pages 257–265. Springer-Verlag, 2006.

[2] A. Ambainis. Upper bound on the communication complexity of private information retrieval. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. of the 24th International Colloquium on Automata, Languages and Programming*, volume 1256 of *Lecture Notes in Computer Science*, pages 401–407. Springer-Verlag, 1997.

[3] O. Barkol, Y. Ishai, and E. Weinreb. On locally decodable codes, self-correctable codes, and t-private PIR. In M. Charikar, K. Jansen, O. Reingold, and J. D. P. Rolim, editors, *RANDOM '07, 11th International Workshop on Randomization and Computation*, volume 4627 of *Lecture Notes in Computer Science*, pages 311–325. Springer-Verlag, 2007.

[4] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In C. Choffrut and T. Lengauer, editors, *STACS '90, 7th Symp. on Theoretical Aspects of Computer Science*, volume 415 of *Lecture Notes in Computer Science*, pages 37–48. Springer-Verlag, 1990.

[5] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Locally random reductions: Improvements and applications. *J. of Cryptology*, 10(1):17–36, 1997. Early version: Security with small communication overhead, CRYPTO '90, volume 537 of *Lecture Notes in Computer Science*, pages 62-76. Springer-Verlag, 1991.

[6] A. Beimel, Y. Ishai, and E. Kushilevitz. General constructions for information-theoretic private information retrieval. *J. of Computer and System Sciences*, 71(2):213–247, 2005.

[7] A. Beimel, Y. Ishai, E. Kushilevitz, and T. Malkin. One-way functions are essential for single-server private information retrieval. In *Proc. of the 31st ACM Symp. on the Theory of Computing*, pages 89–98, 1999.

[8] A. Beimel, Y. Ishai, E. Kushilevitz, and J. F. Raymond. Breaking the $O(n^{\frac{1}{2k-1}})$ barrier for information-theoretic private information retrieval. In *Proc. of the 43rd IEEE Symp. on Foundations of Computer Science*, pages 261–270, 2002.

[9] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *J. of Cryptology*, 17(2):125–151, 2004. Preliminary version: M. Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 56–74. Springer, 2000.

[10] A. Beimel and Y. Stahl. Robust information-theoretic private information retrieval. *J. of Cryptology*, 20(7):295–321, 2007.

[11] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proc. of the 20th ACM Symp. on the Theory of Computing*, pages 1–10, 1988.

[12] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith III. Public key encryption that allows PIR queries. In A. Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 50–67. Springer-Verlag, 2007.

[13] G. Brassard, C. Crépeau, and J.-M. Robert. All-or-nothing disclosure of secrets. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 234–238. Springer-Verlag, 1987.

[14] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer-Verlag, 1999.

[15] R. Canetti, Y. Ishai, R. Kumar, M. K. Reiter, R. Rubinfeld, and R. N. Wright. Selective private function evaluation with applications to private statistics. In *Proc. of the 20th ACM Symp. on Principles of Distributed Computing*, pages 293 – 304, 2001.

[16] Y.-C. Chang. Single database private information retrieval with logarithmic communication. In *Information Security and Privacy: 9th Australasian Conference, ACISP 2004*, volume 3108 of *Lecture Notes in Computer Science*, pages 50 – 61. Springer-Verlag, 2004.

[17] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. of the 20th ACM Symp. on the Theory of Computing*, pages 11–19, 1988.

[18] B. Chor and N. Gilboa. Computationally private information retrieval. In *Proc. of the 29th ACM Symp. on the Theory of Computing*, pages 304–313, 1997.

[19] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. Technical report, Department of Computer Science, Technion, 1997.

[20] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proc. of the 36th IEEE Symp. on Foundations of Computer Science*, pages 41–51, 1995. Journal version: *J. of the ACM*, 45:965–981, 1998.

[21] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In K. Kim, editor, *Public Key Cryptography: 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer-Verlag, 2001.

[22] G. Di-Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for private information retrieval. *J. of Cryptology*, 14(1):37–74, 2001. Preliminary version in *Proc. of the 17th ACM Symp. on Principles of Distributed Computing*, pages 91–100, 1998.

[23] G. Di-Crescenzo, T. Malkin, and R. Ostrovsky. Single-database private information retrieval implies oblivious transfer. In *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 122–138. Springer-Verlag, 2000.

[24] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, 22(6):644–654, 1976.

[25] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *CACM*, 28(6):637–647, 1985.

[26] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright. Secure multiparty computation of approximations. *ACM Trans. Algorithms*, 2(3):435–472, 2006. Conference version in *Proc. of the 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 927–938. Springer-Verlag, 2001.

[27] W. Gasarch. A survey on private information retrieval. *Bulletin of the European Association for Theoretical Computer Science*, 82:72–107, 2004. Also: `www.cs.umd.edu/~Gasarch/pir/pir.html`.

[28] C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Proc. of the 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815. Springer-Verlag, 2005.

[29] Y. Gertner, S. Goldwasser, and T. Malkin. A random server model for private information retrieval. In M. Luby, J. Rolim, and M. Serna, editors, *RANDOM '98, 2nd International Workshop on Randomization and Approximation Techniques in Computer Science*, volume 1518 of *Lecture Notes in Computer Science*, pages 200–217. Springer-Verlag, 1998.

[30] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *J. of Computer and System Sciences*, 60(3):592–629, 2000. Conference version in *Proc. of the 30th ACM Symp. on the Theory of Computing*, pages 151–160, 1998.

[31] O. Goldreich. Short locally testable codes and proofs (survey). Technical Report TR05-014, Electronic Colloquium on Computational Complexity, `www.eccc.uni-trier.de/eccc/`, 2005.

[32] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. of the 19th ACM Symp. on the Theory of Computing*, pages 218–229, 1987.

[33] O. Goldreich and R. Vainish. How to solve any protocol problem—an efficiency improvement. In C. Pomerance, editor, *Advances in Cryptology – CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 73–86. Springer-Verlag, 1988.

[34] S. Goldwasser and S. Micali. Probabilistic encryption. *J. of Computer and System Sciences*, 28(21):270–299, 1984.

[35] Y. Ishai and E. Kushilevitz. On the hardness of information-theoretic multiparty computation. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 439 – 455. Springer-Verlag, 2004.

[36] Y. Ishai, E. Kushilevitz, and R. Ostrovsky. Sufficient conditions for collision-resistant hashing. In J. Kilian, editor, *Proc. of the Second Theory of Cryptography Conference – TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 445–456. Springer-Verlag, 2005.

[37] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Batch codes and their applications. In *Proc. of the 36th ACM Symp. on the Theory of Computing*, pages 262 – 271, 2004.

[38] T. Itoh. Efficient private information retrieval. *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, E82-A(1):11–20, 1999.

[39] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proc. of the 32nd ACM Symp. on the Theory of Computing*, pages 80–86, 2000.

[40] K. Kedlaya and S. Yekhanin. Locally decodable codes from nice subsets of finite fields and prime factors of Mersenne numbers. Technical Report TR07-040, Electronic Colloquium on Computational Complexity, `www.eccc.uni-trier.de/eccc/`, 2007.

[41] I. Kerenidis and R. de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *J. of Computer and System Sciences*, 69(3):395–420, 2004.

[42] J. Kilian. Basing cryptography on oblivious transfer. In *Proc. of the 20th ACM Symp. on the Theory of Computing*, pages 20–31, 1988.

[43] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proc. of the 38th IEEE Symp. on Foundations of Computer Science*, pages 364–373, 1997.

[44] H. Lipmaa. An oblivious transfer protocol with log-squared communication. In J. Zhou and J. Lopez, editors, *the 8th Information Security Conference (ISC'05)*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer-Verlag, 2005.

[45] E. Mann. Private access to distributed information. Master's thesis, Technion – Israel Institute of Technology, Haifa, 1998.

[46] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *Proc. of the 33th ACM Symp. on the Theory of Computing*, 2001. Long version: Communication complexity and secure function evaluation, *Cryptology ePrint Archive*, number 2001/076, `eprint.iacr.org/`, 2001.

[47] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. of the 31st ACM Symp. on the Theory of Computing*, pages 245–254, 1999.

[48] R. Ostrovsky and W. E. Skeith III. A survey of single-database private information retrieval: Techniques and applications. In T. Okamoto and X. Wang, editors, *Public Key Cryptography: 10th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2007*, volume 4450 of *Lecture Notes in Computer Science*, pages 393–411. Springer-Verlag, 2007.

[49] R. Ostrovsky and W. E. Skeith III. Private searching on streaming data. *J. of Cryptology*, 20(4):397–430, 2007.

[50] M. O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981. Available online in the Cryptology ePrint Archive, Report 2005/187, `eprint.iacr.org/2005/187`.

[51] R. Sion and B. Carbunar. On the computational practicality of private information retrieval. In *Proc. of the 14th ISOC Network and Distributed System Security Symposium*, 2007.

[52] J. P. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In K. Ohta and D. Pei, editors, *Advances in Cryptology – ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 357–371. Springer-Verlag, 1998.

[53] Y. Tauman Kalai and R. Raz. Succinct non-interactive zero-knowledge proofs with preprocessing for LOGSNP. In *Proc. of the 47th IEEE Symp. on Foundations of Computer Science*, pages 355–366, 2006.

[54] L. Trevisan. Some applications of coding theory in computational complexity. *Quaderni di Matematica*, 13:347–424, 2004. Also: ECCC TR04-043, `www.eccc.uni-trier.de/eccc`, 2004.

[55] S. Wehner and R. de Wolf. Improved lower bounds for locally decodable codes and private information retrieval. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Proc. of the 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 1424–1436, 2005.

[56] D. P. Woodruff and S. Yekhanin. A geometric approach to information-theoretic private information retrieval. *SIAM Journal on Computing*, 37(4):1046–1056, 2007.

[57] A. C. Yao. Protocols for secure computations. In *Proc. of the 23th IEEE Symp. on Foundations of Computer Science*, pages 160–164, 1982.

[58] S. Yekhanin. *Locally Decodable Codes and Private Information Retrieval Schemes*. PhD thesis, MIT, 2007.

[59] S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. In *Proc. of the 39th ACM Symp. on the Theory of Computing*, pages 266–274, 2007.