BOSTON UNIVERSITY
GRADUATE SCHOOL OF ARTS AND SCIENCES

Thesis

STUDY OF THE COMPUTATIONAL EFFICIENCY OF SINGLE SERVER PRIVATE INFORMATION RETRIEVAL

by

JEFFREY GUARENTE
B.S., Carnegie Mellon University, 2005

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science
2013

Approved by

First Reader

Sharon Goldberg, PhD
Assistant Professor of Computer Science

Second Reader

Leonid Reyzin, PhD
Professor of Computer Science

ACKNOWLEDGEMENTS

I would like to thank my advisors, Profs. Leo Reyzin and Sharon Goldberg, for their time, patience, and insights.

STUDY OF THE COMPUTATIONAL EFFICIENCY OF SINGLE SERVER PRIVATE INFORMATION RETRIEVAL

JEFFREY GUARENTE

ABSTRACT

Private Information Retrieval (PIR) is a protocol for a client to retrieve information from a server without revealing anything about which item (s)he retrieved. It has numerous applications but its impracticality due to extremely poor performance or awkward assumptions about its usage prevent its uptake. This work provides a background on the topic of PIR performance, frames the problem of finding efficient PIR as the problem of finding a code with a local decoding property, shows existing families of locally decodable codes are not suitable, and lists some requirements that codes must have to produce secure PIR.

Contents

List of Abbreviations & Acronyms

Σ	alphabet; i.e. set of possible message or codeword entries
A	PIR answer algorithm
BDD	binary decision diagram
C	code encoding algorithm
CRHFF	collision-resistant hash function family
D	code decoding algorithm
i	index of a list that a client desires in PIR, or of a message to decode using a code
k	security parameter
l	list on which to do PIR
l'	pre-computed list in PIR with pre-computation
LDC	locally decodable code
m	length of a pre-computed list in PIR with pre-computation, or of a codeword
n	length of a list on which to do PIR, or of a message to be encoded with a code
p	polynomial
PIR	Private Information Retrieval
Q	PIR query algorithm
q	PIR query
R	PIR recovery algorithm
r	random value
x	message to encode with a code
y	codeword (or corrupted codeword)

List of Figures

1 Introduction: PIR & Computational Cost

1.1 Introduction & Motivation

Private Information Retrieval (PIR) is a cryptographic protocol between a client and one or more servers where each server has an indexed list of items, the client obtains the item at a certain index by querying the server(s), and each server learns nothing about the index of the obtained item. As one trivial solution is for a server to send the entire list of items to the client, we require that the total communication be less than the size of the list, and in fact, the primary focus in PIR research has been to reduce this communication to far below this size. Another point of optimization is the computational burden on the server(s), and while this problem has been studied, computation in the most useful setting has not been reduced significantly below linear in the length of the list [HL09]. As many potential applications of PIR involve very large lists, such as lists of all medical services a hospital has provided or a list of all transactions a credit card processor has completed, linear computation per retrieval is infeasible. To date, the primary ways of reducing computation involve making additional assumptions in the model, such as that many users can coordinate to batch their queries. The currently necessary choice between exotic models and high computational cost of PIR prevents it from being widely used, and so reducing computation would be a significant breakthrough in the applicability of PIR.

There are many choices in how PIR can be modeled:

- there may be multiple non-colluding servers [CGKS95] or one server [KO97];
- we may require low computation for each query, or amortized over a set of queries, where the latter raises the issues of whether this set must consist of contiguous (or otherwise rigidly structured) indices [CGKS95] [GR05] or may be arbitrary, and whether it must be supplied fully before the server(s) can answer [IKOS04] or if they can answer after each index has been given [OG87];
- the protocol may have one round of communication, meaning that the protocol consists of the client sending one query to the server and then receiving a response, or may have multiple rounds;
- we may want to hide the index of the item that the client receives from computationally unbounded servers [CGKS95], or we may assume that their computational time can't exceed

a polynomial function of a user-chosen parameter [CG97].

Each of these options presents one choice that yields more widely applicable or useful (and therefore "better") PIR and one choice that doesn't. Respectively,

- Multi-server PIR gives security only assuming not all servers collude: each sub-protocol consisting of communication between the client and one (or possibly several, but not all) server provides security, but tuples of these sub-protocols generally do not [CGKS95]. Thus, multi-server PIR requires that servers don't have an incentive to collude, and security can be breached externally from the client, limiting its applicability.

- Similarly, if computational cost is only reasonable when amortized over multiple queries, any query for a single item will incur high cost [OG87], querying for batches of items at a time requires the client to know which items it wants non-adaptively [IKOS04], and querying for several items adaptively requires the server to be willing to devote substantial resources to the client, including holding state between the queries, in order for the upfront cost to pay off [OG87]. As a result, this model is interesting and useful in a variety of settings but is not universally so.

- Additionally, one round PIR is sufficiently standard both in the PIR literature and in database protocols, and provides substantial enough definitional simplifications, that it is worth focusing attention on. Furthermore, the distinction between honest-but-curious adversaries, who try to break privacy simply by viewing communication in the protocol, and active adversaries, who try to manipulate the client by sending nefarious messages or otherwise interfering with execution of the protocol, vanishes in a one round protocol. Nonetheless, many discussions and results will generalize to arbitrary numbers of rounds, and I will note that when applicable.

- On the other hand, it is reasonable to assume that the server is computationally limited, and if one-way functions exist, this assumption is necessary for us to achieve single-server PIR at all.

Therefore the most useful variety of PIR is single-server single-round PIR providing computational security and non-amortized efficiency, which I will refer to as the *preferred model*, and this is the setting in which I will define and primarily study PIR.

In this work I analyze the problem of efficient PIR in the preferred model. I will show that existing techniques to improve PIR efficiency do not help us in the preferred model. I present a simplified model of PIR that does not lose generality, define a class of codes generalizing locally decodable codes, and show that such a code is necessary to construct efficient simplified PIR. I prove that known codes do not suffice and find certain properties of codes that must hold to obtain PIR. I conclude without finding a code satisfying all the required properties, but that finding such a code appears to be a very hard problem in cryptography and coding theory.

1.2 Notation & General Definitions

Generally, uppercase letters denote algorithms, random variables, sets, or matrices; lowercase letters denote numbers, lists, or other values. For bit strings, $|\cdot|$ denotes number of bits. For a list l and index i, l_i denotes the ith element of l; thus, for lists l and m, l_{m_i} is the element of l at index m_i. For set S, $x \leftarrow S$ means x is selected uniformly at random from S; for algorithm A, $x \leftarrow A$ means x is selected at random with probability over A's random input or internal randomness.

For $f : \mathbb{N} \to \mathbb{R}$, $o(f)$, $O(f)$, $\Omega(f)$, and $\omega(f)$ are the classes of functions g such that $\lim_{x\to\infty} \frac{g(x)}{f(x)}$ respectively $= 0$, $< \infty$, > 0, and $= \infty$; $\Theta(f) = O(f) \cap \Omega(f)$. An algorithm is *polynomial-time* if its running time is at most some polynomial function of its input's length. $\mathrm{negl}(k)$, or *negligible in* k, is the class of functions in k that are $o(\frac{1}{p})$ for all polynomials p.

For ensembles of random variables $X = (X_k : k \in \mathbb{N})$ and $Y = (Y_k : k \in \mathbb{N})$, $X \stackrel{c}{\equiv} Y$ denotes *computational indistinguishability*, meaning for all probabilistic algorithms A running in time polynomial in k,

$$|\Pr[A(X_k) = 1] - \Pr[A(Y_k) = 1]| \in \mathrm{negl}(k),$$

where probability is over X_k, Y_k, and A's internal randomness. Intuitively, X_k and Y_k rapidly become arbitrarily hard to distinguish as k increases.

A *collision-resistant hash function family* (CRHFF) is an indexed collection of functions $(\{f_i : i \in \{0,1\}^k\})_k$ for infinitely many integers k with the following security property: for all probabilistic algorithms A running in time polynomial in k,

$$\Pr_{i \leftarrow \{0,1\}^k}[(x, y) \leftarrow A(i); f_i(x) = f_i(y)] \in \mathrm{negl}(k).$$

Intuitively, finding a collision for a random function in the family quickly becomes arbitrarily unlikely

in k. (Note that A can depend on the CRHFF because A is chosen after it.)

1.3 PIR: Definition

Formally, a *single-server, single-round, computationally secure PIR system* gives for each value of a security parameter k a triple of deterministic algorithms (Q, A, R) (for query, answer, recovery) such that Q maps randomness r (of length k) and item index i to a query q, A maps a list of items l and a query q to an answer a, and R, given security parameter k, randomness r, and index i, maps an answer a to a final output. All values may be considered bit strings. Let n be the number of entries of l; each entry has the same bit length independent of the other parameters so that $n = \Theta(|l|)$. Q, A, and R must run in time bounded by a polynomial in $k+$ the lengths of their inputs. We require correctness, security, and communication efficiency:

- Correctness: $\forall n$, $\forall l$ of length n and $i \in 1, \ldots, n$,

$$\Pr_r[R(r, i, A(l, Q(r, i)) \neq l_i] \in \text{negl}(k)$$

  where l_i is the ith element of l.[1]

- Security: $\forall n$, $\forall i, j, Q(r, i) \stackrel{c}{\equiv} Q(r, j)$ with respect to k, treating each side as a random variable with randomness r.

- Communication efficiency: $\forall k$, for sufficiently large n, $|q| + |a| < |l|$ always, where $|\cdot|$ denotes bit string length.

The protocol corresponding to a PIR consists of the client running Q and sending q to the server, who runs A and sends a to the client, who runs R to obtain l_i (see Figure 1). Note that the server's computation in this protocol is the running time of A on a random access machine, which is a function of k, $|l|$, n, and q.

1.4 PIR History

1.4.1 early history and communication optimizations

In 1995, Chor, Goldreich, Kushilevitz, and Sudan [CGKS95] introduced PIR in the multi-server setting with security against computationally unbounded adversaries. Their protocol achieved com-

[1] The typical definition requires this probability to be 1, but allowing negligible error provides generality at no practical cost, and will be beneficial later, when using smooth and locally decodable codes for PIR.

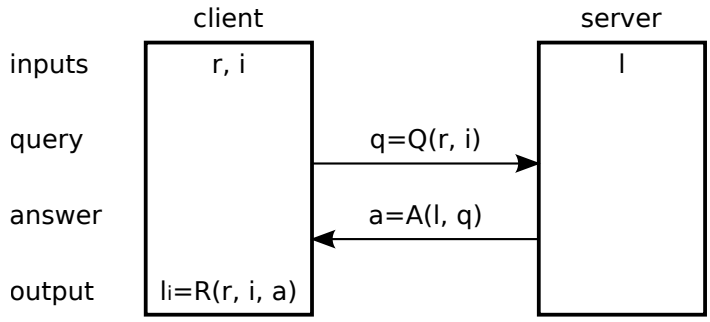

Figure 1: Single-server, single-round, computationally secure PIR protocol.

munication of n^ϵ, where $\epsilon < 1$ is a constant decreasing in the number of servers, and linear total server computation. Subsequent works improved communication complexity in this setting. In 1997, Chor and Gilboa [CG97] introduced PIR with security against computationally bounded adversaries, relying on the existence of one-way functions, providing n^ϵ complexity for all $0 < \epsilon < 1$ with just two servers. Also in 1997, Kushilevitz and Ostrovsky [KO97] introduced PIR in the single-server setting based one the Quadratic Residuosity Assumption, which requires a computationally bounded server, achieving the same communication complexity as before. Subsequent advances in PIR brought communication in the single-server setting to polylogarithmic [CMS99] (Φ-Hiding Assumption), log-squared [HL05] (hardness assumption inherited from underlying encryption), and eventually the optimal logarithmic communication [GR05] (Decision Subgroup Assumption), which extends to $O(1)$ amortized communication per bit as list entries' bit lengths increase sufficiently.

1.4.2 non-amortized computational cost

In the meantime, several results addressed PIR's traditionally high computational cost. In 1998, Gertner, Goldwasser, and Malkin [GGM98] found a way to offload computation from servers with the list l to other peripheral servers. Intuitively, they do this by distributing random shares of the database to the peripheral servers. As the total computation using this scheme is still linear, it is still computationally impractical and thus not relevant to our discussion. Itoh [TI99] reduced the computation in the multi-server model by applying efficient algorithms to reduce the number of bitwise operations by a constant factor, but the computation remained $\Theta(n)$ and so yet again, this result does not make computation practical for large lists.

The first substantial advancement in reducing computation direction was by Beimel, Ishai, and Malkin in 2000 [BIM00]. They proved that any multi-server protocol must have $\Omega(n)$ expected

server computation, but that if the model is changed so that part of A is run only once and before receiving any queries, each query can be answered with $o(n)$ online computation. This model, called "PIR with pre-processing", is formalized for single server PIR in section 2.1. They present two general strategies for reducing the server's online computation. In the protocols they optimize, the answering algorithm A XORs certain entries of l. Their first strategy is to XOR multiple entries of l ahead of time and then use these pre-XORed values when responding to a query. The second strategy is to enumerate all possible queries and compute all possible responses in advance, which applies only when the query space can be enumerated in polynomial time—for example, if the query size is logarithmic.

At about the same time, Katz and Trevisan [KT00] established a connection between locally decodable codes (see section 1.5) and private information retrieval in the multi-server model. This connection creates a multi-server PIR where nearly all of the servers' work may be pushed to pre-processing. Essentially, a locally decodable code can be transformed into a code called a *smooth code* (defined in section 4.1) where when decoding, the probability of reading one entry of the codeword equals the probability of reading any other entry. Thus seeing a single portion of the codeword being read when decoding tells nothing about the message entry being decoded. To turn such a code into PIR, servers encode l according to the code and the client queries a separate server for each entry of the codeword queried in decoding. Security comes from each codeword entry being queried uniformly at random, so that it doesn't reveal anything about the entry of l being decoded. If servers pre-compute the codeword, online computation is constant-time. However, known codes involve a super-polynomially-large blowup of l into a codeword, and thus it is impractical to store or compute the codeword if l is reasonably large.

At this point, advancements in non-amortized PIR efficiency essentially stopped until 2009, when Lipmaa [HL09] showed that it was indeed possible to achieve sublinear computation in the single-server model using pre-processing. Lipmaa's technique is to compute a binary decision diagram (BDD) for the evaluation of an in-the-clear query for a list item. For our purposes, a BDD is a directed acyclic graph with fanout 2 used to evaluate a function, where the input to the BDD is a sequence of left-right directions and the output is a value stored in the leaf at which one arrives via those directions. By previous results, primarily [BHR95], for every list l, there exists a BDD computing $i \mapsto l_i$ with number of vertices a logarithmic factor smaller than the size of the list. The BDD is independent of any query and thus can be computed once in the pre-processing phase.

To execute a PIR query, Lipmaa uses a simple technique for oblivious evaluation of a BDD [IP07]. Lipmaa's result is a breakthrough in that it is the first PIR with sublinear online computation in the preferred model. Unfortunately, the computation savings depend on the size of the BDD, and thus the contents of the list, and in general, are only a factor of $\log(n)$ (and the logarithm of the list entry size), which is insufficient to make PIR practical.[2]

1.4.3 amortized computational cost

In the meantime, there have been numerous results in and relating to PIR that improve efficiency amortized over multiple queries. The earliest idea, oblivious RAM, dates back to an abstract by Goldreich in 1987 [OG87]. An oblivious RAM is a protocol for translating a program on a RAM machine to a program whose memory accesses are independent of the input to the program; thus, an attacker looking at the memory accesses to the RAM cannot tell anything about the input. Running an oblivious RAM program where each value stored in RAM is encrypted to hide it from adversaries reveals nothing about its input, even to adversaries that can see the contents of RAM over time. Thus if the client executes an oblivious RAM protocol where list entries are initially stored sequentially in RAM addresses, the client can continually access entries without revealing the pattern of access. The best generic oblivious RAM translations introduce polylogarithmic communication and computation overhead into a program, and thus they lead to PIR with these amortized overheads. Such protocols require the client to incrementally read more than the entire contents of the RAM before any PIR accesses can occur, so non-amortized PIR computation is $\Omega(n)$. PIR in this setting is only useful if the client must make many queries and has insufficient space to cache the results locally.

In 2004, Ishai, Kushilevitz, Ostrovsky, and Sahai [IKOS04] improved amortized efficiency of PIR by introducing batch codes. Traditionally in the single-server model, the server must scan the entire contents of the list to answer a query. Instead of trying to avoid a full scan, they present codes that expand the list by a factor of less than some k in a way so that a single full scan of the expanded list can answer k simultaneous queries, thereby reducing the computation per query. A batch code works by encoding l into a larger list l' partitioned into buckets, so that one can learn any k entries of l by querying one (or in general more) entry per bucket from l'. Then one can query each bucket using PIR, amounting to one full scan of l', to decode k entries of l. Batch codes provide a significant convenience over oblivious RAM, as they avoid the need to store per-client state and impose lower

[2] Ideally we would want to achieve n^ϵ for fixed $0 < \epsilon < 1$ or even polylogarithmic computation.

upfront cost, but still are in a suboptimal model that requires multiple queries to be sent to the server at once. Thus, this optimization does not help when issuing a single query or when queries must be made adaptively, as when performing a binary search.

Taking advantage of the fully homomorphic encryption revolution [CG09], Lipmaa [HL11] demonstrated a protocol that combines a variety of earlier techniques to reduce computation when evaluating PIR on an arbitrary batch of queries. The protocol works in three stages: in stage one the server applies a batch code so that the number of queries per bucket t is relatively low; in stage two the server then creates for each bucket b a circuit that evaluates a query into b, as $(i_1, \ldots, i_t) \mapsto (b_{i_1}, \ldots, b_{i_t})$; and in stage 3 the client and server evaluate PIR using the batch code, where computation is done homomorphically using the BGV cryptosystem.

1.5 Locally Decodable Codes

Locally decodable codes (LDCs) are error-correcting codes mapping a message to a codeword such that even if the codeword is slightly corrupted, one may recover a small portion of the message by viewing small portions of the codeword. More formally, given message alphabet Σ_1 and length n, codeword alphabet Σ_2 and length m, query count c, error rate δ, and decoder error probability ϵ, a locally decodable code is a map $C : \Sigma_1^n \to \Sigma_2^m$ where there is a randomized decoder D such that:

- correctness: $\forall x \in \Sigma_1^n$ and y with Hamming distance $d(C(x), y) < \delta m$, $\forall i \in \{1, \ldots, n\}$, $\Pr[D(y, i) \neq x_i] < \epsilon$ where the probability is over D's randomness;[3]
- locality: D queries at most c elements of y.

I will review some coding terminology and developments that relate to later portions of this document but will not give a complete exposition of LDCs; for more info, see [LT04] and [DGY11].

While [KT00] first formalized LDCs, the earliest instances date back at least to Bose and Shrikhande [BS59], which introduced the Hadamard code. This code is the simplest LDC and motivated more advanced code constructions. It works as follows: the message is $\vec{x} \in \{0,1\}^n$; let the "dot product by $\vec{x}$" function $\odot_{\vec{x}}(\vec{z}) = \vec{x} \cdot \vec{z}$. Then the codeword is the evaluation of $\odot_{\vec{x}}$ on all 2^n $\vec{z}$s in some fixed (say lexographical) order. To decode bit i, we choose uniformly random $\vec{z}$ and let $\vec{z'}$ be the result of flipping $\vec{z}$'s ith bit; then $\odot_{\vec{x}}(\vec{z}) \oplus \odot_{\vec{x}}(\vec{z'}) = (\vec{x} \cdot \vec{z}) \oplus (\vec{x} \cdot \vec{z'}) = x_i$. If neither of

[3] [KT00] uses ϵ to denote decoder advantage over randomly guessing, whereas [DGY11] uses ϵ as error probability. The latter choice is simpler when generalizing to non-binary alphabets.

those entries in the codeword is corrupted, we decode correctly. (There are better decoders that can handle more errors; for example, a list decoder from [GL89].) Unfortunately, the Hadamard code yields exponentially long codewords.

In the 1990s, a new class of LDCs called polynomial codes were introduced, which draw on ideas from older, non-locally-decodable error-correcting codes like the Reed-Solomon code [RS60]. These older codes essentially work by encoding the message x as a univariate polynomial p and letting $C(x)$ be the evaluation of p at many points (many more than the degree of p). Thus, the code has the same general structure as the Hadamard code, in that the message defines a function and the codeword is the evaluation of this function. To decode, we sample the codeword and given enough points, we can interpolate p and then recover x. We can interpolate p even if some sampled points are incorrect using the Berlekamp-Welch algorithm [BW86]. Note that this process decodes globally, not locally, as we obtain all of x and not just some x_i.

Locally decodable polynomial codes, such as the Reed-Muller code [DEM54] [ISR54], use multivariate polynomials and have the additional property that each message entry x_i is the evaluation of the polynomial p at some fixed point a_i (where a_i is independent of x). Then given a line through a_i, $l(t) = a_i + bt$, $p \circ l$ is a univariate polynomial in variable t with $\deg(p \circ l) \leq \deg(p)$ and $p(l(0)) = p(a_i) = x_i$. Given this observation, x_i can be locally decoded as follows: determine a_i corresponding to i, choose random b, and let $l(t) = a_i + bt$. Sample the codeword entries containing $p(l(1))$, $p(l(2))$, etc. and interpolate $p \circ l$, which is easy because it is univariate. Then, evaluate $p(l(0)) = x_i$. These codes must manage the conflicting requirements that p has enough terms and a high enough degree that there exist coefficients so that $p(a_i) = x_i$ for each i, and that p has a low enough degree or the right algebraic structure to allow interpolation along a line without sampling too many points.

Nearly all known locally decodable codes, including polynomial codes, are linear, meaning $C(x) = Mx$ for some matrix M. Until [DW08], all known LDCs were linear; Woodruff defined a broader family of codes, subsuming all known codes as well as his new codes, having certain types of decoders called *matching sum decoders*. Intuitively, these codes logically assign to each message index i a list of possible queries, where each query is a c-tuple of codeword indices. These queries are chosen to be disjoint, so that no two queries share an index. To decode x_i, the decoder randomly chooses a query corresponding to index i, reads the bits of y corresponding to the query, and XORs these bits to (hopefully) obtain x_i.

1.5.1 Matching Vector Codes

At about the same time, Yekhanin [SY07] introduced a new type of linear polynomial-based code on bitwise messages called *matching vector codes*, and most work on LDCs since then has been on this family. As these are the best LDCs, I will describe them in detail. Let p be a prime power and choose a finite field F of size p. Our message entries will be in F. Choose cyclic subgroup G of size q such that $q|p-1$ and let g be a generator. As always, let n be the message length. Choose integer d; we will work with vectors of this dimensionality. For vector $\vec{v}$ and group element h, let $h^{\vec{v}} = (h^{v_1}, h^{v_2}, \ldots, h^{v_d})$ and let h^M for matrix M be defined similarly.

We choose set $S \subset \mathbb{Z}_q \setminus \{0\}$, $U = (\vec{u_1}, \ldots, \vec{u_n})$ and $V = (\vec{v_1}, \ldots, \vec{v_n})$ with all $\vec{u_i}, \vec{v_i} \in \mathbb{Z}_q^d$ such that for all i, $\vec{u_i} \cdot \vec{v_i} = 0$ and for all $i \neq j$, $\vec{u_i} \cdot \vec{v_j} \in S$. U and V are called an S-matching; see [DGY11] for examples of such matchings. n is the message length and $|S|$ is proportional to decoding work, so the goal is for n to be large but $|S|$ to be small in comparison.

The encoder C works as follows. Given message $\vec{x} \in F^n$, let

$$p(\vec{z}) = \sum_{a=1}^{n} x_a \prod_{b=1}^{d} z_a^{(\vec{u_a})_b}.$$

p is a polynomial in $\vec{z} \in G^d$. Then $C(\vec{x}) = (p(\vec{h}) : \vec{h} \in G^d)$.

The decoder D works as follows. Given message index $i \in \{1, \ldots, n\}$, D obtains the associated vector $\vec{v_i}$, chooses uniformly random $\vec{w} \in \mathbb{Z}_q^d$, and for "line" $L(j) = g^{\vec{w}+j\vec{v_i}}$, queries the codeword indices $(L(j) : j \in \{0, \ldots, |S|\})$. To understand how decoding the resulting values works, consider the function $p \circ L : \mathbb{Z}_q \to F$:

$$\begin{aligned}
p(L(j)) =& \sum_a x_a \prod_b \left(g^{\vec{w}_b + j(\vec{v_i})_b}\right)^{(\vec{u_a})_b} \\
=& \sum_a x_a g^{\sum_b (\vec{u_a})_b (\vec{w}_b + j(\vec{v_i})_b)} \\
=& \sum_a x_a g^{\sum_b (\vec{u_a})_b \vec{w}_b} g^{j \sum_b (\vec{u_a})_b (\vec{v_i})_b} \\
=& \sum_a x_a g^{\vec{u_a} \cdot \vec{w}} (g^j)^{\vec{u_a} \cdot \vec{v_i}} \\
=& x_i g^{\vec{u_i} \cdot \vec{w}} + \sum_{s \in S} \left(\sum_{a : \vec{u_i} \cdot \vec{v_a} = s} x_a g^{\vec{u_a} \cdot \vec{w}} \right) (g^j)^s.
\end{aligned}$$

The last equality follows by partitioning the sum terms by their $\vec{u} \cdot \vec{v}$ values and because $i \neq j \Rightarrow \vec{u_i} \cdot \vec{v_j} \in S$.

If we sample $p \circ L$ at $j = 1, 2, \ldots$ and then perform polynomial interpolation as if we had sampled $g^1, g^2, \ldots$ we will obtain the univariate polynomial

$$p'(y) = x_i g^{\vec{u_i} \cdot \vec{w}} + \sum_{s \in S} \left(\sum_{a : \vec{u_i} \cdot \vec{v_a} = s} x_a g^{\vec{u_a} \cdot \vec{w}} \right) y^s.$$

This polynomial has $|S| + 1$ terms and thus requires $|S| + 1$ interpolation points, which D queries. Furthermore, $x_i = \frac{p'(0)}{g^{\vec{u_i} \cdot \vec{w}}}$, and thus once we obtain p', we can easily get x_i.

2 Efficient PIR & Codes

2.1 PIR with pre-processing

One major stumbling block in achieving efficient PIR in the preferred model is the widely known fact that the server's total computation to answer a query, or the cost of computing A, must be $\Omega(|l|)$ (and thus $\Omega(n)$). This is because the server's answer a cannot depend on the entirety of any entry l_i of l that A does not fully read, so a is invariant under changes to some bit of l_i and thus cannot possibly communicate that bit to the client. Therefore there is no hope for A to achieve $o(|l|)$ computation in the preferred model directly—we may only hope to push as much computation as possible to a pre-processing phase, which motivates the following definition.

PIR with pre-processing in the preferred model: A *PIR with pre-processing system* gives for each value of a security parameter k a quadruple of algorithms $(Q, A_{\text{pre}}, A_{\text{online}}, R)$ such that $(Q, (l, q) \mapsto A_{\text{online}}(A_{\text{pre}}(l), q), R)$ is a PIR system, $A_{\text{pre}}(l) = l'$, and $A_{\text{online}}(l', q) = a$. l' is the pre-processed list and is independent of any q, so that it may be computed once and before queries are issued. As before, we define n as the number of entries of l, and we let m be the number of entries of l'; m is a function of k. For simplicity, we assume that entries of l and l' have the same bit length; if not, we may repartition l' so that this is true so this assumption doesn't lose generality.

There are several efficiency metrics that may interest us. The primary metric is the server's online, per-query computational cost, which for a given l' and q is $\text{cost}(A_{\text{online}}, l', q)$, is the computational cost of evaluating $A_{\text{online}}(l', q)$. As we want our algorithms to save computational cost for any l and i, and as cost may depend on r, we measure the computational cost of a PIR system with

pre-processing as

$$\text{cost}(Q, A_{\text{pre}}, A_{\text{online}}, R) = \max_{l,i} \text{E}_r[\text{cost}(A_{\text{online}}, A_{\text{pre}}(l), Q(r, i))].$$

Other efficiency metrics are the rate of expansion $\frac{m}{n}$, computational cost of A_{pre}, and the cost of updating, inserting, or deleting one entry of l. This document will not focus on these other metrics, but they are worth addressing nonetheless.

2.2 simplified PIR

While Katz and Trevisan observed a connection between codes having a local decoding property and multi-server PIR, I will introduce a straightfroward connection between these codes and efficient single-server PIR with pre-processing, and viewing PIR in this model as a code will be the primary means of analyzing it. Casually, given a PIR system $(Q, A_{\text{pre}}, A_{\text{online}}, R)$, A_{pre} encodes message l into codeword l', A_{online} queries entries of the codeword l', and A_{online} and R decode an entry of l given the results of A_{online}'s queries. In fact, any efficient PIR can be simplified into a scheme where q dictates in the clear which portions of l' that A_{online} will read, as the server would learn this information anyway, and a merely consists of these portions of l'. Now computation just involves reading q and reading the specified entries of l', so computation and communication are proportional and we have turned two optimization problems into one. Therefore the PIR scheme essentially becomes nothing more than a code, where Q is the querying algorithm and R is the decoder.

In this new scheme, all security must come from the code itself. While applying cryptographic techniques to q and a may further reduce communication, they would not reduce computation or enhance privacy beyond what can be done in the simplified model, and as the primary focus of this work is to reduce computation while maintaining privacy, we will only consider the simplified scheme described here.

Summarizing and formalizing, a *simplified PIR with pre-processing system* is a PIR with pre-processing system where for some query index count c,

- $q = (q_1, \ldots, q_c)$ for some $q_j \in \{1, \ldots, m\} \forall j$;
- $a = (l'_{q_1}, \ldots, l'_{q_c})$.

We set A_{online} to be the canonical algorithm that scans q and reads entries of l' as requested. The

computational cost is proportional to the number of entries queried, c. While in general c may depend on the index of the desired list entry i, as we defined cost as the maximum expected work taken over l and i, we can fix c to the maximum number of entries that must ever be read and pad shorter queries with random, unique indices.

2.3 codes with local decoding

The following type of code relates closely to simplified PIR. Given an alphabet Σ, a message length n, codeword length m, query count c, and decoder error probability ϵ, a *code with local decoding* is a function $C : \Sigma^n \to \Sigma^m$ such that for some randomized algorithm D (for decoder),

- correctness: $\forall x \in \Sigma^n$ and $i \in \{1, \ldots, n\}$, $\Pr[D(C(x), i) \neq x_i] < \epsilon$ where randomness is over D;
- locality: D reads at most c entries of $C(x)$ per run.

A family of such codes for all n and security parameter k has m, c, and ϵ depending on n and k such that C and D are polynomial-time in $k+$ the lengths of their inputs, and for each n, ϵ is negligible in k.

There are several important differences, both definitional and in intended usage, between codes with local decoding and LDCs. One such difference is that the study of LDCs typically treats c as constant in n and m; that is, the number of queries one makes does not grow as the data size grows. However, with PIR all we ask for is that $\frac{c}{n}$ is sufficiently small; that is, the server's computational work c is sufficiently less than the data size n. Furthermore, codes with local decoding do not correct errors whereas LDCs do. These two differences suggest that the former are easier to construct than the latter. On the other hand, the former require negligible probability of decoding incorrectly in a security parameter. However, this distinction is only superficial: for any code where probability of decoding is bounded below by a constant above $\frac{1}{|\Sigma|}$, repeatedly decoding and taking the majority pushes correctness to 1 exponentially, while only multiplying effort by some factor times the security parameter.

Also note that while the study of LDCs to date has focused on non-adaptively queried codes, meaning the decoding algorithm queries all entries of the codeword at once instead of deciding which entry to query next based on the values of previously queried entries, neither LDCs nor codes with local decoding are inherently non-adaptive. As any adaptive code gives an equivalent multi-round

PIR and vice versa, by studying PIR as a code, many of the following analyses will generalize to multi-round PIR.

3 Negative Results

3.1 linear codes and collision resistance

As explained in section 2.2, A_{pre} is a c-query code with local decoding and R is a decoder. Thus, relating codes with local decoding to LDCs, we may look for solutions to the problem of efficient PIR in the coding literature, or try to use codes for the multi-server model presented in [BIM00]. Unfortunately, while a PIR scheme is a code, the converse is not usually true—in fact, I will show in this section that most common families of codes don't work. This is because PIR requires that $Q(r,i) \stackrel{c}{\equiv} Q(r,j)$, where $\stackrel{c}{\equiv}$ means computational indistinguishability, and thus that the entries of the code that the decoder reads must not reveal anything about the index being decoded.

For an example of how codes fail to meet this condition, consider the 2-server warmup PIR scheme by Chor et. al. where to obtain l_i, one chooses a uniformly random subset of indices q and queries server 1 for $\bigoplus_{j\in q} l_j$ and server 2 for $\bigoplus_{j\in q'} l_j$, where $q' = q \cup \{i\}$ if $i \notin q$, $q \setminus \{i\}$ if $i \in q$. This is an on-the-fly application of the Hadamard code, and knowing both queries reveals i, so the Hadamard code does not provide security in our model. More generally, the following lemma quickly rules out all the codes in [BIM00] and nearly all known locally decodable codes:

Lemma 1 *No linear code with local decoding gives a simplified PIR with pre-processing system.* □

Proof For $m \times n$ matrix M, let $x \mapsto Mx$ be a linear c-query code from n-vectors to m-vectors. Viewing this code as a simplified PIR scheme, $a = (l'_{q_1}, \ldots, l'_{q_c}) = M_q l' = (M_q M)l$ where M_q is the $c \times m$ matrix whose rows are the standard basis vectors $e_{q_1}, \ldots, e_{q_c}$. $M_q M$ is $c \times n$ and $c < n$ and so $(M_q M)e_1, \ldots, (M_q M)e_n$ must have a linear dependence $\sum_{j=1}^{n} f_j (M_q M)e_j = (M_q M)\sum_j f_j e_j$. Then changing l to $l + f$ where $f = (f_1, \ldots, f_n)$ yields query response $(M_q M)(l + f) = (M_q M)l + (M_q M)\sum_{j=1}^{n} f_j e_j = M_q M l = a$. Let j^* be such that $f_{j^*} \neq 0$. l and $l + f$ differ on entry j^* yet a does not change and so $i \neq j^*$. As the server can efficiently find such a linear dependence by solving $\sum_j f_j (M_q M)e_j$ for the f_js, it can efficiently violate the client's privacy. ■

The fundamental problem with linear codes is that, given q, the server can find some $l_1 \neq l_2$

producing the same a.[4] Let $A_q(l) = A_{\text{online}}(A_{\text{pre}}(l), q) = a$. Then $A_q^{-1}(a) = \{l : A_{\text{online}}(A_{\text{pre}}(l), q) = a\}$. If the server can find multiple lists in $A_q^{-1}(a)$, it can immediately rule out i being any indices at which these lists differ. Note that in general, as A maps a length-n input to a length-c output, $c < n$, we cannot hope to prevent multiple preimages of A_q. Instead, we must require that they are hard to find. Formally,

Lemma 2 *Let $(Q, A_{pre}, A_{online}, R)$ be a PIR with pre-processing system and let $A_q(l)$ be as above. Then $\forall i$, $\{A_{Q(r,i)} : r \in \{0,1\}^k\}$ is a collision-resistant hash function family (CRHFF).*

Proof The lemma's statement expands to $\forall n, i \in 1, \ldots, n$, PPT algorithm $\mathcal{C}$ (for collision finder) taking as input a query q and outputting two distinct lists l_1 and l_2, and polynomial p, $\Pr_r[q \leftarrow Q(r,i); (l_1, l_2) \leftarrow \mathcal{C}(q); A_q(l_1) = A_q(l_2)] < \frac{1}{p(k)}$ for all sufficiently large k. Suppose this is false for some PPT algorithm $\mathcal{C}$. Let $\mathcal{G}$ be a "guesser" of i that works as follows: on input q, $\mathcal{G}$ runs $\mathcal{C}(q)$ to obtain distinct l_1 and l_2. If $A_q(l_1) = A_q(l_2)$ then $\mathcal{G}$ outputs a uniformly random index i' on which l_1 and l_2 agree; otherwise, $\mathcal{G}$ outputs a uniformly random index i'. Then for sufficiently large k and all q,

$$\begin{aligned}
\Pr[\mathcal{G}(q) = i] =& \Pr[\mathcal{G}(q) = i | A_q(l_1) = A_q(l_2)] \cdot \Pr[A_q(l_1) = A_q(l_2)] + \\
& \Pr[\mathcal{G}(q) = i | A_q(l_1) \neq A_q(l_2)] \cdot \Pr[A_q(l_1) \neq A_q(l_2)] \\
\geq& \frac{1}{n-1} p(k) + \frac{1}{n}(1 - p(k)) \\
=& \frac{1}{n} + \frac{1}{n(n-1)} p(k).
\end{aligned}$$

This is a non-negligible advantage in k over uniformly random guessing. Thus, $\mathcal{G}$ is a non-negligible distinguisher between the random variables $Q(r,i)$ and $Q(r,j)$ for $i \neq j$, which contradicts the definition of PIR. ∎

Note that the converse of lemma 2 is not true: given a secure PIR scheme, we may modify it so that each query additionally includes i. A_q remains collision resistant but the scheme is no longer secure.

Also note that the security requirement for PIR is that $Q(r,i) \stackrel{c}{\equiv} Q(r,j)$, which does not depend on the particular choice of answer or recovery algorithms A or R. Therefore if an adversary can

[4] The ability to find preimages allows for list decodability, a normally advantageous property of codes.

find collisions in A_q for PIR system $(Q, A_{\text{pre}}, A_{\text{online}}, R)$, the adversary can distinguish between $Q(r,i) \stackrel{c}{\equiv} Q(r,j)$ for some i and j, so any other PIR system using Q, $(Q, A'_{\text{pre}}, A'_{\text{online}}, R')$, is also insecure. Thus security requires that for *all* codes that Q queries, A_q is collision resistant.

3.2 polynomial codes

A direct consequence of the collision resistance of A_q is that a broad class of polynomial codes, described in section 1.5, do not provide the security required for PIR. All known polynomial codes are such that $p = \sum_{i=1}^{n} x_i p_i$ for message x, multivariate monic monomials p_i, and codeword $y = (p(j) : j \in \mathcal{I})$ for some fixed set of indices $\mathcal{I}$. Then $y = (\sum_{i=1}^{n} x_i p_i(j) : j \in \mathcal{I})$, which is linear in x. We may generalize these linear polynomial codes to ones such that p is fixed except for its coefficients and x determines the coefficients via some arbitrary bijection $b(x) = (b_1(x), b_2(x), \ldots)$: $p = \sum_i b_i(x) p_i$ for fixed monomials p_i, $y = (p(j) : j \in \mathcal{I})$, and decoding x_i interpolates some points $p(q_1), p(q_2), \ldots$ to find some p' with $p'(q_j) = p(q_j)$ for all j and $p'(0) = x_i$. ($q_1, q_2, \ldots$ might not be colinear.)

Lemma 3 *A code of the above generalized form does not give secure PIR.* □

Proof Moving to the notation of PIR, the encoder maps $l \mapsto p \mapsto l'$. A query is some $q \subset \mathcal{I}$ with $|q| = c$ and we require that $(l \mapsto l'_q := (l'_i : i \in q))$ is a CRHFF, meaning it is hard to find lists l_1, l_2 such that $l_j \mapsto p_j \mapsto l'_j$, $j \in \{0,1\}$, and $(l'_1)_q = (l'_2)_q$. Note that $l \mapsto p$ is a bijection and thus has an inverse, and this inverse is efficiently computable as one may compute $p \mapsto l'$ and decode every l_i efficiently. Thus collision-resistance requires that we can't find p_1, p_2 of the required form (same up to coefficients) with corresponding l'_1, l'_2 such that $(l'_1)_q = (l'_2)_q$.

Note that we may set p_1 arbitrarily and to violate collision resistance we need only find p_0 of the required form with $p_0(q) = (0, \ldots, 0)$ and at least one nonzero coefficient; then $p_2 = p_1 + p_0$. Finding p_0 only requires finding a nonzero solution to the system

$$\begin{pmatrix} m_1((l'_1)_{q_1}) & \ldots & m_n((l'_1)_{q_1}) \\ \ldots & \ldots & \ldots \\ m_1((l'_1)_{q_c}) & \ldots & m_n((l'_1)_{q_c}) \end{pmatrix} \begin{pmatrix} a_1 \\ \ldots \\ a_n \end{pmatrix} = \vec{0},$$

where $m_1 \ldots, m_n$ are the monic monomials of our code and $a_1, \ldots, a_n$ are the coefficients we are solving for. As this system has n columns and c rows with $c < n$, there must be a column dependence

so a nonzero solution exists, and it is easy to find. ■

In a different setting where a code maps $l \mapsto p$ non-bijectively (but necessarily injectively as the code must be decodable), an adversary can still find p_1, p_2 agreeing on elements of q as above. However, these polynomials might not be the images of any lists, and thus finding a polynomial collision does not constitute a list collision, in which case security is not violated. If the set of polynomials with preimages is sufficiently sparse and random-looking, it is conceivable for the code to be secure.

3.3 matching sum decoders

Turning to codes with matching sum decoders, defined in section 1.5, it is easy to see that they cannot possibly provide the necessary security for PIR. The first issue is that the decoding process, consisting of XORing queried entries of the codeword, does not involve a secret. Thus the server may decode by itself, so it may decode the lists $l_1 = (1, 0, 0, \ldots), l_2 = (0, 1, 0, \ldots), \ldots, l_n = (\ldots, 0, 0, 1)$ and the list for which the decoded bit is 1 gives i.[5]

The second problem is that a matching sum decoder works by choosing a random query among a set of disjoint queries for the desired index. Therefore the number of such queries must be $O(m) \subseteq O(\text{poly}(k))$, so all queries can be placed in a lookup table in polynomial time. Each query may not correspond to more than c indices of l in this table because the query may not contain more information than its size. Thus, given a query, choosing a random corresponding index in the table distinguishes between some two indices non-negligibly.

By the above argument, the number of queries for each index must grow super-polynomially large in the security parameter to make enumeration infeasible. Thus the performance optimization in [BIM00] where the server enumerates all possible queries and pre-computes the responses does not help us in the preferred model.

Generalizing these attacks, it is clear that both the encoder and decoder must use the client's randomness r. More specifically, the PIR system must work as in Figure 2.

[5] As an aside, for any PIR system, this brute force decoding process allows one in possession of the client's randomness r to decode a query q into the index queried, i, and because PIR's privacy definition is that of semantic security, any PIR gives a semantically secure encryption algorithm $E_r : i \mapsto Q(r, i)$. Furthermore, viewing the list l as an evaluation of a function $i \mapsto l_i$, PIR gives a procedure for mapping $E_r(i)$ to $E_r(l(i))$.

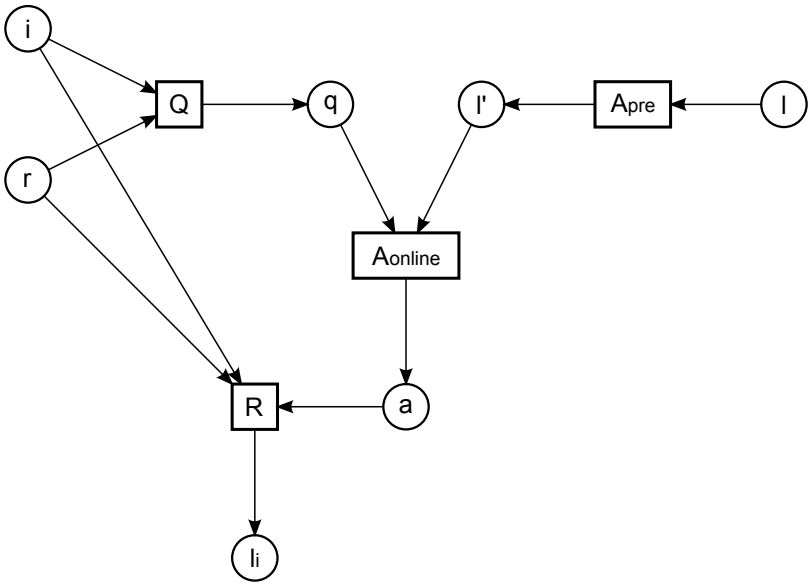

Figure 2: Structure of PIR with precomputation in the preferred model.

4 Information Dispersal

Intuitively it is obvious that a pre-computation phase of PIR must disperse information from the list l throughout the pre-computed list l'. This section seeks to qualify this information dispersal. In particular, I will show that efficient PIR in the preferred model must have a "smoothness" property forcing entries (or subsets) of l' to contain information about many entries of l, and an "injectivity" property requiring that entries of l disperse their information across many entries (or subsets) of l'.

4.1 smoothness

Smooth codes [KT00] are codes where for all desired message indices, the probability that a query for that index contains any one codeword index is close to, or exactly, the probability that a query contains any other codeword index. Formally, and similarly to LDCs, a smooth code C maps a message $x \in \Sigma_1^n$ to a codeword $y \in \Sigma_2^m$ such that for some query count c, decoder error probability ϵ, and roughness allowance factor $\rho \geq 1$, there is a randomized decoder D where:

- correctness: $\forall x$ and $i \in \{1, \ldots, n\}$, $\Pr[D(y, i) \neq x_i] < \epsilon$;
- locality: D queries at most c elements of y;
- ρ-smoothness: for each message x, message index i, and codeword index j, $\Pr[D(y, i)$ reads index $j] \leq$

$\frac{c\rho}{m}$.

Probabilities are over D's randomness. Here ρ is a "roughness allowance factor" ≥ 1, a factor by which probabilities that indices are read may exceed a uniform probability. For some examples of smooth codes, see [KT00].

If $\rho \approx 1$, seeing a single codeword index that a c-element query contains reveals little to nothing about the message index. Furthermore, if $\rho - 1 \in \text{negl}(k)$, we get multi-server PIR where the client queries each codeword index from a different server. However, if the number of entries of a query is sometimes $< c$ then there may be a severe disparity between queries for one index versus for another. For example, if the decoder always queries exactly one codeword index to decode message index i, it may query a codeword index j a fraction $\frac{c\rho}{m}$ of the time, whereas if it queried uniformly, it would only query j a fraction $\frac{1}{m}$ of the time. Therefore will focus on the case where D queries exactly c elements of y; we can always pad queries if necessary.

Smoothness is stronger than what is required for PIR. We may relax the smoothness requirement to:

- inverse δ-smoothness: for each message x, message indices i_1 and i_2, and codeword index j,

$$|\Pr[D(y, i_1) \text{ reads index } j] - \Pr[D(y, i_2) \text{ reads index } j]| \leq \delta.$$

While smoothness requires that each message index corresponds (nearly) uniformly to codeword indices, inverse smoothness requires that each codeword index corresponds (nearly) uniformly to message indices. When $\delta \in \text{negl}(k)$, inverse smoothness is exactly the security requirement for computationally secure multi-server PIR using the one-server-per-query-index technique as $m\delta$ upper-bounds the statistical distance, and thus any adversarial advantage, between single elements of queries for different indices.

Inverse smoothness is strictly more general than smoothness:

Lemma 4 *ρ-smoothness where $\rho - 1 \in \text{negl}(k)$ implies inverse δ-smoothness where $\delta \in \text{negl}(k)$.* □

Proof Each query q contains c indices of the codeword, so for each codeword y and entry to decode

i, $\sum_{j=1}^{m} \Pr[D(y,i) \text{ reads } j] = c$. Then for any j,

$$\begin{aligned}\Pr[D(y,i) \text{ reads } j] =& c - \sum_{j' \neq j} \Pr[D(y,i) \text{ reads } j'] \\ \geq& c - \frac{(m-1)c\rho}{m} \\ =& c(1-\rho) + \frac{c\rho}{m}.\end{aligned}$$

So $\frac{c\rho}{m} \geq \Pr[D(y,i) \text{ reads } j] \geq \frac{c\rho}{m} - c(\rho - 1)$. Therefore for all i_1 and i_2,

$$|Pr[D(y,i_1) \text{ reads } j] - Pr[D(y,i_2) \text{ reads } j| \leq 2c(\rho - 1) \in \text{negl}(k).$$

■

On the other hand, inverse smoothness does not imply smoothness because the former allows for an index that is always queried.

In the single-server setting we clearly require inverse smoothness: otherwise a multi-server attack would translate into a single-server attack, distinguishing between message indices on the basis of whether some codeword index is queried. Furthermore, in single-server PIR the server sees *all* query indices, and thus for any subset S of l', there exists an adversary hardwired to distinguish based on S. Therefore we must require "smoothness" with respect to any fixed-sized (independent of k) subset of l', which we define as follows:

- inverse δ-smoothness with respect to S: for each message x, message indices i_1 and i_2, and $S \subset \mathbb{N}$,

$$|\Pr[D(y,i_1) \text{ reads all indices in } S] - \Pr[D(y,i_2) \text{ reads all indices in } S]| \leq \delta.$$

When $\delta \in \text{negl}(k)$, this requirement intuitively states that for each fixed set of indices S, for sufficiently large k, queries containing S may as well be for any message index i. Because S itself cannot contain information about all message indices, queries must contain sufficiently many indices not in S.

Lemma 5 *Secure PIR implies that* $\forall S \subset \mathbb{N}$, $\exists \delta \in \text{negl}(k)$ *such that the corresponding code is inverse* δ*-smooth with respect to* S*. However, the converse does not hold. In other words, inverse smoothness of all subsets of each fixed size is necessary, but not sufficient, for PIR security.* □

PROOF Necessity is clear: if a code is not S-smooth then there are indices i_1, i_2 between which the adversary $q \mapsto S \stackrel{?}{\subseteq} q$ non-negligibly distinguishes. Insufficiency is via a simple example. Let $(Q, A_{\text{pre}}, A_{\text{online}}, R)$ be a secure PIR with pre-computation. Let $A'_{\text{pre}}(l) = (A_{\text{pre}}(l), 0, \ldots, 0)$ where the number of 0s at the end is $m' = n(1 + \lceil \frac{k}{n} \rceil)$. Let $Q'(r', i) = (Q(r, i), q_{n+1}, \ldots, q_{n+k})$ where r is a sufficient portion of r' for Q and $(q_{n+1}, \ldots, q_{n+k})$ is uniformly random such that each $q_{n+j} \in \{n + 1, \ldots, n + m'\}$ (queries a 0 at the end of l') and $\sum_{j=1}^{k} q_{n+j} \equiv i (\text{mod}\, n)$. Then for each query q from Q', any $k-1$ of $q_{n+1}, \ldots, q_{n+k}$ are uniformly random in $\{n+1, \ldots, n+m'\}$ because for each choice of those $k-1$ indices, there exist exactly $1+\lceil \frac{k}{n} \rceil$ choices for the kth index so that $\sum_{j=1}^{k} q_{n+j} \equiv i (\text{mod}\, n)$. Therefore for each $S \subseteq \{1, \ldots, n+2k\}$ and for any $k > |S|$, $\Pr_r[S \subseteq Q'(r', i_1)] = \Pr_r[S \subseteq Q'(r', i_2)]$. But clearly each query reveals i. ∎

4.2 injective subsets

As a consequence of section 3.3, for each $i \in \{1, \ldots, n\}$, we require super-polynomially many subsets of size $c(k) \in \text{poly}(k)$ from which l_i can be determined. More formally, call an *injective subset for index i* some $S \subset \{1, \ldots, m\}$ for which for all $l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_n$, $x \mapsto (A_{\text{pre}}(l_1, \ldots, l_{i-1}, x, l_{i+1}, \ldots, l_n)_s : s \in S)$ is injective; we require that for all i, $|\{S : S \text{ is an injective subset for index } i\}|$ is super-polynomial in k. Clearly these sets must overlap, and as their total size exceeds m by a super-polynomial factor, on average each element of l' must appear in super-polynomially many of these sets. Each query $Q(r, i)$ must contain an injective subset for i, and cannot contain an injective subset for all other indices, as then it would contain all information about l.

5 Conclusion

Efficient PIR in the preferred model requires pre-computation and reduces to encoding the server's list in a secure and locally decodable manner, similarly to encoding according to a locally decodable code. Unfortunately, known families of these codes don't provide the necessary security guarantees for PIR.

Nonetheless, we can work toward classifying secure codes by finding some properties that they must have. First of all, each query must be an encryption of the queried index and the set of queries for each index must induce a collision-resistant hash function on lists. Secondly, the pre-computation phase must spread information in the list throughout the pre-computed list.

Actually finding such a code appears to be a very difficult problem and it is not even clear whether such a code exists. We must hope that one does exist because the practicality of PIR in many contexts, such as a drop-in replacement for a database or key/value store, requires closing the gap between near linear-time state-of-the-art PIR protocols and insecure protocols running in logarithmic time.

References

[BIM00] A. Beimel, Y. Ishai, T. Malkin. Reducing the Servers' Computation in Private Information Retrieval: PIR with Preprocessing. In M. Bellare, editor, *Advances in Cryptology—CRYPTO 2000*, vol. 1880 of *Lecture Notes in Computer Science*, pages 56-74. Springer, 2000.

[BW86] E. R. Berlekamp and L. Welch. Error correction of algebraic block codes. U.S. Patent, Number 4,633,470, 1986.

[BS59] R. Bose, S. Shrikhande. A note on a result in the theory of code construction. In *Inf. And Control* vol. 2, pages 183-194, 1959.

[BHR95] Y. Breitbart, H. B. Hunt III, D. J. Rosenkrantz. On The Size of Binary Decision Diagrams Representing Boolean Functions. In *Theoretical Computer Science*, vol. 145 (1&2), pages 4569, 1995.

[CG97] B. Chor, N. Gilboa. Computationally Private Information Retrieval. In *Proceedings of the Twenty-ninth Annual ACM Symposium on the Theory of Computing*, 1997.

[CGKS95] B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan. Private Information Retrieval. In *Proc. of FOCS'95*, pages 41-50. IEEE, 1995.

[CMS99] C. Cachin, S. Micali, M. Stadler. Computational Private Information Retrieval with Polylogarithmic Communication. In J. Stern, editor, *Advances in Cryptology—EUROCRYPT '99*, vol. 1592 of *Lecture Notes in Computer Science*, pages 402-414. Springer, 1999.

[DGY11] Z. Dvir, P. Gopalan, S. Yekhanin. Matching vector codes. In *SIAM Journal on Computing*, 2011.

[CG09] C. Gentry. A fully homomorphic encryption scheme. PhD thesis, Stanford University, 2009.

[GL89] O. Goldreich, L. A. Levin. A Hard-Core Predicate for all One-Way Functions. In *Association for Computing Machinery*, pages 25-32, 1989.

[GR05] C. Gentry, Z. Ramzan. Single-Database Private Information Retrieval with Constant Communication Rate. In L. Caires, G. Italiano, L. Monteiro, C. Palmidessi, an M. Yung,

editors, *ICALP 2005*, vol. 3580 of *Lecture Notes in Computer Science*, pages 803-815. Springer, 2005.

[GGM98] Y. Gertner, S. Goldwasser, 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*, vol. 1518 of *Lecture Notes in Computer Science*, pages 200-217. Springer, 1998.

[OG87] O. Goldreich. Towards a Theory of Software Protection and simulation by Oblivious RAMs. In *STOC 87*, 1987.

[IKOS04] Y. Ishai, E. Kushilevitz, R. Ostrovsky, A. Sahai. Batch Codes And Their Applications. In *STOC 2004*, pages 262-271. ACM Press, 2004.

[IP07] Y. Ishai, A. Paskin. Evaluating branching programs on encrypted data. In *4th Theory of Cryptography Conference (TCC'07)*, vol. 4392 of *Lecture Notes in Computer Science*, pages 575-594. Springer, 2007.

[TI99] T. Itoh. Efficient private information retrieval. In *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, E82-A(1):11-20, 1999.

[KO97] E. Kushilevitz, R. Ostrovsky. Replication is Not Needed: Single Database, Computationally-Private Information Retrieval. In *FOCS 1997*, pages 364-373. IEEE Computer Society, 1997.

[KT00] J. Katz, 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 8086. 2000.

[HL05] H. Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In J. Zhou and J. Lopez, editors, *ISC 2005*, vol. 3650 of *Lecture Notes in Computer Science*, pages 314-328. Springer, 2005.

[HL09] H. Lipmaa. First CPIR Protocol with Data-Dependent Computation. In D. Lee and S. Hong, editors, *ICISC 2009*, vol. 5984 of *Lecture Notes in Computer Science*, pages 193-210. Springer, 2009.

[HL11] H. Lipmaa. Efficient Multi-Query CPIR from Ring-LWE. 2011.

[DEM54] D. E. Muller. Application of boolean algebra to switching circuit design for error detection. In *IEEE Transactions on Computers*, EC-3, pages 6-12, 1954.

[LT04] L. Trevisan. Some applications of coding theory in computational complexity. In *Quaderni di Matematica*, vol. 13, pages 347-424, 2004.

[ISR54] I. S. Reed. A class of mutiple-error-correcting codes and the decoding scheme. In *Trans. I.R.E., Prof. Group on Information Theory No. 4*, pages 38-49, 1954.

[RS60] I. S. Reed, G. Solomon. Polynomial codes over certain finite fields. In *J. SIAM* vol. 8, 2 pages 300-304, 1960.

[DW08] D. Woodruff. Corruption and Recovery-Efficient Locally Decodable Codes. In *APPROX-RANDOM*, pages 584595, 2008.

[SY07] S. Yekhanin. Locally Decodable Codes and Private Information Retrieval Schemes. Ph.D. thesis at Massachusetts Institute of Technology, 2007.