A short introduction to PIR and batch codes

Henk D.L. Hollmann *

February 25, 2022

Abstract

We present a short introduction to PIR and batch codes and their applications.

*Institute of Computer Science, University of Tartu, Tartu 50409, Estonia (henk.d.l.hollmann@ut.ee)

1 Introduction

Binary PIR and batch codes both encode blocks of data bits into code words and both facilitate the recovery from a code word of multiple data bits simultaneously, by inspecting disjoint parts of the code word. PIR codes are designed to handle multiple requests for the same data bit, while batch codes can handle requests for arbitrary sequences of data bits. t-PIR codes have applications for example in Private Information Retrieval (PIR), where they allow to reduce the storage overhead in t-server PIR schemes by emulating the t servers. Batch codes are employed for example in network switches for load-balancing purposes. Typically, problems on PIR and batch codes need techniques from combinatorics, linear algebra, and coding theory. The field is relatively new, so there are lots of interesting questions to work on, and there are also opportunities to write smart software for explorative purposes, or to survey parts of the literature. Research problems concern for example the understanding of the known examples of such codes and using these insights to generalize known examples or to investigate variants of these codes for scenarios where some parts of the data are in higher demand than other parts. For non-binary or for nonlinear PIR or batch codes, not much, respectively almost nothing is known.

2 Finite fields and codes

In this short introduction, we consider elements $v = (v_1, \ldots, v_n)$ of $\{0,1\}^n$ as *vectors*, where all the usual computations on vectors now done modulo 2. That is, we consider such elements as vectors over the *finite field* $\mathbb{F}_2$ consisting of 0,1, with addition and multiplication modulo 2. The *inner product* (a, b) of two vectors $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ in $\{0,1\}^n$ is defined by $(a, b) = a_1b_1 + \cdots + a_nb_n$ (again with all operations modulo 2). We will use e_i to denote the i-th *unit vector*, having a 1 in position i and a 0 in the other positions.

A *code* of *length* n is a subset of $\mathbb{F}_2^n$; the code is *linear* of *dimension* if it is a k-dimensional *linear subspace* of $\mathbb{F}_2^n$. A *generator matrix* of a linear k-dimensional code is a $k \times n$ matrix G with entries in $\mathbb{F}_2$ such that the rows of G form a *basis* for C. Such a generator matrix serves as an *encoder* for C, where a vector of k data bits $a = (a_1, \ldots, a_n)$ is encoded into the code word $c = aG$. We say that a generator matrix is *systematic* if (possibly after a permutation of the columns) it is of the form (I_kP), where I_k denotes the $k \times k$ *identity matrix*. Note that when a systematic generator matrix is used as an encoder, then the data vector a encodes into a code word of the form $c = aG = (a_1, \ldots, a_k, c_{k+1}, \ldots, c_n)$, so the data itself appears within the code word.

3 PIR-codes

In the classical model for Private Information Retrieval (PIR), a user wishes to extract one bit of information from a database stored on a set of t servers in such a way that

no individual server gains any information on which bit the user was interested in. A *t-PIR-code* can be used to implement a given (linear) t-server PIR scheme with less storage overhead than the original scheme, where the t-PIR code is employed to emulate the t servers involved in the t-server PIR scheme [1], [2]. See also [7] for a nice introduction to this subject.

Example 3.1 A 2-server PIR scheme.

Servers S_1 and S_2 both store the data record $x = (x_1, ..., x_n)$ in $\{0,1\}^n$. To retrieve a bit x_i, choose a uniformly random query vector q in $\{0,1\}^n$, uniformly at random, and send q to S_1 and $q' = q + e_i$ to S_2. A server responds to a query by returning the inner product of the query with the data it holds. So we get back $r_1 = (q, x)$ from S_1 and $r_2 = (q', x)$ from S_2. Now we compute

$$r = r_1 + r_2 = (q, x) + (q', x) = (q + q', x) = (e_i, x) = x_i$$

to retreive bit x_i. Note that by the uniform randomness of q, servers S_1 and S_2 both learn nothing about in which bit we were interested. □

Example 3.2 To reduce the storage overhead of the 2-server PIR scheme in Example 3.1, we can use the 2-PIR code with generator matrix

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

To use this code, partition the data x as $x = (x^{(1)}, x^{(2)})$, and store x on three servers T_1, T_2, T_3, where T_1 stores $x^{(1)}$, T_2 stores $x^{(2)}$, and T_3 stores $x^{(1)} + x^{(2)}$. Suppose we want to retreive bit x_i with $x_i = x_j^{(1)}$. We have two copies of $x^{(1)}$ available, one on T_1, and one by combining the data on T_2 and T_3. So send q to T_1 and $q + e_j$ to both T_2 and T_3. We receive back

$$r_1 = (x^{(1)}, q), \qquad r_2^{(2)} = (x^{(2)}, q + e_j), \qquad r_3^{(2)} = (x^{(1)} + x^{(2)}, q + e_j).$$

Now by combining the information received from T_2 and T_3, we obtain

$$r_2 = r_2^{(2)} + r_3^{(2)} = (x^{(2)}, q + e_j) + (x^{(1)} + x^{(2)}, q + e_j) = (x^{(1)}, q + e_j);$$

and we are in the position to obtain bit $x_i = x_j^{(1)}$ as

$$r = r_1 + r_2 = (x^{(1)}, q) + (x^{(1)}, q + e_j) = (x^{(1)}, e_j) = x_j^{(1)} = x_i.$$

It should by now be clear what to do to retreive a bit x_i when the i-th bit is contained in the second part $x^{(2)}$ of x, that is, if $x_i = x_j^{(2)}$.

Note that this magic could not be done with a non linear 2-PIR code: if the reconstruction of x_1 from the server content c_2 of T_2 and c_3 of T_3 Is not linear, then there seems to be no way to compute an inner product (q, x_1) from the inner products (q, c_2) and (q, c_3), so it is not sufficient for the servers T_2 and T_3 to simply execute the PIR scheme protocol for server S_2. □

4 Batch codes

A *t-batch code* is a method to store a data record in encoded form on multiple servers in such a way that the bit-values in any batch of t positions from the record can be retrieved by decoding the bit-values in t disjoint groups of positions. Batch codes were initially introduced in [3] as a method to improve load-balancing in distributed data storage systems. Later, so-called *switch codes*, a special type of batch codes, were proposed in [8] as a method to increase the throughput rate in network switches.

Example 4.1 Here we present an example of the use of a batch code in a switch, see Figure 1 (figure and example taken from [8]).

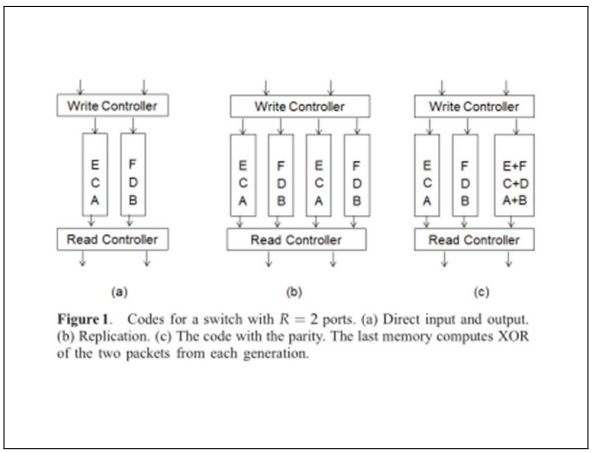

Figure 1. Codes for a switch with $R = 2$ ports. (a) Direct input and output. (b) Replication. (c) The code with the parity. The last memory computes XOR of the two packets from each generation.

Figure 1: A switch with data uncoded, replicated, and batch-encoded

In every clock-cycle, two bits of incoming data are written on two servers (left), and two users attempt to read a bit of data. However, each server can only deliver one bit per clock-cycle. So if both users attempt to read the same data, only one request can be granted. One solution to this problem is *replication*: simply employ four servers (middle), and write each bit on two servers. A smarter solution is to use a 2-batch code with generator matrix

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix},$$

employing just 3 servers (right). Now, every request of the users can be served. Indeed, if user 1 requests bit A and user 2 requests bit B, then they just read it form the first two servers; if both user 1 and user 2 request bit A, then user 1 reads A from the first server and user 2 reads B and $A + B$ from the second and third server, and simply *computes* A. In this way, storage overhead is reduced, at the cost of some additional communication and some computations. □

5 Definitions of PIR- and batch-type codes

We now give a precise mathematical definition of PIR- and batch-type codes. This section can be skipped on first reading, but can be consulted when a certain notion is in doubt. For later use, we initially keep our definitions slightly more general, later we specify these definitions to the cases of PIR and batch codes. First, we need the notion of an *encoder*. Let Σ_q denote an alphabet consisting of q distinct letters. Often,we let $\Sigma_q = \{0, 1, \ldots, q-1\}$, but any set of size q will do. If q is a prime-power, then often we identify the alphabet with the set of elements of the finite field $\mathbb{F}_q$ of q elements, and for integers $n \geq 1$, we identify the set $\mathbb{F}_q^n$ with the vector space of dimension n over $\mathbb{F}_q$.

Definition 5.1 A q-ary k-to-n *encoder* is a on-to-one map $\epsilon : \Sigma_q^k \to \Sigma_q^n$. A *decoder* for the encoder ϵ is a map $\delta : \Sigma_q^n \to \Sigma_q^k$ such that if $\epsilon(a) = c$, then $\delta(c) = a$. The encoder is said to be *linear* if q is a prime power, $\Sigma_q = \mathbb{F}_q$, and ϵ is an $\mathbb{F}_q$-linear map on $\mathbb{F}_q^k$. In that case, the encoder ϵ can be identified with a $k \times n$ *generator matrix* G over $\mathbb{F}_q$, and the encoder map takes the form $\epsilon : a \to aG$ for all vectors $a \in \mathbb{F}_q^k$.

PIR- and batch-type codes are characterized by the property that given the encoded data, certain types of *simultaneous* requests for specific data symbols can each be handled by reading and decoding data from a set of positions called a *recovery set*, where there is limited overlap of these recovery windows. The next definition makes this precise. First we recall some terminology. Let $I = \{i_1, \ldots, i_s\}$ be a subset of $\{1, \ldots, n\}$ with $i_1 < \cdots < i_s$. If $c = (c_1, \ldots, c_n) \in \Sigma_q^n$, then the restriction c_I of c to the positions in I is the word $(c_{i_1}, \ldots, c_{i_s})$.

Definition 5.2 Let $\epsilon : \Sigma_q^k \to \Sigma_q^n$ be an encoder. We say that a subset I of $\{1, \ldots, n\}$ is a *recovery set* of ϵ *for the i-th data symbol* if for every $a \in \Sigma_q^k$, the i-th symbol a_i of a is uniquely determined by the restriction c_I of the code word $c = \epsilon(a)$; that is, if there exists a *decoding function* $\delta_{I,i} : \Sigma_q^{|I|} \to \sigma_q$ such that if $c = \epsilon(a)$, then $\delta_{I,i}(c_I) = a_i$.

Definition 5.3 A *query* for an encoder $\epsilon : \Sigma_q^k \to \Sigma_q^n$ is a sequence $i_1, \ldots, i_t$ of (not necessarily distinct) elements of $\{1, \ldots, k\}$. Given a code word $c = \epsilon(a)$, the query $i_1, \ldots, i_t$ should be considered as a request to obtain the data symbols $a_{i_1}, \ldots, a_{i_t}$.

Definition 5.4 Let $\epsilon : \Sigma_q^k \to \Sigma_q^n$ be an encoder. We say that a sequence of subsets $I_1, \ldots, I_t$ of $\{1, \ldots, n\}$ *serves* a query $i_1, \ldots, i_t$ for ϵ if for every $j = 1, \ldots, t$, the set I_j is a recovery set of ϵ for the i_j-th data symbol. We say that $I_1, \ldots, I_t$ serve the query $i_1, \ldots, i_t$ for ϵ *with width* w and *multiplicity* μ if $|I_j| \leq w$ for all j and if every $i \in \{1, \ldots, n\}$ occurs in at most μ of the sets I_j.

Now we are ready for a definition of batch-type codes.

Definition 5.5 Let $\epsilon : \Sigma_q^k \to \Sigma_q^n$ be an encoder, let $\mathcal{Q}$ be a collection of queries for ϵ, and let w, μ be positive integers. We say that ϵ is a $(\mathcal{R}, w, \mu)$-*batch code* if ϵ can serve every

query from $\mathcal{Q}$ with width at most w and multiplicity at most μ. If no constraint is posed on the width (or on the multiplicity) then we set $w = \infty$ (or $\mu = \infty$). Similarly, we say that a code $C = \Sigma_q^n$ is a $(\mathcal{R}, w, \mu)$-batch code if it has an encoder that is a $(\mathcal{R}, w, \mu)$-batch code.

In particular, a t-PIR code is is a $(\mathcal{Q}, \infty, 1)$ code where $\mathcal{Q}$ consists of all sequences $i, i, \ldots, i$ (t times), and a t-batch code is a $(\mathcal{Q}, \infty, 1)$ where $\mathcal{Q}$ consists of all sequences $i_1, \ldots, i_t$ of (not necessarily distinct) elements from $\{1, \ldots, n\}$.

A (linear) *functional* t-batch code is an encoder that can serve any sequence of t requests for *arbitrary linear combinations* of the data symbols with mutually disjoint recovery sets.

We end this section with a discussion of a relation between the strength of a code as a PIR-code and its minimal Hamming distance. Recall that the *Hamming distance* between two vectors v, w is the number of coordinates in which v and w differ; the minimum (Hamming) distance of a code is the minimal Hamming distance between two distinct code words.

In what follows, an $(n, M, d)_q$-code C is a subset of Σ_q^n of size M, with minimum Hamming distance d. For q a prime power, an $[n, k, d]_q$ code is an $\mathbb{F}_q$-*linear* code of length n and dimension k over the finite field $\mathbb{F}_q$, with minimum distance d. Part of the research into batch-type codes is motivated by the following simple result.

Theorem 5.6 Let C be a (possibly nonlinear) $(n, q^k, d)_q$-code, Suppose furthermore that C has an encoder $\epsilon : \mathbb{F}_q^k \to C$ that turns C into a (t, ∞, μ)-PIR code. Then $\lceil t/\mu \rceil \leq d$.

Proof. Let $c^{(1)} = \epsilon(a^{(1)})$ and $c^{(2)} = \epsilon(a^{(2)})$ be distinct code words from C. Then there is an i such that $a_i^{(1)} \neq a_i^{(2)}$. By our assumption, there are sets $I_1, \ldots, I_t$ that serve the query $i, \ldots, i$ with multiplicity at most μ. So for every set I_j, the restrictions $c_{I_j}^{(1)}$ and $c_{I_j}^{(2)}$ determine distinct data symbols, hence I_j must contain a position i_j for which $c_{i_j}^{(1)} \neq c_{i_j}^{(1)}$. By the multiplicity condition, every position is contained in at most μ of the sets I_j; hence there must be at least $\lceil t/\mu \rceil$ distinct positions among $i_1, \ldots, i_t$, and as a consequence, $c^{(1)}$ and $c^{(2)}$ differ in at least $\lceil t/\mu \rceil$ positions. Since the code words were arbitrary, we conclude that $d \geq \lceil t/\mu \rceil$. □

6 Some further examples

A PIR-code can be a complicated thing of beauty. Here is an example based on 2-designs.

Example 6.1 The 10 subsets

$$\{0,1,2\}, \{0,2,3\}, \{0,1,4\}, \{1,2,5\}, \{0,3,5\}, \{2,3,4\}, \{0,4,5\}, \{1,3,4\}, \{1,3,5\}, \{2,4,5\}$$

of $\{0, 1, 2, 3, 4, 5\}$ form a $2-(v, k, \lambda)$ design with $v = 6$, $k = 3$, and $\lambda = 2$: it is a collection of $b = 10$ subsets of size $k = 3$ (the "blocks") of a set of size $v = 6$ ("the points") with the

property that every pair of points is contained in precisely $\lambda = 2$ blocks. Suppose that a certain point p is contained in r_p blocks. By counting pairs (q, B) of points $q \neq p$ and blocks B for which $p, q \in B$ in two different ways, we see that

$$(v-1)\lambda = r_p(k-1),$$

hence $r_p = \lambda(v-1)/(k-1)$ does not depend on the particular point p. So every point is on r blocks, for some constant r. Next, by counting pairs (p, B) of points p and blocks B with $P \in B$ in two different ways, we obtain that $bk = vr$, hence $b = vr/k = \lambda v(v-1)/k(k-1)$.

Construct a matrix G that has the form $G = (I_6 A)$ with A the 6×10 *incidence matrix* of the design: we index the row with the points, the columns with the blocks, and we let $A_{v,B} = 1$ precisely when $v \in B$, and $A_{v,B} = 0$ otherwise. So we find

$$G = \begin{pmatrix} 1 & & & & & 1 & 1 & 1 & & 1 & & 1 & & & & \\ & 1 & & & & 1 & & 1 & 1 & & & & 1 & 1 & & \\ & & 1 & & & 1 & 1 & & 1 & & 1 & & & & 1 & \\ & & & 1 & & & 1 & & & 1 & 1 & & 1 & 1 & & \\ & & & & 1 & & & 1 & & & 1 & 1 & 1 & & 1 & \\ & & & & & 1 & & & 1 & 1 & & 1 & & 1 & 1 & \end{pmatrix}$$

It can be checked that G is the generatormatrix of a 6-dimensional 6-PIR code of length 16. Indeed, we see for example that the collection containing the single set $\{0\}$ and the collections of three sets $\{1\}, \{0, 3, 5\}, \{1, 3, 5\}$; $\{2\}, \{0, 4, 5\}, \{2, 4, 5\}$; $\{3\}, \{0, 1, 4\}, 1, 3, 4\}$; $\{4\}, \{0, 2, 3\}, \{2, 3, 4\}$; $\{5\}, 0, 1, 2\}, \{1, 2, 5\}$ all have the property that 0 occurs in the union and all other elements occur twice in the sets in the collection.

It is a nice problem to try to figure out what properties of a $2-(v, k, \lambda)$ design are needed to obtain a PIR code in a similar way as here. □

And here is an example of a batch code based on a *simplex code*.

Example 6.2 Consider the 3×7 binary matrix G that has as its columns all the nonzero binary vectors of length 3. So

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

This matrix can be used to encode a datavector $a = (a_2, a_1, a_0)$ into a code word

$$c = aG = (a_0, a_1, a_0 + a_1, a_2, a_2 + a_0, a_2 + a_1, a_2 + a_1 + a_0).$$

It can be checked that every request $a_{i_1}, a_{i_2}, a_{i_3}$ for three of the data symbols can be served by three disjoint set of column positions I_1, I_2, I_3, where a_{i_j} is recovered by adding the code word bits (modulo 2) in the positions of I_j. for example, the request a_0, a_0, a_0 can be served by the recovery sets $I_1 = \{1\}, I_2 = \{2, 3\}, I_3 = \{4, 5\}$ and the request a_0, a_0, a_1 by the recovery sets $I_1 = \{1\}, I_2 = \{4, 5\}, I_3 = \{2\}$. Moreover, for such requests, the size of the recovery sets can always be chosen to be 1 or 2.

Again, it is a nice problem to figure out what exactly makes this example work. □

7 Some research questions and/or project proposals

Here we give a brief summary of a number of research topics, and we propose some research questions and some concrete research problems.

1. Find systematic constructions for known examples of binary linear PIR and batch codes, see, e.g., [1, 2], [10, 9], [4] for a large amount of PIR codes described just by a computer-generated generator matrix, and construct new examples.

2. Find lower bounds for $P_q(k,t)$ and $B_q(k,t)$, the shortest length of a q-ary k-dimensional t-PIR and t-batch code, respectively. Except for the lower bound implicit in Theorem 5.6 and the lower bound in [4] resulting from the bound in [5] for the binary case, almost nothing is known here. The lower bounds for $P_2(k,3)$ and $P_2(k,4)$ in [5] are in fact tight (for the statement of these bounds, see also point 7 below). It would be desirable to find a better proof. There are also some generalizations of techniques known for locally repairable codes, see, e.g., [6], but these are only good when q is large. It would be interesting to try to find other lower bounds.

3. Find further values of $P_q(k,t)$ and $B_q(k,t)$; this needs both lower bounds and constructions. For tables for $q = 2$, see [4]; for $q > 2$, (almost) nothing is known. This would probably also require some programming work to find examples of good codes.

4. Find interesting good *nonlinear* PIR or batch codes, that is, with better parameters than any linear code. In fact, although there must exist very many, there is not a single example known! It would be nice to find such a code with small parameters. One approach would be to exclude possible candidate sets of parameters. Some initial work in this direction has just started.

5. The *strength* of a code as a PIR or a batch code is the largest t for which the code is a t-PIR or a t-batch code. It can sometimes be quite tricky to determine the strength of a given error-correcting code when used as a batch code. An interesting problem could be to take a specific linear or nonlinear code and investigate its strength. Since some of the good nonlinear codes are best understood as codes over $\mathbb{Z}_4$, that would also require understanding of codes over non-finite-field alphabets such as $\mathbb{Z}_4$. Depending on your taste, you could investigate some of these problems.

6. We know that a t-PIR code has minimum Hamming distance at least t; this bound can sometimes be attained and it would be nice to know more examples of these so-called distance-optimal PIR-codes. No example of a nonlinear PIR or batch code is known that beats the existing linear ones; it should be possible to find an example of reasonably small size.

7. Generalize the bounds for $P_2(k,3)$ from [5] to nonlinear codes. For *linear* 3-PIR codes, the bound states that

$$P_2(k,3) \geq k + \min\{r \mid r(r-1) \geq 2k\},$$

and in fact

$$P_2(k,3) = k + \lceil (1+\sqrt{1+8k})/2 \rceil,$$

and in addition, if t is even, then $P_2(k,t) = P_2(k,t-1)+1$. This may be quite difficult.

8. In the theory of error-correcting-codes, there exist so-called unequal-error-protecting codes. In analogy, we could investigate PIR or batch codes especially designed for cases where some parts of the data are more in demand, more popular, than other parts; can we do better than just using a combination of two codes? This question is completely new. It should be quickly possible to discover at least examples where something better can be done than combining two codes, or showing that combining codes is always optimal. *A separate document is now available in which this problem is discussed in more detail.*

9. A $k \times n$ generator matrix is *systematic* if the columns of G can be permuted so that the first k columns form the $k \times k$ identity matrix. There is the question if there exists a code with a non-systematic generator matrix that gives a stronger PIR or batch code than any systematic generator matrix for that code. This has recently been solved for PIR-codes: the answer is yes. However, the proof is by computer and exhaustive search. We could try to find a methodical construction, or at least try to understand why this worked. For batch codes, this problem is still unsolved.

10. No encoder (linear or nonlinear) for the binary linear code with length $n = 2^k - 1$ and dimension $2^k - 1 - k$ can produce a 3-PIR code for $k \geq 3$. Linear codes with these parameters are unique (these are the binary Hamming codes), but for $k > 3$ there exist also nonlinear codes with these parameters. Can such a code ever be a 3-PIR code?

11. For possibly nonlinear $(n, q^k, d)_2$ codes with locality r (essentially systematic possibly nonlinear codes where every information symbol can be recovered from r other positions) there is a generalization of the Singleton bound stating that $n \geq k + (k/r) + d - 2$. Is it possible to generalize such bounds to the case of possibly nonlinear PIR or batch codes? This should certainly be possible for systematic nonlinear codes.

All of these problems could be discussed in much more detail, but to keep the size of this document within reasonable bounds, the reader is referred to the author for more details.

Acknowledgments

This research was supported by the Estonian Research Council grant PRG49.

References

[1] A. Fazeli, A. Vardy, and E. Yaakobi. Codes for distributed PIR with low storage overhead. In *Proc. IEEE Symp. Information Theory (ISIT)*, pages 2852–2856, Hong Kong, 2015.

[2] A. Fazeli, A. Vardy, and E. Yaakobi. PIR with low storage overhead: coding instead of replication, 2015. [Online] Available: https://arxiv.org/abs/1505.06241.

[3] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Batch codes and their applications. In *Proc. 36th ACM Symposium on Theory of Computing (STOC), Chicago*, pages 1057–1061, June 2004.

[4] S. Kurz and E. Yaakobi. PIR codes with short block length. *Des. Codes, Cryptogr.*, 89:559–587, June 2021.

[5] S. Rao and A. Vardy. Lower bound on the redundancy of PIR codes, 2017. [Online] Available: http://arxiv.org/abs/1605.01869.

[6] V. Skachek. Batch and PIR codes and their connections to locally repairable codes. In M. Greferath, M. O. Pavčević, N. Silberstein, and M. Ángeles Vázquez-Castro, editors, *Network Coding and Subspace Designs*, pages 427–442. Springer, 2018.

[7] A. Vardy. Private Information Retrieval: Coding instead of Replication. Talk at the Institate Henri Poincaré, March 25, 2016. [Online] Available: `https://www.youtube.com/watch?v=WU2-6Da8IyE&t=934s`.

[8] Z. Wang, O. Shaked, Y. Cassuto, and J. Bruck. Codes for network switches. In *Proc. 2013 IEEE International Symposium on Information Theory (ISIT)*, pages 1057–1061, Istanbul, 2013.

[9] Y. Zhang, T. Etzion, and E. Yaakobi. Bounds on the length of functional PIR and batch codes. *IEEE Trans. on Inform. Theory*, pages 4917–4934, 2020.

[10] Y. Zhang, E. Yaakobi, and T. Etzion. Bounds on the length of functional PIR and batch codes. In *Proc. 2019 IEEE International Symposium on Information Theory*, pages 2129–2133, Paris, 2019.