Bounds on the Length of Functional PIR and Batch Codes

Yiwei Zhang, Tuvi Etzion, *Fellow, IEEE*, and Eitan Yaakobi, *Senior Member, IEEE*

***Abstract*— A *functional k-Private Information Retrieval* (k-PIR) *code* of dimension s consists of n servers storing linear combinations of s linearly independent information symbols. Any linear combination of the s information symbols can be recovered by k disjoint subsets of servers. The goal is to find the minimum number of servers for given k and s. We provide lower bounds on the minimum number of servers and constructions which yield upper bounds on this number. For $k \leq 4$, exact bounds on this number are proved. Furthermore, we provide some asymptotic bounds. The problem coincides with the well known PIR problem based on a coded database to reduce the storage overhead, when each linear combination contains exactly one information symbol. If any multiset of size k of linear combinations from the linearly independent information symbols can be recovered by k disjoint subset of servers, then the servers form a *functional k-batch code*. A functional k-batch code is a functional k-PIR code, where all the k linear combinations in the multiset are equal. We provide some bounds on the minimum number of servers for functional k-batch codes. In particular we present a random construction and a construction based on simplex codes, Write-Once Memory (WOM) codes, and Random I/O (RIO) codes.**

***Index Terms*— Private Information Retrieval (PIR) codes, batch codes, distributed storage codes.**

I. Introduction

A. General Background

A Private Information Retrieval (PIR) protocol allows a user to retrieve a data item from a database, in such a way that the servers storing the data will get no information about which data item was retrieved. The problem was introduced in [8], [9]. The protocol to achieve this goal assumes that the servers are curious but honest, so they don't collude. It is also assumed that the database is error-free and is synchronized all the time. For a set of k servers, the goal is to design an efficient k-server PIR protocol, where efficiency is measured by the total number of bits transmitted by all parties involved. This model is called *information-theoretic* PIR; there is also *computational* PIR, in which the privacy is defined in terms of the inability of a server to compute which item was retrieved in a reasonable time [22]. We continue to consider only the information-theoretic PIR.

The area of PIR was very active in the last twenty years and a survey on recent developments can be found in a recent paper [3]. While most of the work in this area is theoretical, there have been notable recent advances in bridging the gap between theory and practice. For example, the recent paper [17] reports on the design and implementation of a scalable and private media delivery system — called *Popcorn* — that explicitly targets Netflix-like content distribution. Another practical system for private queries on public datasets — called *Splinter* — is currently in development [41]. This system has been reported to achieve latencies below 1.20 seconds for realistic workloads including a Yelp clone, flight search, and map routing.

The classic model of PIR assumes that each server stores a copy of an s-bit database, so the *storage overhead*, namely the ratio between the total number of bits stored by all servers and the size of the database, is k. However, recent work combines PIR protocols with techniques from distributed storage (where each server stores only a coded fraction of the database) to reduce the storage overhead. This approach was first considered in [33], and several papers have developed this direction further, e.g. [2], [11], [12]. Our discussion on PIR will follow the breakthrough approach presented in [18], [19], which shows that n servers (for some $n > k$) may emulate a k-server PIR protocol with storage overhead significantly lower than k. The scheme used for this purpose is called a *k-PIR* and will be discussed in the next paragraph.

The s-bit database $\mathcal{S}$ is considered as the information bits of a linear code of length n and dimension s. This code has an $s \times n$ generator matrix $\mathbf{G}$. The linear combinations related to the codeword $\mathcal{S}\mathbf{G}$ are stored in the n servers. In other words, the i-th server stores the linear combination generated when the s-bit information word is multiplied by the i-th column of $\mathbf{G}$. The generator matrix $\mathbf{G}$ represents a k-PIR scheme if there are k pairwise disjoint subsets of $[n] \triangleq \{1, 2, \ldots, n\}$, $R_1, R_2, \ldots, R_k$, such that the sum of the columns of $\mathbf{G}$ related to each such subset is the data item (out of the s data items) which the user wants to retrieve. Using these k subsets any known k-PIR protocol can be emulated with the given n servers. The advantage of this scheme is a smaller amount

Manuscript received April 15, 2019; revised December 25, 2019; accepted February 14, 2020. Date of publication March 2, 2020; date of current version July 14, 2020. The work of Yiwei Zhang and Eitan Yaakobi was supported by the ISF Grant 1817/18. The work of Yiwei Zhang and Tuvi Etzion was supported by the BSF-NSF Grant 2016692. The work of Yiwei Zhang was supported by a Technion Fellowship. This work was supported by the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel Cyber Directorate. This article was presented in part at the 2019 IEEE International Symposium on Information Theory (ISIT). *(Corresponding author: Tuvi Etzion.)*

Yiwei Zhang was with the Department of Computer Science, Technion—Israel Institute of Technology, Haifa 3200003, Israel. He is now with the School of Cyber Science and Technology, Shandong University, Qingdao 266237, China, and also with the Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao 266237, China (e-mail: ywzhang@sdu.edu.cn).

Tuvi Etzion and Eitan Yaakobi are with the Department of Computer Science, Technion—Israel Institute of Technology, Haifa 3200003, Israel (e-mail: etzion@cs.technion.ac.il; yaakobi@cs.technion.ac.il).

Communicated by L. Dolecek, Associate Editor for Coding Techniques.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2020.2977631

of storage used for a k-PIR protocol. The goal in the design of such a PIR scheme is to find the smallest n, given s and k. This problem was considered in several papers, e.g. [1], [18], [19], [24], [28], [36], [42].

In all the PIR protocols known in the literature, the user wants to retrieve one out of the s information bits of the database. As will be described in the sequel, PIR codes and their generalizations are similar to other concepts in coding theory. For example, the requirements are very similar to the ones in codes with availability [30], which are important in applications of distributed storage codes. Recently, the study of PIR was further extended to the so-called private function retrieval or private computation [25], [29], [35], which are privacy algorithms for distributed computing schemes. In such schemes, a user (a party involved in a distributed computing task) requests a linear combination of the information symbols in order to perform its computing task. Therefore, it is quite natural to generalize the concept of PIR code to the setting when it is required to retrieve a linear combination of the s bits of information symbols. Such a scheme will be called a *k-functional PIR code* (this is some abuse of definition since for the private information retrieval application such a retrieval of linear combinations is not required). Given s and k we would like to find the smallest n for which a functional k-PIR exists. This is one of the two targets of the current paper.

The definition of a k-PIR code appears to be a special case of a k-batch code. The concept of a batch scheme was first proposed by Ishai et al. [21], which was motivated by different applications for load-balancing in storage and cryptographic protocols. Originally, batch codes were defined in a very general form, i.e., s information symbols are encoded into n-tuples of strings where each string is called a bucket. Each bucket contains a few linear combinations of the information symbols. A single user wants to retrieve a batch of k distinct data items (out of the s data items) by reading at most t symbols from each bucket. The goal in the design of a batch scheme is to find the smallest total length of all the buckets, given s, k, t and n.

A stronger variant of batch codes [21] is intended for a multi-user application instead of a single-user setting, known as the *multiset batch codes*. In this variant we have k different users each requesting a data item, where some of the requests are allowed to be the same. Therefore all the k requests constitute a multiset of data items (each being one out of the s data items, replications allowed). Moreover, each bucket is allowed to be accessed by at most one user. A special case of a multiset batch code is when each bucket contains only one symbol. This model is called a *primitive multiset batch code* [21] (or a *k-batch code* in short) and it is a family of batch codes that was most studied in the literature. In the rest of this paper, we restrict our definition of batch codes only to primitive multiset batch codes. Similarly as for a PIR code, a batch code is represented by an $s \times n$ generator matrix $\mathbf{G}$. It is a k-batch scheme if there are k pairwise disjoint subsets of $[n]$, $R_1, R_2, \ldots, R_k$, such that the k sums from each subset of the columns in $\mathbf{G}$ constitute a multiset of data items which some k users want to retrieve. Hence, the requests in a k-PIR are special cases of the requests in a k-batch when the multiset contains only one specific item k times. Therefore a k-batch code can always work as a k-PIR code but not vice versa. The goal in the design of a batch scheme is to find the smallest n, given s and k. This problem was considered in several papers, e.g. [1], [6], [21], [31], [37].

Similarly as our generalization of PIR into functional PIR, by setting the requests to be a multiset of linear combinations of the s bits of information symbols, a batch code is generalized into a *functional batch code*. Given s and k we would like to find the smallest n for which a functional k-batch code exists. This is the second target of the current paper.

A special case of batch codes, called *switch codes*, were recently studied for network applications [6], [7], [38]–[40]. This family of codes was first proposed by Wang et al. [40] and these codes were designed to increase the parallelism of data writing and reading processes in network switches. A network switch is required to write n incoming packets and read k outgoing packets while using m memory banks, each able to write and read one packet per time unit. Each set of n packets written to the switch simultaneously is called a generation. The objective is to store the packets in the banks such that every request of k packets, which can be from previous generations, can be handled by reading at most one packet from each bank. Even though batch codes and switch codes were proved to be equivalent [6], switch codes are commonly designed for the special case of $k = n$, which balances the output and input switching rates.

A related family of codes to functional batch codes is the family of *random I/O (RIO) codes*. This family of codes was recently introduced by Sharon and Alrod [34] and provides a coding scheme to improve the random input/output performance of flash memories. An (n, M, t) RIO code stores t pages in n cells with $t + 1$ levels such that it is enough to sense a single read threshold in order to read any of the t pages. Sharon and Alrod showed in [34] that the design of RIO codes is equivalent to the design of *write-once memory* (*WOM*) *codes* [13], [20], [32], [43]. The latter family of codes attracted substantial attention in recent years in order to improve the lifetime of flash memories by allowing writing multiple messages to the memory without the need for an erase operation. However, while in WOM codes, the messages are received one after the other and thus are not known in advance, in RIO codes the information of all logical pages can be known in advance when programming the cells. This variant of RIO codes, called *parallel RIO codes*, was introduced in [44]. A recent construction of parallel RIO codes [45] used the coset coding scheme [13] with Hamming codes in order to construct parallel RIO codes. In fact, this construction is equivalent to the requirements of functional batch codes, and thus every functional batch code can be used as a parallel RIO code as well. The other direction does not necessarily hold since parallel RIO codes do not have to be linear, as opposed to functional batch codes. The codes from [45] gave two constructions of functional batch codes (which are parallel RIO codes) with the following parameters: $(s = 3, k = 4, n = 7)$ and $(s = 4, k = 8, n = 15)$.

B. General Description of the Problem

Assume there are n servers, each one is storing a linear combination of s linearly independent items. Each of these s items will be called an *information symbol*. Each linear combination which consists of at least one of these information symbols will be called a *coded symbol*. There are k users who want to retrieve k linear combinations of items from these servers. Each such linear combination which a user wants to retrieve will be called a *request*. Each user has exactly one such request and he should approach a set of servers to obtain his request. The set of servers which are approached by two different users must be disjoint. We would like to know the minimum number of servers which is required to satisfy any k requests of the k users. This scheme will be called a *functional k-batch code* (functional k-batch for short, and similarly done for the related concepts). If each request contains exactly one information symbol, then the scheme will be called a *k-batch code*. If the k requests are the same (linear combination) then the scheme will be called a *functional k-PIR code* and furthermore if these k requests contain the same information symbol, then the scheme will be called a *k-PIR code*. The formal definitions of these four types of codes are as follows.

Definition 1: A *functional k-batch code* of length n and dimension s consists of n servers and s information symbols $\{x_1, x_2, \ldots, x_s\}$. Each server stores a nontrivial linear combination of the information symbols (which are the coded symbols), i.e. the j-th server stores a linear combination y_i, $1 \leq i \leq n$. For any request of k linear combinations $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ (not necessarily distinct) of the information symbols, there are k pairwise disjoint subsets $R_1, R_2, \ldots, R_k$ of $[n]$ such that the sum of the linear combinations in the related servers of R_j, $1 \leq j \leq k$, is $\mathbf{v}_j$, i.e. $\sum_{i \in R_j} y_i = \mathbf{v}_j$. Each such $\mathbf{v}_j$ will be called a *requested symbol* and each such subset R_j will be called a *recovery set*.

A functional k-batch code can be also represented by an $s \times n$ matrix $\mathbf{G}$ in which the j-th column has *ones* in positions $i_1, i_2, \ldots, i_\ell$ if and only if the j-th server stores the linear combination $x_{i_1} + x_{i_2} + \cdots + x_{i_\ell}$. Using this matrix representation, a functional k-batch code is an $s \times n$ matrix $\mathbf{G}$, such that for any k column vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ (not necessarily distinct), there are k pairwise disjoint subsets of columns in $\mathbf{G}$ such that the column vectors in each subset R_j sums up to the vector $\mathbf{v}_j$.

Definition 2: A *k-batch code* is defined similarly to a functional k-batch code, where each one of the requests $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ contains exactly one information symbol.

Definition 3: A *functional k-PIR code* is defined similarly to a functional k-batch code, where all the $\mathbf{v}_i$'s equal to one linear combination $\mathbf{v}$. A *k-PIR code* is defined similarly to a functional k-PIR code, where the linear combination $\mathbf{v}$ contains exactly one information symbol.

This definition for k-PIR coincides with the definition for k-PIR given in [18], [19] for a single user. Let $FB(s,k)$ ($B(s,k)$, $FP(s,k)$, $P(s,k)$, respectively) be the minimum number of servers required for s items and k requests for functional k-batch (k-batch, functional k-PIR, k-PIR, respectively).

By these definitions, a (functional) batch code is also a (functional) PIR code (where all the requests are equal) and a functional batch (functional PIR, respectively) code is also a batch (PIR, respectively) code, but not vice versa. In other words, a user which has a request for a PIR code can obtain his request from a related batch code, i.e., the set of batch codes is a subset of the PIR codes. The reason is that the user can ask, from a batch code, a request with multiple elements, where all the elements in the request are identical. On the other hand, there are PIR codes which cannot be used as batch codes since they are not designed to answer requests which contain different elements. Thus, we have the following relationships among these four families of codes, where $C_1 \longrightarrow C_2$ implies that the code C_1 can be used functionally as the code C_2. In the following Venn diagram $\mathbb{P}$ denote the set of k-PIR codes, $\mathbb{B}$ denote the set of k-batch codes, $\mathbb{FP}$ denote the set of functional k-PIR codes, and $\mathbb{FB}$ denote the set of functional k-batch codes.

C. Basic Results

Our goal in this paper is to obtain lower and upper bounds on $FB(s,k)$ and $FP(s,k)$, since relatively good bounds on $B(s,k)$ and $P(s,k)$ are known from the literature. Some of these bounds on $B(s,k)$ and $P(s,k)$ were derived in [1], [6], [18], [23], [28], [31], [37], [42] and are summarized as follows.

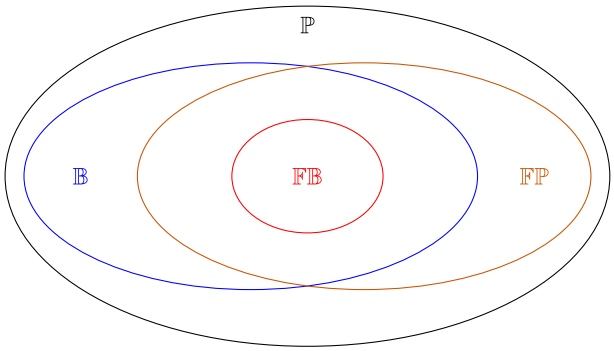
Lemma 4:

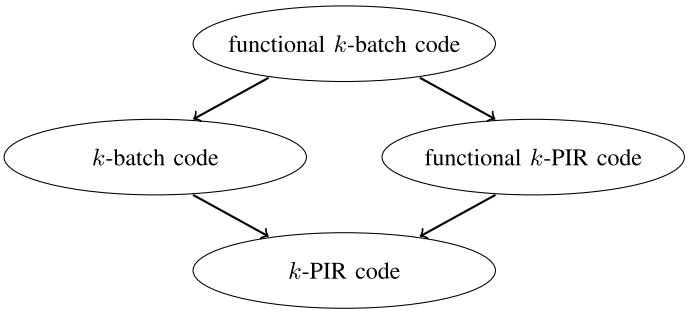
1) For each $s \geq 1$, $P(s, 2^{s-1}) = 2^s - 1$ [19].
2) For each $s \geq 1$, $B(s, 2^{s-1}) = 2^s - 1$ [40].
3) When k is a fixed integer, $P(s,k) = s + \Theta(\sqrt{s})$ [18], [28], [42].
4) $B(s,k) = s + \Theta(\sqrt{s})$ for $k = 3, 4, 5$ [1], [37].
5) $B(s,k) = s + O(\sqrt{s} \log s)$ for $k \geq 6$ [37].
6) $B(s, s^{1/3}) \leq 2s$ [31].
7) $B(s, s^{\varepsilon}) \leq s + s^{7/8}$ for $7/32 \leq \varepsilon \leq 1/4$ [31].
8) $B(s, s^{\varepsilon}) \leq s + s^{4\varepsilon}$ for $1/5 < \varepsilon \leq 7/32$ [31].
9) $B(s,s) \leq 2s^{1.5}$ [6].
10) $P(s, \sqrt{s}) = s + O(s^{(\log 3/2)})$ [23].
11) $P(s, s^{\varepsilon}) = s + O(s^{0.5+\varepsilon})$, $0 < \epsilon < 1/2$ [23].
12) $B(s, k = \Theta(s^{\varepsilon})) = s + o(s)$, $0 < \epsilon < 1$ [1].
13) $P(s, k = \Theta(s^{\varepsilon})) = s + o(s)$, $0 < \epsilon < 1$ [1].
14) $B(s, k = o(n^{1/3}/\log n)) = s + O(k^{3/2}\sqrt{n} \log n)$ [26], [27].
15) For $k < \frac{1}{\ell^2} n^{1/(2\ell+1)}$, ℓ is a positive integer, $B(s,k) = s + O(kn^{\frac{\ell+1}{2\ell+1}})$ [26], [27].

For a binary vector $\mathbf{v}$, let $\mathrm{supp}(\mathbf{v})$ denote the support of $\mathbf{v}$, i.e., the set of nonzero entries of $\mathbf{v}$. Some simple bounds on $FB(s,k)$ and on $FP(s,k)$ are derived in the following theorem.

Theorem 5: If s and k are positive integers, then

1) For $k > 1$, $FB(s,k) > FB(s,k-1)$.
2) For $k > 1$, $FP(s,k) > FP(s,k-1)$.
3) For $s \geq 1$, $FP(s,1) = FB(s,1) = s$.
4) For $s \geq 1$, $FP(s,2) = s + 1$.
5) For $s \geq 1$ and $k \geq 1$, $FP(s,2k) = FP(s,2k-1) + 1$.
6) For $s \geq 1$ and $k \geq 1$, $FP(s,k) \leq FB(s,k)$.





Proof:

1) If any server is removed from a k-batch code then the remaining servers form a $(k-1)$-batch code and hence $FB(s,k) > FB(s,k-1)$ for $k > 1$.
2) If any server is removed from a k-PIR code then the remaining servers form a $(k-1)$-PIR code and hence $FP(s,k) > FP(s,k-1)$ for $k > 1$.
3) If $FP(s,1) < s$ or $FB(s,1) < s$ then the rank of the information stored by the symbols is less than s and hence there is a linear combination not in their spanned information that cannot be recovered, a contradiction. Hence, $FP(s,1) \geq s$ and $FB(s,1) \geq s$. An 1-PIR code (1-batch code) of length s is constructed by storing the information symbol x_j, $1 \leq j \leq s$, in the j-th server. Therefore, $FP(s,1) \leq s$ and $FB(s,1) \leq s$ and the claim follows.
4) Since $FP(s,k) > FP(s,k-1)$ for $k > 1$, it follows that $FP(s,2) \geq s+1$. Consider the code of length $s+1$, where the j-th server stores the information symbol x_j, $1 \leq j \leq s$ and the $(s+1)$-th server stores a parity symbol $\sigma = \sum x_j$. For any requested symbol $\mathbf{v}$, let supp($\mathbf{v}$) be its support set. The requested symbol $\mathbf{v}$ can be recovered from the servers indexed by its support set and simultaneously by the remaining servers, since the sum of the symbols from all servers is zero, i.e., $\mathbf{v} = \sum_{j \in \text{supp}(\mathbf{v})} x_j = \sum_{j \notin \text{supp}(\mathbf{v})} x_j + \sigma$.
5) From the previous parts of the theorem we have $FP(s,2k-1) \leq FP(s,2k) - 1$. On the other hand, suppose we have a functional $(2k-1)$-PIR code with $FP(s,2k-1)$ servers. Add a server storing a global parity symbol, i.e., the sum of the symbols in the other servers. Any requested symbol can be recovered $(2k-1)$ times in the same way as in the functional $(2k-1)$-PIR code. It can be recovered one additional time by using all the remaining servers, since the global parity implies that the sum of the symbols from all servers is zero. This implies that $FP(s,2k) \leq FP(s,2k-1)+1$ and thus, $FP(s,2k) = FP(s,2k-1)+1$.
6) Follows from the observation that a functional k-batch code can serve as a functional k-PIR code.

■

Another basic result concerning PIR and batch codes with s information symbols and n servers is related to their presentation via a binary $s \times n$ matrix $\mathbf{G}$ whose columns represent the information in the servers. In other words, the entries on the i-th column of $\mathbf{G}$ have *ones* which relate to the information symbols used in the coded symbol stored in the i-th server. A code in which each information symbol is stored in a server will be called *systematic*. An intriguing question is whether for all PIR codes and/or batch codes there are related systematic codes with the same parameters? We conjecture that this is indeed the case, but there is no proof for this property for k-PIR and k-batch and it is left as an open problem. We can solve this question in the case of functional PIR and functional batch.

Lemma 6: If there exists a functional k-PIR (batch) code $\mathcal{C}$ of length n and dimension s, then there exists a systematic functional k-PIR (batch) code of length n and dimension s.

Proof: Assume first that $\mathcal{C}$ is a functional k-PIR code that is represented by an $s \times n$ matrix $\mathbf{G}$. If $rank(\mathbf{G}) < s$, then there exists a nonzero vector $\mathbf{v}$ not in the column space of $\mathbf{G}$ which cannot be recovered, a contradiction. Therefore, $rank(\mathbf{G}) = s$. Assume w.l.o.g. that $\mathbf{G} = [\mathbf{A}\ \mathbf{B}]$, where $\mathbf{A}$ is an $s \times s$ matrix, $\mathbf{B}$ is an $s \times (n-s)$ matrix, and $rank(\mathbf{A}) = s$, i.e., A is an invertible matrix. We claim that $\mathbf{G}' = [\mathbf{A}^{-1}\mathbf{A}\ \ \mathbf{A}^{-1}\mathbf{B}]$ is also a matrix representing a functional k-PIR code $\mathcal{C}'$. For each request $\mathbf{v}$ (for the code $\mathcal{C}'$), consider how $\mathbf{Av}$ is recovered k times using $\mathcal{C}$. For any set of columns in $\mathbf{G}$ summing up to $\mathbf{Av}$, we use the columns in $\mathbf{G}'$ with the same indices. These columns sum to $\mathbf{A}^{-1}\mathbf{Av} = \mathbf{v}$. Therefore, a systematic functional k-PIR code of length n and dimension s is obtained.

A similar proof works if $\mathcal{C}$ is a functional k-batch code. ■

Some more simple bounds on $FP(s,k)$ are given in the following theorem.

Theorem 7: If s, t, s_1, s_2, k_1, k_2 are positive integers, then

(1) $FP(s, 2^{s-1}) = 2^s - 1$.
(2) $FP(s, k_1 + k_2) \leq FP(s,k_1) + FP(s,k_2)$.
(3) $FP(s_1 + s_2, k) \leq FP(s_1,k) + FP(s_2,k)$.
(4) $FP(rt, 2^r) \leq 2t(2^r - 1)$.

Proof:

(1) By Lemma 4(1), we have that $FP(s,2^{s-1}) \geq P(s,2^{s-1}) = 2^s - 1$, so we only need to show that $FP(s,2^{s-1}) \leq 2^s - 1$. Indeed, a functional 2^{s-1}-PIR code is obtained from an $s \times (2^s - 1)$ matrix whose columns are all the columns of length s. Each request $\mathbf{v}$ can be recovered 2^{s-1} times, by $2^{s-1}-1$ pairs $(\mathbf{u}, \mathbf{u}+\mathbf{v})$ and by $\mathbf{v}$ itself.

(2) This result follows immediately by concatenating the matrices which represent the functional k_1-PIR code and the functional k_2-PIR code with s information symbols.
(3) Assume A and B are the matrices which represent the functional k-PIR codes which attain $FP(s_1, k)$ and $FP(s_2, k)$, respectively. The matrix $\begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix}$ represents a functional k-PIR code with $s_1 + s_2$ information symbols.
(4) By (1) and (2) we have that $FP(r, 2^r) \leq 2(2^r - 1)$ and applying it t times we obtain $FP(rt, 2^r) \leq 2t(2^r - 1)$.

■

Our first target in this paper is to improve on Theorem 7(4).

D. Our Contribution and Outline

In the rest of the paper new lower and upper bounds on $FB(s, k)$ and $FP(s, k)$ will be presented. In Section II a construction of functional k-PIR codes with k being a power of 2 is presented. Proper puncturing of the code obtained by the construction yields functional k-PIR codes for arbitrary k. In Section III we provide several lower bounds on $FP(s, k)$. First, in Section III-A a general asymptotic lower bound using a counting argument is proved. This argument is applied also on specific values of s and k to get nontrivial lower bounds on $FP(s, k)$. An improved lower bound for $k = 3$ and $k = 4$ is presented in Section III-B. This lower bound is in fact tight. A table on the asymptotic and specific lower and upper bounds for $FP(s, k)$ is also given. A random construction of functional batch codes is given in Section IV. Bounds on the length of functional batch codes are given in this section too. In Section V, we study the performance of simplex codes when used as functional batch codes. Conclusions and problems for future research are outlined in Section VI.

II. A Construction of Functional PIR Codes

In this section an explicit construction of functional k-PIR codes when k is a power of 2, is presented. The code which has rt information symbols will be represented by two $(t + 1) \times 2^r$ arrays. One array will be defined in the construction and the second array will be defined in the proof for the correctness of the construction. In the first array, each entry, except for the entries of the last column, represents the content of different servers. The last column of the array contains *zeroes*. In the second array, each column represents a recovery set. The second array is obtained from the first array by a permutation defined via a translation induced from the requested symbol. By puncturing p times this code for which $k = 2^r$, a functional k-PIR codes for $k = 2^r - 2p$ will be obtained.

Construction 1: Let $\{x_j^i : 1 \leq i \leq t,\ 1 \leq j \leq r\}$ be the set of $s = rt$ information symbols. Let $\mathcal{T}$ be a $(t + 1) \times 2^r$ array whose last column consists of *zeroes*. The columns of $\mathcal{T}$ are indexed by the elements of the power set $2^{[r]}$. The i-th row, $1 \leq i \leq t$, contains the 2^r linear combinations of the symbols $\{x_j^i : 1 \leq j \leq r\}$. In particular, the entry on the column indexed by $A \in 2^{[r]}$ contains the linear combination $x_A^i = \sum_{j \in A} x_j^i$ (note that $x_\varnothing^i = 0$). Finally, the $(t+1)$-th row is a parity row, where the entry in the column indexed by A is $X_A = \sum_{i=1}^t x_A^i = \sum_{i=1}^t \sum_{j \in A} x_j^i$. This entry will be called the *leader* of the column. Note that only the entries of the column indexed by $\varnothing$ do not correspond to information stored in a server. The parity of this column which is *zero* is stored in the $(t + 1)$-th row and it is also called a leader. Each other symbol in the array $\mathcal{T}$ is stored in a different server. The array $\mathcal{T}$ contains all the $n = (2^r - 1)(t + 1)$ symbols and hence it will be called *the stored symbols array*.

By Theorem 7, $FP(rt, 2^r) \leq 2t(2^r - 1)$. In the next theorem this upper bound is improved.

Theorem 8: The code of Construction 1 is a functional 2^r-PIR code. Therefore, $FP(rt, 2^r) \leq (2^r - 1)(t + 1)$.

Proof: Let $\mathbf{v}$ be the requested symbol, i.e., $\mathbf{v}$ is a linear combination

$$\mathbf{v} = \mathbf{v}^1 + \mathbf{v}^2 + \cdots + \mathbf{v}^t \ ,$$

where each $\mathbf{v}^i$ is a linear combination of the information symbols $\{x_j^i : 1 \leq j \leq r\}$, $1 \leq i \leq t$. We also define $\mathbf{v}^{t+1} = 0$.

Given the $(t+1) \times 2^r$ stored symbols array $\mathcal{T}$, we construct a new $(t + 1) \times 2^r$ array $\mathcal{R}^{\mathbf{v}}$ as follows. The rows and the columns of $\mathcal{R}^{\mathbf{v}}$ are indexed exactly in the same way as the rows and columns of $\mathcal{T}$ are indexed. To the symbol in $\mathcal{T}$ in the entry on the i-th row, $1 \leq i \leq t + 1$, and the column indexed by any subset A of $2^{[r]}$, we add $\mathbf{v}^i$ to obtain the corresponding symbol in $\mathcal{R}^{\mathbf{v}}$ in the same entry. The array $\mathcal{R}^{\mathbf{v}}$ will be called *the recovery array for* $\mathbf{v}$ since each column contains the content of the servers which form one of the recovery sets. Note, that the i-th row of $\mathcal{R}^{\mathbf{v}}$, $1 \leq i \leq t + 1$, is a permutation of the i-th row of $\mathcal{T}$ and hence the symbols contained in $\mathcal{R}^{\mathbf{v}}$ are exactly the same symbols contained in $\mathcal{T}$, which implies that the information of each server is contained in exactly one entry of $\mathcal{R}^{\mathbf{v}}$, but usually not in the same entry as in $\mathcal{T}$. The exceptions are the $(t + 1)$-th row and each row i for which $\mathbf{v}^i = 0$. It implies that the array $\mathcal{R}^{\mathbf{v}}$ represents the content of the servers, but in different entries from those of $\mathcal{T}$. We claim now that in each column of $\mathcal{R}^{\mathbf{v}}$ contain the content of a set of servers which form a recovery set.

Hence, to complete the proof it is sufficient to show that the sum of the symbols in each column of $\mathcal{R}^{\mathbf{v}}$ is $\mathbf{v}$. For a subset A of $[r]$ let $\mathcal{T}_A$ be the column of $\mathcal{T}$ indexed by A and let $\mathcal{R}_A^{\mathbf{v}}$ be the column of $\mathcal{R}^{\mathbf{v}}$ indexed by A. The sum of the symbols in $\mathcal{R}_A^{\mathbf{v}}$ is computed from the symbols of $\mathcal{T}_A$ and the request $\mathbf{v}$ as follows

$$\sum_{i=1}^{t} (x_A^i + \mathbf{v}^i) + X_A = \sum_{i=1}^{t} x_A^i + X_A + \sum_{i=1}^{t} \mathbf{v}^i = \sum_{i=1}^{t} \mathbf{v}^i = \mathbf{v}.$$

Therefore, each column of $\mathcal{R}^{\mathbf{v}}$ can serve as a recovery set for the requested symbol $\mathbf{v}$. Thus, the proof of the theorem is completed. ■

Example 1: Let $r = 4$, $t = 3$, $s = rt = 12$, and $k = 2^r = 16$. All the information symbols and the coded symbols are represented in the stored symbols array in Table I, where $x_{j_1\ j_2 \ldots j_\ell}^i \triangleq x_{j_1}^i + x_{j_2}^i + \cdots + x_{j_\ell}^i$ and similarly $X_{j_1\ j_2 \ldots j_\ell} \triangleq \sum_{i=1}^t x_{j_1\ j_2 \ldots j_\ell}^i = \sum_{i=1}^t (x_{j_1}^i + x_{j_2}^i + \cdots + x_{j_\ell}^i)$.

Now suppose that the requested symbol is $\mathbf{v} = x_1^1 + x_1^2 + x_2^2 + x_2^3 + x_3^3 + x_4^3$, i.e. $\mathbf{v}^1 = x_1^1$, $\mathbf{v}^2 = x_1^2 + x_2^2$,

TABLE I

x_1^1	x_2^1	x_3^1	x_4^1	x_{12}^1	x_{13}^1	x_{14}^1	x_{23}^1	x_{24}^1	x_{34}^1	x_{123}^1	x_{124}^1	x_{134}^1	x_{234}^1	x_{1234}^1	0
x_1^2	x_2^2	x_3^2	x_4^2	x_{12}^2	x_{13}^2	x_{14}^2	x_{23}^2	x_{24}^2	x_{34}^2	x_{123}^2	x_{124}^2	x_{134}^2	x_{234}^2	x_{1234}^2	0
x_1^3	x_2^3	x_3^3	x_4^3	x_{12}^3	x_{13}^3	x_{14}^3	x_{23}^3	x_{24}^3	x_{34}^3	x_{123}^3	x_{124}^3	x_{134}^3	x_{234}^3	x_{1234}^3	0
X_1	X_2	X_3	X_4	X_{12}	X_{13}	X_{14}	X_{23}	X_{24}	X_{34}	X_{123}	X_{124}	X_{134}	X_{234}	X_{1234}	0

TABLE II

0	x_{12}^1	x_{13}^1	x_{14}^1	x_2^1	x_3^1	x_4^1	x_{123}^1	x_{124}^1	x_{134}^1	x_{23}^1	x_{24}^1	x_{34}^1	x_{1234}^1	x_{234}^1	x_1^1
x_2^2	x_1^2	x_{123}^2	x_{124}^2	0	x_{23}^2	x_{24}^2	x_{13}^2	x_{14}^2	x_{1234}^2	x_3^2	x_4^2	x_{234}^2	x_{134}^2	x_{34}^2	x_{12}^2
x_{1234}^3	x_{34}^3	x_{24}^3	x_{23}^3	x_{134}^3	x_{124}^3	x_{123}^3	x_4^3	x_3^3	x_2^3	x_{14}^3	x_{13}^3	x_{12}^3	0	x_1^3	x_{234}^3
X_1	X_2	X_3	X_4	X_{12}	X_{13}	X_{14}	X_{23}	X_{24}	X_{34}	X_{123}	X_{124}	X_{134}	X_{234}	X_{1234}	0

$\mathbf{v}^3 = x_2^3 + x_3^3 + x_4^3$. For $1 \leq i \leq 3$, by adding $\mathbf{v}^i$ to each entry in the i-th row we obtain the recovery array as shown in Table II.

It is straightforward to verify that each column of $\mathcal{R}^{\mathbf{v}}$ is a recovery set for the requested symbol $\mathbf{v}$. For example, in the third column we have $(x_1^1 + x_3^1) + (x_1^2 + x_2^2 + x_3^2) + (x_2^3 + x_4^3) + (x_3^1 + x_3^2 + x_3^3) = x_1^1 + x_1^2 + x_2^2 + x_2^3 + x_3^3 + x_4^3 = \mathbf{v}$.

The next step is to consider how to modify Construction 1 for arbitrary k. Since by Theorem 5(5) $FP(s, 2\ell) = FP(s, 2\ell - 1) + 1$ we can consider only even values of k. The main idea is simply to delete some entries of the array $\mathcal{T}$, i.e. removing some servers and hence we can say that the k-PIR code for $k = 2^r$ is being punctured. This simple idea is less trivial to explain and even less trivial to prove that the remaining servers can form the required number of recovery sets. Hence, we start with the simplest case which is $k = 2^r - 2$ to illustrate the idea.

Construction 2: Let $\mathcal{T}$ be the $(t+1) \times 2^r$ stored symbols array constructed in Construction 1. Choose three different subsets A, B, and C of $[r]$ such that $A = (B \setminus C) \cup (C \setminus B)$. Delete the symbols in the first t rows of column $\mathcal{T}_A$ and delete the leader symbols X_B and X_C in columns $\mathcal{T}_B$ and $\mathcal{T}_C$, respectively. The deletion is done by marking the deleted symbols by a red color. Any deleted symbol will be also called a *red symbol*. Each deleted symbol is related to a server which is being removed, i.e. these $t+2$ red symbols are not associated with any server. This array obtained from $\mathcal{T}$ will be denoted by $\tilde{\mathcal{T}}$ and also called *the stored symbols array*. The servers store the content of the entries in $\tilde{\mathcal{T}}$ which are not *zeroes* and do not contain red symbols. Thus, the length of the code is $n = (t+1)(2^r - 1) - (t+2) = (2^r - 2)t + 2^r - 3$.

Theorem 9: The code of Construction 2 is a functional $(2^r - 2)$-PIR code. Therefore, $FP(rt, 2^r - 2) \leq (2^r - 2)t + 2^r - 3$.

Proof: Let $\mathbf{v}$ be the requested symbol, i.e., $\mathbf{v}$ is a linear combination

$$\mathbf{v} = \mathbf{v}^1 + \mathbf{v}^2 + \cdots + \mathbf{v}^t \ ,$$

where each $\mathbf{v}^i$ is a linear combination of the information symbols $\{x_j^i : 1 \leq j \leq r\}$, $1 \leq i \leq t$. We also define $\mathbf{v}^{t+1} = 0$.

Given the $(t+1) \times 2^r$ stored symbols array $\tilde{\mathcal{T}}$, we construct a new $(t+1) \times 2^r$ array $\tilde{\mathcal{R}}^{\mathbf{v}}$ from $\tilde{\mathcal{T}}$ exactly as how $\mathcal{R}^{\mathbf{v}}$ was constructed from $\mathcal{T}$ in the proof of Theorem 8 (adding $\mathbf{v}^i$ to all the 2^r entries of the i-th row, $1 \leq i \leq t+1$). The array $\tilde{\mathcal{R}}^{\mathbf{v}}$ will be called *the recovery array for* $\mathbf{v}$ since each column without a deleted leader will be used to define a recovery set. In $\tilde{\mathcal{R}}^{\mathbf{v}}$ each symbol in a column of a deleted leader will be called a *free symbol* since it is free to join any recovery set. Each symbol which was a red symbol in $\tilde{\mathcal{T}}$ will maintain a red symbol in $\tilde{\mathcal{R}}^{\mathbf{v}}$ (usually in a different entry, unless it is either a leader or in the i-th row and $\mathbf{v}^i = 0$).

Each column with a (non-deleted) leader corresponds to a recovery set as follows.

- If the column contains no red symbol then the sum of the entries in the column is $\mathbf{v}$ exactly as was proved in Theorem 8.
- If the column contains a red symbol in the i-th row then we add the symbols of the i-th row in columns $\mathcal{T}_B$ and $\mathcal{T}_C$ to the recovery set. The red symbol in the i-th row is x_A^i. The free symbols in the i-th row of columns $\mathcal{T}_B$ and $\mathcal{T}_C$ are $x_B^i + \mathbf{v}^i$ and $x_C^i + \mathbf{v}^i$, respectively. $x_A^i = x_B^i + x_C^i = x_B^i + \mathbf{v}^i + x_C^i + \mathbf{v}^i$ and hence the red symbol in the i-th row can be replaced by the related free symbols in columns $\mathcal{T}_B$ and $\mathcal{T}_C$. The rest of the proof is as in the proof of Theorem 8.

Therefore, each column of $\mathcal{R}^{\mathbf{v}}$ with a (non-deleted) leader can serve as a recovery set for the requested symbol $\mathbf{v}$, with replaced symbols for possible red symbols in the recovery set. Thus, the proof of the theorem is completed. ■

Example 2: Continuing Example 1 above, choose three subsets $A = \{1234\}$, $B = \{12\}$, and $C = \{34\}$. Delete the symbols in the first t rows of the column $\mathcal{T}_A = \mathcal{T}_{1234}$ and delete the leader symbols $X_B = X_{12}$ and $X_C = X_{34}$ in columns $\mathcal{T}_B = \mathcal{T}_{12}$ and $\mathcal{T}_C = \mathcal{T}_{34}$, respectively. The deletion is done by marking the deleted symbols in a red color. The result is the stored symbols array as shown in Table III.

Suppose that the requested symbol is $\mathbf{v} = x_1^1 + x_1^2 + x_2^2 + x_2^3 + x_3^3 + x_4^3$, i.e., $\mathbf{v}^1 = x_1^1$, $\mathbf{v}^2 = x_1^2 + x_2^2$, $\mathbf{v}^3 = x_2^3 + x_3^3 + x_4^3$. By adding $\mathbf{v}^i$, $1 \leq i \leq 3$, to each entry in the i-th row, the recovery array as shown in Table IV is obtained. Note that in this array the deleted symbols are still marked in red, i.e., the red color is with the symbol itself rather than the entry. Moreover the entries in columns

TABLE III

x_1^1	x_2^1	x_3^1	x_4^1	x_{12}^1	x_{13}^1	x_{14}^1	x_{23}^1	x_{24}^1	x_{34}^1	x_{123}^1	x_{124}^1	x_{134}^1	x_{234}^1	x_{1234}^1	0
x_1^2	x_2^2	x_3^2	x_4^2	x_{12}^2	x_{13}^2	x_{14}^2	x_{23}^2	x_{24}^2	x_{34}^2	x_{123}^2	x_{124}^2	x_{134}^2	x_{234}^2	x_{1234}^2	0
x_1^3	x_2^3	x_3^3	x_4^3	x_{12}^3	x_{13}^3	x_{14}^3	x_{23}^3	x_{24}^3	x_{34}^3	x_{123}^3	x_{124}^3	x_{134}^3	x_{234}^3	x_{1234}^3	0
X_1	X_2	X_3	X_4	X_{12}	X_{13}	X_{14}	X_{23}	X_{24}	X_{34}	X_{123}	X_{124}	X_{134}	X_{234}	X_{1234}	0

TABLE IV

0	x_{12}^1	x_{13}^1	x_{14}^1	x_2^1	x_3^1	x_4^1	x_{123}^1	x_{124}^1	x_{134}^1	x_{23}^1	x_{24}^1	x_{34}^1	x_{1234}^1	x_{234}^1	x_1^1
x_2^2	x_1^2	x_{123}^2	x_{124}^2	0	x_{23}^2	x_{24}^2	x_{13}^2	x_{14}^2	x_{1234}^2	x_3^2	x_4^2	x_{234}^2	x_{134}^2	x_{34}^2	x_{12}^2
x_{1234}^3	x_{34}^3	x_{24}^3	x_{23}^3	x_{134}^3	x_{124}^3	x_{123}^3	x_4^3	x_3^3	x_2^3	x_{14}^3	x_{13}^3	x_{12}^3	0	x_1^3	x_{234}^3
X_1	X_2	X_3	X_4	X_{12}	X_{13}	X_{14}	X_{23}	X_{24}	X_{34}	X_{123}	X_{124}	X_{134}	X_{234}	X_{1234}	0

TABLE V

0	x_{12}^1	x_{13}^1	x_{14}^1		x_3^1	x_4^1	x_{123}^1	x_{124}^1		x_{23}^1	x_{24}^1	x_{34}^1	x_2^1, x_{134}^1	x_{234}^1	x_1^1
x_2^2	x_1^2	x_{123}^2	x_{124}^2	0	x_{23}^2	x_{24}^2	x_{13}^2	x_{14}^2	x_{1234}^2	x_3^2	x_4^2	x_{234}^2	x_{134}^2	x_{34}^2	x_{12}^2
x_{134}^3, x_2^3	x_{34}^3	x_{24}^3	x_{23}^3		x_{124}^3	x_{123}^3	x_4^3	x_3^3		x_{14}^3	x_{13}^3	x_{12}^3	0	x_1^3	x_{234}^3
X_1	X_2	X_3	X_4	X_{12}	X_{13}	X_{14}	X_{23}	X_{24}	X_{34}	X_{123}	X_{124}	X_{134}	X_{234}	X_{1234}	0

TABLE VI

0	x_{12}^1	x_{13}^1	x_{14}^1	x_2^1	x_3^1	x_4^1	x_{123}^1	x_{124}^1	x_{134}^1	x_{23}^1	x_{24}^1	x_{34}^1	x_{1234}^1	x_{234}^1	x_1^1
x_2^2	x_1^2	x_{123}^2	x_{124}^2	0	x_{23}^2	x_{24}^2	x_{13}^2	x_{14}^2	x_{1234}^2	x_3^2	x_4^2	x_{234}^2	x_{134}^2	x_{34}^2	x_{12}^2
x_{1234}^3	x_{34}^3	x_{24}^3	x_{23}^3	x_{134}^3	x_{124}^3	x_{123}^3	x_4^3	x_3^3	x_2^3	x_{14}^3	x_{13}^3	x_{12}^3	0	x_1^3	x_{234}^3
X_1	X_2	X_3	X_4	X_{12}	X_{13}	X_{14}	X_{23}	X_{24}	X_{34}	X_{123}	X_{124}	X_{134}	X_{234}	X_{1234}	0

$\mathcal{T}_B = \mathcal{T}_{12}$ and $\mathcal{T}_C = \mathcal{T}_{34}$ are marked with a yellow color. Since $X_B = X_{12}$ and $X_C = X_{34}$ are deleted, we do not consider using the related columns $\mathcal{T}_B = \mathcal{T}_{12}$ and $\mathcal{T}_C = \mathcal{T}_{34}$ as recovery sets. Therefore, the symbols on these yellow entries are free symbols and can be used when we need to replace certain deleted symbols.

As for the deleted (red) symbols located on recovery sets, the free symbols (symbols in entries marked with yellow) are used to replace the deleted (red) symbols. For x_{1234}^1 and x_{1234}^3, the two free symbols in the same row can be used to replace the deleted (red) symbol, i.e., $x_{1234}^1 = x_2^1 + x_{134}^1$ and $x_{1234}^3 = x_{134}^3 + x_2^3$. On the second row, the deleted (red) symbol x_{1234}^2 lies in an entry marked with yellow and does not have to be replaced since this column is not used as a recovery set. Hence, the recovery array is adjusted into the form as shown in Table V. It is then straightforward to verify that the symbols on each column with an undeleted leader sum up to the requested symbol $\mathbf{v}$. Therefore, a functional 14-PIR code is obtained.

To sum up, the construction of the functional $(2^r - 2)$-PIR code is a '1-puncturing' of the functional (2^r)-PIR code, where the punctured symbols are determined by a choice of the tuple of subsets $\{A, B, C\}$. To generalize this idea to a 'p-puncturing', it seems natural to just take more tuples of subsets $\{A_j, B_j, C_j\}$ and perform similar puncturing methods. However, this generalization is non-trivial since one may meet the following scenario.

Say we continue Example 2 and intend to do a '2-puncturing' to obtain a functional 12-PIR code. Choose another triple of subsets $\big\{\{13\}, \{4\}, \{134\}\big\}$. Delete the symbols in the first t rows of the column $\mathcal{T}_{134}$ and delete the leader symbols X_{13} and X_4 in columns $\mathcal{T}_{13}$ and $\mathcal{T}_4$, respectively. In the recovering array for the same requested symbol $\mathbf{v} = x_1^1 + x_1^2 + x_2^2 + x_2^3 + x_3^3 + x_4^3$, the deleted symbols are marked in red. The entries in the columns indexed by $\{12\}$, $\{34\}$, $\{13\}$, $\{4\}$ are marked with yellow, indicating that the symbols on these yellow entries are free symbols and can be used to replace certain deleted symbols. The recovery array is presented as in Table VI.

Now, on each row there are two deleted (red) symbols that should be replaced by combinations of free symbols in yellow entries. The problem is that we cannot simply replace x_{1234}^1 with $x_2^1 + x_{134}^1$ as before in Example 2 since now x_{134}^1 is also a deleted symbol. The solution is to replace x_{1234}^1 by $x_2^1 + x_{14}^1 + x_3^1$ and x_{134}^1 does not need repairing since it lies on a yellow entry. This scenario demonstrates that generalizing '1-puncturing' into 'p-puncturing' is non-trivial in the sense that we need an explicit algorithm to describe how to use the free symbols to replace the deleted symbols.

TABLE VII

x_1^1	x_2^1	x_3^1	x_4^1	x_{12}^1	x_{13}^1	x_{14}^1	x_{23}^1	x_{24}^1	x_{34}^1	x_{123}^1	x_{124}^1	x_{134}^1	x_{234}^1	x_{1234}^1	0
x_1^2	x_2^2	x_3^2	x_4^2	x_{12}^2	x_{13}^2	x_{14}^2	x_{23}^2	x_{24}^2	x_{34}^2	x_{123}^2	x_{124}^2	x_{134}^2	x_{234}^2	x_{1234}^2	0
x_1^3	x_2^3	x_3^3	x_4^3	x_{12}^3	x_{13}^3	x_{14}^3	x_{23}^3	x_{24}^3	x_{34}^3	x_{123}^3	x_{124}^3	x_{134}^3	x_{234}^3	x_{1234}^3	0
X_1	X_2	X_3	X_4	X_{12}	X_{13}	X_{14}	X_{23}	X_{24}	X_{34}	X_{123}	X_{124}	X_{134}	X_{234}	X_{1234}	0

Our generalization of Construction 2 and the proof of its correctness in Theorem 9, i.e. generalizing the 1-puncturing to p-puncturing, will consist of four steps. In the first step, p pairwise disjoint triples from $2^{[r]}$ will be defined (two elements of a triple for deleting two leader symbols and the third one for deleting the symbols of the column excluding the leader). In the second step the related recovery array is constructed similarly to the definition in Construction 2. In the third step a replacing operation (in several rounds) to replace the deleted (red) symbols by free symbols will be described. In the last step we will prove that these actual replacements result in the required recovery sets.

Following these ideas, Construction 2 for k-PIR, $k = 2^r - 2$ can be generalized to arbitrary $k = 2^r - 2p$, where $1 \leq p < 2^{r-2}$. For the first step of the construction (defining the pairwise disjoint triples) we need the following definition and results on *partial spreads*.

Definition 10: A partial 2-spread of $\mathbb{F}_2^r$ is a collection of 2-dimensional subspaces $V_1, \ldots, V_M$ of $\mathbb{F}_2^r$ such that $V_i \cap V_j = \{\mathbf{0}\}$ for all $i \neq j$.

It is shown in [16] that a partial 2-spread with $M = 2^{r-2}$ always exists. In each 2-dimensional subspace V_i we have three nonzero vectors. Let A_i, B_i and C_i be their supports which are subsets of $[r]$. By the definition of a partial 2-spread, $A_i = (B_i \setminus C_i) \cup (C_i \setminus B_i)$ and the triples $\{\{A_i, B_i, C_i\} : 1 \leq i \leq M\}$ are pairwise disjoint.

Construction 3: Let $\mathcal{T}$ be the $(t+1) \times 2^r$ stored symbols array constructed in Constructions 1 and 2. Since $p < 2^{r-2}$ there exists a partial 2-spread $\mathbb{F}_2^r$ which contains p pairwise disjoint triples $\{\{A_i, B_i, C_i\} : 1 \leq i \leq p\}$ such that $A_i = (B_i \setminus C_i) \cup (C_i \setminus B_i)$. For each triple $\{A_i, B_i, C_i\}$, delete the symbols in the first t rows of column $\mathcal{T}_{A_i}$ and the leader symbols X_{B_i} and X_{C_i} in columns $\mathcal{T}_{B_i}$ and $\mathcal{T}_{C_i}$, respectively. The deletion is done by marking the deleted symbols by a red color. Any deleted symbol will be called a *red symbol*, These $p(t+2)$ red symbols are not associated with any server. This array obtained from $\mathcal{T}$ will be denoted by $\tilde{\mathcal{T}}$. Thus, the length of the code is $n = (t+1)(2^r-1) - tp - 2p = (2^r - p - 1)t + 2^r - 2p - 1$.

Theorem 11: The code of Construction 3 is a functional $(2^r - 2p)$-PIR code. Therefore, $FP(rt, 2^r - 2p) \leq (2^r - p - 1)t + 2^r - 2p - 1$, for $0 \leq p < 2^{r-2}$.

Proof: Let $\mathbf{v}$ be the requested symbol, i.e., $\mathbf{v}$ is a linear combination

$$\mathbf{v} = \mathbf{v}^1 + \cdots + \mathbf{v}^t \ ,$$

where each $\mathbf{v}^i$ is a linear combination of the information symbols $\{x_j^i : 1 \leq j \leq r\}$. We also define $\mathbf{v}^{t+1} = 0$.

Given the $(t+1) \times 2^r$ stored symbols array $\tilde{\mathcal{T}}$, we construct a new $(t+1) \times 2^r$ array $\tilde{\mathcal{R}}^{\mathbf{v}}$ exactly as in the proofs of Theorems 8 and 9 (adding $\mathbf{v}^i$ to all the 2^r entries of the i-th row, $1 \leq i \leq t+1$). Each symbol which was a red symbol in $\tilde{\mathcal{T}}$ will be also a red symbol in $\tilde{\mathcal{R}}^{\mathbf{v}}$ (usually in a different entry, unless it is either a leader or in the i-th row and $\mathbf{v}^i = 0$).

The $2^r - 2p$ recovery sets relate to the $2^r - 2p$ columns in which the leaders were not deleted. By the proof of Theorem 8, the sum of the symbols (including the red ones) in each such column is $\mathbf{v}$. Our goal is that each column whose leader was not deleted will be a recovery set. Hence, we have to apply a procedure to replace the red symbols in these columns. For each row i, $1 \leq i \leq t$, we apply the following procedure. In each step of the procedure the number of red symbols in the row will be the same as the number of pairs of columns with deleted leaders which have some symbols (red or free). Before the first step the number of red symbols in the row is p and the number of such pairs is also p.

Let $\{B_j, C_j\}$ be a pair from the disjoint triples for which the two related columns do not contain a red symbol. If there is no such pair then all the red symbols are in the columns with deleted leaders and the procedure for the row is completed. The sum of the symbols in column B_j (of the i-th row) is $x_{B_j}^i + \mathbf{v}^i$ and in column C_j is $x_{B_j}^i + \mathbf{v}^i$. $x_{B_j}^i + \mathbf{v}^i + x_{B_j}^i + \mathbf{v}^i = x_{A_j}^i$ and $x_{A_j}^i$ is a red symbol in some column D (neither B_j nor C_j (since the related two entries do not have a red symbol). We replace the red symbol $x_{A_j}^i$ of column D with the two symbols $x_{B_j}^i + \mathbf{v}^i$ and $x_{C_j}^i + \mathbf{v}^i$ (which are not marked in red). Entries B_j and C_j in the i-th row will become empty. The number of red symbols in the i-th row was reduced by one and also the number of pairs of columns with deleted leader which have some symbols was reduced by one. Hence, these numbers remain equal and this property is satisfied at the end of the step for this row. Note, that the red symbol $x_{A_j}^i$ was replaced by two free (non-red) symbols whose sum equals to $x_{A_j}^i$.

After this procedure was applied on all the first t rows, all the recovery sets will not contain any red symbols. The sum of symbols of any recovery set is not changed during the procedure. The non-red symbols in new constructed array $\hat{\mathcal{R}}^{\mathbf{v}}$ are the same as the non-red symbols in $\tilde{\mathcal{R}}^{\mathbf{v}}$.

Therefore, each column of $\hat{\mathcal{R}}^{\mathbf{v}}$ can serve as a recovery set for the requested symbol $\mathbf{v}$. Thus, the proof of the theorem is completed. ∎

Example 3: Continuing Example 2, choose three disjoint triples of subsets $\{\{12\}, \{34\}, \{1234\}\}$, $\{\{13\}, \{4\}, \{134\}\}$ and $\{\{2\}, \{3\}, \{23\}\}$. Delete the symbols in the first t rows of the columns $\mathcal{T}_{1234}$, $\mathcal{T}_{134}$ and $\mathcal{T}_{23}$. Delete the leader symbols X_{12}, X_{34}, X_{13}, X_4, X_2 and X_3. The deletion is done by

TABLE VIII

0	x_{12}^1	x_{13}^1	x_{14}^1	x_2^1	x_3^1	x_4^1	x_{123}^1	x_{124}^1	x_{134}^1	x_{23}^1	x_{24}^1	x_{34}^1	x_{1234}^1	x_{234}^1	x_1^1
x_2^2	x_1^2	x_{123}^2	x_{124}^2	0	x_{23}^2	x_{24}^2	x_{13}^2	x_{14}^2	x_{1234}^2	x_3^2	x_4^2	x_{234}^2	x_{134}^2	x_{34}^2	x_{12}^2
x_{1234}^3	x_{34}^3	x_{24}^3	x_{23}^3	x_{134}^3	x_{124}^3	x_{123}^3	x_4^3	x_3^3	x_2^3	x_{14}^3	x_{13}^3	x_{12}^3	0	x_1^3	x_{234}^3
X_1	X_2	X_3	X_4	X_{12}	X_{13}	X_{14}	X_{23}	X_{24}	X_{34}	X_{123}	X_{124}	X_{134}	X_{234}	X_{1234}	0

TABLE IX

x_{1234}^3	x_{34}^3	x_{24}^3	x_{23}^3	x_{134}^3	x_{124}^3	x_{123}^3	x_4^3	x_3^3	x_2^3	x_{14}^3	x_{13}^3	x_{12}^3	0	x_1^3	x_{234}^3
⇓ ⇓ ⇓															
x_{1234}^3			x_{34}^3, x_{24}^3	x_{134}^3	x_{124}^3	x_{123}^3	x_4^3	x_3^3	x_2^3	x_{14}^3	x_{13}^3	x_{12}^3	0	x_1^3	x_{234}^3
⇓ ⇓ ⇓															
x_{1234}^3				x_{34}^3, x_{24}^3 x_{124}^3		x_{123}^3	x_4^3	x_3^3	x_2^3	x_{14}^3	x_{13}^3	x_{12}^3	0	x_1^3	x_{234}^3
⇓ ⇓ ⇓															
x_{34}^3, x_{24}^3 x_{124}^3, x_2^3						x_{123}^3	x_4^3	x_3^3		x_{14}^3	x_{13}^3	x_{12}^3	0	x_1^3	x_{234}^3

TABLE X

0						x_4^1	x_{123}^1	x_{124}^1		x_{12}^1, x_{13}^1	x_{24}^1	x_{34}^1	x_2^1, x_{14}^1 x_3^1	x_{234}^1	x_1^1
x_2^2				0		x_{24}^2	x_{13}^2	x_{14}^2	x_{1234}^2	x_3^2	x_4^2	x_{234}^2	x_{124}^2, x_1^2 x_{123}^2	x_{34}^2	x_{12}^2
x_{34}^3, x_{24}^3 x_{124}^3, x_2^3						x_{123}^3	x_4^3	x_3^3		x_{14}^3	x_{13}^3	x_{12}^3	0	x_1^3	x_{234}^3
X_1	X_2	X_3	X_4	X_{12}	X_{13}	X_{14}	X_{23}	X_{24}	X_{34}	X_{123}	X_{124}	X_{134}	X_{234}	X_{1234}	0

marking the deleted symbols in a red color. The result is the stored symbols array as shown in VII.

Suppose that the requested symbol is $\mathbf{v} = x_1^1 + x_1^2 + x_2^2 + x_2^3 + x_3^3 + x_4^3$, i.e., $\mathbf{v}^1 = x_1^1$, $\mathbf{v}^2 = x_1^2 + x_2^2$, $\mathbf{v}^3 = x_2^3 + x_3^3 + x_4^3$. By adding $\mathbf{v}^i$, $1 \leq i \leq 3$, to each entry in the i-th row the recovery array is obtained as shown in Table VIII. Note that in this array the deleted symbols are still marked in red. Moreover the entries in columns $\mathcal{T}_{12}$, $\mathcal{T}_{34}$, $\mathcal{T}_{13}$, $\mathcal{T}_4$, $\mathcal{T}_2$ and $\mathcal{T}_3$ are marked with a yellow color. Since X_{12}, X_{34}, X_{13}, X_4, X_2 and X_3 are deleted, we do not consider using the related columns as recovery sets. Therefore, the symbols on yellow entries are free symbols and can be used when we need to replace certain deleted symbols.

Independently, on each row red symbols are replaced step by step, e.g., the third row is transformed step by step as shown in Table IX :

After the appropriate red symbols were replaced in all the rows in a similar way, the final recovery array is as shown in Table X.

It is straightforward to verify that the symbols on each column with undeleted leader sum up to the requested symbol $\mathbf{v}$.

As mentioned in Theorem 5, by deleting any symbol in a functional $(2^r - 2p)$-PIR code we obtain a functional $(2^r - 2p - 1)$-PIR code, therefore we have

Corollary 12: $FP(rt, 2^r - 2p - 1) \leq (2^r - 1 - p)t + 2^r - 2p - 2$, for $0 \leq p < 2^{r-2}$.

Remark 1: Note, that all the constructions above for functional k-PIR codes with $k \in [2^{r-1} + 1, 2^r]$ are described for rt information symbols. When the number of information symbols is not a multiple of r, say $rt + r'$, $0 < r' < r$, we may add $r - r'$ virtual information symbols and apply the constructions above. All the virtual information symbols are set to zero. Also some coded symbols, which are linear combinations of only virtual information symbols are set to zero.

For example, assume we want to construct a functional 2^r-PIR code of dimension $rt + r'$. We add $r - r'$ virtual information symbols and hence we construct a functional 2^r-PIR code of dimension $r(t + 1)$ of length $(2^r - 1)(t + 2)$ using Construction 1. The virtual information symbols are now set to zero and thus some c symbols (linear combinations of virtual information symbols) are set to zero. The number c is $2^{r-r'} - 1$ when $t \geq 1$ or $2^{r-r'+1} - 2$ when $t = 0$ (since some 'leader' symbols are also set to zero when $t = 0$). Therefore for any $0 < r' < r$, $FP(r', 2^r) \leq 2(2^r - 2^{r-r'})$ and $FP(rt + r', 2^r) \leq (2^r - 1)(t + 1) + 2^r - 2^{r-r'}$ when $t \geq 1$.

Similar idea holds when $2^{r-1} < k < 2^r$, but this should be done carefully, since the c symbols set to zero are dependent on

TABLE XI
UPPER BOUNDS ON $FP(s,k)$ ARISING FROM OUR CONSTRUCTION IN SECTION II

	$k=6$	$k=8$
$s=1$	6	8
$s=2$	9	12
$s=3t$ $(t\geq 1)$	$6t+5$	$7t+7$
$s=3t+1$ $(t\geq 1)$	$6t+8$	$7t+11$
$s=3t+2$ $(t\geq 1)$	$6t+10$	$7t+13$

	$k=10$	$k=12$	$k=14$	$k=16$
$s=1$	10	12	14	16
$s=2$	15	18	21	24
$s=3$	19	22	25	28
$s=4t$ $(t\geq 1)$	$12t+9$	$13t+11$	$14t+13$	$15t+15$
$s=4t+1$ $(t\geq 1)$	$12t+14$	$13t+17$	$14t+20$	$15t+23$
$s=4t+2$ $(t\geq 1)$	$12t+18$	$13t+21$	$14t+24$	$15t+27$
$s=4t+3$ $(t\geq 1)$	$12t+20$	$13t+23$	$14t+26$	$15t+29$

the way that the puncturing from the functional 2^r-PIR code to the functional k-PIR code is done. Following this way, some results with small parameters are summarized in Table XI.

III. LOWER BOUNDS ON THE LENGTH OF FUNCTIONAL PIR CODES

This section is devoted to lower bounds on the length of functional PIR codes. When the number of requests k is a fixed constant,[1] $P(s,k)=s+o(s)$ (see Lemma 4) and hence the research objective is to analyze the redundancy part $o(s)$. However, for functional PIR codes this is not the case. By using a counting argument it will be proved in this section that $FP(s,k)$ grows linearly in s, i.e., $\lim_{s\to\infty} FP(s,k)/s \geq c$ for some constant c to be determined. Using another approach in this section, a better lower bound on $FP(s,3)$ and $FP(s,4)$ is derived. Codes for $k=4$ in Construction 1 attain this bound and hence the bound is exact.

A. A General Lower Bound by Counting

In our exposition which follows we will need some properties of the binomial coefficients. These properties are proved in the following lemma.

Lemma 13: If n and r are two positive integers such that $n>3r+2$, then

(1) $\binom{n}{r+1} > 2\binom{n}{r}$.
(2) $\binom{n}{r+1} > \sum_{i=1}^{r}\binom{n}{i}$.
(3) $\binom{n}{r+1} > \sum_{i=1}^{r-1}(r-i)\binom{n}{i}$.

Proof:

(1) follows immediately by comparing $\binom{n}{r+1}$ with $2\binom{n}{r}$.

[1] more precisely $k=o(s)$.

(2) follows by induction on r, where the basis is $\binom{n}{2} > \binom{n}{1}$, and in the induction step (1) is used.
(3) follows by induction on r, where the basis for $r=2$, i.e., $\binom{n}{3} > \binom{n}{1}$. For the induction hypothesis assume that the claim is true for $r-1$, i.e.

$$\binom{n}{r} > \sum_{i=1}^{r-2}(r-1-i)\binom{n}{i} .$$

By (2) we have that

$$\binom{n}{r+1} > \sum_{i=1}^{r}\binom{n}{i} ,$$

and combining this with the induction hypothesis we have that

$$\binom{n}{r+1} > \sum_{i=1}^{r}\binom{n}{i} > \sum_{i=1}^{r-2}(r-1-i)\binom{n}{i} + \sum_{i=1}^{r-1}\binom{n}{i}$$
$$= \sum_{i=1}^{r-2}(r-i)\binom{n}{i} + \binom{n}{r-1} = \sum_{i=1}^{r-1}(r-i)\binom{n}{i} ,$$

which proves the induction step. ∎

For the next theorem we remind the reader that by Theorem 5(5) we have $FP(s,2\ell)=FP(s,2\ell-1)+1$ and hence can consider only even values of k. The even values will be considered since they imply better bounds than the related odd values.

Theorem 14: For a fixed even integer $k\geq 4$,

$$\lim_{s\to\infty}\frac{FP(s,k)}{s} \geq \frac{1}{H(1/k)} ,$$

where $H(\cdot)$ is the binary entropy function defined by $H(p)=-p\log p-(1-p)\log(1-p)$.

Proof: Suppose there exists a functional k-PIR code of dimension s and length n. For each request $\mathbf{v}$, we have k disjoint recovery sets of $[n]$. The sum of the sizes of all these $k(2^s-1)$ recovery sets is at most $n(2^s-1)$. Hence, the average size of a recovery set should be at most $\frac{n}{k}$.

Consider all the subsets of $[n]$ of size at most $\lceil\frac{n}{k}\rceil+1$. If each such subset is used as a recovery set for some request, then the average size of a recovery set is at least

$$\frac{\sum_{i=1}^{\lceil\frac{n}{k}\rceil+1} i\binom{n}{i}}{\sum_{i=1}^{\lceil\frac{n}{k}\rceil+1}\binom{n}{i}} \tag{1}$$

By applying Lemma 13(3) on $\binom{n}{\lceil\frac{n}{k}\rceil+1}$ we have

$$\binom{n}{\lceil\frac{n}{k}\rceil+1} > \sum_{i=1}^{\lceil\frac{n}{k}\rceil-1}\left(\left\lceil\frac{n}{k}\right\rceil-i\right)\binom{n}{i} \tag{2}$$

TABLE XII

LOWER AND UPPER BOUNDS ON $\lim_{s\to\infty}\frac{FP(s,k)}{s}$

k	2	4	6	8	10	12	14	16
lower bound	1	1.2326	1.5384	1.8397	2.1322	2.4165	2.6937	2.9648
upper bound	1	1.5	2	2.3333	3	3.25	3.5	3.75
k	18	20	22	24	26	28	30	32
lower bound	3.2306	3.4917	3.7486	4.0019	4.2518	4.4987	4.7429	4.9845
upper bound	4.8	5	5.2	5.4	5.6	5.8	6	6.2

By developing the numerator in (1) and plugging (2) in the process we obtain

$$\begin{aligned}\sum_{i=1}^{\lceil\frac{n}{k}\rceil+1} i\binom{n}{i} &= \sum_{i=1}^{\lceil\frac{n}{k}\rceil} i\binom{n}{i} + \left\lceil\frac{n}{k}\right\rceil\binom{n}{\lceil\frac{n}{k}\rceil+1} + \binom{n}{\lceil\frac{n}{k}\rceil+1}\\ &> \sum_{i=1}^{\lceil\frac{n}{k}\rceil} i\binom{n}{i} + \left\lceil\frac{n}{k}\right\rceil\binom{n}{\lceil\frac{n}{k}\rceil+1} + \sum_{i=1}^{\lceil\frac{n}{k}\rceil-1}\left(\left\lceil\frac{n}{k}\right\rceil - i\right)\binom{n}{i}\\ &= \left\lceil\frac{n}{k}\right\rceil\sum_{i=1}^{\lceil\frac{n}{k}\rceil+1}\binom{n}{i}\ .\end{aligned}$$

Now, we can evaluate the average in (1) as

$$\frac{\sum_{i=1}^{\lceil\frac{n}{k}\rceil+1} i\binom{n}{i}}{\sum_{i=1}^{\lceil\frac{n}{k}\rceil+1}\binom{n}{i}} > \frac{\lceil\frac{n}{k}\rceil\sum_{i=1}^{\lceil\frac{n}{k}\rceil+1}\binom{n}{i}}{\sum_{i=1}^{\lceil\frac{n}{k}\rceil+1}\binom{n}{i}} = \left\lceil\frac{n}{k}\right\rceil \geq \frac{n}{k},$$

which contradicts our proof that the average size of a recovery set is at most $\frac{n}{k}$.

Therefore, not all the subsets of $[n]$ of size at most $\lceil\frac{n}{k}\rceil+1$ are used as recovery sets, which implies that $\sum_{i=1}^{\lceil\frac{n}{k}\rceil+1}\binom{n}{i} > k(2^s-1)$. The left hand side tends to $2^{nH(1/k)}$ as n tends to infinity. Hence, if $n=cs$, then

$$2^{csH(1/k)} > k(2^s-1)\ ,$$

which implies that $cH(1/k)>1$ and the claim of the theorem follows. ∎

Note, that the counting argument used in the proof of Theorem 14 implies that the recovery sets used for all the possible requests are of the smallest possible size. In practice, it is difficult to assume that this would be the case. Improving the lower bound by taking larger recovery sets into account is a future task.

The first several lower bounds on $\lim_{s\to\infty}\frac{FP(s,k)}{s}$ derived from Theorem 14, together with the related upper bounds implied by Construction 3, are summarized in Table XII. The lower bound for $k=4$ will be further improved in Section III-B.

The technique used in the proof of Theorem 14 can be applied slightly differently to obtain lower bounds on $FP(s,k)$ for specific parameters s and k.

Suppose we have a functional k-PIR code with dimension s and length n. For each request $\mathbf{v}$, we have k disjoint subsets of $[n]$, $R_1,\ldots,R_k$, where each one of them is a recovery set for $\mathbf{v}$. For each such request $\mathbf{v}$ we choose arbitrarily such k recovery sets. Therefore, $k(2^s-1)$ distinct recovery sets are chosen. Let $\Lambda(s)$ be the sum of the size of all these recovery sets. Since the k recovery sets $R_1,\ldots,R_k$ for any request $\mathbf{v}$ are pairwise disjoint, it follows that $\sum_{i=1}^{k}|R_i|\leq n$, which implies that

$$\Lambda(s) \leq n(2^s-1)\ . \tag{3}$$

On the other hand, a lower bound of $\Lambda(s)$ can be obtained by choosing the recovery sets with smallest size as possible, since the size of the recovery sets by such a choice will be a lower bound on the actual size. There are $k(2^s-1)$ distinct recovery sets. Let d be the largest integer such that

$$\sum_{i=1}^{d}\binom{n}{i} \leq k(2^s-1)\ . \tag{4}$$

The smallest lower bound $\Lambda(s)$ will be obtained if all the $\sum_{i=1}^{d}\binom{n}{i}$ subsets of size d or less will be included as recovery sets. It implies that in the chosen $k(2^s-1)$ recovery sets, at least $k(2^s-1)-\sum_{i=1}^{d}\binom{n}{i}$ subsets of size $d+1$ or greater than $d+1$, are included to obtain the lower bound. Therefore,

$$\sum_{i=1}^{d} i\binom{n}{i} + (d+1)\left(k(2^s-1) - \sum_{i=1}^{d}\binom{n}{i}\right) \leq \Lambda(s). \tag{5}$$

The lower bound on $FP(s,k)$ is obtained by comparing (3) and (5), i.e., finding the minimum n for which

$$\sum_{i=1}^{d} i\binom{n}{i} + (d+1)\left(k(2^s-1) - \sum_{i=1}^{d}\binom{n}{i}\right) \leq n(2^s-1).$$

Example 4: Assume that $FP(6,8)=20$, and apply (4) for $s=6$, $k=8$ and $n=20$, i.e.,

$$\binom{20}{1}+\binom{20}{2} = 210 < 8\cdot(2^6-1) = 504$$

and

$$\binom{20}{1}+\binom{20}{2}+\binom{20}{3} = 1350 > 8\cdot(2^6-1) = 504\ .$$

Since in this code of length 20, a total of $8\cdot(2^6-1)=504$ recovery sets are required, it follows that there are at least $504-210=294$ recovery sets of size at least three. Therefore by (5),

$$\binom{20}{1}+2\binom{20}{2}+3\cdot 294 = 1282 \leq \Lambda(6)\ ,$$

which is a contradiction to $\Lambda(6)\leq 20\cdot(2^6-1)=1260$ by (3). Thus, $FP(6,8)>20$ and since by Theorem 8, $FP(6,8)\leq 21$, it follows that $FP(6,8)=21$.

TABLE XIII
NUMERICAL RESULTS ON $FP(s,k)$

s \ k	6	8	10	12	14	16
1	6	8	10	12	14	16
2	9	12	15	18	21	24
3	11	14	18-19	21-22	25	28
4	12-14	15-18	19-21	23-24	27	30
5	15-16	18-20	22-26	25-30	28-34	31-38
6	16-17	21	25-30	29-34	33-38	37-42
7	17-20	22-25	27-32	32-36	37-40	41-44
8	19-22	23-27	29-33	34-37	39-41	44-45
9	21-23	26-28	31-38	35-43	41-48	46-53
10	22-26	28-32	34-42	39-47	43-52	47-57
11	24-28	30-34	36-44	42-49	47-54	52-59
12	26-29	31-35	38-45	45-50	51-55	57-60
13	28-32	34-39	39-50	46-56	53-62	60-68
14	29-34	36-41	42-54	47-60	55-66	62-72
15	30-35	38-42	45-56	51-62	57-68	63-74
16	32-38	39-46	47-57	55-63	61-69	67-75
17	34-40	41-48	49-62	57-69	65-76	72-83
18	35-41	44-49	50-66	59-73	67-80	75-87
19	37-44	46-53	54-68	60-75	69-82	78-89
20	39-46	47-55	56-69	64-76	71-83	79-90
21	40-47	49-56	58-74	67-82	75-90	82-98
22	41-50	51-60	59-78	69-86	79-94	87-102
23	43-52	53-62	62-80	71-88	81-96	91-104
24	45-53	55-63	65-81	73-89	83-97	93-105
25	46-56	56-67	67-86	77-95	84-104	95-113
26	47-58	59-69	69-90	80-99	89-108	97-117
27	49-59	61-70	70-92	82-101	92-110	102-119
28	51-62	62-74	73-93	83-102	95-111	106-120
29	52-64	63-76	76-98	85-108	97-118	108-128
30	54-65	66-77	78-102	89-112	98-122	110-132
31	56-68	68-81	79-104	92-114	103-124	111-134
32	57-70	70-83	81-105	94-115	106-125	116-135

The exact values for $s = 1$ are trivial and the exact values for $s = 2$ are given in Example 5. For $s \geq 3$, the lower bounds are derived by the counting method while the upper bounds are by the main construction in Theorem 11 and Remark 1.

Example 5: When k is even we have $FP(2,k) \leq \frac{3k}{2}$ (encode the two information symbols x_1 and x_2 into x_1, x_2, and $x_1 + x_2$; each one of these three encoded symbol will appear $\frac{k}{2}$ times in the code.)

Assume now that $n = FP(2,k) \leq \frac{3k}{2} - 1$ and apply (5) for $s = 2$, k and $n = \frac{3k}{2} - 1$. For each request, three recovery sets are required for a total of $3k$ recovery sets. There are at most $n = \frac{3k}{2} - 1$ recovery sets of size 1. Therefore, there are at least $\frac{3k}{2} + 1$ recovery sets whose size at least two. Hence, by (5),

$$(\frac{3k}{2} - 1) + 2 \cdot (\frac{3k}{2} + 1) = 3 \cdot \frac{3k}{2} + 1 \leq \Lambda(2) \ .$$

By (3), $\Lambda(2) \leq n \cdot (2^2 - 1) = 3 \cdot \frac{3k}{2} - 3$, a contradiction.

Therefore, $FP(2,k) > \frac{3k}{2} - 1$ and thus $FP(2,k) = \frac{3k}{2}$ when k is even.

Table XIII contains some specific bounds on $FP(s,k)$ for $s \leq 32$ and $6 \leq k \leq 16$, where k is even.

B. A Tight Bound of $FP(s,3)$ and $FP(s,4)$

This subsection is devoted to analyzing $FP(s,3)$ and $FP(s,4)$. Recall that by Lemma 6, a functional PIR code can be always assumed to be systematic.

Let $\left\{ {t \atop b} \right\}$ be the Stirling number of the second kind, which calculates the number of partitions of $[t]$ into b nonempty subsets. It is well known that

$$\left\{ {t \atop b} \right\} = \frac{1}{b!} \sum_{i=0}^{b} (-1)^{b-i} \binom{b}{i} i^t.$$

Now, we derive the following lower bound on $FP(s,3)$.

Theorem 15: For any given $s \geq 3$ we have that

$$FP(s,3) \geq \begin{cases} \frac{3}{2}s + 2 & \text{if s is even} \\ \frac{3}{2}(s+1) & \text{if s is odd} \end{cases} .$$

Proof: Clearly, $FP(s,3) = s + t$, where $t \geq 0$. The $s \times (s+t)$ matrix $\mathbf{G}$ representing the functional 3-PIR code is

of the form $\mathbf{G} = [\mathbf{I}_s\ \mathbf{U}]$, where $\mathbf{I}_s$ is the $s \times s$ identity matrix. The columns of $\mathbf{U}$ are denoted by $\{\mathbf{u}_1, \ldots, \mathbf{u}_t\}$.

A nonzero requested (column) vector $\mathbf{v}$ can be recovered as $\mathbf{v} = \sum_{i\in I_1} \mathbf{e}_i + \sum_{j\in U_1} \mathbf{u}_j = \sum_{i\in I_2} \mathbf{e}_i + \sum_{j\in U_2} \mathbf{u}_j = \sum_{i\in I_3} \mathbf{e}_i + \sum_{j\in U_3} \mathbf{u}_j$, where I_1, I_2, I_3 are three pairwise disjoint subsets of $[s]$ and U_1, U_2, U_3 are three pairwise disjoint subsets of $[t]$. The unordered triple $\{U_1, U_2, U_3\}$ will be called a *feasible triple* corresponding to the requested vector $\mathbf{v}$. W.l.o.g. if we have $U_1 = U_2 = \varnothing$ then I_1 and I_2 have the same indices for unit vectors which sum to $\mathbf{v}$, contradicting the disjointness of I_1 and I_2. Therefore, in a feasible triple at most one of U_1, U_2, U_3 is empty.

Next, it is claimed that no two requested vectors share a common feasible triple.

To prove the claim let $\{U_1, U_2, U_3\}$ be a feasible triple and let $\mathbf{w}_j$ be the sum of the columns related to U_j, $1 \le j \le 3$. The requested vector $\mathbf{v}$ is recovered based on $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$ and some unit vectors. Note that each $\mathbf{e}_i$ can be used only once to recover $\mathbf{v}$. Therefore, $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$ determine a unique request vector $\mathbf{v}$. This can be observed as follows by considering each coordinate of $\mathbf{v}$ and the related coordinate in $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$. Consider now the i-th coordinate, $1 \le i \le s$.

Assume the triple obtained from the value of the triple ($\mathbf{w}_1$,$\mathbf{w}_2$,$\mathbf{w}_3$) in the i-th coordinate is $(0, 0, 1)$. If the i-th coordinate of $\mathbf{v}$ is *one* then we must have $\mathbf{e}_i$ in both I_1 and I_2, contradicting the fact that $\mathbf{e}_i$ can be used only once. Therefore, the value of the i-th coordinate of $\mathbf{v}$ is *zero*.

Similarly, the value of the i-th coordinate of $\mathbf{v}$ is *zero* if the value of the triple ($\mathbf{w}_1$,$\mathbf{w}_2$,$\mathbf{w}_3$) in the i-th coordinate is $(0, 1, 0)$, $(1, 0, 0)$, or $(0, 0, 0)$. The value of the i-th coordinate of $\mathbf{v}$ is *one* if the value of the triple ($\mathbf{w}_1$,$\mathbf{w}_2$,$\mathbf{w}_3$) in the i-th coordinate is $(0, 1, 1)$, $(1, 0, 1)$, $(1, 1, 0)$, or $(1, 1, 1)$.

Therefore, the requested vector $\mathbf{v}$ is uniquely determined by U_1, U_2, and U_3. Thus, no two requested vectors share a common feasible triple which completes the proof of the claim.

Let $U_4 \triangleq [t] \setminus (U_1 \bigcup U_2 \bigcup U_3)$ and distinguish between the following four cases in counting the number of feasible triples $\{U_1, U_2, U_3\}$:

1) If each one of U_1, U_2, U_3, and U_4 is nonempty, then the number of feasible triples is the same as the number of partitions of $[t]$ into four nonempty subsets, where one of them is chosen to be U_4. The number of such partitions, i.e. feasible triples, is $4\left\{{t \atop 4}\right\}$.
2) If each of U_1, U_2, and U_3 is nonempty and U_4 is empty, then the number of feasible triples is the same as the number of partitions of $[t]$ into three nonempty subsets. Hence, number of such feasible triples is $\left\{{t \atop 3}\right\}$.
3) If exactly one of U_1, U_2, and U_3 is empty and U_4 is nonempty, then the number of feasible triples is the same as the number of partitions of $[t]$ into three nonempty subsets, where one of them is chosen to be U_4. Hence, the number of such feasible triple is $3\left\{{t \atop 3}\right\}$.
4) If exactly one of U_1, U_2, and U_3 is empty and U_4 is empty, then the number of feasible triples is the same as the number of partitions of $[t]$ into two nonempty subsets. Therefore, number of such feasible triples is $\left\{{t \atop 2}\right\}$.

Thus, the number of feasible triples is at most

$$4\left\{{t \atop 4}\right\} + 4\left\{{t \atop 3}\right\} + \left\{{t \atop 2}\right\} = \frac{4^t}{6} - 2^{t-1} + \frac{1}{3}.$$

On the other hand, we proved that no two requested vectors share a common feasible triple. Hence, there are at least $2^s - 1$ feasible triples and this implies that

$$2^s - 1 \le \frac{4^t}{6} - 2^{t-1} + \frac{1}{3}\ .$$

Thus, $t > \frac{s+\log 6}{2}$. ∎

The lower bound of Theorem 15 can be combined with the bounds of Theorem 5 to obtain lower bounds on $FP(s, k)$ for $k > 3$. In particular we have.

Corollary 16: For any $s \ge 3$ we have

$$FP(s,4) \ge \begin{cases} \frac{3}{2}s + 3 & \text{if s is even} \\ \frac{3}{2}(s+1) + 1 & \text{if s is odd} \end{cases}.$$

Considering Theorem 15, Theorem 8, Theorem 5, Corollary 16 and Remark 1, we have that

Corollary 17: For any $t \ge 2$, $FP(2t, 3) = 3t + 2$, $FP(2t, 4) = 3t + 3$, $3t + 3 \le FP(2t + 1, 3) \le 3t + 4$ and $3t + 4 \le FP(2t + 1, 4) \le 3t + 5$.

IV. Bounds on the Length of Functional Batch Codes

In this section a random construction of functional batch codes is presented. The random construction relies on a well-known result of random constructions for linear codes which attain the sphere-covering bound [4], [5].

Definition 18: For a binary code $\mathcal{C}$ of length n, the *covering radius* is the smallest integer R such that for any $\mathbf{v} \in \mathbb{F}_2^n$, there exists $\mathbf{u} \in \mathcal{C}$ such that $d(\mathbf{v}, \mathbf{u}) \le R$. The code $\mathcal{C}$ is a code with covering radius R.

Proposition 19 [14]: If $\mathcal{C}$ is a binary linear code of length n, and dimension k, with a parity check matrix $\mathbf{H}$, then $\mathcal{C}$ has covering radius R if and only if every column vector $\mathbb{F}_2^{n-k}$ is the sum of at most R columns of $\mathbf{H}$.

Let $V(n, R)$ be the size of the Hamming ball of radius R. A code with covering radius R has at least $\frac{2^n}{V(n,R)}$ codewords and thus a linear code with covering radius R has dimension $k \ge n - \log V(n, R)$. This is the sphere covering bound for linear codes. Blinovskii [4], [5] proved that almost all linear codes attain the sphere covering bound (see also [10, Ch. 12, p. 325] and the references therein).

Theorem 20: Let $0 \le \rho < 1/2$, $\mathcal{C}_{k,n}$ be the ensemble of 2^{kn} linear codes generated by all possible binary $k \times n$ matrices, and $R_n = \lfloor \rho n \rfloor$. There exists a sequence k_n for which

$$k_n/n \le 1 - H(\rho) + O(n^{-1} \log n)\ ,$$

such that the fraction of codes $C_n \in \mathcal{C}_{k_n,n}$ which have covering radius R_n tends to 1, when n tends to infinity.

In other words, Theorem 20 implies that if a binary random matrix $\mathcal{H}$ of size $s \times n$ is considered as a parity check matrix of a linear code, then the covering radius $R = \rho n$ of the code satisfies $H(\rho) \sim \frac{s}{n}$ with probability tending to 1, when n tends to infinity, i.e., any column vector of length s is the sum of at most R columns of $\mathcal{H}$.

Cooper [15] proved the following result on the invertibility of random binary matrices.

Theorem 21: Let $\mathbf{G}$ be a random binary matrix of size $s \times s$, where each entry is independently and identically distributed with $\Pr[\mathbf{G}_{i,j} = 1] = p(s)$. If $\min\{p(s), 1-p(s)\} \geq (\log s + d(s))/s$ for any $d(s) \to \infty$, then $\Pr[\mathbf{G}$ is invertible$]$ tends to a constant $c \approx 0.28879$, when s tends to infinity.

We are now in a position to present the random construction of functional batch codes. The idea is illustrated first with an example on functional 2-batch codes. For sufficiently large s, randomly choose a binary matrix of size $s \times n$ to represent the functional 2-batch code. Let $\mathbf{u}, \mathbf{v}$ be two arbitrary requests. By Theorem 20, with probability tending to 1, when s and n tend to infinity, the request $\mathbf{u}$ can be recovered as a sum of ρn columns, where $H(\rho) \sim \frac{s}{n}$. The remaining matrix is a random matrix of size $s \times (1-\rho)n$. If $(1-\rho)n > s$, then by Theorem 21, it has an $s \times s$ invertible sub-matrix with probability $c \approx 0.28879$. Using the columns from this invertible sub-matrix, the request $\mathbf{v}$ can be recovered. Therefore, under the constraints $(1-\rho)n > s, H(\rho) \sim \frac{s}{n}$, there exists a binary matrix of size $s \times n$ representing a functional 2-batch code when s and n are sufficiently large. To find the asymptotic relation between n and s, note that the constraints require $s/n \sim H(\rho) < 1 - \rho$. The root of $1 - \rho = H(\rho)$ is $\rho = 0.227$ and thus we can set $n \sim 1.2937s$. The next theorem generalizes this idea to arbitrary functional k-batch codes.

Theorem 22: If $c_1 = \frac{1}{2}$ and c_{k+1} is the root of the polynomial $H(z) = H(c_k) - zH(c_k)$, then

$$\lim_{s\to\infty} \frac{FB(s,k)}{s} \leq \frac{1}{H(c_k)}.$$

Proof: For a sufficiently large s, randomly choose an $s \times n_1$ binary matrix $\mathbf{G}_1$ to represent the functional k-batch code. With probability tending to 1 the first request can be recovered as a sum of $\rho_1\, n_1$ columns of $\mathbf{G}_1$, where $H(\rho_1) \sim \frac{s}{n_1}$. Let $\mathbf{G}_2$ be the matrix obtained by removing these $\rho_1\, n_1$ columns from $\mathbf{G}_1$. $\mathbf{G}_2$ is an $s \times n_2$ random matrix, where $n_2 = (1-\rho_1)n_1$. The second request can be recovered, with probability which tends to 1, as a sum of $\rho_2\, n_2$ columns on $\mathbf{G}_2$, where $H(\rho_2) \sim \frac{s}{n_2}$. This procedure continues and for the j-th request, $1 \leq j \leq k-1$, we have a matrix $\mathbf{G}_j$. The j-th request can be recovered, with probability tending to 1, as a sum of $\rho_j n_j$ columns, where $H(\rho_j) \sim \frac{s}{n_j}$ and $n_j = \prod_{i=1}^{j-1}(1-\rho_i)n_1$. Finally, for the k-th request, we have to show that the remaining matrix $\mathbf{G}_k$ contains an $s \times s$ invertible sub-matrix. This is guaranteed by Theorem 21 with positive probability $c \approx 0.28879$ as long as $s < n_k = \prod_{i=1}^{k-1}(1-\rho_i)n_1$ for sufficiently large s. Therefore, we have a binary matrix of size $s \times n_1$ representing a functional k-batch code if $s < n_k = \prod_{i=1}^{k-1}(1-\rho_i)n_1$.

To complete the proof we have to derive the asymptotic relation between n_1 and s. Note first that

$$\frac{s}{n_1} \sim H(\rho_1) \sim H(\rho_2)(1-\rho_1) \sim \cdots \sim H(\rho_{k-1}) \prod_{i=1}^{k-2}(1-\rho_i) < \prod_{i=1}^{k-1}(1-\rho_i).$$

Hence, to maximize $\frac{s}{n_1}$, we should have $H(\rho_{k-1}) = 1-\rho_{k-1}$, $H(\rho_{k-2}) = H(\rho_{k-1})(1-\rho_{k-2})$, $\ldots$, $H(\rho_j) = H(\rho_{j+1})(1-\rho_j)$, $\ldots$, $H(\rho_1) = H(\rho_2)(1-\rho_1)$. Therefore, we set $\rho_{k-1} = c_2$, $\rho_{k-2} = c_3$, $\ldots$, $\rho_1 = c_k$ and thus asymptotically we have $n_1 \sim \frac{s}{H(c_k)}$. ∎

A lower bound of $FB(s,k)$ can be derived as follows.

Theorem 23:

$$\lim_{s\to\infty} \frac{FB(s,k)}{s} \geq \frac{k}{\log(k+1)}.$$

Proof: Assume there is a functional k-batch code of length n and dimension s, represented by an $s \times n$ matrix $\mathbf{G}$. For any recovery process of a request $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_k)$ with k vectors of length s, assign a label to each column of $\mathbf{G}$. The label is either 0 or some i, $1 \leq i \leq k$. A label 0 indicates that the column is not used in the recovery process of $\mathbf{v}$. A label i, indicates that the column is used in the recovery set for $\mathbf{v}_i$. Then the labeling of $\mathbf{G}$ for the request $\mathbf{v}$ is an element in $\{0, 1, \ldots, k\}^n$. For any two different ordered k-tuples of request vectors $(\mathbf{v}_1, \ldots, \mathbf{v}_k)$ and $(\mathbf{u}_1, \ldots, \mathbf{u}_k)$, where $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are k distinct vectors and $\mathbf{u}_1, \ldots, \mathbf{u}_k$ are also k distinct vectors, the labeling of $\mathbf{G}$ must be different. Therefore, $(k+1)^n \geq \binom{2^s-1}{k} k!$.

Thus,

$$\lim_{s\to\infty} \frac{n}{s} \geq \frac{k}{\log(k+1)},$$

which completes the proof. ∎

Table XIV summarizes the lower and upper bounds of $\lim_{s\to\infty} \frac{FB(s,k)}{s}$.

V. Simplex Codes as Functional Batch Codes

The family of simplex codes is an important class of codes, which has been analyzed in the literature of PIR and batch codes. In [19] it was shown that $P(r, 2^{r-1}) = 2^r - 1$ and in [39] it was proved that $B(r, 2^{r-1}) = 2^r - 1$. Furthermore, in Theorem 7, we also confirmed that by using simplex codes we have $FP(r, 2^{r-1}) = 2^r - 1$. An intuitive explanation is that the simplex codes contain every possible linear combination of information symbols and therefore their availability might be easier to analyze. Moreover, the behaviour of simplex codes will also shed light on the construction of better codes. For example, recall our construction of functional PIR codes in Section II, each row of the array representation is indeed a simplex code. Hence, in this section we analyze whether the same property is also valid for functional batch codes, that is, whether $FB(r, 2^{r-1}) = 2^r - 1$ holds.

Definition 24: A $[2^r - 1, r]$ simplex code is a linear code of length $n = 2^r - 1$ and dimension r whose $r \times n$ generator matrix $\mathbf{G}$ contains each nonzero column vector $\mathbf{z}$ of length r exactly once as a column.

Simplex codes have been used for several more applications, among them are write-once memory (WOM) codes and random I/O (RIO) codes. An $[n, k, t]$ WOM code is a coding scheme comprising of n binary cells such that it is possible to write a k-bit message t times while on each write the cell values can only change from zero to one. An (n, k, t) RIO code assumes that t k-bit messages are stored in n cells each

TABLE XIV
LOWER AND UPPER BOUNDS OF $\lim_{s\to\infty}\frac{FB(s,k)}{s}$ (BY THEOREMS 22 AND 23)

k	2	3	4	5	6
$\lim_{s\to\infty}\frac{FB(s,k)}{s}$	1.2619-1.2937	1.5000-1.5489	1.7227-1.7828	1.9343-2.0028	2.1372-2.2124
k	7	8	9	10	11
$\lim_{s\to\infty}\frac{FB(s,k)}{s}$	2.3333-2.4137	2.5237-2.6089	2.7093-2.7984	2.8906-2.9834	3.0684-3.1641
k	12	13	14	15	16
$\lim_{s\to\infty}\frac{FB(s,k)}{s}$	3.2429-3.3414	3.4144-3.5156	3.5834-3.6869	3.7500-3.8557	3.9144-4.0222
k	17	18	19	20	21
$\lim_{s\to\infty}\frac{FB(s,k)}{s}$	4.0768-4.1865	4.2374-4.3489	4.3962-4.5094	4.5534-4.6683	4.7091-4.8256
k	22	23	24	25	26
$\lim_{s\to\infty}\frac{FB(s,k)}{s}$	4.8634-4.9814	5.0164-5.1358	5.1681-5.2889	5.3187-5.4407	5.4681-5.5914
k	27	28	29	30	31
$\lim_{s\to\infty}\frac{FB(s,k)}{s}$	5.6164-5.7410	5.7637-5.8895	5.9101-6.0369	6.0555-6.1835	6.2000-6.3291

with $t+1$ levels such that every page can be read by sensing a single read threshold. In [44], it was proved that these two families of codes are equivalent and a new variation of RIO codes, called *parallel RIO codes*, has been proposed, where all messages can be written together and thereby can allow the design of codes with parameters that do not exist for WOM codes.

While there are several constructions of WOM codes, we focus here on the one called *linear WOM codes* [13] in which a binary matrix is used to encode messages by the syndromes of parity check matrices of error-correcting codes. The authors of [13] studied this linear construction using Golay codes as well as simplex codes. In particular, the latter family of codes provided WOM codes with the parameters $[2^r-1, r, 2^{r-2}+2]$. Later, this result has been improved by Godlewski [20], who showed the existence of $[2^r-1, r, 2^{r-2}+2^{r-4}+1]$ WOM codes.

The family of parallel RIO codes is very similar to the one of functional batch codes. In fact, if parallel RIO codes are constructed using linear codes and their parity check matrices, such as in [13], [20], then these codes are in essence functional batch codes as well. This approach to construct parallel RIO codes has been initiated recently by Yamawaki, Kamabe, and Lu in [45], where they studied the parameters of parallel RIO codes using simplex codes and showed the construction of $(7,3,4)$ and $(15,4,8)$ parallel RIO codes. These codes assure also that $FB(3,4)=7$ and $FB(4,8)=15$. We also verified that a $(31,5,16)$ parallel RIO code exists which implies that $FB(5,16)=31$, while similarly to the conjecture raised in [45] we also have the following conjecture.

Conjecture 25: The $[2^r-1, r]$ simplex code is a functional 2^{r-1}-batch code and therefore $FB(r, 2^{r-1}) = 2^r-1$.

Remember that for WOM codes the message requests are received in a sequential order and each recovery set should be determined without knowing the upcoming requests. The main idea of the construction for $[2^r-1, r, 2^{r-2}+2^{r-4}+1]$ WOM codes by Godlewski [20] with simplex codes works as follows.

1) The first request $\mathbf{v}$ is simply satisfied by using $\mathbf{v}$ itself.
2) As long as there are at least 2^{r-1} nonzero available vectors, each request $\mathbf{v}$ can always be satisfied by finding a pair $\{\mathbf{u}, \mathbf{u}+\mathbf{v}\}$. This process can satisfy at least 2^{r-2} more requests and only stops when the number of unused vectors is less than 2^{r-1}.
3) The key part of Godlewski's construction is that it is still possible to find recovery sets of size four unless the number of unused vectors is less than 2^{r-2}. Thus in this process 2^{r-4} additional write requests can be satisfied.

To summarize, simplex codes can be used to satisfy roughly any $\frac{5}{16}2^r$ write requests, when considered as WOM codes. Since in the functional batch setting (or in parallel RIO codes) we know all the requests in advance, it is possible to make use of this knowledge and improve upon the $2^{r-2}+2^{r-4}+1$ result. This improvement comes either from the choice of many recovery sets of size one, or from a predetermined usage of the 2^{r-2} remaining vectors in Godlewski's method. Namely, we prove the following theorem.

Theorem 26: The $[2^r-1, r]$ simplex code can be used as a functional $(2^{r-2}+2^{r-4}+\lfloor\frac{2^{r/2}}{\sqrt{24}}\rfloor)$-batch code.

Proof: Consider $\gamma = 2^{r-2}+2^{r-4}+\lfloor\frac{2^{r/2}}{\sqrt{24}}\rfloor$ requests which consist of Δ distinct vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_\Delta\}$. To prove that the simplex code is a γ-functional batch code, we distinguish between the following two cases depending on the value of Δ:

Case 1: If $\Delta \geq \frac{2^{r/2}}{\sqrt{6}}$, we use the Δ subsets of size one of the set $\{\mathbf{v}_1, \ldots, \mathbf{v}_\Delta\}$ as recovery sets of size one. For the remaining $\gamma-\Delta$ requests, we follow Godlewski's method. The number of unused vectors is $2^r-1-\Delta$. Recovery sets of size two can be found until the number of unused vectors is less than 2^{r-1}. Hence, the number of recovery sets of size two is $\frac{2^r-1-\Delta-(2^{r-1}-1)}{2}$ (if Δ is even) or $\frac{2^r-1-\Delta-(2^{r-1}-2)}{2}$ (if Δ is odd), i.e., $2^{r-2}-\lfloor\frac{\Delta}{2}\rfloor$. Similarly, recovery sets of size four can be found until the number of unused vectors is less than 2^{r-2}, yielding 2^{r-4} recovery sets. Therefore, when $\Delta \geq \frac{2^{r/2}}{\sqrt{6}}$, the simplex code satisfies any $2^{r-2}+2^{r-4}+\Delta-\lfloor\frac{\Delta}{2}\rfloor \geq \gamma$ requests.

Case 2: If $\Delta < \frac{2^{r/2}}{\sqrt{6}}$, let $\mathbf{v}_1$ be the vector which is requested the largest number of times. Clearly, $\mathbf{v}_1$ is requested at least $\lceil\frac{\gamma}{\Delta}\rceil$ times and the number of requests other than $\mathbf{v}_1$ is at most $\gamma-\lceil\frac{\gamma}{\Delta}\rceil$ times.

Partition all the 2^r vectors (including the zero vector) into 2^{r-1} pairs of the form $\{\mathbf{u}, \mathbf{u}+\mathbf{v}_1\}$. The two vectors in the same pair are called conjugates of each other. A pair containing no requested vectors is called a *good* pair and the vectors lying in good pairs are called *good* vectors. The number of good vectors is then at least $2^r - 2\Delta$.

For any $\mathbf{v}_j \neq \mathbf{v}_1$ which is requested an odd number of times, $\mathbf{v}_j$ is considered as a recovery set of size one. Hence, now each such $\mathbf{v}_j$ is requested an even number of times. For these requests we find recovery sets using only good vectors similarly to Godlewski's method. Let $\{\mathbf{x}, \mathbf{y}\}$ be a recovery set of size two for $\mathbf{v}_j$, i.e., $\mathbf{v}_j = \mathbf{x} + \mathbf{y}$, where $\mathbf{x}$ and $\mathbf{y}$ are good vectors. $\mathbf{x}$ and $\mathbf{y}$ are not conjugate since $\mathbf{v}_j \neq \mathbf{v}_1$). Hence, their conjugates form another recovery set for $\mathbf{v}_j$, i.e. $\mathbf{v}_j = (\mathbf{x}+\mathbf{v}_1) + (\mathbf{y}+\mathbf{v}_1)$. Similarly, whenever a recovery set of size four for $\mathbf{v}_j$ is found among the good vectors, then there are only two possibilities. On one hand if we have $\mathbf{v}_j = \mathbf{x}+\mathbf{y}+\mathbf{z}+\mathbf{w}$ where no two of the four vectors $\{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w}\}$ are conjugate, then their conjugates form another recovery set $\mathbf{v}_j = (\mathbf{x}+\mathbf{v}_1)+(\mathbf{y}+\mathbf{v}_1)+(\mathbf{z}+\mathbf{v}_1)+(\mathbf{w}+\mathbf{v}_1)$. On the other hand if we have $\mathbf{v}_j = \mathbf{x}+\mathbf{y}+\mathbf{z}+(\mathbf{z}+\mathbf{v}_1)$, then we construct another recovery set $\mathbf{v}_j = (\mathbf{x}+\mathbf{v}_1)+(\mathbf{y}+\mathbf{v}_1)+\mathbf{w}+(\mathbf{w}+\mathbf{v}_1)$, where the good pair $\{\mathbf{w}, \mathbf{w}+\mathbf{v}_1\}$ is chosen arbitrarily from the unused good pairs. After performing this strategy for requests other than $\mathbf{v}_1$ using the modified Godlewski's method, the remaining good vectors will appear in pairs where each pair sums up to $\mathbf{v}_1$. These remaining good pairs will be used for recovering $\mathbf{v}_1$.

To complete the proof we have to show that there exist enough recovery sets. We distinguish between three subcases depending on the number of times λ that $\mathbf{v}_1$ is requested:

Case 2.1: If $\lambda \leq 2^{r-3}$ times, then there are at least $\frac{2^r-2\Delta-(2^{r-1}-2)}{2} = 2^{r-2} - \Delta + 1$ recovery sets of size two and 2^{r-4} recovery sets of size four for the queries which are different from $\mathbf{v}_1$. This satisfies the requirements since the number of queries other than $\mathbf{v}_1$ is upper bounded by

$$\begin{aligned}\gamma - \lceil \frac{\gamma}{\Delta} \rceil &\leq 2^{r-2} + 2^{r-4} + \lfloor \frac{2^{r/2}}{\sqrt{24}} \rfloor - \frac{2^{r-2}+2^{r-4}}{2^{r/2}/\sqrt{6}} \\ &\leq 2^{r-2} + 2^{r-4} + \lfloor \frac{2^{r/2}}{\sqrt{24}} \rfloor - 2^{r/2} \cdot \frac{5\sqrt{6}}{16} \\ &\leq 2^{r-2} + 2^{r-4} - \frac{2^{r/2}}{\sqrt{6}} \\ &\leq 2^{r-2} + 2^{r-4} - \Delta.\end{aligned}$$

Meanwhile, when this modified Godlewski's method concludes, there are still 2^{r-2} good vectors constituting 2^{r-3} pairs for recovering $\mathbf{v}_1$.

Case 2.2: If $\lambda \geq \gamma + \Delta - 2^{r-2}$, then the total number of requests different than $\mathbf{v}_1$ is $\gamma - \lambda$. Hence, the modified Godlewski's method concludes after we choose $\gamma - \lambda$ recovery sets of size two. Initially, there are at least $2^{r-1} - \Delta$ good pairs, among which $\gamma - \lambda$ pairs are involved in recovery sets of size two (since in the modified Godlewski's method every two conjugate recovery sets of size two together occupy two good pairs). Therefore, the number of remaining good pairs is

$$\begin{aligned}&2^{r-1} - \Delta - (\gamma - \lambda) \\ &\quad\geq 2^{r-1} - \frac{2^{r/2}}{\sqrt{6}} - (2^{r-2} + 2^{r-4} + \lfloor \frac{2^{r/2}}{\sqrt{24}} \rfloor) + \lambda \\ &\quad\geq 2^{r-2} - 2^{r-4} - 2^{r/2} \cdot \frac{3}{2\sqrt{6}} + \lambda \\ &\quad\geq \lambda,\end{aligned}$$

where the last inequality holds for $r \geq 6$. Thus, there are enough pairs to be used as recovery sets for $\mathbf{v}_1$.

Case 2.3: If $2^{r-3} < \lambda < \gamma + \Delta - 2^{r-2}$, then the modified Godlewski's method concludes after we choose $2^{r-2} - \Delta$ recovery sets of size two and $\gamma - \lambda - 2^{r-2} + \Delta$ recovery sets of size four. Initially, there are $2^{r-1} - \Delta$ good pairs, among which $2^{r-2} - \Delta$ pairs are involved in recovery sets of size two and $2(\gamma - \lambda - 2^{r-2} + \Delta)$ recovery sets are involved in recovery sets of size four (since in the modified Godlewski's method every two conjugate recovery sets of size two together occupy two good pairs and every two conjugate recovery sets of size four together occupy four good pairs). Thus, the number of remaining good pairs is

$$\begin{aligned}&2^{r-1} - \Delta - (2^{r-2} - \Delta) - 2(\gamma - \lambda - 2^{r-2} + \Delta) \\ &\quad= 2^{r-1} + 2^{r-2} - 2\gamma - 2\Delta + 2\lambda \\ &\quad\geq 2^{r-2} - 2^{r/2} \cdot \frac{3}{\sqrt{6}} + \lambda \qquad (6) \\ &\quad\geq \lambda,\end{aligned}$$

where (6) is derived by plugging the values of $\gamma = 2^{r-2} + 2^{r-4} + \lfloor \frac{2^{r/2}}{\sqrt{24}} \rfloor$, $\Delta < \frac{2^{r/2}}{\sqrt{6}}$, and $\lambda > 2^{r-3}$. Finally, the last inequality holds for $r \geq 5$. Therefore, there are enough pairs for recovering $\mathbf{v}_1$.

Thus, the $[2^r - 1, r]$ simplex code can satisfy any $2^{r-2} + 2^{r-4} + \lfloor \frac{2^{r/2}}{\sqrt{24}} \rfloor$ requests. ■

VI. Conclusion and Future Research

We have considered the shortest length of functional PIR and functional batch codes. Several upper bounds, based on explicit constructions and random ones, are given. Several methods which yield lower bounds are also presented. In particular connections to WOM codes and RIO codes are derived and the parameters of the simplex code when used as a functional batch code are discussed.

There are plenty of problems which remain for future research, some of them are briefly outlined.

1) Prove or disprove that for any given PIR (batch) code, there exists a systematic PIR (batch) code with the same parameters.
2) We would like to see an upper bound on the length of functional batch codes, which is derived from an explicit construction.
3) We would like to see more tight bounds, general, asymptotic, and for specific parameters.
4) We would like to see a proof (or a counter-example) for Conjecture 25, i.e., the $[2^r - 1, r]$ simplex code is a functional 2^{r-1}-batch code and therefore $FB(r, 2^{r-1}) = 2^r - 1$.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their constructive comments.

REFERENCES

[1] H. Asi and E. Yaakobi, "Nearly optimal constructions of PIR and batch codes," *IEEE Trans. Inf. Theory*, vol. 65, no. 2, pp. 947–964, Feb. 2019.

[2] D. Augot, F. Levy-Dit-Vehel, and A. Shikfa, "A storage-efficient and robust private information retrieval scheme allowing few servers," 2014, *arXiv:1412.5012*. [Online]. Available: http://arxiv.org/abs/1412.5012

[3] S. R. Blackburn, T. Etzion, and M. B. Paterson, "PIR schemes with small download complexity and low storage requirements," *IEEE Trans. Inf. Theory*, vol. 66, no. 1, pp. 557–571, Jan. 2020.

[4] V. M. Blinovskii, "Lower asymptotic bound on the number of linear code words in a sphere of given radius in F_q^n," *Problems Inf. Transmiss.*, vol. 23, no. 2, pp. 50–53, 1987.

[5] V. M. Blinovskii, "Asymptotically exact uniform bounds for spectra of cosets of linear codes," *Problems Inf. Transmiss.*, vol. 26, no. 1, pp. 99–103, 1990.

[6] S. Buzaglo, Y. Cassuto, P. H. Siegel, and E. Yaakobi, "Consecutive switch codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 2485–2498, Apr. 2018.

[7] Y. M. Chee, F. Gao, S. T. H. Teo, and H. Zhang, "Combinatorial systematic switch codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Hong Kong, Jun. 2015, pp. 241–245.

[8] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proc. IEEE 36th Annu. Found. Comput. Sci.*, Milwaukee, WI, USA, Oct. 1995, pp. 41–50.

[9] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.

[10] G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein, *Covering Codes*. Amsterdam, The Netherlands: Elsevier, 1997.

[11] T. H. Chan, S.-W. Ho, and H. Yamamoto, "Private information retrieval for coded storage," 2014, *arXiv:1410.5489*. [Online]. Available: http://arxiv.org/abs/1410.5489

[12] T. H. Chan, S.-W. Ho, and H. Yamamoto, "Private information retrieval for coded storage," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Hong Kong, Jun. 2015, pp. 2842–2846.

[13] G. Cohen, P. Godlewski, and F. Merkx, "Linear binary code for write-once memories," *IEEE Trans. Inf. Theory*, vol. 32, no. 5, pp. 697–700, Sep. 1986.

[14] G. Cohen, M. Karpovsky, H. Mattson, Jr., and J. Schatz, "Covering radius—Survey and recent results," *IEEE Trans. Inf. Theory*, vol. 31, no. 3, pp. 328–343, May 1985.

[15] C. Cooper, "On the rank of random matrices," *Random Struct. Algorithms*, vol. 16, no. 2, pp. 209–232, Mar. 2000.

[16] T. Etzion and A. Vardy, "Error-correcting codes in projective space," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 1165–1173, Feb. 2011.

[17] H. T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish, "Scalable and private media consumption with Popcorn," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement.*, Santa Clara, CA, USA, Mar. 2016, pp. 91–107.

[18] A. Fazeli, A. Vardy, and E. Yaakobi, "Codes for distributed PIR with low storage overhead," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Hong Kong, Jun. 2015, pp. 2852–2856.

[19] A. Fazeli, A. Vardy, and E. Yaakobi, "PIR with low storage overhead: Coding instead of replication," May 2015, *arXiv:1505.06241*. [Online]. Available: https://arxiv.org/abs/1505.06241

[20] P. Godlewski, "WOM-codes construits à partir des codes de Hamming," *Discrete Math.*, vol. 65, no. 3, pp. 237–243, Jul. 1987.

[21] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Batch codes and their applications," in *Proc. 36th Annu. ACM Symp. Theory Comput. (STOC)*. Chicago, IL, USA: ACM, 2004, pp. 262–271.

[22] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: Single database, computationally-private information retrieval," in *Proc. 38th Annu. Symp. Found. Comput. Sci. (FOCS)*, 1997, pp. 364–373.

[23] S. Lin and D. J. Costello, *Error Control Coding*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[24] H.-Y. Lin and E. Rosnes, "Lengthening and extending binary private information retrieval codes," 2017, *arXiv:1707.03495*. [Online]. Available: http://arxiv.org/abs/1707.03495

[25] M. Mirmohseni and M. A. Maddah-Ali, "Private function retrieval," in *Proc. Iran Workshop Commun. Inf. Theory (IWCIT)*, Tehran, Iran, Apr. 2018, pp. 1–6.

[26] N. Polyanskii and I. Vorobyev, "Constructions of batch codes via finite geometry," 2019, *arXiv:1901.06741*. [Online]. Available: http://arxiv.org/abs/1901.06741

[27] N. Polyanskii and I. Vorobyev, "Constructions of batch codes via finite geometry," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Paris, France, Jul. 2019, pp. 360–364.

[28] S. Rao and A. Vardy, "Lower bound on the redundancy of PIR codes," 2016, *arXiv:1605.01869*. [Online]. Available: http://arxiv.org/abs/1605.01869

[29] N. Raviv and D. A. Karpuk, "Private polynomial computation from Lagrange encoding," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 553–563, Jul. 2020.

[30] A. S. Rawat, D. S. Papailiopoulos, A. G. Dimakis, and S. Vishwanath, "Locality and availability in distributed storage," *IEEE Trans. Inf. Theory*, vol. 62, no. 8, pp. 4481–4493, Aug. 2016.

[31] A. S. Rawat, Z. Song, A. G. Dimakis, and A. Gal, "Batch codes through dense graphs without short cycles," *IEEE Trans. Inf. Theory*, vol. 62, no. 4, pp. 1592–1604, Apr. 2016.

[32] R. L. Rivest and A. Shamir, "How to reuse a 'write-once' memory," *Inf. Control*, vol. 55, nos. 1–3, pp. 1–19, Oct. 1982.

[33] N. B. Shah, K. V. Rashmi, and K. Ramchandran, "One extra bit of download ensures perfectly private information retrieval," in *Proc. IEEE Int. Symp. Inf. Theory*, Honolulu, HI, USA, Jun. 2014, pp. 856–860.

[34] E. Sharon and I. Alrod, "Coding scheme for optimizing random I/O performance," in *Proc. Non-Volatile Memories Workshop*, San Diego, CA, USA, Apr. 2013, pp. 1–5.

[35] H. Sun and S. A. Jafar, "The capacity of private computation," 2017, *arXiv:1710.11098*. [Online]. Available: http://arxiv.org/abs/1710.11098

[36] M. Vajha, V. Ramkumar, and P. Vijay Kumar, "Binary, shortened projective reed muller codes for coded private information retrieval," 2017, *arXiv:1702.05074*. [Online]. Available: http://arxiv.org/abs/1702.05074

[37] A. Vardy and E. Yaakobi, "Constructions of batch codes with near-optimal redundancy," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Barcelona, Spain, Jul. 2016, pp. 1197–1201.

[38] Z. Wang, H. M. Kiah, and Y. Cassuto, "Optimal binary switch codes with small query size," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Hong Kong, Jun. 2015, pp. 636–640.

[39] Z. Wang, H. M. Kiah, Y. Cassuto, and J. Bruck, "Switch codes: Codes for fully parallel reconstruction," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 2061–2075, Apr. 2017.

[40] Z. Wang, O. Shaked, Y. Cassuto, and J. Bruck, "Codes for network switches," in *Proc. IEEE Int. Symp. Inf. Theory*, Istanbul, Turkey, Jul. 2013, pp. 1057–1061.

[41] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia, "Splinter: Practical private queries on public data," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement.*, Boston, MA, USA, Mar. 2017, pp. 299–313.

[42] M. Wootters. (Feb. 2016). *Linear Codes With Disjoint Repair Groups*.

[43] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, "Codes for write-once memories," *IEEE Trans. Inf. Theory*, vol. 58, no. 9, pp. 5985–5999, Sep. 2012.

[44] E. Yaakobi and R. Motwani, "Construction of random input-output codes with moderate block lengths," *IEEE Trans. Commun.*, vol. 64, no. 5, pp. 1819–1828, May 2016.

[45] A. Yamawaki, H. Kamabe, and S. Lu, "Construction of parallel RIO codes using coset coding with Hamming codes," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Kaohsiung, Taiwan, Nov. 2017, pp. 239–243.

Yiwei Zhang received the B.A. and Ph.D. degrees in mathematics from Zhejiang University, Hangzhou, Zhejiang, China, in 2011 and 2016, respectively. From 2016 to 2017, he was a Post-Doctoral Researcher at the School of Mathematical Sciences, Capital Normal University, Beijing, China. From 2017 to 2019, he was a Post-Doctoral Researcher at the Department of Computer Science, Technion—Israel Institute of Technology, Haifa, Israel. He is currently a Professor with the School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China. He is also with the Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University. His research interests include coding theory, as well as extremal combinatorics and their interactions.

Tuvi Etzion (Fellow, IEEE) was born in Tel Aviv, Israel, in 1956. He received the B.A., M.Sc., and D.Sc. degrees from the Technion—Israel Institute of Technology, Haifa, Israel, in 1980, 1982, and 1984, respectively.

Since 1984, he has been holding a position with the Department of Computer Science, Technion, where he currently holds the Bernard Elkin Chair in computer science. From 1985 to 1987, he was a Visiting Research Professor with the Department of Electrical Engineering—Systems, University of Southern California, Los Angeles, CA, USA. During the summers of 1990 and 1991, he was visiting Bellcore, Morristown, NJ, USA. From 1994 to 1996, he was a Visiting Research Fellow at the Computer Science Department, Royal Holloway, University of London, Egham, U.K. He also had several visits to the Coordinated Science Laboratory, University of Illinois in Urbana–Champaign, from 1995 to 1998, two visits to HP Bristol in summers of 1996 and 2000, a few visits to the Department of Electrical Engineering, University of California at San Diego, from 2000 to 2017, several visits to the Mathematics Department, Royal Holloway, University of London, from 2007 to 2017, a few visits to the School of Physical and Mathematical Science (SPMS), Nanyang Technological University, and to the Department of Industrial Systems Engineering and Management, National University of Singapore, Singapore, from 2016 to 2019, and a few visits to Jiaotong University, Beijing, from 2017 to 2019. His research interests include applications of discrete mathematics to problems in computer science and information theory, coding theory, network coding, and combinatorial designs.

Dr. Etzion was an Associate Editor for Coding Theory for the IEEE TRANSACTIONS ON INFORMATION THEORY from 2006 to 2009. From 2004 to 2009, he was an Editor of the *Journal of Combinatorial Designs*. Since 2011, he has been an Editor for *Designs, Codes, and Cryptography,* and an Editor for the *Advances of Mathematics in Communications* since 2013.

Eitan Yaakobi (Senior Member, IEEE) received the B.A. degrees in computer science and mathematics, and the M.Sc. degree in computer science from the Technion—Israel Institute of Technology, Haifa, Israel, in 2005 and 2007, respectively, and the Ph.D. degree in electrical engineering from the University of California, San Diego, CA, USA, in 2011. From 2011 to 2013, he was a Post-Doctoral Researcher at the Department of Electrical Engineering, California Institute of Technology, and at the Center for Memory and Recording Research, University of California, San Diego. He is currently an Associate Professor with the Computer Science Department, Technion—Israel Institute of Technology. His research interests include information and coding theory with applications to non-volatile memories, associative memories, DNA storage, data storage and retrieval, and private information retrieval. He received the Marconi Society Young Scholar in 2009, and the Intel Ph.D. Fellowship (2010–2011).