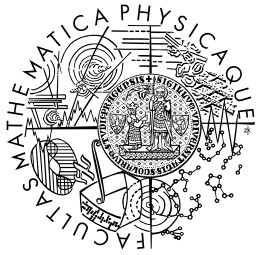


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jana Hrúzová

PIR codes using combinatorial structures

Department of Algebra

Supervisor of the master thesis: Dr. rer. nat. Faruk Gölogluu

Study programme: Mathematics for Information Technologies

Prague 2024

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor, Dr. rer. nat. Faruk Gölogluu, for the useful consultations and feedback he provided. I would also like to thank my family for their never-ending support.

Title: PIR codes using combinatorial structures

Author: Jana Hrúzová

Department: Department of Algebra

Supervisor: Dr. rer. nat. Faruk Gölogl u, Department of Algebra

Abstract: Private information retrieval (PIR) codes are crucial for ensuring user privacy when querying distributed data storage systems. These codes allow users to retrieve specific data items from multiple servers without revealing which item is being retrieved, thereby preserving the user's privacy. The main aim when studying PIR codes is to minimize storage overhead and communication complexity. This thesis provides an introduction to PIR codes and presents recent upper bounds on storage overhead. Furthermore, it discusses code families connected to PIR codes, such as PIR array codes, locally repairable codes, and batch codes.

Keywords: PIR protocol, PIR codes, block design, configurations

Contents

Introduction

Private information retrieval (PIR) protocols are designed to protect the privacy of users querying a database. The goal is to ensure that the database owner cannot determine which data item the user is interested in, even though the user successfully retrieves the data. PIR protocols are generally divided into single-server protocols and multi-server protocols.

In this thesis, we will work with multi-server PIR protocols within a distributed storage system, where the user interacts with multiple servers to retrieve information without disclosing the specific data of interest. This is represented by PIR codes and PIR array codes.

An efficient PIR system must minimize storage overhead while maintaining robust privacy guarantees and fast access times. This study addresses the lower limits of storage overhead and represents the currently achievable limits in this area.

In a k-server PIR $[m, s]$-code, the database is distributed among m servers. Each server stores $1/s$ of the database. We will show how the k-server PIR protocol is used with PIR codes. The advantage is that for PIR codes, the storage overhead is m/s, compared to the higher value k in classical PIR protocols. The main aim of studying PIR codes is to minimize the number of servers m given s and k, thereby reducing the storage overhead.

Furthermore, the concept of PIR array codes is explored. The goal in $[t \times m, p]$ k-PIR array codes is to design distributed storage systems with m servers that can implement classic k-PIR protocols while reducing storage overhead.

In this work, we will begin by introducing PIR protocols and their historical development. In the second and third chapters, we will define k-server PIR codes with a focus on minimizing m. This will be done mainly through combinatorial structures such as k-partial packings or (v_t, b_z)-configurations. We will also explore their geometric representation in affine space or projective space.

The last chapter will discuss code families connected to PIR codes. In particular, we will describe PIR array codes, some of their constructions, and associated problems with PIR codes. Furthermore, the chapter is dedicated to comparing PIR codes with locally repairable codes and batch codes, as these codes share common features.

1. Preliminaries

1.1 Notation

The following list will clarify the notation used in this thesis.

- In the whole thesis p is considered as prime number and q as power of p, i.e. $q = p^n$ for $n \in \mathbb{N}$;
- $\mathbb{F}_p$ denotes the finite field with p elements and $\mathbb{F}$ denotes an arbitrary finite field;
- $\mathbb{F}_q$ denotes the finite field with q elements with characteristic p;
- row vector is denoted by $\mathbf{x}$ and x_i denotes the i-th coordinate of a vector $\mathbf{x}$;
- $\mathbf{0}$ denotes row vector with all zeros;
- $\mathbf{e_i}$ denotes row vector with all zeros and one 1 on position i;
- I_k denotes an identity matrix of size k;
- Σ denotes a finite alphabet.

1.2 Linear codes

This section is inspired by [1, Appendix A.5].

Let $k, n \in \mathbb{N}, k \leq n$ and a mapping $\mathcal{C} : \Sigma^k \to \Sigma^n$ be an injection. Then, a set of coded vectors $\{\mathbf{y} = (y_1, y_2, \ldots, y_n) = \mathcal{C}(\mathbf{x}) : \mathbf{x} \in \Sigma^k\} \subseteq \Sigma^n$ is called an $[n, k]$-code.

An $[n, k]$-code $\mathcal{C}$ is a linear code if and only if $\Sigma = \mathbb{F}_q$ and:

1. $\mathcal{C}$ is closed under vector addition, i.e. the sum of any two valid codewords within the code is also a valid codeword. So if
$$\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C} : \mathbf{c}_1 + \mathbf{c}_2 \in \mathcal{C}.$$
2. $\mathcal{C}$ is closed under scalar multiplication, i.e. if $\mathbf{c} \in \mathcal{C}$ then
$$\forall a \in \mathbb{F}_q : a\mathbf{c} \in \mathcal{C}.$$

If a minimum distance d is specified, the code is denoted as a $[n, k, d]$-code.

The linear code can be defined by specifying only the basis vectors of the code's subspace, rather than enumerating every individual codeword. It can be interpreted as a $k \times n$ matrix $\mathbf{G}$ over $\mathbb{F}_q$ with linearly independent rows. This matrix $\mathbf{G}$ is called a generator matrix for the code $\mathcal{C}$ and

$$\mathcal{C} = \{\mathbf{xG} : \mathbf{x} \in \mathbb{F}_q^k\}.$$

The generator matrix $\mathbf{G}$ can be transformed into standard-form, $[I_k|A]$, where I_k represents the identity matrix of size k.

Additionally, an $[n, k, d]$-code can also be described using a parity-check matrix $\mathbf{H}$. This matrix $\mathbf{H}$ is an $(n-k) \times n$ matrix over $\mathbb{F}_q$ with rank $n-k$, and it defines the code $\mathcal{C}$ as the right null space of $\mathbf{H}$:

$$\mathcal{C} = \{\mathbf{y} \in \mathbb{F}_q^n : \mathbf{y}\mathbf{H}^T = \mathbf{0}\}.$$

The code's minimum distance is determined based on a specified metric. In this thesis, we will specifically consider the Hamming distance for this purpose.

Definition 1. *The* Hamming distance *of vectors* $\mathbf{u}, \mathbf{v} \in \mathbb{F}_q^n$ *is defined as the number of positions at which they have differing entries.*

The minimal distance of a code is bounded by Singleton bound.

Theorem 1 (Singleton bound)**.** *[1, Theorem A5.3] Let* $\mathcal{C}$ *be an* $[n, k, d]$*-code. Then*

$$d \leq n - k + 1.$$

Two more important definitions which will be used when talking about codes are as follows:

Definition 2. *The* communication complexity *is defined as the total number of bits exchanged between the server and the user during the execution of a protocol.*

Definition 3. *The* storage overhead *is defined as the ratio of the total number of bits stored across all servers to the number of bits in the original database.*

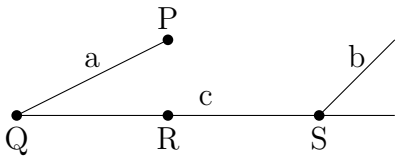
1.3 Incidence structures

An **incidence structure** consist of certain objects (usually called points, lines, planes, etc.) together with certain incidence relations between these objects. When considering only the set of points $\mathcal{P}$ and lines $\mathcal{L}$ (point-line incidence structure), there is an incidence relation $\mathcal{I} \subseteq \mathcal{P} \times \mathcal{L}$ [1, Chapter 1].

Example. Let us consider the point set $\mathcal{P} = \{P, Q, R, S\}$, the line set $\mathcal{L} = \{a, b, c\}$ and the incidence relation

$$\mathcal{I} = \{(P, a), (Q, a), (Q, c), (R, c), (S, b), (S, c)\}.$$

It can be informally presented by the picture:



or by incidence matrix, where the rows and columns are indexes by points and lines respectively. Entries 0 and 1 corresponds to non-incident and incident point-line pairs.

$$A = \begin{matrix} a & b & c \\ \end{matrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{matrix} P \\ Q \\ R \\ S \end{matrix}$$

Neither the picture nor the incidence matrix A is unique, both depends on the order of lines and points.

Definition 4. *A* $(\mathrm{v_t}, \mathrm{b_z})$-configuration *is an incidence structure* $(\mathcal{P}, \mathcal{L}, \mathcal{I})$ *where*

- $|\mathcal{P}| = v$, $|\mathcal{L}| = b$
- *Each line contains* z *points*
- *Each point lies on* t *lines*
- *Any two distinct points are connected by at most one line*

If $v = b$*, and consequently* $t = z$*, the configuration is symmetric and denoted by* v_z*.*

Example. Following example shows us (v_t, b_z)-configuration with $v = 6$, $t = 1$, $b = 2$, $z = 3$ and $v = b = 6$, $t = z = 2$ respectively.

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \quad A' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

The latter is an example of a symmetric configuration.

1.3.1 Block designs and partial packings

Here we provide the definitions of block designs and partial packings. Block designs are combinatorial structures that consist of a set of points and a collection of subsets called blocks, with specific incidence properties. It will be used later in the Chapter 3, where we will explore the relationships between various geometrical structures and partial packings.

Definition 5 (Block Design)**.** *[2, Definition 2.11] Let* v*,* k*,* t*, and* λ *be integers. Assume that* $v > k > t \geq 1$ *and* $\lambda \geq 1$*. A* t*-*(v, k, λ) *block design is an incidence structure consisting of points and blocks* $(V, \mathcal{B})$ *satisfying the following properties:*

1. *The size of* V *is* v*.*
2. *Each block* $B \in \mathcal{B}$ *contains exactly* k *points.*
3. *Every* t*-subset of* V *is contained in exactly* λ *blocks* $B \in \mathcal{B}$*.*

A 2-(v, k, λ) block design is known as a Balanced Incomplete Block Design (BIBD). From the definition, we can see that a (v, k, λ)-BIBD can also be considered as a (v_r, b_k)-configuration for some $r \in \mathbb{N}$.

Next definition is a special case of block designs, where $\lambda = 1$.

Definition 6. *A* Steiner system $S(t, k, n)$ *is a set system* $(S, \mathcal{B})$ *where:*

1. *S is a set of n elements.*
2. *$\mathcal{B}$ is a collection of k-element subsets of S called blocks.*
3. *Each t-subset of S is contained in exactly one block.*

Example. For example, an $S(2, 3, 7)$ system has 7 elements and the following blocks:

$$\{1,2,3\},\{1,4,5\},\{1,6,7\},\{2,4,6\},$$
$$\{2,5,7\},\{3,4,7\},\{3,5,6\}.$$

As we can see each subset of size two is exactly in one block.

Partial packings, on the other hand, are arrangements of points into subsets where each subset contains at least two points. Unlike block designs, there is no exact number of points in each subset. Pairs of points can appear together in at most one subset, similar to 2-Steiner systems.

Definition 7. *[3, Definition 1.1] A* k-partial packing *of a finite set X of size s is a set of $k-1$ partitions of X which satisfy the following conditions:*

(i) each subset in any partition has size at least two;

(ii) two subsets from two distinct partitions meet in at most one point.

The total number of subsets of X belonging to its partitions is the order r.

Moreover, a k-partial packing is homogeneous if all the subsets from any partition have the same size.

1.4 Affine geometry

Let $\mathbb{A}^n(\mathbb{F}_q)$ denote an affine space of dimension n over a finite field $\mathbb{F}_q$. The sections on affine and projective geometry are inspired by [1, Parts II and III].

Definition 8. *The* Gaussian binomial coefficient $\binom{n}{k}_q$ *for $n, k \in \mathbb{N}$ is defined by the number of k-dimensional subspaces of an n-dimensional vector space.*

Theorem 2. *[1, Chapter 19] For $n, k \in \mathbb{N}$*

$$\binom{n}{k}_q = \frac{(q^n-1)(q^n-q)\cdots(q^n-q^{k-1})}{(q^k-1)(q^k-q)\cdots(q^k-q^{k-1})}$$

Proof. To define a k-dimensional subspace, we must specify k linearly independent vectors. The initial vector can be selected from the non-zero vectors (set with $\mathbf{0}$ vector is linearly dependent) in $q^n - 1$ ways. The second vector must be chosen outside the span of the first vector. There are $q^n - q$ choices as the first vector generates a subspace of dimension 1. This process continues, resulting in $(q^n-1)(q^n-q)\cdots(q^n-q^{k-1})$ ways to specify k linearly independent vectors.

Now we need to divide the previously determined number by the count of k-sets generating the same subspace. Remarkably, this is equivalent to what we have already calculated in another manner: we are essentially determining the number of bases for a k-dimensional subspace. This count corresponds to the number of linearly independent k-sets in a k-dimensional space. Setting $n = k$ in the previous calculation gives us $(q^k - 1)(q^k - q) \cdots (q^k - q^{k-1})$. □

Corollary. The number of k-dimensional affine subspaces in $\mathbb{A}^n(\mathbb{F}_q)$ is equal to

$$q^{n-k} \binom{n}{k}_q .$$

Proof. The k-dimensional affine subspaces are described as the cosets in the form $x + U$, where $x \in \mathbb{F}_q^n$ and U is a k-dimensional subspace of $\mathbb{F}_q^n$.

However, $x + U = y + U$ if and only if $x - y \in U$. Consequently, as x goes through all q^n elements of $\mathbb{F}_q^n$, each k-dimensional coset corresponding to the vector subspace U is counted q^k times. □

1.4.1 Affine plane

An affine plane is an affine space of dimension two, denoted as $\mathbb{A}^2(\mathbb{F}_q)$. The following axioms hold in an affine plane:

- For every pair of distinct points, there exists precisely one line passing through both.
- For any line l and any point $P \notin l$, there exists precisely one line through P that does not intersect l.
- There are four points in such a way that no three are collinear.

Definition 9. *The property of a set of points lying on a single line is termed* collinearity.

Let us define parallel classes in affine plane.

Definition 10. *Let l and m be two lines in an affine plane. We define l to be parallel to m (expressed as $l \parallel m$) if either $l = m$ or the two lines do not have any points in common.*

In an affine plane, by definition, any two lines are either parallel or intersect at exactly one point.

Corollary. In an affine plane, parallelism creates an equivalence relation on the lines.

The order of an affine plane is determined by the number of points on any given line within the plane. For an affine plane over the finite field $\mathbb{A}^2(\mathbb{F}_q)$, every line contains exactly q points because each line can be described by a linear equation $ax + by = c$ where $a, b, c \in \mathbb{F}_q$, and for each value of variable x, there is a unique corresponding value of the variable y and vice versa.

1.5 Projective geometry

Let $\mathbb{P}^N(\mathbb{F}_q)$ denotes a projective space of dimension N over a finite field $\mathbb{F}_q$.

Lemma 3. *The number of k-dimensional projective subspaces in $\mathbb{P}^N(\mathbb{F}_q)$ is equal to*

$$\binom{N+1}{k+1}_q.$$

Proof. The number of k-dimensional projective subspaces over N-dimensional projective space over a finite field $\mathbb{F}_q$ is same as the number of $(k+1)$-dimensional subspaces of an $(N+1)$-dimensional vector space, thus $\binom{N+1}{k+1}_q$. □

In the following subsection, we will introduce several geometrical structures. These structures will be useful lately in Chapter 3, where they are put into relation with partial packings.

1.5.1 Projective plane

A projective plane is a projective space of dimension two, denoted as $\mathbb{P}^2(\mathbb{F}_q)$. For a projective plane, there are similar conditions as for the affine plane.

- Two distinct points define exactly one line passing through both of them.
- Two distinct lines intersect at precisely one point.
- There exist four points where no three of them are collinear.

From different point of view a projective plane, denoted as C, can be axiomatically defined as an incidence structure, involving a set $\mathcal{P}$ of points, a set $\mathcal{L}$ of lines, and an incidence relation I that specifies which points lie on which lines. This structure is written as $C = (\mathcal{P}, \mathcal{L}, I)$.

By interchanging the roles of "points" and "lines" in $C = (\mathcal{P}, \mathcal{L}, I)$, we obtain the *dual structure* $C^* = (\mathcal{L}, \mathcal{P}, I^*)$, where I^* represents the dual relation of I. The dual structure C^* is also a projective plane, known as the *dual plane* of C. If the projective plane C and its dual C^* are isomorphic, then C is called *self-dual*.

Definition 11. *[1, Chapter 10] A* collineation σ *of $(\mathcal{P}, \mathcal{L})$ is a permutation of $\mathcal{P} \cup \mathcal{L}$ that maps points to points and lines to lines, such that for $l \in \mathcal{L}, P \in \mathcal{P}$: $P \in l$ if and only if $P^\sigma \in l^\sigma$.*

A correlation σ *of $(\mathcal{P}, \mathcal{L})$ is a permutation of $\mathcal{P} \cup \mathcal{L}$ that transforms points into lines and lines into points, while preserving reversing incidence. Hence, a correlation σ has the property that for $l \in \mathcal{L}, P \in \mathcal{P} : l^\sigma \in P^\sigma$ if and only if $P \in l$.*

A polarity ρ *of $(\mathcal{P}, \mathcal{L})$ is a correlation of order 2, which means $\rho^2(P) = P$ holds for all points $P \in \mathcal{P}$ and $\rho^2(l) = l$ holds for all lines $l \in \mathcal{L}$.*

In a projective plane, let ρ be a polarity. A point P is absolute if $P \in \rho(P)$. Otherwise P is nonabsolute. Similarly, a line l is considered absolute if $\rho(l) \in l$. Otherwise it is nonabsolute.

1.5.2 Maximal arcs

In the next subsection there is summary on arcs and maximal arcs mainly inspired by [4].

Definition 12. *Let $m, d \in \mathbb{N}$ such that $m, d \geq 1$. An* (m, d)-arc *in the projective plane $\mathbb{P}^2(\mathbb{F}_q)$ is defined as a set A of m points such that each line in $\mathbb{P}^2(\mathbb{F}_q)$ contains at most d points from A. The number m is referred to as the size, and d is referred to as the degree. An $(m, 2)$-arc is abbreviated as an m-arc.*

We can see from the definition that $d \leq q + 1$ since there are $q + 1$ points on a line and a line contains $q + 1$ points in $\mathbb{P}^2(\mathbb{F}_q)$. Also, $d \leq m \leq q^2 + q + 1$ since there are $q^2 + q + 1$ lines in $\mathbb{P}^2(\mathbb{F}_q)$.

Let A be an (m, d)-arc, $x \in A$, and $L_1, \ldots, L_{q+1}$ be the lines through x. Define $n_i = |A \cap L_i|$ for $1 \leq i \leq q + 1$. Then the total number of points in A can be expressed as $m = 1 + \sum_{i=1}^{q+1}(n_i - 1)$. Given that $n_i \leq d$, it follows that $m \leq qd + d - q$.

Equality $m = qd + d - q$ holds if and only if $n_i = d$, i.e., every line of $\mathbb{P}^2(\mathbb{F}_q)$ either intersects A in exactly d points or is disjoint from A. An (m, d)-arc is called maximal when $m = qd + d - q$.

Lemma 4. *[5, Section 4] An (m, d)-maximal arc of $\mathbb{P}^2(\mathbb{F}_q)$ exist if there exist $0 \leq n' \leq n$ such that $q = 2^n, d = 2^{n'}, m = dq - q + d = 2^{n+n'} - 2^n + 2^{n'}$.*

1.5.3 Classical unitals

First we begin with what is a unital in projective geometry [1, p. 62], [2].

Definition 13. *A* unital *is a set of $n^3 + 1$ points organized into subsets of size $n + 1$, such that each pair of distinct points is included in exactly one subset.*

From definition the unital can be seen as 2-$(n^3 + 1, n + 1, 1)$ block design.

For next definition we need to define unitary polarity. A projective plane $\mathbb{P}^2(\mathbb{F}_{q^2})$ has an automorphism $x \mapsto \overline{x} = x^q$ of order 2. In such a scenario the conjugate-transpose map is $v \mapsto \overline{v}^T$. The transposition of matrix flips row and column vectors of length 3 and gives a polarity of $\mathbb{P}^2(\mathbb{F}_{q^2})$, establishing a unitary polarity in $\mathbb{P}^2(\mathbb{F}_{q^2})$.

The absolute points of this polarity are those satisfying the condition

$$0 = (x, y, z)(\overline{x}, \overline{y}, \overline{z})^T = x^{q+1} + y^{q+1} + z^{q+1},$$

which are exactly the points in classical unital U in $\mathbb{P}^2(\mathbb{F}_{q^2})$.

Definition 14. *The absolute points and non-absolute lines of unitary polarity create a* classical unital *U, also called a* Hermitian unital. *The set of point consists of $q^3 + 1$ points such that each line of $\mathbb{P}^2(\mathbb{F}_{q^2})$ meets U in either 1 or $q + 1$ points.*

The following definition is more technical but will be helpful for the next lemmas.

Definition 15. *Let U be a classical unital. A lines meeting U in one point is called* tangent.

Let point $P \notin U$. We denote by $\tau_P(U)$ a subset of points from U obtained from tangent lines through P.

Lemma 5. *[2, Theorem 2.3] Let U be a classical unital in a projective plane $\mathbb{P}^2(\mathbb{F}_{q^2})$. Then, for any point $P \in \mathbb{P}^2(\mathbb{F}_{q^2}) \setminus U$, the size of the subset $\tau_P(U)$ is $|\tau_P(U)| = q + 1$.*

Proof. Let U be a classical unital and $P \notin U$. Then, there are q^2+1 lines passing through P which partition the points of U into sets of size 1 or $q + 1$. Let s be the number of lines through P which meets U in $q + 1$ points, so the number of lines through P which meet U in one point is $q^2 + 1 - s$.

This leads to the equation

$$s(q + 1) + (q^2 + 1 - s) = q^3 + 1.$$

Solving for s results in $s = q^2 - q$. Consequently, there exist $q + 1$ lines through P meeting U in one point, and thus, the size of the subsets is $q + 1$. □

Lemma 6. *[2, Corollary 2.4] Let U be a classical unital in $\mathbb{P}^2(\mathbb{F}_{q^2})$ and $P \in \mathbb{P}^2(\mathbb{F}_{q^2}) \setminus U$. Then the points in $\tau_P(U)$ are collinear.*

Proof. Let U be a classical unital in $\mathbb{P}^2(\mathbb{F}_{q^2})$. Then we have a polarity ρ whose absolute points are U. Suppose P is a point not in U.

Then, for all points $Q \in \tau_P(U)$, we have $P \in Q^\rho$. From polarity definition we get $Q \in P^\rho$ for all Q which means all the points lie on the same line, so they are collinear. □

1.5.4 Conics in $\mathbb{P}^2(\mathbb{F}_q)$

In this subsection we will focus on conics in projective plane $\mathbb{P}^2(\mathbb{F}_q)$ over a finite field $\mathbb{F}_q$ with q odd. Definitions and some further observations about conics are from [1, Chapter 12].

Definition 16. *In a projective plane $\mathbb{P}^2(\mathbb{F}_q)$, a* conic C *is defined as the set of points with projective coordinates (X, Y, Z) that satisfy a non-zero homogeneous equation $Q(X, Y, Z) = 0$ given by:*

$$Q(X, Y, Z) = aX^2 + bY^2 + cZ^2 + dXY + eXZ + fYZ.$$

This condition is well-defined for points with projective coordinates (X, Y, Z) since if $\lambda \in \mathbb{F}_q, \lambda \neq 0$ then $Q(\lambda X, \lambda Y, \lambda Z) = \lambda^2 Q(X, Y, Z)$. A conic C is considered non-degenerate if it does not contain an entire line.

We will omit the proof of the following theorem as it would be too technical and our main focus is not on conics. The proof is nicely presented in [1, Theorem 12.1].

Theorem 7. *Consider a non-degenerate conic C in a finite projective plane $\mathbb{P}^2(\mathbb{F}_q)$. In this setting, C contains $q + 1$ points, and no three of these points are collinear. Furthermore, through a linear change of coordinates, C can be transformed to the conic $y^2 = xz$.*

A generalization of a conic in the projective plane is an oval.

Definition 17. *An* oval *in a projective plane of order n is an $(n+1)$-arc.*

The following definitions and lemmas in this subsection hold for both ovals and conics in projective planes of odd order q.

Any line in $\mathbb{P}^2(\mathbb{F}_q)$ intersects C in either 0, 1, or 2 points; these lines are called *external*, *tangent*, and *secant*, respectively.

Lemma 8. *[1, Theorem 12.7] Consider a non-degenerate conic C in a finite projective plane $\mathbb{P}^2(\mathbb{F}_q)$. For any point $P \in C$, there is precisely one tangent line passing through P, and the remaining q lines through P are secant lines, each intersecting C at just one more point other than P.*

The conic C is associated with:

- $q+1$ tangent lines as there is $q+1$ points in C and each have one tangent
- $\frac{q^2+q}{2}$ secant lines as there are q secants through each of $q+1$ points and each secant passes through two point of C
- $\frac{q^2-q}{2}$ external lines as we have totally q^2+q+1 lines, $q+1$ tangent lines and $\frac{q^2+q}{2}$ are secant lines

Definition 18. *Points lying on C are called* absolute. *Points not on C but lying on a tangent line to C are called* exterior points *of C. Points not lying on a tangent line to C are termed* interior points *of C.*

Lemma 9. *[1, Theorem 12.7] Let C be a non-degenerate conic in $\mathbb{P}^2(\mathbb{F}_q)$, where q is odd. With regard to C there is*

1. *$\frac{q^2+q}{2}$ exterior and $\frac{q^2-q}{2}$ interior points*
2. *every interior point lies on $\frac{q+1}{2}$ external and $\frac{q+1}{2}$ secant lines*
3. *every exterior point lies on 2 tangent, $\frac{q-1}{2}$ external and $\frac{q-1}{2}$ secant lines*

Proof. In whole proof be C a non-degenerate conic in $\mathbb{P}^2(\mathbb{F}_{q^2})$, q odd.

1. As we have $q+1$ tangent lines with q exterior point and each exterior point lies on two tangent lines, we have totally $\frac{q^2+q}{2}$ exterior points. The rest of the points not in C are interior and there are

$$q^2+q+1-\frac{q^2+q}{2}-(q+1)=\frac{q^2-q}{2}$$

   of these points.

2. Let P be an interior point of C. The lines passing from P to the points in C are secants. There is $q+1$ points in C and each secant passes through two points, which gives us that P lies on $\frac{q+1}{2}$ secant lines. Number of external lines is calculated as number of all lines going through P minus secant lines

$$q+1-\frac{q+1}{2}=\frac{q+1}{2}.$$

3. Let P be an exterior point of C. We know P lies on a tangent line l, meeting C in point T. As q is odd, P has to lie on even number of tangents. And since each of the n points on l different from T lies on at least one tangent other than l, the tangents other than l must meet l in n distinct points. Therefore, no three tangents are concurrent, and P lies on two tangents.

   The rest is calculated equally as for interior points, considering only $q-1$ points in C for secants.

□

In the dual plane, the tangent lines of C are absolute points of dual conic C', external lines of C change to interior points of dual conic C' and secant lines of C become exterior points of C'. That gives us following corollary.

Corollary. [1, Theorem 12.7] Let C be an irreducible conic in $\mathbb{P}^2(\mathbb{F}_{q^2})$ for q odd. Then the following holds.

1. A tangent line contains one absolute point and q external points
2. An external line contains $\frac{q+1}{2}$ interior and $\frac{q+1}{2}$ exterior points of C.
3. A secant line contains two absolute, $\frac{q-1}{2}$ interior and $\frac{q-1}{2}$ exterior points of C.

Proof. 1. From the definition of absolute and external points.

2. The total number of external lines is $\frac{q^2-q}{2}$, and there are $\frac{q^2+q}{2}$ exterior points, each lying on $\frac{q-1}{2}$ external lines. From duality, all external lines contain the same number of exterior and interior points, which is as follows:

   - for exterior points:

$$\frac{\left(\frac{q^2+q}{2}\right)\cdot\left(\frac{q-1}{2}\right)}{\frac{q^2-q}{2}} = \frac{q+1}{2};$$

   - for interior points:

$$q+1-\frac{q+1}{2} = \frac{q+1}{2}.$$

3. The same computation applies to secant lines, with the difference being the 2 absolute points that the secant line contains by definition.

□

2. PIR codes

2.1 Introduction to PIR protocols

Private information retrieval (PIR) protocol allows user to retrieve information from a database without revealing which specific data is being accessed. It ensures privacy for the user while interacting with the database. The privacy constraint can manifest as either *information-theoretic* or *computational*.

Information-theoretic requirement: In the information-theoretic requirement, each server should remain unable to deduce any information about the identity of the requested message, even when considering the scenario where the server possesses infinite computational power.

Computational privacy requirement: Conversely, the computational privacy requirement operates under the assumption that each server has restricted computational power. Under this constraint, it is essential that the server remains incapable of gaining any insight into the identity of the requested message.

There are two implementation of PIR protocol according to communication with server. It can be used either single-database PIR schema or multi-server PIR schema, where each server has a copy or some part of database.

2.1.1 Single-database PIR schema

We can imagine this approach as a scenario involving two participants: a user and a centralized database. The database contains publicly accessible data (for instance, an n-bit string). The user aims to retrieve a specific item from the database, i.e. the i-th bit, without revealing to the database which item is being queried. It's important to note that in this setup, the database data is public, but it is centrally located. The user, lacking a local copy, must submit a request to obtain the data from the central database.

A straightforward solution would be for the user to download the entire database, ensuring privacy. However, the overall communication complexity in this approach, quantified by the number of bits exchanged between the user and the database, is n.

Private information retrieval protocols offer a more efficient alternative, allowing the user to retrieve data from a public database with communication $< n$, surpassing the communication cost of downloading the entire database.

It has been demonstrated in [6] that for a single database, achieving perfect information-theoretic privacy necessitates downloading the entire database (of n messages). In this case, the optimal rate, defined as the ratio of the amount of desired information (one message) to the total downloaded amount (n messages), is $1/n$.

To overcome this inefficiency, two approaches can be considered: either limit the computational capabilities of the server or presume the existence of multiple independent servers, each possessing a copy or some part of the database.

2.1.2 Multi-server PIR protocol

The multi-server PIR protocol was introduced by [6] in 1995, in a setting where there are many copies of the same database, and none of these copies are allowed to communicate with each other.

Definition 19 (k-server PIR protocol)**.** *[7, Definition 1] Let $S_1, S_2, \ldots, S_k$ be servers, each storing a database x of length n, $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$. A user U wishes to retrieve the i-th bit x_i from the database without revealing i.*

The k-server PIR protocol is an algorithm $P = (Q, A, C)$ consisting of:

1. *The user U flips coins and, based on the coin flips and index i, invokes the query algorithm $Q(k, n; i)$ to generate a randomized k-tuple of queries of some predetermined fixed length, $(q_1, q_2, \ldots, q_k)$.*
2. *For $j \in 1, \ldots, k$, the user sends the query q_j to the j-th server S_j.*
3. *The j-th server, for $j \in 1, \ldots, k$, responds with an answer of some fixed length $a_j = A(k, j, x, q_j)$.*
4. *Finally, the user computes the output by applying the reconstruction algorithm $C(k, n; i, a_1, \ldots, a_k)$.*

In the classical PIR model, each server stores a full copy of the database. If there are k servers and the database size is n bits, the total storage across all servers is kn bits. The storage overhead is then defined as the ratio between the total number of bits stored in the system and the number of information bits, which is in this case k.

Another option, introduced in [7], is to distribute the database among multiple servers in a more efficient way, where each server stores only an encoded part of the database rather than the entire database. This method reduces the storage overhead compared to the classical model.

Parameter k is used for controlling the privacy-communication trade-off. In the context of multi-server PIR protocols, we will consider information-theoretic privacy. Later on, we will focus on multi-server PIR protocol with distributed storage system.

2.2 Definition of PIR codes

Private information retrieval codes are designed to reduce the communication and storage overhead compared to classical PIR model. Let us start with the definition [8].

Definition 20 (k-server PIR code)**.** *A k-server PIR code is a binary linear $[m, s]$-code with a generator matrix G such that for every integer i with $1 \leq i \leq s$, there exist k disjoint subsets of columns of matrix G that add up to the vector of weight 1, with the single 1 in position i. The disjoint subsets of columns of matrix G are called recovery sets.*

Consider a database containing n bits of data. Each of the m servers stores n/s bits, so the total number of bits stored on all servers is $\frac{mn}{s}$. The storage overhead in a k-server PIR $[m, s]$-code is given as the ratio of all bits stored on all servers to the bits in the database. That is,

$$\frac{\frac{mn}{s}}{n} = \frac{m}{s}.$$

The central aim when studying PIR codes is to minimize the value of m while keeping s and k fixed. This minimal value of m is denoted by $P(s, k)$.

Example. Example of a k-server PIR $[9, 4]$-code.

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

For G as the generator matrix, we can obtain these recovery sets for different columns.

For $i = 1$ we get for example

$$\begin{cases} x_1 = y_1 \\ x_1 = y_2 + y_3 + y_6 \\ x_1 = y_4 + y_7 \\ x_1 = y_5 + y_8 + y_9 \end{cases}$$

In the same way for $i = 2, 3$, and $i = 4$, we can obtain

$$\begin{cases} x_2 = y_2 \\ x_2 = y_5 + y_6 \\ x_2 = y_1 + y_3 + y_4 + y_9 \\ x_2 = y_7 + y_8 \end{cases} \quad \begin{cases} x_3 = y_3 \\ x_3 = y_1 + y_2 + y_6 \\ x_3 = y_4 + y_5 + y_7 \\ x_3 = y_8 + y_9 \end{cases} \quad \begin{cases} x_4 = y_4 \\ x_4 = y_1 + y_2 + y_8 \\ x_4 = y_3 + y_5 + y_7 \\ x_4 = y_6 + y_9 \end{cases}$$

In this example we have 4-server PIR $[9, 4]$-code as we have 4 recovery sets for all x_i.

Let $S_1, \ldots, S_m$ be servers, each storing a linear combination of $x_1, \ldots, x_s$, and the user wants to obtain x_i. A k-server PIR $[m, s]$-code can be used to simulate a linear k-server PIR protocol in the following way:

1. The user generates queries $(q_1, \ldots, q_k)$.
2. Let $R_1, \ldots, R_k$ be k recovery sets for x_i. For all $j \in \{1, \ldots, k\}$, the user sends query q_j to the servers in recovery set R_j.
3. Each server responds with an answer, and since we have a linear PIR protocol, all answers from one recovery set R_j sum up to the answer $a_j = A(k, j, x_i, q_j)$.
4. The user computes the output by applying the k-server PIR protocol reconstruction algorithm $C(k, n; i, a_1, \ldots, a_k)$.

2.3 Known bounds for PIR codes

Let $P(s,k)$ be a the minimum value of m for which a k-server PIR $[m,s]$-code exists.

Theorem 10. *For every $s,k \in \mathbb{N}, s,k \geq 1$ the value $P(s,k) \geq k \cdot \frac{2^s-1}{2^{s-1}}$ and equality holds if and only if $k \equiv 0$ mod 2^{s-1}, i.e. k is divisible by 2^{s-1}.*

Proof of the previous theorem can be found in [7, Theorem 18]. That provides the optimal value of $P(s,k)$ for k divisible by 2^{s-1}.

Corollary. Let $s \geq 1$ be an integer, then $P(s,2^{s-1}) = 2^s - 1$.

Here is a summary of inequalities for $P(s,k)$ as derived from [7, Lemma 13, Lemma 14]. These inequalities provide straightforward methods to establish upper bounds for various values of $P(s,k)$.

Theorem 11. *The following holds for $s,k \geq 1$:*

1. *$P(s,1) = s$ and $P(s,2) = s+1$;*
2. *$P(s,k) \leq P(s,k+1) - 1$;*
3. *If k is odd, then $P(s,k) = P(s,k+1) - 1$;*
4. *$P(s,k) \leq P(s+1,k) - 1$.*

Proof. 1. For $k = 1$, consider the $[s,s]$-code which stores all the information symbols, i.e., the generator matrix is I_s. This is a 1-server PIR $[s,s]$-code.

Similarly, consider the parity check code $[s+1,s]$. In the generator matrix G, we can find exactly 2 disjoint subsets that add up to e_i for every $1 \leq i \leq s$ (because s columns are independent). This means we have a 2-server PIR $[s+1,s]$-code.

2. Consider a $(k+1)$-server PIR $[m,s]$-code with generator matrix G. As we know from the definition of PIR codes, for every $1 \leq i \leq s$, there are $k+1$ disjoint subsets of columns that add up to e_i. If we delete one column from G, we will affect at most one of these sets. This means that G' is the generator matrix for a k-server PIR $[m-1,s]$-code.

3. For k odd, we need to prove that $P(s,k+1) \leq P(s,k)+1$. Let us consider a k-server PIR $[m,s]$-code with generator matrix G. By the definition of a k-server PIR $[m,s]$-code, we have k disjoint subsets of columns of G that add up to e_i for $1 \leq i \leq s$.

Now we have two possibilities. If the sum of all columns of G adds up to $\mathbf{0}$, then the sum of the remaining columns (columns that are not in any of the k subsets) is equal to e_i, because k is odd. Thus, we have a $(k+1)$-server $[m,s]$ PIR code.

The second possibility is that the sum of all columns of G is not equal to $\mathbf{0}$. In this case, we can add one column to G to make the sum equal to $\mathbf{0}$. Then we obtain a $(k+1)$-server PIR $[m+1,s]$-code.

4. Proof can be seen in [7] at the end of section V. □

Theorem 12. *[7, Lemma 13] For $s,s_1,s_2 \geq 1$ and $k,k_1,k_2 \geq 1$, the following holds:*

1. *$P(s_1+s_2,k) \leq P(s_1,k) + P(s_2,k)$;*

2. $P(s, k_1 + k_2) \leq P(s, k_1) + P(s, k_2)$.

Proof. 1. Imagine we have a k-server $[m_1, s_1]$-code and a k-server PIR $[m_2, s_2]$-code with generator matrices G_1 and G_2, respectively. Let us create a new generator matrix of shape

$$G = \begin{pmatrix} G_1 & 0_{s_1 \times m_2} \\ 0_{s_2 \times m_1} & G_2 \end{pmatrix}.$$

There are k disjoint subsets of columns that add up to e_i for every $1 \leq i \leq s_1 + s_2$. We have obtained a k-server PIR $[m_1 + m_2, s_1 + s_2]$-code.

2. As previously, assume we have a k_1-server $[m_1, s]$-code and a k_2-server PIR $[m_2, s]$-code with generator matrices G_1 and G_2, respectively. By concatenating G_1 and G_2, we obtain the generator matrix $G = [G_1 \mid G_2]$. Thus, there exist $k_1 + k_2$ disjoint subsets of columns that add up to e_i for every $1 \leq i \leq s$. This results in a $(k_1 + k_2)$-server PIR $[m_1 + m_2, s]$-code. □

Next upper bounds and some optimal values were determined in [7].

Theorem 13. *For $s, k \geq 1$ the following holds:*

1. *For every integer $k > 1$ we have $P(2, k) = \lceil 3k/2 \rceil$;*
2. *For every even integer $k > 2$ we have $P(3, k) = \lceil 7k/4 \rceil$;*
3. *[7, IV. A] $P(s, k) \leq s + (k-1)\lceil s^{\frac{1}{k-1}} \rceil^{k-2}$.*

In [7], the following two constraints were obtained from the construction of PIR codes.

Theorem 14. *[7, Theorem 8] If $r \in \mathbb{N}$ and a Steiner system $S(2, k-1, r)$ exists, and $s = \frac{r(r-1)}{(k-1)(k-2)}$, then $P(s, k) \leq r + s$.*

A sufficient condition for the existence of $S(2, k-1, r)$ is $\frac{r(r-1)}{(k-1)(k-2)} \in \mathbb{Z}$ and $\frac{r-1}{k-2} \in \mathbb{Z}$.

Theorem 15. *[7, Theorem 9] For l, θ and $\lambda \in \mathbb{N}$ the following holds:*

$$P(2^{2\theta l} - (2^{\theta+1} - 1)^l, 2^l + 2) \leq 2^{2\theta l} - 1;$$

$$P((2^\lambda - 1)^l - 1, 2^l) \leq 2^{\lambda l} - 1.$$

In the publication [3], more precise bounds were established with the aid of combinatorial structures. These bounds, along with a more in-depth exploration of these combinatorial structures, are undertaken in Chapter 3. The results are summarized at the end of the chapter in Table 3.2.

2.4 Results of practical tests

The lower bounds on $P(s, k)$ from this and the next chapter are used to compute $P(s, k)$ for small values of s and k. The results are presented in Table 2.1, which was computed using a SageMath program, attached as Appendix A.1.

The first column of Table 2.1 shows the minimal value of m, denoted as $P(s, k)$, while the second column represents the storage overhead for the given values of s and k.

s \ k	1		2		3		4		5		6		7		8		9		10		11		12	
1	1	1.00	2	2.00	3	3.00	4	4.00	5	5.00	6	6.00	7	7.00	8	8.00	9	9.00	10	10.00	11	11.00	12	12.00
2	2	1.00	3	1.50	5	2.50	6	3.00	8	4.00	9	4.50	11	5.50	12	6.00	14	7.00	15	7.50	17	8.50	18	9.00
3	3	1.00	4	1.33	6	2.00	7	2.33	9	3.00	10	3.33	13	4.33	14	4.67	16	5.33	17	5.67	19	6.33	20	6.67
4	4	1.00	5	1.25	8	2.00	9	2.25	11	2.75	12	3.00	14	3.50	15	3.75	19	4.75	20	5.00	23	5.75	24	6.00
5	5	1.00	6	1.20	9	1.80	10	2.00	12	2.40	13	2.60	18	3.60	19	3.80	22	4.40	23	4.60	25	5.00	26	5.20
6	6	1.00	7	1.17	10	1.67	11	1.83	13	2.17	14	2.33	20	3.33	21	3.50	24	4.00	25	4.17	27	4.50	28	4.67
7	7	1.00	8	1.14	12	1.71	13	1.86	14	2.00	15	2.14	22	3.14	23	3.29	27	3.86	28	4.00	29	4.14	30	4.29
8	8	1.00	9	1.12	13	1.62	14	1.75	19	2.38	20	2.50	24	3.00	25	3.12	33	4.12	34	4.25	38	4.75	39	4.88
9	9	1.00	10	1.11	14	1.56	15	1.67	21	2.33	22	2.44	29	3.22	30	3.33	36	4.00	37	4.11	43	4.78	44	4.89
10	10	1.00	11	1.10	15	1.50	16	1.60	23	2.30	24	2.40	31	3.10	32	3.20	39	3.90	40	4.00	47	4.70	48	4.80
11	11	1.00	12	1.09	17	1.55	18	1.64	24	2.18	25	2.27	32	2.91	33	3.00	42	3.82	43	3.91	49	4.45	50	4.55
12	12	1.00	13	1.08	18	1.50	19	1.58	25	2.08	26	2.17	33	2.75	34	2.83	44	3.67	45	3.75	51	4.25	52	4.33
13	13	1.00	14	1.08	19	1.46	20	1.54	26	2.00	27	2.08	34	2.62	35	2.69	46	3.54	47	3.62	53	4.08	54	4.15
14	14	1.00	15	1.07	20	1.43	21	1.50	28	2.00	29	2.07	35	2.50	36	2.57	49	3.50	50	3.57	56	4.00	57	4.07
15	15	1.00	16	1.07	21	1.40	22	1.47	30	2.00	31	2.07	42	2.80	43	2.87	52	3.47	53	3.53	61	4.07	62	4.13
16	16	1.00	17	1.06	23	1.44	24	1.50	32	2.00	33	2.06	46	2.88	47	2.94	56	3.50	57	3.56	62	3.88	63	3.94
17	17	1.00	18	1.06	24	1.41	25	1.47	33	1.94	34	2.00	47	2.76	48	2.82	58	3.41	59	3.47	63	3.71	64	3.76
18	18	1.00	19	1.06	25	1.39	26	1.44	34	1.89	35	1.94	48	2.67	49	2.72	60	3.33	61	3.39	64	3.56	65	3.61
19	19	1.00	20	1.05	26	1.37	27	1.42	35	1.84	36	1.89	49	2.58	50	2.63	62	3.26	63	3.32	65	3.42	66	3.47
20	20	1.00	21	1.05	27	1.35	28	1.40	36	1.80	37	1.85	50	2.50	51	2.55	64	3.20	65	3.25	66	3.30	67	3.35
21	21	1.00	22	1.05	28	1.33	29	1.38	41	1.95	42	2.00	51	2.43	52	2.48	65	3.10	66	3.14	67	3.19	68	3.24
22	22	1.00	23	1.05	30	1.36	31	1.41	42	1.91	43	1.95	52	2.36	53	2.41	66	3.00	67	3.05	68	3.09	69	3.14
23	23	1.00	24	1.04	31	1.35	32	1.39	43	1.87	44	1.91	53	2.30	54	2.35	67	2.91	68	2.96	69	3.00	70	3.04
24	24	1.00	25	1.04	32	1.33	33	1.38	44	1.83	45	1.88	54	2.25	55	2.29	76	3.17	77	3.21	79	3.29	80	3.33
25	25	1.00	26	1.04	33	1.32	34	1.36	45	1.80	46	1.84	55	2.20	56	2.24	77	3.08	78	3.12	80	3.20	81	3.24
26	26	1.00	27	1.04	34	1.31	35	1.35	49	1.88	50	1.92	57	2.19	58	2.23	78	3.00	79	3.04	81	3.12	82	3.15
27	27	1.00	28	1.04	35	1.30	36	1.33	50	1.85	51	1.89	58	2.15	59	2.19	79	2.96	80	3.00	82	3.04	83	3.07
28	28	1.00	29	1.04	37	1.29	38	1.32	52	1.86	53	1.89	59	2.11	60	2.14	81	2.89	82	2.93	83	2.96	84	3.00
29	29	1.00	30	1.03	38	1.34	39	1.38	53	1.83	54	1.86	60	2.07	61	2.10	82	2.83	83	2.86	84	2.90	85	2.93
30	30	1.00	31	1.03	39	1.37	40	1.40	54	1.80	55	1.83	61	2.03	62	2.07	83	2.77	84	2.80	85	2.83	86	2.87
31	31	1.00	32	1.03	41	1.35	42	1.39	58	1.87	59	1.90	62	2.00	63	2.03	84	2.71	85	2.74	86	2.77	87	2.81
32	32	1.00	33	1.03	42	1.38	43	1.41	59	1.84	60	1.88	64	2.16	65	2.19	85	2.66	86	2.69	87	2.72	88	2.75

Table 2.1: Upper bounds for $P(s, k)$ and corresponding storage overhead for given s and k

3. Minimizing storage overhead in PIR codes

In this chapter we will show some combinatorial structures and their connection to PIR codes. In [3] some new bounds for private information retrieval codes by using combinatorial structure of these codes were introduced, especially k-partial packings and (v_t, b_z)-configurations. With these combinatorial structures, we can establish upper bounds on the number of servers m, which consequently provides new upper bounds on storage overhead.

3.1 PIR codes as k-partial packings

Consider a linear $[m, s = m-r]$ k-server PIR code. As a linear code, its generator matrix can be written as $[I_s \mid P]$. Thus, we have s information bits $e_1, \ldots, e_s$ and r redundancy bits $p_1, \ldots, p_r$, where each p_i is characterized by a subset $S_i \subseteq \{e_1, \ldots, e_s\}$ such that $p_i = \sum_{e \in S_i} e$.

As mentioned in [7], the idea behind the construction of any k-server PIR code is to create k mutually disjoint subsets of $\{g_1, \ldots, g_m\}$ (for every information bit), such that the information bit can be recovered by a linear combination of the bits in each set.

Theorem 16. *[3] A k-partial packing of order r for a set of size s gives us a sufficient conditions for a k-server PIR code with parameters $[r+s, s]$. The storage overhead in this construction is $\frac{m}{s} = 1 + \frac{r}{s}$.*

Proof. Let $X = e_1, \ldots, e_s$, with S_i as previously defined, and assume there exists a k-partial packing of X. Take an arbitrary $1 \leq i \leq s$. We will denote the sets S_j such that $e_i \in S_j$ as $S_j^{(i)}$, for $j \in 1, \ldots, r$. By definition, there are $k-1$ partitions of X into subsets $S_j^{(i)}$. We next define, for $j \leq k-1$, the sets $R_j^{(i)} = e_l : e_l \in Sj^{(i)}, l \neq i \cup p_j$ and $R_k^{(i)} = e_i$.

From the k-partial packing definition, every subset $S_j^{(i)}$ has size at least two. Moreover, according to the second condition, all k sets $R_j^{(i)}$ are mutually disjoint because two $S_j^{(i)}$ subsets from two distinct partitions meet in at most one point, and all of the subsets $S^{(i)}j$ have e_i in common. Finally, it is straightforward to verify that e_i is the sum of the bits in every $Rj^{(i)}$ set, thereby creating a k-server PIR $[r+s, s]$-code. □

Theorem 17. *[3, Theorem 2.2] Let $s = a_1 \cdots a_{k-1}$, where $a_i \in \mathbb{N}, a_i \geq 2$. Then for each $w \leq k-1$ there exist a $(w+1)-$server PIR $[m, s]-$code with*

$$m = s + \frac{s}{a_1} + \cdots + \frac{s}{a_w}.$$

The storage overhead is $1 + \sum_{i=1}^{w} \frac{1}{a_i}$.

Specifically if $s = h^{k-1}$, for each $w \leq k-1$ there exist a $(w+1)-$server PIR $[s + w\frac{s}{h}, s]-$code with storage overhead $1 + \frac{w}{h}$.

Proof. Let $k > 3$ and s can be written as the product of $k-1$ integers as follows:

$$s = a_1 \cdot a_2 \cdots a_{k-1}, \quad \text{with} \quad a_i \geq 2.$$

Let C_{a_i} denote the cyclic group of order a_i and G the direct product of the groups C_{a_i} for $i = 1, \ldots, k-1$. Then,

$$G = C_{a_1} \times C_{a_2} \times \cdots \times C_{a_{k-1}}.$$

Each subgroup C_{a_i} creates a partition P_i on G into subsets defined as follows: numbers in all positions in one subset of a partition P_i are identical except for the i-th position. Thus, each subset in the partition P_i contains exactly a_i elements.

For each $w \leq k-1$, $P = \{P_1, \ldots, P_w\}$ satisfies:

(i) $|P_i| = \frac{s}{a_i}$, which means each subset in any partition has size at least two.

(ii) For any two distinct indices i, j, let S_i and S_j be arbitrary subgroups in partitions P_i and P_j, respectively. Elements in S_i differ only in the i-th position, and elements in S_j differ only in the j-th position. Since $i \neq j$, the subsets S_i and S_j have at most one element in common.

That gives us a $(w+1)$-partial packing of G. The order is equal to $\frac{s}{a_1} + \cdots + \frac{s}{a_w}$.

Let $s = h^{k-1}$, then

$$m = s + \sum_{i=1}^{w} \frac{s}{h} = s + w\frac{s}{h}.$$

That gives us $(w+1)$-server PIR $[s + w\frac{s}{h}, s]$-code, with storage overhead equal to $\frac{m}{s} = 1 + w\frac{1}{h}$.

□

Example. Let $s = 18$ and $k = 3$, then we have $w = 2$ and s can be written in terms of previous theorem as $s = 2 \cdot 9$ or $s = 3 \cdot 6$.

So for $a_1 = 2, a_2 = 9$, we obtain $m = 29$ and storage overhead 1.61. Better result is received, if we consider $a_1 = 3$ and $a_2 = 6$. Then $m = 27$ and storage overhead is equal to 1.5.

3.2 Connection to affine geometry

Consider approaching PIR codes from the perspective of affine geometry. Let $\mathbb{A}^N(\mathbb{F}_q)$ be an affine space of dimension N over a finite field $\mathbb{F}_q$. We can divide the affine space into parallel classes, which will give us a connection to partial packings.

Theorem 18. *[3, Theorem 2.5] Let $N \in \mathbb{N}$, $N \geq 2$ and $s = q^N$. Then for any $k - 1 \leq q^{N-1} + q^{N-2} + \cdots + q + 1$ there exist a k-server PIR $[m, s]$-code with*

$$m = s + \frac{s(k-1)}{q} = s + (k-1)q^{N-1}$$

and storage overhead $1 + \frac{k-1}{q}$.

Proof. In an affine space $\mathbb{A}^N(\mathbb{F}_q)$, we have q^N points. We can divide the affine space into parallel classes of lines. In any parallel class, there are q^{N-1} lines, because we have q^N points and each line contains q points, so we have to divide it by q points which generate the same line.

The total number of lines in $\mathbb{A}^N(\mathbb{F}_q)$, according to Corollary 1.4, is $\frac{q^{N-1}(q^N-1)}{q-1}$. As parallelism on lines is an equivalence relation, we can obtain the number of parallel classes as the total number of lines divided by the number of lines in one class, which is

$$\frac{q^{N-1}(q^N-1)}{q-1} \cdot \frac{1}{q^{N-1}} = q^{N-1} + q^{N-2} + \cdots + q + 1.$$

The k-partial packing of $\mathbb{A}^N(\mathbb{F}_q)$ is obtained by taking any $k-1$ parallel classes. The order of such a k-partial packing is $r = \frac{(k-1)q^N}{q}$, which is the total number of subsets of $\mathbb{A}^N(\mathbb{F}_q)$ belonging to its partition.

According to Theorem 16, this proves the existence of a k-server PIR $[m, s]$-code with $m = s + r$ and storage overhead $1 + \frac{r}{s}$. □

A similar approach can be applied when considering parallelism of hyperplanes, which are $(n-1)$-dimensional subspaces of affine space $\mathbb{A}^N(\mathbb{F}_q)$. Two hyperplanes are parallel if and only if they do not have any points in common.

Theorem 19. *[3, Theorem 2.6] Let $N \in \mathbb{N}$, $N \geq 2$ and $s = hq^{N-1}$ for some $h \leq q$. Then for any $k - 1 \leq q^{N-1}$ there exist a k-server PIR $[m, s]$-code with*

$$m = s + (k-1)q^{N-1}$$

and storage overhead $1 + \frac{k-1}{h}$.

Proof. Let us consider a subset S of $\mathbb{A}^N(\mathbb{F}_q)$ consisting of h parallel hyperplanes. Each hyperplane contains q^{N-1} points, so there are q parallel classes in $\mathbb{A}^N(\mathbb{F}_q)$. Therefore, the subset S has size hq^{N-1}.

There are q^{N-1} directions not determined by these hyperplanes. One can imagine this as if from one fixed point in a hyperplane, there are q^{N-1} lines to points on a second hyperplane, each determining a direction. Each of these directions corresponds to lines that are not parallel to the hyperplanes. Each line in one of these directions will intersect every hyperplane exactly once, giving a total of h intersection points per line.

We can select $k-1$ directions from the q^{N-1} available, each giving us q^{N-1} directed lines (one parallel to another). This selection provides us with a k-partial packing of order $r = (k-1)q^{N-1}$, with $m = s + r = hq^{N-1} + (k-1)q^{N-1}$ and storage overhead equal to $\frac{m}{s} = 1 + \frac{k-1}{h}$. □

3.3 Connection to projective geometry

Similarly to an affine space, we can divide a projective space $\mathbb{P}^N(\mathbb{F}_q)$ using the following notation.

Definition 21. *[9, p. 207] A* d-spread *of* $\mathbb{P}^N(\mathbb{F}_q)$ *is a collection of* d*-dimensional subspaces that are mutually disjoint and whose union covers all of* $\mathbb{P}^N(\mathbb{F}_q)$*. A* d-packing *of* $\mathbb{P}^N(\mathbb{F}_q)$ *is a partition of all* d*-dimensional subspaces into* d*-spreads.*

For $d = 1$*, we will denote the* d*-spreads as* resolution classes *and the* d*-packing as a* resolution*.*

A sufficient condition for such a partition of the lines in $\mathbb{P}^N(\mathbb{F}_q)$ to exist is either $N = 2z + 1$ with $q = 2$ for $z \geq 1$, or $N = 2^{i+1} - 1$ for $i \geq 1$ [3].

Theorem 20. *[3, Theorem 2.4] Let* N *be such that a resolution in* $\mathbb{P}^N(\mathbb{F}_q)$ *exist, and* $s = q^N + q^{N-1} + \cdots + q + 1$*. Then for* $k - 1 \leq q^{N-1} + \cdots + q + 1$*, there exists a* k*-server PIR* $[m, s]$*-code with*

$$m = s + \frac{(k-1)s}{q+1}$$

and storage overhead $1 + \frac{k-1}{q+1}$*.*

Proof. In each resolution class, there are $\frac{q^N+\cdots+q+1}{q+1}$ lines, because the total number of points in $\mathbb{P}^N(\mathbb{F}_q)$ is $s = \frac{q^{N+1}-1}{q-1}$ and each line contains $q + 1$ points.

As we showed in Lemma 3, the number of lines in $\mathbb{P}^N(\mathbb{F}_q)$ is

$$\frac{(q^{N+1}-1)(q^N-1)}{(q^2-1)(q-1)}.$$

That gives us

$$\frac{(q^{N+1}-1)(q^N-1)}{(q^2-1)(q-1)} \cdot \frac{q+1}{q^N+\cdots+q+1} = \frac{q^N-1}{q-1}$$

resolution classes.

Additionally, two resolution classes from distinct partitions intersect in at most one point, as two different lines meet at most in one point. Taking $k - 1$ resolution classes creates a homogeneous k-partial packing of order

$$r = (k-1)\frac{q^N+\cdots+q+1}{q+1}.$$

As stated in Theorem 16, this establishes the existence of a k-server PIR $[m, s]$-code with $m = s + r$ and a storage overhead of $1 + \frac{r}{s}$. □

Corollary. For N odd and $q = 2$ we will get a k-server PIR $[m, s]$-code with

$$s = \frac{q^{N+1}-1}{q-1} = 2^{N+1} - 1,$$

$m = s + \frac{(k-1)s}{3}$, $k - 1 \leq \frac{q^N-1}{q-1} = 2^N$ and storage overhead equals to $\frac{m}{s} = 1 + \frac{k-1}{3}$.

3.4 Geometrical objects in projective planes

In this subsection, we will focus on the structures in the projective plane $\mathbb{P}^2(\mathbb{F}_q)$ that were defined in Section 1.5.

3.4.1 Maximal arcs

An (m, d)-arc in a projective plane $\mathbb{P}^2(\mathbb{F}_q)$ is a set of m points such that no d points are collinear. In a maximal arc A, every line in $\mathbb{P}^2(\mathbb{F}_q)$ is either disjoint from A or intersects A in exactly d points.

An (m, d)-maximal arc in $\mathbb{P}^2(\mathbb{F}_q)$ exists if $n, n' \in \mathbb{N}, 0 \leq n' \leq n$ such that $q = 2^n, d = 2^{n'}$ and $m = 2^{n+n'} - 2^n + 2^{n'}$. That will help us with the following theorem.

Theorem 21. *[3, Corollary 2.8] Let $s = 2^{n+n'} - 2^n + 2^{n'}$ for some n, n' such that $1 \leq n' \leq n$. Then for any k, where $k - 1 \leq 2^n + 1$ there exist a k-server PIR $[m, s]$-code with*

$$m = s + \frac{(k-1)s}{2^{n'}}$$

and storage overhead $1 + \frac{k-1}{2^{n'}}$.

Proof. Consider $q = 2^n$ and $d = 2^{n'}$. Then a $(qd - q + d, d)$-maximal arc in $\mathbb{P}^2(\mathbb{F}_q)$ exists, and we will denote it as A.

From the definition of maximal arcs, every line intersects A either in d points or not at all. Take a point x which is not in A. The lines through x that are not disjoint from A intersect A in d points and create a partition into subsets of size $d = 2^{n'}$.

Moreover, there are $s = 2^{n+n'} - 2^n + 2^{n'}$ points in A, divided into subsets of size $d = 2^{n'}$. This means the number of subsets is

$$\frac{s}{d} = \frac{2^{n+n'} - 2^n + 2^{n'}}{2^{n'}}.$$

Now take $k-1$ points from a line disjoint from A (i.e., there are $q+1$ collinear points, so $k - 1 \leq q + 1$). Each of these points will create a partition of A into subsets of size d. Any two subsets in any partition have at most one point in common because the subsets are defined by lines, and for every pair of distinct points, there exists exactly one line passing through both.

Thus, this configuration will create a homogeneous k-partial packing of order $r = s + \frac{(k-1)s}{d}$, providing us with the corresponding PIR code. □

3.4.2 Classical unitals

As a summary of Section 1.5.3, for classical unital U, each point in $\mathbb{P}^2(\mathbb{F}_{q^2}) \setminus U$ defines a partition of U into subsets of $q + 1$ collinear points. There are $\frac{q^3+1}{q+q} = q^2 - q + 1$ of these subsets.

Theorem 22. *[3, Corollary 2.10] Let $s = q^3 + 1$, where q is a prime power. Then for each $k - 1 \leq q^2$ there exist a k-server PIR $[m, s]$-code with*

$$m = s + (k - 1)(q^2 - q + 1)$$

and storage overhead $1 + \frac{k-1}{q+1}$.

Proof. Let us consider a classical unital U over $\mathbb{P}^2(\mathbb{F}_{q^2})$. According to the definition of a unital, each line meets U in either 1 or $q+1$ points. Take a line l that meets U in one point P.

The remaining q^2 points P_i (where $1 \leq i \leq q^2$) on the line l distinct from P will create partitions of U, each into $q^2 - q + 1$ subsets of $q+1$ collinear points.

These partitions are mutually disjoint because they are unequivocally defined by the points on the line l and the polarity, which maps $P_i \mapsto P_i^\rho$, where $P_i \in l$ and $P_i \neq P$. Since all the points P_i lie on the line l, the point $l^\rho \in P_i^\rho$. This means that through the point l^ρ go all the lines P_i^ρ. Also, $l^\rho \notin U$, because if l^ρ were in U, it would coincide with P_i for all $1 \leq i \leq q^2$, which contradicts the choice of P_i as distinct points on l different from P.

Taking $k-1$ of these partitions will create a homogeneous k-partial packing of order $r = (k-1)(q^2 - q + 1)$, which is the number of partitions times the number of subsets in each partition. □

3.4.3 Conics in $\mathbb{P}^2(\mathbb{F}_q)$

A non-degenerate conic C in a projective plane $\mathbb{P}^2(\mathbb{F}_q)$ contains $q+1$ points. These points are called absolute. Points lying on tangent line are exterior points of C. Points neither on C nor lying on tangent line are denoted as interior. There are $\frac{q^2-q}{2}$ interior points.

Every exterior point lies on 2 tangent, $\frac{q-1}{2}$ secant and $\frac{q-1}{2}$ external lines.

An external line contains $\frac{q+1}{2}$ interior and $\frac{q+1}{2}$ exterior points of C. A secant line contains 2 absolute, $\frac{q-1}{2}$ interior and $\frac{q-1}{2}$ exterior points of C.

Theorem 23. *[3, Corollary 2.11] Let $s = \frac{q^2-q}{2}$, where $q > 3$ and q is odd. Then for each $k - 1 \leq q$ there exist a k-server PIR $[m, s]$-code with*

$$m = s + (k-1)(q-1)$$

and storage overhead $1 + \frac{2(k-1)}{q}$.

Proof. Consider a non-degenerate conic C in the projective plane $\mathbb{P}^2(\mathbb{F}_q)$ for q odd. Let P be an exterior point of C.

Secant and external lines through P divide the set of interior points of C into $\frac{q-1}{2} + \frac{q-1}{2} = q - 1$ subsets. Each subset is created by interior points on secant or external lines, so the points in each subset are collinear. The subsets have sizes $\frac{q-1}{2}$ for secant lines and $\frac{q+1}{2}$ for external lines.

Let l be a tangent line to C. The set of interior points has size $s = \frac{q^2-q}{2}$. By selecting $k-1$ distinct exterior points on l out of q, we form $k-1$ partitions of the set of interior points. Each subset has size either $\frac{q-1}{2}$ or $\frac{q+1}{2}$. By definition, in projective planes, two distinct points define exactly one line passing through both of them, so the subsets have at most one point in common.

This results in a non-homogeneous k-partial packing with order

$$r = (k-1)(q-1).$$

The corresponding PIR code has $m = s + (k-1)(q-1)$ and storage overhead $\frac{m}{s} = 1 + \frac{2(k-1)}{q}$.

□

3.5 $(v_t, b_z)-$configurations

As mentioned in Chapter 1, a (v_t, b_z)-configuration is a point-line structure with v points, b lines, and specific restrictions: each line contains z points, each point lies on t lines, and any two distinct points are connected by at most one line. The following theorem will connect these configurations with k-server PIR codes.

Theorem 24. *[3] A (v_t, b_z)-configurations produces a $(t+1)$-server PIR $[m, s]$-code with $s = v$, $m = s + b$ and storage overhead $1 + \frac{b}{s}$.*

Proof. We can imagine the (v_t, b_z)-configuration as a $v \times b$ matrix M with t ones in each row, z ones in each column, and for any two rows j and k, if $M_{j,i} = 1$ and $M_{k,i} = 1$ for some i, then for all l where $M_{j,l} = 1$, it holds that $M_{k,l} = 0$ (and vice versa - if $M_{k,l} = 1$, then $M_{j,l} = 0$).

Consider a $v \times (v + b)$ matrix $G = (I_v \mid M)$. We can then proceed similarly to k-partial packings. We can express redundancy vectors from matrix M as $p_j = \sum_{e \in S_j} e$, where $S_j \subseteq \{e_1, \ldots, e_v\}$.

Take an arbitrary $1 \leq i \leq v$. Denote $S_j^{(i)}$ the set S_j if and only if $e_i \in S_j$. The sets $R_j^{(i)} = \{e_l : e_l \in S_j^{(i)}, l \neq i\} \cup \{p_j\}$ for $1 \leq j \leq t$ and $R_{t+1} = \{e_i\}$ define a partition of column vectors. These subsets are disjoint because any two distinct points are connected by at most one line, which corresponds to the i-th line in the representation of matrix G for sets $S_j^{(i)}$.

The subsets R_i give us $t + 1$ disjoint subsets of columns that add up to the vector with a single 1 in position i. This creates a $(t+1)$-server PIR $[v+b, v]$-code with storage overhead $\frac{m}{v} = 1 + \frac{b}{v}$.

□

For the next theorem, we need to define what is meant by a resolution in the context of (v_t, b_z)-configurations.

Definition 22. *[10, Definition 2.1] Let $t, z \geq 2$ and $v \geq 4$ be integers. Suppose $C = (P, L)$ is a (v_t, b_z)-configuration with a set of points P and a set of lines L. A* parallel class *in C is a subset of lines such that each point in P is incident with exactly one line in the subset. A partition of L into parallel classes is called a* resolution*, and C is said to be a* resolvable configuration *if L has at least one resolution.*

Other interesting results for upper bounds for PIR codes can be obtained using resolvable (v_t, b_z)-configurations.

Theorem 25. *[3, Theorem 3.1] Consider a resolvable configuration (v_t, b_z), with $s = v$ and $k - 1 \leq t$. Then there exists a k-server PIR $[m, s]$-code with*

$$m = s + (k-1)\frac{s}{z},$$

and storage overhead

$$1 + \frac{k-1}{z}.$$

Proof. A resolvable configuration (v_t, b_z) consists of v points and b lines, where each line contains z points, and each point lies on t lines. There are t parallel classes in (v_t, b_z) because each point lies on t lines, and a parallel class is a partition of points according to lines.

Two subsets, each from a different parallel class, have at most one point in common since, in (v_t, b_z), any two distinct points are connected by at most one line.

The size of each parallel class is the number of all points divided by the number of points lying on the same line, which is $\frac{v}{z}$.

Taking $k-1$ parallel classes creates a homogeneous k-partial packing with order $r = (k-1)\frac{v}{z}$. According to Theorem 16, this gives us a corresponding PIR $[m, s]$-code for $s = v$ with storage overhead

$$\frac{m}{s} = 1 + \frac{k-1}{z}.$$

□

A slightly different results are for resolvable (v_t, b_z)-configurations, where any two distinct points are connected exactly by one line. Such a configuration is a Steiner system $S(2, z, v)$ or $(v, k, 1)$-BIBD.

As a reminder, a (v, k, λ)-BIBD can be considered as a (v_r, b_k)-configuration for some $r \in \mathbb{N}$. So we we can talk about resolvable BIBDs.

Theorem 26. *[3, Theorem 3.2] Let (v_t, b_z) be a resolvable BIBD, where any two distinct points are connected by exactly one line. Then for any $k-1 \leq \frac{v-1}{z-1}$, $s = v$ there exist a k-server PIR $[m, s]$-code with*

$$m = s + (k-1)\frac{s}{z},$$

and storage overhead $1 + \frac{k-1}{z}$.

Proof. The number of parallel classes in such resolution is number of lines going through one point which is t. Consider fixed point P in configuration, then there is $v-1$ points which we want to connect with P and on one line there is another $z-1$ points. That gives us $t = \frac{v-1}{z-1}$ lines going through one point. There are $\frac{v}{z}$ points in each parallel class.

□

In the Table 3.1 there is a list of existing resolvable (v_t, b_z) configurations and corresponding parameters for PIR codes.

z	v	**Conditions**
3	$v \equiv 3 \pmod 6$	
4	$v \equiv 4 \pmod{12}$	
5	$v \equiv 5 \pmod{20}$	$v \neq 45, 345, 465, 645$
7	$v \equiv 7 \pmod{42}$	$v > 294427$
8	$v \equiv 8 \pmod{56}$	$v > 24480$

Table 3.1: List of existing resolvable BIDS (v_t, b_z)-configurations [3]

Following list are sufficient conditions for existence of (v_t, b_z)-configurations for $z = 3, 4$ and 5 [3, Section 4.2].

- $z = 3$: $v_t = 3b$ and $v \geq 2t + 1$.
- $z = 4$:
  - $v \equiv 4 \pmod{12}$, $v > 3t + 1$, and $v_t = 4b$;
  - $v \equiv 0 \pmod{12}$, $v \geq 3t + 1$, $v_t = 4b$, and

    $$v \notin \{84, 120, 132, 180, 216, 264, 312, 324, 372, 456, 552, 648, 660, 804, 852, 888\}.$$

  - $v \equiv 0 \pmod{12}$, $v = 3t + 3$, and $v_t = 4b$;
  - $t = 4s$, $v \geq 3t + 1$, $v_t = 4b$, and $1 \leq s \leq 15$, except possibly $s = 3$ and $v = 38$;
  - $t = 6$, $v \geq 20$ even, and $b = \frac{3v}{2}$.
- $z = 5$:
  - $v = 4t + 4$, $v \equiv 0 \pmod{20}$, and $v_t = 5b$;
  - $v \equiv 5 \pmod{20}$, $v \geq 4t + 1$, $v_t = 5b$, and $v \geq 7865$;
  - $t = 5s$, $v \geq 4t + 1$, $v_t = 5b$, $1 \leq s \leq 10$ and

    $$(t, v) \notin \{(1, 22), (2, 42), (2, 43), (3, 62), (3, 63), (4, 82), (5, 102), (7, 142), (9, 182), (9, 183), (9, 185), (9, 186), (9, 187), (9, 188), (9, 189), (9, 190), (9, 191), (9, 192)\}.$$

The following Table 3.2 summarizes the results from this chapter, along with references to the theorems where they are proven.

s	$k-1$	$P(s,k) \leq$	Theorem
$a_1 \cdots a_c$	$\leq c$	$s + \sum_{i=1}^{k-1} \frac{s}{a_i}$	17
q^N	$\leq \frac{q^N-1}{q-1}$	$s(1+\frac{k-1}{q})$	18
$hq^{N-1}, h \leq q$	$\leq q^{N-1}$	$s(1+\frac{k-1}{h})$	19
$\frac{q^{N+1}-1}{q-1}$, $N = 2^{i+1}-1$	$\leq \frac{q^N-1}{q-1}$	$s(1+\frac{k-1}{q+1})$	20
$2^{N+1}-1$, N odd	$\leq 2^N - 1$	$s(1+\frac{k-1}{3})$	20
$2^{n+n'} - 2^n + 2^{n'}$, $0 \leq n' \leq n$	$\leq 2^n + 1$	$s(1+\frac{k-1}{2^{n'}})$	21
q^3+1	$\leq q^2$	$s(1+\frac{k-1}{q+1})$	22
$\frac{q^2-q}{2}$	$\leq q$	$s + (k-1)(q-1)$	23
$\equiv 3 \pmod 6$	$\leq \frac{s-1}{2}$	$s(1+\frac{k-1}{3})$	26
$\equiv 4 \pmod{12}$	$\leq \frac{s-1}{3}$	$s(1+\frac{k-1}{4})$	26
$\equiv 5 \pmod{20}$, $\neq 45, 345, 465, 645$	$\leq \frac{s-1}{4}$	$s(1+\frac{k-1}{5})$	26
$\equiv 7 \pmod{42}$, > 294427	$\leq \frac{s-1}{6}$	$s(1+\frac{k-1}{7})$	26
$\equiv 8 \pmod{56}$, > 24480	$\leq \frac{s-1}{7}$	$s(1+\frac{k-1}{8})$	26

Table 3.2: Known upper bounds on P(s,k) [3]

4. Code families connected to PIR codes

In this chapter we will introduce *PIR array codes*, *locally repairable codes* and *batch codes* and their connection to PIR codes.

4.1 PIR array codes

We can look at k-server PIR $[m, s]$-codes in a different way, where we have n bits in the database, but we divide it into s parts, each part containing n/s bits as a single symbol. Then, m is the total number of parts stored, and s is the number of parts in the database.

PIR array codes were introduced in [7] and extend this concept by allowing each server to store more than one symbol. This means the database can be divided into more parts, for example, $3s$ parts of $\frac{n}{3s}$ bits each, enabling each server to store three symbols. This idea can be generalized so that each server stores a fixed number of symbols. A symbol can be interpreted as a vector in $\mathbb{F}_2$.

This approach can improve the value of $P(s, k)$ for certain instances of s and k, thereby reducing the storage overhead. Since each server stores multiple symbols, we treat the code construction as an array code. When a server receives a query q, it responds with multiple answers corresponding to the number of symbols it stores.

4.1.1 Definition of PIR array codes

This section is mainly inspired by [11] and [12]. Let us start with formal definition.

Definition 23. *Let $x_1, x_2, \ldots, x_p$ be a basis of a vector space with dimension p over finite field $\mathbb{F}$. A* $[t \times m, p]$ array code *is defined as a $t \times m$ array where each entry is a linear combination of the basis elements x_i.*

This array code is $[t \times m, p]$ k-PIR array code *if it satisfies the k-PIR property. That means for every $i \in \{1, 2, \ldots, p\}$, there exist k pairwise disjoint subsets $S_1, S_2, \ldots, S_k$ of columns such that for all $j \in \{1, 2, \ldots, k\}$, the element x_i is a linear combination of the entries of the columns S_j.*

The setting is as follows: the database is divided into p parts, $x_1, x_2, \ldots, x_p$, each encoded as an element of the finite field $\mathbb{F}$. Each of the m servers stores t linear combinations of these parts. The entries in the array consisting of a single basis element are called *singletons*.

Furthermore, if x_i can be accessed from a single server alone, it appears as a singleton entry and is not included in any other entries on that server. Consequently, we may assume that each server contains t linearly independent symbols.

Example. In the following example is $[5 \times 4, 8]$ 3-PIR array code consisting of 17 singletons.

$$\begin{bmatrix} \text{Server 1} & \text{Server 2} & \text{Server 3} & \text{Server 4} \\ x_1 & x_2 & x_3 & x_1+x_2+x_3 \\ x_3 & x_1 & x_2 & x_4 \\ x_4 & x_5 & x_4+x_5+x_6 & x_6 \\ x_5 & x_6 & x_7 & x_8 \\ x_7 & x_5+x_7+x_8 & x_8 & x_5 \end{bmatrix}$$

We can see that each x_i can be obtained from linear span of entries of 3 disjoint columns as follows:

- x_1 from $\{S_1\}, \{S_2\}$ and $\{S_3, S_4\}$ as $S_{1,3} + S_{2,3} + S_{1,4}$;
- x_2 from $\{S_2\}, \{S_3\}$ and $\{S_1, S_4\}$;
- x_3 from $\{S_1\}, \{S_3\}$ and $\{S_2, S_4\}$;
- x_4 from $\{S_1\}, \{S_4\}$ and $\{S_2, S_3\}$;
- x_5 from $\{S_1\}, \{S_2\}$ and $\{S_4\}$;
- x_6 from $\{S_2\}, \{S_4\}$ and $\{S_1, S_3\}$;
- x_7 from $\{S_1\}, \{S_3\}$ and $\{S_2, S_4\}$;
- x_8 from $\{S_3\}, \{S_4\}$ and $\{S_1, S_2\}$.

The storage overhead is given by $\frac{tm}{p}$, which can be significantly smaller than k when a good array code is used. Define $s = \frac{p}{t}$, so $\frac{1}{s}$ represents the fraction of the database stored on each server. This also means that for PIR array codes s could be a fraction. The storage overhead can be viewed as

$$\frac{tm}{p} = \frac{m}{s},$$

which is the number of servers times the fraction of the database stored on each server.

We aim to maximize the ratio

$$\frac{k}{\frac{tm}{p}} = s\frac{k}{m}$$

for several reasons:

- when s is fixed, indicating the amount storage required per server, such schemes offer lower storage overhead compared to k;
- we aim to use the minimal number of servers, so m should be as small as possible.

On the other hand, with k being large, the storage overhead could also be too large for practical usage in applications (though still much smaller than k as required).

As we can see, a PIR array code is characterized by the parameters s, t, k, and m. When $t = 1$, we get k-server PIR codes, for which the goal is to find the smallest m for a given s and k. This value of m is denoted by the function $P(s, k)$.

For PIR array codes, the extra parameter t is introduced. Given s, t, and k, the goal is to find the smallest m, denoted by $P(s, t, k)$.

Definition 24 (Virtual server rate)**.** *We define the virtual server rate of a $[t \times m, p]$ k-PIR array code as $\frac{k}{m}$. The largest virtual server rate of such a code will be denoted as $g(s, t)$ for s and t fixed. Let $g(s) = \limsup_{t\to\infty} g(s, t)$.*

In case we have s and t fixed, then p is consequently fixed as $p = st$.

The following theorem provides an upper bound on the virtual server rate.

Theorem 27. *[11, Theorem 3] For each rational number $s > 1$ we have*

$$g(s) \leq \frac{s+1}{2s}$$

and there is no t such that $g(s, t) = \frac{s+1}{2s}$.

Proof. Let us have a $[t \times m, p]$ k-PIR array code with $s = \frac{p}{t}$, so without loss of generality each server contains t linearly independent vectors. We will show that $\frac{k}{m} < \frac{s+1}{2s}$.

Define α_i as the number of servers holding the singleton x_i in one of their cells. Given that each server out of m contains t cells, we have $\sum_{i=1}^{p} \alpha_i \leq tm$, and thus the average value of the α_i is at most $\frac{tm}{p} = \frac{m}{s}$. Therefore, there exists $u \in \{1, 2, \ldots, p\}$ such that $\alpha_u \leq \frac{m}{s}$. Equality $\alpha_u = \frac{m}{s}$ only holds when $\alpha_i = \frac{m}{s}$ for all $i \in \{1, 2, \ldots, p\}$.

From the definition of a k-PIR array code, we will get k disjoint sets of servers, chosen such that the span of the cells in each subset of servers includes x_u. These servers will be denoted as $S_1, S_2, \ldots, S_k \subseteq \{1, 2, \ldots, m\}$. If no server in a subset S_j contains the singleton x_u, then the subset S_j must contain at least two elements. Hence, at most α_u of the subsets S_j have a cardinality of one, as there are α_u servers holding the singleton x_u. The rest of the $m - \alpha_u$ servers have a cardinality of at least two. This implies that

$$k \leq \alpha_u + \frac{m - \alpha_u}{2}.$$

When divided by m servers, we get the following inequality:

$$\begin{aligned}\frac{k}{m} &\leq \frac{\alpha_u + (m - \alpha_u)/2}{m} = \frac{2\alpha_u + m - \alpha_u}{2m} = \frac{1}{2} + \frac{\alpha_u}{2m} \\ &\leq \frac{1}{2} + \frac{m/s}{2m} = \frac{1}{2} + \frac{1}{2s} = \frac{s+1}{2s}.\end{aligned}$$

Equality holds only when $\alpha_i = \frac{m}{s}$ for all $i \in \{1, 2, \ldots, p\}$, indicating that all cells in every server are singletons. Consequently, the span of a subset of servers includes x_i if and only if it includes a server with a cell x_i, thus $k \leq \alpha_i = \frac{m}{s}$. This means the rate of the array code is at most $\frac{\alpha_i}{m} = \frac{m/s}{m} = \frac{1}{s}$. This contradicts the premise that the rate of the array code is $\frac{k}{m} = \frac{s+1}{2s}$ for $s > 1$. Therefore, we have proven that $\frac{k}{m} < \frac{s+1}{2s}$. □

4.1.2 Construction of PIR array codes

In [11], Blackburn and Etzion developed an optimal construction of PIR array codes, especially for $1 < s \leq 2$. This construction of PIR array codes provides lower bounds for the virtual server rate $g(s,t)$.

Construction 1 Let $t > 1$ and $1 \leq d \leq t$. We define $p = t + d$, which gives us $s = 1 + \frac{d}{t}$. Let u denote the least common multiple $LCM(d,t)$.

In the construction, there are two types of servers:

- Server A: Stores t singletons. Each possible t-subset of parts occurs $\frac{u}{d}$ times as the set of singleton cells of a server.
- Server B: Stores $t-1$ singletons. The last cell contains the sum of the remaining $p-(t-1) = d+1$ parts. All possible $(t-1)$-sets of singletons occur $\frac{u}{t}$ times.

There are $\binom{p}{t}\frac{u}{d}$ servers of Type A and $\binom{p}{t-1}\frac{u}{t}$ servers of Type B.

Example. Let $d = 1$ and $t = 2$. Then $s = \frac{3}{2}$, $p = 3$, and $u = 2$. So we can divide the data in the database into three parts x_1, x_2, x_3.

Server A: We have the following options for servers with t singletons: (x_1, x_2), (x_2, x_3), (x_1, x_3).

Server B can store the following entries: $(x_1, x_2{+}x_3)$, $(x_2, x_1{+}x_3)$, $(x_3, x_1{+}x_2)$.

Each entry of Server A appears twice, and each entry of Server B appears once in the resultant matrix.

$$\begin{bmatrix} \text{Server 1} & \text{Server 2} & \text{Server 3} & \text{Server 4} & \text{Server 5} & \text{Server 6} & \text{Server 7} & \text{Server 8} & \text{Server 9} \\ x_1 & x_1 & x_2 & x_2 & x_1 & x_1 & x_1 & x_2 & x_3 \\ x_2 & x_2 & x_3 & x_3 & x_3 & x_3 & x_2+x_3 & x_1+x_3 & x_1+x_2 \end{bmatrix}$$

As we can see, the virtual server rate in this example is $g(s,t) = \frac{k}{m} = \frac{7}{9}$. The storage overhead is $\frac{tm}{p} = 18$.

The following theorem is obtained from Construction 1 and proof can be found in [11, Theorem 7].

Theorem 28. *For any given t and d, $1 \leq d \leq t$, when $s = 1 + \frac{d}{t}$ we have:*

$$g(s,t) \geq 1 - \frac{d^2+d}{(t+d)(2d+1)} = \frac{s+1+1/d}{s(2+1/d)}.$$

For any rational number $1 < s \leq 2$, we have

$$g(s) \geq \limsup_{t->\infty} g(s,t) = \frac{s+1}{2s}.$$

For $s = 2$, we have $d = 2t - t = t$ and

$$g(2,t) \geq \frac{3+1/t}{4+2/t} = \frac{3t+1}{4t+2}.$$

The lower bound for $g(s)$ matches the upper bound established in Theorem 27, providing an asymptotically optimal virtual server rate.

To achieve the optimal virtual server rate, the Blackburn-Etzion construction requires too large number of servers $m = \binom{p}{t}\frac{u}{d} + \binom{p}{t-1}\frac{u}{t}$ for practical use. In [12], an alternative construction was proposed that uses significantly fewer servers, with only a little reduction in the virtual server rate.

Construction 2 Let $s, t \in \mathbb{N}$, $t+2 \leq s \leq 2t+1$ and $p = st$. Code C is constructed from three types of servers:

- Server T_1: Stores t singletons. Each t-subset appears α_1 times and there are $\binom{p}{t}$ number of distinct subsets.
- Server T_{t+1}: Stores pair (A, B) where A is a $(t-1)$ singletons and B is a sum of $(t+1)$ other entries $\{x_i : i \in \{1, \ldots, p\}\} \setminus A$. Each such a pair appears α_2 times.
- Server 1-factorization: Set $\{x_1, \ldots, x_p\}$ is divided into subsets of size s, where each element occurs exactly in one subset. All the s-subsets of $\{x_1, \ldots, x_p\}$ can be separated into $\binom{p-1}{s-1}$ partitions $\mathcal{P}$. This server stores t sums $\left\{\sum_{x_i \in P} x_i : P \in \mathcal{P}\right\}$. Server from each partition appears α_s times.

Let α_1, α_2, $\alpha_s \in \mathbb{N}$ minimum such that $\alpha_1\binom{p-1}{t} = \alpha_2\binom{p-1}{t-1}\binom{p-t}{t}$ and $\alpha_2\binom{p-1}{t-1}\binom{p-t}{t+1} = \alpha_s\binom{p-1}{s-1}$.

Theorem 29. *[12, Theorem 4] The code C in Construction 2 is a $[t \times m, p = st]$ k-PIR array code, where*

$$k = \alpha_1\binom{p}{t} + \alpha_2\binom{p-1}{t-2}\binom{p-t+1}{t+1} + \alpha_s\binom{p-1}{s-1},$$

$$m = \alpha_1\binom{p}{t} + \alpha_2\binom{p-1}{t-1}\binom{p-t+1}{t+1} + \alpha_s\binom{p-1}{s-1}$$

and the virtual server rate $g(s,t)$ is

$$\frac{k}{m} = \frac{((s-1)^2+s)t+1}{(2(s-1)^2+s)t+s}.$$

4.1.3 Comparison with PIR codes

The main aim when studying PIR codes is to determine the minimal value of servers m, denoted as $P(s,k)$. The lower m is for fixed s and k, the lower the storage overhead.

PIR array codes offer a structured approach that allows for more efficient data distribution. We have m servers, each storing t entries, together containing $1/s$ of the database.The minimal value m, previously denoted as $P(s,t,k)$, also depends on t.

- For $t = 1$ we will get $P(s,t,k) = P(s,k)$.

- For $t > 1$, we will get $P(s,t,k) \leq P(s,k)$, as we can always take a $[m,s]$ k-PIR code and divide each entry on each server into t smaller parts and store them back into t cells. This creates a $[t \times m, ts]$ k-PIR array code.

For PIR array codes, the main aim is to achieve the highest value of the virtual server rate $g(s,t)$, which is k/m. This also results in m being as small as possible compared to k for given s and t. Since $g(s,t)$ is the maximal value of k/m, we have $g(s,t) \geq k/m$. This implies that

$$P(s,t,k) \geq \frac{k}{g(s,t)}.$$

The related aim of studying PIR array codes is to find the range where $P(s,t,k) < P(s,k)$ and the storage overhead is low.

4.2 Locally Repairable Codes

Locally repairable (or recoverable) codes are a class of error-correcting codes and are used to transfer data over channels with error. Error-correcting codes has a specific capability to detect and correct a certain number of errors. However, if the number of errors exceeds the correction capability of the code, the transmitted message cannot be accurately recovered.

Locally repairable codes were designed for distributed storage systems. In such systems, data is stored in multiple servers, and these servers may experience occasional failures.

Definition 25. *[13, Definition 2] The code $\mathcal{C}$ has* locality $r \geq 1$ *and* availability $\delta \geq 1$ *if, for any $\mathbf{y} \in \mathcal{C}$, any symbol in $\mathbf{y}$ can be recovered by using any of δ disjoint sets of symbols, each of size at most r.*

If we write the generator matrix of the code as $[I_k \mid A]$ for some matrix A, the symbols in $\mathbf{y}$ corresponding to I_k are referred to as informational symbols. There are two distinct concepts related to locality: the locality of informational symbols and the locality of all symbols.

- For the locality of an informational symbol, only the recoverability of informational symbols is required (from δ sets of size at most r);
- When all symbols of $\mathbf{y}$ are recoverable from a small sets it is called locality of all symbols.

According to [13] the parameters of a linear $[n,k,d]$ code with locality r and availability δ of all symbols satisfy:

$$n \geq k + d + \left\lceil \frac{\delta(k-1)+1}{\delta(r-1)+1} \right\rceil - 2.$$

This bound can be regarded as a refinement of the classical Singleton bound, incorporating an additive penalty of $\left\lceil \frac{\delta(k-1)+1}{\delta(r-1)+1} \right\rceil - 1$ for the code's locality and availability when compared to the classical Singleton bound.

Definition 26. *A code $\mathcal{C}$ is called* locally repairable code (LRS) *if it satisfies the locality (and availability) property.*

In locally repairable codes, the goal is to ensure that every message bit can be recovered from a small set of coded bits, with only one such recovery set being required. In contrast to PIR codes, where we wish to have many disjoint recovery sets for every message bit, without concern for the size of these sets.

4.3 Batch Codes

Batch codes are a specialized type of error-correcting codes used in distributed storage systems and data storage applications. Unlike traditional error-correcting codes, which are designed to recover from individual symbol errors, batch codes are designed to correct entire batches or subsets of symbols that may be lost or corrupted due to various factors, such as server failures or network errors.

The definition of batch codes is as follows.

Definition 27 (Batch Code)**.** *[13, Definition 3] Let Σ be a finite alphabet. Then C denotes a $(k, n, t, M, \tau)_\Sigma$ batch code over a finite alphabet Σ if it encodes any string $\mathbf{x} = (x_1, x_2, \ldots, x_k) \in \Sigma^k$ into M strings. These M strings are called buckets and have a total length of n. The buckets are denoted as $\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_M}$, such that for each t-tuple, called batch, of (not necessarily distinct) indices $i_1, i_2, \ldots, i_t$, the symbols $x_{i_1}, x_{i_2,\ldots,x_{i_t}}$ can be retrieved by reading at most τ symbols from each bucket.*

Definition 28. *A* primitive batch code *is a batch code, where each bucket contains exactly one symbol. In particular, $n = M$.*

For $\tau = 1$, this model is referred to as multiset batch codes. Essentially, this means that only one symbol is read from each bucket.

4.3.1 Linear Batch Codes

We consider a special case of primitive multiset batch codes with $n = M$ and $\tau = 1$. Under these conditions, each symbol can be viewed as a separate bucket, and only one reading per bucket is allowed [14].

Let $\Sigma = \mathbb{F}_q$, where q is a prime power and encoding mapping $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be linear over $\mathbb{F}$. Then the code is linear $[n, k]$ code over $\mathbb{F}$. The batch code with parameters n, k and t, where t is the size of a query of the code, is denoted by $[n, k, t]$-batch code.

Linear batch codes shares some similarities with locally repairable codes. Both are designed to provide fault tolerance and data recovery capabilities in scenarios where data is distributed across multiple storage devices or servers. In terms of locally repairable codes a batch code has availability M and locality τ.

Main difference between batch codes and locally repairable codes is that in batch codes we are interested in reconstruction of the information symbols in $\mathbf{x}$. On the contrary, in locally repairable codes we are trying to recover the coded symbols in $\mathbf{y}$.

Example. Since batch codes can be viewed as linear codes, we can express the coding as $\mathbf{y} = \mathbf{xG}$. Consider this 3×6 generator matrix of a batch code $\mathcal{C}$:

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

Take, for instance, the following combination of the columns of $\mathbf{G}$:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Now considering that the query includes the information symbols (x_1, x_1, x_2), we can recover these symbols using the following equations:

$$\begin{cases} x_1 = y_1 \\ x_1 = y_2 + y_5 \\ x_2 = y_3 + y_4 \end{cases}$$

Similarly, we can show that any 3-tuple $(x_{i_1}, x_{i_2}, x_{i_3})$, where $i_1, i_2, i_3 \in \{1, 2, 3\}$, can be reconstructed using the symbols from $\mathbf{y}$, with the condition that each symbol is used at most once. The conclusion is that $\mathcal{C}$ is a $[6, 3, 3]_2$ batch code.

It can be seen that batch codes are a specific subset of private information retrieval codes. The main distinction is in the type of queries. PIR codes supports queries in the form of $(x_i, x_i, \ldots, x_i)$, where $i \in \{1, \ldots, k\}$. In contrast, batch codes supports more general queries like $(x_{i_1}, x_{i_2}, \ldots, x_{i_t})$, where $i_1, i_2, \ldots, i_t$ can be different indices selected from the set $\{1, \ldots, k\}$. Consequently, batch codes could be used as PIR codes. However, PIR codes cannot be considered as batch codes because there is no guarantee of the existence of reconstruction sets for mixed queries.

Example. Let us go back to the previous example of the $[6, 3, 3]_2$ batch code. We can recover, for example, the symbols (x_1, x_1, x_1) using the following equations:

$$\begin{cases} x_1 = y_1 \\ x_1 = y_2 + y_5 \\ x_1 = y_3 + y_6 \end{cases}$$

It can be also viewed as 3-server PIR $[6, 3]$-code.

On the other hand, in the example of the k-server PIR $[9, 4]$-code from Chapter 2, where the generator matrix G was as follows

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

there is no recovery set for the query (x_1, x_1, x_4, x_4).

Theorem 30. *[14, Lemma 2] Let $\mathcal{C}$ be an $[n, k, t]_q$ batch code. The minimum Hamming distance of $\mathcal{C}$ is greater than t.*

Proof. Consider an arbitrary row i, for $i \in \{1, \ldots, k\}$. We need to show that the Hamming weight of row i in the generator matrix G is at least t.

We can retrieve the combination $(x_i, x_i, \ldots, x_i)$ if there are t non-intersecting sets of columns, such that the sum of the elements in each set equals e_i.

Since we can retrieve t copies of x_i from t non-intersecting sets of columns, there must be at least t columns in G with a nonzero entry in the i-th position. This is because each of these sets of columns must contribute a nonzero entry in the i-th row to sum to e_i. Meaning the Hamming weight of row i is at least t. □

Conclusion

The aim of this thesis was to introduce private information retrieval (PIR) protocols, explore PIR codes, and establish their connections with combinatorial structures to derive upper bounds on the number of servers for given values of s and k.

We began by introducing linear codes, designs, and various structures in projective planes. In Section 2.1, we discussed PIR protocols, their applications, and their classification into single-server and multi-server PIR protocols. Traditional k-PIR protocols have a storage overhead of k, as each of the k servers stores the entire database.

We then defined k-server PIR $[m, s]$-codes, designed to simulate k-PIR protocols with m servers, where $m \geq k$ and each server stores $1/s$ of the database. We demonstrated how these codes can be used with k-PIR protocols to achieve a storage overhead lower than k. The primary focus when studying PIR codes is to minimize the number of servers m (denoted as $P(s, k)$) for given values of k and s, and thus reduce the storage overhead. We presented constraints on the value of $P(s, k)$ and conducted practical tests using a SageMath program to determine minimal values of $P(s, k)$ for small values of s and k.

Chapter 3 is dedicated to the upper bounds on $P(s, k)$ presented in [3]. We verified all the upper bounds provided in that study.

The final chapter introduced code families similar to PIR codes, such as PIR array codes, which are an updated version where each server possesses more cells for storing entries. We also described locally repairable codes and batch codes.

Research on PIR codes is still relatively new, and there is considerable potential for improving $P(s, k)$ values for better practical use. Further studies on PIR array codes and their in-depth analysis, particularly identifying the conditions under which $P(s, t, k) < P(s, k)$, could also lead to significant advancements in this field.

Bibliography

[1] Eric Moorhouse, G. *Incidence Geometry*. 2007. `http://ericmoorhouse.org/handouts/Incidence_Geometry.pdf`.

[2] S. Barwick and G. Ebert. *Unitals in Projective Planes*. Springer, 2008.

[3] M. Giulietti, A. Sabatini, and M. Timpanella. PIR codes from combinatorial structures. 2021. `https://arxiv.org/abs/2107.01169`.

[4] M. Gezek, R. Mathon, and V. D. Tonchev. Maximal arcs, codes, and new links between projective planes. 2020. `https://arxiv.org/abs/2001.11431`.

[5] J. A. Thas. *Handbook of Incidence Geometry*. North-Holland, 1995.

[6] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50, 1995.

[7] A. Fazeli, A. Vardy, and E. Yaakobi. PIR with low storage overhead: Coding instead of replication. 2015. `http://arxiv.org/abs/1505.06241`.

[8] S. Kurz and E. Yaakobi. PIR codes with short block length. *Designs, Codes and Cryptography*, 89:1–29, 2021.

[9] R. D. Baker. Partitioning the planes of $AG_{2m}(2)$ into 2-designs. *Discrete Mathematics*, 15:205–211, 1976.

[10] G. Gévay. Resolvable configurations. *Discrete Applied Mathematics*, 266:319–330, 2019.

[11] S. R. Blackburn and T. Etzion. PIR array codes with optimal PIR rates. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 2658–2662, 2017.

[12] C. Wang and Y. Zhang. PIR array codes: the optimality of Blackburn-Etzion construction. In *2023 IEEE International Symposium on Information Theory (ISIT)*, pages 1342–1347, 2023.

[13] V. Skachek. Batch and PIR codes and their connections to locally repairable codes. 2016. `http://arxiv.org/abs/1611.09914`.

[14] H. Lipmaa and V. Skachek. Linear batch codes. 2014. `http://arxiv.org/abs/1404.2796`.

A. Attachments

A.1 Upper bounds on the number of servers in PIR codes

```
from sympy import factorint

def is_prime_power(num):
    factors_dict = list(factorint(int(num)).items());
    return len(factors_dict) == 1;

def partition(s, k):
    factors_dict = list(factorint(int(s)).items());
    factors = [];

    for i in range(len(factors_dict)):
        for j in range(factors_dict[i][1]):
            factors.append(factors_dict[i][0]);

    while (k < len(factors) + 1):
        first = factors.pop(0);
        factors[0] = first * factors[0];
        factors.sort();

    if (k == len(factors) + 1):
        suma = 0;
        for i in range(len(factors)):
            suma = suma + s/factors[i];
        return s + suma;
    return s*k;

def affine_planes(s, k):
    for N in range(2, math.ceil(s**(1/2))):
        q = s**(1/N);
        if (
            q.is_integer() and
            is_prime_power(q) and
            k <= (q**N - 1)/(q - 1) + 1
           ):
            return s + s*(k - 1)/q;
    return s*k;

def projective_planes(s, k):
    for N in range(2, math.ceil(s**(1/2))):
        if (
            N % 2 != 0 and
            2**(N + 1) - 1 == s and
            k <= 2**N
           ):
            return s + s*(k - 1)/3;
    return s*k;

def projective_planes_2(s, k):
    for q in range(3, s):
        if (not is_prime_power(q)):
```

```
            continue;
        i = 0;
        N = 2**(i + 1) - 1;
        while ((q**(N + 1) - 1)/(q - 1) < s):
            i = i + 1;
            N = 2**(i + 1) - 1;
        if (
            (q**(N + 1) - 1)/(q - 1) == s and
            k <= (q**N - 1)/(q - 1) + 1
        ):
            return s + s*(k - 1)/(q + 1);
    return s*k;

def maximal_arcs(s, k):
    for n_1 in range(math.ceil(s**(1/2))):
        for n_2 in range(n_1):
            if (
                s == 2**(n_1 + n_2) - 2**n_1 + 2**n_2 and
                k <= 2**n_1 + 2
            ):
                return s + s*(k - 1)/(2**n_2);
    return s*k;

def classical_unitals(s, k):
    q = (s - 1)**(1/3);
    if (
        q.is_integer() and
        is_prime_power(q) and
        k <= q**2 + 1
    ):
            return s + s*(k - 1)/(q + 1);
    return s*k;

def conics(s, k):
    for q in range(1, 2*math.floor(s**(1/2))):
        if (
            is_prime_power(q) and
            2*s == q**2 - q and
            k <= q + 1
           ):
            return s + (k - 1)*(q - 1);
    return s*k;

def steiner_system(s, k):
    value = s*k;
    if (s % 6 == 3 and k <= (s-1)/2 + 1):
        value = s + s*(k-1)/3;
    if (s % 12 == 4 and k <= (s-1)/3 + 1):
        value = s + s*(k-1)/4;
    if (s % 20 == 5 and k <= (s-1)/4 + 1):
        value = s + s*(s-1)/5;
    return value;

def configurations_3(s, k):
    value = s*k;
    b = s * (k - 1)/3;
    if (b.is_integer() and s >= 2 * (k - 1) + 1):
        return s + b;
```

```
    return value;

def configurations_4(s, k):
    value = s*k;
    b = s * (k - 1)/4;
    if (not b.is_integer()):
        return value;
    if (s % 12 == 4 and s >= 3 * (k - 1) + 1):
        return s + b;

    if (
        s % 12 == 0 and
        s >= 3 * (k - 1) + 1 and
        s not in {84, 120, 132, 180, 216, 264, 312, 324, 372, 456, 552, 648, 660, 804, 852, 888}
    ):
        return s + b;

    if (s % 12 == 0 and s == 3 * (k - 1) + 3):
        return s + b;

    a = (k - 1)/4;
    if (a.is_integer() and
        a >= 1 and a <= 15 and
        a != 3 and s != 38 and
        s > 3* (k - 1) + 1):
        return s + b;

    if (k == 5 and s >= 20 and s % 2 == 0):
        return s + (3 * s)/2;

    return value;

def configurations_5(s, k):
    value = s*k;
    b = s * (k - 1)/5;
    if (b.is_integer()):
        return value;
    if (s % 20 == 0 and s == 4 * (k - 1) + 4):
        return s + b;
    if (s % 20 == 5 and s >= 4 * (k - 1) + 1 and s > 7865):
        return s + b;
    a = (k - 1)/5;
    if (
        a.is_integer() and
        a >= 1 and a <= 10 and
        s >= 4*(k - 1) + 1 and
        (k - 1, s) not in {(1, 22), (2, 42), (2, 43), (3, 62), (3, 63), (4, 82), (5, 102), (7, 142), (9, 182),
                           (9, 183), (9, 185), (9, 186), (9, 187), (9, 188), (9, 189), (9, 190), (9, 191), (9, 192)}
    ):
        return s + b;
    return value;

def construction(s, k):
    sigma = math.ceil(s**(1/(k - 1)));
    return s + (k - 1)*sigma**(k - 2);
```

```
164
165 def construction_2(s, k):
166     if (k < 3):
167         return s*k;
168     for l in range(1, math.ceil(log(k, 2))):
169         if (2**l + 2 == k):
170             t = 0;
171             s_2 = 2**(2*l*t) - (2**(t + 1) - 1)**l;
172             while (s_2 < s):
173                 t = t + 1;
174                 s_2 = 2**(2*l*t) - (2**(t + 1) - 1)**l;
175             if (s_2 == s):
176                 return 2**(2*l*t) - 1;
177         if (2**l == k):
178             t = 0;
179             while ((2**t - 1)**l - 2 < s):
180                 t = t + 1;
181             if ((2**t - 1)**l - 2 == s):
182                 return 2**(l*t) - 1;
183     return s*k;
184
185 def construction_3(s, k):
186     n = 1;
187     s_1 = (n*(n - 1))/((k - 1)*(k - 2));
188     while (s_1 < s):
189         n = n + 1;
190         s_1 = (n*(n - 1))/((k - 1)*(k - 2));
191     if (s_1 == s and ((n-1)/(k-2)).is_integer()):
192         return n + s_1;
193     return s*k;
```

Algorithm for computing P(s,k)

```
1  class PIR_codes:
2      def __init__(self, s_range, k_range, debug):
3          self.s_range = s_range;
4          self.k_range = k_range;
5          self.debug = debug;
6          self.tableP = [[j*i for j in range(k_range + 1)] for i in
       range(s_range + 1)];
7
8
9      def P(self, s, k):
10         #known optimal values
11         if (k == 1):
12             self.tableP[s][k] = s;
13             return self.tableP[s][k];
14         elif (k == 2):
15             self.tableP[s][k] = s + 1;
16             if (self.debug):
17                 print('s: ', s, ', k: ', k, ' s + 1', self.tableP
       [s][k]);
18             return self.tableP[s][k];
19         elif (k == 2**(s - 1)):
20             self.tableP[s][k] = 2**s - 1;
21             if (self.debug):
```

```
                print('s: ', s, ', k: ', k, ' 2**s - 1', self.tableP[s][k]);
            return self.tableP[s][k];
        elif (s == 1):
            return k;
        elif (s == 2):
            self.tableP[s][k] = math.ceil((3*k)/2);
            if (self.debug):
                print('s: ', s, ', k: ', k, ' 3k/2', self.tableP[s][k]);
            return self.tableP[s][k];
        elif (s == 3):
            self.tableP[s][k] = math.ceil((7*k)/4);
            if (self.debug):
                print('s: ', s, ', k: ', k, ' 7k/4', self.tableP[s][k]);
            return self.tableP[s][k];

        # values from combinatorial structures
        value = affine_planes(s, k);
        if (value < self.tableP[s][k]):
            if (self.debug):
                print('s: ', s, ', k: ', k, 'affine_planes(s, k)', value)
            self.tableP[s][k] = int(value);

        value = projective_planes(s, k);
        if (value < self.tableP[s][k]):
            if (self.debug):
                print('s: ', s, ', k: ', k, 'projective_planes(s, k)', value)
            self.tableP[s][k] = int(value);

        value = projective_planes_2(s, k);
        if (value < self.tableP[s][k]):
            if (self.debug):
                print('s: ', s, ', k: ', k, 'projective_planes_2(s, k)', value)
            self.tableP[s][k] = int(value);

        value = maximal_arcs(s, k);
        if (value < self.tableP[s][k]):
            if (self.debug):
                print('s: ', s, ', k: ', k, 'maximal_arcs(s, k)', value)
            self.tableP[s][k] = int(value);

        value = classical_unitals(s, k);
        if (value < self.tableP[s][k]):
            if (self.debug):
                print('s: ', s, ', k: ', k, 'classical_unitals(s, k)', value)
            self.tableP[s][k] = int(value);

        value = conics(s, k);
        if (value < self.tableP[s][k]):
            if (self.debug):
```

```
                print('s: ', s, ', k: ', k, 'conics(s, k)', value
    )
            self.tableP[s][k] = int(value);

        value = steiner_system(s, k);
        if (value < self.tableP[s][k]):
            if (self.debug):
                print('s: ', s, ', k: ', k, 'steiner_system(s, k)
    ', value)
            self.tableP[s][k] = int(value);

        value = configurations_3(s, k);
        if (value < self.tableP[s][k]):
            if (self.debug):
                print('s: ', s, ', k: ', k, 'configurations_3(s,
    k)', value)
            self.tableP[s][k] = int(value);

        value = configurations_4(s, k);
        if (value < self.tableP[s][k]):
            if (self.debug):
                print('s: ', s, ', k: ', k, 'configurations_4(s,
    k)', value)
            self.tableP[s][k] = int(value);

        value = configurations_5(s, k);
        if (value < self.tableP[s][k]):
            if (self.debug):
                print('s: ', s, ', k: ', k, 'configurations_5(s,
    k)', value)
            self.tableP[s][k] = int(value);

        value = partition(s, k);
        if (value < self.tableP[s][k]):
            self.tableP[s][k] = int(value);
            if (debug):
                print('s: ', s, ', k: ', k, 'partition(s, k)',
    self.tableP[s][k])

        value = construction(s, k);
        if (value < self.tableP[s][k]):
            self.tableP[s][k] = int(value);
            if (self.debug):
                print('s: ', s, ', k: ', k, 'construction(s, k)',
     self.tableP[s][k])

        value = construction_2(s, k);
        if (value < self.tableP[s][k]):
            self.tableP[s][k] = int(value);
            if (self.debug):
                print('s: ', s, ', k: ', k, 'construction_2(s, k)
    ', self.tableP[s][k])

        value = construction_3(s, k);
        if (value < self.tableP[s][k]):
            self.tableP[s][k] = int(value);
            if (self.debug):
```

```
                print('s: ', s, ', k: ', k, 'construction_3(s, k)', self.tableP[s][k])

        return self.tableP[s][k];


    #values depending on other values
    def P_update(self, s, k):
        updated = False;
        if (k < self.k_range):
            if (k % 2 != 0 and self.tableP[s][k + 1] > self.tableP[s][k] + 1):
                if (self.debug):
                    print('s: ', s, ', k: ', k, ' P(s,k + 1) = P(s, k) + 1', self.tableP[s][k] + 1);
                self.tableP[s][k + 1] = self.tableP[s][k] + 1;
                updated = True;

            value = self.tableP[s][k + 1] - 1;
            if (value < self.tableP[s][k]):
                if (self.debug):
                    print('s: ', s, ', k: ', k, ' P(s, k) = P(s, k + 1) - 1 = ', value);
                self.tableP[s][k] = value;
                updated = True;
        if (s < self.s_range):
            value = self.tableP[s + 1][k] - 1;
            if (value < self.tableP[s][k]):
                if (self.debug):
                    print('s: ', s, ', k: ', k, ' P(s, k) = P(s + 1, k) - 1 = ', value);
                self.tableP[s][k] = value;
                updated = True;

        for s_1 in range(1, math.floor(s/2) + 1):
            s_2 = s - s_1;
            value = self.tableP[s_1][k] + self.tableP[s_2][k];
            if (value < self.tableP[s][k]):
                if (self.debug):
                    print('s: ', s, ', k: ', k,', s_1: ', s_1, ', s_2: ' , s_2, ' P(s1 + s2, k) <= P(s1, k) + P(s2, k) = ', value);
                self.tableP[s][k] = value;
                updated = True;

        for k_1 in range(1, math.floor(k/2) + 1):
            k_2 = k - k_1;
            value = self.tableP[s][k_1] + self.tableP[s][k_2];
            if (value < self.tableP[s][k]):
                if (self.debug):
                    print('s: ', s, ', k: ', k, ', k_1: ', k_1, ', k_2:', k_2, ' P(s, k1 + k2) <= P(s, k1) + P(s, k2) = ', value);
                self.tableP[s][k] = value;
                updated = True;

        return updated;
```

Printing table of $P(s, k)$ values

```
pir_codes = PIR_codes(35, 12, False);

for s in range(1, pir_codes.s_range + 1):
    for k in range(1, pir_codes.k_range + 1):
        pir_codes.P(s,k);

updated = True;
while (updated):
    updated = False;
    for s in range(1, pir_codes.s_range + 1):
        for k in range(1, pir_codes.k_range + 1):
            updated = pir_codes.P_update(s,k) or updated;

for i in range(pir_codes.k_range + 1):
    pir_codes.tableP[0][i] = i;
for i in range(pir_codes.s_range + 1):
    pir_codes.tableP[i][0] = i;
pir_codes.tableP[0][0] = 's/k'

print('P(s, k) table:');
for row in pir_codes.tableP:
    form = [' {:^4} '  for i in range(pir_codes.k_range + 1)];
    print(('|' + '|'.join(form) ).format(*row));

for i in range(1, len(pir_codes.tableP)):
    row = pir_codes.tableP[i];
    for j in range(1, len(row)):
        row[j] = float(row[j])/i;

print('\nStorage overhead table:');
form_str = ' {:^5} ';
for row in pir_codes.tableP:
    form = [form_str for i in range(pir_codes.k_range)];
    print(('| {:^4} |' + '|'.join(form)).format(*row));
    form_str = ' {:5.2f} ';
```