Summer 25

Code Equivalence

Lecturer: Violetta Weger

These lecture notes were prepared for the Summer School Finite Geometry and Friends, being held in September 2025 at the Vrije Universiteit Brussel. Any comments or corrections can be sent to violetta.weger@tum.de.

The distribution of the notes is of course allowed.

Overview

"Code Equivalence" describes the problem of finding a linear isometry between two codes. An isometry is a map which preserves the weight, in our case: the Hamming weight. We can easily identify the linear isometries in the Hamming metric to be all monomial transformations.

Thus the problem reads as follows:

$$\text{Given } \mathcal{C}, \mathcal{C}' \subseteq \mathbb{F}_q^n \text{ find } \varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n, \text{ such that } \varphi(\mathcal{C}) = \mathcal{C}'.$$

Or equivalently: Given their generator matrices, $G, G' \in \mathbb{F}_q^{k\times n}$ find $S \in \mathrm{GL}_q(k)$, a $n \times n$ permutation matrix P and a diagonal matrix $D = \mathrm{diag}(d)$, for $d \in (\mathbb{F}_q^\star)^n$, such that $SGDP = G'$.

While the problem is interesting from a pure coding-theoretic perspective, its recent use in the signature scheme LESS has put the problem in the center of attention of the cryptographic community as well. In fact, LESS is one of the few surviving signature schemes in the second round of the additional standardization call by NIST.

The bothersome part about this problem, is that we *know* it is *not* NP-hard, without having an efficient solver.

The whole research area is only now emerging and thus still under-explored. Thus, as of today, any outcome is within the range of possibilities. In the best (or worst, depending on your stand point) scenario, we might soon find a (quasi-) polynomial time solver, making LESS obsolete.

Due to its novelty, large impact and cute relations to other topics, I decided to present you this problem in the Summer School Finite Geometry and Friends. Although the view point of codes as projective systems is not helpful to solve it, the more people with diverse backgrounds and approaches try tackling this problem, the closer we might get to finally understanding its hardness- or breaking it.

We will start with a quick introduction to coding theory and cryptography, before we delve into the many connections of the problem, the different view points on it and some solvers. We will then conclude this lecture with some new results and ideas.

Material: Most of the content of these lecture notes is collected from the seminal papers

- Mohamed Ahmed Saeed. *Algebraic approach for code equivalence.* Diss. Normandie Université; University of Khartoum, 2017. [23]
- Nicolas Sendrier and Dimitrios E. Simos. *How easy is code equivalence over $\mathbb{F}_q$?* International Workshop on Coding and Cryptography-WCC 2013. 2013.
- Magali Bardet, Ayoub Otmani and Mohamed Saeed-Taha. *Permutation code equivalence is not harder than graph isomorphism when hulls are trivial.* 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019.

Hence, if this introductory lecture got you interested, I highly recommend reading these references.

Contents

Notation

Throughout these lecture notes, we will make use of the following notation

- Let $\mathbb{F}_q$ denote the finite field with q elements and $\mathbb{F}_q^\star$ the multiplicative group $\mathbb{F}_q \setminus \{0\}$.
- For a set S we denote by $|S|$ its cardinality and by S^C its complement.
- Id_n denotes the identity matrix of size n.
- $\mathrm{GL}_q(n)$ denotes the general linear group of degree n in $\mathbb{F}_q$, i.e., all invertible matrices in $\mathbb{F}_q^{n\times n}$.
- For a matrix M we write $\mathrm{rk}(M)$ to denote its rank and by $M^\top$ we denote its transpose.
- For a function f we denote by $\ker(f)$ its kernel and by $\mathrm{im}(f)$ its image.
- For a vector $v \in \mathbb{F}_q^k$ and $i \in \{1,\dots,k\}$ we denote by v_i the ith entry of the vector v. For a subset $S \subset \{1,\dots,k\}$ of size s, $v_S \in \mathbb{F}_q^s$ denotes the vector consisting of the entries of v indexed by S.
- Similarly for a matrix: for a matrix $M \in \mathbb{F}_q^{k\times n}$ and $i \in \{1,\dots,k\}, j \in \{1,\dots,n\}$ we denote by $M_{i,j}$ the entry of M in the ith row and jth column. For a subset $S \subset \{1,\dots,n\}$ of size s, $M_S \in \mathbb{F}_q^{k\times s}$ denotes the matrix consisting of the columns of M indexed by S.
- For a matrix $M \in \mathbb{F}_q^{k\times n}$, we denote by $\langle M\rangle \subseteq \mathbb{F}_q^n$ the span of the rows of M, that is $\mathrm{im}(M) = \langle M\rangle$.

1 Background and Motivation: Coding and Cryptography

As this summer school is intended for *any finite friend*, I do not assume any expertise in coding theory or cryptography and hence we start with introducing all the necessary notions of coding theory, before explaining more in detail, how the code equivalence problem is used in LESS.

1.1 Basics of Coding Theory

In this section, we give the basics of algebraic coding theory. This is a short version of Chapter 2 in `https://user.math.uzh.ch/weger/CT.pdf`.

Algebraic codes are immensely used in our digital lives, due to their ability to correct errors. In the most prominent example, we want to send a message $m \in \mathbb{F}_q^k$ through a noisy channel. Thus, we first encode m to a codeword $c \in \mathcal{C} \subseteq \mathbb{F}_q^n$, and upon receiving $r = c + e$, we can use a decoding algorithm for the code $\mathcal{C}$ to recover c (and thus also m).

However, this lecture will *not* be about decoding, or the error correction capability of codes. We only use this application, to explain the interest in the Hamming metric.

Let us fix that $\mathbb{F}_q$ will denote the finite field of q elements, where q is a prime power.

Definition 1.1 (Linear Code)**.** Let $1 \leq k \leq n$ be integers. Then, an $[n,k]_q$ *linear code* $\mathcal{C}$ over $\mathbb{F}_q$ is a k-dimensional linear subspace of $\mathbb{F}_q^n$.

We say that $\mathcal{C}$ has length n, dimension k, and call its elements *codewords.*
As $\mathcal{C}$ is linear, it must have some basis, which allows us to represent it compactly. In fact, linear codes allow for an easy representation through their *generator matrices*, which have the code as an image.

Definition 1.2 (Generator Matrix)**.** Let $k \leq n$ be positive integers and let $\mathcal{C}$ be an $[n,k]_q$ linear code. Then, a matrix $G \in \mathbb{F}_q^{k \times n}$ is called a *generator matrix* of $\mathcal{C}$ if

$$\mathcal{C} = \left\{ xG \mid x \in \mathbb{F}_q^k \right\},$$

that is, the rows of G form a basis of $\mathcal{C}$.

We will often write $\langle G \rangle$ to denote the code generated by the rows G. Note that by multiplying a generator matrix on the left with an invertible matrix $S \in \mathrm{GL}_q(k)$, we simply change the basis of the code, and thus SG is again a generator matrix.

We say that $\mathcal{C}$ is *degenerate* if there exists a $i \in \{1, \ldots, n\}$ such that for all $c \in \mathcal{C}$, we have $c_i = 0$. Equivalently, a code is degenerate, if a generator matrix G has a zero column.

One can also represent a code through a matrix H, which has the code as kernel.

Definition 1.3 (Parity-Check Matrix)**.** Let $k \leq n$ be positive integers and let $\mathcal{C}$ be an $[n,k]_q$ linear code. Then, a matrix $H \in \mathbb{F}_q^{(n-k) \times n}$ is called a *parity-check matrix* of $\mathcal{C}$, if

$$\mathcal{C} = \left\{ y \in \mathbb{F}_q^n \mid yH^\top = 0 \right\}.$$

For any $x \in \mathbb{F}_q^n$, we call $xH^\top$ the *syndrome* of x through H.

Since $\mathcal{C} = \mathrm{Im}(G) = \ker(H^\top)$, we also get a relation between the two matrices, namely

$$GH^\top = 0.$$

For $x, y \in \mathbb{F}_q^n$ let us denote by $\langle x, y \rangle$ the standard inner product, i.e.,

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i.$$

Then, we can define the dual of an $[n,k]_q$ linear code $\mathcal{C}$ as the orthogonal space of $\mathcal{C}$.

Definition 1.4 (Dual Code)**.** Let $k \leq n$ be positive integers and let $\mathcal{C}$ be an $[n,k]_q$ linear code. The *dual code* $\mathcal{C}^\perp$ is an $[n, n-k]_q$ linear code, defined as

$$\mathcal{C}^\perp = \{x \in \mathbb{F}_q^n \mid \langle x, y \rangle = 0 \; \forall \; y \in \mathcal{C}\}.$$

Thus, the parity-check matrix is in fact the generator matrix of the dual code:

Exercise 1.5. *Let $\mathcal{C}$ be an $[n,k]_q$ linear code and $H \in \mathbb{F}_q^{(n-k)\times n}$ be a parity-check matrix of $\mathcal{C}$. Show that $\langle H \rangle = \mathcal{C}^\perp$.*

In turn, taking the dual of the dual brings us back to our code:

Proposition 1.6. *Let q be a prime power and $k \leq n$ be positive integers. Let $\mathcal{C}$ be an $[n,k]_q$ linear code. Then $(\mathcal{C}^\perp)^\perp = \mathcal{C}$.*

Exercise 1.7. *Prove Proposition 1.6.*

Let us have a look at an example to see all these notions in action.

Example 1.8. *Let us consider $\mathbb{F}_5$ and the code $\mathcal{C} \subseteq \mathbb{F}_5^4$ generated by*

$$G = \begin{pmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 4 \end{pmatrix}.$$

Thus, $\mathcal{C} = \langle G \rangle$ is a $[4,2]_5$ linear code, that is the length is 4 and the dimension is 2.

We could write out all $|\mathcal{C}| = 5^2 = 25$ codewords, but using G is much more efficient. By multiplying G with some invertible matrix S, e.g. $S = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$ we get a different generator matrix of the same code:

$$SG = \begin{pmatrix} 1 & 2 & 2 & 1 \\ 0 & 1 & 0 & 4 \end{pmatrix}.$$

We can also compute a parity-check matrix for $\mathcal{C}$ (a smart trick is revealed later) as

$$H = \begin{pmatrix} 3 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

In fact, for any codeword, e.g. $c = (1,0,2,0)$, we have that $cH^\top = 0$.

The dual code $\mathcal{C}^\perp$ is then generated by H and we can see that $c = (1,0,2,0)$ is also in the dual.

A priori, the code $\mathcal{C}$ and its dual $\mathcal{C}^\perp$ have no apparent connection. If $\mathcal{C} \subseteq \mathcal{C}^\perp$, we call $\mathcal{C}$ a *self-orthogonal* code and if $\mathcal{C} = \mathcal{C}^\perp$, we call $\mathcal{C}$ a *self-dual* code.

Example 1.9. *Let us consider the $[n, 1]_2$ repetition code $\mathcal{C}$ with generator matrix*

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

Note that $\mathcal{C}^\perp = ker(G^\top)$, hence to show that $\mathcal{C} \subseteq \mathcal{C}^\perp$, it is enough to show that any codeword $c \in \mathcal{C}$ is such that $cG^\top = 0$. If n is even, then $GG^\top = \sum_{i=1}^{n} 1 = n$ and thus any code word $c = mG$ is such that $cG^\top = mGG^\top = 0$.

Exercise 1.10. *Let $\mathcal{C}$ be an $[n, k]_q$ linear code with generator matrix G. Show that if $GG^\top = 0$, then $\mathcal{C}$ is self-orthogonal.*

Exercise 1.11. *Let $\mathcal{C}$ be an $[n, k]_q$ linear code. Show that if $k = n/2$ and $\mathcal{C}$ is self-orthogonal, then $\mathcal{C}$ self-dual.*

Definition 1.12 (Systematic Form)**.** Let $k \leq n$ be positive integers and $\mathcal{C}$ be an $[n, k]_q$ linear code. Then, there exist some $n \times n$ permutation matrix P and some invertible matrix $S \in \mathbb{F}_q^{k\times k}$ that bring G in *systematic form*, i.e.,

$$SGP = \begin{pmatrix} \mathrm{Id}_k & A \end{pmatrix},$$

where $A \in \mathbb{F}_q^{k\times(n-k)}$. Similarly, there exist some $n \times n$ permutation matrix P' and some invertible matrix $S' \in \mathbb{F}_q^{(n-k)\times(n-k)}$, that bring H into systematic form, i.e.,

$$S'HP' = \begin{pmatrix} B & \mathrm{Id}_{n-k} \end{pmatrix},$$

where $B \in \mathbb{F}_q^{(n-k)\times k}$.

Together with the relation $GH^\top = 0$, this immediately gives us a way to compute a parity-check matrix, when given a generator matrix:

Proposition 1.13. *Let $\mathcal{C}$ be an $[n, k,]_q$ linear code and G be a generator matrix. If $G = \begin{pmatrix} Id_k & A \end{pmatrix}$, for some $A \in \mathbb{F}_q^{k\times(n-k)}$, then $H = \begin{pmatrix} -A^\top & Id_{n-k} \end{pmatrix}$ is a parity-check matrix of $\mathcal{C}$.*

As in our example, there exist codewords which live in both; the code and the dual code.

Definition 1.14. Let q be a prime power and $k \leq n$ be positive integers. Let $\mathcal{C}$ be a $[n, k]_q$ linear code. The *hull* of the code $\mathcal{C}$ is defined as

$$\mathcal{H}(\mathcal{C}) = \mathcal{C} \cap \mathcal{C}^\perp.$$

Clearly, $\mathcal{H}(\mathcal{C}) = \mathcal{H}(\mathcal{C}^\perp)$, by Proposition 1.6.

Exercise 1.15. *Let q be a prime power and $k \leq n$ be positive integers. Let $\mathcal{C}$ be a $[n, k]_q$ linear code with generator matrix $G \in \mathbb{F}_q^{k\times n}$ and parity-check matrix $H \in \mathbb{F}_q^{(n-k)\times n}$. Show that*

$$\mathcal{H}(\mathcal{C}) = ker\left(\begin{pmatrix} G \\ H \end{pmatrix}^\top\right).$$

In [24], Sendrier computed the average dimension of the hull of random codes. The proof is not easy and the result has been restated several times, to become more of a folklore. But let us start with what we mean by "random".

A *random code*, is a code generated by a full rank matrix $G \in \mathbb{F}_q^{k\times n}$ chosen uniform at random.

For such random codes, the original statement says "the expected dimension of the hull of a random code is constant" and the small constant becomes negligible for large finite fields.

Codes with hull $\mathcal{H}(\mathcal{C}) = \{0\}$ are also called *linear complementary dual codes* and in this case, we say that the hull is trivial.

Theorem 1.16. *Let q be a prime power and $k \leq n$ be positive integers. Let $\mathcal{C}$ be a $[n,k]_q$ linear code with generator matrix $G \in \mathbb{F}_q^{k\times n}$ and parity-check matrix $H \in \mathbb{F}_q^{(n-k)\times n}$. Then $\mathcal{H}(\mathcal{C}) = \{0\}$ with high probability, for n growing.*

While the proof of [24] uses complicated mass formulas, there exist several short-cuts. These usually only show that random codes have trivial hull with probability $\geq 1 - 1/q$, but this will be enough for us.

Exercise 1.17. *Let q be a prime power and $k \leq n$ be positive integers. Let $\mathcal{C}$ be a $[n,k]_q$ linear code with generator matrix $G \in \mathbb{F}_q^{k\times n}$ in systematic form, i.e., $G = \begin{pmatrix} Id_k & A \end{pmatrix}$ for $A \in \mathbb{F}_q^{k\times(n-k)}$. Show that if $AA^\top + Id_{n-k}$ is full rank, then dim$(\mathcal{H}(\mathcal{C})) = 0$.*

Exercise 1.18. *Let q be a prime power and $k \leq n$ be positive integers. Let $\mathcal{C}$ be a $[n,k]_q$ linear code with generator matrix $G \in \mathbb{F}_q^{k\times n}$. Show that if $GG^\top$ has full rank, then dim$(\mathcal{H}(\mathcal{C})) = 0$.*

In fact, the actual short-cut proofs would bound

$$\mathbb{P}(\text{rk}(GG^\top) < k) \leq 1/q.$$

As we are usually interested in the amount of positions which are erroneous, the most natural weight to consider is the *Hamming metric*.

Definition 1.19 (Hamming Metric)**.** Let n be a positive integer. For $x \in \mathbb{F}_q^n$, the *Hamming weight* of x is given by the number of non-zero positions, i.e.,

$$\text{wt}_H(x) = \mid \{i \in \{1,\ldots,n\} \mid x_i \neq 0\} \mid .$$

For $x, y \in \mathbb{F}_q^n$, the *Hamming distance* between x and y is given by the number of positions in which they differ, i.e.,

$$d_H(x,y) = \mid \{i \in \{1,\ldots,n\} \mid x_i \neq y_i\} \mid .$$

Note that the Hamming distance is induced by the Hamming weight, that is $d_H(x,y) = \text{wt}_H(x-y)$. We can also consider the minimum distance of a code, i.e., the smallest distance between any two distinct codewords.

Definition 1.20 (Minimum Distance)**.** Let $\mathcal{C}$ be a linear code over $\mathbb{F}_q$. The *minimum Hamming distance* of $\mathcal{C}$ is denoted by $d_H(\mathcal{C})$ and given by

$$d_H(\mathcal{C}) = \min\{d_H(x,y) \mid x,y \in \mathcal{C},\ x \neq y\} = \min\{\text{wt}_H(c) \mid c \in \mathcal{C}, c \neq 0\}.$$

The minimum Hamming distance of a code turns out to be a very important parameter. Thus, whenever the minimum Hamming distance $d = d_H(\mathcal{C})$ is known, we say $\mathcal{C}$ is an $[n,k,d]_q$ linear code.

Let $\mathcal{C}$ be an $[n,k,d]_q$ linear code with $\langle G \rangle = \mathcal{C} = \ker(H^\top)$.

- The parameter n is called the *length* of the code.
- The parameter k is called the *dimension* of the code.
- The elements in the code are called *codewords*.
- The matrix G is called a *generator matrix* of the code.
- The matrix H is called a *parity-check matrix* of the code.
- The vector $s = xH^\top$ is called the *syndrome* of x.
- The code $\mathcal{C}^\perp$ is called the *dual code* of $\mathcal{C}$.
- The *hull* of $\mathcal{C}$ is given by $\mathcal{H}(\mathcal{C}) = \mathcal{C} \cap \mathcal{C}^\perp$.
- The parameter d is called the *minimum Hamming distance* of the code.

1.2 Equivalence of Codes

In mathematics we often ask when two objects are "essentially" the same. Let us clarify what that means for codes.

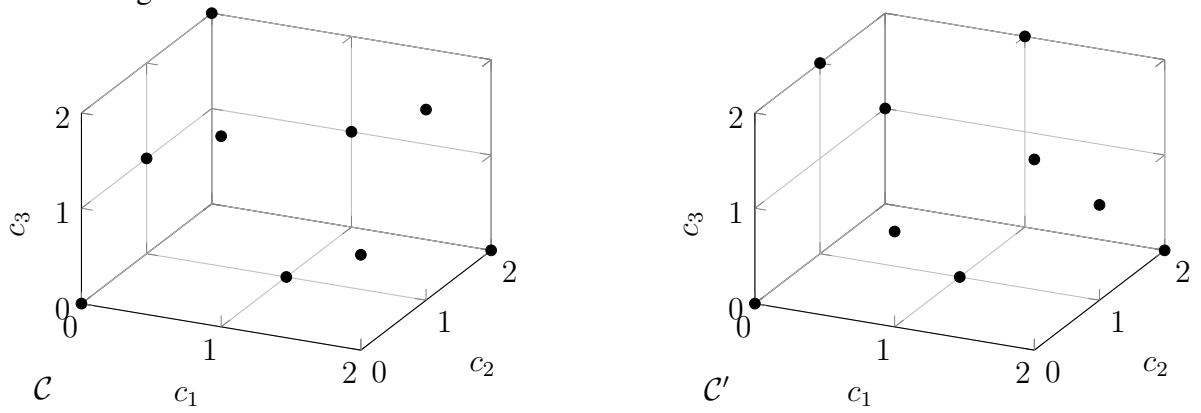
Let us consider the following two codes over $\mathbb{F}_3$: $\mathcal{C} = \langle G \rangle$ and $\mathcal{C}' = \langle G' \rangle$.

$$G = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{pmatrix}, \quad G' = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

Then

$$\mathcal{C} = \{(0,0,0), (1,0,2), (2,0,1), (1,1,0), (2,1,2), (0,1,1), (0,2,2), (1,2,1), (2,2,0)\}$$
$$\mathcal{C}' = \{(0,0,0), (0,1,2), (0,2,1), (1,1,0), (1,2,2), (1,0,1), (2,0,2), (2,1,1), (2,2,0)\}$$

While they are not the same code, their points seem to be just rotated, and all still have the same distance among each other.



Thus, we say two codes are essentially the same, if we can map one code linearly to the other, in such a way that the distances between the codewords stays the same.

Definition 1.21. A *linear isometry* for a distance function d is a linear map $\varphi : \mathbb{F}_q^n \to \mathbb{F}_q^n$, such that for all $x, y \in \mathbb{F}_q^n$ we have that

$$d(x, y) = d(\varphi(x), \varphi(y)).$$

Clearly, when dealing with a distance which is induced from a weight wt, then we can equivalently define a linear isometry φ to be such that for all $x \in \mathbb{F}_q^n$

$$\mathrm{wt}(x) = \mathrm{wt}(\varphi(x)).$$

Proposition 1.22. *The linear isometries with respect to some distance function form a group with respect to the composition.*

Exercise 1.23. *Prove Proposition 1.22 and observe that any linear isometry is a $\mathbb{F}_q$-isomorphism.*

In our case, we are interested in the linear isometries for the Hamming metric.

Proposition 1.24. *The linear isometries for the Hamming metric are given by the semidirect product $(\mathbb{F}_q^\star)^n \rtimes S_n$, where S_n denotes the symmetric group of degree n.*

Proof. For the first direction, we note that $\varphi = (d, \sigma) \in (\mathbb{F}_q^\star)^n \rtimes S_n$ is linear and can be written as matrix multiplication, where $\varphi(x) = xDP$, for $D = \mathrm{diag}(d_1, \ldots, d_n)$ a diagonal matrix with entries $d_i \in \mathbb{F}_q^\star$ and P an $n \times n$ permutation matrix belonging to the permutation σ. Thus,

$$\varphi(x_1, \ldots, x_n) = (d_{\sigma^{-1}(1)} x_{\sigma^{-1}(1)}, \ldots, d_{\sigma^{-1}(n)} x_{\sigma^{-1}(n)})$$

and if $x_i \neq 0$, then $x_{\sigma(i)} \neq 0$ and by multiplying with a non-zero scalar, we still have non-zero. On the other hand, if $x_i = 0$, then $x_{\sigma(i)} = 0$ and it remains zero after multiplying with some non-zero d_j.

For the other direction, let us assume that φ is a linear isometry and denote by e_i the standard vector having all zero entries but a 1 in position i. These vectors clearly span the whole $\mathbb{F}_q^n$ and hence to define a linear map φ, it is enough to know where φ sends the basis vectors e_i for $i \in \{1, \ldots, n\}$. In fact, any $x \in \mathbb{F}_q^n$ is such that $x = \sum_{i=1}^n e_i \lambda_i$ for $\lambda_1, \ldots, \lambda_n \in \mathbb{F}_q$. By the linearity of φ we thus get

$$\varphi(x) = \sum_{i=1}^{n} \varphi(e_i) \lambda_i.$$

For all $e_i = (0, \ldots, 0, 1, 0 \ldots, 0)$ of Hamming weight 1, we must have

$$\varphi(e_i) \in \{\lambda(i) e_{j(i)} \mid \lambda(i) \in \mathbb{F}_q^\star, j(i) \in \{1, \ldots, n\}\}.$$

If we assign each of the e_i for all $i \in \{1, \ldots, n\}$, then there exists a permutation σ, which sends $i \mapsto j(i)$, and scalar multiples $d_{j(i)} = \lambda(i)$.

In fact, if the map $i \mapsto j(i)$ is not a permutation, then there exist $i \neq i'$ with $j(i) = j(i')$ and hence $\varphi(e_i + e_{i'})$ is some multiple of $e_{j(i)}$. This, however, contradicts that φ is an isometry: $\mathrm{wt}_H(e_i + e_{i'}) = 2$, whereas $\mathrm{wt}_H(\lambda e_{j(i)}) = 1$. □

Matrices of the form DP, for D a diagonal matrix and P a permutation matrix are also called *monomial matrices* and $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$ *monomial transforms.* We will use these notions interchangably, i.e., $\varphi = (D, P)$.

Note that there also exist the semi-linear isometries, as we also have the automorphisms of the finite field itself. The *semi-linear isometries* for the Hamming metric are then given by $(\mathbb{F}_q^\star)^n \rtimes (\mathrm{Aut}(\mathbb{F}_q) \times S_n)$. For cryptographic purposes, this type of isometry is however not interesting, and we will hence ignore it.

If we have a linear isometry between two codes, i.e., let $\mathcal{C}, \mathcal{C}'$ be two $[n, k]_q$ linear codes and there exists a linear map $\varphi : \mathcal{C} \to \mathcal{C}'$ which preserves the weight, that is for all $c \in \mathcal{C}$ we have $\mathrm{wt}_H(c) = \mathrm{wt}_H(\varphi(c))$, we might apriori get other maps than the monomial transforms. Luckily, MacWilliams [20] showed that this is not the case.

Theorem 1.25 (Extension Theorem)**.** *Let $\mathcal{C}, \mathcal{C}'$ be two $[n,k]_q$ linear codes and there exists a linear map $\varphi : \mathcal{C} \to \mathcal{C}'$ which preserves the weight, then there exists a linear isometry $\mu : \mathbb{F}_q^n \to \mathbb{F}_q^n$ with $\mu_{|\mathcal{C}} = \varphi$.*

That is, any linear weight preserving map that we may find between two codes can be extended to a linear isometry of the whole ambient space.

Definition 1.26. Let $\mathcal{C}, \mathcal{C}'$ be $[n,k]_q$ linear codes. We say that $\mathcal{C}$ is *linearly equivalent* to $\mathcal{C}'$ if there exists a $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$ such that $\varphi(\mathcal{C}) = \mathcal{C}'$.

Additionally, we say that $\mathcal{C}$ is *permutation equivalent* to $\mathcal{C}'$ if there exists a $\sigma \in S_n$ such that $\sigma(\mathcal{C}) = \mathcal{C}'$.

This leads us to the main problem of this lecture:

Problem 1.27. *Given two $[n,k]_q$ linear codes $\mathcal{C}, \mathcal{C}'$, find, if one exists, a linear isometry $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$ such that $\varphi(\mathcal{C}) = \mathcal{C}'$.*

The equivalence between the two codes also gives rise to a condition for their generator matrices.

Proposition 1.28. *Let $\mathcal{C}, \mathcal{C}'$ be $[n,k]_q$ linear codes with generator matrices G, respectively G'. If $\mathcal{C}$ is linearly equivalent to $\mathcal{C}'$, then there exist matrices $S \in GL_q(k), D = diag(d_1, \ldots, d_n)$ with $d_i \in \mathbb{F}_q^\star$ and a $n \times n$ permutation matrix P, such that*

$$SGDP = G'.$$

This also includes the case of permutation equivalence by setting $d_i = 1$ for all $i \in \{1, \ldots, n\}$.

Exercise 1.29. *Prove Proposition 1.28.*

Example 1.30. *Let us consider $\mathbb{F}_4 = \mathbb{F}_2(\alpha)$ where $\alpha^2 = \alpha + 1$.*
Let $\mathcal{C} = \langle G \rangle$ where

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

Then we may apply the permutation $\sigma = (1,2)$ to get

$$GP = \begin{pmatrix} 0 & 1 & \alpha \\ 1 & 0 & \alpha + 1 \end{pmatrix}$$

and $\langle GP \rangle$ is permutation equivalent to $\mathcal{C}$.
If we also apply the diagonal matrix $D = diag(1, \alpha, \alpha + 1)$, we get

$$GPD = \begin{pmatrix} 0 & \alpha & 1 \\ 1 & 0 & \alpha \end{pmatrix}$$

with $\langle GPD \rangle$ is linearly equivalent to $\langle GP \rangle$ and to $\mathcal{C}$.

On the other hand we also have linear isometries from $\mathcal{C}$ to itself.

Definition 1.31 (Automorphism Group)**.** Let $\mathcal{C}$ be an $[n,k]_q$ linear code. The *automorphism group* of $\mathcal{C}$ is given by the linear isometries that map $\mathcal{C}$ to $\mathcal{C}$:

$$\text{Aut}(\mathcal{C}) = \{\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n \mid \varphi : \mathcal{C} \to \mathcal{C}\}.$$

Just like the hull, the automorphism group of a random linear code is with high probability trivial [17], i.e., $\text{Aut}(\mathcal{C}) = \{\text{id}\}$.

Exercise 1.32. *Give the automorphism group of* $\mathcal{C} = \langle (1,0,0),(0,1,1)\rangle \subseteq \mathbb{F}_2^3$.

Exercise 1.33. *Let* $\varphi \in$ *Aut*$(\mathcal{C})$ *be a permutation. Show that* $\varphi \in$ *Aut*$(\mathcal{C} \cap \mathcal{C}^\perp)$.

Proposition 1.34. *Let* $\mathcal{C}_1, \mathcal{C}_2$ *be two permutation equivalent* $[n,k,d]_q$ *linear codes. Then* $\mathcal{C}_1^\perp$ *is permutation equivalent to* $\mathcal{C}_2^\perp$.

Proof. Let $\sigma \in S_n$ be such that $\sigma(\mathcal{C}_1) = \mathcal{C}_2$ and denote by P the permutation matrix with respect to σ. Let G_1, G_2 be generator matrices for $\mathcal{C}_1$, respectively $\mathcal{C}_2$, then there exist a $S \in \text{GL}_q(k)$ such that $SG_1P = G_2$.

Let H_1, H_2 be the parity-check matrices for $\mathcal{C}_1$, respectively $\mathcal{C}_2$. Since $G_2H_2^\top = 0$, we also have $G_1PH_2^\top = G_1(H_2P^\top)^\top = 0$. This implies that $H_2P^\top$ is a parity-check matrix for $\mathcal{C}_1$ and hence $H_2 = S'H_1P$, for some $S' \in \text{GL}_q(n-k)$. Thus, $\sigma(\mathcal{C}_1^\perp) = \mathcal{C}_2^\perp$.

□

Exercise 1.35. *Let* $\mathcal{C}_1, \mathcal{C}_2$ *be linearly equivalent codes. Show that* $\mathcal{C}_1^\perp$ *is linearly equivalent to* $\mathcal{C}_2^\perp$. Hint: *Use the fact that* $G_2H_2^\top = 0$ *and* $SG_1PD = G_2$.

For two permutation equivalent codes, their hulls are also permutation equivalent.

Proposition 1.36. *Let* $\mathcal{C}_1, \mathcal{C}_2$ *be two permutation equivalent* $[n,k,d]_q$ *linear codes. Then* $\mathcal{H}(\mathcal{C}_1)$ *is permutation equivalent to* $\mathcal{H}(\mathcal{C}_2)$.

Proof. Let $\sigma \in S_n$ be such that $\sigma(\mathcal{C}_1) = \mathcal{C}_2$ and denote by P the permutation matrix with respect to σ. Let G_1 be a generator matrix for $\mathcal{C}_1$ and H_1 be a parity-check matrix for $\mathcal{C}_1$.

Recall that G_1P is a generator matrix for $\mathcal{C}_2$ and H_1P is a parity-check matrix for $\mathcal{C}_2$. Finally, we have that

$$\mathcal{H}(\mathcal{C}_2) = \ker\left(\begin{pmatrix} G_2 \\ H_2 \end{pmatrix}^\top\right) = \ker\left(\begin{pmatrix} G_1P \\ H_1P \end{pmatrix}^\top\right) = \ker\left(\begin{pmatrix} G_1 \\ H_1 \end{pmatrix}^\top\right) P.$$

□

Let $\mathcal{C}, \mathcal{C}'$ be $[n, k, d]_q$ linear codes with $\langle G \rangle = \mathcal{C}, \langle G' \rangle = \mathcal{C}'$.

- The *linear isometries* in the Hamming metric are given by monomial transforms $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$.
- We say that $\mathcal{C}$ is *linearly equivalent* to $\mathcal{C}'$ if there exists $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$ such that $\varphi(\mathcal{C}) = \mathcal{C}'$.
- We say that $\mathcal{C}$ is *permutation equivalent* to $\mathcal{C}'$ if there exists $\sigma \in S_n$ such that $\sigma(\mathcal{C}) = \mathcal{C}'$.
- The *automorphism group* of $\mathcal{C}$ is given by all $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$, such that $\varphi(\mathcal{C}) = \mathcal{C}$.
- If $\mathcal{C}$ and $\mathcal{C}'$ are linearly equivalent, then $\mathcal{C}^\perp$ and $\mathcal{C}'^\perp$ are linearly equivalent.

1.3 Invariants

To determine whether two codes are equivalent is important for coding theory, especially when claiming one has found a new construction of a code. In this case, one should first check whether this new family of codes is not equivalent to an already known family.

However, determining whether two codes are equivalent or not is not an easy task - it is exactly what this lecture is about. We might instead look for *invariants*, i.e., properties of a code $\mathcal{C}$ that remain the same for $\varphi(\mathcal{C})$.

There are several parameters or properties of equivalent codes which remain invariant. Clearly, equivalent codes have the same length, dimension and minimum distance. But we can also find more interesting invariants.

Definition 1.37 (Weight Enumerator)**.** Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a linear code. For any $w \in \{1, \dots, n\}$, let us denote by $A_w(\mathcal{C}) = |\{c \in \mathcal{C} \mid \mathrm{wt}_H(c) = w\}|$ the *weight enumerator* of $\mathcal{C}$.

Proposition 1.38. *Let $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathbb{F}_q^n$ be linearly equivalent codes, then for all $w \in \{1, \dots, n\}$ we have that*

$$A_w(\mathcal{C}_1) = A_w(\mathcal{C}_2).$$

Exercise 1.39. *Prove Proposition 1.38.*

Note that the other direction is not true: We can have codes with the same weight enumerator, which are not linearly equivalent!

Example 1.40. *Let us consider $\mathbb{F}_4 = \mathbb{F}_2(\alpha)$ with $\alpha^2 = \alpha + 1$. The two codes $\mathcal{C}_1 = \langle G_1 \rangle, \mathcal{C}_2 = \langle G_2 \rangle$ with*

$$G_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & \alpha \end{pmatrix}, G_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & \alpha + 1 \end{pmatrix}$$

have the same weight enumerators. In fact, all non-zero codewords of $\mathcal{C}_1$, respectively $\mathcal{C}_2$, either have no zero, 4 zeros in $\{1, 2, 3, 4\}$, 3 zeros in $\{5, 6, 7\}$, 2 zeros in $\{8, 9\}$ or 1 zero in $\{10\}$, but no mixed zeros between these index sets. Thus,

$$\begin{aligned} A_0(\mathcal{C}_1) = A_0(\mathcal{C}_2) &= 1, \\ A_1(\mathcal{C}_i) = A_2(\mathcal{C}_i) = A_3(\mathcal{C}_i) = A_4(\mathcal{C}_i) = A_5(\mathcal{C}_i) &= 0, \\ A_6(\mathcal{C}_1) = A_6(\mathcal{C}_2) &= 3, \\ A_7(\mathcal{C}_1) = A_7(\mathcal{C}_2) &= 3, \\ A_8(\mathcal{C}_1) = A_8(\mathcal{C}_2) &= 3, \\ A_9(\mathcal{C}_1) = A_9(\mathcal{C}_2) &= 3, \\ A_{10}(\mathcal{C}_1) = A_{10}(\mathcal{C}_2) &= 3. \end{aligned}$$

However, there is no linear equivalence between $\mathcal{C}_1$ and $\mathcal{C}_2$. To see this, let us assume that there exists a $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$, which is such that $\varphi(\mathcal{C}_1) = \mathcal{C}_2$. Such φ would need to map the weight

7 codewords of $\mathcal{C}_1$ to the weight 7 codewords of $\mathcal{C}_2$, that is if $(1,1,1,1,0,0,0,1,1,\alpha) = x$ and $(1,1,1,1,0,0,0,1,1,\alpha+1) = y$, then $\varphi(x) \in \{y, \alpha y, (\alpha+1)y\}$.

If $\varphi(x) = y$, then φ would be a permutation, except for the index which gets sent to $y_{10} = \alpha+1$, and the index which sends x_{10} somewhere, i.e., if $\varphi = DP$, for $D = diag(d_1, \ldots, d_n)$, P a permutation matrix belonging to the permutation $\sigma \in S_n$, then $d_{10} \neq 1$ and $d_{\sigma^{-1}(10)} \neq 1$.

The same φ also needs to send the codewords of weight 6 to each other. Here the two sets are the same for $\mathcal{C}_1, \mathcal{C}_2$, implying that φ can only have $d_5 = \cdots = d_{10}$, a contradiction.

The cases $\varphi(x) \in \{\alpha y, (\alpha+1)y\}$ work similarly.

Another invariant are the generalized weights. For this, we need to introduce the support of a code.

Definition 1.41 (Support of a Code)**.** Let $\mathcal{C}$ be a $[n,k]_q$ linear code. The *support* of $\mathcal{C}$ is defined as

$$\text{supp}_H(\mathcal{C}) = \{i \in \{1,\ldots,n\} \mid \exists c \in \mathcal{C} : c_i \neq 0\}.$$

Clearly, for a non-degenerate code, the support will be full, i.e., $\{1,\ldots,n\}$, however, as soon as we go to subcodes of $\mathcal{C}$, this will change. Similar to how the weight of a vector is the size of its support, we may define the *weight* of a code as the size of its support.

Definition 1.42. Let $\mathcal{C}$ be an $[n,k,d]_q$ linear code. The *weight* of $\mathcal{C}$ is given by

$$\text{wt}_H(\mathcal{C}) = |\text{supp}_H(\mathcal{C})|.$$

Again, if $\mathcal{C}$ is non-degenerate then $\text{wt}_H(\mathcal{C}) = n$.

Clearly, the weight of a code is also an invariant for code equivalence: if $\mathcal{C}$ is linearly equivalent to $\mathcal{C}'$ then $\text{wt}_H(\mathcal{C}) = \text{wt}_H(\mathcal{C}')$.

We may now consider the smallest weights of any subcode:

Definition 1.43. Let $\mathcal{C}$ be an $[n,k,d]_q$ linear code and let $r \in \{1,\ldots,k\}$. The rth *generalized weight* of $\mathcal{C}$ is given by

$$d_r(\mathcal{C}) = \min\{\text{wt}_H(\mathcal{D}) \mid \mathcal{D} \subset \mathcal{C}, \dim(\mathcal{D}) = r\}.$$

If $r = 1$, we are asking for the smallest weight of any $c \in \mathcal{C}$, i.e., the first generalized weight $d_1(\mathcal{C})$ is the minimum distance $d_H(\mathcal{C})$.

On the other hand, if $r = k$, we are asking for the weight of the whole code, i.e., $d_k(\mathcal{C}) = \text{wt}_H(\mathcal{C})$.

Example 1.44. *Let $\mathcal{C} = \langle G \rangle \subset \mathbb{F}_2^4$, where*

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

Then $d_1 = d_H(\mathcal{C}) = 1$ as $\mathcal{D}_1 = \langle (0,1,0,0) \rangle$ has the smallest weight $wt_H(\mathcal{D}_1) = 1$. $d_2 = 3$ as $\mathcal{D}_2 = \langle (0,1,0,0), (0,0,1,1) \rangle$ has the smallest weight $wt_H(\mathcal{D}_2) = 3$ and finally $d_3 = wt_H(\mathcal{C}) = 4$.

Exercise 1.45. *Show that generalized weights are strictly increasing, that is for* $r \in \{1, \ldots, k-1\}$ *we have* $d_r(\mathcal{C}) < d_{r+1}(\mathcal{C})$.

Proposition 1.46. *Let* $\mathcal{C}_1, \mathcal{C}_2$ *be* $[n, k, d]_q$ *linear codes, which are linearly equivalent. For all* $r \in \{1, \ldots, k\}$ *we have that*

$$d_r(\mathcal{C}_1) = d_r(\mathcal{C}_2).$$

Exercise 1.47. *Prove Proposition 1.46.*

A last invariant we want to introduce is the size of the automorphism group.

Proposition 1.48. *Let* $\mathcal{C}_1, \mathcal{C}_2$ *be two linearly equivalent* $[n, k, d]_q$ *linear codes. Then*

$$|\mathit{Aut}(\mathcal{C}_1)| = |\mathit{Aut}(\mathcal{C}_2)|.$$

Proof. If $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$ is such that $\varphi(\mathcal{C}_1) = \mathcal{C}_2$, then for any $\psi \in \mathrm{Aut}(\mathcal{C}_1)$ we have that

$$\psi' = \varphi \circ \psi \circ \varphi^{-1} \in \mathrm{Aut}(\mathcal{C}_2).$$

□

Let $\mathcal{C}$ be an $[n, k]_q$ linear code.

- The *weight enumerator* $A_w(\mathcal{C})$ is the amount of codewords in $\mathcal{C}$ of weight w.
- The weight enumerator is invariant for equivalent codes.
- The rth *generalized weight* is $d_r(\mathcal{C}) = \min\{\mathrm{wt}_H(\mathcal{D}) \mid \mathcal{D} \subseteq \mathcal{C}, \dim(\mathcal{D}) = r\}$.
- The rth generalized weight is invariant for equivalent codes.
- The size of the automorphism group is invariant for equivalent codes.

1.4 Cryptography

As coding theory is the art of *reliable* communication, this goes hand in hand with cryptography, the art of *secure* communication. We will keep this section short, as it only serves as motivation to study the code equivalence problem, but try to (as rigorously as possible) introduce the main concepts.

In public-key cryptography, we have three main algorithms; Key-Encapsulation Mechanisms (KEMs), Public-Key Encryptions (PKEs) and signature schemes. PKEs and KEMs are directly related, as any PKE can be turned into a KEM.

PKE are very likely the kind of cryptographic protocol you are familiar with: we want to send an encrypted message such that only a legitimate receiver is able to decrypt.

We do have very famous code-based PKEs such as Classic McEliece [3], chosen to be standardized in Germany, or HQC [1], chosen to be standardized in the US by NIST.
However, our main focus will be on *signature schemes* and more precisely, on signatures derived from Zero-Knowledge (ZK) protocols.

Currently, our public-key encryption relies one the following schemes: RSA, DH, DSA, ECDH, ECDSA - that is on the hardness of integer factorization or the discrete logarithm problem.

These algorithms are quite old (at least in the cryptographic world) dating back to 1978 and will become obsolete if (or when) a capable quantum computer is available.

While the arrival of such computer is greatly discussed and uncertain, it makes sense to update the cryptographic protocols with more secure ones. This triggered the transition to *post-quantum cryptography*: cryptographic protocols, which are secure against attacks from quantum computers.

The transition is quite pressing, as already by 2035, NIST wants to "disallow" RSA. The standardization process is still ongoing for signature schemes, where we are currently in the second round, with 14 surviving schemes (6 of those are code-based).

1.4.1 Signature Schemes

In a signature scheme, we want a guarantee of the legitimate origin of the message, exactly as signing a letter to prove that the sender of this letter is really you.

In this process we speak of *authentication*, meaning that a receiver of the message can (with some probability) be sure that the sender is legit, and of *integrity*, meaning that the message has not been altered.

A digital signature scheme consists of three steps: key generation, signing and verification. We consider two parties, one is the *signer*, who wants to send and sign a message to the second party, called *verifier*, who wants to verify the signature of the message.

As a first step, the signer constructs a secret key $\mathcal{S}$, which is kept private and a public key $\mathcal{P}$, which is made public. The signer then chooses a message m, and creates a signature s using the secret key $\mathcal{S}$ and the message m, getting a signed message (m, s).

The verifier can easily read the message m, but wants to be sure that the sender is legit and the message has not been altered. Thus, the verifier uses the public key $\mathcal{P}$ and the message m to verify the validity of the signature s.

Table 1: Signature Scheme

SIGNER		VERIFIER
KEY GENERATION		
Construct a secret key $\mathcal{S}$		
Construct a connected public key $\mathcal{P}$		
	$\xrightarrow{\mathcal{P}}$	
SIGNING		
Choose a message m		
Construct a signature s from $\mathcal{S}$ and m		
	$\xrightarrow{m,s}$	
		VERIFICATION
		Verify the signature s using $\mathcal{P}$ and m

The security of a digital signature scheme introduces a new person, called *impersonator*. An impersonator, tries to cheat the verifier and acts as a signer, however without the knowledge of the secret key $\mathcal{S}$. An impersonator wins if a verifier accepts a forged signature. This comes with a certain probability, called *cheating probability*.

For the signature scheme to be secure, we want that it is *infeasible* to forge a signature or to recover the secret key from the publicly known key.

What exactly does infeasible mean, though? We speak of *computational security*, assuming that any attacker has only a limited computational power. Hence we define the *security level* to be the number of binary operations needed for an adversary to break the cryptosystem, i.e., either to forge a signature or to recover the secret key.

Usual security levels are 2^λ, with $\lambda \in \{80, 128, 256, 512\}$, meaning for example that an adversary is expected to need at least 2^{80} binary operations in order to forge a signature. These are referred to as 80 bit, 128 bit, 256 bit, or 512 bit security levels.

Hence, when proposing a signature scheme, we consider the best known attack, whether it is a forgery or a secret key recovery, and choose parameters of our scheme in such a way that the attack costs more than 2^λ.

1.4.2 ZK Protocols

There exist several approaches to construct a signature scheme, if lucky one can "invert" a PKE, like it is done for RSA. This is called the *hash-and-sign* approach. The one we are interested in, is also the most used one in the NIST standardization call: using ZK protocols.

The process and notation for ZK protocols are similar to that of a signature scheme. We have two parties, a *prover* and a verifier. Different to a digital signature scheme, the prover does not want to sign a message, but rather convince the verifier of the knowledge of a secret object, without revealing said object.

A ZK protocol consists of two stages: key generation and verification. The verification process can consist of several communication steps between the verifier and the prover. In case there are N such steps, we usually call it an N-pass scheme. For this lecture, we are interested in one particular ZK protocol, called *sigma protocol*, which is a 3-pass scheme.

1. The prover prepares two *commitments* c_0, c_1, and sends them to the verifier.
2. The verifier randomly picks a *challenge* $b \in \{0, 1\}$, and sends it to the prover.
3. The prover provides a *response* r_b that allows to verify c_b.
4. The verifier checks the validity of c_b, by recovering c_b using r_b and the public key.

Table 2: ZK Protocol

PROVER		VERIFIER
KEY GENERATION		
Construct a secret key $\mathcal{S}$		
Construct a connected public key $\mathcal{P}$		
	$\xrightarrow{\mathcal{P}}$	
	VERIFICATION	
Construct commitments c_0, c_1		
	$\xrightarrow{c_0, c_1}$	
		Choose $b \in \{0, 1\}$
	$\xleftarrow{b}$	
Construct response r_b		
	$\xrightarrow{r_b}$	
		Verify c_b using r_b

A ZK protocol has three important attributes:

1. *Zero-knowledge:* this means that no information about the secret is revealed during the process.

2. *Completeness:* meaning that an honest prover will always get accepted.

3. *Soundness:* for this, we want that an impersonator has only a small probability to get accepted.

In order to achieve that the soundness error (the probability that a dishonest prover will be accepted) is below a certain threshold, usually $2^{-\lambda}$, the protocols are often repeated several times (called *rounds*) and only if each round was verified will the prover be accepted. Thus, if the ZK protocol previously had soundness error α, after N parallel rounds we have a soundness error of α^N.

Once we have a ZK protocol, we can use the *Fiat-Shamir transform* to get a signature scheme. In this transform, instead of asking for a challenge from the verifier, the prover challenges themselves, by computing the hash of the message and the commitments. The signature then consists of the commitment and the response, as the verifier can use the message and the commitment to compute their hash and thus the challenge. The verification process then continues in the same manner as for the ZK protocol.

1.5 LESS

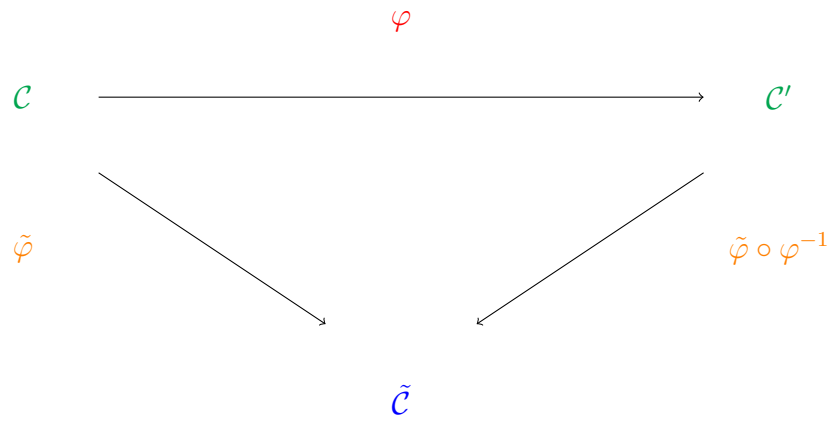
LESS [6] is one of the 14 surviving signature schemes in the NIST standardization call. It is based on the sigma protocol and uses our favorite problem: code equivalence.

The high-level idea is as follows: Choose $G \in \mathbb{F}_q^{k\times n}$ of full rank, uniformly at random (that is good for cryptography: it means we can send a seed). Also choose at random a secret monomial transform $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$, i.e., $\varphi = (D, P)$ and a matrix $S \in \mathrm{GL}_q(k)$, then compute the generator matrix of a linearly equivalent code $G' = SGDP$.

We can then use G, G' as public key and $S, \varphi = (D, P)$ as secret key. The verifier wants to check that we know the secret monomial transform, but we should reply in such a way, that we do not reveal any information on φ (other than it is a monomial transform).

This can be done by preparing as commitment another linearly equivalent code: choose $\tilde{\varphi} = (\tilde{D}, \tilde{P}) \in (\mathbb{F}_q^\star)^n \rtimes S_n$ and a $\tilde{S} \in \mathrm{GL}_q(k)$ and compute the commitment $\tilde{G} = \tilde{S}G\tilde{D}\tilde{P}$.

The verifier may now challenge the prover, and ask either for $b = 0$: revealing the transform $\tilde{\varphi}$ from G to $\tilde{G}$ or for $b = 1$: revealing the transform $\tilde{\varphi} \circ \varphi^{-1}$ from G' to $\tilde{G}$.



By revealing another monomial transform, nothing is revealed about the secret φ. If we are a honest verifier, we can also always reply with a response that gets accepted. However, if we are a dishonest verifier, not knowing the secret φ, we have quite a large cheating probability.

Proposition 1.49. *Assuming that the code equivalence problem is infeasible to solve, the cheating probability of the sigma protocol is given by* $1/2$.

Exercise 1.50. *Prove Proposition 1.49.*

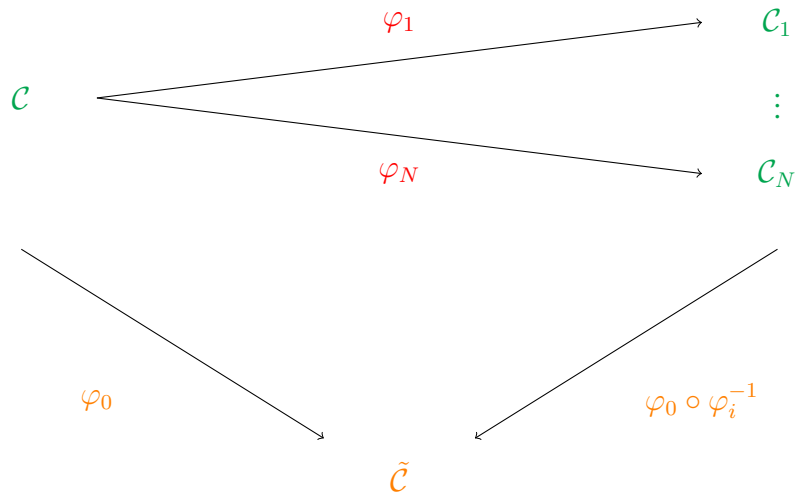
Thus, forging a signature or recovering the secret key mount both to solving the code equivalence problem.

The actual LESS signature scheme is more involved, having multiple public keys and using canonical forms to reduce signature and public key sizes.

That is: we start with a random $G \in \mathbb{F}_q^{k \times n}$, generating the code $\mathcal{C} = \langle G \rangle$. We then select randomly the secret monomials $\varphi_i = (D_i, P_i) \in (\mathbb{F}_q^\star)^n \rtimes S_n$ and the public keys are then computed as $G_i = GD_iP_i$ for all $i \in \{1, \ldots, N\}$.

We then commit to a single $\tilde{G} = GDP$, i.e., to another code $\tilde{\mathcal{C}} = \langle \tilde{G} \rangle$, which is such that $\varphi_0(\mathcal{C}) = \tilde{\mathcal{C}}$, for $\varphi_0 = (D, P)$.

The verifier then selects as challenge $i \in \{0, \ldots, N\}$ and the prover either reveals $\varphi_0 = (D, P)$ when $i = 0$, or $\varphi_0 \circ \varphi_i^{-1} = P_i^{-1} D_i^{-1} DP$.



This increases the challenge space from $\{0, 1\}$ to $\{0, \ldots, N\}$ and thus decreases the cheating probability from $1/2$ to $1/(N + 1)$.

1.6 Complexity Classes

Complexity theory tries to arrange problems in terms of how hard it is to solve them. It usually focuses on decision problems, i.e., a problem with answer "yes" or "no". We often use the complexity terms also for computational problems, as clearly, solving the computational problem would also solve the decisional version. Hence, we will not make a difference between the computational and the decisional version here.

Let $\mathcal{P}$ denote a problem. In order to estimate how hard it is to solve $\mathcal{P}$ we have two main complexity classes.

Definition 1.51. P denotes the class of problems that can be solved by a deterministic Turing machine in polynomial time.

We might think of this class, as "we can efficiently, i.e., in polynomial time, solve this problem".

Definition 1.52. NP denotes the class of problems that can be solved by a non-deterministic Turing machine in polynomial time.

Thus, in contrary to the popular belief that NP stands for non-polynomial time, it actually stands for non-deterministic polynomial time. The difference is important: all problems in P live inside NP!

Instead of going into non-deterministic Turing machines, we prefer to use the equivalent statement: a problem $\mathcal{P}$ is in NP if and only if one can check that a candidate is a solution to $\mathcal{P}$ in polynomial time.

The most important complexity class, for us, will be that of NP-hard problems. In order to define this class, we first have to define polynomial-time reductions.

A polynomial-time reduction from $\mathcal{R}$ to $\mathcal{P}$ follows the following steps:

1. take any instance I of $\mathcal{R}$,
2. transform I to an instance I' of $\mathcal{P}$ in polynomial time,
3. assume that (using an oracle) you can solve $\mathcal{P}$ in the instance I' in polynomial time, getting the solution s',
4. transform the solution s' in polynomial time to get a solution s of the problem $\mathcal{R}$ in the input I.

The existence of a polynomial-time reduction from $\mathcal{R}$ to $\mathcal{P}$, informally speaking, means that if we can solve $\mathcal{P}$, we can also solve $\mathcal{R}$ and thus solving $\mathcal{P}$ is at least as hard as solving $\mathcal{R}$.

Definition 1.53. $\mathcal{P}$ is NP-hard if for every problem $\mathcal{R}$ in NP, there exists a polynomial-time reduction from $\mathcal{R}$ to $\mathcal{P}$.

Hence, this class contains all problems which are at least as hard as the hardest problems in NP.

In order to prove that a problem $\mathcal{P}$ is NP-hard, fortunately we do not have to give a polynomial-time reduction from *every* problem in NP: there are already problems which are known to be NP-hard, thus it is enough to give a polynomial-time reduction from an NP-hard problem to $\mathcal{P}$.

Finally, NP-completeness denotes the intersection of NP-hardness and NP.

Definition 1.54. A problem $\mathcal{P}$ is NP-complete, if it is NP-hard and in NP.

This class is perfect for cryptography: it is in NP, thus we can efficiently check a solution, e.g. decrypt or verify, but to break the system one has to solve one of the hardest problems in mathematics.

1.6.1 Two Examples

Let us quickly recall the big-$\mathcal{O}$ notation, used to estimate the cost of algorithms.

Definition 1.55. Let $f(n), g(n)$ be functions over the reals. We write $f(n) \in \mathcal{O}(g(n))$ to denote that there exists a positive real constant N such that $|f(n)| < N|g(n)|$ for all $n > n_0$, meaning that if n grows, $f(n)$ will not grow faster than $g(n)$.

Example 1.56. *For example* $n + 2n^2 \in \mathcal{O}(n^2)$, *while* $2^{n/2}n^5 + n^2 2^n \in \mathcal{O}\left(2^n\right)$.

Thus, we should read it as "we ignore all lower terms".

We have several problems in NP, where we can assess their complexity even more precisely: Most famously, the Graph Isomorphism (GI) problem has lived in its own complexity class (also called GI) for several decades, until Babai [5] showed that one can solve GI in quasi-polynomial time. That is we can solve it with a cost in

$$\mathcal{O}\left(2^{\log(n)^c}\right),$$

for some constant c.

Most code-based systems base their security on the *decoding problem*, which was shown to be NP-complete [11, 9].

Problem 1.57 (Decoding Problem)**.** *Let* $G \in \mathbb{F}_q^{k\times n}, r \in \mathbb{F}_q^n$ *and* t *be a positive integer. Find a vector* $e \in \mathbb{F}_q^n$*, such that* $t = wt_H(e)$ *and there exists a* $m \in \mathbb{F}_q^k$ *with* $r = mG + e$.

The cost of solving the decoding problem (e.g. using ISD) is exponential, that is $\mathcal{O}\left(2^{nc}\right)$, for some constant c, and being an NP-hard problem, we should not get a lower cost. This makes it attractive for cryptography, as we can get reasonable sizes for a given security level 2^λ. On the other hand, the GI problem is a sub-optimal choice, as it requires much larger parameters to reach the same security level.

2 The Problem

After having seen the application of code equivalence in cryptography, and how we compute the security level of cryptosystems based on a problem, we come back to our main protagonist with the question:

How hard is code equivalence?

Recall that we have two types of equivalences, linear equivalence and permutation equivalence. Thus, we get in turn, two kinds of code equivalence problems:

Problem 2.1 (Linear Equivalence Problem (LEP))**.** *Given $\mathcal{C}, \mathcal{C}'$ two $[n,k]_q$ linear codes, find, if one exists, a monomial transform $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$, such that $\varphi(\mathcal{C}) = \mathcal{C}'$.*

And the weaker version:

Problem 2.2 (Permutation Equivalence Problem (PEP))**.** *Given $\mathcal{C}, \mathcal{C}'$ two $[n,k]_q$ linear codes, find, if one exists, a permutation $\sigma \in S_n$, such that $\sigma(\mathcal{C}) = \mathcal{C}'$.*

We have a first obvious reduction: if we can solve LEP, we can also solve PEP, thus LEP is harder than PEP.

We can use our description of code equivalence using the generator matrix to reformulate the two problems as follows:

Problem 2.3 (Linear Equivalence Problem (LEP))**.** *Given $G, G' \in \mathbb{F}_q^{k\times n}$, find, if any exists, a $S \in \mathbf{GL}_q(k), D = \mathrm{diag}(d)$, for $d \in (\mathbb{F}_q^\star)^n$ and an $n \times n$ permutation matrix P such that $SGDP = G'$.*

And the weaker version:

Problem 2.4 (Permutation Equivalence Problem (PEP))**.** *Given $G, G' \in \mathbb{F}_q^{k\times n}$, find, if any exists, a $S \in \mathbf{GL}_q(k)$ and an $n \times n$ permutation matrix P such that $SGP = G'$.*

We start with discovering in which complexity class they live.

2.1 Arthur and Merlin

The area of complexity classes is much larger than our small introduction, in this section, we will encounter something similar to a ZK protocol, which lets us determine the complexity class of code equivalence.

Contrary to what you might hope for,

all isomorphism problems

whether it is code equivalence, GI, Isomorphism of Polynomials (IP), Lattice Isomorphism Problem (LIP), isogenies between elliptic curves, or any other objects with isomorphisms between them:

are not NP-hard[1].

This kind of statement is usually possible, when we have found an algorithm solving the problem in time less than exponential, e.g., in quasi-polynomial or in polynomial time.

This is not the case here: instead we need the help of Arthur and Merlin.

The complexity class we are interested in is called AM. In this class live all problems that can be decided through an Arthur-Merlin protocol.

The protocol is similar to the ZK protocol we have seen before, with the prover "Merlin" and the verifier "Arthur". The protocol is a 3-pass protocol and does not need to have the ZK property. The main difference lies in the power of the two parties: while Arthur still has polynomial computational power, Merlin has infinite computation power (indeed Merlin is a wizard).

We say that a problem $\mathcal{P}$ can be decided by the AM protocol if Merlin is able to convince Arthur that the answer upon the instance I is "yes". Merlin might be cheating, i.e., the answer to I is actually "no", and we allow for a soundness error of $\leq 1/3$.

No NP-hard problem can live in AM, else we have AM=PH (the polynomial hierarchy) and this implies a collapse of polynomial hierarchy - which we assume does not collapse.

Theorem 2.5. *Assuming PH $\neq AM$, code-equivalence is not NP-hard.*

Proof. To show this, we construct the 3-pass Arthur-Merlin protocol - not for code equivalence, but its opposite; that is we want to show two codes are *not equivalent*.

This will then place code-equivalence in what we call co-AM.

Both parties see the instance $(\mathcal{C}_1, \mathcal{C}_2)$ and Merlin wants to convince Arthur, that the two codes are *not* equivalent.

[1] Unless of course, P=NP and the whole hierarchy collapses, which we will from now on assume is not the case.

Arthur chooses one of the codes, $\mathcal{C}_i$, a random isometry φ and computes a generator matrix G' for $\varphi(\mathcal{C}_i)$ and sends G' to Merlin.

Merlin, with the infinite computational power, can compute which code $\mathcal{C}_i$ Arthur has chosen and reply with i. If Merlin was honest, then only one of the codes $\mathcal{C}_1, \mathcal{C}_2$ will be equivalent to the sent $\mathcal{C}' = \langle G' \rangle$.

If Merlin was cheating and $\mathcal{C}_1$ is equivalent to $\mathcal{C}_2$, then Merlin has two choices and has a success probability of $1/2$.

By repeating this protocol for t rounds, we get a soundness error of 2^{-t} that Arthur accepts a cheating Merlin. □

Now we know the problem we want to solve is not one of the hardest problems, but it might still be quite hard for an algorithm to solve.

2.2 Solvers

There are two different kind of solvers, the first being an algebraic one, the second a combinatorial one.

The algebraic solver goes back to Saeed's thesis [23], where he uses that if $\mathcal{C} = \langle G \rangle = \ker(H^\top)$ and $\mathcal{C}' = \langle G' \rangle = \ker(H'^\top)$ are linearly equivalent, i.e., there exist $S \in \mathrm{GL}_q(k), D = \mathrm{diag}(d), P \in S_n$ such that $SGDP = G'$, then

$$G'H'^\top = 0 \quad \text{implies that} \quad GDPH^\top = 0.$$

Hence it is enough to solve the linear system $GMH^\top = 0$, with $k(n-k)$ equations and where the entries of M are the n^2 unknowns. The main problem is: how to put the fact that we are only interested in $M = DP$ into this model as equations?

As it happens so often in code-based cryptography, the algebraic solvers are not as efficient as the combinatorial solvers (meaning they have a much larger cost).

The combinatorial solvers date back to Leon [18], and have been improved by Sendrier [25], Beullens [12] and recently in [8]. The combinatorial solvers are split into two directions: the first one is called *codeword-search*, whereas the second one relies on canonical forms [15, 21]. In the latter, it was observed that it is enough to know the action of the monomial transform on an information set to recover the whole transform. This did not only speed up the solvers, but also allowed them to reduce the signature and public key sizes of LESS.

The idea of the codeword-search solvers is to find subsets of codewords $S \subset \mathcal{C}$ and $S' \subset \mathcal{C}'$ which are also invariant under the isometry φ, i.e., $\varphi(S) = S'$. For the smaller sets S, S' it will become easier to find an isometry between them. The most preferred way of constructing such subsets, is to use invariants, such as the weight enumerator, or in case of permutation equivalence the hull. However, to find a subset S consisting of small weight codewords, these solvers rely on Information Set Decoders (ISD). Thus, they all come with a cost in $\mathcal{O}(2^{nc})$. Note that this is, however, solving a much harder problem: the problem of finding low weight codewords (equivalent to the decoding problem), which is known to be NP-hard.

As our main goal is not to improve upon the constant c in the solvers, but to break code equivalence completely (i.e., show it is (quasi-) polynomial time), we will only cover a short overview of the ideas of the solvers.

Algorithm	Problem	Invariant	Cost
Leon	LEP	weight enumerator	exponential in n (bottleneck: ISD)
Beullens	LEP	second generalized weight	exponential in n (bottleneck: ISD)
SSA	PEP	(punctured) hull	exponential in dimension of the hull

Table 3: Summary of the existing solvers

As a very short summary: Leon is using weight enumerator, Beullens is using the second generalized weight (which is indeed a more refined invariant) and SSA is using that the hull is an invariant for permutation equivalence.

Leon's Algorithm for LEP The main observation of Leon [18], is that the sought isometry has to map all codewords of weight w in $\mathcal{C}$ to all codewords of weight w in $\mathcal{C}'$. Thus, Leon first constructs the sets

$$S = \{c \in \mathcal{C} \mid \mathrm{wt}_H(c) = w\},$$
$$S' = \{c \in \mathcal{C} \mid \mathrm{wt}_H(c) = w\}$$

and then searches for $\varphi \in (\mathbb{F}_q^\star)^n \rtimes S_n$ such that $\varphi(S) = S'$.

The cost of finding such a map φ is polynomial in the size of S. However, to construct S, we rely on ISD.

Beullens' Algorithm for PEP The main observation of Beullens [12], is that a permutation σ does not only fix the weight of a vector, but also the multiset of its entries. Thus, instead of seeing the elements in S and S' as vectors c, Beullens' algorithm treats them as multisets and searches for a collision in $S \times S'$, i.e., $c \in S$ has the same multiset as $c' \in S'$. This allows us to store less elements in S.

Each found collision is then used to piece-wise reconstruct the permutation: if c, c' have the same multisets and $c_i \neq c'_j$, then we guess $\sigma(i) \neq j$. Again, the bottleneck remains constructing the sets S, S' using ISD.

Beullens' Algorithm for LEP Beullens also provides an algorithm to solve LEP. In the case of linear equivalence, the multiset of the entries are clearly not preserved. However, for any subcode $\mathcal{D} < \mathcal{C}$ there exists a $\varphi(\mathcal{D}) = \mathcal{D}' < \mathcal{C}'$ with the same dimension and support size. In particular, Beullens proposes to use subcodes of dimension 2, that is: Beullens uses as invariant the second generalized weight. The lists now contain generator matrices in $\mathbb{F}_q^{2\times n}$:

$$S = \{D \in \mathbb{F}_q^{2\times n} \mid \langle D\rangle < \mathcal{C}, |\mathrm{supp}(\langle D\rangle)| = s\},$$
$$S' = \{D' \in \mathbb{F}_q^{2\times n} \mid \langle D'\rangle < \mathcal{C}', |\mathrm{supp}(\langle D'\rangle)| = s\}.$$

One then again searches for collisions, i.e., an isometry $\varphi(\langle D\rangle) = \langle D'\rangle$. In order to find such isometry, one can either use Leon's algorithm or first Beullens' algorithm to find the permutation and then reconstruct the scalar factors.

This algorithm has also been improved in [8], modifying the subroutine to find small support subcodes, while relying on the same post-processing to recover the secret monomial.

Support Splitting for PEP A bit a special case is the Support Splitting Algorithm (SSA) from Sendrier [25]: it requires a "signature" function, i.e., a property for each position of the code which is invariant under the permutation. More precisely, for a given code $\mathcal{C}$ and position $i \in \{1, \ldots, n\}$, one wants $S(\mathcal{C}, i) = S(\sigma(\mathcal{C}), \sigma(i))$.

Sendrier chooses this signature function (or rather the invariant of the code) to be the weight enumerator of the hull (of a punctured code), i.e.,

$$S(\mathcal{C}, i) = A(\mathcal{H}(\mathcal{C}_i)),$$

where $A(\mathcal{C}) = (A_0(\mathcal{C}), \ldots, A_n(\mathcal{C}))$ are all the weight enumerators, $\mathcal{C}_i$ is the code $\mathcal{C}$ punctured in the position i.

Having defined such a signature function, one can now compare $S(\mathcal{C}, i)$ with $S(\mathcal{C}', i)$ for all $i \in \{1, \ldots, n\}$ and if a match is found, recover the permutation between the original codes.

The hull, however, is only an invariant for permutation equivalence. Hence, the SSA has a cost of solving PEP in

$$\mathcal{O}(q^{\dim(\mathcal{H}(\mathcal{C}))}).$$

Thus, if the code is self-orthogonal, the SSA algorithm has an exponential cost, while if the code has a trivial hull (as we expect random codes to have), we get a *polynomial-time* solver.

The attentive reader has already realized a big spoiler now: PEP can be solved (with high probability) in polynomial time for random codes! We come back to this in the next chapter.

2.3 Different View Points

What if we consider the many connections of coding theory to other finite friends in order to solve this problem?

2.3.1 Projective Systems

Our first finite friend is finite geometry. I suspect you to be more familiar with this topic than I am, but I will do my best to recap the main notions correctly:

Definition 2.6. Let q be a prime power and k be a positive integer. The *finite projective geometry* of dimension k and order q is then given by

$$\text{PG}(k,q) = (\mathbb{F}_q^{k+1} \setminus \{0\})/\sim,$$

where $\sim$ denotes the following equivalence relation on $\mathbb{F}_q^{k+1} \setminus \{0\}$:

$$u \sim v \quad \text{if and only if} \quad u = \lambda v \quad \text{for some } \lambda \in \mathbb{F}_q^\star.$$

We call elements in $\text{PG}(k,q)$ points and say $H \subset \text{PG}(k,q)$ is a *hyperplane* if it is isomorphic to $\text{PG}(k-1,q)$.

This allows us to see codes as *projective systems:*

Definition 2.7. Let q be a prime power and $k \leq n$ be positive integers. We say that $\mathcal{M}$ is a *projective* $[n,k,d]_q$ *system*, if $\mathcal{M}$ is a finite set of n points (counted with multiplicity) of $\text{PG}(k-1,q)$, which do not all lie on a hyperplane. Further,

$$d = n - \max\{|H \cap \mathcal{M}| \ \mid \ H \subseteq \text{PG}(k-1,q), \dim(H) = k-2\}.$$

Let $\mathcal{C}$ be an $[n,k,d]_q$ linear non-degenerate code with generator matrix $G \in \mathbb{F}_q^{k\times n}$. If we consider the (multi-) set of one dimensional subspaces of $\mathbb{F}_q^n$ generated by the columns of G, then these form a multiset $\mathcal{M}$ of points in $\text{PG}(k-1,q)$.

Additionally, for any vector $v \in \mathbb{F}_q^k$, the projective hyperplane

$$\sum_{i=1}^{k} v_i x_i = 0$$

contains $|\mathcal{M}| - w$ points of $\mathcal{M}$, if and only if $\text{wt}_H(vG) = w$.

Hence, we have a one-to-one correspondence between $[n,k,d]_q$ linear non-degenerate codes and projective $[n,k,d]_q$ systems.

Example 2.8. *Let us consider*

$$G = \begin{pmatrix} 1 & 0 & 2 & 3 \\ 0 & 1 & 4 & 0 \end{pmatrix}$$

over $\mathbb{F}_5$. Then the rows span $\mathcal{C} = \langle G \rangle$ is a $[4, 2]_5$ linear code. We can easily check that its minimum distance $d = 2$, as $c = (0, 1, 4, 0) \in \mathcal{C}$. On the other hand, the columns form points in the projective line PG$(1, 5)$, being

$$\mathcal{M} = \{[0 : 1], [1 : 0], [3 : 1], [1 : 0]\}.$$

Thus, we want to find a hyperplane

$$H = \{[x : y] \in \textbf{PG}(1, 5) \mid ax + by = 0\},$$

containing the most points of $\mathcal{M}$. The maximal amount of points we can contain is 2, being $[1 : 0]$, counted with multiplicity two and thus $H = \{[x : y] \mid y = 0\}$, giving us

$$d = n - \max\{|H \cap \mathcal{M}| \ \mid \ H \subseteq \textbf{PG}(k - 1, q), \textbf{dim}(H) = k - 2\} = 4 - 2 = 2.$$

We quickly observe that any two linearly equivalent codes define the *same* projective system.

In fact, by permuting the columns of G, we do not change the multiset $\mathcal{M}$, and neither by multiplying a column of G with a non-zero scalar. Indeed, the projective system is another invariant.

2.3.2 Matroids

Our next finite friend is matroid theory.

Definition 2.9. A *matroid* M is a pair (E, I), where E is a finite set and I is a collection of subsets of E, called *independent sets*, such that

1. $\emptyset \in I$,
2. if $A \in I, B \subseteq A$ then $B \in I$,
3. if $a, B \in I$ with $|A| < |B|$, then there exists a $b \in B \setminus A$ such that $A \cup \{b\} \in I$.

Starting from a generator matrix $G \in \mathbb{F}_q^{k \times n}$ we can get a *representable matroid* $M(G)$ by setting $E = \{1, \ldots, n\}$ and

$$I = \{S \subset E \mid G_S \text{ has full rank}\}.$$

Example 2.10. *In our previous example,*

$$G = \begin{pmatrix} 1 & 0 & 2 & 3 \\ 0 & 1 & 4 & 0 \end{pmatrix}$$

over $\mathbb{F}_5$, we set $E = \{1, 2, 3, 4\}$ and get the following independent sets:

$$I = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1, 2, 3\}, \{2, 3, 4\}\}.$$

Equivalently, one can define a matroid using the rank function. For this we denote by $\mathcal{P}(E)$ the powerset of E, i.e., all subsets of E.

Definition 2.11. A *matroid* is a pair (E, r), where E is a finite set and $r : \mathcal{P}(E) \to \mathbb{N}_0$ is a *rank function*, such that

1. $0 \leq r(X) \leq |X|$ for all $X \subseteq E$,
2. if $X \subseteq Y \subseteq E$ then $r(X) \leq r(Y)$,
3. for all $X, Y \subseteq E$ we have

$$r(X \cup Y) + r(X \cap Y) \leq r(X) + r(Y).$$

With this the definition, a representable matroid $M(G)$ becomes: $E = \{1, \ldots, n\}$ and for all $S \in \mathcal{P}(E)$ we set

$$r(S) = \dim(\langle G_S \rangle).$$

Example 2.12. *In our favorite example, we get*

$$\begin{aligned} r(\emptyset) &= 0 \\ r(\{i\}) &= 1 \text{ for all } i \in E, \\ r(\{1,4\}) &= 1 \end{aligned}$$

and all other subsets have rank 2 (as this is already the dimension of the code itself).

Note that the minimum distance $d = d_H(\langle G \rangle)$ can then also be determined as

$$d = \min\{|S| \mid r(E \setminus S) < k\},$$

that is the smallest number of columns of G, such that removing these columns reduces the rank.

In our example, we get again $d = 2$, since we need to remove the columns indexed by $S = \{2, 3\}$ to get $G_{S^C} = \begin{pmatrix} 1 & 3 \\ 0 & 0 \end{pmatrix}$ of rank 1.

If $\mathcal{C}, \mathcal{C}'$ are two linearly equivalent $[n, k]_q$ codes, then the linear dependency relations among columns of G, G' are the same: the permutation of columns only introduces a relabeling of E and the scalar multiplications do not change the linear dependencies.

Thus, they define the same representable matroid- yet another invariant.

2.3.3 Designs

We can also go to the finite friend of design theory.

Definition 2.13. A $t - (v, k, \lambda)$ *design* is a pair (X, B), where X is a set of v points and B is a collection of k-elements subsets of X, called *blocks*, such that every t-element subset of X is contained in exactly λ blocks.

Given a $[n, k, d]_q$ linear code $\mathcal{C}$, we may define $X = \{1, \ldots, n\}$ and B being the support of all codewords of weight d.

The connection between codes and designs is established through the Assmus-Mattson theorem [4], which states the following:

Theorem 2.14. *Let $\mathcal{C}$ be a $[n, k, d]_q$ linear code and $\mathcal{C}^\perp$ be a $[n, n-k, d']_q$ linear code. Let us denote by A_i the weight enumerators of $\mathcal{C}$ and by A'_i the weight enumerators of $\mathcal{C}^\perp$, for $i \in \{0, \ldots, n\}$. Fix a positive integer $t < d$ and denote by s the number of $i \in \{1, \ldots, n-t\}$ with $A'_i \neq 0$. If $s \leq d - t$, then the supports of all codewords in $\mathcal{C}$ of any fixed weight $d \leq u \leq n$ form a t-design.*

Example 2.15. *Our running example is unfortunately* not *projective, so we get a degenerate design, but we can still do it: recall that*

$$G = \begin{pmatrix} 1 & 0 & 2 & 3 \\ 0 & 1 & 4 & 0 \end{pmatrix},$$

then the minimum weight codewords are given by $(0, 1, 4, 0), (0, 2, 3, 0), (0, 3, 2, 0), (0, 4, 1, 0)$, and all of them have support $\{2, 3\}$. Thus, we set $X = \{1, 2, 3, 4\}$ and $B = \{\{2, 3\}\}$. We can easily check that this is a $1 - (4, 2, 1)$ design.

By considering two linearly equivalent codes $\mathcal{C}, \mathcal{C}'$, the permutation will only introduce a relabeling of X such that the support of their minimum weight codewords is the same. Thus, again the design from minimum weight codewords is ultimately an invariant.

There is one final finite friend, which will be helpful in solving code equivalence: graph theory. This will be the topic of the next chapter.

3 Connections to other Problems

In this chapter we want to explore the connections of LEP and PEP to other problems, hoping they might be easier to solve.

To this extend, we will show two main reductions:

1. The reduction from LEP to PEP.
2. The reduction from PEP to GI.

We then ask ourselves; shouldn't this also allow us to reduce LEP to GI? And recalling that GI is quasi-polynomial time: haven't we achieved our goal then?

The answer is: unfortunately no (to both), as we hid some details under the rug. Thus, as last point in this chapter, we will lift the rug and discover the unfortunate truth.

3.1 Reduction from LEP to PEP or How to Close a Code

The reduction from PEP to LEP is straightforward: if we have a solver for LEP, we can also give this solver any instance of PEP and get a solution. The more interesting direction is to go from LEP to PEP instead.

This has been done in [26], by defining the *closure* of a code.

Definition 3.1. Let $\mathcal{C}$ be an $[n,k]_q$ linear code, let $\alpha \in \mathbb{F}_q$ be a primitive element and denote by $\lambda = (1, \alpha, \ldots, \alpha^{q-2}) \in \mathbb{F}_q^{q-1}$. The *closure* of $\mathcal{C}$ is given by the Kronecker product $\lambda \otimes \mathcal{C}$.

The new code is now of length $n(q-1)$ and still of dimension k. In fact, if G is a generator matrix of $\mathcal{C}$, then $\lambda \otimes G$ is a generator matrix of $\lambda \otimes \mathcal{C}$.

Example 3.2. *Let us consider the code over $\mathbb{F}_5$ generated by*

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 3 \end{pmatrix}$$

and take $\alpha = 2$, that is $\lambda = (1,2,4,3)$, then its closure $\lambda \otimes \mathcal{C}$ is generated by

$$\lambda \otimes G = \begin{pmatrix} 1 & 2 & 4 & 3 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 3 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 3 & 2 & 4 & 3 & 1 \end{pmatrix}.$$

Clearly, the closure depends on the choice of λ, however, for a different λ' we simply get a permutation equivalent code.

Example 3.3. *Let us consider again the code over $\mathbb{F}_5$ generated by*

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 3 \end{pmatrix}$$

and take a different $\alpha' = 3$, giving $\lambda' = (1, 3, 4, 2)$, then its closure $\lambda' \otimes \mathcal{C}$ is generated by

$$\lambda' \otimes G = \begin{pmatrix} 1 & 3 & 4 & 2 & 0 & 0 & 0 & 0 & 1 & 3 & 4 & 2 \\ 0 & 0 & 0 & 0 & 1 & 3 & 4 & 2 & 2 & 1 & 3 & 4 \end{pmatrix}.$$

Clearly, we have $\lambda \otimes G = (\lambda' \otimes G)P$, where P is the permutation matrix corresponding to the permutation $(2, 4)(6, 8)(10, 12)$ or equivalently $Id_4 \otimes (1, 2)$.

We could extend the definition of closure to allow any λ which consists of all non-zero elements of $\mathbb{F}_q$, but this definition will be enough for us.

Proposition 3.4. *Let $\mathcal{C}_1, \mathcal{C}_2$ be two linearly equivalent $[n, k]_q$ linear codes. Then $\lambda \otimes \mathcal{C}_1$ is permutation equivalent to $\lambda \otimes \mathcal{C}_2$.*

Proof. Let $\varphi = DP$, with $D = \text{diag}(d_1, \ldots, d_n)$ and P a $n \times n$ permutation matrix, such that $\varphi(\mathcal{C}_1) = \mathcal{C}_2$.

If $G_1 = \begin{pmatrix} g_1^\top & \cdots & g_n^\top \end{pmatrix}$ is a generator matrix for $\mathcal{C}_1$ then

$$\lambda \otimes G_1 = \begin{pmatrix} g_1^\top & \alpha g_1^\top & \cdots & \alpha^{q-2} g_1^\top & \cdots & g_n^\top & \alpha g_n^\top & \cdots & \alpha^{q-2} g_n^\top \end{pmatrix}$$

is a generator matrix of $\lambda \otimes \mathcal{C}_1$.

We note that multiplying with d_i is a permutation in $\mathbb{F}_q^\star$, that is $\sigma_i : \mathbb{F}_q^\star \to \mathbb{F}_q^\star, x \mapsto x d_i$ can be seen as $\sigma_i \in S_{q-1}$. Thus, multiplying column g_i with d_i, means $(d_i g_i^\top, d_i \alpha g_i^\top, \ldots, d_i \alpha^{q-2} g_i^\top) = \sigma(g_i^\top, \alpha g_i^\top, \ldots, \alpha^{q-2} g_i^\top)$. Hence the scalars d_i are introducing permutations σ_i within the n blocks of length m.

The permutation P instead shifts around these blocks, that is if σ is the permutation corresponding to P and σ sends the index i to j, then we have to send the ith block to the jth block.

Thus,

$$Q = \begin{pmatrix} P_1 & & \\ & \ddots & \\ & & P_n \end{pmatrix} (\text{Id}_{q-1} \otimes P) \in S_{n(q-1)}$$

is such that $(\lambda \otimes G_1)Q$ is a generator matrix of $\lambda \otimes \mathcal{C}_2$. □

This is a nice result, allowing us to focus on PEP more.

3.2 Reduction from PEP to GI

Due to Babai's algorithm [5], we know that Graph Isomorphism (GI) takes at most quasi-polynomial time to solve. Thus, a reduction from PEP to GI, i.e., showing that if we can solve GI then we can also solve PEP, implies that PEP is easier than GI. In particular, PEP should not be used for cryptography.

The reduction has been proposed in [7] and has a small drawback: it only works for codes with trivial hull. Since random codes have with high probability a trivial hull, we call this a "randomized" reduction, meaning that it will not work for any instance, but it works with high probability.

Before we can give the reduction, let us recall some graph theory.

A graph $\mathcal{G} = (V, E)$ consists of vertices V and edges E between the vertices, i.e., $E \subseteq V \times V$.

We will focus on undirected graphs, thus whenever $\{u, v\} \in E$ also $\{v, u\} \in E$ and we label the edges with a weight $w(u, v)$.

We say that two weighted graphs $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V', E')$ are isomorphic, if there exists a bijective map $f : V \to V'$ with

1. $\{u, v\} \in E \leftrightarrow \{f(u), f(v)\} \in E'$,
2. $w(u, v) = w(f(u), f(v))$.

Thus, we can focus on $V = V' = \{1, \ldots, n\}$ and maps $f = \sigma \in \mathcal{S}_n$.

Problem 3.5 (Weighted Graph Isomorphism Problem)**.** *Given $\mathcal{G} = (V, E), \mathcal{G}' = (V, E')$, find $\sigma \in \mathcal{S}_n$, such that $\{u, v\} \in E \leftrightarrow \{\sigma(u), \sigma(v)\} \in E'$ and $w(u, v) = w(\sigma(u), \sigma(v))$.*

The quasi-polynomial time solver also works for weighted graphs, as it was shown that the weight graphs problem is GI-complete.

Definition 3.6. The *adjacency matrix* of a weighted graph $\mathcal{G}$ is defined as the $n \times n$ matrix A with entries

$$A_{i,j} = \begin{cases} w(i, j) & \text{if } \{i, j\} \in E, \\ 0 & \text{else.} \end{cases}$$

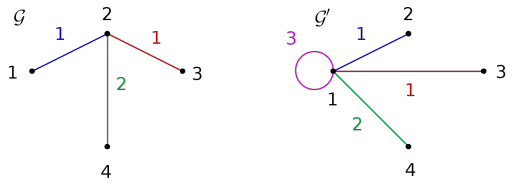
Since we are only interested in undirected graphs, the adjacency matrices are symmetric.

Proposition 3.7. *Two graphs $\mathcal{G}, \mathcal{G}'$ are isomorphic if and only if there exists a permutation matrix P such that $P^\top AP = A'$.*

This almost looks like what we need for PEP, except for the fact that in PEP (treating A as generator matrix) we also accept $SAP = A'$ for any invertible matrix S, not necessarily of the form $P^\top$.

In fact, one can easily make an example of two graphs, where there exists $S \in \mathrm{GL}_q(n), P \in S_n$ with $SAP = A'$ but the two graphs are clearly not isomorphic.

Example 3.8. *Let $\mathcal{G} = (V, E)$ with $V = \{1, 2, 3, 4\}$ and $E = \{w(1, 2) = 1, w(2, 3) = 1, w(2, 4) = 2\}$ and $\mathcal{G}' = (V, E')$ with $E' = \{w(1, 1) = 3, w(1, 2) = 1, w(1, 3) = 1, w(1, 4) = 2\}$.*



These graphs can clearly not be isomorphic. However, their adjacency matrices

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{pmatrix} \text{ and } A' = \begin{pmatrix} 3 & 1 & 1 & 2 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{pmatrix}$$

generate the codes $\mathcal{C} = \langle G \rangle$ with $G = \begin{pmatrix} 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 \end{pmatrix}$ and $\mathcal{C}' = \langle G' \rangle$ with $G' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 \end{pmatrix}$, which are clearly permutation equivalent through the permutation $\sigma = (2134)$. And there exists $S \in GL_5(4), P \in S_4$ (the permutation matrix of σ) such that $SAP = A'$, namely

$$S = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 \end{pmatrix}.$$

Luckily, in order to reduce PEP to GI, we do not have to start with an instance of GI and transform it to an instance of PEP. Instead, we start from codes and transform them to graphs.

Hence, the main question is: given $G \in \mathbb{F}_q^{k \times n}$, how to define a symmetric matrix in $\mathbb{F}_q^{n \times n}$, which can act as adjacency matrix?

For this [7] introduced the following: For $\mathcal{C}$ with trivial hull, we define

$$A = G^\top (GG^\top)^{-1} G.$$

Clearly, this matrix can only exist if $GG^\top$ is invertible, i.e., if the hull of $\mathcal{C}$ is trivial. The matrix $A \in \mathbb{F}_q^{n \times n}$ is symmetric, $\langle A \rangle = \mathcal{C}$ and, moreover, independent of the choice of G.

In fact, taking any other generator matrix, SG, we get

$$(SG)^\top (SG(SG)^\top)^{-1} SG = G^\top (GG^\top) G.$$

Theorem 3.9. *PEP is easier than weighted GI, for codes with trivial hull.*

Proof. Assume that the codes in the instance of PEP $(\mathcal{C}, \mathcal{C}')$ have trivial hulls. For arbitrary generator matrices G, respectively G', take

$$A = G^\top (GG^\top)^{-1} G,$$
$$A' = G'^\top (G'G'^\top)^{-1} G'.$$

And define $\mathcal{G}$, respectively $\mathcal{G}'$, to have adjacency matrices A, respectively A'.

We now show that the answer to the constructed weighted GI instance is also the answer to the PEP instance. In fact, $\sigma(\mathcal{G}) = \mathcal{G}'$ if and only if $\sigma(\mathcal{C}) = \mathcal{C}'$.

For the first direction, note that for any choice of generator matrices, there exist $S \in \mathrm{GL}_q(k)$ with $SGP = G'$. However, since A is independent of the choice of basis, we can ignore the S and get

$$A' = (GP)^\top (GP(GP)^\top)^{-1} GP = P^\top AP.$$

Thus, the two graphs are isomorphic.

The other direction is straightforward, as the $P^\top AP = A'$ implies that the two codes are permutation equivalent. □

3.2.1 Reduction from GI to PEP

Note, that we can also go the other direction (which is less interesting as it says PEP is harder than GI), but it does have a positive effect on subcode equivalence.

We say that $\mathcal{G} = (V, E)$ with $|V| = v, |E| = e$ has *incidence matrix* $B \in \mathbb{F}_2^{e \times v}$, if B has entries $b_{i,j}$ with

$$b_{i,j} = \begin{cases} 1 & \text{if } i = \{\ell, j\} \in E, \\ 0 & \text{else.} \end{cases}$$

That is the rows correspond to the edges and the columns to the vertices. Considering the edge $\{u, v\}$, we set a 1 in the position u and in the position v.

Example 3.10. *The graph $\mathcal{G}$ with vertex set $V = \{1, 2, 3, 4\}$ and edge set $E = \{\{(1, 2\}, \{2, 3\}, \{3, 4\}\}$ has incidence matrix*

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

Clearly, there are different incidence matrices, depending on the ordering of the edges.

Similar to the characterization with the adjacency matrix, two graphs $\mathcal{G}, \mathcal{G}'$ with incidence matrices B, respectively B', are isomorphic, if there exist a $e \times e$ permutation matrix Q and a $v \times v$ permutation matrix P, such that $QBP = B'$.

Theorem 3.11. *There exists a polynomial-time reduction from GI to PEP.*

We follow the proof of [22].

Proof. Let $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V, E')$ be an instance of GI. Let B and B' be two incidence matrices for $\mathcal{G}$, respectively $\mathcal{G}'$. We can transform this instance to an instance of PEP, by defining the two generator matrices in $\mathbb{F}_q^{e\times(3e+v)}$

$$G = \begin{pmatrix} \mathrm{Id}_e & \mathrm{Id}_e & \mathrm{Id}_e & B \end{pmatrix},$$
$$G' = \begin{pmatrix} \mathrm{Id}_e & \mathrm{Id}_e & \mathrm{Id}_e & B' \end{pmatrix}.$$

Let us consider two cases. In the first case, the answer to GI is "yes", as there exists a $f : V \to V$, such that $\{f(u), f(v)\} \in E'$ for all $\{u, v\} \in E$. Thus, there exists a permutation of V which maps one graph to the other and the two incidence matrices B and B' are such that

$$QBP = B'$$

for some $e \times e$ permutation matrix Q and $v \times v$ permutation matrix P. Clearly, the codes generated by G and G' are then also permutation equivalent.

In the second case, we assume that the two graphs are not isomorphic, hence there exists no permutation on V, which maps $\mathcal{G}$ to $\mathcal{G}'$. Thus, no $v \times v$ permutation matrix P and no $e \times e$ permutation matrix Q exists for which $QBP = B'$.

The two codes generated by G and G' are only permutation equivalent, if we can find $S \in \mathrm{GL}_2(e)$ and $(3e + v) \times (3e + v)$ permutation matrix P such that

$$SGP = \begin{pmatrix} S & S & S & SB \end{pmatrix} P = G'.$$

Both generator matrices have the following properties: each row has weight 5 and any linear combination of them has weight ≥ 6. (Actually ≥ 8, but I guess the original proof considered directed graphs).

Since G' and GP generate the same code, the two matrices must be the same up to permutations of the rows. Hence S must be a permutation matrix.

Note that the first $3e$ columns of SG consist of all unit vectors of length e, each appearing exactly three times. Hence, the first $3e$ columns of G' are obtained by permuting the first $3e$ columns of SG and thus, we also have the permutation matrix $P' = \mathrm{diag}(S^{-1}, S^{-1}, S^{-1}, T)$, where T is a $v \times v$ permutation matrix, which is such that

$$G' = SGP' = \begin{pmatrix} \mathrm{Id}_e & \mathrm{Id}_e & \mathrm{Id}_e & SBT \end{pmatrix}.$$

Hence, we must have $B' = SBT$, which is against the assumption that $\mathcal{G}$ and $\mathcal{G}'$ are not isomorphic. □

Due to this result, we know that PEP (and thus also LEP) are at least as hard as GI.

This has the nice implication, that the *subcode equivalence problem* is NP-hard, as the subgraph isomorphism problem is NP-hard [16]!

Problem 3.12 (Subcode Equivalence Problem (SEP))**.** *Let $k' \leq k \leq n$ be positive integers. Given $G \in \mathbb{F}_q^{k \times n}, G' \in \mathbb{F}_q^{k' \times n}$, find, if any exists, permutation matrix P such that $\langle G' \rangle \subseteq \langle GP \rangle$.*

3.3 Under the Rug

Let us come back to our question: If LEP $\to$ PEP $\to$ GI, then LEP $\to$ GI and LEP is quasi-polynomial time -right?

Indeed, we can solve PEP using Babai's quasi-polynomial time solver for GI [5] *if* the codes have trivial hulls.

When first reducing linearly equivalent codes to permutation equivalent codes, we cannot reduce them further to GI, as the closure of codes is in fact self-orthogonal for $q \geq 4$.

Before we can prove this, we need the following Lemma.

Lemma 3.13. *Let q be a prime power, then*

$$\sum_{\alpha \in \mathbb{F}_q^\star} \alpha^\ell = \begin{cases} 0 & \text{if } (q-1) \nmid \ell, \\ -1 & \text{if } (q-1) \mid \ell. \end{cases}$$

Exercise 3.14. *Prove Lemma 3.13.*

Proposition 3.15. *If $q \geq 4$, then $\lambda \otimes \mathcal{C}$ is self-orthogonal.*

Proof. Recall that a code $\mathcal{D}$ is self-orthogonal if $\mathcal{D} \subseteq \mathcal{D}^\perp$, thus $\mathcal{H}(\mathcal{D}) = \mathcal{D}$. We have also seen that if G is a generator matrix of $\mathcal{D}$, then $GG^\top = 0$ implies that $\mathcal{D}$ is self-orthogonal.

To understand the hull of the closure, we thus have to compute

$$(\lambda \otimes G)(\lambda \otimes G)^\top = \begin{pmatrix} g_1^\top & \alpha g_1^\top & \cdots & \alpha^{q-2} g_n^\top \end{pmatrix} \begin{pmatrix} g_1 \\ \alpha g_1 \\ \vdots \\ \alpha^{q-2} g_n \end{pmatrix}.$$

One can easily check that

$$(\lambda \otimes G)(\lambda \otimes G)^\top = \lambda\lambda^\top GG^\top.$$

While we assumed that for random G we have that $GG^\top$ is full rank, we also have that $\lambda\lambda^\top = 0$, as

$$\lambda\lambda^\top = \sum_{i=0}^{q-2} \alpha^{2i} = \sum_{\beta \in \mathbb{F}_q^\star} \beta^2 = 0,$$

by Lemma 3.13, unless $q = 2, 3$.

□

Example 3.16. *Let us consider* $\mathbb{F}_4 = \mathbb{F}_2(\alpha)$, *where* $\alpha^2 = \alpha + 1$. *Let* $\mathcal{C} = \langle G \rangle \subset \mathbb{F}_4^3$, *where*

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

Let $D = diag(\alpha, 1, \alpha)$ *and* P *be the permutation matrix corresponding to* $\sigma = (1, 3)$, *then*

$$GDP = \begin{pmatrix} \alpha + 1 & 0 & \alpha \\ \alpha & 1 & 0 \end{pmatrix}.$$

Bringing this into systematic form, we get a linearly equivalent code $\mathcal{C}'$ *generated by*

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

Let $\lambda = (1, \alpha, \alpha + 1)$. *The closure of* $\mathcal{C}$ *is generated by*

$$\lambda \otimes G = \begin{pmatrix} 1 & \alpha & \alpha+1 & 0 & 0 & 0 & \alpha & \alpha+1 & 1 \\ 0 & 0 & 0 & 1 & \alpha & \alpha+1 & 1 & \alpha & \alpha+1 \end{pmatrix},$$

while the closure of $\mathcal{C}'$ *is generated by*

$$\lambda \otimes G' = \begin{pmatrix} 1 & \alpha & \alpha+1 & 0 & 0 & 0 & \alpha+1 & 1 & \alpha \\ 0 & 0 & 0 & 1 & \alpha & \alpha+1 & 1 & \alpha & \alpha+1 \end{pmatrix}.$$

We can find

$$Q = \begin{pmatrix} 0 & 0 & 1 & & & & & & \\ 1 & 0 & 0 & & 0 & & & 0 & \\ 0 & 1 & 0 & & & & & & \\ & & & 1 & 0 & 0 & & & \\ & 0 & & 0 & 1 & 0 & & 0 & \\ & & & 0 & 0 & 1 & & & \\ & & & & & & 0 & 0 & 1 \\ & 0 & & & 0 & & 1 & 0 & 0 \\ & & & & & & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & Id_3 \\ 0 & Id_3 & 0 \\ Id_3 & 0 & 0 \end{pmatrix} \in S_9$$

which is such that $S(\lambda \otimes G)Q = \lambda \otimes G'$, *for some* $S \in GL_q(k)$.

3.4 A bit of Hope

We have seen the limitations of the reductions - but this is not the end of the story! No one said, we have to use this construction of an adjacency matrix, or this construction of a closure!

In fact, we can also handle $q = 4$, however, not with the classical dual defined through the standard inner product.

Definition 3.17 (Hermitian Inner Product)**.** Let $q = p^{2m}$, for some prime p and positive integer m. For $x, y \in \mathbb{F}_q^n$ let us denote by $\langle x, y\rangle_H$ the *Hermitian* inner product, i.e.,

$$\langle x, y\rangle_H = \sum_{i=1}^{n} x_i y_i^{p^m}.$$

Note that the Hermitian inner product is positive definite, linear but is not symmetric! Instead we have $\langle x, y\rangle_H = (\langle y, x\rangle_H)^{p^m}$.

Thus, to define the Hermitian dual, we have to fix on which side we place the codewords.

Definition 3.18 (Hermitian Dual Code)**.** Let $k \leq n$ be positive integers and let $\mathcal{C}$ be an $[n, k]_q$ linear code over $\mathbb{F}_q$. The *Hermitian dual code* $\mathcal{C}^\star$ is an $[n, n-k]$ linear code over $\mathbb{F}_q$, defined as

$$\mathcal{C}^\star = \{x \in \mathbb{F}_q^n \mid \langle x, y\rangle_H = 0 \; \forall \; y \in \mathcal{C}\}.$$

Note that since $p = \text{char}(\mathbb{F}_q)$, we get

$$\langle x, y\rangle_H = \sum_{i=1}^{n} x_i y_i^{p^m} = (\sum_{i=1}^{n} x_i^{p^m} y_i)^{p^m}. \tag{1}$$

For a matrix $M \in \mathbb{F}_q^{m\times n}$ and a positive integer x, let us denote by M^x the matrix, where each entry of M is raised to the power x.

Definition 3.19 (Hermitian Parity-Check Matrix)**.** Let $k \leq n$ be positive integers and let $\mathcal{C}$ be an $[n, k]_q$ linear code over $\mathbb{F}_q$ with Hermitian dual code $\mathcal{C}^\star$. Then, a matrix $H^\star \in \mathbb{F}_q^{(n-k)\times n}$ is called a *Hermitian parity-check matrix* of $\mathcal{C}$ if $H^\star$ is a generator matrix of $\mathcal{C}^\star$.

Exercise 3.20. *Let $\mathcal{C}$ be an $[n, k]_q$ linear code and $G \in \mathbb{F}_q^{k\times n}$ be a generator matrix and $H^\star \in \mathbb{F}_q^{(n-k)\times n}$ be a Hermitian parity-check matrix of $\mathcal{C}$. Show that*

$$H^\star (G^{p^m})^\top = 0.$$

That is $\mathcal{C}^\star = ker((G^{p^m})^\top)$.

Exercise 3.21. *Let $\mathcal{C}$ be an $[n, k]_q$ linear code and $H \in \mathbb{F}_q^{(n-k)\times n}$ be a parity-check matrix of $\mathcal{C}$. Use Equation* (1) *to show that $H^\star = H^{p^m}$ is a Hermitian parity-check matrix.*

We want to translate the following description of the dual to the Hermitian dual:

$$\mathcal{C} = \langle G \rangle = \ker(H^\top), \qquad \mathcal{C}^\perp = \langle H \rangle = \ker(G^\top), \qquad \text{and} \quad (\mathcal{C}^\perp)^\perp = \mathcal{C}.$$

Exercise 3.22. *Let $\mathcal{C}$ be a $[n,k]_q$ linear code with $\mathcal{C} = \langle G \rangle = ker(H^\top)$. Show that $(\mathcal{C}^\star)^\star = \mathcal{C}$.*

Thus, we get

$$\mathcal{C} = \langle G \rangle = \ker(H^\top), \qquad \mathcal{C}^\star = \langle H^{p^m} \rangle = \ker((G^{p^m})^\top), \qquad \text{and} \quad (\mathcal{C}^\star)^\star = \mathcal{C}.$$

Let us see an example of this new dual.

Example 3.23. *Let us consider $\mathbb{F}_4 = \mathbb{F}_2(\alpha)$ with $\alpha^2 = \alpha + 1$ and a generator matrix of $\mathcal{C}$ is given by*

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

A normal parity-check matrix would be

$$H = \begin{pmatrix} \alpha & \alpha+1 & 1 \end{pmatrix},$$

as it is such that $GH^\top = 0$. To get to a Hermitian parity-check matrix, we can compute $H^\star = H^{1/2}$, that gives

$$H^\star = \begin{pmatrix} \alpha+1 & \alpha & 1 \end{pmatrix}.$$

This generates the Hermitian dual

$$\mathcal{C}^\star = \{(0,0,0), (\alpha+1, \alpha, 1), (1, \alpha+1, \alpha), (\alpha, 1, \alpha+1)\}.$$

If we compute the Hermitian parity-check matrix of that, we can start with a normal parity-check matrix

$$G^\star = \begin{pmatrix} 1 & 0 & \alpha+1 \\ 0 & 1 & \alpha \end{pmatrix}$$

and compute

$$(G^\star)^2 = \begin{pmatrix} 1 & 0 & \alpha \\ 0 & 1 & \alpha+1 \end{pmatrix} = G.$$

The *Hermitian hull* is then defined as $\mathcal{H}^\star(\mathcal{C}) = \mathcal{C} \cap \mathcal{C}^\star$.

Exercise 3.24. *Let q be a prime power and $k \leq n$ be positive integers. Let $\mathcal{C}$ be a $[n,k]_q$ linear code with generator matrix $G \in \mathbb{F}_q^{k \times n}$ and parity-check matrix $H \in \mathbb{F}_q^{(n-k) \times n}$. Show that*

$$\mathcal{H}^\star(\mathcal{C}) = ker\left(\begin{pmatrix} G^{p^m} \\ H \end{pmatrix}^\top \right).$$

The Hermitian dual and Hermitian hull are still invariants.

Exercise 3.25. *Let $\mathcal{C} \subset \mathbb{F}_q^n$ be linearly equivalent to $\mathcal{C}'$. Show that $\mathcal{C}^\star$ is linearly equivalent to $(\mathcal{C}')^\star$.* Hint: *Use again that $G((H^\star)^{p^m})^\top = 0$ and $GDP = G'$.*

Exercise 3.26. *Let $\mathcal{C} \subset \mathbb{F}_q^n$ be permutation equivalent to $\mathcal{C}'$. Show that $\mathcal{H}^\star(\mathcal{C})$ is permutation equivalent to $\mathcal{H}^\star(\mathcal{C}')$.*

Having this new definition of hull, we can define

$$A^\star = (G^{p^m})^\top (G(G^{p^m})^\top)^{-1} G,$$

again $A^\star$ is independent of the choice of generator matrix, symmetric and exists if $\mathcal{C}$ has trivial Hermitian hull.

Exercise 3.27. *Show that $A^\star$ is independent on the choice of G.*

Similar to before, we can assume that random codes have with high probability trivial Hermitian hull.

Exercise 3.28. *Let q be a prime power and $k \leq n$ be positive integers. Let $\mathcal{C}$ be a $[n,k]_q$ linear code with generator matrix $G \in \mathbb{F}_q^{k\times n}$. Show that if $G(G^{p^m})^\top$ has full rank, then dim$(\mathcal{H}^\star(\mathcal{C})) = 0$.*

The only thing left to check is that the closure of a code in $\mathbb{F}_4$ has with high probability trivial Hermitian hull. For this let α be a primitive element in $\mathbb{F}_4$ and consider $\lambda = (1, \alpha, \alpha^2)$, thus we get

$$X = (\lambda \otimes G)((\lambda \otimes G)^2)^\top = \begin{pmatrix} | & | & | & & | & | & | \\ g_1^\top & \alpha g_1^\top & \alpha^2 g_1^\top & \cdots & g_n^\top & \alpha g_n^\top & \alpha^2 g_n^\top \\ | & | & | & & | & | & | \end{pmatrix} \begin{pmatrix} - & g_1^2 & - \\ - & \alpha^2 g_1^2 & - \\ - & \alpha g_1^2 & - \\ & \vdots & \\ - & g_n^2 & - \\ - & \alpha^2 g_n^2 & - \\ - & \alpha g_n^2 & - \end{pmatrix}.$$

One can easily check that

$$X = \lambda(\lambda^2)^\top G(G^2)^\top.$$

For $\lambda = (1, \alpha, \alpha + 1)$, we now get

$$\lambda(\lambda^2)^\top = 1 \cdot 1 + \alpha \cdot (\alpha + 1) + (\alpha + 1) \cdot \alpha = -1.$$

Hence, under our assumption that $G(G^2)^\top$ has full rank, the adjacency matrix $A^\star$ exists and we can give the same reduction to GI as before.

The immediate question we have in mind now is: why not do this for large $\mathbb{F}_{p^m}$?

Let's try: if $\lambda = (1, \alpha, \ldots, \alpha^{p^{2m}-2})$ then we are considering

$$\begin{aligned}\lambda(\lambda^{p^m})^\top &= (1, \alpha, \ldots, \alpha^{p^{2m}-2})(1, \alpha^{p^m}, \ldots, \alpha^{(p^{2m}-2)p^m})^\top \\ &= \sum_{i=0}^{p^{2m}-2} \alpha^i \cdot \alpha^{ip^m} = \sum_{i=0}^{p^{2m}-2} \alpha^{i(p^m+1)} \\ &= \sum_{\beta \in \mathbb{F}_q^\star} \beta^{p^m+1}.\end{aligned}$$

Recall from Lemma 3.13 that in order to get $\neq 0$, we need $(p^{2m} - 1) \mid (p^m + 1)$. However, this is only given for $p = 2$ and $m = 1$, that is for $\mathbb{F}_4$.

4 New Directions

As the topic is very young, we of course have some new results:

4.1 New Results

Two is all it takes: We have seen that we can set up a linear system $GMH^\top = 0$, where M are the unknowns, and we are only interested in the unique solution M of the form DP. The system is clearly under-determined, as there is no easy way to incorporate "M is a monomial matrix" to the system.

On the other hand, the combinatorial solvers are trying to find pairs of codewords $(c, c') \in \mathcal{C} \times \mathcal{C}'$ which are mapped to each other through the secret monomial DP. In these solvers, one needed roughly $\log(n)$ such pairs of codewords to recover the monomial.

A very old result from basic linear algebra has been resurrected to lower this amount to only 2. The paper "Two is all it takes" [14], uses the *Rouché-Capelli test*.

Theorem 4.1. *Let $A \in \mathbb{F}_q^{m \times n}$ be a matrix of rank rk$(A) = r$. Let $b \in \mathbb{F}_q^m$ and $x = (x_1, \ldots, x_n)$ be unknown variables. The system $Ax^\top = b^\top$ admits a solution if and only if $\begin{pmatrix} A & b \end{pmatrix}$ has rank r.*

If we are given (say by an oracle) two pairs $(c_1, c_1'), (c_2, c_2') \in \mathcal{C} \times \mathcal{C}'$ which are such that $c_i DP = c_i'$ for $i \in \{1, 2\}$, then these codewords leak already a lot of information on the monomial DP.

For example, if c_i has in the first entry a zero, this might only be mapped to a zero entry of c_i'. Hence, we know where the first position is *not* mapped to. This allows us to set some entries of DP. Similarly, if the secret monomial sends the ith entry to the jth entry and multiplies with $\lambda \in \mathbb{F}_q^\star$, then $c_{1,i} = \lambda c_{1,j}'$ and $c_{2,i} = \lambda c_{2,j}$, thus their ratios must be the same:

$$\frac{c_{1,i}}{c_{2,i}} = \frac{c_{1,j}'}{c_{2,j}'}.$$

We can use these *hints* on the entries of the secret $M = DP$ in the algebraic modeling $GMH^\top = 0$, however, we are still left with a under-determined system. Thus, we require to make certain guesses on the entries of M.

Whenever a guess is made, we can check if the system still admits a solution with the Rouché Capelli test. Clearly, if it *does not* admit a solution anymore is much stronger: then we know the guess was wrong.

Clearly, the main problem is now to find two pairs $(c_1, c_1'), (c_2, c_2') \in \mathcal{C} \times \mathcal{C}'$ which are mapped through the secret DP.

Don't use it twice: Recall that LESS uses multiple public keys; $G, G_1, \ldots, G_N$, and they are all linearly equivalent to each other, but using *different* monomial transformations.

In the paper "Don't use it twice" [13], the authors show that if we are given two pairs of codes connected through the *same* secret monomial, that is

$$\varphi(\mathcal{C}_1) = \mathcal{C}'_1 \quad \text{and} \quad \varphi(\mathcal{C}_2) = \mathcal{C}'_2$$

then we can find the secret monomial φ easily; again using Rouché Capelli.

4.2 Square Codes

The *square code* is usually used in cryptanalysis of structured systems, i.e., to distinguish between algebraically structured codes, such as Reed-Solomon codes, from random codes. However, it will also come in handy for code equivalence.

Definition 4.2. Let $x, y \in \mathbb{F}_q^n$. Let us denote by $*$ the componentwise product or *Schur product* between x and y, i.e.,

$$x * y = (x_1 y_1, \ldots, x_n y_n).$$

The Schur product has many nice properties, in particular, it is symmetric and bilinear. That means

1. For $\lambda \in \mathbb{F}_q$ we have $(\lambda x) * y = x * (\lambda y) = \lambda(x * y)$.
2. For $z \in \mathbb{F}_q^n$ we have $(x + z) * y = x * y + z * y$ and $x * (y + z) = x * y + x * z$.
3. We have $x * y = x * y$.

We can thus also consider this operation on codes, leading to

Definition 4.3. Let $\mathcal{C}_1$ be an $[n, k_1]_q$ linear code and $\mathcal{C}_2$ be an $[n, k_2]_q$ linear code. The *Schur product code* of $\mathcal{C}_1$ and $\mathcal{C}_2$ is defined as

$$\mathcal{C}_1 * \mathcal{C}_2 = \langle \{c_1 * c_2 \mid c_1 \in \mathcal{C}_1, c_2 \in \mathcal{C}_2\} \rangle.$$

If we apply this to the same code, we get the *square code* of $\mathcal{C}$, defined as

$$\mathcal{C}^{(2)} = \langle \{c * c' \mid c, c' \in \mathcal{C}\} \rangle.$$

Example 4.4. *Let us consider the* $[3, 2]_3$ *linear code* $\mathcal{C}$ *generated by* $G = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{pmatrix}$. *That is*

$$\mathcal{C} = \{(0,0,0), (1,0,2), (2,0,1), (0,1,1), (1,1,0), (2,1,2), (0,2,2), (1,2,1), (2,2,0)\}.$$

Then the square code is given by

$$\mathcal{C}^{(2)} = \mathbb{F}_3^3.$$

Proposition 4.5. *Let* $\mathcal{C}$ *be generated by* $G = \begin{pmatrix} g_1 \\ \vdots \\ g_k \end{pmatrix} \in \mathbb{F}_q^{k \times n}$. *Then* $\mathcal{C}^{(2)}$ *is generated by*

$$G^{(2)} = \begin{pmatrix} g_1 * g_1 \\ \vdots \\ g_1 * g_k \\ \vdots \\ g_k * g_k \end{pmatrix} \in \mathbb{F}_q^{\binom{k+1}{2} \times n}.$$

Exercise 4.6. *Prove Proposition 4.5.*

With this in mind, we have a clear bound on the dimension of the square code:

Theorem 4.7. *Let $\mathcal{C}$ be a $[n,k]_q$ linear code. Then*

$$\dim(\mathcal{C}^{(2)}) \leq \min\left\{\binom{k+1}{2}, n\right\}.$$

We again know, that random codes attain this bound with high probability, for n growing.

If we start with two linearly equivalent codes, their square codes will also be linearly equivalent.

Theorem 4.8. *Let $\mathcal{C}, \mathcal{C}'$ be two $[n,k]_q$ linear codes and $\varphi = (D,P) \in (\mathbb{F}_q^\star)^n \rtimes S_n$ be such that $\varphi(\mathcal{C}) = \mathcal{C}'$. Then $\varphi' = (D^2, P) \in (\mathbb{F}_q^\star)^n \rtimes S_n$ is such that*

$$\varphi'(\mathcal{C}^{(2)}) = \mathcal{C}'^{(2)}.$$

Exercise 4.9. *Prove Theorem 4.8.*

If we consider the hardest instance of PEP, that is $\mathcal{C}$ is self-orthogonal, i.e., $\mathcal{C} \subseteq \mathcal{C}^\perp$, does this imply that $\mathcal{C}^{(2)}$ is also self-orthogonal?

Not necessarily!

Exercise 4.10. *Let $\mathcal{C}$ be a $[n,k]_q$ linear code. Show that $\mathcal{H}(\mathcal{C})^{(2)} \neq \mathcal{H}(\mathcal{C}^{(2)})$.*

However, we can use the square of the hull to solve PEP faster: for this consider two linearly equivalent codes $\mathcal{C}, \mathcal{C}'$, which are self-orthogonal. The idea of [10], is to first puncture $\mathcal{C}$ and $\mathcal{C}'$ in the positions I, respectively I', denoted by $\mathcal{P}_I(\mathcal{C})$ and $\mathcal{P}_{I'}(\mathcal{C}')$, and then to compute the square codes of their hulls. These are again permutation equivalent.

$$\begin{array}{ccc}
\mathcal{C} & \xrightarrow{\varphi} & \mathcal{C}' \\
\downarrow & & \downarrow \\
\mathcal{P}_I(\mathcal{C}) & & \mathcal{P}_{I'}(\mathcal{C}') \\
\downarrow & & \downarrow \\
\mathcal{H}(\mathcal{P}_I(\mathcal{C})) & & \mathcal{H}(\mathcal{P}_{I'}(\mathcal{C}')) \\
\downarrow & & \downarrow \\
\mathcal{H}(\mathcal{P}_I(\mathcal{C}))^{(2)} & \xrightarrow{\varphi'} & \mathcal{H}(\mathcal{P}_{I'}(\mathcal{C}'))^{(2)}
\end{array}$$

Using this idea, the paper [10] recently attacked the proposal [2].

The case $q = 5$ In fact, the square code can now also handle LEP for $q = 5$.

Let $\mathcal{C}, \mathcal{C}'$ be two $[n, k]_q$ linear codes and $\varphi = (D, P) \in (\mathbb{F}_q^\star)^n \rtimes S_n$ be such that $\varphi(\mathcal{C}) = \mathcal{C}'$. By Theorem 4.8 we get that $\varphi' = (D^2, P)$ is such that $\varphi'(\mathcal{C}^{(2)}) = \mathcal{C}'^{(2)}$.

However, for $q = 5$, D^2 has only entries of $\{\pm 1\}$ on the diagonal. Thus, if we compute the adjacency matrices A, A', we get

$$\begin{aligned} A &= {G^{(2)}}^\top (G^{(2)} {G^{(2)}}^\top)^{-1} G^{(2)}, \\ A' &= P^\top D^2 {G^{(2)}}^\top (G^{(2)} D^2 P P^\top D^2 {G^{(2)}}^\top)^{-1} G^{(2)} D^2 P \\ &= P^\top A P. \end{aligned}$$

Since square codes of random codes have with high probability a trivial hull, $G^{(2)} {G^{(2)}}^\top$ is with high probability invertible.

Exercise 4.11. *Perform the reduction for the code generated by*

$$G = \begin{pmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 3 & 0 \end{pmatrix} \in \mathbb{F}_5^{2\times 4}$$

and the linearly equivalent code generated by

$$G' = \begin{pmatrix} 4 & 1 & 0 & 2 \\ 0 & 4 & 2 & 0 \end{pmatrix}.$$

4.3 Power Codes

We can also go to larger powers, $\mathcal{C}^{(\ell)}$. The dimension of a ℓth power code of a random code is now a bit more complicated, but has been shown [19] to be with high probability

$$\dim(\mathcal{C}^{(\ell)}) = \min\left\{\binom{k+\ell-1}{\ell}, n\right\},$$

for $\ell \in \{0, \ldots, q\}$.

If we use the power codes in combination with the closure, we first note that these operations are not commutative:

Exercise 4.12. *Let $\mathcal{C}$ be a $[n,k]_q$ linear code and α be a primitive element in $\mathbb{F}_q$. Define $\lambda = (1, \alpha, \ldots, \alpha^{q-2})$. Show that*

$$(\lambda \otimes \mathcal{C})^{(2)} \neq \lambda \otimes \mathcal{C}^{(2)}.$$

If we first compute the ℓth power code and then the closure, i.e., $\lambda \otimes \mathcal{C}^{(2)}$, then we get again a self-orthogonal code and cannot build the adjacency matrix.

Instead, we start by first computing the closure and then form the ℓth power code.

Exercise 4.13. *Let $G \in \mathbb{F}_q^{k\times n}$ and $\lambda = (1, \alpha, \ldots, \alpha^{q-2})$. Show that*

$$(\lambda \otimes G)^{(\ell)} = \lambda^{\ell} \otimes G^{(\ell)}.$$

Hence,

$$(\lambda \otimes G)^{(\ell)}(\lambda \otimes G)^{(\ell)^\top} = \lambda^{\ell}\lambda^{\ell^\top} \otimes G^{(\ell)}G^{(\ell)^\top}.$$

Thus a first approach would be to use the power $(q-1)/2$, as then

$$\lambda^{(q-1)/2}\lambda^{(q-1)/2^\top} \neq 0,$$

due to Lemma 3.13.

The main problem of this idea is that by first computing the closure of a code, and then the $(q-1)/2$-th power code, we get that the dimension quickly reaches the length n, that is

$$\binom{k+(q-1)/2-1}{(q-1)/2} \geq n.$$

Only in the unlikely case that $n > \binom{k+(q-1)/2-1}{(q-1)/2}$ can the $((q-1)/2)$-th power code approach be applied.

4.4 More Philosophy than Math

Why are we choosing A or $A^\star$ in this way? If we take a closer look at the adjacency matrix A for the generator matrix G, let us call it $A(G, G^\top)$ for now:

$$A(G, G^\top) = (G^\top)(G(G^\top))^{-1}G,$$

we can see the involvement of G, as a generator of $\mathcal{C}$, and the involvement of $G^\top$ as the parity-check matrix of $\mathcal{C}^\top$. Equivalently, we could state that the inner part $GG^\top$ is for the hull of $\mathcal{C}$.

We are using the fact, that if $\varphi \in S_n$ is such that $\varphi(\mathcal{C}) = \mathcal{C}'$, then $\varphi(\mathcal{C}^\perp) = \mathcal{C}'^\perp$.

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{\varphi} & \mathcal{C}' \\ \downarrow & & \downarrow \\ \mathcal{C}^\perp & \xrightarrow{\varphi} & \mathcal{C}'^\perp \end{array}$$

However, this reduction does not work for LEP, as

$$\begin{aligned} A(G, G^\top) &= (G^\top)(GG^\top)^{-1}G, \\ A(GDP, (GDP)^\top) &= P^\top DG^\top (GD^2G^\top)^{-1}GDP \\ &\neq P^\top A(G, G^\top)P, \end{aligned}$$

since if $\varphi = (D, P)$ is such that $\varphi(\mathcal{C}) = \mathcal{C}'$, we do not have that $\varphi(\mathcal{C}^\perp) = \mathcal{C}'^\perp$, instead, the duals are connected through a different monomial $\varphi' = (D^{-1}, P)$.

One idea is to try and find another construction $F(\mathcal{C})$ starting from $\mathcal{C}$, such that for $\varphi = (D, P)$ we do get

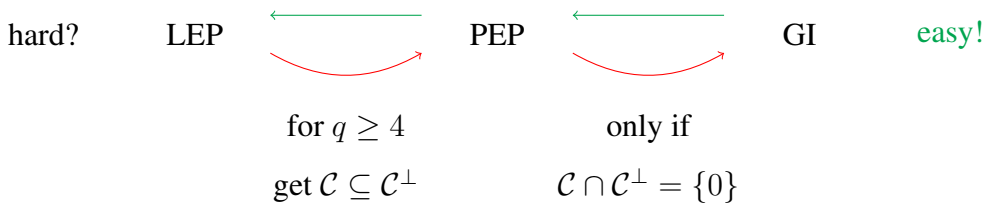
$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{\varphi} & \mathcal{C}' \\ \downarrow & & \downarrow \\ F(\mathcal{C}) & \xrightarrow{\varphi} & F(\mathcal{C}') \end{array}$$

5 Summary and Open Questions

We have seen that the code equivalence problem is much more involved than previously thought. What we do know:

- We differentiate between linear equivalence (LEP) and permutation equivalence (PEP).
- If two codes are linearly equivalent, their duals are as well.
- Only for permutation equivalence are the duals connected through the same permutation.
- There are several invariants, such as weight enumerators and generalized weights.
- The hull is only an invariant for permutation equivalence.
- Hulls of random codes are with high probability trivial.
- LEP and PEP are not NP-hard, they are in co-AM $\cap$ NP.
- There exist several solvers which make use of these invariants but have exponential cost.
- We can clearly reduce PEP to LEP and we can reduce GI to PEP.
- We can reduce LEP to PEP using the closure.
- The closure of a code is a self-orthogonal code for $q \geq 4$.
- We can reduce PEP to GI, if the codes have trivial hulls.

Let us depict the reductions:



But there is much more that we do *not* know:

- How to find a code $F(\mathcal{C})$ from $\mathcal{C}$ which is connected through the same monomial?
- Can we change the definition of the adjacency matrix to make it work also for LEP?
- Can we ultimately reduce LEP to GI?

The intend of these lectures was to show you certain tools and connections we have explored, but more importantly, show how much we have *not explored* yet.

If you are intrigued with the problem now, I hope these notes will help you and that in the near future we might finally solve the mystery: *how hard is code equivalence really?*

References

[1] C. Aguilar Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J. Bos, J.-C. Deneuville, A. Dion, P. Gaborit, J. Lacan, E. Persichetti, J.-M. Robert, P. Véron, and G. Zémor. Hamming Quasi-Cyclic (HQC). *Selected for Standardization by NIST*, 2025.

[2] M. R. Albrecht, B. Benčina, and R. W. Lai. Hollow LWE: A new spin: Unbounded updatable encryption from LWE and PCE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 363–392. Springer, 2025.

[3] M. R. Albrecht, D. J. Bernstein, T. Chou, C. Cid, J. Gilcher, T. Lange, V. Maram, I. von Maurich, R. Misoczki, R. Niederhagen, K. G. Paterson, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, C. J. Tjhai, M. Tomlinson, and W. Wang. Classic McEliece: Conservative Code-Based Cryptography. *NIST PQC Call for Proposals*, 2022. Round 4 Submission.

[4] E. Assmus Jr and H. Mattson Jr. New 5-designs. *Journal of Combinatorial Theory*, 6(2):122–151, 1969.

[5] L. Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697, 2016.

[6] M. Baldi, A. Barenghi, L. Beckwith, J.-F. Biasse, A. Esser, K. Gaj, K. Mohajerani, G. Pelosi, E. Persichetti, M.-J. O. Saarinen, P. Santini, and R. Wallace. LESS. In *Second Round Submission to the additional NIST Postquantum Cryptography Call*, 2025.

[7] M. Bardet, A. Otmani, and M. Saeed-Taha. Permutation code equivalence is not harder than graph isomorphism when hulls are trivial. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2464–2468. IEEE, 2019.

[8] A. Barenghi, J.-F. Biasse, E. Persichetti, and P. Santini. On the computational hardness of the code equivalence problem in cryptography. *Cryptology ePrint Archive*, 2022.

[9] S. Barg. Some new NP-complete coding problems. *Problemy Peredachi Informatsii*, 30(3):23–28, 1994.

[10] M. Battagliola, R. Mora, and P. Santini. Using the Schur product to solve the code equivalence problem. *Cryptology ePrint Archive*, 2025.

[11] E. Berlekamp, R. McEliece, and H. Van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[12] W. Beullens. Not enough LESS: an improved algorithm for solving code equivalence problems over $\mathbb{F}_q$. In *International Conference on Selected Areas in Cryptography*, pages 387–403. Springer, 2020.

[13] A. Budroni, J.-J. Chi-Domínguez, G. D'Alconzo, A. J. Di Scala, and M. Kulkarni. Don't use it twice! solving relaxed linear equivalence problems. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 35–65. Springer, 2024.

[14] A. Budroni, A. Esser, E. Franch, and A. Natale. Two is all it takes: Asymptotic and concrete improvements for solving code equivalence. *Cryptology ePrint Archive*, 2025.

[15] T. Chou, E. Persichetti, and P. Santini. On linear equivalence, canonical forms, and digital signatures. *Designs, Codes and Cryptography*, pages 1–43, 2025.

[16] S. A. Cook. The complexity of theorem-proving procedures. In *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*, pages 143–152. 2023.

[17] H. Lefmann, K. T. Phelps, and V. Rödl. Rigid linear binary codes. *Journal of Combinatorial Theory, Series A*, 63(1):110–128, 1993.

[18] J. Leon. Computing automorphism groups of error-correcting codes. *IEEE Transactions on Information Theory*, 28(3):496–511, 1982.

[19] C. Lin and G. Zémor. Kneser's theorem for codes and ℓ-divisible set families. *arXiv preprint arXiv:2504.19304*, 2025.

[20] F. J. MacWilliams. *Combinatorial problems of elementary abelian groups*. PhD thesis, 1962.

[21] J. Nowakowski. An improved algorithm for code equivalence. In *International Conference on Post-Quantum Cryptography*, pages 71–103. Springer, 2025.

[22] E. Petrank and R. M. Roth. Is code equivalence easy to decide? *IEEE Transactions on Information Theory*, 43(5):1602–1604, 2002.

[23] M. A. Saeed. *Algebraic approach for code equivalence*. PhD thesis, Normandie Université; University of Khartoum, 2017.

[24] N. Sendrier. On the dimension of the hull. *SIAM Journal on Discrete Mathematics*, 10(2):282–293, 1997.

[25] N. Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2002.

[26] N. Sendrier and D. E. Simos. How easy is code equivalence over $\mathbb{F}_q$? In *International Workshop on Coding and Cryptography-WCC 2013*, 2013.