

On the Hardness of Finding Permutation Automorphisms of Linear Codes

Michele Battagliola¹, Giuseppe D’Alconzo², Laura Mattiuz³,
Paolo Santini³, and Federica Zanetti^{4,5}

¹University of St.Gallen, St.Gallen, Switzerland

²Politecnico di Torino, Turin, Italy

³Università Politecnica delle Marche, Ancona, Italy

⁴University of Trento, Trento, Italy

⁵Center for Cybersecurity, Fondazione Bruno Kessler, Trento, Italy

battagliola.michele@proton.me, giuseppe.dalconzo@polito.it,
l.mattiuz@pm.univpm.it, p.santini@univpm.it, fzanetti@fbk.eu

Abstract. We study the hardness of finding permutation automorphisms of linear codes. We look into three main problems: AUT, AUT-Gen and AUT-Count which, respectively, ask to find a non-trivial automorphism, a generating set and the order of the automorphism group.

We give polynomial time reductions from AUT to ISO (the problem of finding a permutation isomorphism between two linear codes), and from ISO to AUT-Gen and AUT-Count. This shows that AUT-Gen and AUT-Count are unlikely to be easy in the worst case, since all known solvers for ISO take exponential time if the hull dimension (the subspace given by the intersection between the code and its dual) is large enough. Furthermore, we give a reduction from AUT-Gen to the problem of finding generators for the automorphism group of a graph, which we refer to as GraphAUT-Gen. Up to some polynomial factors, the reduction runs in time $O(n^h)$, with n being the code length and h the hull dimension: for $h = O(\log(n))$, this implies that AUT-Gen can be solved in worst case quasi-polynomial time.

We also give a simpler version of the reduction, running in time which is exponential in the worst case (arguably, only for rather contrived codes), even for small h . However, heuristically, the algorithm is on average extremely efficient for small h . We use it as a concrete solver and show that a proof-of-concept implementation is significantly faster than the built-in implementations of Leon’s and Feulner’s algorithms provided by Magma and Sagemath.

1 Introduction

An *automorphism* of a linear code is a transformation that stabilizes the code, i.e., maps the code into itself. In this paper, we restrict our attention to permutations¹: given a linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$, a permutation $\varphi \in S_n$ is an automorphism

¹ There are more general notions of automorphisms, such as monomial and semi-linear transformations, which we do not investigate.

of $\mathcal{C}$ if and only if

$$\varphi(\mathcal{C}) = \{\varphi(\mathbf{c}) \mid \mathbf{c} \in \mathcal{C}\} = \mathcal{C}.$$

Representing a code with a generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$, we have that $\varphi \in S_n$ can be an automorphism if and only if there exists a non singular matrix $\mathbf{S} \in GL_k(\mathbb{F}_q)$ such that $\mathbf{G} = \mathbf{S}\varphi(\mathbf{G})$, where $\varphi(\mathbf{G})$ is the matrix obtained by permuting the columns of $\mathbf{G}$ according to φ . Analogous considerations hold if one represents the code through a parity-check matrix.

The set of all permutation automorphisms of a given code $\mathcal{C} \subseteq \mathbb{F}_q^n$ forms its permutation automorphism group, that is,

$$\text{P-Aut}(\mathcal{C}) = \{\varphi \in S_n \mid \varphi(\mathcal{C}) = \mathcal{C}\}.$$

From now on, we will frequently just speak of automorphisms, implicitly referring to permutation automorphisms.

For any code, $\text{P-Aut}(\mathcal{C})$ is a subgroup of the symmetric group. Moreover, for any linear code, the automorphism group coincides with that of its dual. If the automorphism group contains only the identity, then we say that it is trivial.

The automorphism group of linear codes is a topic that has always fascinated coding theorists. There is an extensive literature about algebraic and geometric properties of codes having automorphism groups with desired properties [BC93; Bou00; BMW10; CT13; JMX20; LCE+24]. Certain decoding algorithms, known as permutation decoding [Mac64], rely on the availability of a large automorphism group [KMM10; PS21; Pra62]. Analogously, the knowledge about automorphisms can be used to speed-up Information Set Decoding (ISD), which is the standard attack against many code-based post-quantum cryptosystems [CT19; Sen11]². Moreover, automorphisms are important also for quantum codes, since they can be exploited to boost fault tolerance [GR13; Sha08].

In cryptography, the security in the Quantum Random Oracle Model of group actions-based digital signatures is linked to the problem of finding non-trivial automorphisms as proven in [BCD+24]. In the linear code equivalence scenario, like the LESS scheme [BBB+24], this means finding non-trivial automorphisms of linear codes. Furthermore, the security of the linkable ring signature from [BDE+26] relies on the problem of finding an automorphism of a given code which does not fix a second code [BDE+26, Sect.5].

Computational hardness At the best of our knowledge, the computational hardness of the problem of finding automorphisms has not received so much attention. We are only aware about two main algorithms, due to Leon [Leo82] and Feulner [Feu09], which can be deemed as the state-of-the-art. As a matter of fact, implementations of both Feulner’s and Leon’s algorithms are provided as built-in functions in programming languages such as **Sagemath** and **Magma**.

² While the need for automorphisms is explicit in [CT19], this may not be obvious, at a first glance, from [Sen11]. Yet, it is not hard to see that knowledge about non-trivial automorphisms is a manner to create multiple instances of the decoding problem, which is the starting assumption in [Sen11].

Both algorithms determine generators for the automorphism group of any given linear code and, actually, work also for more general notions of automorphisms (e.g., monomials instead of permutations). However, they come with a significant computational cost and are, in practice, unusable for codes with sufficiently large dimension. For instance, Leon’s algorithm requires to find all codewords with weight not exceeding some threshold value which, as a rule of thumb, can be set equal to (or slightly larger than) the minimum distance of the code [BBP+23; Beu20]. However, finding low weight codewords in a given linear code is a well known hard problem and all known solvers take average exponential time³. Analogous considerations hold also for Feulner’s algorithm.

1.1 Our contribution

In this paper, we study the hardness of finding (permutation) automorphisms of linear codes.

For what concerns automorphisms of codes, we focus on the following computational problems (all taking as input a linear code):

- AUT: finding one non-trivial automorphism (if it exists);
- AUT-Gen: finding a generating set for the automorphism group;
- AUT-Count: counting the number of automorphisms.

We relate the above problems with the problem of finding a permutation isomorphism between two given linear codes, which we refer to as **ISO**⁴, and the problem of finding a generating set for the automorphism group of a graph, which we indicate as **GraphAUT-Gen**.

Reductions We show that **AUT** reduces in polynomial time to **ISO**, i.e., that an algorithm that finds permutation isometries between two codes can be used as a fundamental subroutine for determining (at least) one non-trivial automorphism. At the same time, we provide a polynomial time reduction from **ISO** to **AUT-Gen**: this implies that **AUT-Gen** is unlikely to be easy. Indeed, thanks to the reduction from **ISO** to the Graph Isomorphism Problem (**GIP**) due to Bardet, Otmani and Saeed-Taha [BOS19], we know that **ISO** can be solved in worst case quasi-polynomial time whenever the hull dimension is $O(\log(n))$. However, when the hull dimension is large, then all known solvers [BOS19; BMS26; Sen02] take exponential time and the problem is consequently deemed hard.

Taking a fundamental inspiration from the reduction from **ISO** to **GIP** given in [BOS19], we show that a similar relation holds also for the problems of finding automorphisms on codes and graphs. Indeed, we prove that for a code with length n and hull dimension h , **AUT-Gen** can be reduced to the task of finding

³ Unless the considered codes have extreme parameters, e.g., dimension $k = O(\log(n))$

⁴ This problem is often referred to as Permutation Equivalence Problem or Permutation Code Equivalence Problem and consequently shortened as either **PEP** or **PCE**. In this paper, we prefer to specify isomorphism in the same, for the sake of consistency with the automorphism problems.

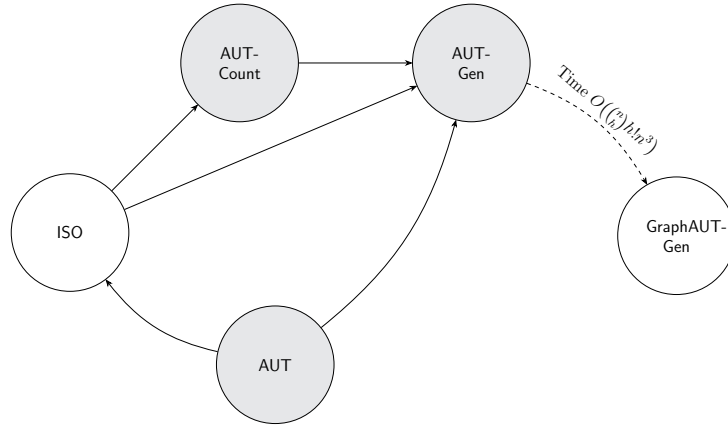


Fig. 1: Summary of the reductions we give in this paper. Continuous lines are referred to reductions that always take polynomial time.

generators for the automorphism group of a graph, a problem which we refer to as **GraphAUT-Gen**, in time $O(\binom{n}{h} h! n^3)$ ⁵.

This, in particular, implies that all problems about code automorphisms are easy whenever $h = O(\log(n))$. Indeed, **GraphAUT-Gen** can be solved in worst-case polynomial time: when the reduction to **GraphAUT-Gen** runs in quasi-polynomial time as well, then overall we get that **AUT-Gen** can be solved in worst case quasi-polynomial time. this implies that **AUT**, **AUT-Count** and **AUT-Gen** can be solved in worst case quasi polynomial time as well.

A summary of the reductions we provide in the paper is given in Figure 1. While we have not been able to show anything about the hardness of **AUT** (i.e., the problem of finding at least one non-trivial automorphism), we conjecture it should not be much easier than **AUT-Count** and **AUT-Gen**. Indeed, all of these problems are very similar and, therefore, it would be rather surprisingly if one of them turns out to be much easier than the others.

The relation with the hull We establish some simple conditions relating the hull with the automorphism group of a given code. In particular, the automorphism group of a code must be a subgroup of the automorphism group of its hull and, if the hull does not have repeated coordinates, then its automorphism group has order at most $\sim q^{h^2-1}$. Thus, if we desire a code with a large automorphism group, we need a large hull dimension and/or many repeated coordinates in the hull. Notice that a trivial hull (i.e., a hull containing only the null vector) does not exclude existence of many automorphisms, since in this case the hull has only repeated coordinates.

⁵ The term n^3 can be replaced with n^ω , where ω is the exponent for matrix inversion.

The Cartesian product of codes Many of the reductions we give in this paper make use of a code operation which we refer to as *Cartesian product of codes*: given two codes $\mathcal{A}$ and $\mathcal{B}$, their Cartesian product is

$$\mathcal{A} \times \mathcal{B} = \{(\mathbf{a}, \mathbf{b}) \mid \mathbf{a} \in \mathcal{A}, \mathbf{b} \in \mathcal{B}\}.$$

Some authors refer to it as direct sum [HP10, §1.5.4]. We prefer Cartesian product since, as we will see, the automorphism group of $\mathcal{A} \times \mathcal{B}$ is related to products of automorphisms, one for $\mathcal{A}$ and the other for $\mathcal{B}$. In particular, we will frequently rely on the Cartesian product to turn an isomorphisms search into an automorphisms search, since the automorphism group of $\mathcal{A} \times \mathcal{B}$ contains full information about all the isomorphisms between $\mathcal{A}$ and $\mathcal{B}$.

Reducing to GraphAUT-Gen Even if at a first glance translating the reduction in [BOS19] from an isomorphism problem into an automorphism problem may seem immediate, there is a non-trivial obstacle that one has to face.

Indeed, remember that the fundamental mapping from a code (represented with a generator matrix $\mathbf{G}$) to a graph is done by computation of the adjacency matrix of the graph as follows:

$$\text{Adj}(\mathbf{G}) = \mathbf{G}^\top \cdot (\mathbf{G}\mathbf{G}^\top)^{-1} \cdot \mathbf{G}.$$

Since the hull dimension corresponds to the rank deficiency of $\mathbf{G}\mathbf{G}^\top$, computation of $\text{Adj}(\mathbf{G})$ requires that the hull is trivial.

Observe that, via computation of $\text{Adj}(\mathbf{G})$, one immediately gets a graph whose automorphism group corresponds to the automorphism group of the code generated by $\mathbf{G}$. Thus, the reduction to GraphAUT-Gen for codes with trivial hull is immediate; if the hull is non-trivial, instead, adjacency matrices cannot be computed as above and some workaround is needed.

To deal with non-trivial hulls, the authors of [BOS19] propose to shorten $\mathcal{A}$ and $\mathcal{B}$ on two information sets $J, U \subseteq \{1, \dots, n\}$ for the hull. When U is indeed the image of J under the sought isometry, then the shortened codes are isomorphic as well. Notice that, for any isomorphism between the shortened codes, we have at most $h!$ candidate isomorphisms between the initial codes since any map from J and U would be valid (this map moves null coordinates to null coordinates). Thus, enumerating all choices for U and all maps from J to U , we get a worst case computational overhead of $O\left(\binom{n}{h} h!\right)$.

At a first glance, one may think that the same approach can work also for automorphisms. Indeed, starting from a code $\mathcal{C}$, for each pair (J, U) , one can construct two codes $\mathcal{A}, \mathcal{B}$ that i) have trivial hull, and ii) are such that isomorphisms between $\mathcal{A}$ and $\mathcal{B}$ are inherited from automorphisms of $\mathcal{C}$ that send J to U . All such isomorphisms can be found by first computing the product code $\mathcal{A} \times \mathcal{B}$ and then reducing to GraphAUT-Gen (the Cartesian product of codes with a trivial hull has trivial hull, as well). If we fix J and brute force over all choices for U , a solver for GraphAUT-Gen is called at most $\binom{n}{h}$ times. Again, because of the null coordinates created by shortening, every found automorphism

of $\mathcal{A} \times \mathcal{B}$ can be associated with at most $h!$ candidate automorphisms for the initial code $\mathcal{C}$.

When h is small, this reduction looks efficient; however, some caution is needed. Indeed, we cannot guarantee that each isomorphism between the shortened codes $\mathcal{A}$ and $\mathcal{B}$ is associated with an automorphism of $\mathcal{C}$. For this reason, every isomorphism from $\mathcal{A}$ to $\mathcal{B}$ needs to be checked, in order to see if it is associated with an automorphism of $\mathcal{C}$. Since the amount of isomorphisms from $\mathcal{A}$ to $\mathcal{B}$ can be as large as S_n , we may end up with an exponential time complexity. We actually give examples where this indeed happens: while one may argue that such examples are rather contrived (and, perhaps, easy to tackle with ad-hoc techniques), their existence is enough to show that the approach based on shortenings is somewhat incomplete.

Notice that, for a code with an exponentially large automorphism group (for instance, the repetition code), the above reduction would take exponential time as well. Indeed, we actually materialize and check each automorphism: to handle such cases, we would need to work only with generating sets.

We are actually able to do this by introducing a novel *de-hulling* technique which consists in enriching a generator matrix of $\mathcal{C}$ with carefully chosen amounts of replicas of the columns indexed by an information set for the hull. By doing this, we get a new code with trivial hull and length $\tilde{n} = n + h + \frac{ph(h+1)}{2} = O(n^2)$, with p being the characteristic of the finite field. For each pair (J, U) (and for each order of the indices in U) we use this procedure to get the two codes $\mathcal{A}$ and $\mathcal{B}$. As a fundamental difference with the case of shortenings, now handling *spurious isomorphisms* is much easier. In particular, it is enough to look at (and modify accordingly) generators for $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$: even if such a group is exponentially large, the number of generators is upper bounded by a polynomial in n .

All in all, we get a reduction running in time $O(\binom{n}{h} h! n^3)$: when $h = O(\log(n))$, this yields quasi-polynomial worst case time. When h is linear in n , the reduction runs in exponential time.

Efficient solvers and easy instances As we have already said, state-of-the-art algorithms such as Leon’s and Feulner’s algorithms for solving AUT-Gen take time which is exponential in the code dimension (or code length), and get slower as q increases. Notice that both algorithms do not have an immediate relation with the hull.

Instead, in light of our reductions to GraphAUT-Gen, we discover that AUT-Gen is worst case easy when h is small enough. In practice, on average, GraphAUT-Gen can be solved in polynomial time for many codes. Indeed, AUT-Gen can be solved in polynomial time for the wide majority of graphs [BES80]: consequently, when $h = O(1)$, our algorithms should run in average polynomial time.

To corroborate this claim, we have developed proof-of-concept implementations (using Magma and Sagemath) of the shortening-based reduction and have

conducted several benchmarks on codes having a prescribed (large) automorphism group. We have also compared with the built-in implementations of Leon’s and Feulner’s algorithms provided by **Magma** and **Sagemath**. As we expected, our implementation outperforms significantly the timings of the built-in functions.

For instance, on a standard laptop, using **Magma**, for binary codes with trivial hull of length $n = 100$ and dimension $k = 50$, Leon’s algorithm takes 1.5 seconds, while our solver takes less than 1 millisecond. Working on codes with hull of dimension $h = 2$, length $n = 140$ and dimension $k = 70$, Leon’s algorithm takes approximately 1928 seconds, while our solver takes less than 9 seconds.

1.2 Paper organization

In Section 2 we provide the notation used in this work and the preliminaries needed. Section 3 provides results about the relation between the automorphism group of a code and its hull. In Section 4 we introduce the Cartesian product of code and study its automorphism group. In Section 5 we study the relationship between the automorphism problem and the code equivalence problem. Section 6 concerns reductions from the code automorphism problem to the graph automorphism problem. In Section 7 we give another reduction which, differently from the one in Section 6, never achieves exponential time complexity unless h is linear in n . Finally, in Section 8 we provide some benchmarks about the running time of a proof-of-concept implementation of our reductions and compare with state-of-the-art solvers.

The code we have developed for preparing this paper is made publicly available and can be found at

<https://github.com/gdalc/permutation-automorphisms>

1.3 Relation with other works

A preliminary version of this paper has been presented at the workshops *AMPQC*⁶, held in August 10-14, 2026 in Ohrid, Macedonia, and *Mathematics of Post-Quantum Cryptography*, held in September 7-11, 2026 in Raitenhaslach, Germany⁷.

While we were finalizing the paper, we became aware of [BHJ26], which has been uploaded on Arxiv on September 21, 2026. We remark that the two papers have been conducted as independent and concurrent works. While some reductions are given in both papers, [BHJ26] does not mention reductions to GraphAUT-Gen and does not give concrete algorithms to solve AUT-Gen, as we do instead. Moreover, [BHJ26] does not consider AUT. Notice that, remarkably, [BHJ26] gives reduction also for the case of monomial automorphisms, which we do not treat in this paper.

⁶ <https://pqcrypto.cs.ru.nl/ampqc/>

⁷ <https://mathpqc26.cry.cit.tum.de/>

2 Preliminaries

In this section, we settle the notation and give the main background notions about linear codes.

2.1 Notation

Sets are denoted with the usual curly brackets $\{x_1, \dots, x_n\}$. Instead, we denote ordered sets (or tuples) using round brackets $(x_1, \dots, x_n)$. We will frequently indicate $[n] = \{1, \dots, n\}$. Let $q = p^m$ with p a prime and $m \in \mathbb{N} \setminus \{0\}$; then, $\mathbb{F}_q$ indicates a finite field with q elements and characteristic p . Bold uppercase letters indicate matrices, bold lowercase letters indicate vectors. The identity of size k is $\mathbf{I}_k$. The general linear group $GL_k(\mathbb{F}_q)$ is the group of all non-singular, $k \times k$ matrices over $\mathbb{F}_q$. For $\mathbf{a} \in \mathbb{F}_q^n$ and $J \subseteq [n]$, $\mathbf{a}_J$ is the vector formed by the entries of $\mathbf{a}$ which are indexed by J . Analogous notation applies to matrices: for a matrix $\mathbf{A}$ with n columns, we use $\mathbf{A}_J$ to indicate the submatrix formed only by the columns of $\mathbf{A}$ which are indexed by J . The complementary of a set $J \subseteq [n]$ is indicated as $\bar{J} = [n] \setminus J$. Given vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}_q^n$, we denote their inner product as $\langle \mathbf{a}; \mathbf{b} \rangle = \mathbf{a}\mathbf{b}^\top$.

Group theory We indicate with S_n the symmetric group on n elements and view its elements as permutations. For $\pi = (i_1, \dots, i_n) \in S_n$, with the meaning that $\pi(j) = i_j$, its action on a vector $\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{F}_q^n$ is determined as

$$\pi(\mathbf{a}) = (a_{\pi^{-1}(1)}, \dots, a_{\pi^{-1}(n)}).$$

We naturally extend the action of permutations on matrices: if $\mathbf{A} = (\mathbf{a}_1, \dots, \mathbf{a}_n) \in \mathbb{F}_q^{k \times n}$ (i.e., $\mathbf{a}_i$ is the i -th column of $\mathbf{A}$), then

$$\pi(\mathbf{A}) = (a_{\pi^{-1}(1)}, \dots, a_{\pi^{-1}(n)}).$$

For a set $J \subseteq [n]$, we write $\pi(J) = \{\pi(j) \mid j \in J\}$; in other words, $\pi(J)$ is the (unordered) set containing the images of the indices in J .

We sometimes write permutations using the cycles decomposition, e.g.,

$$\pi = (\sigma_1, \sigma_2, \dots, \sigma_d),$$

where each $\sigma_i = (u_1^{(i)}, u_2^{(i)}, \dots, u_{\ell_i}^{(i)})$ is a single cycle, i.e., a permutation such that

$$u_1^{(i)} \rightarrow u_2^{(i)} \rightarrow \dots \rightarrow u_{\ell_i}^{(i)} \rightarrow u_1^{(i)}.$$

If an index is fixed by a permutation, we omit it in the cycles decomposition. For instance, let $\pi = (3, 5, 2) \in S_5$: this permutation fixes the indices 1 and 4. The permutation fixing all indices is the identity, and we indicate as *id*. Lastly, given $\pi \in S_n$ and $J \subset \{1, \dots, n\}$, we denote the restriction of the permutation π on the elements of J as $\pi|_J$.

If G is a subgroup of S_n , we write $G \leq S_n$. We write $G = \langle g_1, \dots, g_m \rangle$ if G admits $g_1, \dots, g_m \in S_n$ as generators.

A set $A = \{g_1, \dots, g_m\}$ is called a *generating set* (or set of *generators*) of G if $\langle A \rangle = G$, meaning every element in G can be written as a finite product of elements in A and their inverses. We say A is a *minimal generating set* if no proper subset of A generates G ; that is, $\langle A \setminus \{g\} \rangle \subsetneq G$ for every $g \in A$.

It is known that any subgroup of S_n admits a finite set of generators, with size at most $\lceil \log_2(n!) \rceil$ which is bounded by a polynomial of n and such generating sets can be efficiently constructed in polynomial time [FHL80; Sim70].

We use the symbol $\cong$ to denote group isomorphism.

Graphs In this paper, we consider simple, undirected and (possibly) weighted graphs. Let $V = \{v_1, \dots, v_n\}$ be a set of n vertices, labeled as $v_1, \dots, v_n$. An edge for the vertices set V is a triple $(v_i, v_j, a_{i,j})$, which stands for the fact that there is an edge joining the i -th and the j -th vertices, labeled by the value $a_{i,j}$. We denote with E the collection of such triples, and with $\mathcal{G} = (V, E)$ the graph having vertices defined by V and edges defined by E .

The geometry of the graphs we are interested in is fully determined by an $n \times n$ matrix called *adjacency matrix*, defined as

$$\mathbf{A} = \{a_{i,j}\}_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}.$$

If there is an edge between the i -th and the j -th vertices, then the weight of the edge is equal to $a_{i,j} \neq 0$; if there is no edge, then $a_{i,j} = 0$.

Remark 1. As we shall see later on, for the graphs we are interested in, the edge labels will be elements of $\mathbb{F}_q$. Therefore, for $q = 2$, all edges have the same label $1 \in \mathbb{F}_2$: we can consider that the resulting graph is actually not weighted or, more conveniently, that it is a special case of the more general case of weighted graphs. When $q > 2$, instead, edge labels can be possibly different.

Definition 1 (Graph Automorphism). Let $\mathcal{G} = (V, E)$ be a simple, weighted and undirected graph with vertices set V and edges set E . An automorphism of $\mathcal{G}$ is a bijection $\phi : V \rightarrow V$ that preserves the adjacency relation, i.e. for any pair of vertices $u, v \in V$,

$$(u, v, a) \in E \iff (\phi(u), \phi(v), a) \in E.$$

If $\mathbf{A}$ is the $n \times n$ adjacency matrix, an automorphism of $\mathcal{G}$ can be equivalently defined as a permutation matrix $\mathbf{P} \in S_n$ such that

$$\mathbf{P}^\top \mathbf{A} \mathbf{P} = \mathbf{A}.$$

2.2 Linear codes

A linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with length n and dimension k is a k -dimensional linear subspace of $\mathbb{F}_q^n$. Sometimes, we will say that $\mathcal{C}$ is an $[n, k]_q$ code, or just an $[n, k]$

code when the finite field size is clear from the context. To represent a code, we can use either a generator matrix, a matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ whose rows form a basis for $\mathcal{C}$, or a parity-check matrix, a matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ whose rows form a basis for the null space of $\mathcal{C}$. In other words,

$$\mathcal{C} = \{\mathbf{u}\mathbf{G} \mid \mathbf{u} \in \mathbb{F}_q^k\} = \{\mathbf{c} \in \mathbb{F}_2^n \mid \mathbf{c}\mathbf{H}^\top = \mathbf{0}\}.$$

Notice that $\mathbf{G}\mathbf{H}^\top = \mathbf{0}$. For any $\mathbf{S} \in GL_k(\mathbb{F}_q)$, $\mathbf{G}$ and $\mathbf{S}\mathbf{G}$ generate the same code. In this case, we say that $\mathbf{S}$ is a change of basis. Analogously, for any $\mathbf{U} \in GL_{n-k}(\mathbb{F}_q)$, $\mathbf{H}$ and $\mathbf{U}\mathbf{H}$ are parity-check matrices for the same code.

The space of all vectors which are orthogonal to the codewords in $\mathcal{C}$ is called dual code and indicated as $\mathcal{C}^\perp$; if $\mathcal{C}$ has dimension k , then $\mathcal{C}^\perp$ has dimension $n - k$. Notice that, for $\mathbf{G}$ and $\mathbf{H}$ generator and parity-check matrices for $\mathcal{C}$, respectively, we have that $\mathcal{C}^\perp$ is generated by $\mathbf{H}$ and admits $\mathbf{G}$ as parity-check matrix.

Definition 2 (Information set). *A set $J \subseteq \{1, \dots, n\}$ is an information set if $\mathcal{C}_J = \{\mathbf{c}_J \mid \mathbf{c} \in \mathcal{C}\} = \mathbb{F}_q^k$. Equivalently, J is an information set if and only if, for each generator matrix $\mathbf{G}$ and each parity-check matrix $\mathbf{H}$, it holds that*

$$\text{rank}(\mathbf{G}_J) = k, \quad \text{rank}(\mathbf{H}_{\overline{J}}) = n - k.$$

For any information set J , we can define a generator matrix where the columns indexed by J form the identity matrix. Such a matrix is said to be in *standard form* and can be obtained from any generator matrix with a change of basis:

$$\mathbf{G} \xrightarrow{\text{Change of basis: } \mathbf{G}_J^{-1} \in GL_k(\mathbb{F}_q)} \mathbf{G}_J^{-1} \cdot \mathbf{G}.$$

When $J = \{1, \dots, k\}$, then the standard form is somewhat more special and we speak of *systematic form*.

Let $\mathcal{C}$ be a code with length n . We denote with $\text{Supp}(\mathcal{C})$ the subset of $\{1, \dots, n\}$ of all positions where at least one codeword in the code has a non-zero value.

We recall the notion of hull, which plays a crucial role in all our results:

Definition 3 (Hull). *The hull of a linear code is the intersection between the code and its dual*

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

We say the hull is trivial if it has dimension 0, i.e., if it contains only the null codeword. A code with trivial hull is called Linear Complementary Dual (LCD).

The hull can be computed in polynomial time $O(n^3)$: for instance, it is enough to construct the matrix $\begin{pmatrix} \mathbf{G} \\ \mathbf{H} \end{pmatrix}$ and then determine its right kernel. The hull dimension corresponds to $k - \text{rank}(\mathbf{G}\mathbf{G}^\top)$.

If $\mathcal{H}(\mathcal{C}) = \mathcal{C}$, then we say that $\mathcal{C}$ is self-orthogonal (sometimes, also called weakly self-dual). For a self-orthogonal code generated by $\mathbf{G}$, it holds that

$\mathbf{G}\mathbf{G}^\top = \mathbf{0}$. Notice that, by definition, the hull is a self-orthogonal subcode of both $\mathcal{C}$ and $\mathcal{C}^\perp$. If $\mathcal{H}(\mathcal{C}) = \mathcal{C} = \mathcal{C}^\perp$, then we say that $\mathcal{C}$ is a self-dual code.

For a given finite field with q elements, the number of self orthogonal codes can be counted precisely [Sen97, Theorem 1]. We recall this result, together with some asymptotics on how such a number grows, in Appendix A.

Finally, we define the shortening of a code.

Definition 4 (Shortening). *Given a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, the shortened code of $\mathcal{C}$ over $J \subset \{1, \dots, n\}$ is defined as*

$$\mathcal{C}_{-J} = \{\mathbf{c} \in \mathcal{C} \mid \forall j \in J, \mathbf{c}_j = 0\}.$$

2.3 Isomorphisms and automorphisms for linear codes

For a permutation $\varphi \in S_n$, we write

$$\varphi(\mathcal{C}) = \{\varphi(\mathbf{c}) \mid \mathbf{c} \in \mathcal{C}\}.$$

If, given two codes $\mathcal{C}'$ and $\mathcal{C}$, there exists a permutation $\varphi \in S_n$ such that $\mathcal{C}' = \varphi(\mathcal{C})$, we say the two codes are permutation equivalent or isomorphic. Observe that, in such a case, if $\mathbf{P}$ is the permutation matrix associated with φ , then $\mathcal{C}' = \varphi(\mathcal{C})$ implies that, for any pair of generator matrices $\mathbf{G}$ and $\mathbf{G}'$ for $\mathcal{C}$ and $\mathcal{C}'$, then

$$\exists \mathbf{S} \in GL_k(\mathbb{F}_q) \text{ such that } \mathbf{G}' = \mathbf{S}\mathbf{G}\mathbf{P}.$$

If $\mathcal{C}' = \varphi(\mathcal{C})$, we also have that $\mathcal{C}'^\perp = \varphi(\mathcal{C}^\perp)$ and $\mathcal{H}(\mathcal{C}') = \varphi(\mathcal{H}(\mathcal{C}))$.

When a permutation maps a code into itself (i.e. $\mathcal{C} = \mathcal{C}'$), then we have an *automorphism*.

Definition 5 (Permutation automorphism). *Given a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, a (permutation) automorphism is a permutation $\varphi \in S_n$ such that $\mathcal{C} = \varphi(\mathcal{C})$. Equivalently, a permutation $\mathbf{P}$ is an automorphism for a code having $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ as generator matrix if there exists a change of basis $\mathbf{S} \in GL_k(\mathbb{F}_q)$ such that*

$$\mathbf{G} = \mathbf{S}\mathbf{G}\mathbf{P}.$$

It can be easily verified that the combination between any two automorphisms leads to another automorphism and that the identity is an automorphism for every linear code; we thus speak of *automorphism group*.

Definition 6 (Permutation automorphism group). *Given a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, its (permutation) automorphism group is the subgroup of S_n containing all automorphisms of $\mathcal{C}$, i.e.,*

$$\text{P-Aut}(\mathcal{C}) = \{\varphi \in S_n \mid \varphi(\mathcal{C}) = \mathcal{C}\}$$

As said before, any code admits, as automorphism, the identity; we say this is a *trivial automorphism*. When the automorphism group contains only this element, we say it is trivial as well.

2.4 Problems and computational complexity

We say that a problem A reduces (or is reducible) to a second problem B if there exists a polynomial-time algorithm p that solves A using an oracle which solves B in one step. In this case we write $A \leq_p B$. Finally, we say that two problems A and B are equivalent, and we write $A \sim_p B$, if $A \leq_p B$ and $B \leq_p A$. We remark that polynomial-time means that the algorithm runs in time which can be upper bounded by a polynomial of the input size.

We proceed by formalizing the problems we study in this paper.

List of code automorphism problems

Definition 7 (AUT). *Given $\mathcal{C}$, find a non-trivial automorphism of $\mathcal{C}$.*

Definition 8 (AUT-Gen). *Given $\mathcal{C}$, find a generating set for the automorphism group of $\mathcal{C}$.*

Definition 9 (AUT-Count). *Given $\mathcal{C}$, count the order of the automorphism group of $\mathcal{C}$.*

List of code isomorphism problems

Definition 10 (ISO). *Given $\mathcal{C}, \mathcal{C}' \subseteq \mathbb{F}_q^n$, find an isomorphism π such that $\mathcal{C}' = \pi(\mathcal{C})$.*

Definition 11 (Dec-ISO). *Given $\mathcal{C}, \mathcal{C}' \subseteq \mathbb{F}_q^n$, decide if there is an isomorphism π such that $\mathcal{C}' = \pi(\mathcal{C})$.*

Clearly $\text{AUT} \leq_p \text{AUT-Gen}$. We also have $\text{AUT-Count} \leq_p \text{AUT-Gen}$ thanks to the Schreier–Sims algorithm [Knu91; Sim70]. Finally, it was recently proved in [BM23] that $\text{Dec-ISO} \sim_p \text{ISO}$.

3 Automorphisms and Hulls

We derive some relations between the automorphism group of a linear code and its hull. Thanks to such (simple) relations, we get some necessary conditions on the hull for a code having a large automorphism group.

3.1 Upper bounds on the size of the automorphism group

One can observe that if the code has many repeated coordinates, then there are some obvious automorphisms. However, when no repeated coordinates exist, then providing bounds on the size of the automorphism group is definitely not trivial. At the best of our knowledge, there are no general bounds, apart from some results relating the number of automorphisms with the size of $GL_k(\mathbb{F}_q)$. We recall (and prove, for the sake of completeness) this main result, for the case of a code with no repeated coordinates.

Theorem 1 *If a code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with dimension n has no repeated coordinates, then*

$$|\text{P-Aut}(\mathcal{C})| \leq \min \{|GL_k(\mathbb{F}_q)|, n!\}.$$

Proof. Since $\mathcal{C}$ does not have repeated coordinates, any generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ has distinct columns. Let $\mathbf{S} \in GL_k(\mathbb{F}_q)$, this can be associated with an automorphism only if there exists some $\mathbf{P} \in S_n$ such that $\mathbf{S}\mathbf{G} = \mathbf{G}\mathbf{P}$. In other words, $\mathbf{S}\mathbf{G}$ and $\mathbf{G}$ have the same columns. Notice that since $\mathbf{G}$ does not have repeated coordinates, then for any $\mathbf{S}$ there can be at most one valid permutation $\mathbf{P}$. Therefore, the maximum number of automorphisms cannot be larger than $|GL_k(\mathbb{F}_q)|$. Trivially, $|\text{P-Aut}(\mathcal{C})| \leq |S_n| = n!$, so we have the thesis. $\square$

Since $\text{P-Aut}(\mathcal{C}) = \text{P-Aut}(\mathcal{C}^\perp)$, the above bound can be refined in the case in which also the dual has no repeated coordinates.

Corollary 1. *If a code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with dimension k has no repeated coordinates, and the same holds for $\mathcal{C}^\perp$, then*

$$|\text{P-Aut}(\mathcal{C})| \leq \min \{|GL_k(\mathbb{F}_q)|, |GL_{n-k}(\mathbb{F}_q)|, n!\}.$$

Remark 2. Observe that if $\mathcal{C}$ does not have repeated coordinates but $\mathcal{C}^\perp$ does, then $\mathcal{C}$ has codewords of weight 2. This is the case, for instance, of the single parity-check code with length n : it has no repeated coordinates but its dual is the repetition code of length n , where all coordinates are identical.

When there are repeated coordinates, the above bounds get (unsurprisingly) higher. We make this intuition more formal in the following theorem.

Theorem 2 *For a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, let $J_1, \dots, J_u \subseteq \{1, \dots, n\}$ be a partition of $\{1, \dots, n\}$ such that, for each J_i of cardinality ℓ_i , all coordinates indexed by J_i are repeated, and the coordinates indexed by J_i are different from the coordinates indexed by J_j for each $i \neq j$. Then,*

$$|\text{P-Aut}(\mathcal{C})| \leq \min \{|GL_k(\mathbb{F}_q)|, u!\} \cdot \prod_{i=1}^u (\ell_i!).$$

Proof. Notice that every permutation $\pi \in S_n$ such that $\pi(J_i) = J_i$ and that fixes all other indices is an automorphism for $\mathcal{C}$, since it is just shuffling identical coordinates. The number of such permutations is $\prod_{i=1}^u (\ell_i!)$. Repeating the reasoning from the proof of Theorem 1, we now have that for each $\mathbf{S} \in GL_k(\mathbb{F}_q)$ we can have a maximum of $\prod_{i=1}^u (\ell_i!)$ automorphisms. The factor $\min \{|GL_k(\mathbb{F}_q)|, u!\}$ appears thanks to the following reasoning. Any generator matrix for $\mathcal{C}$ is, up to a reordering of its columns, given by

$$\left(\underbrace{\mathbf{g}_1, \dots, \mathbf{g}_1}_{\ell_1 \text{ times}}, \dots, \underbrace{\mathbf{g}_u, \dots, \mathbf{g}_u}_{\ell_u \text{ times}} \right) = (\mathbf{G}_1, \dots, \mathbf{G}_u),$$

where the vectors $\mathbf{g}_1, \dots, \mathbf{g}_u$ are distinct by hypothesis and each $\mathbf{G}_i$ has size $k \times \ell_i$. If we left multiply by some $\mathbf{S} \in GL_k(\mathbb{F}_q)$, we get

$$\mathbf{S}(\mathbf{G}_1, \dots, \mathbf{G}_u) = (\mathbf{G}'_1, \dots, \mathbf{G}'_u),$$

with $\mathbf{G}'_i = \mathbf{S}\mathbf{G}_i$. The matrix $\mathbf{S}$ can be associated with an automorphism only if, for any $i \in \{1, \dots, u\}$, there is an index $j_i \in \{1, \dots, u\}$ such that $\mathbf{G}'_i = \mathbf{G}_{j_i}$. In other words, each valid $\mathbf{S}$ must be associated with a bijection of $\{1, \dots, u\}$, and therefore there cannot be more than $u!$ valid such matrices. $\square$

Remark 3. The repetition code meets the bound from the previous theorem. Indeed, since all coordinates are repeated, we have $u = 1$ and $\ell_1 = n$: the bound yields $n!$, which corresponds to the size of S_n .

Remark 4. The bound in the previous theorem can be made sharper considering that, for the number of bijections, one may consider also the sizes of the subsets $J_1, \dots, J_u$. For instance, if two subsets have distinct sizes ℓ_i and ℓ_j , then a matrix $\mathbf{S} \in GL_k(\mathbb{F}_q)$ will never send $\mathbf{G}_i$ to $\mathbf{G}_j$. With such a reasoning, one can refine the bound. However, arguably, codes with repeated coordinates are not the most interesting case-study (as they have some obvious and easy to determine automorphisms) so we do not enter into further details.

3.2 Relations between automorphisms and hulls

We first show that every automorphism for the code must be an automorphism for the hull, as well.

Theorem 3 *For every code $\mathcal{C}$, $\text{P-Aut}(\mathcal{C}) \leq \text{P-Aut}(\mathcal{H}(\mathcal{C}))$, i.e., every automorphism of the code must be an automorphism of the hull.*

Proof. If the hull is trivial, then its automorphism group is equal to S_n , so the thesis is trivially true. We consider the more interesting case of hull dimension $h \geq 1$. Let $B = \{\mathbf{c}_1, \dots, \mathbf{c}_h\}$ be a basis for the hull of a code $\mathcal{C}$. Let $\varphi \in S_n$ and $\mathcal{C}' = \varphi(\mathcal{C})$. Let us consider $B' = \{\mathbf{c}'_1 = \varphi(\mathbf{c}_1), \dots, \mathbf{c}'_h = \varphi(\mathbf{c}_h)\}$. Let us prove that the space generated by B' has dimension h and stays inside the hull of $\mathcal{C}'$. For every $\mathbf{a}' \in \mathcal{C}'$ there exists an $\mathbf{a} \in \mathcal{C}$ such that $\mathbf{a}' = \varphi(\mathbf{a})$. Then

$$\langle \mathbf{c}'_i ; \mathbf{a}' \rangle = \langle \varphi(\mathbf{c}_i) ; \varphi(\mathbf{a}) \rangle = \langle \mathbf{c}_i ; \mathbf{a} \rangle = 0.$$

Hence, the space generated by B' lives inside $\mathcal{H}(\mathcal{C}')$ and $h' = \dim(\mathcal{H}(\mathcal{C}')) \geq h$. Let us now prove that $h' = h$ and hence the space generated by B' is exactly the hull of $\mathcal{C}'$. Let φ^{-1} be the inverse of φ . Hence $\varphi^{-1}(\mathcal{C}') = \mathcal{C}$. Let $D' = \{\bar{\mathbf{c}}'_1, \dots, \bar{\mathbf{c}}'_{h'}\}$ be a basis for the hull of $\mathcal{C}'$. Let us consider $D = \{\bar{\mathbf{c}}_1 = \varphi^{-1}(\bar{\mathbf{c}}'_1), \dots, \bar{\mathbf{c}}_h = \varphi^{-1}(\bar{\mathbf{c}}'_{h'})\}$. Following the same reasoning used above, we can prove that the space generated by D , that has dimension h' , lives inside $\mathcal{H}(\mathcal{C})$. Consequently, $h' \leq h$, and so $h' = h$. $\square$

Thanks to the above theorem, one can use the hull of the code to get some ideas on the automorphism group of the code. For instance, if the hull is found to have only trivial automorphisms, then the code must have trivial automorphisms as well.

Remark 5. Notice that Theorem 3 works only in one direction, i.e., there may be automorphisms of the hull which are not automorphisms of the code. For instance, if $\mathcal{H}(\mathcal{C}) = \{(0, \dots, 0)\}$, we have $\text{P-Aut}(\mathcal{H}(\mathcal{C})) = S_n$ but not necessarily $\text{P-Aut}(\mathcal{C}) = S_n$.

As a consequence of the previous theorem combined with the bounds from Theorem 1 and Corollary 1, we get an upper bound on the size of the automorphism group that depends on properties of the hull.

Theorem 4 *Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a linear code whose hull $\mathcal{H}(\mathcal{C})$ has dimension h . Let $J_1, \dots, J_u \subseteq \{1, \dots, n\}$ as in Corollary 1, with J_i having cardinality ℓ_i . Then,*

$$|\text{P-Aut}(\mathcal{C})| \leq \min \{ |GL_h(\mathbb{F}_q)|, u! \} \cdot \prod_{i=1}^u \ell_i!.$$

Proof. By Theorem 3, $\mathcal{C}$ cannot have more automorphisms than $\mathcal{H}(\mathcal{C})$. Then, we just apply Corollary 1 to the hull. $\square$

The above theorem establishes some relations between the hull dimension and the size of the automorphism group of a linear code. Indeed, a code having a large automorphism group must necessarily have either a large hull dimension and/or a hull with many repeated coordinates.

Notice that a trivial hull does not put any limit on the number of automorphisms, since it corresponds to the (trivial) code formed by the all-zeros codeword, so the upper bound gives us $n!$.

Example 1. Let $\mathcal{C} \subseteq \mathbb{F}_{127}^6$ be the code with generator matrix

$$\mathbf{G} = \begin{pmatrix} 103 & 4 & 79 & 62 & 96 \\ 19 & 47 & 76 & 46 & 115 \\ 21 & 47 & 92 & 45 & 40 \end{pmatrix}.$$

This code has hull dimension $h = 1$ and a basis for the hull is $\mathbf{B} = (1 \ 53 \ 48 \ 116 \ 90)$. Since there are no repeated coordinates, an application of Theorem 4 tells us that the maximum number of automorphisms of $\mathcal{C}$ is $|GL_1(\mathbb{F}_{127})| = 126$. It can be verified that the hull has only the trivial automorphism; so, also $\mathcal{C}$ admits only the trivial automorphism.

Example 2. Let $\mathcal{C} \subseteq \mathbb{F}_{17}^6$ be the code with generator matrix

$$\mathbf{G} = \begin{pmatrix} 0 & 8 & 9 & 15 & 2 \\ 6 & 2 & 10 & 13 & 11 \\ 14 & 6 & 15 & 15 & 0 \end{pmatrix},$$

with hull generated by $\mathbf{B} = \begin{pmatrix} 0 & 1 & 16 & 4 & 13 \end{pmatrix}$. Since there are no repeated coordinates, we know the hull and the code as well can have at most $|GL_1(\mathbb{F}_{17})| = 16$ automorphisms. In this case, the hull has some automorphisms: indeed, if we multiply $\mathbf{B}$ by some non-zero $\alpha \in \mathbb{F}_{17}$, for $\alpha = \pm 1, \pm 4$ we get that $\alpha\mathbf{B}$ and $\mathbf{B}$ have the same entries (but in a different order). We have

$$\text{P-Aut}(\mathcal{H}(\mathcal{C})) = \underbrace{\langle (2, 5, 3, 4) \rangle}_{\varphi} = \{id, \varphi, \varphi^2, \varphi^3\}.$$

So, in this case $\text{P-Aut}(\mathcal{H}(\mathcal{C}))$ is a cyclic group of order 4. It can be verified that $\text{P-Aut}(\mathcal{C})$ is the subgroup of $\langle \varphi \rangle$ generated by $\varphi^2 = (2, 3)(4, 5)$, with order 2. So, this code has a unique non-trivial automorphism, given by φ^2 .

3.3 Looking solely at the hull may not be enough

While the relations from the previous subsection may be rather useful, looking only at the hull may not be the best strategy to find automorphisms.

Indeed, a simple strategy to determine automorphisms of a given code $\mathcal{C}$ may be to first determine $\text{P-Aut}(\mathcal{H}(\mathcal{C}))$ and then, for each $\varphi \in \text{P-Aut}(\mathcal{H}(\mathcal{C}))$, verify if indeed $\varphi(\mathcal{C}) = \mathcal{C}$.

Checking that φ is indeed an automorphism is computationally easy: for any pair $\mathbf{G}$ and $\mathbf{H}$ of generator and parity-check matrices, respectively, it is enough to check that $\varphi(\mathbf{G})\mathbf{H}^\top = \mathbf{0}$. We observe that since the hull is a code with presumably low dimension, finding its automorphisms using any of the solvers from the literature may be easy. However, the number of automorphisms possessed by the hull may be extremely large. For instance, in the case of a trivial hull, we have $\text{P-Aut}(\mathcal{H}(\mathcal{C})) = S_n$. More generally, it can be seen that when both q and h are not large enough, the hull of a code necessarily has non-trivial automorphisms. For instance, if $q^h < n$, then the number of columns in any generator matrix for $\mathcal{C}$ is less than $\mathbb{F}_q^h$, hence duplicated columns must exist.

More generally, considering that the hull is a self-orthogonal code, one can get a more restrictive condition, which we give in the next theorem (the proof is deferred to Appendix A).

Theorem 5 *Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ with hull of dimension h . If $|\mathfrak{C}_{n,h,q}| < n!$, then the hull has non-trivial automorphisms. For constant q , let h^* indicate the maximum value of h for which the previous inequality is satisfied. Then, asymptotically, $h^* = \log_q(n/e) \cdot (1 + o(1))$, where e is the Napier's constant.*

The inequality in the above theorem can be used as a rule of thumb to determine when looking only at automorphisms of the hull may yield a good strategy for finding automorphisms of a code. As we shall see in the next sections, a large hull dimension is an obstacle we are not able to overcome, since our algorithm has a complexity growing at least as $O(n^h)$. However, notably, our algorithm has not strong dependency over q , as it only affects the cost of arithmetic (thus, leads only to a logarithmic overhead).

As a final remark, we comment on the reason why we need to check each automorphism of the hull, which may not be obvious. We observe that, even when the automorphism group of the hull is large, it can be represented using generators. As we have already said, the largest amount of generators for a subgroup of S_n is $\lceil n/2 \rceil$, hence we always dispose of an efficient representation for the group. However, we cannot just look at generators but need to test each element, as already shown by Example 2. Indeed, we cannot exclude the existence of code automorphisms which are not in the list of generators for $\text{P-Aut}(\mathcal{H}(\mathcal{C}))$ but are generated by some combination of said generators. A similar limitation applies also to the algorithm we give in Section 6 and, it will also be the only reason why our algorithm can have exponential complexity.

4 The Cartesian Product of Codes

We introduce and study a code construction which, later on, will be frequently used.

Definition 12. Let $\mathcal{A} \subseteq \mathbb{F}_q^{n_1}$ and $\mathcal{B} \subseteq \mathbb{F}_q^{n_2}$ be two linear codes with lengths n_1 and n_2 and dimensions k_1 and k_2 , respectively. We refer to them as *component codes* and define their *Cartesian product* as

$$\mathcal{A} \times \mathcal{B} = \{(\mathbf{a}, \mathbf{b}) \mid \mathbf{a} \in \mathcal{A}, \mathbf{b} \in \mathcal{B}\},$$

which is a linear code with length $n_1 + n_2$ and dimension $k_1 + k_2$. Given generator matrices $\mathbf{A} \in \mathbb{F}_q^{k_1 \times n_1}$ and $\mathbf{B} \in \mathbb{F}_q^{k_2 \times n_2}$ for $\mathcal{A}$ and $\mathcal{B}$, respectively, a generator matrix for their Cartesian product is the $(k_1 + k_2) \times (n_1 + n_2)$ matrix $\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix}$.

The above definition generalizes to the case of more than two component codes: the dimension and length of the resulting product code are the sum of the dimensions and lengths of the component codes, respectively. Analogously, a generator matrix for the product code is obtained by stacking, in the main diagonal, the generator matrices of the component codes.

Since this will be crucial in the following, we recall the notion of *decomposability* [Sle60].

Definition 13. Given a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, we say it is *decomposable* if it is permutation equivalent to a Cartesian product code, i.e., if there exist codes $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_m$ and a permutation $\pi \in S_n$ such that

$$\mathcal{C} = \pi(\mathcal{C}_1 \times \mathcal{C}_2 \times \dots \times \mathcal{C}_m).$$

Otherwise, we say that $\mathcal{C}$ is *indecomposable*.

Remark 6. A code with some null coordinates can be deemed as a decomposable code. Indeed, if a code has m null coordinates, then it is permutation equivalent to a Cartesian product code in which one of the components is $\{0^m\}$, with 0^m

being the length- m all-zeros vector. Such a code is defined by a generator matrix as in Definition 12, with one of the generator matrices of the component codes being the null $1 \times m$ matrix. Notice that this is a slight abuse of notation since the resulting matrix would not have full rank.

We continue the section with some useful results about the Cartesian product of codes. Some of the proofs of the theorems in this section are referred to Appendix B.

4.1 The automorphism group of the Cartesian product of codes

If $\mathcal{A}$ and $\mathcal{B}$ are distinct and equivalent via the permutation π , it can be seen that the automorphism group of $\mathcal{A} \times \mathcal{B}$ is non-trivial since there is a special automorphism induced by π .

Definition 14. *Let $\mathcal{A}$ and $\mathcal{B}$ be two permutation equivalent $[n, k]_q$ codes via the isomorphism π . The automorphism of the product $\mathcal{A} \times \mathcal{B}$ induced by π , which we indicate as (π^{-1}, π) , is the permutation of length $2n$ such that coordinates in $\{1, \dots, n\}$ are moved to $\{n+1, \dots, 2n\}$ and then permuted according to π , while coordinates in $\{n+1, \dots, 2n\}$ are moved to $\{1, \dots, n\}$ and then permuted according to π^{-1} .*

Instead of considering only π^{-1} , Definition 14 can be extended to a more general formulation that considers any permutation χ mapping $\mathcal{B}$ to $\mathcal{A}$.

Definition 15. *Let $\mathcal{A}$ and $\mathcal{B}$ be two permutation equivalent $[n, k]_q$ codes. Let $\pi, \chi \in S_n$ be permutations such that $\pi(\mathcal{A}) = \mathcal{B}$ and $\chi(\mathcal{B}) = \mathcal{A}$: the existence of the two permutations is guaranteed by the permutation equivalence property. The automorphism of the product $\mathcal{A} \times \mathcal{B}$ induced by π, χ , which we indicate as (χ, π) is the permutation of length $2n$ such that coordinates in $\{1, \dots, n\}$ are moved to $\{n+1, \dots, 2n\}$ and then permuted according to π , while coordinates in $\{n+1, \dots, 2n\}$ are moved to $\{1, \dots, n\}$ and then permuted according to χ .*

The latter trivially generalizes Definition 14, since it suffices to consider $\chi = \pi^{-1}$.

If $\mathbf{P}_1$ is the permutation matrix associated with $\pi \in S_n$ and $\mathbf{P}_2$ is the permutation matrix associated to $\chi \in S_n$, then the permutation matrix associated with the automorphism (χ, π) can be written as

$$\mathbf{P}' = \begin{pmatrix} \mathbf{0} & \mathbf{P}_1 \\ \mathbf{P}_2 & \mathbf{0} \end{pmatrix} \in \mathbb{F}_q^{2n \times 2n}. \quad (1)$$

We now prove that Definition 15 is well-defined, i.e. that the permutation defined as (χ, π) is indeed an automorphism of the product code.

Theorem 6 *Let $\mathcal{A}, \mathcal{B} \subseteq \mathbb{F}_q^n$ be two equivalent codes, i.e., there is a permutation $\pi \in S_n$ such that $\mathcal{B} = \pi(\mathcal{A})$. Let $\chi \in S_n$ be such that $\mathcal{A} = \chi(\mathcal{B})$. Then, (χ, π) , as in Definition 15, is an automorphism of the code $\mathcal{A} \times \mathcal{B}$.*

Proof. Let $\mathbf{A}$ and $\mathbf{B}$ be generators for $\mathcal{A}$ and $\mathcal{B}$, respectively. Then, $\mathbf{B} = \mathbf{S} \cdot \pi(\mathbf{A}) = \mathbf{S}_1 \cdot \mathbf{A} \cdot \mathbf{P}_1$ and $\mathbf{A} = \mathbf{S}_2 \cdot \mathbf{B} \cdot \mathbf{P}_2$ for some non singular $\mathbf{S}_1, \mathbf{S}_2 \in \mathbb{F}_q^{k \times k}$. Let $\mathbf{G}$ be a generator matrix for the Cartesian product code, as in Definition 12, and $\mathbf{P}'$ as in Equation (1). Then

$$\begin{aligned} \mathbf{G} \cdot \mathbf{P}' &= \begin{pmatrix} \mathbf{0} & \mathbf{A} \cdot \mathbf{P}_1 \\ \mathbf{B} \cdot \mathbf{P}_2 & \mathbf{0} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{0} & \mathbf{S}_2^{-1} \cdot \mathbf{B} \\ \mathbf{S}_1^{-1} \cdot \mathbf{A} & \mathbf{0} \end{pmatrix} \\ &= \underbrace{\begin{pmatrix} \mathbf{0} & \mathbf{S}_2^{-1} \\ \mathbf{S}_1^{-1} & \mathbf{0} \end{pmatrix}}_{\mathbf{S}'} \cdot \underbrace{\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix}}_{\mathbf{G}}. \end{aligned}$$

The matrix $\mathbf{S}' \in \mathbb{F}_q^{2k \times 2k}$ has full rank since $\mathbf{S}$ has full rank by hypothesis. Since we found that $\mathbf{G} \cdot \mathbf{P}'$ and $\mathbf{G}$ generate the same code, then $\mathbf{P}'$ is an automorphism. $\square$

We now look into the automorphism group of the Cartesian product of codes. First, we consider the automorphism groups of two isomorphic codes (the proof can be found in Appendix B).

Theorem 7 *Let $\mathcal{A}$ and $\mathcal{B} = \pi(\mathcal{A})$ for some $\pi \in S_n$. Then we have that $\text{P-Aut}(\mathcal{A}) \cong \text{P-Aut}(\mathcal{B})$ and in particular:*

$$\text{P-Aut}(\mathcal{B}) = \{ \pi \circ \sigma \circ \pi^{-1} \mid \sigma \in \text{P-Aut}(\mathcal{A}) \}.$$

Then, we provide a useful result about the hull dimension of the Cartesian product code, which proof can be found in Appendix B.

Theorem 8 *Given two codes $\mathcal{A}, \mathcal{B}$ with hulls $\mathcal{H}(\mathcal{A})$ and $\mathcal{H}(\mathcal{B})$, respectively, the Cartesian product code has hull*

$$\mathcal{H}(\mathcal{A} \times \mathcal{B}) = \mathcal{H}(\mathcal{A}) \times \mathcal{H}(\mathcal{B}).$$

In particular, the hull dimension of the product code is the sum of the dimensions of the hulls.

We now provide a description of the automorphism group of codes obtained as the Cartesian product of two indecomposable codes $\mathcal{A}$ and $\mathcal{B}$. In particular, we focus on the following two kinds of automorphism.

Definition 16. *Let $\mathcal{A}, \mathcal{B} \subseteq \mathbb{F}_q^n$ be two linear codes with dimension k and length n . We say that an automorphism $\sigma \in \text{P-Aut}(\mathcal{A} \times \mathcal{B})$ is:*

– *component-wise, if*

$$\sigma(\{1, \dots, n\}) = \{1, \dots, n\}, \quad \sigma(\{n+1, \dots, 2n\}) = \{n+1, \dots, 2n\};$$

– *swapping, if*

$$\sigma(\{1, \dots, n\}) = \{n+1, \dots, 2n\}, \quad \sigma(\{n+1, \dots, 2n\}) = \{1, \dots, n\}.$$

In other words, a component-wise automorphism acts independently on each component code. Instead, swapping automorphisms move the leftmost n coordinates to the rightmost n coordinates, and viceversa. We prove in the following that i) if the component codes are indecomposable, then there are no other types of automorphisms, and ii) that swapping automorphisms exist only when the component codes are isomorphic, and iii) that component-wise automorphisms form a group which is isomorphic to the Cartesian product of the automorphism groups of the component codes. The proofs of the following two results are deferred to Appendix B.

Theorem 9 *Let $\mathcal{A}, \mathcal{B} \subseteq \mathbb{F}_q^n$ be non-equivalent codes of dimension k . Moreover, suppose that $\mathcal{A}$ and $\mathcal{B}$ are indecomposable, i.e. they are not the Cartesian product of other codes. Then*

$$\text{P-Aut}(\mathcal{A} \times \mathcal{B}) \cong \text{P-Aut}(\mathcal{A}) \times \text{P-Aut}(\mathcal{B}).$$

Corollary 2. *Let $\mathcal{A}, \mathcal{B} \subseteq \mathbb{F}_q^n$ be two equivalent codes of dimension k . Moreover, suppose that $\mathcal{A}$ and $\mathcal{B}$ are indecomposable, i.e. they are not the Cartesian product of other codes. Then*

$$\text{P-Aut}(\mathcal{A} \times \mathcal{B}) \cong (\text{P-Aut}(\mathcal{A}) \times \text{P-Aut}(\mathcal{B})) \rtimes S_2,$$

where $\rtimes$ indicates the semidirect product of groups. Moreover, if G is a group isomorphic to $\text{P-Aut}(\mathcal{A})$, it holds that $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$ is isomorphic to $G \wr S_2$.

Example 3. Let us consider $\mathbb{F}_5$ and define the linear code $\mathcal{A} \subseteq \mathbb{F}_5^5$ with length $n = 5$ and dimension $k = 3$ via the generator matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 1 & 2 \\ 0 & 1 & 0 & 2 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

It is easy to see that the permutation $\phi = (1, 2)(4, 5) \in S_5$ is a non-trivial automorphism of $\mathcal{A}$.

Next, consider the permutation $\sigma = (1 \ 3 \ 5) \in S_5$ and define $\mathcal{B} = \sigma(\mathcal{A})$. Then $\mathcal{B}$ has the following generator matrix (not in systematic form)

$$\mathbf{B} = \begin{pmatrix} 2 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 2 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Notice that $\mathcal{B}$ has a non-trivial automorphism given by $\phi' = \sigma \circ \phi \circ \sigma^{-1} = (1, 4)(2, 3) \in S_5$. The Cartesian product $\mathcal{A} \times \mathcal{B}$ has generator matrix

$$\mathbf{G} = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix} = \left(\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{array} \right),$$

and $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$ is the group generated by the following three permutations:

$$\underbrace{(1, 2)(4, 5)}_{\varphi_1}, \quad \underbrace{(6, 9)(7, 8)}_{\varphi_2}, \quad \underbrace{(1, 7)(2, 8)(3, 10)(4, 6)(5, 9)}_{\varphi_3}.$$

Observe that φ_1 and φ_2 are component-wise automorphisms and can be associated with (ϕ, id) and (id, ϕ') , respectively⁸. In particular, we have four non-trivial component-wise automorphisms and four swapping automorphisms:

- *component-wise automorphisms*: all combinations of φ_1 and φ_2 , that is,
 - the identity (id, id) ,
 - $(\phi, id) = (1, 2)(4, 5)$,
 - $(id, \phi') = (6, 9)(7, 8)$,
 - $(\phi, \phi') = (1, 2)(4, 5)(6, 9)(7, 8)$;
- *swapping automorphisms*: automorphisms obtained by using φ_3 in the combination, that is,
 - $\varphi_3 = (1, 7)(2, 8)(3, 10)(4, 6)(5, 9)$,
 - $\varphi_1 \circ \varphi_3 = \varphi_3 \circ \varphi_2 = (1, 7, 2, 8)(3, 10)(4, 6, 5, 9)$,
 - $\varphi_2 \circ \varphi_3 = \varphi_3 \circ \varphi_1 = (1, 8, 2, 7)(3, 10)(4, 9, 5, 6)$,
 - $\varphi_3 \circ \varphi_1 \circ \varphi_2 = (1, 8)(2, 7)(3, 10)(4, 9)(5, 6)$.

4.2 Handling decomposable codes

All the above results can be extended to decomposable codes. While in some cases the generalization is trivial (for instance, the hull dimension is the sum of the dimensions of the hulls of the component codes, regardless of how many components we have), for some other properties some caution is needed.

In particular, for Definition 16 and Corollary 2, the generalization may not be immediate. Indeed, when we have more than two components, there may be automorphisms that are neither swapping nor component-wise, as in Definition 16: they may act by swapping only some components and stabilizing the others. Actually, at a closer look, one can see that the definition of swapping

⁸ With some abuse of notation, this means that φ_1 is acting as φ on the first n coordinates and as id is acting locally on the last n coordinates. Analogous for φ_2 . We use the same notation also for the other component-wise automorphisms (and only in this example).

automorphisms we have in Definition 16 is specific to the case of the product of two decomposable codes.

We refrain from giving precise definitions and just comment about this with examples. Let $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_m$ with $m \geq 3$. If all components are pairwise not isomorphic, then

$$\text{P-Aut}(\mathcal{C}) = \text{P-Aut}(\mathcal{C}_1) \times \cdots \times \text{P-Aut}(\mathcal{C}_m).$$

For instance, assume that $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$ are isomorphic. Then, $\mathcal{C}$ has automorphisms that stabilize each $\mathcal{C}_i$ with $i \geq 4$, stabilize also one of the first three components and swaps the other two. One such automorphism is not swapping, nor component-wise, as per Definition 16.

Notice that, though, it is still true that for two decomposable codes $\mathcal{A}$ and $\mathcal{B}$, the product code $\mathcal{A} \times \mathcal{B}$ has swapping automorphisms (as per Definition 16) if and only $\mathcal{A}$ and $\mathcal{B}$ are isomorphic.

Finally, we observe that, for a given code, the components can be found in polynomial time. Essentially, one just needs to compute a generator matrix in Reduced Row Echelon Form (RREF) and then group coordinates according to rows having non null entries. Details about this procedure can be found in Section C.

Example 4. Let us take the code over $\mathbb{F}_{11}$, with the following generator matrix

$$\begin{pmatrix} 8 & 10 & 9 & 4 & 5 & 5 & 9 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 10 \\ 7 & 5 & 3 & 7 & 7 & 10 & 4 & 1 & 3 & 1 & 1 & 8 & 7 & 5 & 1 & 1 \\ 5 & 5 & 4 & 0 & 10 & 8 & 2 & 3 & 5 & 0 & 6 & 2 & 3 & 2 & 0 & 4 \\ 1 & 7 & 2 & 1 & 3 & 7 & 10 & 0 & 10 & 6 & 3 & 3 & 3 & 7 & 2 & 10 \\ 4 & 8 & 4 & 8 & 6 & 4 & 9 & 1 & 6 & 10 & 1 & 0 & 1 & 4 & 1 & 7 \\ 9 & 9 & 5 & 0 & 8 & 0 & 2 & 7 & 3 & 3 & 9 & 4 & 4 & 0 & 1 & 5 \\ 3 & 5 & 5 & 4 & 7 & 7 & 8 & 9 & 7 & 8 & 3 & 0 & 4 & 1 & 7 & 9 \\ 9 & 5 & 2 & 3 & 4 & 6 & 5 & 4 & 4 & 5 & 5 & 7 & 4 & 8 & 7 & 5 \\ 4 & 0 & 9 & 3 & 4 & 1 & 0 & 10 & 4 & 5 & 5 & 7 & 5 & 6 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 6 & 8 & 6 & 6 & 7 & 8 & 3 & 0 & 0 & 2 \end{pmatrix}.$$

After RREF computation, we obtain

$$\begin{pmatrix} \boxed{1 & 0 & 5 & 9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 9 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{1 & 0 & 0 & 5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{1 & 0 & 10 & 7} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{0 & 1 & 4 & 9} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{1 & 0 & 0 & 8} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{0 & 1 & 0 & 1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{0 & 0 & 1 & 6} & 0 & 0 \end{pmatrix}$$

which shows that the code is actually the cartesian product of four codes $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ and $\mathcal{C}_4$. Let $\mathcal{A} = \mathcal{C}_1 \times \mathcal{C}_2$ and $\mathcal{B} = \mathcal{C}_3 \times \mathcal{C}_4$, therefore, $\mathcal{C} = \mathcal{A} \times \mathcal{B}$.

It can be seen that both $\mathcal{A}$ and $\mathcal{B}$ have trivial automorphism groups and that they are not isomorphic. However, $\mathcal{C}_1$ is isomorphic to $\mathcal{C}_3$.

We have $\text{P-Aut}(\mathcal{C}) = \langle (1, 12)(2, 9)(3, 10)(4, 11) \rangle$: it has order 2 and the unique non-trivial automorphism is acting as the identity on both $\mathcal{C}_2$ and $\mathcal{C}_4$, and it is swapping only $\mathcal{C}_1$ (having support $\{1, 2, 3, 4\}$) with $\mathcal{C}_3$ (having support $\{9, 10, 11, 12\}$). Notice that the fact that $\text{P-Aut}(\mathcal{C})$ does not have a swapping automorphism is a confirmation that $\mathcal{A}$ and $\mathcal{B}$ are not isomorphic.

5 Reductions Between Problems on Codes

In this section, we prove that $\text{ISO} \leq_p \text{AUT-Gen}$ and that $\text{AUT} \leq_p \text{ISO}$. While we are not able to prove that AUT and ISO are equivalent, this result suggests that AUT is unlikely to be easy in the worst case. Indeed, remember that ISO is currently deemed hard for codes with a large hull: therefore, we expect that also for these codes AUT is hard⁹.

A summary of the reductions we give in this section is given below.

Theorem 10 *ISO can be reduced in polynomial time to AUT-Gen and AUT-Count, and AUT can be reduced in polynomial time to ISO.*

Remark 7. It is likely that the reductions we give in this section can be extended, with little to no effort, to more general notions of isomorphisms, such as monomial maps. Indeed, we will never explicitly need the isomorphisms (and the automorphisms) to be permutations. Moreover, analogously to the case of isomorphisms, one can map a problem on monomials to a problem on permutations using the closure [SS13]. However, for $q \geq 5$, the resulting code has hull dimension $h = k$ (i.e., it is self-orthogonal) and, as we shall later on, this implies that we do not know about efficient solvers. Therefore, it seems that for $q \geq 5$ finding monomial automorphisms is hard, regardless of the hull dimension. We do not comment further about monomial automorphisms since, as we have already stressed, in this paper we focus just on permutation automorphism.

5.1 Reduction from AUT to ISO

Obviously, for a code $\mathcal{C}$ with repeated coordinates, non-trivial automorphisms exist and can be found in polynomial time¹⁰. We therefore look at the more interesting case of $\mathcal{C}$ not having repeated coordinates. In principles, a non-trivial automorphism may be found by calling a solver for ISO , on input the same $\mathcal{C}$ twice. While this may even work in some cases (i.e., the solver may return a non-trivial permutation mapping $\mathcal{C}$ to $\mathcal{C}$), this is obviously not going to be useful for the case of $\mathcal{C}$ having only the trivial automorphism: by definition, the solver will always return *id*. As a workaround, we aim to modify $\mathcal{C}$ so that we get two

⁹ We remark that this belief about the hardness of ISO , for codes with large hull, is based on the state-of-the-art of known solvers.

¹⁰ Any permutation that swaps two equal columns is a non-trivial automorphism.

codes $\mathcal{A}$ and $\mathcal{B}$ that are different and such that an isomorphism exists only if $\mathcal{C}$ has indeed a non-trivial automorphism. We do this by repeating coordinates, similarly to [BM23].

Theorem 11 *For any finite field and for any $k \leq n \in \mathbb{N}$, $AUT \leq_p ISO$.*

Proof. Let $\mathcal{C}$ be a $[n, k]$ code with no repeated columns. For $i \in \{1, \dots, n\}$, let $\mathcal{C}_i$ be the $[n+1, k]$ code obtained from $\mathcal{A}$ by repeating the i -th column in position $n+1$. If $\varphi \in \text{P-Aut}(\mathcal{C})$ is a non-trivial automorphism we have that there exist $i, j \in \{1, \dots, n\}$, with $i \neq j$, such that $\varphi(i) = j$. Under this condition, we have that $\mathcal{C}_i$ and $\mathcal{C}_j$ are different codes and are isomorphic. Moreover, every isomorphism π such that $\pi(\mathcal{C}_i) = \mathcal{C}_j$ is such that $\sigma(\{i, n+1\}) = \{j, n+1\}$. On the other hand, also the converse is true: if $\pi \in S_{n+1}$ is an isomorphism from $\mathcal{C}_i$ to $\mathcal{C}_j$ then the map φ defined as $\varphi(x) = \pi(x)$ for $x \notin \{i, n+1\}$ and $\varphi(i) = j$ is an automorphism of $\mathcal{A}$. Thus, if $\mathcal{A}$ has at least one non-trivial automorphism, with at most $n(n-1) = O(n^2)$ calls to a solver for ISO (one for each pair (i, j) with $i \neq j$) it is possible to solve AUT.

Finally, notice that if for all $i \neq j$, $\mathcal{C}_i$ is not equivalent to $\mathcal{C}_j$ then $\mathcal{C}$ has only the identity as automorphism. $\square$

5.2 Reductions from ISO

We first give the reductions for the somewhat easier case of indecomposable codes, and later extend them to decomposable codes. While the reductions for indecomposable codes can be seen as a special case of the ones for decomposable codes, we split between the two cases in order to make the treatment easier to follow and more pedagogical. Moreover, since the majority of codes is indeed indecomposable, this avoids overcomplicating the proof for the most common case.

Reductions for indecomposable codes For the reduction to AUT-Count, we exploit the fact that Dec-ISO and ISO are equivalent [BM23] and prove only that Dec-ISO can be reduced in polynomial time to AUT-Count.

Lemma 1. *For any pair of indecomposable codes, defined over any finite field, with any length n and dimension $k \leq n$, $\text{Dec-ISO} \leq_p \text{AUT-Count}$ and $\text{ISO} \leq_p \text{AUT-Gen}$.*

Proof. Let $\mathcal{A}, \mathcal{B}$ be two indecomposable $[n, k]_q$ -codes, with generator matrices $\mathbf{A}$ and $\mathbf{B}$ respectively. Let us now consider the $[2n, 2k]_q$ -code $\mathcal{C} = \mathcal{A} \times \mathcal{B}$. Using Theorem 9 and Corollary 2, we know that

1. if $\mathcal{A}$ and $\mathcal{B}$ are equivalent, then $|\text{P-Aut}(\mathcal{A})| = |\text{P-Aut}(\mathcal{B})|$ and $\text{P-Aut}(\mathcal{C}) \cong (\text{P-Aut}(\mathcal{A}) \times \text{P-Aut}(\mathcal{B})) \rtimes S_2$ and, therefore,

$$|\text{P-Aut}(\mathcal{C})| > |\text{P-Aut}(\mathcal{A})| \cdot |\text{P-Aut}(\mathcal{B})|; \quad (2)$$

2. otherwise, if $\mathcal{A}$ and $\mathcal{B}$ are not equivalent, then $\text{P-Aut}(\mathcal{C}) \cong \text{P-Aut}(\mathcal{A}) \times \text{P-Aut}(\mathcal{B})$ and, therefore,

$$|\text{P-Aut}(\mathcal{C})| = |\text{P-Aut}(\mathcal{A})| \cdot |\text{P-Aut}(\mathcal{B})|. \quad (3)$$

This means that the number of automorphisms of $\mathcal{C}$ can be used as a distinguishing quantity. In fact, if $|\text{P-Aut}(\mathcal{A})| \neq |\text{P-Aut}(\mathcal{B})|$ then $\mathcal{A}$ and $\mathcal{B}$ are not equivalent. Otherwise, if $|\text{P-Aut}(\mathcal{A})| = |\text{P-Aut}(\mathcal{B})|$, then we can only have two cases: either (2) holds (and, then, the codes are equivalent), or (3) holds (and, then, the codes are not equivalent). Therefore, in the worst case, we need to solve three instances of **AUT-Count**: twice for $[n, k]_q$ codes and once for a $[2n, 2k]_q$ code.

For the second reduction, notice that any isomorphism from $\mathcal{A}$ to $\mathcal{B}$ is in the form $\pi \circ \sigma$, for $\sigma \in \text{P-Aut}(\mathcal{A})$ and a fixed isomorphism π , as per Theorem 7. Moreover, if $\mathcal{A}$ and $\mathcal{B}$ are isomorphic, then for every possible generating set A of $\text{P-Aut}(\mathcal{C})$, at least one element $\sigma \in A$ must be swapping, otherwise the automorphism (π^{-1}, π) of Definition 14 would not be generated by the elements in A . We then just need to go through each of the elements of A , which is a subgroup of S_{2n} and, therefore, admits a generating set whose number of elements is upper bounded by a polynomial of $2n$. Deciding if a permutation is swapping can obviously be done in polynomial time. $\square$

Reductions for decomposable codes In practice, we just need to show that an isomorphism between two decomposable codes can be reconstructed by solving at most n^2 instances of **ISO** on indecomposable codes. Each of these instances can then be reduced to either **AUT-Gen** or **AUT-Count**, relying on Lemma 1.

Lemma 2. *For any pair of decomposable codes, defined over any finite field, with any length n and dimension $k \leq n$, $\text{ISO} \leq_p \text{AUT-Count}$ and $\text{ISO} \leq_p \text{AUT-Gen}$.*

Proof. We assume that $\mathcal{A}$ and $\mathcal{B}$ can be written as the Cartesian product of m_1 and m_2 indecomposable codes, respectively. Remember that, as we have already said, finding the components can be done in polynomial time. Notice that m_1 and m_2 can be at most n , since any component code has dimension at least 1. If $m_1 \neq m_2$, then the input codes cannot be isomorphic. Therefore, from now on, we assume $m := m_1 = m_2$, and, without loss of generality, write the codes as¹¹

$$\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_m, \quad \mathcal{B} = \mathcal{B}_1 \times \cdots \times \mathcal{B}_m.$$

For each pair of indices $i, j \in \{1, \dots, m\}$, we solve **ISO** on $(\mathcal{A}_i, \mathcal{B}_j)$. Since $(\mathcal{A}_i, \mathcal{B}_j)$ are indecomposable, we can do this by reducing either to **AUT-Count** or **AUT-Gen**, as we already seen.

If for some i there is no j such that $\mathcal{A}_i$ and $\mathcal{B}_j$ are isomorphic, then $\mathcal{A}$ and $\mathcal{B}$ cannot be isomorphic.

¹¹ According to the definition of decomposability, we may have $\mathcal{A} = \pi(\mathcal{A}_1 \times \cdots \times \mathcal{A}_m)$ for some permutation π . This permutation can be easily computed and one can apply our reduction on $\pi^{-1}(\mathcal{A})$. Analogously for $\mathcal{B}$.

If instead we have at least one isomorphic component for each $\mathcal{A}_i$, we proceed as follows. We split $\{1, \dots, m\}$ into two disjoint partitions, $U_1, \dots, U_s$ and $V_1, \dots, V_s$ (with $s \leq m$) such that:

- for every t , it holds that for every $i \in U_t$, $j \in V_t$, $\mathcal{A}_i$ and $\mathcal{B}_j$ are isomorphic;
- for every $t \neq t'$, it holds that for every $i \in U_t$, $j \in V_{t'}$, $\mathcal{A}_i$ and $\mathcal{B}_j$ are not isomorphic.

Observe that partitions are easy to determine. For instance, we start by setting $U_1 = \{1\}$ and then find all codes $\mathcal{B}_j$ which are isomorphic to $\mathcal{A}_1$. For each such j , we put j in V_1 . Then, we look at $\mathcal{A}_2$ and again solve ISO on all pairs $(\mathcal{A}_2, \mathcal{B}_j)$. If, for some $j \in V_1$, we find that $\mathcal{A}_2$ and $\mathcal{B}_j$ are isomorphic, then include 2 in U_1 and continue with $\mathcal{A}_3$. Indeed, if $\mathcal{A}_2$ is isomorphic to some code $\mathcal{B}_j$ with $j \in V_1$, then we know $\mathcal{A}_2$ is isomorphic also to $\mathcal{A}_1$ and to every code $\mathcal{B}_j$ with $j \in V_1$. If instead $\mathcal{A}_2$ is isomorphic only to codes $\mathcal{B}_j$ with j not in U_1 , we set $U_2 = \{2\}$ and put in V_2 each of such indices j . We then iterate this procedure. In practice, at step i , we reason as follows¹²:

1. assume we already have $U_1, \dots, U_d$ and $V_1, \dots, V_d$ (with $d \leq i$);
2. we start with $j = 1$ and check if $\mathcal{A}_i$ and $\mathcal{B}_j$ are isomorphic; if they are not, we increase j ;
3. if instead $\mathcal{A}_i$ and $\mathcal{B}_j$ are isomorphic, we see if there exists some ℓ such that $j \in V_\ell$. If it exists, then we put i in U_ℓ and continue with the next value of i ;
4. if ℓ does not exist, we set $U_{d+1} = \{i\}$ and $V_{d+1} = \{j\}$ and consider the subsequent values of j . For any j such that $\mathcal{B}_j$ is isomorphic to $\mathcal{A}_i$, we put j in V_{d+1} .

When we are done with this procedure, we then easily get an isomorphism from $\mathcal{A}$ to $\mathcal{B}$: for $t = 1, \dots, s$, we just need to take a permutation sending each of the codes in each U_t to a distinct code in V_t .

Observe that we have just solved ISO on $m^2 \leq n^2$ pairs of indecomposable codes. All the auxiliary steps (i.e., decomposing $\mathcal{A}$ and $\mathcal{B}$, constructing the partitions of $\{1, \dots, m\}$ and composing the final isomorphism) obviously take polynomial time. $\square$

6 The Relation with Graph Automorphisms: the First Reduction

We show how AUT-Gen can be solved through a solver for GraphAUT-Gen. We do this by first considering a direct application of the reduction in [BOS19], for the case of LCD codes. Then, we deal with the case of non-trivial hulls. First, we present an “imperfect” reduction, which is expected to run efficiently when h is small enough and for the wide majority of codes but, unfortunately, can take

¹² We are now assuming that, for every i , there is at least one index j such that $\mathcal{A}_i$ and $\mathcal{B}_j$ are isomorphic. If such index does not exist, as we have already said, then $\mathcal{A}$ and $\mathcal{B}$ are not isomorphic.

exponential time for some input codes. Then, using the same main ideas but relying on a different de-hulling procedure (instead of removing coordinates and/or codewords, we actually add coordinates), we finally get a reduction running in worst case time $O(n^{h+\omega})$ and we present it in Section 7.

6.1 The case of trivial hull

We recall the method introduced in [BOS19] with which one can associate to an LCD code over a finite field an undirected graph, with the edge labels being finite field values.

For $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ a generator matrix for a linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$, we compute the adjacency matrix for the associated graph as

$$\text{Adj}(\mathbf{G}) = \mathbf{G}^\top \cdot (\mathbf{G} \cdot \mathbf{G}^\top)^{-1} \cdot \mathbf{G}.$$

Note that the code being LCD is fundamental since, otherwise, the inverse of $\mathbf{G} \cdot \mathbf{G}^\top$ cannot be defined.

Since $\text{Adj}(\mathbf{G}) = \text{Adj}(\mathbf{S} \cdot \mathbf{G})$ for any $\mathbf{S} \in GL_k(\mathbb{F}_q)$, we have that Adj does not depend on which change of basis matrix $\mathbf{S}$ is used to represent the code. We now prove the opposite direction, i.e., that if two $k \times n$ matrices lead to the same value of Adj , then they differ only by a change of basis matrix $\mathbf{S} \in GL_k(\mathbb{F}_q)$.

Theorem 12 *Let $\mathbf{G}, \mathbf{G}' \in \mathbb{F}_q^{k \times n}$ with rank k be such that both $\mathbf{G}\mathbf{G}^\top$ and $\mathbf{G}'\mathbf{G}'^\top$ are non singular. If $\text{Adj}(\mathbf{G}) = \text{Adj}(\mathbf{G}')$, then $\mathbf{G}' = \mathbf{S} \cdot \mathbf{G}$ for some $\mathbf{S} \in GL_k(\mathbb{F}_q)$.*

Proof. Let $\mathbf{U} = \mathbf{G}^\top \cdot (\mathbf{G} \cdot \mathbf{G}^\top)^{-1}$ and $\mathbf{U}' = \mathbf{G}'^\top \cdot (\mathbf{G}' \cdot \mathbf{G}'^\top)^{-1}$. Notice that $\mathbf{U}, \mathbf{U}' \in \mathbb{F}_q^{n \times k}$ and both have rank k . Since $\text{Adj}(\mathbf{G}) = \text{Adj}(\mathbf{G}')$, we have $\mathbf{U} \cdot \mathbf{G} = \mathbf{U}' \cdot \mathbf{G}'$. Since both matrices have the same rank k , the span of their rows is a k -dimensional subspace of $\mathbb{F}_q^n$. But the rows of each of these matrices are just linear combinations of $\mathbf{G}$ and $\mathbf{G}'$, respectively; moreover, we know that k out of such linear combinations are linearly independent (because $\mathbf{U}$ has rank k). Then, we have that $\mathbf{G}$ and $\mathbf{G}'$ must generate the same k -dimensional space. $\square$

In the next theorem, we prove the relation between automorphisms for a code and automorphisms for the associated graph.

Theorem 13 *Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a linear code with trivial hull and $\mathcal{G}$ be the graph whose adjacency matrix is $\text{Adj}(\mathbf{G})$, where $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ is any generator matrix for $\mathcal{C}$. Then, every permutation automorphism for $\mathcal{C}$ is also an automorphism for $\mathcal{G}$, and viceversa.*

Proof. Let $\mathbf{P} \in \mathbb{F}_q^{n \times n}$ be the permutation matrix associated with an automorphism for $\mathcal{C}$. Then, for any generator matrix $\mathbf{G}$ for $\mathcal{C}$, there must exist a non

singular $\mathbf{S} \in \mathbb{F}_q^{k \times k}$ such that $\mathbf{G} = \mathbf{S} \cdot \mathbf{G} \cdot \mathbf{P}$. We then have

$$\begin{aligned}
\mathbf{A} &= \text{Adj}(\mathbf{G}) = \text{Adj}(\mathbf{S} \cdot \mathbf{G} \cdot \mathbf{P}) \\
&= (\mathbf{SGP})^\top \cdot (\mathbf{SGP} \cdot (\mathbf{SGP})^\top)^{-1} \cdot \mathbf{SGP} \\
&= \mathbf{P}^\top \mathbf{G}^\top \mathbf{S}^\top \cdot (\mathbf{S} \cdot \mathbf{G} \cdot \mathbf{G}^\top \cdot \mathbf{S}^\top)^{-1} \cdot \mathbf{SGP} \\
&= \mathbf{P}^\top \mathbf{G}^\top \mathbf{S}^\top \cdot \mathbf{S}^{-\top} \cdot (\mathbf{G} \cdot \mathbf{G}^\top)^{-1} \cdot \mathbf{S}^{-1} \cdot \mathbf{SGP} \\
&= \mathbf{P}^\top \cdot \underbrace{\mathbf{G}^\top \cdot (\mathbf{G} \cdot \mathbf{G}^\top)^{-1} \cdot \mathbf{G}}_{\text{Adj}(\mathbf{G})=\mathbf{A}} \cdot \mathbf{P} \\
&= \mathbf{P}^\top \cdot \mathbf{A} \cdot \mathbf{P}.
\end{aligned}$$

Hence, $\mathbf{P}$ is also an automorphism for $\mathcal{G}$.

We now prove the other direction, i.e., that an automorphism for the graph is an automorphism for the code, as well. We start from $\mathbf{A} = \mathbf{P}^\top \cdot \mathbf{A} \cdot \mathbf{P}$, and consider that

$$\begin{aligned}
\underbrace{\mathbf{G}^\top \cdot (\mathbf{G} \cdot \mathbf{G}^\top)^{-1} \cdot \mathbf{G}}_{\text{Adj}(\mathbf{G})} &= \mathbf{P}^\top \cdot \mathbf{G}^\top \cdot (\mathbf{G} \cdot \mathbf{G}^\top)^{-1} \cdot \mathbf{G} \cdot \mathbf{P} \\
&= \mathbf{P}^\top \cdot \mathbf{G}^\top \cdot \underbrace{(\mathbf{G} \cdot \mathbf{P} \cdot \mathbf{P}^\top \cdot \mathbf{G}^\top)^{-1}}_{\text{Adj}(\mathbf{G} \cdot \mathbf{P})} \cdot \mathbf{G} \cdot \mathbf{P}.
\end{aligned}$$

Now, recalling Theorem 12, this implies that $\mathbf{G}$ and $\mathbf{G} \cdot \mathbf{P}$ generate the same space (so, there exists a non singular $\mathbf{S} \in \mathbb{F}_q^{k \times k}$ such that $\mathbf{G} = \mathbf{S} \cdot \mathbf{G} \cdot \mathbf{P}$). This implies that $\mathbf{P}$ is an automorphism for the code having $\mathbf{G}$ as generator matrix. $\square$

As an immediate corollary of the previous theorem, for codes with trivial hull we get a reduction from AUT-Gen to GraphAUT-Gen.

Corollary 3. *Let $\mathcal{A}$ be a solver for GraphAUT-Gen, running in worst case time $T_{\mathcal{A}}(n)$, where n indicates the number of vertices of the graph. Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be an LCD code with dimension k and length n . Then, AUT-Gen for $\mathcal{C}$ can be solved in time*

$$O(n^3 + T_{\mathcal{A}}(n)).$$

Proof. The reduction computes $\text{Adj}(\mathbf{G})$ and then calls $\mathcal{A}$ to find generators for the automorphism group of the graph whose adjacency matrix is $\text{Adj}(\mathbf{G})$. We consider a cost of $O(n^3)$ operations over $\mathbb{F}_q$ for the computation of the graph. $\square$

Remark 8. We are treating q as a constant and, therefore, we are not reporting any cost for doing operations over $\mathbb{F}_q$. We also notice that one can replace the term n^3 (which is due to schoolbook matrix multiplication and inversion) with n^ω , with $\omega \approx 2.373$ being the (currently known) exponent for matrix inversion and multiplication. Notice that, however, algorithms achieving complexity better than n^3 are rarely used in practice, because of several practical overheads which are hidden by the asymptotic big O notation (such as huge constants factors and large memory requirements).

Remark 9. Since the code and the obtained graph have the same automorphism group, every problem related to code automorphisms translates to the associated problem for graphs. For instance, for LCD codes, it is true that solving AUT implies solving also AUT-Gen, as it was proven for graphs in [Mat79].

6.2 The case of non-trivial hull: a first solver

We now consider the case in which the hull of the code is non-trivial, i.e., it has dimension $h \geq 1$. In this case, we cannot directly compute adjacency matrices since, for any generator matrix $\mathbf{G}$, we have that $\mathbf{G}\mathbf{G}^\top$ has rank deficiency exactly h . In [BOS19], as a workaround, the authors consider shortenings with respect to information sets for the hull. The idea is that, if $\mathcal{A}, \mathcal{B} \subseteq \mathbb{F}_q^n$, with hull dimension h , are such that $\mathcal{B} = \pi(\mathcal{A})$ for $\pi \in S_n$, then we want to determine two sets $J, U \subseteq \{1, \dots, n\}$ which are information sets for the hulls of $\mathcal{A}$ and $\mathcal{B}$, respectively, and such that $\pi(J) = U$. Once we know these sets, then we can shorten $\mathcal{A}$ in J and $\mathcal{B}$ in U : the resulting codes are LCD and are isomorphic, hence we can reduce to GIP using adjacency matrices as before. To determine a valid pair of subsets, one fixes J and brute forces over all the possible choices for U . As an additional overhead, one has to reconstruct also the isomorphism

While it would be tempting to replicate the very same procedure, in our case it seems that this is not a good idea. Indeed, the shortening (or puncturing) of a code may modify significantly its automorphism group, by removing automorphisms as well as by introducing spurious automorphisms. Without entering into further details, we give some examples to show what can go wrong.

Example 5. Let us consider $q = 29$, $n = 6$, $k = 3$, $h = 1$, and the code $\mathcal{C} \subseteq \mathbb{F}_{29}^6$ having the following generator matrix

$$\mathbf{G} = \begin{pmatrix} 19 & 18 & 14 & 22 & 10 & 23 \\ 14 & 13 & 10 & 27 & 3 & 0 \\ 18 & 27 & 4 & 25 & 2 & 9 \end{pmatrix}.$$

Its automorphism group is $\text{P-Aut}(\mathcal{C}) = \langle (1, 5, 4)(2, 3, 6) \rangle$. Every shortening on one position yields a code with only trivial automorphisms.

Example 6. Even puncturing can lead to significant modifications to the automorphism group. For instance, let us take $q = 17$, $n = 4$, $k = 3$ and $h = 1$, and the code generated by the following generator matrix:

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 13 \\ 0 & 0 & 1 & 14 \end{pmatrix}.$$

This code has a trivial automorphism group. However, if for instance we puncture it in the last coordinate, the resulting code is the full space and, therefore, has the symmetric group S_3 as automorphism group.

Combining shortenings with the cartesian product The previous examples show that shortenings (or puncturings) alone are not enough. However, these operations can be combined with the tools and ideas we have already used in the previous section.

We first prove that, for every automorphism sending J to U , we have an automorphism of $\mathcal{C}_{-J} \times \mathcal{C}_{-U}$.

Theorem 14 *Let $\mathcal{C} \subseteq \mathbb{F}_q^n$, with hull of dimension h and automorphism group $\text{P-Aut}(\mathcal{C})$. Let $J \subseteq \{1, \dots, n\}$ be an information set for $\mathcal{H}(\mathcal{C})$. Let $\sigma \in \text{P-Aut}(\mathcal{C})$ such that $\sigma(J) = U$. Let $\mathcal{C}_{-J}$ and $\mathcal{C}_{-U}$ be the shortenings of $\mathcal{C}$ on sets J and U , respectively. Then, $(\sigma^{-1}, \sigma) \in \text{P-Aut}(\mathcal{C}_{-J} \times \mathcal{C}_{-U})$ is an automorphism for the Cartesian product code $\mathcal{C}_{-J} \times \mathcal{C}_{-U}$. Moreover, the code $\mathcal{C}_{-J} \times \mathcal{C}_{-U}$ has trivial hull.*

Proof. Recall that $\mathcal{C}_{-J} := \{\mathbf{c} \in \mathcal{C} \mid \mathbf{c}_j = 0 \ \forall j \in J\}$. Then, since σ maps J to U , it is easy to see that $\sigma(\mathcal{C}_{-J}) \subseteq \mathcal{C}_{-U}$: in particular, this implies $|\mathcal{C}_{-J}| \leq |\mathcal{C}_{-U}|$. By performing the same reasoning for σ^{-1} , we obtain that $\sigma^{-1}(\mathcal{C}_{-U}) \subseteq \mathcal{C}_{-J}$ and $|\mathcal{C}_{-U}| \leq |\mathcal{C}_{-J}|$. Therefore, it follows that σ is an isomorphism between $\mathcal{C}_{-J}$ and $\mathcal{C}_{-U}$ and that, by Theorem 6, (σ^{-1}, σ) is an automorphism of the code $\mathcal{C}_{-J} \times \mathcal{C}_{-U}$. Lastly, since σ is an automorphism, $U = \sigma(J)$ is also an information set of $\mathcal{H}(\mathcal{C})$. Then, $\mathcal{C}_{-J} \times \mathcal{C}_{-U}$ has trivial hull by Theorem 8, since both $\mathcal{C}_{-J}$ and $\mathcal{C}_{-U}$ have trivial hull ([BOS19, Proposition 9]). $\square$

As a simple corollary, we get the following result.

Corollary 4. *Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ with hull dimension $h > 0$ and let $J \subseteq \{1, \dots, n\}$ be an information set of $\mathcal{H}(\mathcal{C})$, $|J| = h$. Let $\text{P-Iso}(\mathcal{C}_{-J}, \mathcal{C}_{-U})$ be the set of all isomorphisms from $\mathcal{C}_{-J}$ to $\mathcal{C}_{-U}$, i.e.,*

$$\text{P-Iso}(\mathcal{C}_{-J}, \mathcal{C}_{-U}) = \{\sigma \in S_n \mid \sigma(\mathcal{C}_{-J}) = \mathcal{C}_{-U}\}.$$

Then, $\text{P-Aut}(\mathcal{C}) \subseteq \bigcup_{\substack{U \subseteq \{1, \dots, n\} \\ |U|=h}} \text{P-Iso}(\mathcal{C}_{-J}, \mathcal{C}_{-U})$.

Proof. Because of Theorem 14, every automorphism sending J to some U is turned into an isomorphism from $\mathcal{C}_{-J}$ to $\mathcal{C}_{-U}$. Enumerating all possible sets U , we enumerate all images that J can have: therefore, every automorphism of $\mathcal{C}$ must be contained in $\bigcup_{\substack{U \subseteq \{1, \dots, n\} \\ |U|=h}} \text{P-Iso}(\mathcal{C}_{-J}, \mathcal{C}_{-U})$. $\square$

Remark 10. Corollary 4 can be further refined by considering only the subsets $U \subseteq \{1, \dots, n\}$ of size h such that $\mathcal{C}_{-U}$ has trivial hull.

Building on the above results, we have Algorithm 1. The idea is to turn the automorphism search into an isomorphism search: for a chosen J , information set for $\mathcal{H}(\mathcal{C})$, we enumerate all its candidate images U (under the action of automorphisms) and then, for each pair (J, U) , find all isomorphisms from $\mathcal{C}_{-J}$ to $\mathcal{C}_{-U}$. To do this, we rely on the Cartesian product and find all swapping

Algorithm 1: From AUT-Gen to GraphAUT-Gen (Reduction I)

Input: Linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with hull dimension h
Output: Automorphism group of the input code

```

1 Set  $\text{P-Aut}(\mathcal{C}) = \{id\}$ ; /* Initialize P-Aut( $\mathcal{C}$ ) */
2 Set  $J \subseteq \{1, \dots, n\}$ : information set for  $\mathcal{H}(\mathcal{C})$ ;
3 Compute  $\mathcal{A} := \mathcal{C}_{-J}$ ; /* Shortening of  $\mathcal{C}$  on  $J$  */
4 for  $U \subseteq \{1, \dots, n\}$ ,  $U$  information set for  $\mathcal{H}(\mathcal{C})$  do
5     Compute  $\mathcal{B} := \mathcal{C}_{-U}$ ; /* Shortening of  $\mathcal{C}$  on  $U$  */
6     /* Reduce to GraphAUT-Gen via Cartesian product code */
7     Compute  $\{\phi_1, \dots, \phi_m\}$ : generators for  $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$ ;
8     /* Check each of the found automorphisms */
9     for  $\sigma \in \langle \phi_1, \dots, \phi_m \rangle$  do
10        if  $\sigma$  is swapping then
11            Set  $\varphi = \text{Isolate}(\sigma)$ ; /* Length- $n$  permutation */
12            if  $\varphi(\mathcal{C}) = \mathcal{C}$  then
13                Update  $\text{P-Aut}(\mathcal{C}) \leftarrow \text{P-Aut}(\mathcal{C}) \cup \{\varphi\}$ ;
14 return  $\text{P-Aut}(\mathcal{C})$ 

```

automorphisms via the reduction to GraphAUT-Gen. Because of Corollary 4, this guarantees that the algorithm finds all automorphisms of $\mathcal{C}$.

Observe that, as a nuisance, we cannot guarantee that each isomorphism from $\mathcal{C}_{-J}$ to $\mathcal{C}_{-U}$ is associated with an automorphism of $\mathcal{C}$. For instance, consider that since both $\mathcal{C}_{-J}$ and $\mathcal{C}_{-U}$ have at least h null coordinates (in the positions indexed by J and U , respectively), if there is indeed an automorphism sending J to U , then we have at least $h!$ associated isomorphisms from $\mathcal{A}$ to $\mathcal{B}$. However, we cannot guarantee that they will all correspond to automorphisms of $\mathcal{C}$. More generally, it can also happen that $\mathcal{C}_{-J}$ and $\mathcal{C}_{-U}$ are isomorphic even if there is no automorphism sending J to U .

Therefore, we need to test each of the found automorphisms: we do this in lines 7–10 of the algorithm. This can be done efficiently: for instance, it is enough to check that $\varphi(\mathbf{G})\mathbf{H}^\top = \mathbf{0}$, with $\mathbf{G}$ and $\mathbf{H}$ being any generator and parity-check matrices for $\mathcal{C}$, respectively. We remark that we only need to consider swapping automorphisms, since they are directly associated with isomorphisms from $\mathcal{C}_{-J}$ to $\mathcal{C}_{-U}$. The procedure `Isolate` is used to reconstruct the length- n permutation automorphism from the length- $2n$ automorphism: this can be done, for instance, by looking at the matrix representation of σ and considering only its top right block.

Remark 11. Algorithm 1 can be improved in several ways. For example the coordinates of J and U in $\mathcal{C}_{-J}$ and $\mathcal{C}_{-U}$, respectively, can be deleted. In this way the product code will be a $2(n-h)$ long code and the search for automorphisms will not be impacted by the $(2h)!$ permutations of the $2h$ zero coordinates. In this case, a remapping of coordinates is needed. Note also that the algorithm outputs

the entire automorphism group of the code, that, in some cases, can have exponential cardinality. Hence, at line 11 of the algorithm, a procedure that keeps only a set of generators and removes the other elements can be added. This procedure can be executed in polynomial time in n [Ser03] and ensures that the algorithm output has polynomial size.

Our aim is not to present an optimized algorithm, but a more intuitive version. Nonetheless, notice that these optimizations only yield very mild improvements.

Computational complexity Estimating the time complexity of Algorithm 1 with a closed-form formula turns out to be a rather convoluted task. Indeed, a computational bottleneck may be the loop in lines 7–11, which goes through a number of iterations that is

$$\bigcup_{\substack{U \subseteq \{1, \dots, n\}, |U|=h \\ U \text{ information set for } \mathcal{H}(\mathcal{C})}} |\text{P-Aut}(\mathcal{C}_{-J} \times \mathcal{C}_{-U})|. \quad (4)$$

Notice that this quantity is dependant on the special structure of the code and, likely, is also related to its automorphism group, as well; so, it looks like the problem is circular.

Yet, one can identify two major causes that may make the algorithm rather slow:

- *large automorphism group*: since Algorithm 1 goes through each automorphism of $\mathcal{C}$, the cost cannot be less than $|\text{P-Aut}(\mathcal{C})|$. Actually, there is some overhead due to how the reduction works. Indeed, if $\mathcal{C}$ has an automorphism φ sending J to U , then there is an isomorphism from $\mathcal{C}_{-J}$ to $\mathcal{C}_{-U}$ which, in turns, is associated with at least one automorphism of $\mathcal{C}_{-J} \times \mathcal{C}_{-U}$. Taking into account the fact that the product code has $2h$ null coordinates, we have that φ is actually associated with $(2h)!$ automorphisms. Neglecting that we may actually do some overcounting, we can estimate the number of such automorphisms as $(2h)! \cdot |\text{P-Aut}(\mathcal{C})|$;
- *large h* : the number of information sets of a given code (unless the code is extremely peculiar) grows as $\binom{n}{h}$. Then, when h is large, this quantity may be extremely high.

The above reasoning already tells us that Algorithm 1 cannot be efficient whenever the automorphism group of the code is too large (and the hull has dimension at least 1), and/or h is too high.

Example 7. Let n be even and take $\mathcal{C} \subseteq \mathbb{F}_2^n$ as the parity-bit code. Since n is even, it can be easily seen that this code has hull dimension 1. Moreover, its automorphism group is S_n , therefore $|\text{P-Aut}(\mathcal{C})| = n! = 2^{n \cdot \log_2(n/e) \cdot (1+o(1))}$. Running Algorithm 1 on such a code, we would incur in exponential time complexity.

Example 8. To make a less trivial example, let us consider the following code construction. Let $k' < n'$ be two constants. Let us take m codes $\mathcal{C}_1, \dots, \mathcal{C}_m$, each with an automorphism group of order at least $c \geq 2$. Let us choose these codes so that at least one of them has a non-trivial hull. Let $\mathcal{C} = \mathcal{C}_1 \times \dots \times \mathcal{C}_m$: this code has length $n = mn'$, a non-trivial hull, and $|\text{P-Aut}(\mathcal{C})| \geq c^m = c^{n/n'}$, which is an exponential of the code length.

6.3 On exponential-time instances

For large h , say, for h linear in n , Algorithm 1 will run in exponential time regardless of the size of the automorphism group. Instead, when h is small, our algorithm is expected to be rather fast for the wide majority of codes. Indeed, as proved in [LWZ26], the majority of codes have only trivial automorphisms. Moreover, in [Sen97], it is shown that the majority of codes have very small hull: the fraction of codes with hull dimension $h \geq 1$ is (roughly) approximately $q^{-h(h+1)/2}$. Putting the two facts together, we have that the majority of codes are expected to have small hull and small automorphism group. Thus, even if in a rather informal way, we argue that Algorithm 1 will run efficiently for the majority of codes. In Section 8, we corroborate this claim with intensive numerical experiments.

We notice that, however, there may be other causes leading to an exponential time complexity. In particular, there may be codes for which the number of automorphisms in Equation (4) is abnormally large, even when the hull is small and the code has very few automorphisms. Formalizing the properties of such codes turns out to be rather hard and, perhaps, even out of scope for this paper. Indeed, as we show in the next section, we can “repair” Algorithm 1 so that, for small hull, all codes can be handled without incurring in exponential time complexity. We just give an example to show how such codes may look like.

Example 9. We consider q prime and codes where the generator matrix is in the form $\mathbf{G} = \begin{pmatrix} \mathbf{A} \\ 0 \dots 0 \ 1 \dots 1 \end{pmatrix} \in \mathbb{F}_q^{k \times n}$ where, in the last row, we have a run of h zeros followed by a run of $n - h$ ones. Let $\mathbf{A} \in \mathbb{F}_q^{(k-1) \times n}$ generate a self-orthogonal code having $\{1, \dots, k-1\}$ as information set. When q is not a divisor of $n - (k-1)$, then the code generated by $\mathbf{G}$ has hull dimension $h = k-1$: indeed, the last row of $\mathbf{G}$ is not self-orthogonal since its inner product with itself yields $n - (k-1)$, which is different from 0. Using $J = U = \{1, \dots, k-1\}$, we have that $\mathcal{C}_{-J} = \mathcal{C}_{-U}$ yields a 1-dimensional code with automorphism group of size $h!(n-h)!$.

We give a practical example of such a construction by using $q = 17$, $k = 2$ and $n = 7$:

$$\mathbf{G} = \begin{pmatrix} 4 & 5 & 14 & 16 & 7 & 8 & 4 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

The code generated by $\mathbf{G}$ has hull dimension 1 and a trivial automorphism group. However, shortening the code on $J = U = \{1\}$ yields the code with generator matrix $(0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$, which has $6!$ automorphisms.

7 The Relation with Graph Automorphisms: Repairing the Reduction

At a closer look, the possibility that Algorithm 1 incurs in exponential running time is due to the following phenomenon: shortening on J and U can create automorphisms of the product code that are not associated with automorphisms of the initial code. This requires to check each of the found automorphisms and, ultimately, can lead to an exponential time complexity. To handle a large automorphism group, we would more conveniently work with generating sets (which, in our case, have always polynomial size). However, in Algorithm 1 this is not possible since, a priori, we cannot tell how automorphisms of $\mathcal{C}_{-J} \times \mathcal{C}_{-U}$ would look like. Indeed, since shortening eliminates codewords, we may end up creating extra automorphisms.

We circumvent this issue by introducing a novel *de-hulling* procedure: starting from $\mathcal{C}$, we obtain two codes $\mathcal{A}$ and $\mathcal{B}$ by replicating coordinates indexed by J and U , respectively. Using the proper amount of replicas, we can guarantee that the obtained codes have trivial hulls and that they are isomorphic if and only if $\mathcal{C}$ has an automorphism sending J to U . Notice that replicating coordinates creates many artificial automorphisms. However, we already know, a priori, how these automorphisms look like and how they are related with an automorphism of the initial code.

This gives us a way to handle, more efficiently, the artificial automorphisms created by the de-hulling procedure. Namely, starting from a generating set for $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$, we can obtain a (possibly redundant) generating set for a subgroup of $\text{P-Aut}(\mathcal{C})$ just by modifying each of the generators. Technically, if $\{\chi_1, \dots, \chi_t\}$ is a generating set for $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$, we get a generator set for a subgroup of $\text{P-Aut}(\mathcal{C})$ as $\{\text{Refine}(\chi_1), \dots, \text{Refine}(\chi_t)\}$. Here, $\text{Refine}(\chi_1)$ is a polynomial time function which will be defined later on. Notice that we just need to work on each generator individually, without ever materializing $\langle \chi_1, \dots, \chi_t \rangle$.

We remark that the possibility to modify just generators comes, also, from the guarantee that artificial automorphisms must have a certain form (namely, they arise from repeated coordinates). Again, this rather technical but fundamental detail is due to our novel de-hulling procedure: since codewords are not removed, automorphisms cannot be created from scratch and the only artificial automorphisms are obtained by extending the ones that already exist.

7.1 De-hulling by replicating coordinates

The novel de-hulling procedure is given next; an example of the construction of the matrices $\mathbf{A}$ and $\mathbf{B}$ is given in Figure 2.

Theorem 15 *Let $\mathbb{F}_q$ be a finite field with characteristic p . Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a linear code with dimension k and hull dimension h . Let $J = (j_1, \dots, j_h) \subseteq \{1, \dots, n\}$ be an ordered set corresponding to an information set for $\mathcal{H}(\mathcal{C})$, and let $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ be a generator matrix for $\mathcal{C}$. Assume that $\mathbf{G}$ has no repeated columns. Let $U = (u_1, \dots, u_h)$, with $u_i \in \{1, \dots, n\} \forall i \in \{1, \dots, h\}$, be an ordered*

set corresponding to another information set for $\mathcal{H}(\mathcal{C})$. For $i \in \{1, \dots, h\}$, let $\ell_i = i \cdot p + 1$ and let

$$\begin{aligned} \mathbf{A} &= (\mathbf{G} \mid \mathbf{M}_1 \mid \dots \mid \mathbf{M}_h) , \quad \mathbf{M}_i = \underbrace{(\mathbf{g}_{j_i} \mid \mathbf{g}_{j_i} \mid \dots \mid \mathbf{g}_{j_i})}_{\ell_i} \in \mathbb{F}_q^{k \times \ell_i} \\ \mathbf{B} &= (\mathbf{G} \mid \mathbf{W}_1 \mid \dots \mid \mathbf{W}_h) , \quad \mathbf{W}_i = \underbrace{(\mathbf{g}_{u_i} \mid \mathbf{g}_{u_i} \mid \dots \mid \mathbf{g}_{u_i})}_{\ell_i} \in \mathbb{F}_q^{k \times \ell_i} \end{aligned} \quad (5)$$

Let $\mathcal{A} = \text{span}(\mathbf{A})$ and $\mathcal{B} = \text{span}(\mathbf{B})$ and let $\tilde{n} = n + \sum_{i=1}^h \ell_i$, then:

- a) both $\mathcal{A}$ and $\mathcal{B}$ have trivial hulls;
- b) for any $\varphi \in \text{P-Aut}(\mathcal{C})$ sending J to U (namely, $\varphi(j_i) = u_i$ for every $i = 1, \dots, h$), there is an isomorphism $\chi \in S_{\tilde{n}}$ sending $\mathcal{A}$ to $\mathcal{B}$ such that $\chi(\{1, \dots, n\} \setminus J) = \varphi(\{1, \dots, n\} \setminus J)$;
- c) for any isomorphism $\chi \in S_{\tilde{n}}$ sending $\mathcal{A}$ into $\mathcal{B}$, there exists $\varphi \in \text{P-Aut}(\mathcal{C})$ such that $\varphi(\{1, \dots, n\} \setminus J) = \chi(\{1, \dots, n\} \setminus J)$;
- d) for any isomorphism $\chi \in S_{\tilde{n}}$ between $\mathcal{A}$ and $\mathcal{B}$, when written as a product of disjoint cycles, there is no cycle moving one index of $\{1, \dots, n\} \setminus J$ to $U \cup \{n+1, \dots, \tilde{n}\}$.

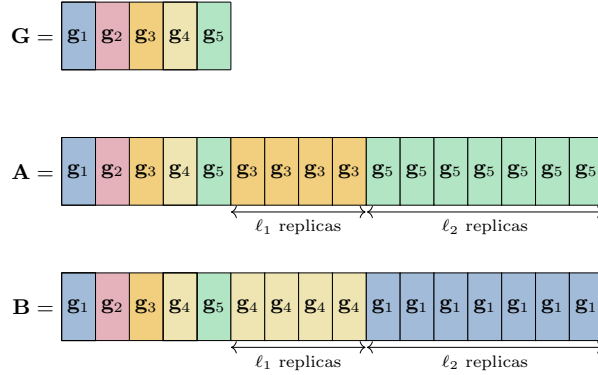


Fig. 2: Example over $\mathbb{F}_3$, with hull dimension $h = 2$ and code length $n = 5$. The field characteristic is $p = 3$. For these parameters, we have $\ell_1 = 4$ and $\ell_2 = 7$. We are using $J = \{3, 5\}$ and $U = (4, 1)$, so, we are assuming existence of an automorphism $\varphi \in S_5$ so that $\varphi(3) = 4$, $\varphi(5) = 1$

Here, we report only the proof of thesis a) since it is the least intuitive among the four theses. The proofs of the remaining theses can be found in Appendix D.

Before presenting the proof of thesis a) of Theorem 15, we introduce the concept of *Standard Hollow Form* for the generator matrix of a code. To this end, we recall a result from [Ben26].

Theorem 16 [Ben26, Corollary 4] *Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ with dimension k and hull dimension $h \leq k$. Then, there exists a permutation $\pi \in S_n$ such that $\pi(\mathcal{C})$ admits a generator matrix of the form*

$$\begin{pmatrix} \mathbf{U} \\ \mathbf{V} \end{pmatrix} = \left(\begin{array}{c|c} \tilde{\mathbf{U}} & \mathbf{0}_{(k-h) \times h} \\ \hline \tilde{\mathbf{V}} & \mathbf{I}_h \end{array} \right), \quad \tilde{\mathbf{U}} \in \mathbb{F}_q^{(k-h) \times (n-h)}, \quad \tilde{\mathbf{V}} \in \mathbb{F}_q^{h \times (n-h)}$$

so that $\mathbf{U}$ generates an LCD code with dimension $k-h$ and $\mathbf{V}$ generates the hull of $\pi(\mathcal{C})$. The permutation π and the matrices $\mathbf{U}$ and $\mathbf{V}$ can be computed in time which is polynomial in n .

From now on, for any $i \in \{1, \dots, h\}$, we denote $\tilde{\mathbf{e}}_i = (\overbrace{0, \dots, 0}^{k-h}, \mathbf{e}_i)^\top$, with $\mathbf{e}_i$ being the i -th elementary vector of length h .

Note that, since π can be easily found, we can first get a matrix as above and then permute back its columns using π^{-1} . This means that, in polynomial time and for any code $\mathcal{C} \subseteq \mathbb{F}_q^n$, we can build a generator matrix $\mathbf{G}$ such that:

- the top $k-h$ rows generate an LCD code with dimension $k-h$;
- the bottom h rows generate $\mathcal{H}(\mathcal{C})$;
- there is a set $J \subseteq \{1, \dots, n\}$ with $|J| = h$ which is an information set for the hull and such that $\mathbf{G}_J = (\tilde{\mathbf{e}}_1, \dots, \tilde{\mathbf{e}}_h)$.

Since matrices of this form will be rather handy in the following proof, we will refer to a matrix with the above properties as *Standard Hollow Form (SHF) with respect to J* . Observe that, for the same code, we can have multiple generator matrices in this form, even with respect to the same information set for the hull.

Proof (Proof of thesis a of Theorem 15). We first prove that $\mathbf{A}$ generates a code having trivial hull. Since J is an information set for the hull of $\mathcal{C}$, we can compute a generator matrix $\tilde{\mathbf{G}}$ for $\mathcal{C}$ in SHF with respect to J , i.e. $\mathbf{G}_J = (\tilde{\mathbf{e}}_1 \mid \dots \mid \tilde{\mathbf{e}}_h)$ and

$$\tilde{\mathbf{G}} = \begin{pmatrix} \mathbf{U} \\ \mathbf{V} \end{pmatrix}$$

where $\mathbf{U}$ generates an LCD code of dimension $k-h$ and $\mathbf{V}$ generates $\mathcal{H}(\mathcal{C})$. Since $\mathbf{U}$ generates an LCD code, we have $\text{rank}(\mathbf{U}\mathbf{U}^\top) = k-h$. Moreover, since $\mathbf{V}$ generates the hull of $\mathcal{C}$, $\mathbf{V}\mathbf{U}^\top = \mathbf{0}$ and $\mathbf{V}\mathbf{V}^\top = \mathbf{0}$. Let

$$\tilde{\mathbf{A}} = \left(\tilde{\mathbf{G}} \mid \tilde{\mathbf{M}}_1 \mid \dots \mid \tilde{\mathbf{M}}_h \right) = \left(\begin{array}{c|c|c|c|c} \mathbf{U} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \hline \mathbf{V} & \mathbf{E}_1 & \mathbf{E}_2 & \dots & \mathbf{E}_h \end{array} \right),$$

with $\widetilde{\mathbf{M}}_i = (\overbrace{\tilde{\mathbf{e}}_i \mid \cdots \mid \tilde{\mathbf{e}}_i}^{\ell_i}) = \begin{pmatrix} \mathbf{0}_{(k-h) \times \ell_i} \\ \mathbf{E}_i \end{pmatrix}$, where $\mathbf{E}_i$ is the $h \times \ell_i$ matrix where the i -th row has only ones and the other rows have only zeros. Then,

$$\begin{aligned} \widetilde{\mathbf{A}}\widetilde{\mathbf{A}}^\top &= \left(\begin{array}{c|c|c|c|c} \mathbf{U} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \hline \mathbf{V} & \mathbf{E}_1 & \mathbf{E}_2 & \cdots & \mathbf{E}_h \end{array} \right) \begin{pmatrix} \mathbf{U}^\top & \mathbf{V}^\top \\ \hline \mathbf{0} & \mathbf{E}_1^\top \\ \hline \mathbf{0} & \mathbf{E}_2^\top \\ \hline \vdots & \vdots \\ \hline \mathbf{0} & \mathbf{E}_h^\top \end{pmatrix} \\ &= \left(\begin{array}{c|c} \mathbf{U}\mathbf{U}^\top & \mathbf{U}\mathbf{V}^\top \\ \hline \mathbf{V}\mathbf{U}^\top & \mathbf{V}\mathbf{V}^\top + \sum_{i=1}^h \mathbf{E}_i\mathbf{E}_i^\top \end{array} \right) \\ &= \left(\begin{array}{c|c} \mathbf{U}\mathbf{U}^\top & \mathbf{0} \\ \hline \mathbf{0} & \sum_{i=1}^h \mathbf{E}_i\mathbf{E}_i^\top \end{array} \right). \end{aligned}$$

Since $\mathbf{U}\mathbf{U}^\top$ has full rank $k - h$, the rank of $\widetilde{\mathbf{A}}\widetilde{\mathbf{A}}^\top$ is $k - h + \text{rank}(\sum_{i=1}^h \mathbf{E}_i\mathbf{E}_i^\top)$. The matrix $\mathbf{E}_i\mathbf{E}_i^\top$ is such that all entries are null, apart from the one in the i -th row and i -th column (so, $\mathbf{E}_i\mathbf{E}_i^\top$ is a diagonal matrix), which is equal to

$$\langle \underbrace{(1, \dots, 1)}_{\ell_i \text{ times}}; \underbrace{(1, \dots, 1)}_{\ell_i \text{ times}} \rangle = \ell_i \pmod{p}.$$

By construction, $\ell_i = i \cdot p + 1$ hence $\ell_i = 1 \pmod{p}$. Then, $\sum_{i=1}^h \mathbf{E}_i\mathbf{E}_i^\top = \mathbf{I}_h$ and has rank h : this implies that the rank of $\widetilde{\mathbf{A}}\widetilde{\mathbf{A}}^\top$ is k so the hull dimension of the code generated by $\mathbf{A}$ is 0.

Notice that, since $\mathbf{G}$ and $\widetilde{\mathbf{G}}$ generate the same code, there exists a non-singular matrix $\mathbf{S} \in GL_k(\mathbb{F}_q)$ such that $\mathbf{G} = \mathbf{S}\widetilde{\mathbf{G}}$. Moreover, it can be easily seen that $\mathbf{A} = \mathbf{S}\widetilde{\mathbf{A}}$: in particular, this implies that also the code generated by $\mathbf{A}$ has trivial hull.

We now look at $\mathbf{B}$. Since U is an information set for the hull, we can compute the SHF of $\mathbf{G}$ with respect to U . Then, with the same arguments, we can prove that $\mathbf{B}$ generates a code having trivial hull. $\square$

7.2 Refining automorphisms of de-hulled codes

We now state how to practically obtain an automorphism of $\mathcal{C}$ starting from an automorphism of $\mathcal{A} \times \mathcal{B}$. In this section, we restrict our attention to indecomposable codes. Hereafter, we say that an automorphism of $\mathcal{A} \times \mathcal{B}$ is *trivial* if it is equal to the identity id or if it only permutes repeated columns. Hence, if σ is a non-trivial automorphism of $\mathcal{A} \times \mathcal{B}$, then there are indices i and j such that $\sigma(i) = j$ and $\mathbf{g}_i \neq \mathbf{g}_j$.

Corollary 5. *Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be an indecomposable linear code with dimension k and hull dimension h . Let $J = (j_1, \dots, j_h) \subseteq \{1, \dots, n\}$ be an ordered set*

corresponding to an information set for the hull, and let $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ be a generator matrix for $\mathcal{C}$. Assume that $\mathbf{G}$ has no repeated columns. Let $U = (u_1, \dots, u_h) \subseteq \{1, \dots, n\}$ be an ordered set corresponding to another information set for the hull. Let $\mathcal{A}$ and $\mathcal{B}$ be defined as in Theorem 15. Then, starting from a non-trivial automorphism of $\mathcal{A} \times \mathcal{B}$, we can obtain, in time which is polynomial in n , k and q , a non-trivial automorphism of $\mathcal{C}$.

The core of the idea is that if a code $\mathcal{A}$ has only one column repeated ℓ times, then every automorphism of $\mathcal{A}$ must fix these ℓ columns. This can be easily seen from the matrix representation of the automorphism. For the sake of simplicity, let $\mathbf{g}_1$ be the only repeated column and $\mathbf{g}_1, \dots, \mathbf{g}_\ell$ be the ℓ identical columns. If $\mathbf{G} = \mathbf{S}\mathbf{G}\mathbf{P}$, then, since $\mathbf{S}$ acts “individually” on the columns of $\mathbf{G}$, it must be that $\mathbf{S}\mathbf{g}_i = \mathbf{g}_1$ for all $i \in \{1, \dots, \ell\}$, because $\mathbf{g}_1, \dots, \mathbf{g}_\ell$ are the only identical columns in $\mathbf{G}$. Hence, $\mathbf{P}$ must fix these ℓ columns to correctly recover $\mathbf{G}$. Moreover, notice also that if π is an automorphism and σ' is a permutation of $\{1, \dots, \ell\}$ then, $\pi \circ \sigma$, where σ is such that $\sigma(i) = \sigma'(i)$ if $1 \leq i \leq \ell$ and $\sigma(i) = i$ otherwise, is an automorphism. This is a consequence of the fact that a permutation swapping two equal columns is an automorphism of the code.

Proof. Let σ be an automorphism of $\mathcal{A} \times \mathcal{B}$. By the structure of the product code, σ can be component-wise automorphism or a swapping automorphism.

- If σ is a component-wise automorphism, let $\sigma^{\mathcal{A}}$ be the restriction of σ to the first $\tilde{n}$ components. By definition, $\sigma^{\mathcal{A}}$ is an automorphism of $\mathcal{A}$. Then, consider the image of $\sigma^{\mathcal{A}}$ on the first n components. If this is a permutation, then $\sigma^{\mathcal{A}}_{\{1, \dots, n\}}$ is an automorphism of $\mathcal{C}$ since $\sigma^{\mathcal{A}}$ has a block-diagonal structure as a permutation matrix, leaving invariant the block of $\mathbf{A}$ given by $\mathbf{G}$. Suppose instead that $\sigma^{\mathcal{A}}(\{1, \dots, n\}) \neq \{1, \dots, n\}$. Notice that, for all j_i , the image $\sigma^{\mathcal{A}}(j_i)$ is either j_i or one of the ℓ_i copies of $\mathbf{g}_{j_i}$ appended to the end (and vice-versa, the pre-image of j_i is either j_i itself or one of the ℓ_i copies). Indeed, since $\mathbf{g}_{j_i}$ is the only column repeated $\ell_i + 1$ times every automorphism of $\mathcal{A}$ must fix these column. Then, define σ' as $\sigma^{\mathcal{A}}$ on $\{1, \dots, n\} \setminus J$ and as the identity on J . Thanks to this observation and Theorem 15 we have that σ' is an automorphism of $\mathcal{C}$. The same can be done defining $\sigma^{\mathcal{B}}$ as the restriction of σ to the last $\tilde{n}$ components, obtaining an automorphism σ'' of $\mathcal{C}$.
- If σ is a swapping automorphism, there are an isomorphism π from $\mathcal{A}$ to $\mathcal{B}$ and an isomorphism χ from $\mathcal{B}$ to $\mathcal{A}$ such that $\sigma = (\chi, \pi)$. With a similar argument as the previous point, we must have that $\pi(j_i) = u_i$ or one of the ℓ_i copies of $\mathbf{g}_{u_i}$ on the right hand side of $\mathbf{B}$, since $\mathbf{g}_{u_i}$ is the only column of $\mathbf{B}$ repeated $\ell_i + 1$ times. An analogous argument holds for χ . Hence, define σ' and σ'' as follows

$$\sigma'(\ell) = \begin{cases} \pi(\ell) & \text{if } \ell \in \{1, \dots, n\} \setminus J, \\ u_i & \text{if } \ell = j_i, \end{cases}$$

$$\sigma''(\ell) = \begin{cases} \chi(\ell) & \text{if } \ell \in \{1, \dots, n\} \setminus U, \\ j_i & \text{if } \ell = u_i. \end{cases}$$

By Theorem 15, we have that both σ' and σ'' are automorphisms of $\mathcal{C}$. Notice that to obtain the claim for σ'' , it suffices to change the role of $\mathcal{A}$ and $\mathcal{B}$ in the proof. $\square$

We call **Refine** the polynomial-time procedure returning a *pair* of automorphisms of $\mathcal{C}$ from a non-trivial element of $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$. Namely, following the notation from the proof of Corollary 5, we obtain

$$\text{Refine}(\sigma) = \{\sigma', \sigma''\}$$

both when σ is component-wise or swapping.

7.3 The reduction for indecomposable codes

We first give the reduction for indecomposable codes. Then, we show how it can be used as a subroutine for the more general (and, arguably, less interesting) case of decomposable codes. The reduction from **AUT-Gen** to **GraphAUT-Gen** is detailed in Algorithm 2.

The idea is to fix an information set J for the hull and then enumerate all possible images of J under automorphisms. Notice that, once we have chosen an information set J , we can compute the matrix $\mathbf{A}$ as in Theorem 15. Then, for each possible image $U \subseteq \{1, \dots, n\}$, $|U| = h$ of J , we reconstruct the matrix $\mathbf{B}$ according to Theorem 15. Notice that, for each candidate U , we perform these steps only if U is also an information set for the hull since, otherwise, it cannot be the image of J under an automorphism.

Then, we construct the Cartesian product code of $\mathcal{A} = \text{span}(\mathbf{A})$ and $\mathcal{B} = \text{span}(\mathbf{B})$, which has trivial hull because the component codes are LCD, and find a generating set of $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$. Finally, we use the procedure described in the proof of Corollary 5 to recover automorphisms of $\mathcal{C}$ from those of $\mathcal{A} \times \mathcal{B}$.

After each iteration over the information sets U of $\mathcal{H}(\mathcal{C})$, we update the generating set **Gen** removing the redundant elements. This can be done in time polynomial in n [Ser03] and we call this procedure **Prune**.

The time complexity of the Reduction in Algorithm 2 is given below.

Proposition 1. *Let $\mathcal{A}$ be a solver for **GraphAUT-Gen**, running in time $T_{\mathcal{A}}(n)$ where n indicates the number of vertices of the graph. Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a linear code with the hull of dimension h . Then, **AUT-Gen** for $\mathcal{C}$ can be solved in time*

$$O\left(\binom{n}{h} h! \cdot \text{poly}(n) \cdot T_{\mathcal{A}}(\tilde{n})\right),$$

where $\tilde{n} = n + \sum_{i=1}^h (ip + 1)$ and $\text{poly}(n)$ is a polynomial of n .

Proof. Correctness. Let $\mathcal{C}$ be an $[n, k]$ code over $\mathbb{F}_q$ with hull dimension h . Let **Gen** be the set of generators of $\text{P-Aut}(\mathcal{C})$ which is updated in the algorithm. If J and U are information sets for the hull $\mathcal{H}(\mathcal{C})$, the codes generated by $\mathbf{A}$

Algorithm 2: Reduction from AUT-Gen to GraphAUT-Gen, indecomposable codes

Data: Linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with hull dimension $h > 0$ via a generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$

Result: a generating set of $\text{P-Aut}(\mathcal{C})$

```

1 Set  $\text{Gen} = \{id\}$ ;
2 Set  $J \subseteq \{1, \dots, n\}$ : information set for  $\mathcal{H}(\mathcal{C})$ ;
3 Compute  $\mathbf{A} \in \mathbb{F}_q^{k \times \tilde{n}}$  as in Theorem 15;
4 for  $U = (u_1, \dots, u_h) \subseteq \{1, \dots, n\}$  do
5   if  $U$  is an information set for  $\mathcal{H}(\mathcal{C})$  then
6     Compute  $\mathbf{B} \in \mathbb{F}_q^{k \times \tilde{n}}$  as in Theorem 15;
7     Set  $\mathbf{D} = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix}$  // Generator matrix for cartesian product code
8     /* Solve GAP on the graph having adjacency matrix  $\text{Adj}(\mathbf{D})$  */
8      $\chi_1, \chi_2, \dots, \chi_t \leftarrow \text{GraphAutGen}(\text{Adj}(\mathbf{D}))$ ;
9     /* Update and prune the generating set */
9      $\text{Gen} \leftarrow \text{Prune}(\text{Gen} \cup \text{Refine}(\chi_1) \cup \dots \cup \text{Refine}(\chi_t))$ 
10 return Gen
```

and $\mathbf{B}$ using the procedure from Theorem 15 have trivial hull. Hence, their Cartesian product $\mathcal{A} \times \mathcal{B}$ has trivial hull, and we can find a generating set of its automorphisms group $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$ using the reduction to GraphAUT-Gen from Theorem 13. After applying the function **Refine** on every generator, we obtain a set of automorphisms of $\mathcal{C}$ and we update the set Gen adding these new elements of $\text{P-Aut}(\mathcal{C})$ to it and removing the redundant ones.

Observe that the automorphisms obtained through **Refine** can be of two types:

- if the starting automorphism in $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$ is component-wise, then we obtain automorphisms in $\text{P-Aut}(\mathcal{C})$ fixing J , U or both;
- if the starting automorphism in $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$ is swapping and $J \neq U$, we obtain an automorphism in $\text{P-Aut}(\mathcal{C})$ sending J to U .

Since we cycle over all possible images U of the information set of the hull J , we recover all the automorphisms of $\mathcal{C}$.

Complexity. The number of ordered information sets of the hull is at most $\binom{n}{h}h!$ which equals the number of iterations in our reduction. Checking whether a U is an information set costs n^3 . Computing $\mathbf{A}$ and $\mathbf{B}$ has a complexity linear in $\tilde{n} = O(n)$, and the same happens for the cost of **Refine**. The maximal number of generators returned by GraphAUT-Gen is polynomial in $\tilde{n}$. Pruning the non redundant ones is again polynomial in n . Hence, the total cost of Algorithm 2 is

$$O\left(\binom{n}{h}h! \cdot \text{poly}(n) \cdot T_{\mathcal{A}}(\tilde{n})\right).$$

□

7.4 Sketch of the reduction for decomposable codes

When the code $\mathcal{C}$ is decomposable, the procedure above must be generalized; we outline the main idea below.

Suppose that $\mathcal{C}$ can be written as $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_m$, where each $\mathcal{C}_i$ is an indecomposable component. First, using the procedure above, we find the automorphism group of each of the components.

Then, we have two possible cases:

- If all components are pairwise non-isomorphic, then $\text{P-Aut}(\mathcal{C}) = \text{P-Aut}(\mathcal{C}_1) \times \cdots \times \text{P-Aut}(\mathcal{C}_m)$.
- Suppose that some components are isomorphic. For instance, assume that $\{\mathcal{C}_{j_1}, \dots, \mathcal{C}_{j_s}\}$, with $\{j_1, \dots, j_s\} \subseteq \{1, \dots, m\}$, are pairwise isomorphic. In this case, $\text{P-Aut}(\mathcal{C})$ also contains automorphisms that permute these components while keeping the others fixed. To construct these, we must find an isomorphism from $\mathcal{C}_{j_l}$ to $\mathcal{C}_{j_t}$, for every pair (j_l, j_t) with $j_l \neq j_t$ in $\{j_1, \dots, j_s\}$. For each pair, this can be done by computing $\text{P-Aut}(\mathcal{C}_{j_l} \times \mathcal{C}_{j_t})$ and considering a swapping automorphism. As we show later, this can be done in worst case time $O\left(\binom{n}{h} h! \cdot \text{poly}(n) \cdot T_{\mathcal{A}}(\tilde{n})\right)$ (relying on Proposition 1). Then, to a set of generators for $\text{P-Aut}(\mathcal{C}_1) \times \cdots \times \text{P-Aut}(\mathcal{C}_m)$, we need to add generators for automorphisms that swap the components indexed by $\{j_1, \dots, j_s\}$. This can be done efficiently, as we argue next. First, we just need to determine a set of generators for all permutations fixing $\{1, \dots, m\} \setminus \{j_1, \dots, j_s\}$ (in other words, all permutations moving only $\{j_1, \dots, j_s\}$). Let W_i be the support of the i -th code $\mathcal{C}_i$; then, we “expand” each of the generators as follows:
 - if it is moving i to $u \neq i$ (then, it must be $i, u \in \{j_1, \dots, j_s\}$), then we make it so that each index in W_i goes to one of the indices in W_u according to an isomorphism sending $\mathcal{C}_i$ to $\mathcal{C}_u$;
 - if it is stabilizing i (then, it must be $i \notin \{j_1, \dots, j_s\}$), then we make it so that it stabilizes each of indices in W_i .

If we have more than one set of isomorphic components, then we repeat the procedure for each set. Notice that this procedure is almost identical to the one used in the proof of Lemma 2. Therefore, here we only present an intuition instead of a fully detailed steps.

Notice that the above procedure requires to solve **AUT-Gen** for a (worst case) polynomial amount of times. We argue that, when $\mathcal{C}$ has hull dimension h , then for all of the required **AUT-Gen** instances, the maximum hull dimension we need to consider is h . Indeed, each of the components cannot have hull dimension $> h$. To find an isomorphism between two component codes $\mathcal{C}_i$ and $\mathcal{C}_j$, we construct $\mathcal{C}_i \times \mathcal{C}_j$ and then solve **AUT-Gen**. The maximum hull dimension of $\mathcal{C}_i \times \mathcal{C}_j$ can be h . Indeed, if the two codes have different hull dimensions, then they cannot be isomorphic. Therefore, in the worst case, they have the same hull dimension $\lfloor h/2 \rfloor$ and the hull of their product has maximum size $2 \cdot \lfloor h/2 \rfloor \leq h$.

8 Benchmarks

We have developed proof-of-concept implementations of the reduction presented in Section 6.1 for the trivial hull case, and of Algorithm 1 for the non-trivial hull case. The code can be found at

<https://github.com/gdalc/permutation-automorphisms>.

We have considered both **Magma** and **Sagemath**, and have compared with the built-in implementations of Leon’s and Feulner’s algorithms¹³. We have ran intensive numerical simulations considering both codes with a given automorphism group and random codes, as well as the cases of trivial and non-trivial hulls.

In this section we report about the obtained results. As we expected, we show that the reductions to **GraphAUT-Gen** we gave in Section 6 are rather fast whenever the hull dimension is small enough. Also, in the case of non-trivial hulls, we show that, despite the worst-case exponential time complexity of Algorithm 1, in practice the algorithm is, on average, extremely efficient.

In particular, as we expected, for small hull dimension our reductions yield the fastest algorithms for finding permutation automorphisms.

8.1 Codes with trivial hull

Let us first consider codes with trivial hull. To test the algorithm, we have generated codes where, for sure, the automorphism group contains a cyclic permutation group of order k . We have considered only the case $n = 2k$ and have sampled such codes as follows:

1. sample a random $\varphi \in S_n$ which is the product of two disjoint cycles, each of length k ;
2. sample a random $\mathbf{g} \in \mathbb{F}_q^n$, construct $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ so that the i -th row is $\varphi^{i-1}(\mathbf{g})$, where $\varphi^{i-1}(\mathbf{g})$ implies that the permutation φ is applied for $i-1$ times. For $i = 1$, we have $\varphi^{i-1} = \varphi^0 = id$;
3. if $\mathbf{G}$ has rank $< k$, restart from step 1;
4. if $\mathbf{G}$ has hull dimension $h > 1$, restart from step 1. Otherwise, output $\mathbf{G}$.

It can be easily seen that, for each $\mathbf{G}$ as above, we get a code where φ^i for any $i = \{0, \dots, k-1\}$ is an automorphism. Indeed, φ has order k (i.e., the smallest integer m such that $\varphi^m = id$ is k), hence

$$\varphi^i(\mathbf{G}) = \begin{pmatrix} \varphi^i \circ \varphi^0(\mathbf{g}) \\ \varphi^i \circ \varphi^1(\mathbf{g}) \\ \vdots \\ \varphi^i \circ \varphi^{k-1}(\mathbf{g}) \end{pmatrix} = \begin{pmatrix} \varphi^i \bmod k(\mathbf{g}) \\ \varphi^{i+1} \bmod k(\mathbf{g}) \\ \vdots \\ \varphi^{i+k-1} \bmod k(\mathbf{g}) \end{pmatrix},$$

¹³ For **Sagemath**, the method `permutation_automorphism_group`, see https://doc.sagemath.org/html/en/reference/coding/sage/coding/linear_code.html. For **Magma**, the function `PermutationGroup`, see <https://magma.maths.usyd.edu.au/magma/handbook/text/2019>

which is just a cyclic shift of the rows of the initial $\mathbf{G}$.

Therefore, for $\mathcal{C}$ being the code generated by $\mathbf{G}$, we have that

$$\langle \varphi \rangle \subseteq \text{P-Aut}(\mathcal{C}) \implies |\text{P-Aut}(\mathcal{C})| \geq k.$$

For increasing k and several values of q , we have measured the running time of the solver resulting from Corollary 3, and have compared it with the built-in functions of both **Sagemath** and **Magma**. Experiments in **Sagemath** have been performed on a computer with an Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz, with 32 GB RAM and running the Ubuntu 24.04.5 LTS operating system. The experiments in **Magma**, instead, have been conducted on a computer with an Intel(R) Core(TM) Ultra 7 165H (1.40 GHz) with 32 GB RAM and running on the Windows 11 26H2 operating system.

In Figures 3 and 4 we report the measured timings. As we can see, our algorithm is always faster than the built-in implementations. Observe that, in both figures, we have used the logarithmic scale on the y -axis: to emphasize this, in Figure 3 we have also plotted the curves $\alpha \cdot q^k$ and $\beta \cdot k^3$, for some constant α and β ¹⁴. The figures show that, for built-in functions, the required time grows exponentially with k ; for our algorithm, instead, it grows approximately as a polynomial of k . For the experiments in **Magma**, analogous considerations hold.

We remark that, for larger values of q , we have not been able to perform meaningful experiments since the built-in functions were taking an unfeasible amount of time. Instead, for our solver, the dependence on q is rather mild (as q increases, the arithmetic over $\mathbb{F}_q$ gets more costly).

We also observe that, since the resulting running time is essentially polynomial in n , this means also that, on average, the obtained **GraphAUT-Gen** instances can be solved efficiently (say, in polynomial time).

We remark that the different options in **Sagemath** yield very mild differences in timings; therefore, from now on, we will not specify the option with which the **Sagemath** function has been called.

We have also run experiments on random codes, with analogous results; see Figure 5.

Remark 12. **Magma** has a built-in function for finding the automorphism group of an unweighted, undirected graph. However, there is no built-in function for weighted graphs. Consequently, we used a standard workaround suggested to us by the **Magma** group to make the current **Magma** code-base work. However, this strategy slows down the overall algorithm and using an ad-hoc function for weighted graphs would make our solver even faster.

¹⁴ The constants have been chosen in order to have $\alpha \cdot q^k$ and $\beta \cdot k^3$ almost overlap with empirical data. For the two subfigures in Figure 3, the constants are different.

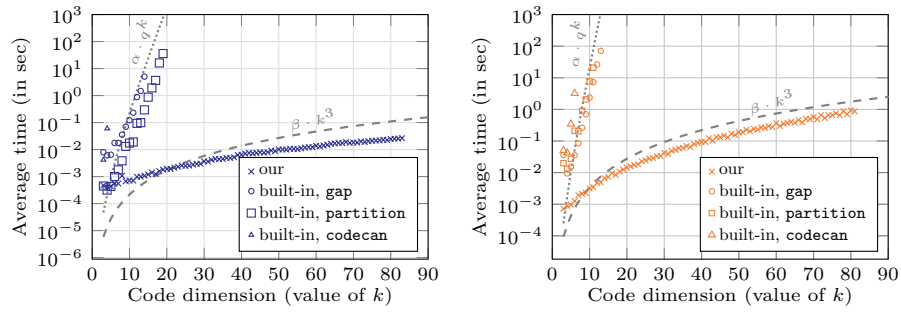


Fig. 3: Timings on **Sagemath** for trivial hull, cyclic permutation group. On the left $q = 2$, while on the right $q = 3$. For each k , we have generated 100 codes. For the built-in function, we have simulated the three different options with which the function can be called.

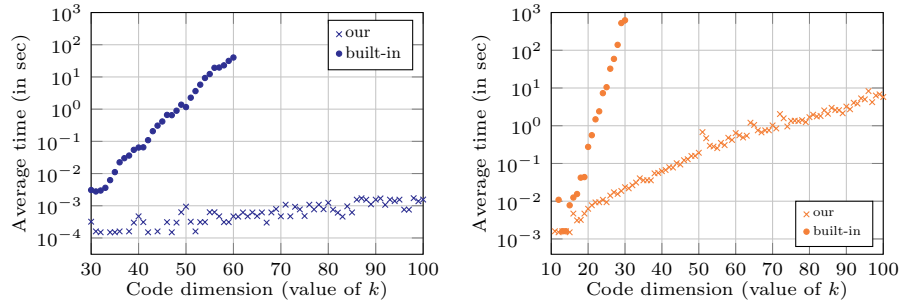


Fig. 4: Timings on **Magma** for trivial hull, cyclic permutation group. On the left $q = 2$, while on the right $q = 5$. For each k , we have generated 100 codes.

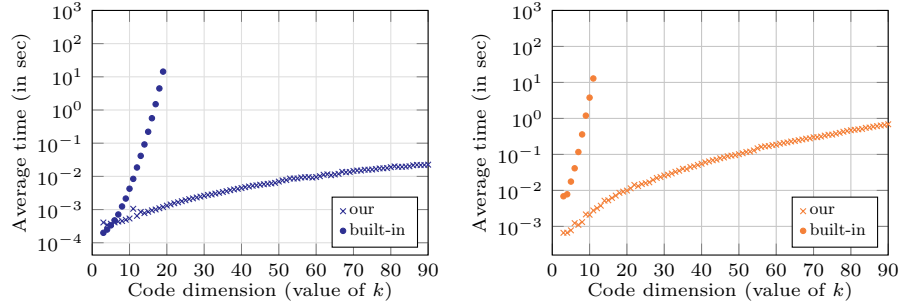


Fig. 5: Timings on **Sagemath** for random codes with trivial hull. For all codes, we have $n = 2k$. On the left $q = 2$, while on the right $q = 3$. For each k , we have generated 100 codes.

8.2 Non-trivial hull

The experiments from the previous section have been repeated for codes with non-trivial hull. In such a case, we have solved **AUT-Gen** using Algorithm 1. We report just about the experiments for **Magma**; results about **Sagemath** are analogous.

Again, it can be seen that our algorithm takes time which is polynomial in n . Notice that, when $h \leq 1$ is constant, the number of calls to a solver for **GraphAUT-Gen** grows as $\binom{n}{h} = O(n^h)$. This behavior is reflected by the experimental timings. Since the benchmarks show that essentially the algorithm takes polynomial time, this is also an implicit confirmation of the fact that the codes described in Section 6.3 are very rare. Indeed, this implies that even if shortening sometimes introduces artificial automorphisms, there are very few such automorphisms. Overall, the computational bottleneck is the enumeration of U and the shortening, taking $O(n^{3+h})$ for constant h .

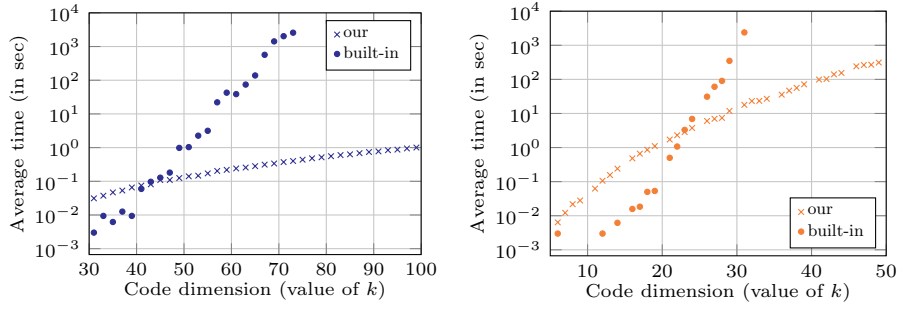


Fig. 6: Timings on **Magma** for hull dimension 1, cyclic permutation group. On the left $q = 2$, while on the right $q = 5$. For each k , we have generated 5 codes.

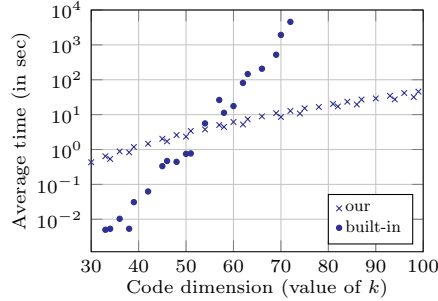


Fig. 7: Timings on **Magma** for hull dimension 2, cyclic permutation group, with $q = 2$. For each k , we have generated 5 codes.

9 Conclusions

In this paper, we discussed the complexity of finding permutation automorphisms of random linear codes. Firstly, we showed that **AUT-Gen** and **AUT-Count**, respectively the problems of finding a generating set and the order of the automorphism group of a code, are not easier than **ISO**. At the same time, we proved that this latter problem is not easier than **AUT**.

Moreover, we defined relations between **AUT-Gen** and **GraphAUT-Gen**, the problem of finding a generating set of the automorphism group of a graph. In particular, we constructed a reduction from **AUT-Gen** to **GraphAUT-Gen** that takes time $O(n^{3+h})$, with h being the dimension of the hull of the code. This shows that, for constant h , all problems related to finding permutation automorphisms of linear codes can be solved, in the worst case, in quasi-polynomial time. Hence, finding permutation automorphisms is easy unless h is very large. Since, however, the majority of the codes have trivial hull (i.e. $h = 0$), this implies that finding permutation automorphisms is on average easy. At the same time, in light of our results, we can conclude that all automorphisms related problems shall currently be deemed hard for codes with sufficiently large hull dimension.

In future works, we plan to extend the set of easy instances by considering known attacks to **ISO** (e.g., doing the square code and computing the hull according to Hermitian inner products) and understanding whether they can be extended to automorphisms problems. At the same time, we also plan to investigate applications of code automorphisms to post-quantum cryptography, either for the design of primitives based on novel assumptions or for the cryptanalysis of existing schemes.

Acknowledgements

The third and fourth authors acknowledge the Italian Ministry of University and Research (MUR) under the Italian Fund for Applied Science (FISA 2022), project “Quantum-safe cryptographic tools for the protection of national data and information technology assets” (QSAFEIT) - No. FISA 2022-00618 (CUP I33C24000520001).

References

- [BES80] László Babai, Paul Erdős, and Stanley M. Selkow. “Random Graph Isomorphism”. In: *SIAM Journal on Computing* 9.3 (1980), pp. 628–635.
- [BBB+24] Marco Baldi, Alessandro Barenghi, Luke Beckwith, et al. “LESS—linear equivalence signature scheme”. In: *Round 2 Additional Signatures to the NIST PostQuantum Cryptography: Digital Signature Schemes Call* (2024).
- [BES+25] Marco Baldi, Rahmi El Mechri, Paolo Santini, et al. “On the representation of self-orthogonal codes and applications to cryptography”. In: *Cryptology ePrint Archive* (2025).

- [BOS19] Magali Bardet, Ayoub Otmani, and Mohamed Saeed-Taha. “Permutation code equivalence is not harder than graph isomorphism when hulls are trivial”. In: *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2019, pp. 2464–2468. DOI: 10.1109/ISIT.2019.8849855.
- [BBP+23] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, et al. “On the computational hardness of the code equivalence problem in cryptography”. In: *Advances in Mathematics of Communications* 17.1 (2023).
- [BMS26] Michele Battagliola, Rocco Mora, and Paolo Santini. “Using the schur product to solve the code equivalence problem”. In: *IEEE Transactions on Information Theory* (2026).
- [Ben26] Benjamin Benčina. *On Local Invariants for Permutation Equivalence*. Cryptology ePrint Archive, Paper 2026/995. 2026. URL: <https://eprint.iacr.org/2026/995>.
- [BC93] Thierry Berger and Pascale Charpin. “The automorphism group of Generalized Reed-Muller codes”. In: *Discrete Mathematics* 117.1 (1993), pp. 1–17. ISSN: 0012-365X. DOI: [https://doi.org/10.1016/0012-365X\(93\)90321-J](https://doi.org/10.1016/0012-365X(93)90321-J). URL: <https://www.sciencedirect.com/science/article/pii/0012365X9390321J>.
- [Beu20] Ward Beullens. “Not enough LESS: an improved algorithm for solving code equivalence problems over F_q ”. In: *International Conference on Selected Areas in Cryptography*. Springer. 2020, pp. 387–403.
- [BHJ26] Jean-François Biasse, Alexandra V Hostetler, and Anuvrat Jaindungarwal. “Code Equivalence and Automorphism Problems for Codes”. In: *arXiv preprint arXiv:2609.25483* (2026).
- [BM23] Jean-François Biasse and Giacomo Micheli. “A Search-to-Decision Reduction for the Permutation Code Equivalence Problem”. In: *2023 IEEE International Symposium on Information Theory (ISIT)* (2023), pp. 602–607. URL: <https://api.semanticscholar.org/CorpusID:261081695>.
- [BCD+24] Markus Bläser, Zhili Chen, Dung Hoang Duong, et al. “On Digital Signatures Based on Group Actions: QROM Security and Ring Signatures”. In: *Post-Quantum Cryptography*. Ed. by Markku-Juhani Saarinen and Daniel Smith-Tone. Cham: Springer Nature Switzerland, 2024, pp. 227–261. ISBN: 978-3-031-62743-9. DOI: 10.1007/978-3-031-62743-9_8.
- [Bou00] S. Bouyuklieva. “A method for constructing self-dual codes with an automorphism of order 2”. In: *IEEE Transactions on Information Theory* 46.2 (2000), pp. 496–504. DOI: 10.1109/18.825812.
- [BMW10] Stefka Bouyuklieva, Anton Malevich, and Wolfgang Willems. “Automorphisms of Extremal Self-Dual Codes”. In: *IEEE Transactions on Information Theory* 56.5 (2010), pp. 2091–2096. DOI: 10.1109/TIT.2010.2043763.

- [BDE+26] Alessandro Budroni, Giuseppe D’Alconzo, Leonardo Errati, et al. *On Canonical Liftings, Group Actions, and Linkable Ring Signatures*. Cryptology ePrint Archive, Paper 2026/1348. 2026. URL: <https://eprint.iacr.org/2026/1348>.
- [CT19] Rodolfo Canto-Torres and Jean-Pierre Tillich. “Speeding up decoding a code with a non-trivial automorphism group up to an exponential factor”. In: *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2019, pp. 1927–1931.
- [CT13] Alonso Sepúlveda Castellanos and Guilherme Chaud Tizziotti. “On the Automorphism Group of Generalized Hermitian Codes”. In: *IEEE Transactions on Information Theory* 59.10 (2013), pp. 6642–6645. DOI: 10.1109/TIT.2013.2271034.
- [Feu09] Thomas Feulner. “The automorphism groups of linear codes and canonical representatives of their semilinear isometry classes.” In: *Adv. Math. Commun.* 3.4 (2009), pp. 363–383.
- [FHL80] Merrick Furst, John Hopcroft, and Eugene Luks. “Polynomial-time algorithms for permutation groups”. In: *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*. 1980, pp. 36–41. DOI: 10.1109/SFCS.1980.34.
- [GR13] Markus Grassl and Martin Roetteler. “Leveraging automorphisms of quantum codes for fault-tolerant quantum computation”. In: *2013 IEEE International Symposium on Information Theory*. 2013, pp. 534–538. DOI: 10.1109/ISIT.2013.6620283.
- [HP10] W Cary Huffman and Vera Pless. *Fundamentals of error-correcting codes*. Cambridge university press, 2010.
- [JMX20] Lingfei Jin, Liming Ma, and Chaoping Xing. “Construction of Optimal Locally Repairable Codes via Automorphism Groups of Rational Function Fields”. In: *IEEE Transactions on Information Theory* 66.1 (2020), pp. 210–221. DOI: 10.1109/TIT.2019.2946637.
- [KMM10] J. D. Key, T. P. McDonough, and V. C. Mavron. “Reed-Muller codes and permutation decoding”. In: *Discrete Math.* 310.22 (Nov. 2010), pp. 3114–3119. ISSN: 0012-365X. DOI: 10.1016/j.disc.2009.06.001. URL: <https://doi.org/10.1016/j.disc.2009.06.001>.
- [Knu91] Donald E. Knuth. “Efficient representation of perm groups”. In: *Combinatorica* 11 (1991), pp. 33–43. DOI: 10.1007/BF01375471.
- [Leo82] Jeffrey Leon. “Computing automorphism groups of error-correcting codes”. In: *IEEE Transactions on Information Theory* 28.3 (1982), pp. 496–511.
- [LWZ26] Xiaoru Li, Qi Wang, and Yue Zhou. “The automorphism groups of random linear codes”. In: *arXiv preprint arXiv:2607.23936* (2026).
- [LCE+24] Gaojun Luo, Xiwang Cao, Martianus Frederic Ezerman, et al. “On the Weights of Linear Codes With Prescribed Automorphisms”. In: *IEEE Transactions on Information Theory* 70.7 (2024), pp. 4983–4989. DOI: 10.1109/TIT.2023.3347701.

- [Mac64] Jessie Macwilliams. “Permutation Decoding of Systematic Codes”. In: *Bell System Technical Journal* 43.1 (1964), pp. 485–505. DOI: <https://doi.org/10.1002/j.1538-7305.1964.tb04075.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.1538-7305.1964.tb04075.x>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1964.tb04075.x>.
- [Mat79] Rudolf Mathon. “A note on the graph isomorphism counting problem”. In: *Information Processing Letters* 8.3 (1979), pp. 131–136.
- [PS21] Nicola Pace and Angelo Sonnino. *On the existence of PD-sets: Algorithms arising from automorphism groups of codes*. 2021. DOI: 10.3934/amc.2020065. URL: <https://www.aims sciences.org/article/id/60130bb0-d92f-4a8c-a825-3aef92741790>.
- [Pra62] E. Prange. “The use of information sets in decoding cyclic codes”. In: *IRE Transactions on Information Theory* 8.5 (1962), pp. 5–9. DOI: 10.1109/TIT.1962.1057777.
- [Sen97] Nicolas Sendrier. “On the dimension of the hull”. In: *SIAM Journal on Discrete Mathematics* 10.2 (1997), pp. 282–293.
- [Sen02] Nicolas Sendrier. “Finding the permutation between equivalent linear codes: The support splitting algorithm”. In: *IEEE Transactions on Information Theory* 46.4 (2002), pp. 1193–1203.
- [Sen11] Nicolas Sendrier. “Decoding one out of many”. In: *International Workshop on Post-Quantum Cryptography*. Springer. 2011, pp. 51–67.
- [SS13] Nicolas Sendrier and Dimitrios E. Simos. “How easy is code equivalence over $\mathbb{F}_q$?” In: *International Workshop on Coding and Cryptography — WCC 2013*. Available at HAL: hal-00790861v2. Bergen, Norway, Apr. 2013. URL: <https://hal.science/hal-00790861>.
- [Ser03] Ákos Seress. “Permutation Groups: A Complexity Overview”. In: *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003, pp. 48–54.
- [Sha08] T. Shaska. “Quantum codes from algebraic curves with automorphisms”. In: *Condensed Matter Physics* 11.2 (2008), pp. 383–396. DOI: 10.5488/CMP.11.2.383.
- [Sim70] Charles C. Sims. “Computational methods in the study of permutation groups”. In: *Computational Problems in Abstract Algebra*. Ed. by John Leech. Pergamon, 1970, pp. 169–183. ISBN: 978-0-08-012975-4. DOI: <https://doi.org/10.1016/B978-0-08-012975-4.50020-5>. URL: <https://www.sciencedirect.com/science/article/pii/B9780080129754500205>.
- [Sle60] David Slepian. “Some further theory of group codes”. In: *Bell System Technical Journal* 39.5 (1960), pp. 1219–1252.

A Number of codes with given hull dimension and proof of Theorem 5

For a given finite field with q elements, the number of self orthogonal codes can be counted precisely [Sen97, Theorem 1].

Theorem 17 *For q a prime power and $k, n \in \mathbb{N}$ with $k \leq n/2$, let $\mathfrak{C}_{n,k,q}$ denote the set of all self-orthogonal codes with length n and dimension k over a finite field with q elements. The cardinality of $\mathfrak{C}_{n,k,q}$ is*

- $\prod_{i=1}^k \frac{q^{n-2i+1}-1}{q^i-1}$ if n is odd and $k \leq (n-1)/2$,
- $\frac{q^{n-k}-1}{q^n-1} \prod_{i=1}^k \frac{q^{n-2i+2}-1}{q^i-1}$ if n and q are even,
- $\frac{q^{n/2-k}+1}{q^{n/2}+1} \prod_{i=1}^k \frac{q^{n-2i+2}-1}{q^i-1}$ if $n \equiv 0 \pmod{4}$, or $n \equiv 2 \pmod{4}$ and $q \equiv 1 \pmod{4}$,
- $\frac{q^{n/2-k}-1}{q^{n/2}-1} \prod_{i=1}^k \frac{q^{n-2i+2}-1}{q^i-1}$ if $n \equiv 2 \pmod{4}$, $q \equiv 3 \pmod{4}$ and $k \leq n/2 - 1$,
- 0 otherwise.

Putting together some results from [BES+25], we get the asymptotic behavior for the number of self-orthogonal codes.

Theorem 18 *For constant q , when both k and n go to infinity,*

$$|\mathfrak{C}_{n,k,q}| \sim q^{k(n-k) - \frac{k(k+1)}{2}} \cdot \omega_q,$$

where $\omega_q = \frac{\gamma_q}{(\prod_{i=1}^{\infty} \frac{1-q^{-i}}{1-q^{-i}})^2}$ is a positive constant and $\gamma_q = \prod_{i=1}^{\infty} \frac{1-q^{-i}}{1-q^{-2i}}$.

Remark 13. As a crude but simple approximation, one can use $\omega_q \approx 1/(1-1/q-1/q^2)$. For large q , the approximation becomes more precise and, in particular, $\omega_q \approx 1$.

A.1 Proof of Theorem 5

Theorem 5 *Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ with hull of dimension h . If $|\mathfrak{C}_{n,h,q}| < n!$, then the hull has non-trivial automorphisms. For constant q , let h^* indicate the maximum value of h for which the previous inequality is satisfied. Then, asymptotically, $h^* = \log_q(n/e) \cdot (1 + o(1))$, where e is the Napier's constant.*

Proof. Let $\mathcal{H} = \mathcal{H}(\mathcal{C})$ be the hull of $\mathcal{C}$, and let us consider the orbit of $\mathcal{H}$ under the action of S_n , that is,

$$\mathcal{O}_{S_n}(\mathcal{H}) = \{\varphi(\mathcal{H}) \mid \varphi \in S_n\}.$$

If the orbit has less than $n!$ elements, then there are (at least) two distinct permutations $\varphi \neq \varphi' \in S_n$ such that

$$\varphi(\mathcal{H}) = \varphi'(\mathcal{H}) \implies \mathcal{H} = \varphi^{-1} \circ \varphi'(\mathcal{H}),$$

so $\varphi^{-1} \circ \varphi$ is a non-trivial automorphism for $\mathcal{H}$. Since applying a permutation to a self-orthogonal code, we get another self-orthogonal code, we also have $\mathcal{O}_{S_n}(\mathcal{H}) \subseteq \mathfrak{C}_{n,h,q}$. Then, existence of automorphisms is guaranteed whenever

$$|\mathcal{O}_{S_n}(\mathcal{H})| \leq |\mathfrak{C}_{n,h,q}| < n!.$$

Let h^* be the maximum value of h for which $|\mathfrak{C}_{n,h,q}| < n!$. If the hull dimension is greater than h^* , then the inequality is not satisfied. Using the asymptotic expansion for $|\mathfrak{C}_{n,h^*,q}|$ and Stirling's approximation for the factorial, and neglecting low-order terms, we get

$$h^*(n - h^*) - \frac{h^*(h^* + 1)}{2} < n \cdot \log_q(n/e). \quad (6)$$

We already know that h^* can be at most $\log_q(n)$, otherwise the hull has repeated coordinates and non-trivial automorphisms exist. Then, $h^* = O(\log_q(n))$ and simple manipulations of Equation (6) yields

$$h^* - \underbrace{\frac{h^{*2}}{n} - \frac{h^*(h^* + 1)}{2n}}_{o(1)} < \log_q(n/e) \implies h^* < \log_q(n/e)(1 + o(1)). \quad (7)$$

On the other hand, we have that since h^* is the maximum value of h for which $|\mathfrak{C}_{n,h,q}| < n!$, it means that $h^* + 1$ does not satisfy the inequality, that is,

$$|\mathfrak{C}_{n,h^*+1,q}| \geq n!.$$

Applying to $h^* + 1$ the same asymptotic expansion used to derive Equation (6), but with the inequality reversed, we obtain

$$(h^* + 1)(n - h^* - 1) - \frac{(h^* + 1)(h^* + 2)}{2} \geq n \log_q(n/e). \quad (8)$$

Dividing by n and using $h^* + 1 = O(\log_q n)$, so that $\frac{(h^*+1)^2}{n}$ and $\frac{(h^*+1)(h^*+2)}{2n}$ are $o(1)$, Equation (8) becomes

$$(h^* + 1) \left(1 - \frac{h^* + 1}{n}\right) - \frac{(h^* + 1)(h^* + 2)}{2n} \geq \log_q(n/e),$$

implying

$$h^* + 1 - o(1) \geq \log_q(n/e).$$

Hence

$$h^* \geq \log_q(n/e) - 1 - o(1). \quad (9)$$

Combining Equation (9) with the bound Equation (7), we get

$$\log_q(n/e) - 1 - o(1) \leq h^* < \log_q(n/e) + o(1).$$

Since q is constant, $\log_q(n/e) \rightarrow \infty$ as $n \rightarrow \infty$; hence both -1 and $o(1)$ are $o(\log_q(n/e))$. Dividing throughout by $\log_q(n/e)$ and letting $n \rightarrow \infty$ yields

$$\frac{h^*}{\log_q(n/e)} \rightarrow 1,$$

that is,

$$h^* = \log_q(n/e) \cdot (1 + o(1)).$$

□

B Deferred Proofs from Section 4

This section contains some of the proofs for Section 4.

B.1 Proof of Theorem 7

Theorem 7 *Let $\mathcal{A}$ and $\mathcal{B} = \pi(\mathcal{A})$ for some $\pi \in S_n$. Then we have that $\text{P-Aut}(\mathcal{A}) \cong \text{P-Aut}(\mathcal{B})$ and in particular:*

$$\text{P-Aut}(\mathcal{B}) = \{\pi \circ \sigma \circ \pi^{-1} \mid \sigma \in \text{P-Aut}(\mathcal{A})\}.$$

Proof. We first prove that $\text{P-Aut}(\mathcal{B}) \supseteq \{\pi \circ \sigma \circ \pi^{-1} \mid \sigma \in \text{P-Aut}(\mathcal{A})\}$: for each automorphism $\sigma \in \text{P-Aut}(\mathcal{A})$, we get an automorphism of $\mathcal{B}$ as $\pi \circ \sigma \circ \pi^{-1}$. Indeed,

$$\begin{aligned} \pi \circ \sigma \circ \pi^{-1}(\mathcal{B}) &= \pi \circ \sigma \circ \pi^{-1} \circ \pi(\mathcal{A}) \\ &= \pi \circ \underbrace{\sigma(\mathcal{A})}_{\mathcal{A}} = \pi(\mathcal{A}) = \mathcal{B}. \end{aligned}$$

We now prove that $\text{P-Aut}(\mathcal{B}) \subseteq \{\pi \circ \sigma \circ \pi^{-1} \mid \sigma \in \text{P-Aut}(\mathcal{A})\}$. To this end, let $\varphi \in \text{P-Aut}(\mathcal{B})$ and consider that

$$\pi^{-1} \circ \varphi \circ \pi(\mathcal{A}) = \pi^{-1} \circ \underbrace{\varphi(\mathcal{B})}_{\mathcal{B}} = \pi^{-1}(\mathcal{B}) = \mathcal{A}.$$

Thus $\phi = \pi^{-1} \circ \varphi \circ \pi \in \text{P-Aut}(\mathcal{A})$. Now, we have that

$$\pi \circ \phi \circ \pi^{-1} = \pi \circ \underbrace{(\pi^{-1} \circ \varphi \circ \pi)}_{\in \text{P-Aut}(\mathcal{A})} \circ \pi^{-1} = \varphi,$$

thus $\varphi \in \{\pi \circ \sigma \circ \pi^{-1} \mid \sigma \in \text{P-Aut}(\mathcal{A})\}$. □

B.2 Proof of Theorem 8

Theorem 8 *Given two codes $\mathcal{A}, \mathcal{B}$ with hulls $\mathcal{H}(\mathcal{A})$ and $\mathcal{H}(\mathcal{B})$, respectively, the Cartesian product code has hull*

$$\mathcal{H}(\mathcal{A} \times \mathcal{B}) = \mathcal{H}(\mathcal{A}) \times \mathcal{H}(\mathcal{B}).$$

In particular, the hull dimension of the product code is the sum of the dimensions of the hulls.

Proof. We first prove that $\mathcal{H}(\mathcal{A}) \times \mathcal{H}(\mathcal{B})$ is included in $\mathcal{H}(\mathcal{A} \times \mathcal{B})$, then prove the opposite direction.

(Direction $\subseteq$). Let $(\mathbf{c}', \mathbf{c}'') \in \mathcal{H}(\mathcal{A}) \times \mathcal{H}(\mathcal{B})$. Then, let $(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}$ and we have

$$\langle (\mathbf{c}', \mathbf{c}''), (\mathbf{a}, \mathbf{b}) \rangle = \langle \mathbf{c}', \mathbf{a} \rangle + \langle \mathbf{c}'', \mathbf{b} \rangle,$$

with $\langle \mathbf{c}', \mathbf{a} \rangle = 0$ since $\mathbf{c}' \in \mathcal{H}(\mathcal{A})$ and $\mathbf{a} \in \mathcal{A}$. Analogously, $\langle \mathbf{c}'', \mathbf{b} \rangle = 0$. Therefore $(\mathbf{c}', \mathbf{c}'') \in \mathcal{H}(\mathcal{A} \times \mathcal{B})$ and $\mathcal{H}(\mathcal{A}) \times \mathcal{H}(\mathcal{B}) \subseteq \mathcal{H}(\mathcal{A} \times \mathcal{B})$.

(Direction $\supseteq$). Let $(\mathbf{c}', \mathbf{c}'') \in \mathcal{H}(\mathcal{A} \times \mathcal{B})$: for each $\mathbf{a} \in \mathcal{A}$ and $\mathbf{b} \in \mathcal{B}$, it holds that

$$\langle (\mathbf{c}', \mathbf{c}''), (\mathbf{a}, \mathbf{b}) \rangle = \langle \mathbf{c}', \mathbf{a} \rangle + \langle \mathbf{c}'', \mathbf{b} \rangle = 0.$$

In particular, it holds for $\mathbf{b} = \mathbf{0}$, obtaining $\langle \mathbf{c}', \mathbf{a} \rangle = 0$, and hence $\mathbf{c}' \in \mathcal{H}(\mathcal{A})$. Analogously, using $\mathbf{a} = \mathbf{0}$, we obtain that $\mathbf{c}'' \in \mathcal{H}(\mathcal{B})$. Hence, $(\mathbf{c}', \mathbf{c}'') \in \mathcal{H}(\mathcal{A}) \times \mathcal{H}(\mathcal{B})$ and, consequently, $\mathcal{H}(\mathcal{A} \times \mathcal{B}) \subseteq \mathcal{H}(\mathcal{A}) \times \mathcal{H}(\mathcal{B})$.

The observation on the hull dimension follows trivially. $\square$

B.3 Proof of Theorem 9

Theorem 9 *Let $\mathcal{A}, \mathcal{B} \subseteq \mathbb{F}_q^n$ be non-equivalent codes of dimension k . Moreover, suppose that $\mathcal{A}$ and $\mathcal{B}$ are indecomposable, i.e. they are not the Cartesian product of other codes. Then*

$$\text{P-Aut}(\mathcal{A} \times \mathcal{B}) \cong \text{P-Aut}(\mathcal{A}) \times \text{P-Aut}(\mathcal{B}).$$

Proof. Let $\sigma \in \text{P-Aut}(\mathcal{A} \times \mathcal{B})$. We define $\mathcal{A}^0 = \mathcal{A} \times \{0^n\}$ and $\mathcal{B}^0 = \{0^n\} \times \mathcal{B}$. Observe that $\mathcal{A} \times \mathcal{B}$ can also be seen as $\mathcal{A}^0 + \mathcal{B}^0$. We now consider how σ splits over $\{1, \dots, 2n\}$: there are only the following three cases.

- Case $\sigma(\{1, \dots, n\}) = \{1, \dots, n\}$ and $\sigma(\{n+1, \dots, 2n\}) = \{n+1, \dots, 2n\}$. Since this is an automorphism this means that $\sigma(\mathcal{A}^0) = \mathcal{A}^0$ and $\sigma(\mathcal{B}^0) = \mathcal{B}^0$. Let $\mathcal{A}', \mathcal{B}'$ be two codes such that

$$\sigma(\mathcal{A}^0) = \mathcal{A}' \times \{0^n\}, \quad \sigma(\mathcal{B}^0) = \{0^n\} \times \mathcal{B}'.$$

Since $\mathcal{A}$ and $\mathcal{B}$ are indecomposable, in particular, they also do not have null coordinates. Then, the only possibility is $\mathcal{A}' = \mathcal{A}$ and $\mathcal{B}' = \mathcal{B}$. This means

that each automorphism σ is associated with a permutation matrix of the form

$$\begin{pmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}' \end{pmatrix}, \quad \mathbf{P} \in \text{P-Aut}(\mathcal{A}), \quad \mathbf{P}' \in \text{P-Aut}(\mathcal{B}).$$

This implies that $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$ contains automorphisms forming a group which is isomorphic to $\text{P-Aut}(\mathcal{A}) \times \text{P-Aut}(\mathcal{B})$.

As we prove in the following, since $\mathcal{A}$ and $\mathcal{B}$ are indecomposable and not isomorphic, these are the only automorphisms of $\mathcal{A} \times \mathcal{B}$.

- *Case* $\sigma(\{1, \dots, n\}) = \{n+1, \dots, 2n\}$ and $\sigma(\{n+1, \dots, 2n\}) = \{1, \dots, n\}$. This implies $\sigma(\mathcal{A}^0) = \mathcal{B}^0$ and $\sigma(\mathcal{B}^0) = \mathcal{A}^0$, which further implies that the codes $\mathcal{A}$ and $\mathcal{B}$ are isomorphic: this contradicts the hypothesis.
- *Case* $\sigma(\{1, \dots, n\}) \neq \{1, \dots, n\}$ and $\sigma(\{1, \dots, n\}) \neq \{n+1, \dots, 2n\}$. Let $\mathbf{A}$ and $\mathbf{B}$ be generator matrices for $\mathcal{A}$ and $\mathcal{B}$, respectively, and let $\mathbf{G}$ be the matrix formed by stacking $\mathbf{A}$ and $\mathbf{B}$ in the main diagonal. Then, we have

$$\sigma(\mathbf{G}) = \begin{pmatrix} \mathbf{A}' & \mathbf{0} & \mathbf{A}'' & \mathbf{0} \\ \mathbf{0} & \mathbf{B}' & \mathbf{0} & \mathbf{B}'' \end{pmatrix} \cdot \begin{pmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}' \end{pmatrix} = (\mathbf{C}' \cdot \mathbf{P} | \mathbf{C}'' \cdot \mathbf{P}'),$$

with $\mathbf{P}, \mathbf{P}' \in \mathbb{F}_q^{n \times n}$ being permutations, $\mathbf{A} = (\mathbf{A}', \mathbf{A}'')$ and $\mathbf{B} = (\mathbf{B}', \mathbf{B}'')$. Since both codes $\mathcal{A}$ and $\mathcal{B}$ are indecomposable, in particular, they do not have null coordinates: this means that $\mathbf{A}', \mathbf{A}'', \mathbf{B}'$ and $\mathbf{B}''$ cannot be null matrices and, therefore, have rank at least 1. For each codeword generated by $\sigma(\mathbf{G})$, the restriction to the coordinates $\{1, \dots, n\}$ must be a codeword of $\mathcal{A}$. But, then, $\mathbf{C}' \cdot \mathbf{P} \in \mathcal{A}$ must be a generator for $\mathcal{A}$. Since both $\mathbf{A}'$ and $\mathbf{B}'$ have rank at least 1, this means that they generate codes of dimension at least 1, therefore $\mathbf{C}' \cdot \mathbf{P}'$ generates a decomposable code. This contradicts the hypothesis that $\mathcal{A}$ is indecomposable. $\square$

B.4 Proof of Corollary 2

We first define the semidirect product of groups, then prove the corollary.

Definition 17 (Semidirect product). Let G and K be groups, and let $\varphi : K \rightarrow \text{Aut}(G)$ be a group homomorphism. The semidirect product of G by K with respect to φ , denoted by $G \rtimes_{\varphi} K$, is defined as the group whose underlying set is the Cartesian product $G \times K$, equipped with the binary operation:

$$(g_1, k_1) \cdot (g_2, k_2) = (g_1 \varphi_{k_1}(g_2), k_1 k_2)$$

for all $g_1, g_2 \in G$ and $k_1, k_2 \in K$, where $\varphi_{k_1}(g_2)$ denotes the action of the automorphism associated to k_1 on the element g_2 .

Corollary 2. Let $\mathcal{A}, \mathcal{B} \subseteq \mathbb{F}_q^n$ be two equivalent codes of dimension k . Moreover, suppose that $\mathcal{A}$ and $\mathcal{B}$ are indecomposable, i.e. they are not the Cartesian product of other codes. Then

$$\text{P-Aut}(\mathcal{A} \times \mathcal{B}) \cong (\text{P-Aut}(\mathcal{A}) \times \text{P-Aut}(\mathcal{B})) \rtimes S_2,$$

where $\rtimes$ indicates the semidirect product of groups. Moreover, if G is a group isomorphic to $\text{P-Aut}(\mathcal{A})$, it holds that $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$ is isomorphic to $G \wr S_2$.

Proof. The key difference from Theorem 9 is that $\mathcal{A}$ and $\mathcal{B}$ are equivalent, thus the second bullet point of the previous proof cannot be excluded for this case.

More formally, let $\sigma \in \text{P-Aut}(\mathcal{A} \times \mathcal{B})$ and consider $\sigma(\mathcal{A}^0)$, where $\mathcal{A}^0 = \mathcal{A} \times \{0\}^n$. In the previous proof we had that $\mathcal{A}$ and $\mathcal{B}$ were not equivalent and thus we could not have that $\sigma(\mathcal{A}^0) = \mathcal{B}^0$.¹⁵ Since in this case we cannot exclude this scenario, we have that every σ satisfies exactly one of the following properties

1. $\sigma(\mathcal{A}^0) = \mathcal{A}^0$ (and so $\sigma(\mathcal{B}^0) = \mathcal{B}^0$), or
2. $\sigma(\mathcal{A}^0) = \mathcal{B}^0$ (and so $\sigma(\mathcal{B}^0) = \mathcal{A}^0$).

We call A_1, A_2 the corresponding subsets of $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$ satisfying the two above properties, so that $\text{P-Aut}(\mathcal{A} \times \mathcal{B}) = A_1 \sqcup A_2$. Formally,

$$\begin{aligned} A_1 &= \{\sigma \in \text{P-Aut}(\mathcal{A} \times \mathcal{B}) \mid \sigma(\mathcal{A}^0) = \mathcal{A}^0, \sigma(\mathcal{B}^0) = \mathcal{B}^0\} \\ A_2 &= \{\sigma \in \text{P-Aut}(\mathcal{A} \times \mathcal{B}) \mid \sigma(\mathcal{A}^0) = \mathcal{B}^0, \sigma(\mathcal{B}^0) = \mathcal{A}^0\}. \end{aligned}$$

One can show that A_1 is a subgroup of $\text{P-Aut}(\mathcal{A} \times \mathcal{B})$, while A_2 is just a (proper) subset.

As per Theorem 7 we have that $\text{P-Aut}(\mathcal{A}) \cong \text{P-Aut}(\mathcal{B})$. By defining $G \cong \text{P-Aut}(\mathcal{A})$ we can prove that $A_1 \cong G \times G$. To show this, let $\sigma \in A_1$, then there exist $g \in \text{P-Aut}(\mathcal{A})$, $h \in \text{P-Aut}(\mathcal{B})$ such that:

$$\sigma : \mathcal{A} \times \mathcal{B} \rightarrow \mathcal{A} \times \mathcal{B}, (\mathbf{u}, \mathbf{v}) \mapsto (g(\mathbf{u}), h(\mathbf{v})).$$

Observe that this correspondence leads to the isomorphism

$$A_1 \rightarrow \text{P-Aut}(\mathcal{A}) \times \text{P-Aut}(\mathcal{B}), \sigma \mapsto (g, h).$$

For this reason, in the following, we can denote the automorphism $\sigma \in A_1$ as $\phi_{g,h}$ and we have that $A_1 \cong G \times G$.

Let π be an isomorphism leading to the equivalence between $\mathcal{A}$ and $\mathcal{B}$, namely $\pi(\mathcal{A}) = \mathcal{B}$. Then define the block-swapping map

$$\rho : \mathcal{A} \times \mathcal{B} \rightarrow \mathcal{A} \times \mathcal{B}, (\mathbf{u}, \mathbf{v}) \mapsto (\pi^{-1}(\mathbf{v}), \pi(\mathbf{u})).$$

Clearly, ρ is an automorphism of $\mathcal{A} \times \mathcal{B}$ and has order 2 (i.e., $\rho^2 = id$), implying $\langle \rho \rangle \cong S_2$. Moreover, by construction $\rho \in A_2$ and hence, $\rho \notin A_1$.

Now we show that A_2 is a coset of A_1 via ρ . Let $\sigma_1 \in A_1$, then one has $\rho(\sigma_1(\mathcal{A}^0)) = \rho(\mathcal{A}^0) = \mathcal{B}^0$, so $\rho\sigma_1 \in A_2$; thus $\rho A_1 \subseteq A_2$. Conversely, Let $\sigma_2 \in A_2$. It holds that $\rho(\sigma_2(\mathcal{A}^0)) = \rho(\mathcal{B}^0) = \mathcal{A}^0$, so $\rho\sigma_2 \in A_1$; thus $\rho A_2 \subseteq A_1$. Since ρ has order 2, we obtain $A_2 \subseteq \rho A_1$. It follows that $A_2 = \rho A_1$, giving

$$\text{P-Aut}(\mathcal{A} \times \mathcal{B}) = A_1 \sqcup \rho A_1. \quad (10)$$

¹⁵ Notice that the third bullet point, about the exclusion of the “splitted” support case relied solely on the irreducibility of $\mathcal{A}$, and is unaffected here.

This implies that A_1 has index 2, and hence it is normal. Moreover, from Equation (10), we have that $\text{P-Aut}(\mathcal{A} \times \mathcal{B}) = A_1 \cdot \langle \rho \rangle$, with $A_1 \cap \langle \rho \rangle = \{id\}$. Hence, we can write $\text{P-Aut}(\mathcal{A} \times \mathcal{B}) = A_1 \rtimes \langle \rho \rangle$, namely

$$\text{P-Aut}(\mathcal{A} \times \mathcal{B}) \cong (\text{P-Aut}(\mathcal{A}) \times \text{P-Aut}(\mathcal{B})) \rtimes S_2.$$

Furthermore, we have $\text{P-Aut}(\mathcal{A} \times \mathcal{B}) \cong G \wr S_2$. $\square$

C Finding indecomposable factors of a code

In this section we show a polynomial-time algorithm to decompose a given code $\mathcal{C}$ into indecomposable factors $\mathcal{C}_1, \dots, \mathcal{C}_\ell$ such that $\mathcal{C} \cong \mathcal{C}_1 \times \dots \times \mathcal{C}_\ell$. In this case we say that $\mathcal{C} \cong \mathcal{C}_1 \times \dots \times \mathcal{C}_\ell$ along a partition $I_1, \dots, I_\ell$ of $\{1, \dots, n\}$ if and only if $\mathcal{C}_i \cong \mathcal{C}_{|I_i}$, where $\mathcal{C}_{|I_i}$ denotes the code obtained from $\mathcal{C}$ by removing all the column outside I_i .

Given a generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ for $\mathcal{C}$, consider its reduced row-echelon form

$$\text{RREF}(\mathbf{G}) = \begin{pmatrix} \mathbf{g}^{(1)} \\ \vdots \\ \mathbf{g}^{(k)} \end{pmatrix}.$$

We show that given $\mathcal{C}$, finding $\mathcal{C}_1, \dots, \mathcal{C}_\ell$ is equivalent to finding the connected components of the graph $\mathcal{G}(\mathcal{C})$ with vertex set $\{1, \dots, n\}$ and adjacency matrix

$$\mathbf{A}_{i,j} = \begin{cases} 1 & \text{if } \exists r \in \{1, \dots, k\} : g_i^{(r)} \neq 0 \wedge g_j^{(r)} \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

This step can be done in polynomial time with worst-case complexity $O(n^2)$.

Lemma 3. *Let $\mathcal{C}$ decompose as $\mathcal{C} \cong \mathcal{C}_1 \times \dots \times \mathcal{C}_\ell$ along a partition of $I_1, \dots, I_\ell$ of $\{1, \dots, n\}$. Then every row of $\text{RREF}(\mathbf{G})$ has support contained in a single I_t , where $\mathbf{G}$ is the generator matrix of $\mathcal{C}$.*

Proof. For all $i = 1, \dots, \ell$ let $\mathbf{G}_i$ be the generator matrix in reduced row-echelon form of $\mathcal{C}_i$ and let $\mathbf{G}'_i$ be its extension obtained by adding column of zeroes outside of I_i . Clearly, since $\mathbf{G}_i$ is in reduced row-echelon form then also $\mathbf{G}'_i$ is so, and all the pivot columns are in I_i .

Let $\mathbf{G}''$ be the matrix obtained by stacking all the matrices $\mathbf{G}'_i$:

$$\mathbf{G}'' = \begin{pmatrix} \mathbf{G}'_1 \\ \vdots \\ \mathbf{G}'_\ell \end{pmatrix}.$$

Consider now the matrix $\mathbf{G}'$ obtained from $\mathbf{G}''$ by reordering its rows by increasing pivot column: clearly $\mathbf{G}, \mathbf{G}'$ and $\mathbf{G}''$ generates the same code. It is easy to see that $\mathbf{G}'$ is in reduced row-echelon form: pivots are strictly increasing by construction, and every pivot column has zero entries in all other rows: within

a block because $\mathbf{G}'_i$ is reduced for all i , and across blocks because a row of $\mathbf{G}'_j$ have all zeroes in the column indexed by I_i for all $j \neq i$. By the uniqueness of the reduced row-echelon form we have the thesis.

Theorem 1. *Let $\mathcal{C}$ be a $[k, n]$ -code. Let $I_1, \dots, I_\ell$ be the connected components of the graph $\mathcal{G}(\mathcal{C})$. Then $\mathcal{C} \cong \mathcal{C}_1 \times \dots \times \mathcal{C}_\ell$ along a partition $I_1, \dots, I_\ell$ of $\{1, \dots, n\}$ and each $\mathcal{C}_i$ is indecomposable.*

Proof. Let $\mathbf{G}$ be the generator matrix of $\mathcal{C}$ in reduced row-echelon form. We divide the proof in two steps: first we show that $\mathcal{C}$ decompose along $I_1, \dots, I_\ell$ then we show that each $\mathcal{C}_i$ indecomposable.

Decomposition. For each row $\mathbf{g}^{(i)}$ of $\mathbf{G}$, every pair of coordinates in $\text{Supp}(\mathbf{g}^{(i)})$ is adjacent in $\mathcal{G}(\mathcal{C})$ by definition of $\mathbf{A}$; hence $\text{Supp}(\mathbf{g}^{(i)})$ lies inside a single connected component. Let f be the function from $\{1, \dots, k\}$ to $\{1, \dots, \ell\}$ such that $f(i) = j$ if and only if $\text{Supp}(\mathbf{g}^{(i)})$ lies inside I_j . We can define

$$V_s := \text{span}\{\mathbf{g}^{(i)} : f(i) = s\},$$

thus we get

$$\mathcal{C} = \text{span}(\mathbf{g}^{(1)}, \dots, \mathbf{g}^{(k)}) = V_1 \times \dots \times V_\ell,$$

with $\text{Supp}(V_s) \subseteq I_s$.

Now, consider any codeword $\mathbf{c} \in \mathcal{C}$. By definition of $V_1, \dots, V_\ell$, $\mathbf{c}$ can be uniquely written as a sum of component vectors:

$$\mathbf{c} = \sum_{s=1}^{\ell} \mathbf{v}_s, \quad \text{where } \mathbf{v}_s \in V_s \text{ for each } s \in \{1, \dots, \ell\}.$$

Since the components $I_1, \dots, I_\ell$ form a partition of $\{1, \dots, n\}$ and $\text{Supp}(V_t) \subseteq I_t$, it follows that $\mathbf{v}_t|_{I_s} = \mathbf{0}$ for all $t \neq s$. Consequently, restricting the codeword $\mathbf{c}$ to the coordinate set I_s yields exactly $\mathbf{v}_s$

$$\mathbf{c}|_{I_s} = \mathbf{v}_s.$$

This shows that the projection of the code $\mathcal{C}$ onto the coordinate set I_s coincides with the subspace V_s , i.e., $V_s = \mathcal{C}|_{I_s}$. This implies

$$\mathcal{C} = \mathcal{C}|_{I_1} \times \dots \times \mathcal{C}|_{I_\ell}.$$

Indecomposability of the factors. Suppose some $\mathcal{C}|_{I_s}$ decomposes non-trivially along the partition K_1, K_2 of I_s . Combined with the above, $\mathcal{C}$ decomposes along the finer partition obtained by splitting I_s into K_1, K_2 . By Lemma 3, every row of $\mathbf{G}$ has support inside a single set of this partition, so no row has non-zero entries in both K_1 and K_2 ; hence $\mathcal{G}(\mathcal{C})$ has no edge between K_1 and K_2 . But I_s is connected by hypothesis, so a path joins some vertex of K_1 to some vertex of K_2 must exist, obtaining a contradiction. Hence $\mathcal{C}|_{I_s}$ is indecomposable.

As a trivial corollary of the above proof we also have that decomposition is unique.

D Deferred Proofs from Section 6

D.1 Proof of Theorem 15

Theorem 15 *Let $\mathbb{F}_q$ be a finite field with characteristic p . Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a linear code with dimension k and hull dimension h . Let $J = (j_1, \dots, j_h) \subseteq \{1, \dots, n\}$ be an ordered set corresponding to an information set for $\mathcal{H}(\mathcal{C})$, and let $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ be a generator matrix for $\mathcal{C}$. Assume that $\mathbf{G}$ has no repeated columns. Let $U = (u_1, \dots, u_h)$, with $u_i \in \{1, \dots, n\} \forall i \in \{1, \dots, h\}$, be an ordered set corresponding to another information set for $\mathcal{H}(\mathcal{C})$. For $i \in \{1, \dots, h\}$, let $\ell_i = i \cdot p + 1$ and let*

$$\begin{aligned} \mathbf{A} &= (\mathbf{G} \mid \mathbf{M}_1 \mid \dots \mid \mathbf{M}_h) \quad , \quad \mathbf{M}_i = \underbrace{(\mathbf{g}_{j_i} \mid \mathbf{g}_{j_i} \mid \dots \mid \mathbf{g}_{j_i})}_{\ell_i} \in \mathbb{F}_q^{k \times \ell_i} \\ \mathbf{B} &= (\mathbf{G} \mid \mathbf{W}_1 \mid \dots \mid \mathbf{W}_h) \quad , \quad \mathbf{W}_i = \underbrace{(\mathbf{g}_{u_i} \mid \mathbf{g}_{u_i} \mid \dots \mid \mathbf{g}_{u_i})}_{\ell_i} \in \mathbb{F}_q^{k \times \ell_i} \end{aligned} \quad (5)$$

Let $\mathcal{A} = \text{span}(\mathbf{A})$ and $\mathcal{B} = \text{span}(\mathbf{B})$ and let $\tilde{n} = n + \sum_{i=1}^h \ell_i$, then:

- a) both $\mathcal{A}$ and $\mathcal{B}$ have trivial hulls;
- b) for any $\varphi \in \text{P-Aut}(\mathcal{C})$ sending J to U (namely, $\varphi(j_i) = u_i$ for every $i = 1, \dots, h$), there is an isomorphism $\chi \in S_{\tilde{n}}$ sending $\mathcal{A}$ to $\mathcal{B}$ such that $\chi(\{1, \dots, n\} \setminus J) = \varphi(\{1, \dots, n\} \setminus J)$;
- c) for any isomorphism $\chi \in S_{\tilde{n}}$ sending $\mathcal{A}$ into $\mathcal{B}$, there exists $\varphi \in \text{P-Aut}(\mathcal{C})$ such that $\varphi(\{1, \dots, n\} \setminus J) = \chi(\{1, \dots, n\} \setminus J)$;
- d) for any isomorphism $\chi \in S_{\tilde{n}}$ between $\mathcal{A}$ and $\mathcal{B}$, when written as a product of disjoint cycles, there is no cycle moving one index of $\{1, \dots, n\} \setminus J$ to $U \cup \{n+1, \dots, \tilde{n}\}$.

Proof. We prove each thesis separately. The proof of thesis a) is found in Section 7.1.

Proof of thesis b) Let $\mathcal{A}, \mathcal{B} \subseteq \mathbb{F}_q^n$ be the codes generated by $\mathbf{A}$ and $\mathbf{B}$, respectively.

Let $\varphi \in S_n$ be an automorphism for $\mathcal{C}$ that maps $J = \{j_1, \dots, j_h\}$ into $U = (u_1, \dots, u_h)$ (remember that U is an ordered set). Since φ is an automorphism for $\mathcal{C}$, there is a matrix $\mathbf{S} \in GL_k(\mathbb{F}_q)$ such that $\mathbf{S} \cdot \varphi(\mathbf{G}) = \mathbf{G}$. Let us indicate:

$$\varphi(\{1, \dots, n\} \setminus J) = \{1, \dots, n\} \setminus U = (f_1, f_2, \dots, f_{n-h}).$$

Then:

$$\mathbf{S} \cdot \mathbf{G}_J = (\mathbf{g}_{u_1}, \dots, \mathbf{g}_{u_h}), \quad (11)$$

$$\mathbf{S} \cdot \mathbf{G}_{\{1, \dots, n\} \setminus J} = (\mathbf{g}_{f_1}, \dots, \mathbf{g}_{f_{n-h}}). \quad (12)$$

Observe that φ is defining a mapping from $\{1, \dots, n\} \setminus J = \{z_1, \dots, z_{n-h}\}$ to $\{1, \dots, n\} \setminus U$ given by

$$\begin{aligned} z_1 &\longrightarrow f_1, \\ z_2 &\longrightarrow f_2, \\ &\vdots \\ z_{n-h} &\longrightarrow f_{n-h}. \end{aligned} \tag{13}$$

We will indicate with $\mathbf{a}_i$ (resp., with $\mathbf{b}_i$) the i -th column of $\mathbf{A}$ (resp., the i -th column of $\mathbf{B}$). Let us first focus on Equation (11): by construction, we have

$$\mathbf{G}_J = \mathbf{A}_J = (\mathbf{a}_{j_1}, \dots, \mathbf{a}_{j_h}), \quad (\mathbf{g}_{u_1}, \dots, \mathbf{g}_{u_h}) = (\mathbf{b}_{u_1}, \dots, \mathbf{b}_{u_h}).$$

Therefore, we get

$$\mathbf{S} \cdot \mathbf{A}_J = \mathbf{S} \cdot (\mathbf{a}_{j_1}, \dots, \mathbf{a}_{j_h}) = (\mathbf{b}_{u_1}, \dots, \mathbf{b}_{u_h}).$$

Notice that, since each block $\mathbf{M}_i$ contains replica of $\mathbf{a}_{j_i}$ and each block $\mathbf{W}_i$ contains replica of $\mathbf{b}_{u_i}$, we also get that $\mathbf{S} \cdot \mathbf{M}_i = \mathbf{W}_i$. Therefore, left multiplication by $\mathbf{S}$ creates a mapping from columns indexed by $J \cup \{n+1, \dots, \tilde{n}\}$ to columns indexed by $U \cup \{n+1, \dots, \tilde{n}\}$, as follows

$$\begin{aligned} \{j_1\} \cup \{n+1, \dots, n+\ell_1\} &\longrightarrow \{u_1\} \cup \{n+1, \dots, n+\ell_1\}, \\ \{j_2\} \cup \{n+1+\ell_1, \dots, n+\ell_2\} &\longrightarrow \{u_2\} \cup \{n+1+\ell_1, \dots, n+\ell_2\}, \\ &\vdots \\ \{j_h\} \cup \left\{n+1+\sum_{i=1}^{h-1} \ell_i, \dots, \tilde{n}\right\} &\longrightarrow \{u_h\} \cup \left\{n+1+\sum_{i=1}^{h-1} \ell_i, \dots, \tilde{n}\right\}. \end{aligned} \tag{14}$$

Analogously, starting from Equation (12) and indicating $\{1, \dots, n\} \setminus J = \{z_1, \dots, z_{n-h}\}$, we get

$$\mathbf{G}_{\{1, \dots, n\} \setminus J} = (\mathbf{a}_{z_1}, \dots, \mathbf{a}_{z_{n-h}}), \quad (\mathbf{g}_{f_1}, \dots, \mathbf{g}_{f_{n-h}}) = (\mathbf{b}_{f_1}, \dots, \mathbf{b}_{f_{n-h}}),$$

therefore

$$\mathbf{S} \cdot (\mathbf{a}_{z_1}, \dots, \mathbf{a}_{z_{n-h}}) = (\mathbf{b}_{f_1}, \dots, \mathbf{b}_{f_{n-h}}),$$

which gives rise to the same bijection from Equation (13).

Putting together Equation (14) and Equation (13), we get a bijection $\chi : \{1, \dots, \tilde{n}\} \rightarrow \{1, \dots, \tilde{n}\}$. In particular, χ acts on $\{1, \dots, n\} \setminus J$ in the same way of φ .

Proof of thesis c) Let $\chi \in S_{\tilde{n}}$ and $\mathbf{S} \in GL_k(\mathbb{F}_q)$ such that $\mathbf{S} \cdot \chi(\mathbf{A}) = \mathbf{B}$. Observe that both codes have blocks of repeated columns, all with distinct lengths $\ell_1 + 1$, $\ell_2 + 1$, and so on. Then, any permutation sending the code generated by $\mathbf{A}$ to the code generated by $\mathbf{B}$ must preserve such blocks. Indeed, we have that

$$\mathbf{S} \cdot (\mathbf{g}_{j_i}, \mathbf{M}_i) = \mathbf{S} \cdot \underbrace{(\mathbf{g}_{j_i}, \dots, \mathbf{g}_{j_i})}_{\ell_i + 1 \text{ replica}} = \underbrace{(\mathbf{x}, \dots, \mathbf{x})}_{\ell_i + 1 \text{ replica}},$$

for some $\mathbf{x} \in \mathbb{F}_q^k$. Multiplication between $\mathbf{S}$ and any other column of $\mathbf{A}$ must produce a vector $\mathbf{x}' \neq \mathbf{x}$. Indeed, for any such column $\mathbf{g}' \neq \mathbf{g}_{j_i}$, having $\mathbf{S} \cdot \mathbf{g}_{j_i} = \mathbf{S} \cdot \mathbf{g}'$ implies $\mathbf{S} \cdot (\mathbf{g}_{j_i} - \mathbf{g}') = \mathbf{0}$, which can happen only if $\mathbf{S}$ is not full rank (but this is not possible since $\mathbf{S} \in GL_k(\mathbb{F}_q)$) or $\mathbf{g}_{j_i} = \mathbf{g}'$, which is also not possible since, by hypothesis, $\mathcal{C}$ does not have repeated coordinates.

Then, if $\mathbf{S}$ is associated with an isomorphism χ from $\mathcal{A}$ to $\mathcal{B}$, we have that the only possibility for χ is that it acts as a bijection from

$$\{j_i\} \cup \left\{ n + \sum_{z=1}^{i-1} \ell_z + 1, \dots, n + \sum_{z=1}^i \ell_z \right\}$$

to

$$\{u_i\} \cup \left\{ n + \sum_{z=1}^{i-1} \ell_z + 1, \dots, n + \sum_{z=1}^i \ell_z \right\},$$

for each $i = 1, \dots, h$. In particular, since all the above sets are disjoint, χ acts also as a bijection from

$$\{1, \dots, \tilde{n}\} \setminus \left(\bigcup_{i=1}^h \left(\{j_i\} \cup \left\{ n + \sum_{z=1}^{i-1} \ell_z + 1, \dots, n + \sum_{z=1}^i \ell_z \right\} \right) \right) = (z_1, \dots, z_{n-h})$$

to

$$\{1, \dots, \tilde{n}\} \setminus \left(\bigcup_{i=1}^h \left(\{u_i\} \cup \left\{ n + \sum_{z=1}^{i-1} \ell_z + 1, \dots, n + \sum_{z=1}^i \ell_z \right\} \right) \right) = (f_1, \dots, f_{n-h}).$$

Assume that, additionally to all of the above, $\chi(J) = U$, which implies that $\chi(\{1, \dots, n\}) = \{1, \dots, n\}$. This implies that

$$\mathbf{S} \cdot \chi(\mathbf{A}_{\{1, \dots, n\}}) = \mathbf{B}_{\{1, \dots, n\}}.$$

Since, however, $\mathbf{A}_{\{1, \dots, n\}} = \mathbf{G} = \mathbf{B}_{\{1, \dots, n\}}$, we actually obtain that

$$\mathbf{S} \cdot \chi(\mathbf{G}) = \mathbf{G},$$

i.e., $\chi|_{\{1, \dots, n\}}$ is an automorphism of $\mathcal{C}$. Let us denote $\varphi = \chi|_{\{1, \dots, n\}} \in \text{P-Aut}(\mathcal{C})$. Then, $\varphi(\{1, \dots, n\} \setminus J) = \chi(\{1, \dots, n\} \setminus J)$. Moreover, the same holds for any $\bar{\chi} \in \text{P-Iso}(\mathcal{A}, \mathcal{B})$ such that $\bar{\chi}(\{1, \dots, n\} \setminus J) = \chi(\{1, \dots, n\} \setminus J)$.

Proof of thesis d) The thesis follows trivially as an observation that any isomorphism from $\mathcal{A}$ to $\mathcal{B}$ must map each block of identical columns into a block with the same number of identical columns and follows directly from the proof of thesis c). $\square$