LESS-FM: Fine-tuning Signatures from a Code-based Cryptographic Group Action

Alessandro Barenghi[1], Jean-François Biasse[2], Edoardo Persichetti[3] and Paolo Santini[4]

Politecnico di Milano[1], University of South Florida[2], Florida Atlantic University[3], Universitá Politecnica delle Marche[4]

Abstract. Code-based cryptographic schemes are highly regarded among the quantum-safe alternatives to current standards. Yet, designing code-based signatures using traditional methods has always been a challenging task, and current proposals are still far from the target set by other post-quantum primitives (e.g. lattice-based). In this paper, we revisit a recent work using an innovative approach for signing, based on the hardness of the code equivalence problem. We introduce some optimizations and provide a security analysis for all variants considered. We then show that the new parameters produce instances of practical interest.

1 Introduction

Digital signature schemes are a fundamental primitive in modern times. In fact, such schemes offer a way to achieve authentication, one of the major cryptographic goals, and an all-important task in the digital world. Since their inception, signature schemes have traditionally been designed using classical number theory problems like integer factorization and computing discrete logarithms. The latter is an instance of a *cryptographic group action* (also described by Couveignes as a hard homogeneous space [16]). In the discrete logarithm setting, the action is defined by the exponentiation of elements in a group of prime order. Such an action satisfies a number of very good properties, due to which it was naturally chosen to be the base of several fundamental cryptographic protocols; besides signatures (DSA, ECDSA), it is worth mentioning at least El Gamal encryption, and, most importantly, the Diffie-Hellman key exchange. All of these schemes, however, will be obsolete once a quantum computer with sufficient computational power and stability is able to run attacks such as Shor's algorithm [32]. It is therefore a pressing issue to establish new algorithms for signature schemes, as highlighted by NIST's call for Post-Quantum Standardization [1].

Recently, cryptographic group actions came into the spotlight again, with several improvements over the original work of Couveignes [16] and Stolbunov [33]. This enabled many improvements in the field of *isogeny-based* cryptography, including primitives that were previously missing in the post-quantum scenario such as static-static key exchange protocols.

Our Contribution In this paper, we build on the work of [14], where the LESS signature scheme was proposed, based on the one-wayness of what is the first instance of a code-based cryptographic group action [20]. The scheme relies entirely on the hardness of finding isometries between linear codes, and thus represents a new direction in code-based cryptography. Our main contribution, is to leverage the cryptographic group action framework to introduce a number of significant optimizations. More specifically, we present two techniques that can be applied to the basic LESS protocol. The first is a generalization of the underlying identification scheme that makes use of multi-bit challenges, by changing the role of the selected challenge bits. This results in a tradeoff, with a reduction in signature size, at the expense of an increase in public key size. The second technique, instead, exploits the imbalance between the cost of different responses corresponding to the chosen challenge bits. Choosing the challenge string to have a fixed, low Hamming weight ends up in much shorter signatures, as well as providing constant-time verification. We show that the two techniques can be combined, providing a flexible and practical scheme. We give an explicit proof for the EUF-CMA security property of the original LESS scheme, with minor tweaks. This proof serves as a basis for the security of the variant schemes. Note that the multi-bit variants rely on a new problem which we call Multiple Codes

Linear Equivalence (MCLE, Problem 2), and for which we give a tight reduction to the Code Equivalence problem. Finally, we present multiple sets of parameters for a concrete instantiation of our scheme, and make practical considerations, including a comparison with the existing code-based alternatives.

The paper is organized as follows. We begin by recalling some useful background notions in Section 2. The LESS signature scheme, and the underlying group action, are presented in Section 3, which includes a dedicated proof of security for the EUF-CMA security notion. In Section 4, we present an extensive analysis of the different attacks techniques against the code equivalence problem. The various optimizations for the scheme are described in Section 5. Finally, parameters and implementation details are given in Section 6.

2 Background

We will use the following conventions throughout the rest of the paper:

a	a scalar
A	a set
$\boldsymbol{a}$	a vector
$\boldsymbol{A}$	a matrix
a	a function or relation
$\mathcal{A}$	an algorithm
$\boldsymbol{I}_n$	the $n \times n$ identity matrix
$[a; b]$	the set of integers $\{a, a+1, \ldots, b\}$
$\mathbb{U}(A)$	the uniform distribution over the set A
$\xleftarrow{\$} A$	sampling uniformly at random from A

We denote with $\mathbb{Z}_q$ the ring of integers modulo q, and with $\mathbb{F}_q$ the finite field of order q, as is customary; obviously, we have $\mathbb{Z}_q = \mathbb{F}_q$ when q is a prime. The multiplicative group of $\mathbb{F}_q$ is indicated as $\mathbb{F}_q^*$. We denote with $\mathsf{Aut}(\mathbb{F}_q)$ the group of automorphisms of the field $\mathbb{F}_q$. The sets of vectors and matrices with elements in $\mathbb{Z}_q$ (resp. $\mathbb{F}_q$) are denoted by $\mathbb{Z}_q^n$ and $\mathbb{Z}_q^{m\times n}$ (resp. $\mathbb{F}_q^n$ and $\mathbb{F}_q^{m\times n}$). We also write $\mathbb{Z}_{q,w}^n$ (resp. $\mathbb{F}_{q,w}^n$) to indicate the set of vectors with components in $\mathbb{Z}_q$ (resp. $\mathbb{F}_q$) having length n and Hamming weight w. We write $\mathsf{GL}_k(q)$ for the set of invertible $k \times k$ matrices with elements in $\mathbb{F}_q$, or simply GL_k when the finite field is implicit. Let S_n be the set of permutations over n elements. Given a vector $\boldsymbol{x} = (x_1, \cdots, x_n) \in \mathbb{F}_q^n$ and a permutation $\pi \in \mathsf{S}_n$, we write the action of π on $\boldsymbol{x}$ as $\pi(\boldsymbol{x}) = (x_{\pi(1)}, \cdots, x_{\pi(n)})$. Note that a permutation can equivalently be described as an $n \times n$ matrix with exactly one 1 per row and column. Analogously, for *linear isometries*, i.e. transformations $\tau = (\boldsymbol{v}; \pi) \in \mathbb{F}_q^{*n} \rtimes \mathsf{S}_n$, we write the action on a vector $\boldsymbol{x}$ as $\tau(\boldsymbol{x}) = (v_1 x_{\pi(1)}, \cdots, v_n x_{\pi(n)})$. Then, we can also describe these in matrix form as a product $\boldsymbol{Q} = \boldsymbol{DP}$ where $\boldsymbol{P}$ is an $n \times n$ permutation matrix and $\boldsymbol{D} = \{d_{ij}\}$ is an $n \times n$ diagonal matrix with entries in $\mathbb{F}_q^*$. We denote with M_n the set of such matrices, usually known as *monomial* matrices.

2.1 Cryptographic Group Actions

At a high level, a *group action* is an operator involving a group, for which an identity exists, and that verifies the *compatibility* property, as follows.

Definition 1. *Let X be a set and $(G, \circ)$ be a group. A group action is a mapping*

$$\begin{aligned} \star : X \times G &\to X \\ (x, g) &\to x \star g \end{aligned}$$

such that, for all $x \in X$ and $g_1, g_2 \in G$, it holds that $(x \star g_1) \star g_2 = x \star (g_1 \circ g_2)$.

A group action is usually called *cryptographic* if it satisfies some additional properties that make it interesting in a cryptographic context. In the first place, besides efficient sampling, computation, and membership testing, a cryptographic group action should certainly be *one-way*, i.e. given randomly chosen $x_1, x_2 \in X$, it should be hard to find $g \in G$ such that $x_1 \star g = x_2$ (if such a g exists). Other desirable properties include, for instance, *pseudorandomness* of the output, as well as more traditional ones such as commutativity, transitivity etc. Due to space constraints, we refer the reader to [2] for an extensive treatment of cryptographic group actions and their properties.

2.2 Code Equivalence

An $[n,k]$-*linear code* $\mathfrak{C}$ of length n and dimension k over $\mathbb{F}_q$ is a k-dimensional vector subspace of $\mathbb{F}_q^n$. It can be represented by a matrix $\boldsymbol{G} \in \mathbb{F}_q^{k\times n}$ with rank k, called *generator matrix*, whose rows form a basis for the vector space, i.e., $\mathfrak{C} = \{\boldsymbol{uG},\ \ \boldsymbol{u} \in \mathbb{F}_q^k\}$. Alternatively, a linear code can be represented as the kernel of a rank $n-k$ matrix $\boldsymbol{H} \in \mathbb{F}_q^{(n-k)\times n}$ with rank $n-k$, known as *parity-check matrix*, i.e. $\mathfrak{C} = \{\boldsymbol{x} \in \mathbb{F}_q^n : \boldsymbol{Hx}^T = 0\}$. For both representations, there exists a standard choice, called *systematic form*, which corresponds, respectively, to $\boldsymbol{G} = (\boldsymbol{I}_k \mid \boldsymbol{M})$ and $\boldsymbol{H} = (-\boldsymbol{M}^T \mid \boldsymbol{I}_{n-k})$. Generator (resp. parity-check) matrices in systematic form can be obtained very simply by calculating the row-reduced echelon form[1] starting from any other generator (resp. parity-check) matrix. We denote such a procedure by sf. The parity-check matrix is important also as it is a generator for the *dual code*, defined as the set of words that are orthogonal to the code, i.e. $\mathfrak{C}^\perp = \{\boldsymbol{y} \in \mathbb{F}_q^n \ : \ \forall \boldsymbol{x} \in \mathfrak{C},\ \ \boldsymbol{x}\cdot\boldsymbol{y}^T = 0\}$. Codes that are contained in their dual, i.e. $\mathfrak{C} \subseteq \mathfrak{C}^\perp$, are called *self-orthogonal* or *weakly self-dual*, and codes that are equal to their dual, i.e. $\mathfrak{C} = \mathfrak{C}^\perp$, are called simply *self-dual*.

The concept of *equivalence* between two codes, in its most general formulation, is defined as follows.

Definition 2 (Code Equivalence). *We say that two linear codes $\mathfrak{C}$ and $\mathfrak{C}'$ are* equivalent*, and write $\mathfrak{C} \sim \mathfrak{C}'$, if there exist a field automorphism $\alpha \in \mathsf{Aut}(\mathbb{F}_q)$ and a linear isometry $\tau = (\boldsymbol{v};\pi) \in \mathbb{F}_q^{*n} \rtimes \mathsf{S}_n$ that map $\mathfrak{C}$ into $\mathfrak{C}'$, i.e. such that $\mathfrak{C}' = \tau(\alpha(\mathfrak{C})) = \{\boldsymbol{y} \in \mathbb{F}_q^n \ : \boldsymbol{y} = \mu(\alpha(\boldsymbol{x})),\ \boldsymbol{x} \in \mathfrak{C}\}$.*

Clearly, if $\mathfrak{C}$ and $\mathfrak{C}'$ are two codes with generator matrices $\boldsymbol{G}$ and $\boldsymbol{G}'$, respectively, it holds that

$$\mathfrak{C} \sim \mathfrak{C}' \iff \exists(\boldsymbol{S};(\alpha,\boldsymbol{Q})) \in \mathsf{GL}_k \rtimes (\mathsf{Aut}(\mathbb{F}_q) \times \mathsf{M}_n) \text{ s.t. } \boldsymbol{G}' = \boldsymbol{S}\alpha(\boldsymbol{GQ}).$$

The notion we just presented is usually known as *semilinear equivalence* and it is the most generic. If the field automorphism is the trivial one (i.e. $\alpha = id$), then the notion is simply known as *linear equivalence*. If, furthermore, the monomial matrix is a permutation (i.e. $\boldsymbol{Q} = \boldsymbol{DP}$ with $\boldsymbol{D} = \boldsymbol{I}_n$), then the notion is known as *permutation equivalence*. Note that, in this work, we always work with prime fields $\mathbb{F}_q$, and therefore the last two notions are the only ones of interest to us. Finally, we state the following computational[2] problem.

Problem 1 (Code Equivalence) *Let $\boldsymbol{G}, \boldsymbol{G}' \in \mathbb{F}_q^{k\times n}$ be two generator matrices for two linearly equivalent codes $\mathfrak{C}$ and $\mathfrak{C}'$. Find a field automorphism $\alpha \in \mathsf{Aut}(\mathbb{F}_q)$ and two matrices $\boldsymbol{S} \in \mathsf{GL}_k$ and $\boldsymbol{Q} \in \mathsf{M}_n$ such that $\boldsymbol{G}' = \boldsymbol{S}\alpha(\boldsymbol{GQ})$.*

We normally refer, respectively, to *semilinear*, *linear* or *permutation equivalence problem*, according to what is the notion of code equivalence considered. Alternatively, we refer simply to the *code equivalence problem* where such distinction is not important.

[1] In general, it is possible that computing the row-reduced echelon form of $\boldsymbol{G}$ returns a matrix that does not have full rank. If so, there are procedures to obtain a matrix in systematic form by reducing with respect to a different minor (see e.g. [3]).

[2] Note that this problem is traditionally formulated as a decisional problem in literature, yet for our purposes it is more natural to present here the search version.

3 Code-based Group Actions and Applications

We begin by describing the group action associated to code equivalence. To do this we consider the set $X \subseteq \mathbb{F}_q^{k\times n}$ comprised of all full-rank $k \times n$ matrices, i.e. the set of generator matrices of $[n, k]$-linear codes, and $G = \mathsf{GL}_k \rtimes (\mathsf{Aut}(\mathbb{F}_q) \times \mathsf{M}_n)$. Note that this group is isomorphic to the group $(\mathsf{GL}_k \times (\mathbb{F}_q^*)^n) \rtimes (\mathsf{Aut}(\mathbb{F}_q) \times \mathsf{S}_n)$ if we decompose each monomial matrix $\boldsymbol{Q} \in \mathsf{M}_n$ into the product $\boldsymbol{D} \cdot \boldsymbol{P} \in (\mathbb{F}_q^*)^n \rtimes \mathsf{M}_n$; then the group operation $\circ$ is defined as

$$((\boldsymbol{S}, \boldsymbol{D}); (\alpha, \boldsymbol{P})) \circ ((\boldsymbol{S}', \boldsymbol{D}'); (\alpha', \boldsymbol{P}')) = ((\boldsymbol{S}\alpha(\boldsymbol{S}'), \boldsymbol{D} \cdot \alpha(\boldsymbol{D}'\boldsymbol{P})); (\alpha\alpha', \boldsymbol{P}\boldsymbol{P}')).$$

The group action is given by

$$\begin{array}{rccc} \star : & X \times G & \to & X \\ & (\boldsymbol{G}, (\boldsymbol{S}; (\alpha, \boldsymbol{Q}))) & \to & \boldsymbol{S}\alpha(\boldsymbol{G}\boldsymbol{Q}) \end{array}$$

It is easy to see that the action is well-formed, with the identity element being $(\boldsymbol{I}_k; (id, \boldsymbol{I}_n))$. Furthermore, it satisfies some essential properties that are of cryptographic interest. First of all, the action verifies all the basic requirements[3] such as efficient membership testing, sampling, computation etc., to which the authors in [2] assign the nomenclature of *effective*. The action is also *one-way*, based on the presumed hardness of the code equivalence problem. In fact, given $\boldsymbol{G}$ and $\boldsymbol{S}\alpha(\boldsymbol{G}\boldsymbol{Q})$, it should be infeasible to recover $\boldsymbol{S}, \alpha$ and $\boldsymbol{Q}$ in polynomial time, else this would provide a solver for the problem. Unfortunately, our group action does not satisfy some useful additional properties (as formalized in [2]). For instance, it is not *transitive*, meaning that it is not possible to connect every element of X (i.e. every generator matrix) via a group element. Most importantly, the action is not commutative, which represents a considerable obstacle in the design of cryptographic protocols. Nevertheless, it is possible to employ the group action for this purpose successfully, as we will see.

Remark 1. Note that the above formulation includes some trivial instances, for example those such that $\boldsymbol{Q} = \boldsymbol{D} \cdot \boldsymbol{I}_n$, in which case the action returns just a different generator matrix for the same code. Thus, in practice, it makes sense to consider a simplified version of the group action, where X contains only the (full-rank) generator matrices in systematic form, and $G = \mathsf{Aut}(\mathbb{F}_q) \times \mathsf{M}_n$.

The work of [14] introduces a 3-pass identification scheme, with soundness error $1/2$, which defines a zero-knowledge proof of knowledge of an isometry between codes, and is based precisely on the code equivalence setting. The authors then suggest that such a scheme can be turned into a signature scheme by applying the Fiat-Shamir transformation, without however providing full details. We give here an explicit description of such a scheme, with the addition of some minor tweaks[4].

In Table 1, for convenience, we have set $\boldsymbol{G}_0 = \boldsymbol{G}$ and $\boldsymbol{Q}_0 = \boldsymbol{I}_n$, which allows to formulate a simple and compact description. It is then immediate to verify the correctness of the scheme, which follows from the argument given in [14, Section 4]. In particular, following the notation of Table 1, when $h_i = 0$, we have $\mu_i = \tilde{\boldsymbol{Q}}_i$ and so $\hat{\boldsymbol{G}}_i = \mathsf{sf}(\boldsymbol{G}_0\mu_i) = \mathsf{sf}(\boldsymbol{G}\tilde{\boldsymbol{Q}}_i) = \tilde{\boldsymbol{G}}_i$; on the other hand, when $h_i = 1$, we have $\mu_i = \boldsymbol{Q}_1^{-1}\tilde{\boldsymbol{Q}}_i$ and so again $\hat{\boldsymbol{G}}_i = \mathsf{sf}(\boldsymbol{G}_1\mu_i) = \mathsf{sf}(\boldsymbol{G}\boldsymbol{Q}_1\boldsymbol{Q}_1^{-1}\tilde{\boldsymbol{Q}}_i) = \tilde{\boldsymbol{G}}_i$. We remark that the scheme may be equivalently instantiated by relying on permutations, instead of monomials. To this end, it is enough to replace the set group M_n with that of permutations S_n.

We now show that the LESS signature scheme is EUF-CMA secure. We begin with the following trivial result.

Lemma 1. *Let M_n be the set of monomial matrices as defined in Section 2. Then for any $\boldsymbol{A} \in \mathsf{M}_n$ and $\boldsymbol{B} \xleftarrow{\$} \mathsf{M}_n$, we have that $\boldsymbol{A}^{-1}\boldsymbol{B}$ is uniformly distributed over M_n.*

The main result is given next.

[3] Note that the concepts of *origin* is not needed and does not apply to our case.
[4] For example, the original scheme did not use public keys in systematic form.

Setup	Input parameters $q, n, k, \lambda \in \mathbb{N}$, then set $t = \lambda$. Choose matrix $\boldsymbol{G} \in \mathbb{F}_q^{k \times n}$ and hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$. Set $\boldsymbol{Q}_0 = \boldsymbol{I}_n$ and $\boldsymbol{G}_0 = \mathsf{sf}(\boldsymbol{G})$.
Private Key	Monomial matrix $\boldsymbol{Q}_1 \in \mathsf{M}_n$.
Public Key	Generator matrix $\boldsymbol{G}_1 = \mathsf{sf}(\boldsymbol{G}\boldsymbol{Q}_1)$.

SIGNER		VERIFIER
For $i = 0 \dots t-1$, choose $\tilde{\boldsymbol{Q}}_i \xleftarrow{\$} \mathsf{M}_n$ and set $\tilde{\boldsymbol{G}}_i = \mathsf{sf}(\boldsymbol{G}\tilde{\boldsymbol{Q}}_i)$.		
Set $h = \mathsf{H}(\tilde{\boldsymbol{G}}_0, \dots, \tilde{\boldsymbol{G}}_{t-1}, \boldsymbol{m})$.		
Parse $h = h_0, \dots h_{t-1}$, for $h_i \in \{0,1\}$.		
For $i = 0 \dots t-1$, compute $\mu_i = \boldsymbol{Q}_{h_i}^{-1}\tilde{\boldsymbol{Q}}_i$.		
Set $\sigma = (\mu_0, \dots \mu_{t-1}, h)$.		
	$\xrightarrow{(\boldsymbol{m},\sigma)}$	
		Parse $h = h_0, \dots h_{t-1}$, for $h_i \in \{0,1\}$.
		For $i = 0 \dots t-1$, compute $\hat{\boldsymbol{G}}_i = \mathsf{sf}(\boldsymbol{G}_{h_i}\mu_i)$.
		Accept if $\mathsf{H}(\hat{\boldsymbol{G}}_0, \dots, \hat{\boldsymbol{G}}_{t-1}, \boldsymbol{m}) = h$.

Table 1: The LESS Signature Scheme.

Theorem 1. *The LESS signature scheme described in Table 1 is existentially unforgeable under adaptive chosen-message attacks, in the random oracle model, under the hardness of the linear code equivalence problem.*

Proof. Let $\mathcal{A}$ be a polynomial-time EUF-CMA adversary for the signature scheme, as defined in Definition 4. $\mathcal{A}$ takes as input a verification key vk, then performs a polynomial number of signing queries, say Q_s, and a polynomial number of random oracle queries, say Q_r. Eventually, $\mathcal{A}$ outputs a forgery (m^*, σ^*), with a certain probability of success p. We now show how to construct an adversary $\mathcal{A}'$ that is able to solve the linear code equivalence problem. $\mathcal{A}'$ will interact with $\mathcal{A}$ and use it as a subroutine, playing the role of the challenger in the EUF-CMA game and simulating correct executions of the LESS protocol, without obviously having access to the private key.

To begin with, $\mathcal{A}'$ is given an instance $(\boldsymbol{G}, \boldsymbol{G}' = \boldsymbol{S}\boldsymbol{G}\boldsymbol{Q})$ of Problem 1, which he sets up as public key in the simulated LESS protocol. $\mathcal{A}'$ will answer signing queries and random oracle queries as described below; to ensure consistency of the simulation, the queries will be tracked with the help of a table T, initially empty, where the calls to the random oracle will be stored as they are answered, in the form of pairs $(input, output)$.

Setup. Set $\boldsymbol{G}_0 = \boldsymbol{G}$ and $\boldsymbol{G}_1 = \boldsymbol{G}'$.

Random Oracle Queries. In a random oracle query, $\mathcal{A}$ submits an input $\boldsymbol{x}$ of the form $(\hat{\boldsymbol{G}}_0, \dots, \hat{\boldsymbol{G}}_{t-1}, \boldsymbol{m})$ and expects to receive a λ-bit string h. $\mathcal{A}'$ proceeds as follows:

1. Look up $\boldsymbol{x}$ in T. If $(\boldsymbol{x}, h) \in \mathsf{T}$ for some h, return h and halt.
2. Generate uniformly at random a λ-bit string h.
3. Add $(\boldsymbol{x}, h)$ to T.
4. Return h.

Signing Queries. In a signing query, $\mathcal{A}$ submits a message m and expects to receive a valid signature σ for it. $\mathcal{A}'$ proceeds as follows:

1. Generate uniformly at random a λ-bit string h.
2. Generate uniformly at random matrices $\hat{\boldsymbol{Q}}_0, \dots, \hat{\boldsymbol{Q}}_{t-1}$.
3. Set $\mu_i = \hat{\boldsymbol{Q}}_i$.
4. Return signature $\sigma = (\mu_0, \dots \mu_{t-1}, h)$.

After that, $\mathcal{A}'$ adjusts his registry of queries by recording the query corresponding to h in table T. More specifically, $\mathcal{A}'$ will parse $h = h_0, \dots h_{t-1}$, where $h_i \in \{0, 1\}$, then compute $g_i = \mathsf{sf}(\boldsymbol{G}_{h_i}\mu_i)$ and finally set h to be the response to the random oracle query with input $(\hat{\boldsymbol{G}}_0, \dots, \hat{\boldsymbol{G}}_{t-1}, \boldsymbol{m})$. Note that, due to Lemma 1, signatures produced in this way are indistinguishable from authentic signatures, since they follow the exact same distribution.

The simulation halts if, during a signing query, the input to the random oracle had already been queried before, in which case the signing query outputs $\perp$ instead. Note that this can only happen with negligible probability; more precisely, the probability is at most q'/K^t, where $q' = q_s + q_r$ is the total number of queries performed, and K is an upper bound on the probability of finding a collision, i.e. sampling two monomial matrices that lead to linearly equivalent codes (see Proposition 10 of Appendix B).

Once $\mathcal{A}$ has finished performing queries, it will output a forgery $(\boldsymbol{m}^*, \sigma^*)$, where $\sigma^* = (\mu_0^*, \dots \mu_{t-1}^*, h_0^*, \dots h_{t-1}^*)$, that successfully passes verification. At this point, $\mathcal{A}'$ rewinds his tape and plays the simulation again, in the exact same way, except that one of the random oracle queries is answered differently. By the Forking Lemma (see [10]), $\mathcal{A}$ will now output, with non-negligible probability, a forgery $(\boldsymbol{m}', \sigma')$, where $\sigma' = (\mu_0', \dots \mu_{t-1}', h_0', \dots h_{t-1}')$, for the same message $\boldsymbol{m}' = \boldsymbol{m}^*$ and the same random oracle input $(\hat{\boldsymbol{G}}_0, \dots, \hat{\boldsymbol{G}}_{t-1}, \boldsymbol{m})$, such that $\sigma' \neq \sigma^*$. Let j be the index such that $h_j' \neq h_j^*$; then $\mathsf{sf}(\boldsymbol{G}_{h_j'}\mu_j') = \mathsf{sf}(\boldsymbol{G}_{h_j^*}\mu_j^*)$, which means that the monomial matrix $\mu_j^*\mu_j^{-1}$ is a solution to the linear code equivalence problem as desired. □

4 Solving the Code Equivalence Problem

As proven in [27], the permutation equivalence problem is unlikely to be NP-complete, since this property would imply a collapse of the polynomial hierarchy. Yet, while the problem can be efficiently solved for some families of codes, there are however many instances that, after almost 40 years of study, are still intractable. Below, we discuss the best known solvers for the code equivalence problem, in order to carefully assess its security. We will first deal with the case of permutation equivalence, and report the complexity of all techniques to solve this problem. Then, we will see how these techniques adapt to the case of linear equivalences.

We start by describing how the number of codewords of some desired weight (without considering scalar multiples) can be estimated for a random code.

Proposition 1. *Let $\mathfrak{C} \subseteq \mathbb{F}_q^n$ be a random linear code with length n and dimension k. Let A^w be the set of codewords of $\mathfrak{C}$ such that*

- *if $\boldsymbol{c} \in \mathfrak{C}$ has weight w, then $b\boldsymbol{c} \in A^w$ for some $b \in \mathbb{F}_q^*$;*
- *$b\boldsymbol{c} \neq b'\boldsymbol{c}'$, for all $b, b' \in \mathbb{F}_q^*$ and all $\boldsymbol{c} \neq \boldsymbol{c}' \in A^w$.*

Then, the average cardinality of A^w is given by

$$N_w = \binom{n}{w}(q-1)^{w-2}q^{k-n+1}.$$

Proof. See Appendix C.

We anticipate that, in the subsequent analysis, we will frequently make use of this result. In the following, we recall a trivial property of code equivalence.

Proposition 2. *Let $\mathfrak{C}_1, \mathfrak{C}_2 \subseteq \mathbb{F}_q^n$ be two linear codes with dimension k, and let $\mathfrak{C}_1^\perp$, $\mathfrak{C}_2^\perp$ be their duals. Then*

i. if $\pi \in \mathsf{S}_n$ is such that $\pi(\mathfrak{C}_1^\perp) = \mathfrak{C}_2^\perp$, then also $\pi(\mathfrak{C}_1) = \mathfrak{C}_2$;
ii. if $\mu \in \mathsf{M}_n$ is such that $\mu(\mathfrak{C}_1^\perp) = \mathfrak{C}_2^\perp$, then also $\mu'(\mathfrak{C}_1) = \mathfrak{C}_2$, where μ' is derived from μ by taking the reciprocal of the scaling factors.

The above proposition is fundamental to understand the hardness of solving the code equivalence problem. Indeed, the problem can equivalently be solved by looking at the given codes, or at their duals. For the sake of simplicity, in the following we will describe all the algorithms and procedures by considering solely the initially given codes; to derive the corresponding complexity for the attack on the duals, it is enough to replace k with $n-k$ in all the provided formulas. Finally, we will first focus on methods to solve the permutation equivalence problem, and then describe how they can be adapted to tackle the more general case of linear equivalences.

4.1 Leon's Algorithm

Chronologically, the first method capable of solving the code equivalence problem is due to Leon [24], and is based on the following reasoning. Let $\mathfrak{C}_1$ and $\mathfrak{C}_2$ be two linear codes with length n and dimension k, and $\pi \in \mathsf{S}_n$ such that $\pi(\mathfrak{C}_1) = \mathfrak{C}_2$. Then, for each $X \subseteq \mathfrak{C}_1$ there must exist $Y \subseteq \mathfrak{C}_2$, having the same cardinality of X and such that $\pi(X) = Y$. Thus, among all the maps from X to Y, there must necessarily also be those mapping $\mathfrak{C}_1$ into $\mathfrak{C}_2$. In [24], Leon proposes an algorithm that constructs the group of permutations between two sets, with a running time that is polynomial in the cardinality of the sets. Starting from the observation that permutations (as well as linear and semilinear isometries) preserve the Hamming weight, Leon suggests to use the sets of codeword with a fixed (and properly low) weight w.

Let $\mathsf{Mor}_{\mathsf{S}_n}(B_1^w, B_2^w)$ be the group of all permutations $\pi \in \mathsf{S}_n$ such that $\pi(B_1^w) = B_2^w$. In a nutshell, Leon's algorithm operates as follows:

1. for $i = 1, 2$, compute $B_i^w = \{\boldsymbol{c} \in \mathfrak{C}_i \mid \mathrm{wt}(\boldsymbol{c}) = w\}$;
2. find generators of $\mathsf{Mor}_{\mathsf{S}_n}(B_1^w, B_2^w)$;
3. check if there exists $\pi \in \mathsf{Mor}_{\mathsf{S}_n}(B_1^w, B_2^w)$ such that $\pi(\mathfrak{C}_1) = \mathfrak{C}_2$.

As Leon proves in the original paper, the complexity of the second and third steps is polynomial in the cardinality of B_1^w and B_2^w, which we estimate as $(q-1)N_w$ (see Proposition 1). This also allows us to properly choose the value of w. Indeed, when w is too high, then also N_w may become so high that the second step (i.e., the construction of $\mathsf{Mor}_{\mathsf{S}_n}(B_1^w, B_2^w)$) becomes too time-consuming. On the other hand, if w is too low, then the sets B_1^w and B_2^w are rather small and do not possess enough structure, in the sense that there may exist a very large number of maps from B_1^w to B_2^w (so, the third step becomes too time-consuming). Heuristically, optimal values of w are those that are slightly larger than the minimum distance of the codes (which can be estimated with the Gilbert-Varshamov distance). Indeed, this normally guarantees that the sets B_1^w and B_2^w are moderately small and, at the same time, contain a sufficient number of codewords.

It is evident that the bottleneck, in the algorithm's time complexity, is given by the codeword search in Step 1. Given that we are interested in producing low weight codewords, the best option is to rely on Information-Set Decoding (ISD) type algorithms. We refer the reader to [6] for a review of ISD algorithms in the binary case, and present a brief treatment of the non-binary case in Appendix E.

Putting everything together, we are able to assess the complexity of Leon's algorithm as follows.

Proposition 3. *Let $\mathfrak{C}_1 \subseteq \mathbb{F}_q^n$ be a random code with dimension k, and $\mathfrak{C}_2 = \pi(\mathfrak{C}_1)$, with π being a randomly picked permutation. Then, the complexity of using Leon's algorithm to solve the permutation equivalence problem on the pair $\mathfrak{C}_1$, $\mathfrak{C}_2$ is at least*

$$O\left(C_{ISD}(q, n, k, w) \cdot 2 \sum_{i=1}^{N_w} \frac{1}{i}\right),$$

where N_w is the number of weight w-codewords (neglecting scalar multiples) estimated as in Proposition 1, w is chosen as the smallest integer such that $N_w \geq 2$, and $C_{ISD}(q, n, k, d_{GV})$ denotes the time complexity of an ISD algorithm, searching for a codeword with weight d_{GV} in a linear code with length n, dimension k and defined over $\mathbb{F}_q$.

Proof. We first consider the cost of the codeword enumeration in Step 1. To set w, we make the following conservative reasoning. When B_1^w and B_2^w are too small, there may exist a prohibitively large number of permutations mapping B_1^w into B_2^w. Assume that w is so small that $N_w = 1$; then, the sets B_1^w and B_2^w are formed by $q-1$ codewords, and we trivially have that $\mathsf{Mor}_{\mathsf{S}_n}(B_1^w, B_2^w)$ contains at least $(n-w)!$ permutations. Unless n is extremely low, such a number is prohibitively large so that the third step becomes too time-consuming. To avoid this, we must require B_1^w and B_2^w to show sufficient structure, i.e., we require N_w to be sufficiently large. However, setting a precise bound on the size of N_w requires troublesome computations. Hence, conservatively, we assume that whenever $N_w \geq 2$ the size of B_1^w and B_2^w is large enough to make the set $\mathsf{Mor}_{\mathsf{S}_n}(B_1^w, B_2^w)$ small enough, so that testing all the contained permutations can be done in reasonable time.

Note that, in practice, when the value of w is properly chosen, it happens that the second and third steps of Leon's algorithm run with a computational complexity that is significantly lower than that of codewords enumeration. Hence, conservatively, we assess the cost of Leon's algorithm with that of its first step. To this end, we model an ISD algorithm as an oracle that, for each call, returns a random codeword with the desired weight, using time complexity given by $C_{ISD}(q, n, k, w)/N_w$. Let us first focus on $\mathfrak{C}_1$: since the code contains N_w codewords of the desired weight, the first ISD call will return one of them with time complexity $C_{ISD}(q, n, k, w)/N_w$. In the second call, we desire a distinct codeword, so we have a complexity that is $C_{ISD}(q, n, k, w)/(N_w - 1)$. If we iterate this reasoning, we get that the codewords enumeration for $\mathfrak{C}_1$ takes time

$$O\left(C_{ISD}(q, n, k, w) \cdot \sum_{i=1}^{N_w} \frac{1}{i}\right).$$

Finally, we double the cost because the enumeration has to be repeated for $\mathfrak{C}_2$. □

4.2 Beullens' Algorithm

In a recent work [13], Beullens introduced a novel approach to solve the code equivalence problem. Basically, Beullens' algorithm can be thought of as a refinement of Leon's algorithm, which tries to reduce the computational complexity by avoiding to compute the whole set of codewords with some fixed weight. Before entering into the details of the algorithm, we hint at its flavour with a preliminary reasoning. As before, we denote with $\mathfrak{C}_1$ and $\mathfrak{C}_2$ two linear codes such that $\pi(\mathfrak{C}_1) = \mathfrak{C}_2$ for some permutation π, and indicate with $A_1^w \subseteq \mathfrak{C}_1$ and $A_2^w \subseteq \mathfrak{C}_2$ the sets of codewords with fixed weight w (again, neglecting multiple scalars). Let $X \subseteq A_1^w$ and $Y \subseteq A_2^w$ and, for $(\boldsymbol{x}, \boldsymbol{y}) \in X \times Y$, denote

$$\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, \boldsymbol{y}) = \{\varphi \in \mathsf{S}_n \mid \varphi(\boldsymbol{x}) = \boldsymbol{y}\}.$$

In other words, $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, \boldsymbol{y})$ is the set of all permutations mapping $\boldsymbol{x}$ into $\boldsymbol{y}$. Note that $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, \boldsymbol{y})$ is non-empty if and only if the multisets defined by the entries of $\boldsymbol{x}$ and $\boldsymbol{y}$ are identical. Furthermore, if $\boldsymbol{x}$ and $\boldsymbol{y}$ are such that $\pi(\boldsymbol{x}) = \boldsymbol{y}$, then we have $\pi \in \mathsf{Mor}_{\mathsf{S}_n}$.

In a nutshell, Beullens' algorithm works as follows. First, it computes at random the sets X and Y, then searches for collisions in the multisets defined by the entries of all pairs $\boldsymbol{x} \in X$, $\boldsymbol{y} \in Y$. For each collision found, the algorithm computes the set $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, \boldsymbol{y})$, containing candidates for the permutation transforming $\mathfrak{C}_1$ into $\mathfrak{C}_2$. Finally, it checks each candidate permutation $\varphi \in \mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, \boldsymbol{y})$, by testing if it indeed holds that $\varphi(\mathfrak{C}_1) = \mathfrak{C}_2$. If this relation does not hold true, the algorithm restarts by picking another colliding pair $(\boldsymbol{x}, \boldsymbol{y})$.

The procedure is detailed in Algorithm 1. We have used MS to denote the function returning the multiset of entries for the input vector, while Lex denotes the function that, on input $\boldsymbol{a}$, returns the first element, in lexicographical order, of the set $\{\mathsf{MS}(b\boldsymbol{a}) \mid b \in \mathbb{F}_q^*\}$. To understand how these functions operate, we have reported a toy example in Figure 1.

$\mathsf{MS}(1\boldsymbol{a}) = \{1,2,2,3,3,3,4\}$
$\mathsf{MS}(2\boldsymbol{a}) = \{1,1,1,2,3,4,4\}$
$\mathsf{MS}(3\boldsymbol{a}) = \{1,1,2,3,4,4,4\}$
$\mathsf{MS}(4\boldsymbol{a}) = \{1,2,2,2,3,3,4\}$

(a)

$\mathbf{1}^\circ$ - $\mathsf{MS}(2\boldsymbol{a}) = \{1,1,1,2,3,4,4\}$
$\mathbf{2}^\circ$ - $\mathsf{MS}(3\boldsymbol{a}) = \{1,1,2,3,4,4,4\}$
$\mathbf{3}^\circ$ - $\mathsf{MS}(4\boldsymbol{a}) = \{1,2,2,2,3,3,4\}$
$\mathbf{4}^\circ$ - $\mathsf{MS}(1\boldsymbol{a}) = \{1,2,2,3,3,3,4\}$

(b)

Fig. 1: Example of lexicograph ordering, for the finite field with $q = 5$ elements and a vector $\boldsymbol{a} = (0,3,2,0,0,3,3,2,4,1)$, for which $\mathsf{Lex}(\boldsymbol{a}) = \{1,1,1,2,3,4,4\}$. Figure (a) shows the multisets of entries for all scalar multiples of $\boldsymbol{a}$, while figure (b) reports the lexicographic order of such multisets.

The computational complexity of Beullens' algorithm is described in the following proposition.

Proposition 4. *Let $\mathfrak{C}_1 \subseteq \mathbb{F}_q^n$ be a random code with dimension k, and $\mathfrak{C}_2 = \pi(\mathfrak{C}_1)$, with π being a random permutation. Let $L, w \in \mathbb{N}$, with $w \leq n$ and $L \leq N_w$, where N_w is computed as in Proposition 1. Then, the computational complexity of Beullens's algorithm with parameters L and w, to solve the permutation equivalence problem on $\mathfrak{C}_1$ and $\mathfrak{C}_2$, is at least*

$$O\left(\frac{1}{\beta(w,L)} \cdot \left(\frac{2C_{ISD}(q,n,k,w)}{\sum_{i=0}^{L-1}(N_w - i)^{-1}} + C_{perm}\right)\right),$$

where $C_{ISD}(q,n,k,w)$ denotes the time complexity of an ISD algorithm, searching for a codeword with weight w in a linear code with length n, dimension k and defined over $\mathbb{F}_q$, and

$$C_{perm} = Lw(q-2)\big(1 + \log_2(L)\big) + \frac{M''}{M}(n-w)!(m!)^{q-1}(n+k^2),$$

with $m = \max\{1\ ,\ \lceil w/(q-1)\rfloor\}$, $\beta(w,L) = 1 - \binom{N_w - L}{L}/\binom{N_w}{L}$, $M = L^2/N_w$ and $M' = (1 - \binom{w+q-3}{w-1}^{-1})M + L^2\binom{w+q-3}{w-1}^{-1}$.
If $L \ll N_w$, q is rather large and $\binom{w+q-3}{w-1}^{-1} \ll L^2$, the complexity simplifies as

$$O\left(\frac{2L \cdot C_{ISD}(q,n,k,w) + (n-w)!(n+k^2)}{N_w\left(1 - 2^{L\log_2(1-L/N_w)}\right)}\right).$$

Proof. The proof is provided in Appendix C.

Improving the Permutation Reconstruction In Proposition 4, the term C_{perm} corresponds to the cost of reconstructing the permutation mapping the pairs of codewords in P. Notice that it is exponential in n, so that it may rapidly grow (especially when w is low). However, as Beullens points out, when $\mathfrak{C}_1$ does not contain codewords having the same multiset of entries, there is a very efficient way to reconstruct the permutation mapping the two codes. Indeed, let us assume that P does not contain "bad" collisions, and that it contains a sufficiently large number of pairs $\{\boldsymbol{x}, \boldsymbol{y}\}$. Under such hypothesis, there is a very simple way to reconstruct the target permutation, exploiting a crucial observation: if we know that $\pi(\boldsymbol{x}) = \boldsymbol{y}$ for some permutation π (which are trying to recover), and $x_i \neq y_j$, then we know that π does not map i to j. Considering all pairs of indexes (i,j), we gather a significant amount of information about π or, to put it differently, we filter out a wide number of candidates. Exploiting all pairs in P, and putting all the information together, it becomes possible to recover the secret π with a simple procedure as the one in Algorithm 2 (in other words, one may replace the instructions in Lines 12–17 with those in Algorithm 2).

To estimate the probability that Algorithm 2 is successful, and to additionally derive the minimum number of elements that P should contain, we rely on the following proposition.

Data: Subset size $L \in \mathbb{N}$, weight $w \in \mathbb{N}$, ISD routine
Input: linear codes $\mathfrak{C}_1, \mathfrak{C}_2 \subseteq \mathbb{F}_q^n$ with dimension k
Output: permutation $\varphi \in \mathsf{S}_n$ such that $\varphi(\mathfrak{C}_1) = \mathfrak{C}_2$

/* Produce a list X of L codewords from $\mathfrak{C}_1$ with weight w */
1 $X = \varnothing$;
2 **while** $|X| < L$ **do**
3 | Call ISD to find $\boldsymbol{x} \in \mathfrak{C}_1$ with weight w;
4 | $X \leftarrow X \cup \{\boldsymbol{x}, \mathsf{Lex}(\boldsymbol{x})\}$;

/* Produce a list Y of L codewords from $\mathfrak{C}_2$ with weight w */
5 $Y = \varnothing$;
6 **while** $|Y| < L$ **do**
7 | Call ISD to find $\boldsymbol{y} \in \mathfrak{C}_2$ with weight w;
8 | $Y \leftarrow Y \cup \{\boldsymbol{y}, \mathsf{Lex}(\boldsymbol{y})\}$;

/* Find collisions between the lists X and Y */
9 **for** $\{\boldsymbol{x}, \boldsymbol{y}\} \in X \times Y$ **do**
10 | **if** $\mathsf{Lex}(\boldsymbol{x}) = \mathsf{Lex}(\boldsymbol{y})$ **then**
11 | | $P \leftarrow P \cup \{\boldsymbol{x}, \boldsymbol{y}\}$;

/* Go through all permutations mapping the pairs of codewords in P, and check whether they also define an equivalence between $\mathfrak{C}_1$ and $\mathfrak{C}_2$ */
12 **for** $\{\boldsymbol{x}, \boldsymbol{y}\} \in P$ **do**
13 | Compute $b \in \mathbb{F}_q^*$ such that $\mathsf{MS}(\boldsymbol{x}) = \mathsf{MS}(b\boldsymbol{y})$;
14 | Compute $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, b\boldsymbol{y})$;
15 | **for** $\varphi \in \mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, b\boldsymbol{y})$ **do**
16 | | **if** $\varphi(\mathfrak{C}_1) = \mathfrak{C}_2$ **then**
17 | | | **return** φ

Algorithm 1: Beullens' algorithm to solve the permutation equivalence problem

Proposition 5. *Let $\mathfrak{C}_1 \subset \mathbb{F}_q^n$ be a random linear code with dimension k, and let $\mathfrak{C}_2 = \pi(\mathfrak{C}_1)$ with π being a random permutation. Let N_w estimate the number of weight-w codewords in $\mathfrak{C}_1$, as in Proposition 1, and P be the list obtained through the instructions in lines 1–11 of Algorithm 1, using parameters L and w. Let $\binom{w+q-3}{w-1}^{-N_w} \cdot \prod_{j=1}^{N_w-1} \binom{N_w}{2} - j \ll 1$. Then, Algorithm 2 retrieves the correct permutation π with probability*

$$\gamma(L, N_w) = \left(1 - \left(\left(1 - \frac{w}{n}\right)\left(1 - \frac{w-1}{n-1}\right) + \frac{w(w-1)}{n(n-1)(q-1)}\right)^{\frac{L^2}{N_w}}\right)^{n(n-1)}.$$

Proof. We first want to assure that, with very high probability, the code $\mathfrak{C}_1$ does not contain weight-w codewords having the same multiset of entries. When this condition is satisfied, then the set P contains only "good" collisions and Algorithm 2 returns a valid permutation. To derive the probability with which such an event occurs, we assume that each weight-w codeword is random. Then, we can consider that the multiset of entries of each weight-w codeword is obtained by picking at random an element among the $\binom{w+q-3}{w-1}$ multisets formed by a 1 and by other $w-1$ elements from $\mathbb{F}_q^*$. Hence, the probability that all the codewords with weight w (in number equal to N_w) have distinct multisets is obtained as

$$\frac{\binom{w+q-3}{w-1} - 1}{\binom{w+q-3}{w-1}} \frac{\binom{w+q-3}{w-1} - 2}{\binom{w+q-3}{w-1}} \cdots \frac{\binom{w+q-3}{w-1} - (N_w - 1)}{\binom{w+q-3}{w-1}} = \frac{\prod_{j=1}^{N_w-1} \binom{w+q-3}{w-1} - j}{\binom{w+q-3}{w-1}^{N_w}}.$$

When this probability is extremely low, one can guarantee that P is well-formed (i.e., does not contain bad collisions).

Input: list P, containing couples $\{\boldsymbol{x}, \boldsymbol{y}\} \in \mathbb{F}_q^n \times \mathbb{F}_q^n$
Output: permutation $\pi \in \mathsf{S}_n$, or report failure

$\boldsymbol{U} \leftarrow n \times n$ matrix made of all ones;
for $\{\boldsymbol{x}, \boldsymbol{y}\} \in P$ **do**
  for $i \in \{1, \cdots, n\}$ **do**
    for $j \in \{1, \cdots, n\}$ **do**
      if $x_i \neq y_j$ **then**
        $u_{i,j} = 0$

/* Use $\boldsymbol{U}$ to reconstruct the permutation; if not possible, report failure */
if $\boldsymbol{U}$ is a permutation matrix **then**
  return π;
else
  report failure

Algorithm 2: Fast permutation recovery, for the permutation equivalence version of Beullens' algorithm

Now, we estimate how the size of P contributes to the success probability of the permutation reconstruction phase. To do this, let us consider a pair $\{\boldsymbol{x}, \boldsymbol{y}\} \in P$, and focus on a generic j-th entry of $\boldsymbol{x}$, that is, x_j. We consider the probability that a generic index ℓ is such that $x_i = y_\ell$, but $\pi(i) \neq \ell$. This can happen only in two cases: either $x_j = y_\ell = 0$, or $x_j = y_\ell \neq 0$. By putting everything together, we have that $x_j = y_\ell$ happens with probability

$$\rho = \frac{\binom{n-w}{2}}{\binom{n}{2}} + \frac{\binom{w}{2}}{\binom{n}{2}} \cdot \frac{1}{q-1} = \left(1 - \frac{w}{n}\right)\left(1 - \frac{w-1}{n-1}\right) + \frac{w(w-1)}{n(n-1)(q-1)}.$$

The algorithm will fail, i.e., will not be able to recover the valid permutation, if for all pairs $\{\boldsymbol{x}, \boldsymbol{y}\} \in P$ we have $x_j = y_\ell$. This happens with probability $1 - \rho^{|P|}$: in other words, this is the probability that we correctly guess $\pi(j) \neq \ell$. Remember that, on average, $|P| = L^2/N_w$. Since we have to guess correctly for a total of $n(n-1)$ pairs of indexes, we finally estimate the success probability as $\gamma(L, N_w) = \left(1 - \rho^M\right)^{n(n-1)}$. □

When one uses Algorithm 2 to reconstruct the permutation between the two codes, the complexity of Beullens' algorithm reduces. To this end, we consider the following proposition.

Proposition 6. *Let w and q be such that the condition of Proposition 5 is satisfied. Then, Beullen's algorithm using Algorithm 2 to reconstruct the permutation runs in time*

$$O\left(\frac{2L \cdot C_{ISD}(q, n, k, w)}{N_w \left(1 - 2^{L \log_2(1 - L/N_w)}\right)}\right),$$

and succeeds with probability $\gamma(L, N_w)$, computed as in Proposition 5.

Proof. With respect to the estimate provided in Proposition 4, we neglect the cost of reconstructing the permutation (that is, the term C_{perm}). □

Remark 2. In practice, Beullens' algorithm is able to outperform Leon only in some specific cases, that is, the ones in which the finite field is rather large. Indeed, this is the case when multiset of entries of weight-w codewords do not collide with high probability. Otherwise, the permutation cannot be reconstructed with the fast method explained in Algorithm 2, and one has to go through all permutations mapping all pairs in the list P. The number of such permutations is given by $(n-w)!(m!)^{q-1}$, and m grows with q. For instance, consider that for $q = 2$ we have that the number of permutations is maximum, and corresponds to $(n-w)!w!$: this number explodes unless n is extremely small.

Even when the finite field is large enough, so that one can rely on the fast permutation reconstruction phase, we observe that the advantage with respect to Leon's algorithm is rather limited. Indeed, assuming that both algorithms are optimized with the same value of w, we have that the speed-up of Beullens' algorithm is given by $\frac{N_w \ln(N_w)}{L}$. As suggested in [13], Beullens' approach has a non-trivial success probability when $L = \Theta\left(\sqrt{N_w \ln(n)}\right)$, which means that the advantage over Leon can be roughly assessed as

$$\frac{\sqrt{N_w}\ln(N_w)}{\sqrt{\ln(n)}} = 2^{0.5\log_2(N_w)+\log_2(\ln(N_w))-0.5\log_2(\ln(n))}.$$

In practice, Leon can use slightly smaller values of w, so that the real advantage is actually lower.

4.3 The Support Splitting Algorithm

The Support Splitting Algorithm (SSA), introduced by Sendrier [30], follows a very different approach from the previous ones, based on the idea of *signature function*. This is defined as a map $\mathcal{S}$ such that, for a given code $\mathfrak{C}$ with length n and for each position $i \in [0; n-1]$, verifies

$$\mathcal{S}(\mathfrak{C}, i) = \mathcal{S}\big(\pi(\mathfrak{C}), \pi(i)\big), \ \ \forall \pi \in \mathsf{S}_n.$$

A signature function is said to be *fully discriminant* if $\mathcal{S}(\mathfrak{C}, i) \neq \mathcal{S}(\mathfrak{C}, j), \ \ \forall i \neq j$. Once in possession of a fully discriminant signature, a permutation π between two codes $\mathfrak{C}_1$ and $\mathfrak{C}_2$ can immediately be recovered, since

$$\mathcal{S}(\mathfrak{C}_1, i) = \mathcal{S}(\mathfrak{C}_2, j) \iff j = \pi(i).$$

Assuming that such a fully discriminant function $\mathcal{S}$ is available, SSA essentially consists of searching for collisions between the sets of values $\mathcal{S}(\mathfrak{C}_1, i)$ and $\mathcal{S}(\mathfrak{C}_2, j)$, for $(i, j) \in [0; n-1] \times [0; n-1]$. We point out that the existence of such a function (and one that doesn't require unfeasible computation) is clearly not guaranteed, a priori. When a fully discriminant signature is not available, one can use refinements, new computations and combinations of signatures, that proceed until a fully discriminant function is obtained.

The signature function proposed by Sendrier in [30] is based on the *hull space* of a code, that is, the intersection between a code and its dual. In particular, to create a dependence between the signature value and the code positions, one can *puncture* the code, i.e. remove coordinates from the codewords. Putting these considerations together, in [30, Section 5.2] Sendrier proposes to build a signature as

$$\mathcal{S}(\mathfrak{C}, i) := \left\{ \mathsf{Wef}\Big(\mathfrak{H}\big(\mathfrak{C}_{\setminus i}\big)\Big) \ , \ \mathsf{Wef}\Big(\mathfrak{H}\big(\mathfrak{C}^{\perp}_{\setminus i}\big)\Big) \right\}, \tag{1}$$

where $\mathfrak{C}_{\setminus i}$ is the code obtained from $\mathfrak{C}$ punctured in position i, $\mathfrak{H}$ denotes the hull and Wef denotes the Weight Enumerator Function. The hull computation requires simple linear algebra, and comes with a cost of $O(n^3)$ operations in the finite field. To compute the weight enumerator of a code, one usually needs to enumerate all of its codewords: assuming that the hull has dimension h, we can use $O(nq^h)$ as an estimate for the cost of each Wef computation. Finally, heuristically, we observe that using $\ln(n)$ refinements is enough to obtain a fully discriminant signature. Putting everything together, we are able to estimate the complexity of SSA as follows.

Proposition 7. *Let $\mathfrak{C}_1 \subseteq \mathbb{F}_q^n$ be a random code with dimension k, with hull of dimension $h \leq \min\{k \ , \ n-k\}$. Let $\mathfrak{C}_2 = \pi(\mathfrak{C}_1)$, with π being a randomly picked permutation. Then, the computational complexity of using the SSA algorithm to solve the permutation equivalence problem on $\mathfrak{C}_1$ and $\mathfrak{C}_2$ can be estimated as*

$$O\left(n^3 + n^2 q^h \ln(n)\right).$$

For complete details on how the computational complexity is assessed, we refer the interested reader to [30].

As highlighted by the above formula, the hull dimension plays a crucial role in defining the algorithm performances, from a complexity standpoint. We remark that, for random codes, this dimension is with high probability equal to a small constant [31], de facto making SSA a polynomial-time solver for permutation equivalence. On the other hand, SSA is very inefficient for codes that have a large hull. In fact, since the time complexity is dominated by q^h, SSA becomes quickly unfeasible as h grows. This is, for instance, the case of self-dual codes, for which $h = k$: for such codes, SSA can be made arbitrarily hard by choosing codes with a sufficiently large dimension.

4.4 The BOS Algorithm

In 2019, Bardet et al. [7] proposed a new method to solve the permutation equivalence problem, which fully exploits the connection between the permutation equivalence problem and the one called *graph isomorphism*. In this section we briefly recall this approach, which we refer to as BOS using the authors' initials.

Let us first define the Weighted Graph Isomorphism (WGI) problem. We consider only undirected and weighted graphs. Each graph of this type can be fully represented by an *adjacency matrix*, such that the element in position (i, j) is equal to ξ if and only the vertices i and j are connected by an edge with weight ξ. Two graphs are isomorphic if one is obtained from the other by permuting its vertices, while keeping both adjacent vertices and edges weights. Equivalently, two graphs with adjacency matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are isomorphic if and only if there exists a permutation matrix $\boldsymbol{P}$ such that $\boldsymbol{A}_2 = \boldsymbol{P}^\top \boldsymbol{A}_1 \boldsymbol{P}$.

The main observation behind the BOS method is reported below [7, Theorem 5].

Theorem 2. *Let $\mathfrak{C}_1, \mathfrak{C}_2 \subseteq \mathbb{F}_q^n$ be linear codes with dimension k and trivial hull. For $i = 1, 2$, let $\boldsymbol{G}_i$ be a generator of the code $\mathfrak{C}_i$, and*

$$\boldsymbol{A}_{\mathfrak{C}_i} = \boldsymbol{G}_i^\top \left(\boldsymbol{G}_i \boldsymbol{G}_i\right)^{-1} \boldsymbol{G}_i.$$

Then, $\mathfrak{C}_1$ and $\mathfrak{C}_2$ are permutation equivalent, i.e., there exists $\pi \in \mathsf{S}_n$ such that $\pi(\mathfrak{C}_1) = \mathfrak{C}_2$ if and only if $\boldsymbol{A}_{\mathfrak{C}_1} = \boldsymbol{P}^\top \boldsymbol{A}_{\mathfrak{C}_1} \boldsymbol{P}$, where $\boldsymbol{P}$ is the permutation matrix associated to π.

The above theorem is at the core of the BOS approach to solve the permutation equivalence problem. Indeed, $\boldsymbol{A}_{\mathfrak{C}_1}$ and $\boldsymbol{A}_{\mathfrak{C}_2}$ are interpreted as the adjacency matrices of two graphs, and hence are given as input to some routine which solves the WGI problem. The output permutation $\boldsymbol{P}$ corresponds to a permutation mapping also the two codes. Given that to compute $\boldsymbol{A}_{\mathfrak{C}_1}$ and $\boldsymbol{A}_{\mathfrak{C}_2}$ only $O(n^{2.373})$ operations in the finite field are required (this is essentially the cost of matrix inversion), we have that the BOS approach in this case gives a complexity of

$$O\left(n^{2.373} C_{WGI}(n)\right),$$

where $C_{WGI}(n)$ denotes the complexity of a solver for the weighted graph isomorphism problem. Note that the problem can be solved, for many classes of graphs, with very efficient algorithms. Furthermore, Babai's recent breakthrough paper [5] shows that the problem can be solved, in the worst case, with quasi-polynomial complexity. Hence, even in the worst case scenario, the BOS approach on codes with trivial hull runs in time that is quasi-polynomial in the code length.

For the more general case of codes with a non-trivial hull, the reduction from WGI works in a different way. We do not enter into full details here, and simply report that the complexity of the approach, whose proof can be found in [7, Theorem 10], is asymptotically estimated as

$$O\left(h n^{2.373+h+1} C_{WGI}(n)\right).$$

Putting everything together, we estimate the complexity of the BOS approach as follows.

Proposition 8. *Let $\mathfrak{C}_1 \subseteq \mathbb{F}_q^n$ be a random code with dimension k. Let $\mathfrak{C}_2 = \pi(\mathfrak{C}_1)$, with π being a randomly picked permutation. Let $C_{WGI}(n)$ denote the time complexity of an algorithm solving the WGI problem on two graphs with n nodes. Then, the computational complexity of using the BOS approach to solve the permutation equivalence problem on $\mathfrak{C}_1$ and $\mathfrak{C}_2$ can be estimated as*

- $O\left(n^{2.373} C_{WGI}(n)\right)$, *if the codes have trivial hulls;*
- $O\left(n^{2.373+h+1} C_{WGI}(n)\right)$, *if the codes have hulls of dimension h.*

4.5 Solving the Linear Equivalence Problem

In this section we describe the state-of-the-art solvers for the linear equivalence problem. We anticipate that all known solvers inherit the approaches which we have already presented for the permutation equivalence case; yet, using monomial transformations instead of permutations makes, in certain cases, the hardness of the algorithms radically different.

Leon's algorithm Leon's algorithm can be used to solve the linear equivalence problem as well, with an operating procedure that is essentially identical to the one we have already discussed in Section 4.1. The only difference is in the fact that, after the codewords enumeration, one searches for a monomial matrix instead of a permutation. When the value of w is properly chosen, this can be reconstructed in polynoimal time, so that the bottleneck in the computational complexity is (again) in the codewords enumeration. Hence, we rely on Proposition 3 to estimate the cost of the algorithm.

Beullens' algorithm Beullens' algorithm can also be adapted to tackle the case of linear equivalences. However, in this case, codewords of low weight do not provide enough information about the searched transformation, since monomials also modify the multiset of entries. To overcome this issue, the author in [13] first observes that if $\mu \in \mathsf{M}_n$ is such that $\mu(\mathfrak{C}_1) = \mathfrak{C}_2$, then for any subcode $\mathfrak{V}_1 \subseteq \mathfrak{C}_1$ there must exist a subcode $\mathfrak{V}_2 \subseteq \mathfrak{C}_2$ such that $\mu(\mathfrak{V}_1) = \mathfrak{V}_2$. To exploit this fact, one can consider subcodes of small dimension and small support, i.e., such that a rather large portion of the coordinates is null. In this case, one can operate as in Algorithm 1, with only three exceptions:

1. the lists X and Y are populated with pairs $\{\boldsymbol{X}, \mathsf{Lex}(\boldsymbol{X})\}$ and $\{\boldsymbol{Y}, \mathsf{Lex}(\boldsymbol{Y})\}$, where $\boldsymbol{X}$, $\boldsymbol{Y}$ are subcodes of the given $\mathfrak{C}_1$ and $\mathfrak{C}_2$, and Lex returns the first basis of the input subspace, in lexicographical order. For an example of how such a basis can be computed, we refer the reader to Appendix D;
2. the list P contains the pairs $\{\boldsymbol{X}, \boldsymbol{Y}\}$ for which $\mathsf{Lex}(\boldsymbol{X}) = \mathsf{Lex}(\boldsymbol{Y})$;
3. in the reconstruction phase, one first finds the permutation, and then recovers the scaling factors. To recover the permutation, one proceeds in a way that is analogous to that for the permutation equivalence case; for the sake of completeness, we have reported the procedure in Algorithm 3. To recover the scaling factors, many efficient choices are possible. For instance, we may proceed as follows: we first apply the found permutation π to each low weight codeword of $\mathfrak{C}_1$, and then search among the already found low weight codewords of $\mathfrak{C}_2$, to find a pair having the same support. With overwhelming probability, a pair of codewords with the same support corresponds to a codeword of $\pi(\mathfrak{C}_1)$, where each non null entry has been scaled as an effect of the scaling factors in the monomial. Hence, each pair of such codewords exactly gives us w coefficients out of n. Then, finding a bunch of such pairs will be enough to completely retrieve the scaling vectors in $\boldsymbol{v}$. After this, we have obtained the desired monomial as $\pi \rtimes \boldsymbol{v}$.

Neglecting the cost of Algorithm 3, we have that the complexity of Beullens' algorithm to solve the linear equivalence is identical to that in Proposition 6. The only difference is in the fact that one may use different values for w and L, and that the success probability changes.

Input: list P, containing couples $\{\boldsymbol{X}, \boldsymbol{Y}\} \in F_q^{2\times n} \times \mathbb{F}_q^{2\times n}$, codes $\mathfrak{C}_1, \mathfrak{C}_2$
Output: permutation π, or report failure

$\boldsymbol{U} \leftarrow n \times n$ matrix made of all ones;
for $\{\mathfrak{X}, \boldsymbol{y}\} \in P$ **do**
  for $i \in \{1, \cdots, n\}$ **do**
    $\boldsymbol{x}_i \leftarrow i$-th column of $\boldsymbol{X}$;
    for $j \in \{1, \cdots, n\}$ **do**
      $\boldsymbol{y}_j \leftarrow j$-th column of $\boldsymbol{Y}$;
      /* If only one of the columns is null, remove j from the possible images of i */
      if $(\boldsymbol{x}_i == \boldsymbol{0}) \neq (\boldsymbol{y}_j == \boldsymbol{0})$ **then**
        $u_{i,j} = 0$;

/* Use $\boldsymbol{U}$ to reconstruct the permutation; if not possible, report failure */
if $\boldsymbol{U}$ is a permutation matrix **then**
  $\pi \leftarrow$ permutation described by $\boldsymbol{U}$;
  return π;
else
  report failure;

Algorithm 3: Fast permutation recovery, for the linear equivalence version of Beullen's algorithm.

Proposition 9. *Algorithm 3 with parameters L and N_w succeds with probability*

$$\gamma(L, N_w) = \left(1 - \left(1 - 2\left(1 - \frac{(n-w)^2}{n^2}\right)\frac{(n-w)^2}{n^2}\right)^{\frac{L^2(L^2-N_w)}{2N_w^2}}\right)^{n(n-1)}.$$

Proof. After having found L codewords with weight w in both codes $\mathfrak{C}_1$ and $\mathfrak{C}_2$, we expect on average to have $M = L^2/N_w$ pairs such that $\mu(\boldsymbol{x}) = \boldsymbol{y}$, with $\boldsymbol{x} \in \mathfrak{C}_1$ and $\boldsymbol{y} \in \mathfrak{C}_2$. Let us consider a pair $\{\boldsymbol{X}, \boldsymbol{Y}\} \in P$; furthermore, let $\boldsymbol{x}_j$ denote the j-th column of $\boldsymbol{X}$ and $\boldsymbol{y}_\ell$ the ℓ-th column of $\boldsymbol{Y}$. We assume $\pi(j) \neq \ell$ if and only if $\boldsymbol{x}_j \neq 0$ and $\boldsymbol{y}_\ell = 0$, or $\boldsymbol{x}_j = 0$ and $\boldsymbol{y}_\ell \neq 0$. Since each row of these matrices contain w non-zero entries, we have that the assumption is correct with probability

$$\rho = 2\left(1 - \frac{(n-w)^2}{n^2}\right)\frac{(n-w)^2}{n^2}.$$

Given that we can form $\binom{M}{2} = \frac{M(M-1)}{2}$ subcodes, the probability that at the end of the algorithm we have concluded that $\pi(j) \neq \ell$ is given by $1 - (1-\rho)^{\binom{M}{2}}$. Since for each index $j \in [1; n]$ we have $n-1$ guesses to make, we finally estimate the success probability as

$$\gamma(L, N_w) = \left(1 - (1-\rho)^{\binom{M}{2}}\right)^{n(n-1)} = \left(1 - (1-\rho)^{\frac{L^2(L^2-N_w)}{2N_w^2}}\right)^{n(n-1)} \square$$

Support Splitting Finally, SSA can be used to solve the linear equivalence problem as well, as the problem can be reduced to that of finding a permutation equivalence between the closures of the codes in question. Here, by *closure* of a linear code $\mathfrak{C}$, we mean the code $\tilde{\mathfrak{C}}$ defined as $\{\boldsymbol{c} \otimes \boldsymbol{a}, \ \ \boldsymbol{c} \in \mathfrak{C}\}$, where $\boldsymbol{a} = (a_1, \cdots, a_{q-1})$ is any ordering of the non-zero elements of $\mathbb{F}_q$. A fundamental point, though, is that, for $q \geq 5$, the closure of a code is always weakly-self dual, and thus has a hull of maximum dimension k, leading to exactly the hardest instances for SSA to solve. These results are corroborated by the analysis in [29]. In the end, the complexity of the SSA algorithm, to solve the linear equivalence problem, can be estimated through Proposition 7, assuming that the hull has maximum dimension $h = k$.

4.6 A Complete Picture

For the sake of clarity, in Table 2 we briefly resume the features of all the algorithms we have presented in this section. It is possible to note that the complexity of the algorithms to solve the code equivalence problem essentially does not change with the equivalence type. In fact, apart from some small choices in setting the attack options (for instance, the value of w and L in Beullens' algorithm), we have that the complexities for the linear case are identical to those of the permutation case. Then, one may be led into thinking that the two problems are equally hard; yet, this claim is fundamentally false.

To be clear, the permutation equivalence problem is actually easy in many cases. Using either SSA or BOS, one can comfortably solve random instances since, with very high probability, they have a hull with very small dimension. Yet, these algorithms fail in successfully solving the problem when the hull dimension increases, which is the case, for example, of (weakly) self-dual codes. In such cases, the problem becomes harder and one must rely on algorithms such as those of Leon and Beullens, whose running time is exponential. So we can conclude that the permutation equivalence is hard only for some very specific instances.

Type of equivalence	**Algorithm**	**Complexity**	**Notes**
Permutation	Leon	$O\big(C_{ISD}(q,n,k,d_{GV})\cdot 2\ln(N_w)\big)$	Preferable with small finite fields and large hulls.
	Beullens	$O\left(\frac{2L\cdot C_{ISD}(q,n,k,w)}{N_w\left(1-2^{L\log_2(1-L/N_w)}\right)}\right)$	Preferable with large finite fields and large hulls. It may fail, when L is too small.
	SSA	$O\left(n^3+n^2q^h\log n\right)$	Efficient with small, non-trivial hulls
	BOS	$\begin{cases} O\left(n^{2.373}C_{WGI}(n)\right) & \text{if } h=0 \\ O\left(n^{2.373+h+1}C_{WGI}(n)\right) & \text{if } h>0 \end{cases}$	Efficient with trivial hulls
Linear	Leon	$O\big(C_{ISD}(q,n,k,d_{GV})\cdot 2\ln(N_w)\big)$	Preferable with small finite fields and large hulls.
	Beullens	$O\left(\frac{2L\cdot C_{ISD}(q,n,k,w)}{N_w\left(1-2^{L\log_2(1-L/N_w)}\right)}\right)$	Preferable with large finite fields and large hulls. It may fail, when L is too small.
	SSA	$\begin{cases} O\left(n^3+n^2q^h\log n\right) & \text{if } q<5 \\ O\left(n^3+n^2q^k\log n\right) & \text{if } q\geq 5 \end{cases}$	Efficient if $q<5$ and the hull is trivial.

Table 2: Summary of techniques to solve the code equivalence problem

The situation is quite the opposite, when dealing with the linear equivalence problem. Indeed, in this case, SSA is efficient only for random codes defined over finite fields with $q \leq 5$, and becomes exponentially hard if the field size grows, regardless of the code properties. Hence, for large q, it is fair to say that no polynomial-time solver is currently known. This seems to suggest that, for large finite fields, the linear equivalence problem is a much harder problem, in general, compared to permutation equivalence.

5 Optimizations

In this section we discuss possible strategies to optimize the LESS signature scheme which, in its basic form, we have recalled in Table 1.

5.1 Multi-bit Challenges

A first natural observation is that signature size can be reduced by decreasing the number of rounds that are necessary to reach the desired preimage security level. This, obviously, requires the soundness error in the underlying ZK identification scheme to decrease proportionally. Such a scenario can be realized, for instance, by increasing the number of challenge bits in each round, as described in Seasign [17]. Each challenge bit becomes an ℓ-bit challenge string, which can be interpreted as an integer between 0 and $2^\ell - 1$, i.e. as an element of $\mathbb{Z}_{2^\ell}$, using the well-known correspondence $\mathbb{Z}_2^\ell = \mathbb{Z}_{2^\ell}$. Accordingly, the scheme is modified to feature $r = 2^\ell$ independent public keys; each challenge string is then used to select one of the keys, for which a response is produced (using the corresponding private key). To keep notation simple, we exploit the bijection mentioned above, and interchangeably use the same symbol to denote an ℓ-bit string (as part of a hash output) or an integer in $[0; 2^\ell - 1]$ (for example, when used as an index). A pictorial representation of this variant is given in Table 3, below, where we call the scheme LESS-M (for Multi-bit).

Setup	Input parameters $q, n, k, \ell, \lambda \in \mathbb{N}$, then set $r = 2^\ell$ and $t = \lceil \lambda/\ell \rceil$. Choose matrix $\boldsymbol{G} \in \mathbb{F}_q^{k \times n}$ and hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$.
Private Key	Monomial matrices $\boldsymbol{Q}_0 = \boldsymbol{I}_n \dots \boldsymbol{Q}_{r-1} \in \mathsf{M}_n$.
Public Key	Generator matrices $\boldsymbol{G}_0 \dots \boldsymbol{G}_{r-1}$, where $\boldsymbol{G}_i = \mathsf{sf}(\boldsymbol{G}\boldsymbol{Q}_i)$ for $i = 0 \dots r-1$.

SIGNER		VERIFIER
For $i = 0 \dots t-1$, choose $\tilde{\boldsymbol{Q}}_i \xleftarrow{\$} \mathsf{M}_n$		
and set $\tilde{\boldsymbol{G}}_i = \mathsf{sf}(\boldsymbol{G}\tilde{\boldsymbol{Q}}_i)$.		
Set $h = \mathsf{H}(\tilde{\boldsymbol{G}}_0, \dots, \tilde{\boldsymbol{G}}_{t-1}, \boldsymbol{m})$.		
Parse $h = h_0, \dots h_{t-1}$, for $h_i \in \mathbb{Z}_2^\ell$.		
For $i = 0 \dots t-1$, compute $\mu_i = \boldsymbol{Q}_{h_i}^{-1}\tilde{\boldsymbol{Q}}_i$.		
Set $\sigma = (\mu_0, \dots \mu_{t-1}, h)$.		
	$\xrightarrow{(\boldsymbol{m}, \sigma)}$	
		Parse $h = h_0, \dots h_{t-1}$, for $h_i \in \mathbb{Z}_2^\ell$.
		For $i = 0 \dots t-1$, compute $\hat{\boldsymbol{G}}_i = \mathsf{sf}(\boldsymbol{G}_{h_i}\mu_i)$.
		Accept if $\mathsf{H}(\hat{\boldsymbol{G}}_0, \dots, \hat{\boldsymbol{G}}_{t-1}, \boldsymbol{m}) = h$.

Table 3: The LESS-M Signature Scheme.

Note that this variant is more natural than what it may seem at a first glance. In fact, the original LESS signature scheme of Table 1 can be seen as a particular case of the LESS-M scheme, where $\ell = 1$ and $\boldsymbol{Q}_0 = \boldsymbol{I}_n$. The main difference is in the security notion underlying the scheme. The security assumption in this case becomes the following.

Problem 2 (Multiple Codes Linear Equivalence) *Consider a collection of linearly equivalent $[n, k]$-linear codes $\mathcal{C}_0 \dots \mathcal{C}_{r-1}$, admitting generator matrices $\boldsymbol{G}_0, \dots, \boldsymbol{G}_{r-1}$ of the form $\boldsymbol{S}_0\boldsymbol{G}\boldsymbol{Q}_0, \dots, \boldsymbol{S}_{r-1}\boldsymbol{G}\boldsymbol{Q}_{r-1}$. Find matrices $\boldsymbol{S}^* \in \mathsf{GL}_k$ and $\boldsymbol{Q}^* \in \mathsf{M}_n$ such that $\boldsymbol{G}_{j'} = \boldsymbol{S}^*\boldsymbol{G}_j\boldsymbol{Q}^*$, for some $j \neq j'$.*

This problem is still hard, and directly connected to the hardness of the linear code equivalence problem. A reduction is given below.

Theorem 3. *Given an algorithm to solve the Multiple Codes Linear Equivalence Problem (Problem 2), that runs in time T and succeeds with probability ε, it is possible to solve the Linear Equivalence Problem (Problem 1), in time approximately equal to $T + O(rn^3)$, with probability of success equal to $\varepsilon/2$.*

Proof. Let $\mathcal{A}$ be an adversary for Problem 2. We now show how to construct an adversary $\mathcal{A}'$ that is able to solve the linear code equivalence problem. $\mathcal{A}'$ will interact with $\mathcal{A}$ and use it as a subroutine. To begin, $\mathcal{A}'$ is given an instance $(\boldsymbol{G}, \boldsymbol{G}' = \boldsymbol{SGQ})$ of Problem 1. It will then proceed to generate $r = 2^\ell$ equivalent codes, in the following way. First, $\mathcal{A}'$ samples uniformly at random matrices $\boldsymbol{S}_0, \dots, \boldsymbol{S}_{r-1}$ and $\boldsymbol{Q}_0, \dots, \boldsymbol{Q}_{r-1}$. Then, it computes half of the codes starting from $\boldsymbol{G}$, and half starting from $\boldsymbol{G}'$; wlog, we can imagine that $\boldsymbol{G}_i$ is generated as $\boldsymbol{S}_i\boldsymbol{G}\boldsymbol{Q}_i$ when $i \in [0; r/2 - 1]$, and as $\boldsymbol{S}_i\boldsymbol{G}\boldsymbol{Q}_i$ when $i \in [r/2; r - 1]$ (and then reordered). It is clear that this computation can be done in polynomial time, at most $O(rn^3)$, and that there is no way to distinguish how an individual matrix was generated (i.e. from $\boldsymbol{G}$ rather than $\boldsymbol{G}'$). At this point $\mathcal{A}'$ runs $\mathcal{A}$ on input $\boldsymbol{G}_0, \dots, \boldsymbol{G}_{r-1}$, and $\mathcal{A}$ will output, with probability ε, a response $(\boldsymbol{S}^*, \boldsymbol{Q}^*)$ such that $\boldsymbol{G}_{j'} = \boldsymbol{S}^*\boldsymbol{G}_j\boldsymbol{Q}^*$. Now, if one of the two matrices was of the first type, and the other of the second type, $\mathcal{A}'$ is able to win. In fact, consider wlog that $\boldsymbol{G}_j = \boldsymbol{S}_j\boldsymbol{G}\boldsymbol{Q}_j$ and $\boldsymbol{G}_{j'} = \boldsymbol{S}_{j'}\boldsymbol{G}'\boldsymbol{Q}_{j'}$; then it must be that $\boldsymbol{Q}^* = \boldsymbol{Q}_j^{-1}\boldsymbol{Q}\boldsymbol{Q}_{j'}$, which immediately reveals[5] $\boldsymbol{Q}$. Since this happens with probability $1/2$, we get the thesis. □

5.2 Fixed-weight Challenges

In this variant, contrary to what we were doing before, we propose to increase the number of rounds, rather than decreasing them. This may seem counterintuitive, but in fact it works out to be very efficient, and it is based on just as natural an observation as the previous one. In this case, the key idea is that different responses, corresponding to different challenge bits, have a very unbalanced impact on the size of the signature. In particular, for the original LESS scheme described in Table 1, in the case $h_i = 0$ the response μ_i consists of the purely random monomial matrix $\tilde{\boldsymbol{Q}}_i$, and therefore the signer can transmit just the seed used to generate such random object. This, of course, is much more compact than the monomial matrix $\boldsymbol{Q}_1^{-1}\tilde{\boldsymbol{Q}}_i$ which needs to be transmitted, in full, when $h_i = 1$. It makes sense, therefore, to try and minimize the amount of bits h_i that are equal to 1, in the challenge string h output by H.

In our case, a natural way to implement this idea is to switch the output distribution of H, so that it returns a vector of fixed weight, rather than a uniformly distributed binary string. In other words, we need to pick H to be a *weight-restricted* hash function, whose range is the set $\mathbb{Z}_{2,\omega}^t$. The modified protocol is described in Table 4 below, where we call the scheme LESS-F (for Fixed-weight).

Setup	Input parameters $q, n, k, \lambda, t, \omega \in \mathbb{N}$. Choose matrix $\boldsymbol{G} \in \mathbb{F}_q^{k\times n}$ and w.r. hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_{2,\omega}^t$. Set $\boldsymbol{Q}_0 = \boldsymbol{I}_n$ and $\boldsymbol{G}_0 = \mathsf{sf}(\boldsymbol{G})$.
Private Key	Monomial matrix $\boldsymbol{Q}_1 \in \mathsf{M}_n$.
Public Key	Generator matrix $\boldsymbol{G}_1 = \mathsf{sf}(\boldsymbol{GQ}_1)$.

SIGNER		VERIFIER
For $i = 0 \dots t-1$, choose $\tilde{\boldsymbol{Q}}_i \xleftarrow{\$} \mathsf{M}_n$ and set $\tilde{\boldsymbol{G}}_i = \mathsf{sf}(\boldsymbol{G}\tilde{\boldsymbol{Q}}_i)$.		
Set $h = \mathsf{H}(\tilde{\boldsymbol{G}}_0, \dots, \tilde{\boldsymbol{G}}_{t-1}, \boldsymbol{m})$.		
Parse $h = h_0, \dots h_{t-1}$, for $h_i \in \{0,1\}$.		
For $i = 0 \dots t-1$, compute $\mu_i = \boldsymbol{Q}_{h_i}^{-1}\tilde{\boldsymbol{Q}}_i$.		
Set $\sigma = (\mu_0, \dots \mu_{t-1}, h)$.		
	$\xrightarrow{(\boldsymbol{m},\sigma)}$	
		Parse $h = h_0, \dots h_{t-1}$, for $h_i \in \{0,1\}$.
		For $i = 0 \dots t-1$, compute $\hat{\boldsymbol{G}}_i = \mathsf{sf}(\boldsymbol{G}_{h_i}\mu_i)$.
		Accept if $\mathsf{H}(\hat{\boldsymbol{G}}_0, \dots, \hat{\boldsymbol{G}}_{t-1}, \boldsymbol{m}) = h$.

Table 4: The LESS-F Signature Scheme.

[5] If needed, $\boldsymbol{S}$ can then be found in polynomial time also.

This idea is not entirely new in the context of identification and signature schemes. In fact, as reported in [28], the suggestion to use a fixed-weight challenge vector is already present in the original Fiat-Shamir work [21]. More recently, some signature schemes appeared that also make use of a similar approach, albeit in a different context. For instance, Picnic, which earned much praise during the NIST post-quantum standardization process [1], uses a pre-processing stage and a cut-and-choose procedure to achieve the desired security level. This technique was later revisited and generalized by Beullens [13], who presents an application to multivariate schemes, as well as PKP. In all cases, it is evident how picking the challenge vector from a carefully crafted distribution beats the simple parallel repetition of the protocol.

A crucial aspect is, of course, that it is necessary to avoid losing security during the process. This implies that the final preimage security level of the protocol needs to remain equal to the original goal of $2^{-\lambda}$, which was naturally obtained via parallel repetition. In our case, simply constraining the challenge vector to a target Hamming weight is guaranteed to lose security bits. Indeed, this happens even in the most basic scenario, i.e. if we restrict to the expected value $\omega = t/2$; for instance, when $\lambda = 128$, sampling h among the vectors of weight 64 only leads to approximately 124 preimage security bits.

To understand this phenomenon, recall that *preimage security* corresponds, essentially, to the difficulty of guessing the entire challenge vector. In the case of parallel repetition, since each instance is independent from the others, this is equivalent to correctly picking the challenge in each round, which leads to a probability of ε^t, where ε is the soundness error (in our case $1/2$). However, if the challenge is sampled among vectors of fixed Hamming weight, then the difficulty of guessing is the reciprocal of

$$\left|\mathbb{Z}^t_{2,\omega}\right| = \binom{t}{\omega}.$$

From this, it follows that, in order to safely switch to constrained-weight challenge vectors, it is necessary to ensure that $\log_2 \binom{t}{\omega} \geq \lambda$. This leads to an increase in the overall length of the challenge vector, thus increasing the number of rounds, as claimed in the beginning of this section. We will discuss the optimal choices of t and ω for this variant when selecting parameters, in Section 6.

5.3 Combining the Approaches

In this section we explain how to combine the approaches illustrated in the previous sections. The result is depicted in Table 5, below, where we call the scheme LESS-FM (as it is a combination of the two techniques).

This formulation is the most generic, as it includes the previous ones as particular cases. The quantity to consider for preimage security is

$$\left|\mathbb{Z}^t_{2^\ell,\omega}\right| = \binom{t}{\omega}(2^\ell - 1)^\omega. \tag{2}$$

As mentioned before, the parameter choice is discussed in the next section.

6 Concrete parameter sets and Implementation strategies

Selecting optimal parameters for LESS-FM involves a multi-target optimization where the considered figures of merit are: i) the desired security level, ii) the size of the keypair and of the transmitted message, and iii) the computational load required. In this work, we propose parameter sets which are targeted to a computational effort of 2^{128} classical gates, as is standard in literature. This will also facilitate a comparison with other existing signature schemes. Nevertheless, we also ensured that our parameters achieve at least 64 quantum security bits, according to the best known quantum algorithm techniques, an analysis of which is reported in Appendix F.

Table 5: The LESS-FM Signature Scheme.

Setup	Input parameters $q, n, k, \ell, \lambda, t, \omega \in \mathbb{N}$, then set $r = 2^\ell$. Choose matrix $\boldsymbol{G} \in \mathbb{F}_q^{k \times n}$ and w.r. hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_{2^\ell,\omega}^t$. Set $\boldsymbol{Q}_0 = \boldsymbol{I}_n$ and $\boldsymbol{G}_0 = \mathsf{sf}(\boldsymbol{G})$.
Private Key	Monomial matrices $\boldsymbol{Q}_1 \dots \boldsymbol{Q}_{r-1} \in \mathsf{M}_n$.
Public Key	Generator matrices $\boldsymbol{G}_1 \dots \boldsymbol{G}_{r-1}$, where $\boldsymbol{G}_i = \mathsf{sf}(\boldsymbol{G}\boldsymbol{Q}_i)$ for $i = 1 \dots r-1$.

SIGNER		VERIFIER
For $i = 0 \dots t-1$, choose $\tilde{\boldsymbol{Q}}_i \xleftarrow{\$} \mathsf{M}_n$ and set $\tilde{\boldsymbol{G}}_i = \mathsf{sf}(\boldsymbol{G}\tilde{\boldsymbol{Q}}_i)$.		
Set $h = \mathsf{H}(\tilde{\boldsymbol{G}}_0, \dots, \tilde{\boldsymbol{G}}_{t-1}, \boldsymbol{m})$.		
Parse $h = h_0, \dots h_{t-1}$, for $h_i \in \{0,1\}^\ell$.		
For $i = 0 \dots t-1$, compute $\mu_i = \boldsymbol{Q}_{h_i}^{-1}\tilde{\boldsymbol{Q}}_i$.		
Set $\sigma = (\mu_0, \dots \mu_{t-1}, h)$.		
	$\xrightarrow{(\boldsymbol{m},\sigma)}$	
		Parse $h = h_0, \dots h_{t-1}$, for $h_i \in \{0,1\}^\ell$.
		For $i = 0 \dots t-1$, compute $\hat{\boldsymbol{G}}_i = \mathsf{sf}(\boldsymbol{G}_{h_i}\mu_i)$.
		Accept if $\mathsf{H}(\hat{\boldsymbol{G}}_0, \dots, \hat{\boldsymbol{G}}_{t-1}, \boldsymbol{m}) = h$.

We now recall the main features of each version of the LESS signature scheme. Let $N = \lceil \log_2(n) \rceil$ and $Q = \lceil \log_2(q) \rceil$.

Version	Type	Num. of Rounds	$\lvert\mathsf{pk}\rvert$	$\lvert\sigma\rvert$
-	Perm	λ	$k(n-k)Q$	$\lambda\left(1 + \frac{\lambda+nN}{2}\right)$
	Mono	λ	$k(n-k)Q$	$\lambda\left(1 + \frac{\lambda+nN+nQ}{2}\right)$
M	Perm	$\lceil \frac{\lambda}{\ell} \rceil$	$(2^\ell - 1)k(n-k)Q$	$\lceil \frac{\lambda}{r} \rceil \left(2\ell + \frac{nN}{2}\right)$
	Mono	$\lceil \frac{\lambda}{\ell} \rceil$	$(2^\ell - 1)k(n-k)Q$	$\lceil \frac{\lambda}{r} \rceil \left(2r + \frac{nN+nQ}{2}\right)$
F	Perm	t s.t. $\binom{t}{\omega} > 2^\lambda$	$k(n-k)Q$	$t + (t-\omega)\lambda + \omega nN$
	Mono	t s.t. $\binom{t}{\omega} > 2^\lambda$	$k(n-k)Q$	$t + (t-\omega)\lambda + \omega n\,(N+Q)$
FM	Perm	t s.t. $\binom{t}{\omega}(2^\ell - 1)^\omega > 2^\lambda$	$(2^\ell - 1)k(n-k)Q$	$\ell t + (t-\omega)\lambda + \omega nN$
	Mono	t s.t. $\binom{t}{\omega}(2^\ell - 1)^\omega > 2^\lambda$	$(2^\ell - 1)k(n-k)Q$	$\ell t + (t-\omega)\lambda + \omega n\,(N+Q)$

Table 6: Overview of the number of rounds and public key/signature sizes in bits as a function of the LESS variant parameters.

Table 6 gives a synthetic view of public key and signature sizes as a function of the LESS variant parameters.

Efficient key storage and manipulation Storing LESS-FM keypairs in an efficient-to-employ format can be achieved representing the field elements of $\mathbb{F}_q$ as $Q = \lceil \log_2(q) \rceil$ integers, and linearizing the non-trivial portion of the $\mathbf{G}_i$ matrices of the public keys performing bitpacking. This yields a public key size of $(2^\ell - 1)k(n-k)Q$ (considering one can always set $\boldsymbol{Q}_0 = \boldsymbol{I}_n$ and $\boldsymbol{G}_0 = \mathsf{sf}(\boldsymbol{G})$). Concerning the storage representation of the monomial matrices composing the private key, they can be materialized as an n element vector of pairs which stores the index of the permuted element, and the value of the multiplicative coefficient on $\mathbb{F}_q$ of the said element. We note that, while a compact representation of the permutation alone is possible over $\lceil \log_2(n!) \rceil$ bits, this saves a relatively small amount of space (about 4.5% in our "minimize pk+sig.size" parameter set), at the cost of performing the relatively demanding computation of permutation unranking, to bring the permutation in a usable representation.

Optimization Criterion	LESS	Type	n	k	q	ℓ	t	ω	**pk (kB)**	**sig (kB)**	**pk + sig (kB)**
Min. pk size	F	Mono	198	94	251	1	283	28	9.77	15.2	24.97
Min. sig size	FM	Perm	235	108	251	4	66	19	205.74	5.25	210.99
Min. pk + sig size	F	Perm	230	115	127	1	233	31	11.57	10.39	21.96
Beullens [13]	-	Mono	250	125	53	1	128	-	11	28	39

Table 7: Parameter sets for LESS-FM, for a security level of $\lambda = 128$ classical bits.

Table 7 reports the result of the optimization of the LESS-FM parameters when targeting the minimization of i) the public key size, ii) the signature size or iii) the sum of the aforementioned quantities. The rationale behind these criteria is to highlight the flexibility of LESS-FM in application scenarios where i) the space for public key storage is constrained (e.g. microcontrollers with tight Flash memory limits), ii) digital certificates, which employ a concatenation of public keys and signatures, and iii) application scenarios where a large amount of signed messages are exchanged between two endpoints employing long-term keypairs. Despite our conservative quantification of the computational effort required by the most effective cryptanalytic approaches, LESS can be instantiated with parameters pushing the size of the public key below 10 kB, or keep the sum of the public key and signature below 22kB.

Scheme	**Security Level**	**pk (kB)**	**sig (kB)**	**pk + sig (kB)**	**Security Assumption**
Stern [19]	80	18.48	113.5	131.98	Low-weight Hamming
Veron [19]	80	18.52	109.05	127.57	Low-weight Hamming
CVE [19]	80	5.31	66.44	71.75	Low-weight Hamming
Wave [18]	128	3205	1.04	3206.04	High-weight Hamming
cRVDC [11]	125	0.15	22.48	22.63	Low-weight Rank
Durandal - I [4]	128	15.24	4.06	19.3	Low-weight Rank
Durandal - II [4]	128	18.60	5.01	23.61	Low-weight Rank
LESS-F min. pk size	128	9.77	15.2	24.97	Linear Equivalence
LESS-FM min. sig size	128	205.74	5.25	210.99	Perm Equivalence
LESS-F min. pk + sig size	128	11.57	10.39	21.96	Perm Equivalence

Table 8: A comparison of public keys and signature sizes with alternative code-based signature schemes

Our balanced optimization criterion, combined with the use of fixed-weight challenges allows us to reduce to less than half the signature size, with respect to the parameter sets proposed by Beullens in [13], while retaining the same public key size. Finally, we are able, at the cost of a larger public key size, to achieve a minimum signature size of 5.25 kB, reducing the signature size by close to $3\times$ with respect to the other parameter sets.

Table 8 reports a comparison of the data sizes achieved by LESS variants with other code-based signature schemes. Considering the algorithms employing Fiat-Shamir, we see that LESS consistently outperforms the traditional schemes based on the Hamming metric (such as Stern, Veron and CVE), even when compared to 80-bit security versions, while its characteristics are orthogonal to those of cRVDC (which is rank-based). Indeed, despite a much larger public key, LESS achieves more compact signatures, and therefore compares favourably in scenarios ii) and iii). Finally, we consider two recent signature algorithms (namely, Wave and Durandal). We observe that LESS also provides favourable figures with respect to Wave, as it features a public key which is smaller by two orders of magnitude, albeit at the cost of an increase of an order of

magnitude in signature size. This provides a practical advantage in scenarios where a public key/signature pair is transferred at each communication, such as in the TLS authentication phase, which transmits X.509 certificates. In this regard, the performance of LESS is very similar to that of Durandal, which is an adaptation of the Schnorr's paradigm to the rank metric. It is then worth noting that the closest competition for LESS, in terms of performance, is represented by rank metric schemes, an area which is somewhat further away from traditional coding theory (while strongly related to multivariate cryptography), and, in the case of Durandal, relying on younger, ad-hoc computational assumptions.

Efficiency of LESS-FM and constant-time implementation techniques. The operations involved in LESS act, at low level, on elements of $\mathbb{F}_q$ for relatively small values of q. This in turn allows to perform efficient modular reduction through Barrett's reduction techniques, as the triple-precision multiplier required only needs to be able to deal with a 10- and a 20-bit operand. This, in turn, fits easily into a 32-bit multiplier with a single precision output, which is available also on embedded platforms (e.g. ARM Cortex-M3 and ARM Cortex-M4).

Multiplications by monomial matrices can be implemented as simple column permutations of the corresponding generator matrix, combined with a scalar-by-vector multiplication over $\mathbb{F}_q$. Such operations allow for a significant amount of inner parallelism in the latter operation, which can be leveraged for consistent speedups if vector ISA extensions are available on the computing platform. Care should be taken in performing the column-wise permutation, as its value is part of the private key. To this end, the use of permutation network-based strategies (e.g. Beneš networks or Omega-Flip networks) provides an effective and efficient solution.

Finally, concerning the computation of the systematic form of the $\tilde{\mathbf{G}}_i$ matrices, we note that a random selection of a k-column minor may not yield a non-singular $k \times k$ matrix. However, a non-singular matrix will be selected with probability $\prod_{i=1}^{k}(1-\frac{1}{q^i})$, which, given the choice of the values of q and k, provides a $> 95\%$ probability of selecting a non-singular k column minor. Therefore, the conservative strategy of drawing a new value for the ephemeral random monomial matrix $\tilde{\mathbf{Q}}_i$ and recomputing the systematic form of $\mathbf{G}\tilde{\mathbf{Q}}_i$ will trade off a minimal performance penalty for an easily auditable constant-time systematic form computation.

Acknowledgments

The authors would like to thank Ward Beullens and Robert Ransom for their insightful comments, that greatly contributed to the development of this work.

References

[1] `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`. 2017.

[2] N. Alamati et al. "Cryptographic Group Actions and Applications". In: *ASIACRYPT*. Springer. 2020, pp. 411–439.

[3] M. R. Albrecht et al. "Classic McEliece: conservative code-based cryptography". In: (). URL: `https://classic.mceliece.org/`.

[4] N. Aragon et al. "Durandal: A Rank Metric Based Signature Scheme". In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Y. Ishai and V. Rijmen. Cham: Springer International Publishing, 2019, pp. 728–758.

[5] L. Babai. "Graph Isomorphism in Quasipolynomial Time". In: *CoRR* abs/1512.03547 (2015). arXiv: `1512.03547`. URL: `http://arxiv.org/abs/1512.03547`.

[6] M. Baldi et al. "A Finite Regime Analysis of Information Set Decoding Algorithms". In: *Algorithms* 12.10 (2019). ISSN: 1999-4893. URL: `https://www.mdpi.com/1999-4893/12/10/209`.

[7] M. Bardet, A. Otmani, and M. Saeed-Taha. "Permutation Code Equivalence is Not Harder Than Graph Isomorphism When Hulls Are Trivial". In: *IEEE ISIT 2019*. July 2019, pp. 2464–2468.

[8] R. Beals et al. "Quantum Lower Bounds by Polynomials". In: *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*. IEEE Computer Society, 1998, pp. 352–361. DOI: `10.1109/SFCS.1998.743485`. URL: `https://doi.org/10.1109/SFCS.1998.743485`.

[9] A. Becker et al. "Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding". In: *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*. Ed. by D. Pointcheval and T. Johansson. Vol. 7237. Lecture Notes in Computer Science. Springer, 2012, pp. 520–536. DOI: 10.1007/978-3-642-29011-4_31. URL: https://doi.org/10.1007/978-3-642-29011-4%5C_31.

[10] M. Bellare and G. Neven. "Multi-signatures in the plain public-key model and a general forking lemma". In: *CCS*. 2006, pp. 390–399.

[11] E. Bellini et al. "Improved Veron Identification and Signature Schemes in the Rank Metric". In: *ISIT*. Paris, France, 2019, pp. 1872–1876.

[12] D. J. Bernstein et al. "Quantum Algorithms for the Subset-Sum Problem". In: *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*. Ed. by P. Gaborit. Vol. 7932. Lecture Notes in Computer Science. Springer, 2013, pp. 16–33. DOI: 10.1007/978-3-642-38616-9_2. URL: https://doi.org/10.1007/978-3-642-38616-9%5C_2.

[13] W. Beullens. *Not enough LESS: An improved algorithm for solving Code Equivalence Problems over F_q*. Cryptology ePrint Archive, Report 2020/801.

[14] J.-F. Biasse et al. "LESS is More: Code-Based Signatures Without Syndromes". In: *AFRICACRYPT*. Ed. by A. Nitaj and A. Youssef. Springer, 2020, pp. 45–65.

[15] G. Brassard, P. Høyer, and A. Tapp. "Quantum Counting". In: *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*. Ed. by K. G. Larsen, S. Skyum, and G. Winskel. Vol. 1443. Lecture Notes in Computer Science. Springer, 1998, pp. 820–831. DOI: 10.1007/BFb0055105. URL: https://doi.org/10.1007/BFb0055105.

[16] J. M. Couveignes. "Hard Homogeneous Spaces." In: *IACR Cryptol. ePrint Arch.* 2006 (2006), p. 291.

[17] L. De Feo and S. D. Galbraith. "SeaSign: Compact isogeny signatures from class group actions". In: *EUROCRYPT*. Springer. 2019, pp. 759–789.

[18] T. Debris-Alazard, N. Sendrier, and J.-P. Tillich. "Wave: A new family of trapdoor one-way preimage sampleable functions based on codes". In: *ASIACRYPT*. Springer. 2019, pp. 21–51.

[19] S. M. El Yousfi Alaoui et al. "Code-Based Identification and Signature Schemes in Software". In: *Security Engineering and Intelligence Informatics*. Ed. by A. Cuzzocrea et al. Springer Berlin Heidelberg, 2013, pp. 122–136.

[20] T. 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.

[21] A. Fiat and A. Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *CRYPTO*. Springer. 1986, pp. 186–194.

[22] S. Goldwasser, S. Micali, and R. L. Rivest. "A digital signature scheme secure against adaptive chosen-message attacks". In: *SIAM Journal on computing* 17.2 (1988), pp. 281–308.

[23] G. Kachigar and J. Tillich. "Quantum Information Set Decoding Algorithms". In: *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*. Ed. by T. Lange and T. Takagi. Vol. 10346. Lecture Notes in Computer Science. Springer, 2017, pp. 69–89. DOI: 10.1007/978-3-319-59879-6_5. URL: https://doi.org/10.1007/978-3-319-59879-6%5C_5.

[24] J. Leon. "Computing automorphism groups of error-correcting codes". In: *IEEE Transactions on Information Theory* 28.3 (May 1982), pp. 496–511.

[25] A. May, A. Meurer, and E. Thomae. "Decoding Random Linear Codes in $2^{0.054n}$". In: *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*. Ed. by D. H. Lee and X. Wang. Vol. 7073. Lecture Notes in Computer Science. Springer, 2011, pp. 107–124. DOI: 10.1007/978-3-642-25385-0_6. URL: https://doi.org/10.1007/978-3-642-25385-0%5C_6.

[26] C. Peters. "Information-set decoding for linear codes over $\mathbb{F}_q$". In: *International Workshop on Post-Quantum Cryptography*. Springer. 2010, pp. 81–94.

[27] E. Petrank and R. M. Roth. "Is code equivalence easy to decide?" In: *IEEE Transactions on Information Theory* 43.5 (Sept. 1997), pp. 1602–1604.

[28] R. Ransom. *Constant-time verification for cut-and-choose-based signatures*. Cryptology ePrint Archive, Report 2020/1184. 2020.

[29] M. A. Saeed. In: *PhD thesis* (2017).

[30] N. Sendrier. "The Support Splitting Algorithm". In: *Information Theory, IEEE Transactions on* (Aug. 2000), pp. 1193–1203.

[31] N. Sendrier and P. Symbolique. "On the Dimension of the Hull". In: *SIAM Journal on Discrete Mathematics* 10 (Nov. 1995). DOI: 10.1137/S0895480195294027.
[32] P. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509.
[33] A. Stolbunov. "Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves". In: *Advances in Mathematics of Communications* 4.2 (2010), p. 215.
[34] S. Tani. "An Improved Claw Finding Algorithm Using Quantum Walk". In: *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*. Ed. by L. Kucera and A. Kucera. Vol. 4708. Lecture Notes in Computer Science. Springer, 2007, pp. 536–547. DOI: 10.1007/978-3-540-74456-6_48. URL: https://doi.org/10.1007/978-3-540-74456-6%5C_48.

A Standard Definitions for Signature Schemes

Definition 3. *A Digital Signature Scheme, or simply Signature Scheme (SS), is a 6-tuple* $(\mathsf{K}, \mathsf{M}, \mathsf{S}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ *defined as follows.*

- $\mathsf{K} = \mathsf{K}_{\mathsf{sign}} \times \mathsf{K}_{\mathsf{ver}}$ *is the key space, containing pairs of private/public keys* $(\mathsf{sgk}, \mathsf{vk})$*, respectively the* signing key *and the* verification key*.*
- M *and* S *are, respectively, the* message space *and the* signature space*.*
- KeyGen *is a probabilistic* key-generation algorithm *that takes as input a security parameter* λ *and outputs a keypair* $(\mathsf{sgk}, \mathsf{vk}) \in \mathsf{K}$*.*
- Sign *is a (possibly probabilistic) private* signing algorithm *that receives as input a signing key* $\mathsf{sgk} \in \mathsf{K}_{\mathsf{sign}}$ *and a message* $\mu \in \mathsf{M}$ *and returns a signature* $\sigma \in \mathsf{S}$*.*
- Ver *is the (deterministic) public* verification algorithm *that receives as input a verification key* $\mathsf{vk} \in \mathsf{K}_{ver}$*, a message* $\mu \in \mathsf{M}$ *and a signature* $\sigma \in \mathsf{S}$ *and outputs 1, if the signature is recognized as valid, or 0 otherwise.*

Intuitively, a signature scheme is secure if a forger has only a negligible probability of producing a valid signature without knowing the private key. Among the several models described in literature, the one which is usually considered the most desirable, is the chosen-message attack model, in which an attacker is allowed access to an arbitrary number of message/signature pairs of his choosing, via so-called *signing queries*. The resulting security notion is known as *Existential Unforgeability under Chosen-Message Attacks (EUF-CMA)* [22], and can be formalized as follows.

Definition 4. *An adversary* $\mathcal{A}$ *for SS in the EUF-CMA attack model is a polynomial-time algorithm playing the following attack game:*

1. *Query a key generation oracle to obtain a verification key* vk*. The corresponding signing key* sgk *is kept private and is unknown to* $\mathcal{A}$*.*
2. *Perform a polynomial number of signing queries. In each signing query,* $\mathcal{A}$ *chooses a message* m *and submits it to a signing oracle. The oracle replies with* $\sigma = \mathsf{Sign}_{\mathsf{sgk}}(m)$*.*
3. *Output a pair* (m^*, σ^*)*.*

The adversary succeeds if $\mathsf{Ver}_{\mathsf{vk}}(m^*, \sigma^*) = 1$ *and* m^* *had not been queried before. We say that a signature scheme is* EUF-CMA secure *if the probability of success of any adversary* $\mathcal{A}$ *is negligible in the security parameter.*

B The automorphism group of a random code

In this appendix we derive an estimate on the size of the automorphism group of a random linear code. We first consider the case of permutations, and then extend our results to the case of monomials. We anticipate the main results we derive, and then proceed by showing how they can be proven.

Proposition 10. *Let $\mathfrak{C} \subseteq \mathbb{F}_q^n$ be a random linear code with dimension k. Let d_{GV} denote the GV distance of $\mathfrak{C}$, and $N_{d_{GV}} = \left\lceil \binom{n}{d_{GV}}(q-1)^{d_{GV}-2}(q^{k-n+1}) \right\rceil$. Let $d_{GV}^{\perp}$ be the GV distance of the dual code, and $N_{d_{GV}}^{\perp} = \left\lceil \binom{n}{d_{GV}^{\perp}}(q-1)^{d_{GV}^{\perp}-2}(q^{-k+1}) \right\rceil$. The probability that $\pi \xleftarrow{\$} \mathsf{S}_n$ is in the permutations automorphism group of $\mathfrak{C}$, i.e., $\pi(\mathfrak{C}) = \mathfrak{C}$, is not greater than*

$$\begin{aligned}\epsilon_{\mathsf{S}_n}(n,k,q) &= \frac{(q-1)\min\left\{N_{d_{GV}}! d_{GV}! (n-d_{GV})!\ ,\ N_{d_{GV}}^{\perp}! d_{GV}^{\perp}! (n-d_{GV}^{\perp})!\right\}}{n!} \\ &= (q-1)\min\left\{\frac{N_{d_{GV}}!}{\binom{n}{d_{GV}!}}\ ,\ \frac{N_{d_{GV}}^{\perp}!}{\binom{n}{d_{GV}^{\perp}!}}\right\},\end{aligned}$$

while the probability that $\tau \xleftarrow{\$} \mathsf{M}_n$ is in the monomials automorphism group of $\mathfrak{C}$, i.e., $\tau(\mathfrak{C}) = \mathfrak{C}$, is not greater than $\epsilon_{\mathsf{M}_n}(n,k,q) =$

$$\begin{aligned}&= \frac{\min\left\{N_{d_{GV}}! d_{GV}! (n-d_{GV})! (q-1)^{n-d_{GV}+1}\ ,\ N_{d_{GV}}^{\perp}! d_{GV}^{\perp}! (n-d_{GV}^{\perp})! (q-1)^{n-d_{GV}^{\perp}+1}\right\}}{n!(q-1)^n} \\ &= \min\left\{\frac{N_{d_{GV}}!(q-1)^{-d_{GV}+1}}{\binom{n}{d_{GV}!}}\ ,\ \frac{N_{d_{GV}}^{\perp}!(q-1)^{-d_{GV}^{\perp}+1}}{\binom{n}{d_{GV}^{\perp}!}}\right\}.\end{aligned}$$

B.1 Proof for the permutations automorphism group

To derive a bound on the size of the automorphism group of a code, we will consider the action of permutations on the set of minimum weight codewords. To this end, we first derive some preliminary results.

Proposition 11. *Let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_q^n$ with the same Hamming weight d and same entries multisets. Let $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{a}, \boldsymbol{b}) = \{\pi \in \mathsf{S}_n \mid \pi(\boldsymbol{a}) = \boldsymbol{b}\}$. Then, the cardinality of $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{a}, \boldsymbol{b})$ is not greater than $w!(n-w)!$.*

Proof. Let $E = \{i \in [0; n-1] \mid a_i = 0\}$. For a permutation π, we can have $\pi(i) = j$ if and only if $a_i = b_j$. Let m_x, for $x \in \mathbb{F}_q$, be the number of entries with value equal to x in both $\boldsymbol{a}$ and $\boldsymbol{b}$; since $\boldsymbol{a}$ and $\boldsymbol{b}$ have Hamming weight w, it holds that $m_0 = n - w$ and $\sum_{x \in \mathbb{F}_q^*} m_x = w$. Then, we have

$$|\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{a}, \boldsymbol{b})| = \prod_{x \in \mathbb{F}_q} m_x! = (n-w)! \prod_{x \in \mathbb{F}_q^*} m_x!.$$

It is immediately seen that $\prod_{x \in \mathbb{F}q^*} m_x! \leq \left(\sum_{x \in \mathbb{F}_q^*} m_x\right)! = w!$, so that as an upper bound on the size of $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{a}, \boldsymbol{b})$ we can use $(n-w!)w!$. □

Proposition 12. *Let $A \subseteq \mathbb{F}_q^n$, with cardinality M, such that all the contained vectors have Hamming weight w. Let $\mathsf{Aut}_{\mathsf{S}_n}(A) = \{\pi \in \mathsf{S}_n \mid \pi(\boldsymbol{a}) \in A,\ \ \forall \boldsymbol{a} \in A\}$; then, the size of $\mathsf{Aut}_{\mathsf{S}_n}(A)$ is not greater than $M!w!(n-w)!$.*

Proof. If $\pi \in \mathsf{Aut}_{\mathsf{S}_n}(A)$, then for each $\boldsymbol{a} \in A$, either $\pi(\boldsymbol{a}) = \boldsymbol{a}$ or there exists $\boldsymbol{a}' \in A$, $\boldsymbol{a}' \neq \boldsymbol{a}$, such that $\pi(\boldsymbol{a}) = \boldsymbol{a}'$. Let us define some order for the elements of A and write $A = \left\{\boldsymbol{a}^1, \boldsymbol{a}^2, \cdots, \boldsymbol{a}^M\right\}$. For each $\pi \in \mathsf{Aut}_{\mathsf{S}_n}(A)$, there exists one and only one bijection $f : \{1, \cdots, M\} \mapsto \{1, \cdots, M\}$ such that $f(i) = j$ if and only if $\pi(\boldsymbol{a}^i) = \boldsymbol{a}^j$. On the contrary, for a fixed bijection f, we may have more than one valid permutation, i.e., a permutation that places i in position j if and only if $f(i) = j$. It is easily seen that, for a bijection f, the set of all valid permutations is obtained as follows

$$\mathsf{Aut}_{\mathsf{S}_n}^{(f)}(A) = \bigcap_{i=1}^{M} \mathsf{Mor}_{\mathsf{S}_n}\left(\boldsymbol{a}^i, \boldsymbol{a}^{f(i)}\right).$$

Each bijection f can be seen as an element of the symmetric group on M elements (which we denote as S_M), so that the number of possible bijections is given by $M!$. Notice that, if $\pi \in \mathsf{Aut}_{\mathsf{S}_n}^{(f)}(A)$, then it is also in $\mathsf{Aut}_{\mathsf{S}_n}(A)$: hence, $\mathsf{Aut}_{\mathsf{S}_n}(A)$ corresponds to the union of all sets $\mathsf{Aut}_{\mathsf{S}_n}^{(f)}(A)$, that is

$$\mathsf{Aut}_{\mathsf{S}_n}(A) = \bigcup_{f \in \mathsf{S}_M} \mathsf{Aut}_{\mathsf{S}_n}^{(f)}(A) = \bigcup_{f \in \mathsf{S}_M} \left(\bigcap_{i=1}^{M} \mathsf{Mor}_{\mathsf{S}_n}\left(\boldsymbol{a}^i, \boldsymbol{a}^{f(i)}\right) \right).$$

We are now able to derive a simple upper bound on the size of $\mathsf{Aut}_{\mathsf{S}_n}(A)$, as follows

$$\begin{aligned} |\mathsf{Aut}_{\mathsf{S}_n}(A)| &= \left| \bigcup_{f \in \mathsf{S}_M} \left(\bigcap_{i=1}^{M} \mathsf{Mor}_{\mathsf{S}_n}\left(\boldsymbol{a}^i, \boldsymbol{a}^{f(i)}\right) \right) \right| \leq |\mathsf{S}_M| \cdot \left| \bigcap_{i=1}^{M} \mathsf{Mor}_{\mathsf{S}_n}\left(\boldsymbol{a}^i, \boldsymbol{a}^{f(i)}\right) \right| \\ &= M! \cdot \left| \bigcap_{i=1}^{M} \mathsf{Mor}_{\mathsf{S}_n}\left(\boldsymbol{a}^i, \boldsymbol{a}^{f(i)}\right) \right| \leq M! w! (n - w!), \end{aligned}$$

where the last inequality comes from Proposition 11. □

Using the previous results, we are finally able to prove the main result of this section.

Theorem 4. *Let $\mathfrak{C} \subseteq \mathbb{F}_q^n$ be a linear code with minimum distance d. Let $T_q(\boldsymbol{c}) = \left\{ b\boldsymbol{c} \mid b \in \mathbb{F}_q^* \right\}$, and let $V \subset \mathbb{F}_q$ be a set of codewords such that*

i) if $\boldsymbol{c} \in \mathfrak{C}$ has weight d, then $T_q(\boldsymbol{c})$ and V have only one element in common;
ii) all codewords in V have weight d.

Let $\mathsf{Aut}_{\mathsf{S}_n}(\mathfrak{C})$ be the permutations automorphism group of $\mathfrak{C}$. Then, the cardinality of $\mathsf{Aut}_{\mathsf{S}_n}(\mathfrak{C})$ is not greater than $(N_d)!(q-1)d!(n-d)!$, where N_d denotes the cardinality of V.

Proof. Without loss of generality, we can define V such that all of its codewords have the first entry that is equal to 1. Now, let $\pi \in \mathsf{Aut}_{\mathsf{S}_n}(\mathfrak{C})$; then, π must map the set of codewords of $\mathfrak{C}$ with weight d into itself. Since this set is obtained as $V_q = \bigcup_{\boldsymbol{c} \in V} T_q(\boldsymbol{c})$, we have that the image of V_q under the permutation π is equal to itself. Hence, for each $\boldsymbol{c} \in \mathfrak{C}$ with weight d, there must be $\boldsymbol{c}' \in V$ such that $\pi(\boldsymbol{c}) \in T_q(\boldsymbol{c}')$. Note that this also guarantees that, for each $\hat{\boldsymbol{c}} \in T_q(\boldsymbol{c})$, one also has $\pi(\hat{\boldsymbol{c}}) \in T_q(\boldsymbol{c}')$. To put it differently, for each $\boldsymbol{c} \in V$ there must exist i) another codeword $\boldsymbol{c}' \in V$, and ii) a non null element $b \in \mathbb{F}_q^*$, such that $\pi(\boldsymbol{c}) = b\boldsymbol{c}'$. Hence, we have

$$\mathsf{Aut}_{\mathsf{S}_n}(V_q) = \bigcup_{f \in \mathsf{S}_{N_d}} \bigcup_{b \in \mathbb{F}_q^*} \left(\bigcap_{i=1}^{N_d} \mathsf{Mor}_{\mathsf{S}_n}\left(\boldsymbol{c}^i, b\boldsymbol{c}^{f(i)}\right) \right).$$

This allows us to derive a bound on the size of $\mathsf{Aut}_{\mathsf{S}_n}(V_q)$, using the union bound for two times

$$\begin{aligned} |\mathsf{Aut}_{\mathsf{S}_n}(V)| &= \left| \bigcup_{f \in \mathsf{S}_{N_d}} \bigcup_{b \in \mathbb{F}_q^*} \left(\bigcap_{i=1}^{N_d} \mathsf{Mor}_{\mathsf{S}_n}\left(\boldsymbol{c}^i, b\boldsymbol{c}^{f(i)}\right) \right) \right| \\ &\leq |\mathsf{S}_{N_d}| \cdot \left|\mathbb{F}_q^*\right| \cdot \left| \bigcap_{i=1}^{N_d} \mathsf{Mor}_{\mathsf{S}_n}\left(\boldsymbol{c}^i, b\boldsymbol{c}^{f(i)}\right) \right| \\ &\leq N_d!(q-1)d!(n-d)!. \end{aligned}$$

Finally, we consider that if $\pi \in \mathsf{Aut}_{\mathsf{S}_n}(\mathfrak{C})$, then it must necessarily be $\pi \in \mathsf{Aut}_{\mathsf{S}_n}(V_q)$: hence, it must be $\mathsf{Aut}_{\mathsf{S}_n}(\mathfrak{C}) \subseteq \mathsf{Aut}_{\mathsf{S}_n}(V_q)$. So, we can use the bound on the cardinality of $\mathsf{Aut}_{\mathsf{S}_n}(V_q)$ as an upper bound for the size of $\mathsf{Aut}_{\mathsf{S}_n}(\mathfrak{C})$. □

The above results allow to prove the bound on the permutations automorphism group stated in Proposition 10. To estimate the minimum distance of a code, we use the well known Gilbert-Varshamov bound, and rely on Proposition 1 to estimate the cardinality of the set V used in Proposition 4. Hence, we set d as the GV distance, and ceil the obtained $N_{d_{GV}}$ to guarantee that the cardinality of V is not lower than 1. Finally, we consider that the automorphism group of a code coincides with that of its dual.

B.2 Proof for the monomials automorphism group

We now generalize the results in the previous section to the case of monomials.

Proposition 13. *Let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_q^n$ with the same Hamming weight d and same entries multisets. Let $\mathsf{Mor}_{\mathsf{M}_n}(\boldsymbol{a}, \boldsymbol{b}) = \{\tau \in \mathsf{M}_n \mid \tau(\boldsymbol{a}) = \boldsymbol{b}\}$. Then, the cardinality of $\mathsf{Mor}_{\mathsf{M}_n}(\boldsymbol{a}, \boldsymbol{b})$ is equal to $w!(n-w)!(q-1)^{n-w}$.*

Proof. We reason on the characteristics that a monomial $\tau \in \mathsf{M}_n$ must have, in order to guarantee that the image of $\boldsymbol{a}$ is $\boldsymbol{b}$. To this end, we write $\tau = \pi \rtimes \boldsymbol{v}$, with $\pi \in \mathsf{S}_n$ and $\boldsymbol{v} \in \mathbb{F}_q^{*n}$. Let $E(\boldsymbol{a})$ be the set of positions pointing at null entries in $\boldsymbol{a}$, and let $\bar{E}(\boldsymbol{a})$ be that of indexes pointing an non-null entries in $\boldsymbol{a}$; the same notation is employed for $\boldsymbol{b}$. To have $\tau(\boldsymbol{a}) = \boldsymbol{b}$, the following conditions must be verified:

i) $\pi\left(E(\boldsymbol{a})\right) = E(\boldsymbol{b})$;
ii) if $i \in E$, then v_i can have whichever value;
iii) $\pi\left(\bar{E}(\boldsymbol{a})\right) = \bar{E}(\boldsymbol{b})$;
iv) if $\pi(i) = j$, then $v_i = a_i^{-1} b_j$.

The number of permutations satisfying conditions i) and iii) is given by $w!(n-w)!$, while that of vectors satisfying ii) and iv) corresponds to $(q-1)^{n-w}$. □

Proposition 14. *Let $A \subseteq \mathbb{F}_q^n$, with cardinality M, such that all the contained vectors have Hamming weight w. Let $\mathsf{Aut}_{\mathsf{S}_n}(A) = \{\pi \in \mathsf{S}_n \mid \pi(\boldsymbol{a}) \in A, \ \ \forall \boldsymbol{a} \in A\}$; then, the size of $\mathsf{Aut}_{\mathsf{M}_n}(A)$ is not greater than $M!w!(n-w)!(q-1)^{n-w}$.*

Proof. We reason as in the proof of Proposition 12. If $\tau \in \mathsf{Aut}_{\mathsf{M}_n}(A)$, then for each $\boldsymbol{a} \in A$, either $\tau(\boldsymbol{a}) = \boldsymbol{a}$ or there exists $\boldsymbol{a}' \in A$, $\boldsymbol{a}' \neq \boldsymbol{a}$, such that $\tau(\boldsymbol{a}) = \boldsymbol{a}'$. We write again $A = \left\{\boldsymbol{a}^1, \boldsymbol{a}^2, \cdots, \boldsymbol{a}^M\right\}$, and consider that for each $\tau \in \mathsf{Aut}_{\mathsf{M}_n}(A)$, there exists one and only one bijection $f : \{1, \cdots, M\} \mapsto \{1, \cdots, M\}$ such that $f(i) = j$ if and only if $\tau(\boldsymbol{a}^i) = \boldsymbol{a}^j$. Let $\mathsf{Aut}_{\mathsf{M}_n}^{(f)}(A) = \bigcap_{i=1}^{M} \mathsf{Mor}_{\mathsf{M}_n}\left(\boldsymbol{a}^i, \boldsymbol{a}^{f(i)}\right)$, and consider that

$$\mathsf{Aut}_{\mathsf{M}_n}(A) = \bigcup_{f \in \mathsf{S}_M} \mathsf{Aut}_{\mathsf{M}_n}^{(f)}(A) = \bigcup_{f \in \mathsf{S}_M} \left(\bigcap_{i=1}^{M} \mathsf{Mor}_{\mathsf{M}_n}\left(\boldsymbol{a}^i, \boldsymbol{a}^{f(i)}\right) \right).$$

Using the union bound, we find that the cardinality of $\mathsf{Aut}_{\mathsf{M}_n}(A)$ cannot be greater than $M! \left|\mathsf{Mor}_{\mathsf{M}_n}(\boldsymbol{a}^i, \boldsymbol{a}^{f(i)})\right|$, and we finally rely on Proposition 13 to bound the cardinality of $\mathsf{Mor}_{\mathsf{M}_n}(\boldsymbol{a}^i, \boldsymbol{a}^{f(i)})$. □

Finally, we adapt Theorem 4 to the case of monomials.

Theorem 5. *Let $\mathfrak{C} \subseteq \mathbb{F}_q^n$ be a linear code with minimum distance d. Let $T_q(\boldsymbol{c}) = \left\{b\boldsymbol{c} \mid b \in \mathbb{F}_q^*\right\}$, and let $V \subset \mathbb{F}_q$ be a set of codewords such that*

i) if $\boldsymbol{c} \in \mathfrak{C}$ has weight d, then $T_q(\boldsymbol{c})$ and V have only one element in common;
ii) all codewords in V have weight d.

Let $\mathsf{Aut}_{\mathsf{M}_n}(\mathfrak{C})$ be the monomials automorphism group of $\mathfrak{C}$. Then, the cardinality of $\mathsf{Aut}_{\mathsf{M}_n}(\mathfrak{C})$ is not greater than $(N_d)!(q-1)d!(n-d)!$, where N_d denotes the cardinality of V.

Proof. As in the proof of Theorem 4, we define V such that all of its codewords have the first entry that is equal to 1, and $V_q = \bigcup_{\boldsymbol{c} \in V} T_q(\boldsymbol{c})$. If $\tau(V_q) = V_q$, then for each $\boldsymbol{c} \in V$ there must exist i) another codeword $\boldsymbol{c}' \in V$, and ii) a non null element $b \in \mathbb{F}_q^*$, such that $\pi(\boldsymbol{c}) = b\boldsymbol{c}'$. Then, we have

$$\mathsf{Aut}_{\mathsf{M}_n}\left(V_q\right) = \bigcup_{f \in \mathsf{S}_{N_d}} \bigcup_{b \in \mathbb{F}_q^*} \left(\bigcap_{i=1}^{N_d} \mathsf{Mor}_{\mathsf{M}_n}\left(\boldsymbol{c}^i, b\boldsymbol{c}^{f(i)}\right) \right).$$

Using twice the union bound, we find that an upper bound on the size of $\mathsf{Aut}_{\mathsf{M}_n}\left(V_q\right)$ is given by $N_d!(q-1)w!(n-w)!(q-1)^{n-w}$. Again, the proof is completed by noticing that $\mathsf{Aut}_{\mathsf{M}_n}\left(\mathfrak{C}\right) \subseteq \mathsf{Aut}_{\mathsf{M}_n}\left(V_q\right)$. □

C Proofs of Section 4

Proposition 1. *Let $\mathfrak{C} \subseteq \mathbb{F}_q^n$ be a random linear code with length n and dimension k. Let A^w be the set of codewords of $\mathfrak{C}$ such that*

- *if $\boldsymbol{c} \in \mathfrak{C}$ has weight w, then $b\boldsymbol{c} \in A^w$ for some $b \in \mathbb{F}_q^*$;*
- *$b\boldsymbol{c} \neq b'\boldsymbol{c}'$, for all $b, b' \in \mathbb{F}_q^*$ and all $\boldsymbol{c} \neq \boldsymbol{c}' \in A^w$.*

Then, the average cardinality of A^w is given by

$$N_w = \binom{n}{w}(q-1)^{w-2}q^{k-n+1}.$$

Proof. Let $T_q(\boldsymbol{c}) = \{b\boldsymbol{c} \mid b \in \mathbb{F}_q^*\}$, i.e., the orbit of a vector $\boldsymbol{c}$ under the scalar multiplication by an element of $\mathbb{F}_q^*$. Since $\mathfrak{C}$ is linear, we can find a set $V \subseteq \mathfrak{C}$, containing $(q^k-1)/(q-1)$ codewords, such that

1. $T_q(\boldsymbol{c}) \cap T_q(\boldsymbol{c}') = \varnothing$, $\forall \boldsymbol{c}, \boldsymbol{c}' \in V$, $\boldsymbol{c} \neq \boldsymbol{c}'$;
2. $\mathfrak{C} \setminus \{\mathbf{0}_n\} = \bigcup_{\boldsymbol{c} \in V} T_q(\boldsymbol{c})$, where $\mathbf{0}_n$ is the null codeword.

Without loss of generality, we can require that each $\boldsymbol{c} \in V$ has, as the first non-null entry, the value 1. Now, we focus on the number of codewords in V having a desired weight w, which we indicate as N_w: it is immediately seen that such codewords form the set A^w. Let us pick at random a codeword from V: since the code is random, the probability that it has weight w is given by

$$\frac{\binom{n}{w}(q-1)^{w-1}}{q^{n-1}-1}.$$

Indeed, the numerator corresponds to the number of vectors with length n, weight w and whose first entry is a 1, while the denominator gives the number of vectors whose first non-null entry is a 1. Since V contains $(q^k-1)/(q-1)$ codewords, under the assumption that all of such codewords are randomly and independently picked, we estimate N_w as

$$\frac{q^k-1}{q-1} \cdot \frac{\binom{n}{w}(q-1)^{w-1}}{q^{n-1}-1} \approx \binom{n}{w}(q-1)^{w-2}q^{k-n+1}.$$

□

Proposition 4. *Let $\mathfrak{C}_1 \subseteq \mathbb{F}_q^n$ be a random code with dimension k, and $\mathfrak{C}_2 = \pi(\mathfrak{C}_1)$, with π being a random permutation. Let $L, w \in \mathbb{N}$, with $w \leq n$ and $L \leq N_w$, where N_w is computed as in Proposition 1. Then, the computational complexity of Beullens's algorithm with parameters L and w, to solve the permutation equivalence problem on $\mathfrak{C}_1$ and $\mathfrak{C}_2$, is at least*

$$O\left(\frac{1}{\beta(w,L)} \cdot \left(\frac{2C_{ISD}(q,n,k,w)}{\sum_{i=0}^{L-1}(N_w-i)^{-1}} + C_{perm}\right)\right),$$

where $C_{ISD}(q,n,k,w)$ denotes the time complexity of an ISD algorithm, searching for a codeword with weight w in a linear code with length n, dimension k and defined over $\mathbb{F}_q$, and

$$C_{perm} = Lw(q-2)\big(1+\log_2(L)\big) + \frac{M''}{M}(n-w)!(m!)^{q-1}(n+k^2),$$

with $m = \max\{1\ ,\ \lceil w/(q-1) \rfloor\}$, $\beta(w,L) = 1 - \binom{N_w-L}{L}/\binom{N_w}{L}$, $M = L^2/N_w$ and $M' = (1-\binom{w+q-3}{w-1}^{-1})M + L^2\binom{w+q-3}{w-1}^{-1}$.
If $L \ll N_w$, q is rather large and $\binom{w+q-3}{w-1}^{-1} \ll L^2$, the complexity simplifies as

$$O\left(\frac{2L \cdot C_{ISD}(q,n,k,w) + (n-w)!(n+k^2)}{N_w\left(1-2^{L\log_2(1-L/N_w)}\right)}\right).$$

Proof. We use N_w, as in Proposition 1, to estimate the number of weight-w codewords in the codes $\mathfrak{C}_1$ and $\mathfrak{C}_2$. The algorithm starts by producing the set X, i.e., produces L codewords in $\mathfrak{C}_1$ with weight w. To do this, we perform multiple calls to an ISD algorithm. Let $C_{ISD}(q,n,k,w)$ the complexity of a single call. Since the code contains N_w codewords with the desired weight, and we want to find L of them, we have a complexity that is

$$C_{ISD}(q,n,k,w)\cdot\sum_{i=0}^{L-1}\frac{1}{N_w-i}.$$

Notice that if $L \ll N_w$ (as it is preferable), we have $\sum_{i=0}^{L-1}\frac{1}{N_w-i}\approx\frac{L}{N_w}$.

Notice that, for each found codeword $\boldsymbol{x}$, we have to compute the result of $\mathsf{Lex}(\boldsymbol{x})$: this is done with, at least, $w(q-2)$ multiplications in the finite field. Indeed, we have to compute all multisets $\mathsf{MS}(b\boldsymbol{x})$, for $b\in\mathbb{F}_q^*$: for $b=1$ we do not perform any multiplication, while for $b\neq 1$ the product $b\boldsymbol{x}$ requires w single element multiplications. Neglecting the cost of computing the lexicographically smallest multiset (which we expect to be negligible, with respect to the other terms), we obtain that to build X one faces a cost

$$O\left(\frac{C_{ISD}(q,n,k,w)}{\sum_{i=0}^{L-1}(N_w-i)^{-1}}+Lw(q-2)\right).$$

Then, we do the same for Y. Notice that the list Y does not need to be materialized, since for each found codeword $\boldsymbol{y}$ it is enough to sarch for a collision in X. Using a proper binary search algorithm, a collision can be found with $O\left(\log_2(L)\right)$ operations. Hence, if we put everything together, we get that to build the lists X and Y, and to merge them into P, we have a complexity of

$$O\left(\frac{C_{ISD}(q,n,k,w)}{\sum_{i=0}^{L-1}(N_w-i)^{-1}}+Lw(q-2)\left(1+\log_2(L)\right)\right).$$

Let us now estimate the average size of the list P. Consider that it may contain two types of elements:

- *good collisions*: pairs $\{\boldsymbol{x},\boldsymbol{y}\}$ such that $\pi(\boldsymbol{x})=\boldsymbol{y}$. Since $\mathfrak{C}_1$ contains N_w codewords with weight w, under the assumption that the automorphism group of the code is trivial (i.e., there is only one permutation $\pi\in\mathsf{S}_n$ so that $\pi(\mathfrak{C}_1)=\mathfrak{C}_2$), we get that for each $\boldsymbol{x}\in\mathfrak{C}_1$ there is only one codeword $\boldsymbol{y}\in\mathfrak{C}_2$ such that $\pi(\boldsymbol{x})=\boldsymbol{y}$. Hence, since ISD returns random codewords of weight w, we have that on average the number of good collisions is given by
$$M=\sum_{i=1}^{L}i\cdot\frac{\binom{L}{i}\binom{N_w-L}{L-i}}{\binom{N_w}{L}}=\frac{L^2}{N_w}.$$
- *bad collisions*: pairs $\{\boldsymbol{x},\boldsymbol{y}\}$ such that $\pi(\boldsymbol{x})\neq\boldsymbol{y}$. We now try to estimate the number of such pairs; to this end, we first make some preliminary considerations. First, it is easily seen that for any vector $\boldsymbol{a}$, we have that $\mathsf{Lex}(\boldsymbol{a})$ contains at least a 1. Hence, for each pair $\{\boldsymbol{x},\mathsf{Lex}(\boldsymbol{x})\}\in X$, we may assume that $\mathsf{Lex}(\boldsymbol{x})$ is a random multiset with one entry equal to 1, and the other $w-1$ ones picked at random over $\mathbb{F}_q^*$. The same goes for each $\{\boldsymbol{y},\mathsf{Lex}(\boldsymbol{y})\}\in Y$. To have $\{\boldsymbol{x},\boldsymbol{y}\}\in P$, it must be $\mathsf{Lex}(\boldsymbol{x})=\mathsf{Lex}(\boldsymbol{y})$: since there are $\binom{q+w-3}{w-1}$ possible outputs for the function Lex, we assume that a collision happens with probability $\binom{q+w-3}{w-1}^{-1}$. Hence, the number of bad collisions can be estimated as
$$\tilde{M}=(L^2-M)\binom{w+q-3}{w-1}^{-1}.$$

Hence, we estimate the number of entries in P as

$$M'=M+\tilde{M}=\left(1-\binom{w+q-3}{w-1}^{-1}\right)M+L^2\binom{w+q-3}{w-1}^{-1}.$$

We notice that, when $L^2\binom{w+q-3}{w-1} \ll 1$, the number of bad collisions is extremely low and can be assumed to be equal to 0, so that all pairs in P are good collisions.

For each pair $\{\boldsymbol{x}, \boldsymbol{y}\} \in P$, the algorithm goes through all permutations in $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, \boldsymbol{y})$, i.e., considers all permutations σ such that $\sigma(\boldsymbol{x}) = \boldsymbol{y}$. For a pair $\{\boldsymbol{x}, \boldsymbol{y}\}$, since both $\boldsymbol{x}$ and $\boldsymbol{y}$ have Hamming weight w, the number of such permutations is derived as

$$(n-w)! \prod_{a=1}^{q-1} m_a!,$$

where m_a corresponds to the number of entries that, in both $\boldsymbol{x}$ and $\boldsymbol{y}$, are equal to $a \in \mathbb{F}_q^*$. On average, we expect to have

$$m_a = m := \max\{1\ ,\ \lceil w/(q-1) \rfloor\},$$

and hence estimate the size of $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, \boldsymbol{y})$, for each $\{\boldsymbol{x}, \boldsymbol{y}\} \in P$, as

$$(n-w)!(m!)^q.$$

Hence, we use $(n-w)!$ to estimate To test a candidate permutation σ, it is enough to compute a generator for $\sigma(\mathfrak{C}_1)$ and see if its systematic form is equal to that of $\mathfrak{C}_2$. This can be done with basic linear algebra, and, at a high level, costs $O(n+k^3)\log_2^2(q)$ operations. It is clear that if $\{\boldsymbol{x}, \boldsymbol{y}\}$ is a good collision, then the correct permutation π is in $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, \boldsymbol{y})$. Otherwise, the algorithm goes through all permutations in $\mathsf{Mor}_{\mathsf{S}_n}(\boldsymbol{x}, \boldsymbol{y})$, verifies that no one of them is valid and starts with another couple. The number of couples that one has to test, on average, before finding a good collision is given by M'/M. Hence, the cost of the producing the valid permutation is lower bounded by

$$O\left(\frac{M'}{M} \cdot (n-w)!(n+k^3)\right).$$

Finally, we consider the success probability, i.e., the probability that P contains at least a good collision. This is given by

$$\beta(w, L) = 1 - \frac{\binom{N_w - L}{L}}{\binom{N_w}{L}} = 1 - \prod_{i=0}^{L-1}\left(1 - \frac{L}{N_w - i}\right).$$

If $L \ll N_w$ (as it should be), we get that the failure probability $1 - \beta(w, L)$ is roughly $1 - 2^{L\log_2(1-L/N_w)}$.

Before concluding the proof, we consider that the preferable setting for the algorithm is the one in which the set P contains only good collisions. To do this, one has to carefully choose the value of w and L, so that $L \ll N_w$ and $L^2\binom{w+q-3}{w-1}^{-1} \ll 1$. If such conditions are met, it happens that the bottleneck in the algorithm complexity is represented by the cost of finding the low weight codewords in the given codes. Hence, as a consequence of this consideration and applying some simple approximations (which we have already highlighted through the proof), we can simplify the complexity of the algorithm as

$$O\left(\frac{2L \cdot C_{ISD}(q, n, k, w)}{N_w\left(1 - 2^{L\log_2(1-L/N_w)}\right)}\right) \square$$

D Lexicographic ordering of basis

Let $\mathfrak{V} \subseteq \mathbb{F}_q^n$ be a 2-dimensional subspace. To compute the first lexicographic basis of all possible monomial transformations of $\mathfrak{V}$, we can operate as follows. Let us start with a generic basis $\boldsymbol{B}$ for $\mathfrak{V}$. Now, we multiply each column of $\boldsymbol{B}$ by the inverse of the element in the first row, in order to remain with either zeros or ones in the first row. Now, we permute the columns of the obtained matrix with the goal of placing the zeros in the leftmost part of the first row. To do this, we consider the element in the second row: when two columns have the same element in the first row, we look at the element in the second row, and put on the left the

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

(a)

$$\begin{pmatrix} 0\,1\,0\,0\,1\,1\,1\,0\,1 \\ 1\,0\,0\,2\,0\,1\,2\,0\,3 \end{pmatrix}$$

(b)

$$\begin{pmatrix} 0\,0\,0\,0\,1\,1\,1\,1\,1 \\ 0\,0\,1\,2\,0\,0\,1\,2\,3 \end{pmatrix}$$

(c)

$$\begin{pmatrix} 0\,0\,0\,0\,1\,1\,1\,2\,1 \\ 0\,0\,1\,1\,0\,0\,1\,2\,3 \end{pmatrix}$$

(d)

Fig. 2: Example of lexicographig ordering of a basis, for the finite field with $q = 5$ elements. In figure (a) we show the initial basis, while in the other figures we detail the steps we perform to find the corresponding lexicographic minimum. The matrix in figure (b) is obtained by scaling all columns so that the entry in the first row is a 1. To obtain the matrix in figure (b), we sort the columns. Finally, we see that we have some degrees of freedom, since the third and fourth columns have a zero in the first row and a non null entry in the second row. Hence, we scale these columns and finally obtain the minimum lexicograph basis as in figure (d).

one with the lowest entry. Finally, if we have some non null entry in the second row which corresponds to a null entry in the first row, we can scale the corresponding column to put a one in the second row. For the sake of clarity, in Figure 2 we show an example of this procedure.

We repeat this procedure for all possible basis for $\mathfrak{V}$, and for each basis keep only the resulting lexicographically minimum matrix, obtained as above. Finally, we compare all of such matrices and pick the one which comes first, in the lexicographically order. Notice that, for a given subspace, we test a total of $(q^2-1)/(q^2-q)$ basis, and for each basis we use $O(n)$ operations to find the lexicographic minimum matrix. Finally, we use no more than $O\left(n(q^2-1)(q^2-q)\right)$ operations to compare the resulting matrices.

Type	Variant	Parameters	$q = 13$			$q = 31$			$q = 61$			$q = 127$			$q = 251$		
			$\lvert\mathsf{pk}\rvert$	$\lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert + \lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert + \lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert + \lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert + \lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert + \lvert\sigma\rvert$
Perm	Plain	-	13.52	24.72	38.24	11.39	20.48	31.87	11.16	16.66	27.82	10.56	15.12	25.68	10.5	14.16	24.66
	LESS-F	$t = 247$, $w = 30$	13.52	14.47	27.99	11.39	12.58	23.97	11.16	10.82	21.98	10.56	10.08	20.64	10.5	9.61	20.11
	LESS-M	$\ell = 2$	40.58	18.03	58.61	34.17	14.85	49.02	33.48	11.98	45.14	31.69	10.83	42.52	31.51	10.11	41.62
		$\ell = 3$	94.70	14.02	108.72	79.73	11.53	91.26	78.12	9.28	87.40	73.95	8.37	82.32	73.52	7.81	81.33
		$\ell = 4$	202.93	11.15	214.08	170.85	9.16	180.01	167.40	7.37	174.77	158.48	6.64	165.12	157.56	6.19	163.75
	LESS-FM	$\ell = 2$, $t = 144$, $w = 24$	40.58	10.73	51.31	34.17	9.21	43.38	33.48	7.81	41.29	31.69	7.22	38.91	31.51	6.85	38.36
		$\ell = 3$, $t = 95$, $w = 21$	94.70	8.85	103.55	79.73	7.55	87.28	78.12	6.34	84.46	73.95	5.83	79.78	73.52	5.52	79.04
		$\ell = 4$, $t = 79$, $w = 18$	202.93	7.62	210.55	170.85	6.48	177.33	167.40	5.41	172.81	158.48	4.96	163.44	157.56	4.68	162.24
Mono	Plain	-	12.87	34.42	47.29	10.64	30.27	40.91	10.26	27.25	37.51	9.82	26.48	36.3	9.77	26.38	36.15
	LESS-F	$t = 283$, $w = 28$	12.87	18.72	31.59	10.64	16.90	27.54	10.26	15.58	25.84	9.82	15.24	25.06	9.77	15.2	24.97
	LESS-M	$\ell = 2$	38.62	25.31	63.93	31.92	22.19	54.11	30.78	19.93	50.71	29.47	19.35	48.82	29.32	19.28	48.6
		$\ell = 3$	90.11	19.72	109.83	74.48	17.28	91.76	71.82	15.51	87.33	68.76	15.05	83.81	68.43	15	83.43
		$\ell = 4$	193.11	15.69	208.8	159.60	13.75	173.35	153.90	12.33	166.23	147.35	11.97	159.32	146.64	11.92	158.56
	LESS-FM	$\ell = 2$, $t = 181$, $w = 22$	38.62	13.99	52.61	31.92	12.63	44.55	30.78	11.60	42.38	29.47	11.33	40.8	29.32	11.3	40.62
		$\ell = 3$, $t = 130$, $w = 19$	90.11	11.68	101.79	74.48	10.50	84.98	71.82	9.61	81.43	68.76	9.37	78.13	68.43	9.34	77.77
		$\ell = 4$, $t = 97$, $w = 17$	193.11	10.13	203.24	159.60	9.09	168.69	153.90	8.29	162.19	147.35	8.08	155.43	146.64	8.06	154.7

Table 9: Exploration of the possible parameter sets for LESS instances with security equivalent to a 2^{128} computational effort.

E A Note on Information-Set Decoding

Almost all the algorithms we have analyzed in this section employ ISD as a subroutine. Hence, for the sake of completeness, we provide here an analysis of the ISD algorithm we have chosen, namely, Peters' ISD [26], which comes as an adaptation of Stern's ISD in the non-binary case. Note that, to obtain conservative parameters for our scheme, we consider an optimistic analysis for the algorithm.

We consider a code described by a generator matrix $\boldsymbol{G} \in \mathbb{F}_q^{k\times n}$, and assume that we seek to find a codeword with weight w. Let $E_{m,p} = \left\{\boldsymbol{e} \in \mathbb{F}_q^m,\ \mathrm{wt}(\boldsymbol{e}) = p\right\}$. Peters' algorithm uses as options two integers p and ℓ, with $0 \leq \ell \leq n-k$ and $0 \leq w \leq \min\{w, k+\ell\}$, and works as follows:

1. choose a random permutation $\pi \in \mathsf{S}_n$ and use Gaussian elimination to transform $\pi(\boldsymbol{G})$ into
$$\boldsymbol{G}' = \left[\begin{array}{cc|c|c} \boldsymbol{I}_{\lfloor k/2\rfloor} & \boldsymbol{0}_{\lfloor k/2\rfloor\times\lceil k/2\rceil} & \boldsymbol{G}'_{X,Z} & \\ \boldsymbol{0}_{\lceil k/2\rceil\times\lceil k/2\rceil} & \boldsymbol{I}_{\lceil k/2\rceil} & \boldsymbol{G}'_{Y,Z} & \boldsymbol{G}'_V \end{array}\right],$$
where $\boldsymbol{G}'_{X,Z} \in \mathbb{F}_q^{\lfloor k/2\rfloor\times\ell}$, $\boldsymbol{G}'_{Y,Z} \in \mathbb{F}_q^{\lceil k/2\rceil\times\ell}$ and $\boldsymbol{G}'_V \in \mathbb{F}_q^{k\times(n-k-\ell)}$. If this is not possible, restart by picking another permutation π;
2. create a list $X = \left\{(\boldsymbol{e}, \boldsymbol{s} = \boldsymbol{e}\boldsymbol{G}'_{X,Z}),\ \text{with } \boldsymbol{e} \in E'_{\lfloor k/2\rfloor}, p\right\}$;
3. create a list $Y = \left\{(\boldsymbol{e}, \boldsymbol{s} = \boldsymbol{e}\boldsymbol{G}'_{Y,Z}),\ \text{with } \boldsymbol{e} \in E_{\lceil k/2\rceil}, p\right\}$;
4. produce $Z = \{[\boldsymbol{x} \mid \boldsymbol{y}],\ \text{such that } (\boldsymbol{x}, \boldsymbol{s}_x) \in X,\ (\boldsymbol{y}, \boldsymbol{s}_y) \in Y,\ \boldsymbol{s}_x = -\boldsymbol{s}_y\}$;
5. for each $\boldsymbol{e} \in Z$, compute $\boldsymbol{e}\boldsymbol{G}'_V$; if it has weight $w - 2p$, return $\pi^{-1}([\boldsymbol{e} \mid \boldsymbol{e}\boldsymbol{G}'_V])$.

We now analyze the cost of this algorithm, under some conservative assumptions and optimizations that will ultimately lead to a cost that is likely to be below the actual one.

Optimized Gaussian Elimination In [26], the author proposes to use, for each new iteration, the matrix $\boldsymbol{G}'$ that one has obtained in the previous iteration (clearly, after having removed the effect of the permutation). The gain is in the fact that some columns have already been pivoted: this simplifies the Gaussian elimination step, de facto reducing the count of operations. If we select completely at random a new permutation, we have that on average k^2/n of the selected columns have already been pivoted, meaning that the Gaussian elimination has to be repeated only on the remaining $k - k^2/n$ columns.

Yet, as observed in [26], this procedure can further optimized by considering some additional aspects.

1. We can force some columns to be exactly the same as the ones that were previously selected. If we select only t new columns, then only these need to be reduced. However, this makes iterations dependant and, as an effect, decreases the success probability of each iteration. The probability to succeed after N iterations can be estimated by considering a Markov chain, where the states are identified with the number of set coordinates which are present in the chosen information set. Assuming that we start from $b \in [0; \min\{w, k\}]$ set coordinates (which happens with probability $\binom{w}{b}\binom{n-w}{k-n}/\binom{n}{k}$), replacing only t columns in the information set makes the number of set coordinates in the information set vary from $b-t$ to $\min\{b+t, \min\{w, k\}\}$. In the most extreme case, one sets $t = 1$, which minimizes the cost of Gaussian elimination (since we only swap one column, with respect to the information set we have chosen in the previous iteration), but as a side-effect reduces to a minimum the probability of succeeding after a fixed number of iterations.
2. We can exploit precomputations, in the following way. We notice that certain combinations of the same rows are considered multiple times, when performing Gaussian elimination. Hence, we can precompute such sums and use them when perfomring Gaussian elimination. We denote with m the number of rows we consider, with $m \leq t$. Assuming that we swap only t columns, we have that to perform a complete Gaussian elimination, we use [26] a cost of
$$C_g = (n-1)\left((k-1)(1-q^{-m}) + (q^m - m)\right)\frac{t}{m}\log_2(q). \tag{3}$$
Once again, as noted in [26], the gain becomes ineffective as soon as the finite field becomes large.

Aiming at a rather conservative estimate for the cost of Gaussian elimination, we choose to i) rely on (3) for the cost of Gaussian elimination, selecting the values of t and m that minimizes the cost, and ii) assess the success probability of each iteration by neglecting that we have forced some columns (i.e., we assume that the new permutation is chosen completely at random).

List Merging To merge two lists A and B of the same size L, we consider a cost of $2L\log_2(L)$. Indeed, we first sort one of the two lists (say, B), with a cost of $L\log_2(L)$ operations, and then exploit binary search to determine collisions. In particular, for each element in A, we are able to search for a collision in B with $\log_2(L)$ operations: since A contains L entries, we do the whole search with a cost of $L\log_2(L)$ collisions. If the lists have different sizes, say, L_1 and L_2, we instead get a total cost of $L_1\log_2(L_1) + L_2\log_2(L_1)$, where L_1 and L_2 can be swapped according to which list is sorted. If the difference between the sizes is rather limited, we can only consider the size of the smallest one, and use $2L\log_2(L)$ as a reliable estimate for the collisions search complexity, with $L = \min\{L_1, L_2\}$.

Let us now consider the case in which the two lists contain some repeated elements. To capture this situation, we assume that the elements in the lists A and B take values in $\mathbb{F}_q^\ell$, and that $q^\ell < L$. In such a case, instead of sorting B, we can proceed in a slightly different way. We sort again the list B, but with the aim of producing a new list B' that does not contain replicated elements. For each entry b in B', we store (say, in an auxiliary list $\widetilde{B}$) the indexes pointing to the entries in B corresponding to the value b. Note that this comes with essentially the same cost of the sorting procedure, so we have again a cost of $L\log_2(L)$ operations. Then, we search for collisions between A and B', with a cost of $L\log_2(|B'|)$. For the size of B', we can set a safe upper bound as q^ℓ, so that the collisions search takes no more than $L\ell\log_2(q)$. Every time we find a collisions for an element b, we read from $\widetilde{B}$ the positions of all entries in B with value b. Hence, we can set the cost of this procedure as

$$L\log_2(L) + L\log_2(q^\ell) = L\log_2(Lq^\ell).$$

Enumeration We consider, again, that we search for a codeword having the first entry in X which is equal to 1. By doing this, we have that the list X can be built to contain $\binom{\lfloor k/2\rfloor}{p}(q-1)^{p-1}$ pairs $(\boldsymbol{e}, \boldsymbol{e}\boldsymbol{G}'_{X,Z})$. To compute $\boldsymbol{e}\boldsymbol{G}'_Z$, we can first multiply each row of $\boldsymbol{G}'_Z$, and then combine such scaled vectors. Hence, we get an overall cost of

$$\begin{aligned}&(q-2)\lfloor k/2\rfloor\ell\log_2(q) + \binom{\lfloor k/2\rfloor}{p}(q-1)^{p-1}(p-1)\ell\log_2(q)\\ &= \left((q-2)\lfloor k/2\rfloor + \binom{\lfloor k/2\rfloor}{p}(q-1)^{p-1}(p-1)\right)\ell\log_2(q).\end{aligned}$$

We do the same for Y, using $E_{\lceil k/2\rceil,p}$, and building Y with a cost of

$$\left((q-2)\lceil k/2\rceil + \binom{\lceil k/2\rceil}{p}(q-1)^{p}(p-1)\right)\ell\log_2(q).$$

Then, we proceed by merging the two lists; let $L = \min\{|E_{\lceil k/2\rceil,p}|, q^\ell\}$. To merge the lists we use a complexity given by

$$\binom{\lfloor k/2\rfloor}{p}(q-1)^{p-1}\left((p-1)\log_2(q-1) + \log_2\left(\binom{\lfloor k/2\rfloor}{p}\right) + \log_2(L)\right).$$

Then, the overall cost to produce Z is given by

$$\begin{aligned}C_l = &\left((q-2)\lfloor k/2\rfloor + \binom{\lfloor k/2\rfloor}{p}(q-1)^{p-1}(p-1)\right)\ell\log_2(q)\\ &+ \left((q-2)\lceil k/2\rceil + \binom{\lceil k/2\rceil}{p}(q-1)^{p}(p-1)\right)\ell\log_2(q)\\ &+ \binom{\lfloor k/2\rfloor}{p}(q-1)^{p-1}\left((p-1)\log_2(q-1) + \log_2\left(\binom{\lfloor k/2\rfloor}{p}\right) + \log_2(L)\right).\end{aligned}$$

Weight Testing Finally, we consider the cost of testing the candidates stored in Z. To do this, we first consider that the number of elements in Z can be estimated as

$$|Z| = \frac{|X| \cdot |Y|}{q^\ell} = \frac{\binom{\lfloor k/2 \rfloor}{p}\binom{\lceil k/2 \rceil}{p}(q-1)^{2p-1}}{q^\ell}.$$

For each candidate $\boldsymbol{e} \in Z$, we compute $\boldsymbol{e}\boldsymbol{G}'_V$ and check its weight. Again, we rely on early abort, and hence stop the computation as soon as we reach the weight $w - 2p + 1$. For each entry, we compute $2p - 1$ products (since we always have the first entry of $\boldsymbol{e}$ equal to 1), and $2p - 1$ sums. Hence, we estimate the cost of this step as

$$\begin{aligned} C_t &= |Z| \cdot (w - 2p + 1)\frac{q}{q-1}(4p-2)\log_2(q) \\ &= \frac{\binom{\lfloor k/2 \rfloor}{p}\binom{\lceil k/2 \rceil}{p}(q-1)^{2p-2}}{q^{\ell-1}}(w - 2p + 1)(4p-2)\log_2(q). \end{aligned}$$

Success Probability The success probability of the algorithm is given by

$$P_{p,\ell} = \frac{\binom{w}{p}\binom{w-p}{p}\binom{n-w}{\ell}\binom{n-w-\ell}{k-2p-\ell}}{\binom{n}{k}}.$$

Completing the Puzzle Taking the above analysis into account, we estimate the cost of Peters' ISD as

$$C = \frac{C_g + C_l + C_t}{P_{p,\ell}}. \tag{4}$$

To verify that our treatment yields an optimistic analysis of Peters' algorithm, consider Figures 3 and 4, where we have compared our analysis of Peters' algorithm with that given in [26], for codes with moderate lengths and different code rates and values of q. For each setting, we have set w as the Gilbert - Varshamov distance. This shows that our analysis is conservative and likely an under-estimation of the actual algorithm cost.

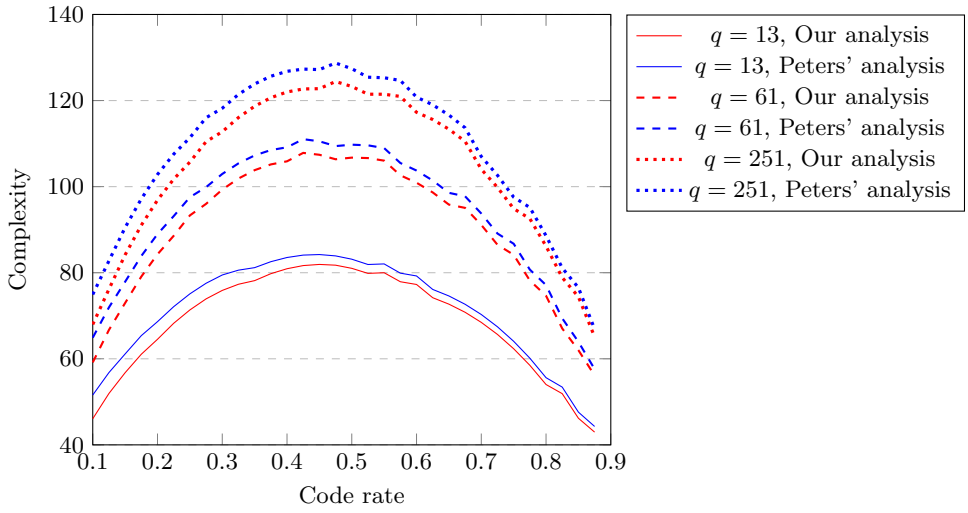

Fig. 3: Complexities to find a minimum-weight codeword, for codes with $n = 200$. The minimum distances of the codes have been estimated through the Gilbert-Varshamov bound. The complexity of the algorithms is expressed in bits.

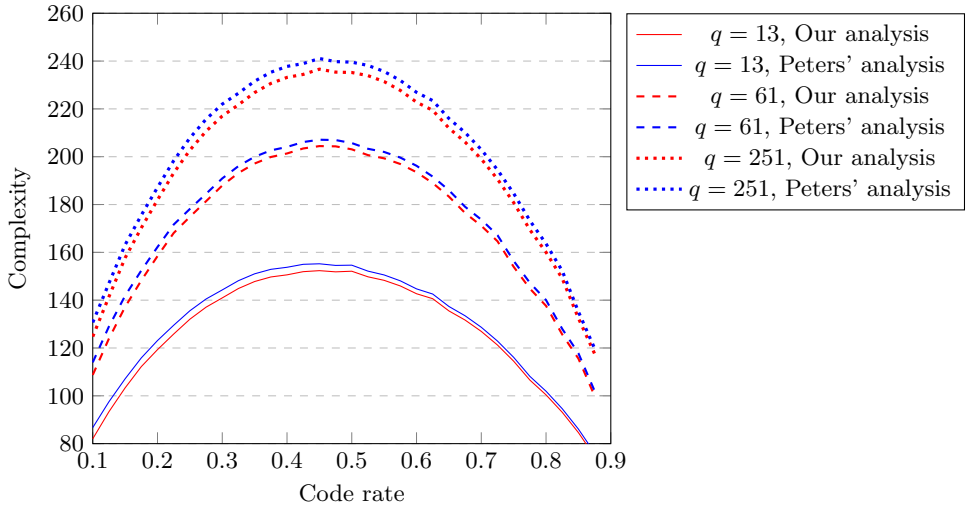

Fig. 4: Complexities to find a minimum-weight codeword, for codes with $n = 400$. The minimum distances of the codes have been estimated through the Gilbert-Varshamov bound. The complexity of the algorithms is expressed in bits.

F Quantum Algorithms for Code Equivalence

To analyze the quantum security of the code equivalence problem, we begin by investigating SDP as a first stepping stone.

F.1 SDP as a 4-sum Problem

We solve the syndrome decoding problem using the approach presented by Kachigar and Tillich [23] which consists in rephrasing it as a 4-sum problem and then using a quantum walk (and Grover search) to solve it with quantum computers. In this section, we briefly review the problem and how it can be rephrased as a 4-sum problem.

Definition 5 (Syndrome Decoding Problem (SDP)). *Let $\mathfrak{C}$ be a code of length n and dimension k defined by a parity-check matrix $\boldsymbol{H} \in \mathbb{F}_q^{(n-k)\times n}$, and let $\boldsymbol{s} \in \mathbb{F}_q^{n-k}$, $w \leq n$, find $\boldsymbol{e} \in \mathbb{F}_q^n$ such that $\boldsymbol{H}\boldsymbol{e}^T = \boldsymbol{s}^T$ and $\boldsymbol{e}$ is of weight w.*

Given that the matrix $\boldsymbol{H}$ has n columns but only $n-k$ rows, SDP is equivalent to solving an underdetermined linear system. As long as $w < n-k$, we can hope that the solution $\boldsymbol{e}$ has k zero coefficients. Let $\pi \in \mathsf{S}_n$ be the permutation of the columns of $\boldsymbol{H}$ that brings all these zeros to the last coefficients, then we have

$$\boldsymbol{H}\boldsymbol{e}^T = \begin{pmatrix}\boldsymbol{H}_1 & \boldsymbol{H}_2\end{pmatrix}\begin{pmatrix}\boldsymbol{e}_1^T \\ \boldsymbol{0}\end{pmatrix} = \boldsymbol{H}_1\boldsymbol{e}_1^T = \boldsymbol{s}^T$$

where $\boldsymbol{H}_1 \in \mathbb{F}_q^{(n-k)\times(n-k)}$, and $\boldsymbol{e}_1 \in \mathbb{F}_q^{n-k}$. We can then solve for $\boldsymbol{e}_1$ and recover $\boldsymbol{e}$. The original strategy to solve SDP due to Prange (which is usually recognized as the first instance of an ISD algorithm) consists in sampling random $\pi \in \mathsf{S}_n$, and for each π, apply the permutation to $\boldsymbol{H}$ and attempt to solve the system of $n-k$ unknowns $\boldsymbol{H}_1\boldsymbol{e}_1^T = \boldsymbol{s}^T$. When an appropriate π is found (which is the difficult part), this yields a solution to SDP at little extra cost.

The above strategy puts the entire burden of the computation on the search for a good permutation π and almost none on the resolution of the subsequent linear system $\boldsymbol{H}_1\boldsymbol{e}_1^T = \boldsymbol{s}^T$. Instead, we can use tradeoff where more permutations are considered (thus making the search for permutations more likely to succeed) at the cost of a more difficult system to solve. More specifically, we introduce two parameters l, p, and denote by $\mathscr{S}$ a set of $k+l$ indices, where instead of assuming that all entries of the solution are zero on these indices, we rather assume that only p of them are non-zero. Thus, the first construction corresponds to $l = p = 0$. The probability that exactly p non-zero coordinates of *a fixed* $\boldsymbol{e}$ belong to $\mathscr{S}$ while the remaining $w-p$ are outside of it is denoted by

$$P_{l,p} := \frac{\binom{k+l}{p}\binom{n-k-l}{w-p}}{\binom{n}{w}}.$$

Assuming a *good* permutation is found, we now need to solve the a new linear algebra problem. By good, we mean that it moves the $k+l$ coordinates of indices in $\mathscr{S}$ to the left of the matrix. We apply this permutation π to the columns of $\boldsymbol{H}$ and assume that the restriction of $\pi(\boldsymbol{H})$ to its last $n-k-l$ columns is of full row rank. We perform a Gaussian elimination, which corresponds to the multiplication of $\pi(\boldsymbol{H})$ by an invertible matrix $\boldsymbol{U}$ on the left such that the resulting matrix has the shape:

$$\boldsymbol{U}\cdot\pi(\boldsymbol{H}) = \begin{pmatrix}\boldsymbol{H}' & \boldsymbol{0}_l \\ \boldsymbol{H}'' & \boldsymbol{I}_{n-k-l}\end{pmatrix}$$

We can write $\pi(\boldsymbol{e}) = (\boldsymbol{e}'||\boldsymbol{e}'')$ where $\boldsymbol{e}' \in \mathbb{F}_q^{l+k}$ has weight p and $\boldsymbol{e}'' \in \mathbb{F}_q^{n-k-l}$. Then, given a good permutation, we can reduce our problem to the resolution of the overdetermined system defined by $\boldsymbol{H}' \in \mathbb{F}_q^{l\times(k+l)}$. Indeed. we have

$$\boldsymbol{U}\cdot\boldsymbol{s}^T = \begin{pmatrix}\boldsymbol{s}'^T \\ \boldsymbol{s}''^T\end{pmatrix} = \boldsymbol{U}\cdot\boldsymbol{H}\boldsymbol{e}^T = \begin{pmatrix}\boldsymbol{H}'\boldsymbol{e}'^T \\ \boldsymbol{H}''\boldsymbol{e}'^T + \boldsymbol{e}''^T\end{pmatrix}$$

Once we find a solution $\boldsymbol{e}'$ to the system $\boldsymbol{H}'\boldsymbol{e}'^T = \boldsymbol{s}'^T$ (which is somewhat expensive), we can immediately derive $\boldsymbol{e}''^T$ as $\boldsymbol{s}''^T - \boldsymbol{H}''\boldsymbol{e}'^T$.

Input: $\boldsymbol{H}$, s, l, p, w.
Output: $\boldsymbol{e}$ of weight w such that $\boldsymbol{H}\boldsymbol{e}^T = \boldsymbol{s}^T$.

1: **for all** $\pi \in \mathsf{S}_n$ **do**
2: Compute row reduction of $\pi(\boldsymbol{H})$: $\boldsymbol{U} \cdot \pi(\boldsymbol{H}) = \begin{pmatrix} \boldsymbol{H}' & \boldsymbol{0}_l \\ \boldsymbol{H}'' & \boldsymbol{I}_{n-k-l} \end{pmatrix}$
3: Compute $\boldsymbol{s}', \boldsymbol{s}''$ defined by $\boldsymbol{U} \cdot \boldsymbol{s}^T = \begin{pmatrix} \boldsymbol{s}'^T \\ \boldsymbol{s}''^T \end{pmatrix}$
4: Find $\boldsymbol{e}'$ of weight p such that $\boldsymbol{H}'\boldsymbol{e}'^T = \boldsymbol{s}'^T$. (method to be determined)
5: $\boldsymbol{e}''^T \leftarrow \boldsymbol{s}''^T - \boldsymbol{H}''e'^T$, $\boldsymbol{e} \leftarrow (\boldsymbol{e}'||\boldsymbol{e}'')$.
6: **if** $\boldsymbol{e}$ has weight w **then**
7: **break**
8: **end if**
9: **end for**
10: **return** $\boldsymbol{e}$.

Algorithm 4: Generic ISD approach

We are now concerned with solving the overdetermined linear system of Step 4 of Algorithm 4. This means that we are given $\boldsymbol{H}' \in \mathbb{F}_q^{l\times(k+l)}$, and $\boldsymbol{s}' \in \mathbb{F}_q^l$, and we are trying to find $\boldsymbol{e}' \in \mathbb{F}_q^{l+k}$ of weight p such that $\boldsymbol{H}'\boldsymbol{e}'^T = \boldsymbol{s}'^T$. This problem can be rephrased as a k-sum problem.

Definition 6 (Generalized k-sum problem). *Consider an abelian group G, and arbitrary set E, a map $f : E \to G$, k subsets $V_0, V_1, \ldots, V_{k-1} \subseteq E$, a map $g : E^k \to \{0,1\}$, and an element $s \in G$. Find $(v_0, \ldots, v_{k-1}) \in V_0 \times \ldots \times V_{k-1}$ such that*

1. $f(v_0) + f(v_1) + \ldots + f(v_{k-1}) = s$.
2. $g(v_0, \ldots, v_{k-1}) = 0$.

The search for $\boldsymbol{e}'$ can be reduced to the 2-sum problem with the following parameters:

$$G = \mathbb{F}_q^l, E = \mathbb{F}_q^{l+k}, f(v) = \boldsymbol{H}'^T\boldsymbol{v}^T, s = \boldsymbol{s}'$$
$$V_0 = \{(\boldsymbol{e}_0, \boldsymbol{0}_{(k+l)/2}) \in \mathbb{F}_q^{l+k}, \boldsymbol{e}_0 \in \mathbb{F}_q^{(l+k)/2}, \mathrm{wt}(\boldsymbol{e}_0) = p/2\}$$
$$V_1 = \{(\boldsymbol{0}_{(k+l)/2}, \boldsymbol{e}_1) \in \mathbb{F}_q^{l+k}, \boldsymbol{e}_1 \in \mathbb{F}_q^{(l+k)/2}, \mathrm{wt}(\boldsymbol{e}_1) = p/2\}$$

and g defined as $g(v_0, v_1) = 0$ if and only if $\boldsymbol{e} = (\boldsymbol{e}'||\boldsymbol{e}'')$ is of weight w where $\boldsymbol{e}' = v_0 + v_1$ and $\boldsymbol{e}''^T = \boldsymbol{s}''^T - \boldsymbol{H}''\boldsymbol{e}'^T$. The cost of the resolution of the 2-sum problem is a tradeoff between the size of the V_i's and the size of G.

To further reduce the size of the sets involved and create more possibilities for tradeoffs, we can reduce the search for $\boldsymbol{e}' \in \mathbb{F}_q^{l+k}$ of weight p such that $\boldsymbol{H}'\boldsymbol{e}'^T = \boldsymbol{s}'^T$ to a 4-sum problem with the following parameters.

$$G = \mathbb{F}_q^l, E = \mathbb{F}_q^{l+k}, f(v) = \boldsymbol{H}'^T v^T, s = s'$$
$$V_{00} = \{(\boldsymbol{e}_{00}, \boldsymbol{0}_{3(k+l)/4}) \in \mathbb{F}_q^{l+k}, \boldsymbol{e}_{00} \in \mathbb{F}_q^{(l+k)/4}, \mathrm{wt}(\boldsymbol{e}_{00}) = p/4\}$$
$$V_{01} = \{(\boldsymbol{0}_{(k+l)/4}, \boldsymbol{e}_{01}, \boldsymbol{0}_{(k+l)/2}) \in \mathbb{F}_q^{l+k}, \boldsymbol{e}_{01} \in \mathbb{F}_q^{(l+k)/4}, \mathrm{wt}(\boldsymbol{e}_{01}) = p/4\}$$
$$V_{10} = \{(\boldsymbol{0}_{(k+l)/2}, \boldsymbol{e}_{10}, \boldsymbol{0}_{(k+l)/4}) \in \mathbb{F}_q^{l+k}, \boldsymbol{e}_{10} \in \mathbb{F}_q^{(l+k)/4}, \mathrm{wt}(\boldsymbol{e}_{10}) = p/4\}$$
$$V_{11} = \{(\boldsymbol{0}_{3(k+l)/4}, \boldsymbol{e}_{11}) \in \mathbb{F}_q^{l+k}, \boldsymbol{e}_{11} \in \mathbb{F}_q^{(l+k)/4}, \mathrm{wt}(\boldsymbol{e}_{11}) = p/4\}$$

and g defined as $g(v_{00}, v_{01}, v_{10}, v_{11}) = 0$ if and only if $\boldsymbol{e} = (\boldsymbol{e}'||\boldsymbol{e}'')$ is of weight w where $\boldsymbol{e}' = v_{00}+v_{01}+v_{10}+v_{11}$ and $\boldsymbol{e}''^T = \boldsymbol{s}''^T - \boldsymbol{H}''\boldsymbol{e}'^T$.

F.2 Quantum Algorithm for the 4-sum Problem

The strategy for solving the 4-sum problem used by Kachigar and Tillich [23] relies on two different ingredients:

- Grover's search algorithm
- Quantum walk

Note that the former is a special case of the latter. Grover's algorithm assumes we know a set S and a function $f : S \to \{0,1\}$ that is implemented by a quantum algorithm $\mathcal{O}_f$. Grover's algorithm returns (with constant probability) a marked element, that is $x \in S$ such that $f(x) = 1$. If we denote by $f^{-1}(\{1\}) = M \subseteq S$ and $\varepsilon = |M|/|S|$, the cost of Grover's algorithm is in

$$O\left(\frac{\text{Cost}(\mathcal{O}_f)}{\sqrt{\varepsilon}}\right).$$

Grover's search algorithm is generalized by the notion of random walk on a graph. We assume that a graph G is given by a set of vertices V and edges E, and we assume that we are looking for a marked element in $M = f^{-1}(\{1\})$ for some $f : V \to \{0,1\}$. The general strategy of a random walk is to start from a vertex $x \in V$, check if $f(x) = 1$, and if not, then walk in the graph by sampling neighboors uniformly at random long enough to ensure the new vertex x' attained is distributed almost uniformly at random in V, then test if $f(x') = 1$. This is repeated until a marked element is found. In addition to running $\mathcal{O}_f$, there are two main steps in a quantum walk that contribute to the overall cost:

- Setup: sampling the first vector and initializing the data structure.
- Update: sampling a neighboor and updating the data structure (we need to update the current node and its neighbors).

Each of the aforementioned steps have a cost that depend on the data structure that is chosen to navigate the graph (note that depending on the model of computation chosen, memory-intensive data structures can penalize the cost). Moreover, the cost is impacted by the shape of the transition matrix $\boldsymbol{M}$. In the case of a d-regular graph (which is relevant to our problem), $\boldsymbol{M} = \frac{1}{d}\boldsymbol{A}$ where $\boldsymbol{A}$ is the adjacency matrix of the graph. The number of update steps required to reach a node almost uniformly distributed is $\tilde{O}\left(\frac{1}{\delta}\right)$ where δ is the *spectral gap* of $\boldsymbol{M}$, i.e. $\delta := 1 - \max_{i>1} |\lambda_i|$ where $(\lambda_i)_{i>1}$ are the eigenvalues of $\boldsymbol{M}$ not equal to 1. The cost of a quantum walk is given by

$$\text{Cost(Setup)} + \frac{1}{\sqrt{\varepsilon}}\left(\text{Cost}(\mathcal{O}_f) + \frac{1}{\sqrt{\delta}}\text{Cost(Update)}\right)$$

The search for solutions of the 4-sum problem reduces to a walk in the product of Johnson graphs of the V_i's. In general, a Johnson graph $J(n,r)$ is an undirected graph whose vertices and the subsets of size r of a given set of size n. There is an edge between vertices S and S' if and only if $|S \cap S'| = r - 1$ (i.e. they differ by only 1 element). The Johnson graph $J(n,r)$ has $\binom{n}{r}$ elements, is $r(n-r)$-regular and its spectral gap is

$$\delta = \frac{n}{r(n-r)}.$$

The product $J^m(n,r)$ of m copies of $J(n,r)$ is the graph whose vertices are of the form $(v_1, \ldots, v_m)$ where each v_i is a vertex of $J(n,r)$, and there is an edge between $(v_1, \ldots, v_m)$ and $(v'_1, \ldots, v'_m)$ if and only if there is an edge between v_i and v'_i for some i, and $v_j = v'_j$ for all $j \neq i$. As recalled in [23], $J^m(n,r)$ has $\binom{n}{r}^m$ elements, is $mr(n-r)$-regular, and its spectral gap satisfies

$$\delta(J^m(n,r)) \geq \frac{1}{m}\delta(J(n,r)).$$

To solve the 4-sum problem with a combination of Grover's search algorithm and a random walk in a product of 4 copies of a Johnson graph, we make the assumption that $|V_i| = V$ for all i, and that $G = G_0 \times G_1$.

We denote by $\pi_i : G \to G_i$ the projection of G onto one of its components. We keep the same notation as in the formulation of the search for $\boldsymbol{e}'$ such that $\boldsymbol{H}'\boldsymbol{e}'^T = \boldsymbol{s}'^T$ as a 4-sum problem. The algorithm is a Grover search for an element $r \in G_1$ such that $g(r) = 1$ where $g : G_1 \to \{0,1\}$ evaluates to 1 if and only if there exists $(\boldsymbol{v}_{00}, \boldsymbol{v}_{01}, \boldsymbol{v}_{10}, \boldsymbol{v}_{11}) \in V_{00} \times V_{01} \times V_{10} \times V_{11}$ such that

$$\begin{aligned}
\pi_1(f(\boldsymbol{v}_{00})) + \pi_1(f(\boldsymbol{v}_{01})) &= r \\
\pi_1(f(\boldsymbol{v}_{10})) + \pi_1(f(\boldsymbol{v}_{11})) &= \pi_1(s) - r \\
\pi_0(f(\boldsymbol{v}_{00})) + \pi_0(f(\boldsymbol{v}_{01})) + \pi_0(f(\boldsymbol{v}_{10})) + \pi_0(f(\boldsymbol{v}_{11})) &= \pi_0(s) \\
g(\boldsymbol{v}_{00}, \boldsymbol{v}_{01}, \boldsymbol{v}_{10}, \boldsymbol{v}_{11}) &= 0
\end{aligned}$$

The overall cost of this procedure is $O\left(\sqrt{|G_1|}\mathrm{Cost}(\mathcal{O}_g)\right)$. The cost of the Grover oracle $\mathcal{O}_g$ is determined by the strategy we employ to find a quadruplet with the desired properties. In [23], Kachigar and Tillich use a quantum walk in the product of the 4 Johson graphs defined by the V_i's and subsets $U_i \subseteq V_i$ of cardinality $U = \Theta\left(V^{4/5}\right)$. By using a similar data structure and update strategy as in [12], they show the following statement.

Proposition 1 (Prop. 3 of [23]). *Assuming that $|G_1| = \Omega(V^{4/5})$, and $|G| = \Omega(V^{8/5})$, it is possible to set up a data structure of size $O(U)$ such that the above quantum walk takes time $\tilde{O}\left(V^{4/5}\right)$.*

Hence the time (i.e. cost expressed in terms of circuit depth) taken to solve our 4-sum problem is in $\tilde{O}\left(|G_1|^{1/2}V^{4/5}\right)$ (see [23, Prop. 2]).

Let us see how this applies to the time complexity of Algorithm 4 for solving the ISD problem. We use Grover's quantum search to determine an appropriate $\pi \in \mathsf{S}_n$ (main "for" loop from Step 1 to Step 9). The oracle we denote by $g_{\mathrm{perm}} : \mathsf{S}_n \to \{0,1\}$ satisfies $g_{\mathrm{perm}}(\pi) = 1$ if and only if Steps 2 to 7 lead to the creation of an appropriate $e \in \mathbb{F}_q^n$. Hence, the cost of $\mathcal{O}_{g_{\mathrm{perm}}}$ is dominated by that of solving the 4-sum problem (i.e. Step 4 of Algorithm 4). We denote by ε the proportion of marked elements (i.e. good permutation π leading to a solution of the 4-sum problem). Hence the total cost of the procedure is

$$O\left(\frac{1}{\sqrt{\varepsilon}}\mathrm{Cost}\left(\mathcal{O}_{g_{\mathrm{perm}}}\right)\right) = \tilde{O}\left(\frac{1}{\sqrt{\varepsilon}}|G_1|^{1/2}V^{4/5}\right).$$

For a given e of weight w, a permutation π yields a solution to the 4-sum problem if the $k+l$ positions of $\mathscr{S}$ are split into 4 sets of size $\frac{k+l}{4}$ containing exactly $p/4$ non-zero coefficients each. We denote by N_w the number of possible e's of weight w, which yields

$$\varepsilon = N_w \frac{\binom{\frac{k+l}{4}}{\frac{p}{4}}^4 \binom{n-k-l}{w-p}}{\binom{n}{w}}.$$

Meanwhile, $G = \mathbb{F}_q^l$ and $G_1 = \mathbb{F}_q^{l/2}$, while the cardinality V of the V_i's satisfies

$$V = (q-1)^{\frac{p}{4}} \binom{\frac{k+l}{4}}{\frac{p}{4}}.$$

Finally, the condition $|G| = \Omega\left(V^{8/5}\right)$ induces the constraint on l and p:

$$l \geq \frac{8}{5}\log_q\left((q-1)^{\frac{p}{4}}\binom{\frac{k+l}{4}}{\frac{p}{4}}\right).$$

The total time is obtained by finding the optimum of this cost when l and p vary.

F.3 Representation Technique and $1 + 1 = 0$

The solution relying on the quantum algorithm to solve the 4-sum problem can be optimized using techniques of [9, 25]. This consists in restricting the search space of elements in G that yield a solution $(\boldsymbol{v}_{00}, \boldsymbol{v}_{01}, \boldsymbol{v}_{10}, \boldsymbol{v}_{11})$ to the 4-sum problem. To do this, we need to increase the ways we can represent a solution $\boldsymbol{e}'$ to the system $\boldsymbol{H}'\boldsymbol{e}'^T = \boldsymbol{s}'^T$. The first way we can do this, introduced in [25], is called the *representation technique*. It consists in relaxing the conditions on the positions of the positions of the $p/4$ non-zero coordinates of the solutions $\boldsymbol{v}_{00}, \boldsymbol{v}_{01}, \boldsymbol{v}_{10}, \boldsymbol{v}_{11}$. Previously, we assumed the permutation π mapped the p indices in $\mathscr{S}$ to 4 groups of size $p/4$ each within $[1, (k+l)/4]$, $[(k+l)/4+1, (k+l)/2]$, $[(k+l)/2+1, 3(k+l)/4]$, $[3(k+l)/4+1, k+l]$. Thus, we requested that the $\boldsymbol{v}_i$' be of the form

$$\begin{aligned}
\boldsymbol{v}_{00} &= (\boldsymbol{e}_{00}, \boldsymbol{0}_{3(k+l)/4}) \\
\boldsymbol{v}_{01} &= (\boldsymbol{0}_{(k+l)/4}, \boldsymbol{e}_{01}, \boldsymbol{0}_{(k+l)/2}) \\
\boldsymbol{v}_{10} &= (\boldsymbol{0}_{(k+l)/2}, \boldsymbol{e}_{10}, \boldsymbol{0}_{(k+l)/4}) \\
\boldsymbol{v}_{11} &= (\boldsymbol{0}_{3(k+l)/4}, \boldsymbol{e}_{11})
\end{aligned}$$

for $\mathrm{wt}(\boldsymbol{e}_{00}) = \mathrm{wt}(\boldsymbol{e}_{01}) = \mathrm{wt}(\boldsymbol{e}_{10}) = \mathrm{wt}(\boldsymbol{e}_{11}) = p/4$. Instead of this, we may only assume that π maps the p indices in $\mathscr{S}$ to 2 groups of size $p/2$ each within $[1, (k+l)/2]$ and $[(k+l)/2+1, k+l]$. In this case, we can write $\boldsymbol{e}' = \boldsymbol{v}_{00} + \boldsymbol{v}_{01} + \boldsymbol{v}_{10} + \boldsymbol{v}_{11}$ with $\boldsymbol{v}_{00}, \boldsymbol{v}_{01}, \boldsymbol{v}_{10}, \boldsymbol{v}_{11}$ of the shape

$$\begin{aligned}
\boldsymbol{v}_{00} &= (\boldsymbol{e}_{00}, \boldsymbol{0}_{(k+l)/2}) \\
\boldsymbol{v}_{01} &= (\boldsymbol{0}_{(k+l)/2}, \boldsymbol{e}_{01}) \\
\boldsymbol{v}_{10} &= (\boldsymbol{e}_{10}, \boldsymbol{0}_{(k+l)/2}) \\
\boldsymbol{v}_{11} &= (\boldsymbol{0}_{(k+l)/2}, \boldsymbol{e}_{11})
\end{aligned}$$

for $\mathrm{wt}(\boldsymbol{e}_{00}) = \mathrm{wt}(\boldsymbol{e}_{01}) = \mathrm{wt}(\boldsymbol{e}_{10}) = \mathrm{wt}(\boldsymbol{e}_{11}) = p/4$. This way, $\boldsymbol{v}_{00} + \boldsymbol{v}_{01}$ and $\boldsymbol{v}_{10} + \boldsymbol{v}_{11}$ both have weight $p/2$ with half of their non-zero coordinates in $[1, (k+l)/2]$, and the other half in $[(k+l)/2+1, k+l]$. Each choice of $p/4$ coordinates of $\boldsymbol{e}'$ within its $p/2$ non-zero coordinates in $[1, (k+l)/2]$ and $p/4$ coordinates within its $p/2$ non-zero coordinates in $[(k+l)/2+1, k+l]$ fixes a quadruplet $\boldsymbol{v}_{00}, \boldsymbol{v}_{01}, \boldsymbol{v}_{10}, \boldsymbol{v}_{11}$ with the shape described above such that $\boldsymbol{e}' = \sum_i \boldsymbol{v}_i$. Therefore we can re-write $\boldsymbol{e}'$ in $\binom{p/2}{p/4}^2$ different ways. With this relaxation, a subset of the original search space over the parameter $r \in G$ yields the solution $\boldsymbol{e}'$ to the system $\boldsymbol{H}'\boldsymbol{e}'^T = \boldsymbol{s}'^T$. In [9], a further refinement of this technique was introduced to take advantage of potential cancellations of coefficients in the sum $(\boldsymbol{v}_{00} + \boldsymbol{v}_{01}) + (\boldsymbol{v}_{10} + \boldsymbol{v}_{11})$ when the weight of $\boldsymbol{v}_{00} + \boldsymbol{v}_{01}$ and of $\boldsymbol{v}_{10} + \boldsymbol{v}_{11}$ are $\frac{p}{2} + \Delta p$ for some Δp. Indeed, if the weight of $\boldsymbol{v}_{00} + \boldsymbol{v}_{01}$ is $\frac{p}{2} + \Delta p$, then if the Δp extra non-zero coefficients of $\boldsymbol{v}_{00} + \boldsymbol{v}_{01}$ are on the same indices as the Δp extra non-zero coefficients of $\boldsymbol{v}_{10} + \boldsymbol{v}_{11}$, then these will cancel and thus won't contribute to the weight of $\boldsymbol{e}'$. This method was described as "$1+1=0$" since it took advantage of the fact that over $\mathbb{F}_2$, 1's in matching indices canceled out. Over $\mathbb{F}_q$, this could be rephrased as "$x + (q-x) = 0$" meaning that if $\boldsymbol{v}_{00} + \boldsymbol{v}_{01}$ has coefficient $x \neq 0$ at the index i, and if $\boldsymbol{v}_{10} + \boldsymbol{v}_{11}$ has coefficient $q - x$ at index i, then the coefficient i of $\boldsymbol{e}' = (\boldsymbol{v}_{00} + \boldsymbol{v}_{01}) + (\boldsymbol{v}_{10} + \boldsymbol{v}_{11})$ is zero and thus does not contribute to $\mathrm{wt}(\boldsymbol{e}')$. The search for a solution of the 4-sum problem is therefore over the new V_i's given by

$$\begin{aligned}
V_{00} = V_{10} &= \left\{ (\boldsymbol{e}_0, \boldsymbol{0}_{(k+l)/2}) \in \mathbb{F}_q^{l+k}, \boldsymbol{e}_0 \in \mathbb{F}_q^{(l+k)/2}, \mathrm{wt}(\boldsymbol{e}_0) = \frac{p}{4} + \frac{\Delta p}{2} \right\} \\
V_{01} = V_{11} &= \left\{ (\boldsymbol{0}_{(k+l)/2}, \boldsymbol{e}_1) \in \mathbb{F}_q^{l+k}, \boldsymbol{e}_1 \in \mathbb{F}_q^{(l+k)/2}, \mathrm{wt}(\boldsymbol{e}_1) = \frac{p}{4} + \frac{\Delta p}{2} \right\}
\end{aligned}$$

The solution $\boldsymbol{e}' \in \mathbb{F}_q^{l+k}$ of weight p can be represented in $\binom{p/2}{p/4}^2 \binom{\frac{k+l}{2} - \frac{p}{2}}{\frac{\Delta p}{2}}^2 (q-1)^{\Delta p}$ different ways as $\boldsymbol{e}' = v_{00} + v_{01} + v_{10} + v_{11}$ where $v_i \in V_i$. This is due to the fact that for each choice of the $p/2$ non-zero coordinates of $v_{00} + v_{01}$, we can choose an additional Δp indices among the $k+l-p$ indices where e' has a zero coefficient (split evenly between $[1, (k+l)/2]$ and $[(k+l)/2+1, k+l]$), together with Δp non-zero coordinates of $v_{00} + v_{01}$ at these indices.

Now that more r's in G can yield a 4-tuple solution to the 4-sum problem, we restrict the search space accordingly by writing $G = G_0 \times G_1 \times G_2$. In this new setting, G_1 where we search r is replaced by $G_1 \times G_2$, and we only search for an r_1 in G_1 (having fixed the r_2 coordinate of $r = (r_1, r_2)$ arbitrarily). We denote by π_0, π_2, π_{12} the projections of G onto G_0, G_1, and $G_1 \times G_2$ respectively. The size of G_2 is adjusted so that for a given (arbitrary) choice of $r_2 \in G_2$, there is only one $r = (r_1, r_2) \in G_1 \times G_2$ such that there is $(v_{00}, v_{01}, v_{10}, v_{11}) \in V_{00} \times V_{01} \times V_{10} \times V_{11}$ such that

$$\begin{aligned}
\pi_{12}(f(\boldsymbol{v}_{00})) + \pi_{12}(f(\boldsymbol{v}_{01})) &= r \\
\pi_{12}(f(\boldsymbol{v}_{10})) + \pi_{12}(f(\boldsymbol{v}_{11})) &= \pi_{12}(s) - r \\
\pi_0(f(\boldsymbol{v}_{00})) + \pi_0(f(\boldsymbol{v}_{01})) + \pi_0(f(\boldsymbol{v}_{10})) + \pi_0(f(\boldsymbol{v}_{11})) &= \pi_0(s) \\
g(\boldsymbol{v}_{00}, \boldsymbol{v}_{01}, \boldsymbol{v}_{10}, \boldsymbol{v}_{11}) &= 0
\end{aligned}$$

For a choice of r_2, we define the search oracle $g_{r_2} : G_1 \to \{0,1\}$ where $g_{r_2}(r_1) = 1$ if and only if $r = (r_1, r_2)$ satisfies the above conditions for a 4-tuple $(\boldsymbol{v}_i)$. As before, the overall cost of this procedure is $O\left(\sqrt{|G_1|}\mathrm{Cost}(\mathcal{O}_{g_{r_2}})\right)$. The cost of the Grover oracle $\mathcal{O}_{g_{r_2}}$ is determined by the cost of the quantum walk in the product of the 4 Johnson graphs defined by the V_i's and subsets $U_i \subseteq V_i$ of cardinality $U = \Theta\left(V^{4/5}\right)$.

Proposition 2 (Prop. 4 of [23]). *Assuming that $|G_1||G_2| = \Omega(V^{4/5})$, $|G| = \Omega(V^{8/5})$, and that there are $\Omega(|G_2|)$ solutions to the 4-sum problem, then the above quantum walk takes time $\tilde{O}\left(V^{4/5}\right)$.*

Hence the time (i.e. cost expressed in terms of circuit depth) taken to solve our 4-sum problem is in $\tilde{O}\left(|G_1|^{1/2}V^{4/5}\right)$.

Let us see how this applies to the time complexity of Algorithm 4 for solving SDP. We use Grover's quantum search to determine an appropriate $\pi \in \mathsf{S}_n$ (main "for" loop from Step 1 to Step 9). The oracle we denote by $g_{\mathrm{perm}} : \mathsf{S}_n \to \{0,1\}$ satisfies $g_{\mathrm{perm}}(\pi) = 1$ if and only if Steps 2 to 7 lead to the creation of an appropriate $\boldsymbol{e} \in \mathbb{F}_q^n$. Hence, the cost of $\mathcal{O}_{g_{\mathrm{perm}}}$ is dominated by that of solving the 4-sum problem (i.e. Step 4 of Algorithm 4). We denote by ε the proportion of marked elements (i.e. good permutation π leading to a solution of the 4-sum problem). Hence the total cost of the procedure is

$$O\left(\frac{1}{\sqrt{\varepsilon}}\mathrm{Cost}\left(\mathcal{O}_{g_{\mathrm{perm}}}\right)\right) = \tilde{O}\left(\frac{1}{\sqrt{\varepsilon}}|G_1|^{1/2}V^{4/5}\right).$$

For a given $\boldsymbol{e}$ of weight w, a permutation π yields a solution to the 4-sum problem if the $k+l$ positions of $\mathscr{S}$ are split into 2 sets of size $\frac{k+l}{2}$ containing exactly $p/2$ non-zero coefficients each. Still denoting the number of weight-w solutions by N_w, we get

$$\varepsilon = N_w \frac{\binom{\frac{k+l}{2}}{\frac{p}{2}}^2 \binom{n-k-l}{w-p}}{\binom{n}{w}}.$$

Meanwhile, we still have $G = \mathbb{F}_q^l$ and to ensure that there are $\Omega(|G_2|)$ solutions to the 4-sum problem, we set $G = \mathbb{F}_q^{l_2}$ for

$$l_2 := \log_q \left(\underbrace{\binom{p/2}{p/4}^2 \binom{\frac{k+l}{2} - \frac{p}{2}}{\frac{\Delta p}{2}}^2 (q-1)^{\Delta p}}_{\text{number of representations of } e'} \right)$$

This yields $G_1 = \mathbb{F}_q^{\frac{l}{2} - l_2}$, while the cardinality V of the V_i's satisfies

$$V = (q-1)^{\frac{p}{4} + \frac{\Delta p}{2}} \binom{\frac{k+l}{2}}{\frac{p}{4} + \frac{\Delta p}{2}}.$$

Finally, the condition $|G| = \Omega\left(V^{8/5}\right)$ induces the constraint on l and p:

$$l \geq \frac{8}{5}\log_q\left((q-1)^{\frac{p}{4}+\frac{\Delta p}{2}}\binom{\frac{k+l}{2}}{\frac{p}{4}+\frac{\Delta p}{2}}\right).$$

The total time is obtained by finding the optimum of this cost when l, p, and Δp vary.

F.4 Leon/Beullens: Code Equivalence as a Claw-Finding Problem

Recall the notation from Section 4.1. We now show how to recast Beullens' algorithm as a claw finding procedure. Assume that $m = \log_2(X) = \log_2(Y)$, and assume that the quantum version of Algorithm 4 to solve SDP takes as input a seed in $\{0,1\}^m$ to search a fraction of the possible permutations expected to yield on average 1 weight-w codeword. Then we have two functions

$$f : \boldsymbol{x} \in \{0,1\}^m \mapsto \text{multiset of weight-}w\text{ codeword given by Algorithm 4 on } \mathfrak{C}_1$$
$$g : \boldsymbol{x} \in \{0,1\}^m \mapsto \text{multiset of weight-}w\text{ codeword given by Algorithm 4 on } \mathfrak{C}_2$$

Finding matching multisets between elements of X and Y corresponds to finding claws of f and g, i.e. pairs $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^m$ such that $f(\boldsymbol{x}) = g(\boldsymbol{y})$. Tani's quantum algorithm [34] allows us to find such claws. According to Beullens, $\Theta(\log(n))$ claws are needed to solve the Code Equivalence problem. The procedure to reconstruct the permutation from these claws is described in Algorithm 2.

Theorem 6 (Tani). *The time to find $\Theta(\log(n))$ pairs of vectors in $X \times Y$ with matching multisets with Tani's claw finding algorithm is in*

$$2^{m+1}C_{ISD} + \tilde{O}\left(2^m 2^{2m\frac{2p}{2p+1}}\right),$$

where $p = \Theta(\log(n))$ is the number of claws required.

Proof. We call the quantum ISD 2^m times (with different seeds) in $\mathfrak{C}_1$ and 2^m times in $\mathfrak{C}_2$. This way, we have two lists X, Y of weight-w codewords of length 2^m. Then we define our functions to be

$$f : \boldsymbol{i} \in \{0,1\}^m \mapsto \text{multiset of } i\text{th codeword of } X$$
$$g : \boldsymbol{j} \in \{0,1\}^m \mapsto \text{multiset of } j\text{th codeword of } Y$$

Note that the circuit depth of the oracle evaluating the check of equality between $f(i), g(j)$ is $\tilde{O}(2^m)$, due to the fact that the circuit needs to contain the information of the codewords for each index in $\{0,1\}^m$. Then from [34], we know that we can find p unique claws in $O(2^{2m\frac{2p}{2p+1}})$ oracle calls.

Note that the above approach is a naive combination of ISD and Tani's claw finding method. If m is so large that the cost of ISD is less than 2^m, then one would rather have ISD evaluated *in superposition* inside of the oracle of Tani's quantum walk-based claw algorithm. This would replace the proposed oracle that has all 2^m weight-w elements hard coded. Typically, we expect the size 2^m of the lists X, Y to be small, thus making ISD the bottleneck. Beullens shows that the right amount of claws is guaranteed as long as $2^m = \Theta\left(\sqrt{|A_1^w|\log(n)}\right)$. To estimate this cost, we approximate $|A_1^w|$ by its average cardinality

$$N_w = \binom{n}{w}(q-1)^{w-2}q^{k-n+1}.$$

Beullens [13, Sec. 4] also proposed a claw-finding procedure to solve the linear code equivalence. In this case, it is not sufficient to compare vectors of low weight from $\mathfrak{C}_1$ and $\mathfrak{C}_2$ to infer information on the hidden

map from $\mathfrak{C}_1$ to $\mathfrak{C}_2$. However, this can be done if we consider 2-dimensional subspaces of $\mathfrak{C}_1$ and $\mathfrak{C}_2$ of support bounded by w. We denote

$$X_1(w) = \{V \subset \mathfrak{C}_1 \mid \dim(V) = 2 \text{ and } |\operatorname{Supp}(V)| \leq w\}$$
$$X_2(w) = \{V \subset \mathfrak{C}_2 \mid \dim(V) = 2 \text{ and } |\operatorname{Supp}(V)| \leq w\}$$

Testing whether $\mu(V) = W$ for $V \in X_1(w)$ and $W \in X_2(w)$ and μ the secret monomial permutation, can be done by comparing $\operatorname{lex}(V)$ and $\operatorname{lex}(W)$ which are values that identify the orbits of V and W under the action of M_n, the group of monomial permutations. More specifically, for a given V, one compute all possible bases $\boldsymbol{x}, \boldsymbol{y}$ of V and find the one that minimizes $\mu'(\boldsymbol{x}), \mu'(\boldsymbol{y})$ where μ' runs over all possible monomial permutations. This procedure costs $O(q)$ time. With $\Omega(\log(n))$ pairs $V, \mu(V)$, we can recover the secret monomial permutation. It is made of the information of a permutation $\pi \in \mathsf{S}_n$ which is recovered in the same way as in the permutation code equivalence problem (as outlined in Algorithm 3), and of a vector of scaling factors in $\mathbb{F}_q^\times$ which are recovered from pairs of codewords with same support.

The bulk of the work in the above procedure is the search for elements of $X_1(w)$ and $X_2(w)$. Beullens proposes an adaptation of the general high level routine of Algorithm 4. First, assume we fix $V \in X_1(w)$, and let $\pi \in S_n$ be chosen at random. Then the probability that 2 indices of $\operatorname{Supp}(V)$ get mapped to $[1, k]$ while the $w - 2$ remaining ones get mapped to $[k + 1, n]$ is

$$P := \frac{\binom{n-k}{w-2}\binom{k}{2}}{\binom{n}{w}}.$$

For each good permutation π, we apply π to the generating matrix of the code and compute its row echelon form according to the first k columns (assuming linear Independence of its restriction to these columns). Then one of the $\binom{k}{2}$ vector spaces spanned by two rows of the resulting matrix is $\pi(V)$. Finding it costs $O(k^2)$ row operations. There are an average of

$$|X_1(w)| = \binom{n}{w} \frac{(q^w - 1)(q^w - q)}{(q^2 - 1)(q^2 - q)} q^{-2(n-k)}$$

different V's to be found, and thus the probability that a given $\pi \in S_n$ yields some $V \in X_1(w)$ is thus

$$\varepsilon := P|X_1(w)| = q^{-2(n-k)} \frac{\binom{n-k}{w-2}\binom{k}{2}}{\binom{n}{w}} \binom{n}{w} \frac{(q^w - 1)(q^w - q)}{(q^2 - 1)(q^2 - q)}.$$

Let $g : S_n \to \{0, 1\}$ be the function that returns 1 if and only if a $V \in X_1(w)$ is found by applying the procedure described above. Then the cost of finding a V is in $O\left(\frac{1}{\sqrt{\varepsilon}} \operatorname{Cost}(\mathcal{O}_g)\right)$ where $\mathcal{O}_g$ is the quantum circuit implementing g. To find the list of 2^m spaces $V \in X_1(w)$ and 2^m spaces $W \in X_2(w)$, we simply apply the above algorithm 2^{m+1} times sequentially. Then we use a claw finding method to recover $p \in \Omega(\log(n))$ pairs V, W with $\operatorname{lex}(V) = \operatorname{lex}(W)$, which costs $\tilde{O}\left(2^m 2^{2m\frac{2p}{2p+1}}\right)$ time with Tani's claw finding algorithm, as noted before. Here we assume that $\operatorname{lex}(V)$ and $\operatorname{lex}(W)$ for $V \in X_1(w), W \in X_2(w)$ are all precomputed classically in time $O(q2^m)$. Then the functions f, g used in Tani's algorithm are:

$$f : \boldsymbol{i} \in \{0, 1\}^m \mapsto \text{lex of } i\text{th space of } X$$
$$g : \boldsymbol{j} \in \{0, 1\}^m \mapsto \text{lex of } j\text{th space of } Y$$

Overall, the time complexity of finding p pairs of matching lex is in

$$O\left(\frac{\log(n)}{\sqrt{\varepsilon}} k^2 n \log(q)\right) + O(q2^m) + \tilde{O}\left(2^m 2^{2m\frac{2p}{2p+1}}\right).$$

As for the case of permutation equivalence resolution, this approach is a naive juxtaposition of quantum ISD and Tani's algorithm. The description of other ways to combine these two approaches (likely through the execution of quantum ISD within Tani's algorithm) is left for future work.

F.5 SSA

To conclude, we have a look at the quantum complexity of the Support Splitting Algorithm. Recall that this, essentially, boils down to the computation of the Weight Enumerator Function (WEF) on the hull of the considered codes. The hull computation requires simple linear algebra, and comes with a cost of $O(n^3)$ operations in the finite field. The computation of the WEF of a code is the bottleneck of SSA. By definition, $\mathsf{Wef}(\mathfrak{C})$ is a bivariate polynomial given by

$$\mathsf{Wef}(\mathfrak{C})(x,y) = \sum_{w=0}^{n} N_w x^w y^{n-w} \quad \text{where } N_w = |\{c \in \mathfrak{C} \mid \mathrm{wt}(c) = w\}.$$

Hence, the computation of $\mathsf{Wef}(\mathfrak{C})$ boils down to the counting of elements of weight w for all $w \in [0,n]$ of a code. This can be seen as n instances of the quantum counting problem, which is defined as follows.

Definition 7 (Counting problem). *Given a set X and a function $f : X \to \{0,1\}$, find $f^{-1}(\{1\})$.*

The computation of $\mathsf{Wef}(\mathfrak{C})$ is thus directly rephrased as n counting problems defined by the functions $f_w : \boldsymbol{c} \in \mathfrak{C} \mapsto 1$ if and only if $\mathrm{wt}(\boldsymbol{c}) = w$. It is pretty clear that the cost of evaluating f_w is in $O(n)$. We denote by $\mathcal{O}_f$ the quantum circuit that reversibly evaluates f. The cost of finding an approximation of N_w can be obtained from a result of Brassard, Høyer and Tapp [15].

Proposition 3 (Cor. 4 of [15]). *Let $f : X \to \{0,1\}$ be a function, $N = |X|$, and $t = f^{-1}(\{1\})$. Then there is an algorithm requiring an expected number of $\Theta\left(\sqrt{tN}\right)$ evaluations of f an estimate $\bar{t}$ such that $\bar{t} = t$ with probability at least $3/4$ using space linear in $\log(N)$.*

Beals, Buhrman, Cleve, Mosca, and de Wolf [8] proved that any quantum algorithm capable of deciding with high probability whetheror not a function $F : \{0,\dots,N-1\} \to \{0,1\}$ is such that $\mid F^{-1}(\{1\}) \mid \leq t$, given some $0 < t < N/2$, must query F at least $\Omega(\sqrt{Nt})$ times, showing the optimality of Proposition 3. The issue to directly apply this result to the computation of $\mathsf{Wef}(\mathfrak{C})$ is the probability of success of quantum couting. Indeed, $\mathsf{Wef}(\mathfrak{C})$ is only successfully computing if the n instances of the couting problems return the correct N_w. In [15, Sec. 4], it is shown that an exact count can be reached with high probability through the use of $\Theta(t)$ additional quantum memory. It is beyond the scope of this document to analyse which trade-off between success probability of quantum counting and quantum memory required would procure us an acceptable probability of success for the comptutation of $\mathsf{Wef}(\mathfrak{C})$. Instead, we focus on time (i.e. circuit depth) to provide a lower bound on the gate count. In this setting, the extra memory does not impact the performance, and we assume quantum counting with high probability of success.

Proposition 4. *Let $\mathfrak{C}$ be an $[n,k]$ linear code over $\mathbb{F}_q$. There is a quantum algorithm for computing $\mathsf{Wef}(\mathfrak{C})$ in time*

$$O\left(n^2 q^{k/2} \sqrt{\max_{w \leq n} N_w}\right),$$

where $N_w \sim \binom{n}{w}(q-1)^{w-2} q^{k-n+1}$ is the number of codewords of weight w.

Proof. We simply apply the counting algorithm for each possible weight (i.e. at most n times). For each call, the query complexity is $O(\sqrt{q^k N_w})$ while the cost of the oracle is $O(n)$, hence the final result.

The above gives us a (conservative) estimate on the cost of compute $\mathcal{S}(\mathfrak{C}, i)$ for $i \in [1,n]$ (as we assume large amounts of memory and only focus on circuit depth). Then the SSA can be reprhased as finding claws for

$$f : i \in [1,n] \longmapsto \mathcal{S}(\mathfrak{C}_1, i)$$
$$g : j \in [1,n] \longmapsto \mathcal{S}(\mathfrak{C}_2, j)$$

Here again, for the sake of finding conservative estimates, we only focus on execution time of this algorithm, ignoring issues relative to quantum memory use. We precompute all $f(i)$ and $g(j)$ in a list. The bit size of each element is in $O(n^2 \log(n))$. We use Tani's algorithm to produce n claws for f, g. The oracle testing $f(i) = g(j)$ has depth $O(n^3 \log(n))$ as it must have all values of $f(i), g(j)$ hard coded.

Theorem 7 (Tani). *The time to find n pairs of indices $(i, j) \in [1, n]^2$ with $f(i) = f(j)$ using Tani's claw finding algorithm is in*

$$O\left(n^3 q^{k/2} \sqrt{\max_{w \leq n} N_w}\right) + \tilde{O}\left(n^3 \cdot n^{2\frac{2n}{2n+1}}\right).$$

Note that as n grows to infinity, Tani's algorithm does not seem to procure a significant advantage through the strategy outlined above. It is however possible that more clever algorithms could take advantage of quantum counting done in superposition. Such improvement is left for future work.

F.6 Deriving Quantum-Safe Parameters

While quantum algorithms for solving the ISD problem have already been fairly optimized, the situation is less clear regarding the Code Equivalence problem. Indeed, the strategies outlined in this document are straightforward combinations of quantum ISD and claw finding through a classical channel. There might be better ways to leverage the general idea of matching low-weight codewords by running quantum ISD in superposition (although it is not clear which gains we could obtain this way, if any).

To derive secure parameters on the code equivalence problem, we use lower bounds on the costs computed in the previous sections. First, note that all costs are in term of circuit depth, and memory costs were not factored in (a very conservative approach in the case of quantum counting, and to some extend for Tani's claw-finding algorithm as well). Then, for the Leon/Beullens claw-finding approach, we under-estimate the cost of creating a list of p weight-w codewords by C_{ISD}. Then the finding of p claws is lower-bounded by the number of oracle calls which is in $O(2^{2m\frac{2p}{2p+1}})$. Finally, we identify the weight w that minimizes the lower bound on the cost given by

$$C_{ISD} + 2^{2m\frac{2p}{2p+1}},$$

where C_{ISD} is parametrized by weight w, $p = \log(n)$ and $m = \frac{1}{2}\log\left(N_w \log(n)\right)$.

A similar conservative lower bound can be obtained for the quantum SSA algorithm by considering only 1 call to the quantum counting routine while the proposed upper bound consists in n successive calls. By counting this cost only once, we anticipate a potentially more sophisticated ad-hoc method capable of deriving the weight enumerator function in one amplitude amplification routine rather than n consecutive ones. The precise description of such an algorithm would be a significant contribution outside of the scope of this document. We therefore use the lower bound for the cost given by

$$n^2 q^{k/2} \sqrt{\max_{w \leq n} N_w} + n^3 \cdot n^{2\frac{2n}{2n+1}}.$$

Note that even when assuming costless quantum memory, the quantum SSA described in this document does not achieve a square root improvement over its classical counterpart. This is due to the fact that the oracle complexity to count the target t is in $O(\sqrt{Nt})$. In instances where $t = N_w$ is large, this heavily penalizes the cost.