On the representation of self-orthogonal codes and applications to cryptography

Marco Baldi[1], Rahmi El Mechri[1,2], Paolo Santini[1], and Riccardo Schiavoni[1]

[1]Università Politecnica delle Marche, Ancona, Italy
[2] Scuola IMT Alti Studi, Lucca, Italy
{m.baldi, p.santini}@univpm.it, {r.elmechri, r.schiavoni}@pm.univpm.it

Abstract. The *hull* of a linear code is the intersection between the code and its dual. When the hull is equal to the code (i.e., the code is contained in the dual), the code is called *self-orthogonal* (or *weakly self-dual*); if, moreover, the code is equal to its dual, then we speak of a *self-dual* code. For problems such as the Permutation Equivalence Problem (PEP) and (special instances of) the Lattice Isomorphism Problem (LIP) over q-ary lattices, codes with a sufficiently large hull provide hard-to-solve instances.
In this paper we describe a technique to compress the representation of a self-orthogonal code. Namely, we propose an efficient compression (and decompression) technique that allows representing the generator matrix of a self-orthogonal code with slightly more than $k(n-k) - \binom{k+1}{2}$ finite field elements. The rationale consists in exploiting the relationships deriving from self-orthogonality to reconstruct part of the generator matrix entries from the others, thus reducing the amount of entries one needs to uniquely represent the code. For instance, for self-dual codes, this almost halves the amount of finite field elements required to represent the code. We first present a basic version of our algorithm and show that it runs in polynomial time and, moreover, its communication cost asymptotically approaches the lower bound set by Shannon's source coding theorem. Then, we provide an improved version which reduces both the size of the representation and the time complexity, essentially making the representation technique as costly as Gaussian elimination.
As concrete applications, we show that our technique can be used to reduce the public key size in cryptosystems based on PEP such as LESS and SPECK (achieving approximately a 50% reduction in the public key size), as well as in the Updatable Public Key Encryption Scheme recently proposed by Albrecht, Benčina and Lai, which is based on LIP.

1 Introduction

Given a linear code $\mathscr{C} \subseteq \mathbb{F}_q^n$ defined over a finite field $\mathbb{F}_q$, its *hull* is the intersection between $\mathscr{C}$ and its dual $\mathscr{C}^\perp$. In other words, the hull contains all the codewords of $\mathscr{C}$ that also belong to $\mathscr{C}^\perp$. Thus, every two codewords in the hull

Table 1: Classification of codes according to the hull dimension. We indicate with n and k the code length and dimension, respectively, while h indicates the hull dimension.

Name	Relation with dual	Parameters
Linear Complementary Dual	$\mathscr{C} \cap \mathscr{C}^{\perp} = \{\mathbf{0}\}$	$h = 0$
Self-orthogonal	$\mathscr{C} \subseteq \mathscr{C}^{\perp}$	$h = k \leqslant n - k$
Self-dual	$\mathscr{C} = \mathscr{C}^{\perp}$	$h = k = n/2$

are orthogonal; this holds also when considering the product of a codeword by itself (we will sometimes refer to this property as self-orthogonality).

Codes with trivial hull (i.e., a hull containing only the null vector) are called Linear Complementary Dual (LCD) codes. If instead the hull is equal to the code itself, then we say that the code is self-orthogonal (or weakly self-dual). A self-orthogonal code which is also equal to its dual is called self-dual. In Table 1 we recall some relations between the hull dimension and the code parameters.

It turns out that the wide majority of codes have small hull with large probability. Indeed, as proven in [16], for a finite field with q elements, a random code is LCD with probability approximately $1 - 1/q - 1/q^2$ and has hull dimension $h \geqslant 1$ with probability approximately $q^{-h(h+1)/2}$.

In certain applications, we need codes with large hull. For instance, when considering quantum stabilizer codes constructed according to the Calderbank-Shor-Steane (CSS) framework [4, 8, 17], a popular solution is that of employing a classical code which is dual containing, i.e., such that the code contains the dual (which means that the dual is self-orthogonal). Analogously, the study of self-orthogonal codes arises in other areas, such as the design of locally recoverable codes for distributed storage systems [12] and, last but not least, post-quantum cryptography. In fact, it turns out that codes with a large hull provide hard instances of the Permutation Equivalence Problem (PEP), that is, the problem of determining whether, on input two linear codes, there exists a permutation mapping one code onto the other. PEP is employed as a security assumption in schemes such as SPECK [2] and the updatable public encryption scheme in [1], which we will refer to as ABL in the following (ABL is the acronym obtained from the authors' initials). Interestingly, ABL is built upon the Learning With Errors (LWE) problem and makes use of PEP as a special case of the Lattice Isomorphism Problem (LIP)[1].

[1] In the ABL scheme, lattices are defined over $\mathbb{Z}_q$ with q prime and the employed lattice isomorphisms are restricted to (signed) permutations. However, for the security proof, the authors require to find a permutation (not a signed permutation) sending one lattice into the other. This is exactly PEP for codes over prime finite fields. This is also acknowledged by the authors in [1] which, among the various contributions of the paper, propose an algorithm to sample uniformly at random codes with some desired hull dimension.

In schemes based on PEP, codes with large hull are necessary to avoid efficient attacks. Indeed, attacks such as [3, 15] take time which is exponential in the hull dimension and, in particular, become polynomial time when the hull is trivial. When the hull dimension is large enough, such attacks are ruled out and currently best known solvers take time which is instead exponential in the code length, regardless of the hull dimension [6, 9].

As another scheme based on the problem of determining an equivalence between linear codes, we mention also LESS [7]. LESS is based on a slightly different flavor of the code equivalence problem, called Linear Equivalence Problem (LEP), in which the isometry can be any *monomial*, a transformation that permutes and scales coordinates. Observe that permutations are just a special case of monomials.

In SPECK, LESS and ABL, the public key contains descriptions of linear codes. The resulting key size grows quadratically with the code length and, ultimately, is definitely large. This represents a major drawback for such schemes.

1.1 Our contribution

We introduce a technique to reduce the amount of bits required to represent a self-orthogonal code, based on the observation that self-orthogonality creates some relations between the entries of the code generator matrix. Namely, let $\mathbf{G} = (\mathbf{I}_k,\ \mathbf{A})$, with $\mathbf{I}_k$ being the identity matrix of side k and $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$[2]. Since the code is self-orthogonal, we know that $\mathbf{G}\mathbf{G}^\top = \mathbf{0}$ which, in turns, implies $\mathbf{A}\mathbf{A}^\top = -\mathbf{I}_k$. Let us view the $k(n-k)$ entries of $\mathbf{A}$ as unknowns and $\mathfrak{S} : \mathbf{A}\mathbf{A}^\top = -\mathbf{I}_k$ as a system of multivariate equations. Notice that $\mathfrak{S}$ contains at most $\binom{k+1}{2}$ algebraically independent equations. Therefore, if one knows the values of $k(n-k) - \binom{k+1}{2}$ entries of $\mathbf{A}$, the remaining entries may be obtained by solving $\mathfrak{S}$. If this holds true, then the whole code can be represented by using only $k(n-k) - \binom{k+1}{2}$ entries.

This simple idea comes with some technical caveats. First, the system $\mathfrak{S}$ is quadratic, hence finding a solution can be hard [18]. However, we show that with the knowledge of some specific entries of $\mathbf{A}$ (namely, those in the upper triangular part), the system's solution can be found by solving only linear equations. This approach can be described as follows. At step $i \in \{1, \dots, k\}$, we want to recover the i-th row of $\mathbf{A}$, which we indicate as $\mathbf{a}_i$. All the entries in the above rows $\mathbf{a}_1, \dots, \mathbf{a}_{i-1}$ are known (because the previous steps have been completed). For row $\mathbf{a}_i$, we only know the elements in columns $i, \dots, n-k$: the unknown $i-1$ entries are found by solving the subsystem $\mathfrak{S}_i$ formed by the equations $\langle \mathbf{a}_j\ ;\ \mathbf{a}_i \rangle = \sum_{\ell=1}^{n} a_{j,\ell} \cdot a_{i,\ell} = 0$, for each $j \in \{1, \dots, i-1\}$.

As another issue, one has to deal with the possibility that some of the subsystems $\mathfrak{S}_i$ may not be uniquely solvable. Indeed, when $\mathfrak{S}_i$ contains some linearly

[2] To avoid burdening the explanation here, for simplicity we assume that the considered code admits $\{1, \dots, k\}$ as information set. However, as we show in the paper, this is not required and our technique allows to represent any self-orthogonal code.

dependent equations, then the system is underdetermined and its solution is not unique. We show that, following the approach we have previously described, this happens whenever (some of) the leading submatrices of $\mathbf{A}$ (i.e., the square matrices formed by the entries in the top-left corner of $\mathbf{A}$) are singular. We show that, as a workaround, applying a permutation of the columns of $\mathbf{A}$ is enough and that, remarkably, such a permutation exists for every self-orthogonal code and can be computed in polynomial time. This comes with a very mild communication overhead, since the permutation shall be represented as well.

All in all, our representation technique for self-orthogonal codes runs in polynomial time ($O(n^4)$ with a schoolbook implementation) and has a communication cost (i.e., number of bits required to represent a self-orthogonal code) whose worst case converges asymptotically to the optimal value resulting from Shannon's source coding theorem.

We then refine the above procedure by observing that the entries $a_{i,i}$, for each $i \in \{1, \dots, k\}$, do not need to be included in the code representation. Indeed, we can enrich each subsystem $\mathfrak{S}_i$ with the self anti-orthogonality equation $\langle \mathbf{a}_i \, ; \, \mathbf{a}_i \rangle = -1$. With standard algebraic manipulations, the system $\mathfrak{S}_i$ remains efficiently solvable and we save additional bits.[3].

As another tweak, we show how to handle underdetermined subsystems more efficiently. Indeed, especially when q increases, the expected amount of undetermined subsystems is rather small. Whenever this happens, we communicate a number of variables equal to the rank deficiency of the coefficient matrix for the corresponding subsystem. The communication cost overhead is, on average, much smaller than the cost of communicating a permutation of length $n - k$.

As a final tweak, we show that with careful manipulations of the equations in $\mathfrak{S}$, the complexity of solving the system can be reduced down to $O(n^3)$.

As concrete applications, we consider how the cryptographic schemes SPECK, LESS and ABL would benefit from our technique. Noteworthy, since SPECK uses self-dual codes having rate 1/2, our technique would lead to almost halve the size of its public keys without any modification to the scheme. For what concerns LESS, the required modifications appear to be somewhat heavier since we propose to switch from LEP on random codes to PEP on self-orthogonal codes. Yet, tweaking LESS to use self-orthogonal codes and PEP, we can again essentially halve the public key size (without any relevant change in security, as state-of-the-art attacks on PEP and LEP are essentially the same).

Similar considerations hols for the ABL scheme, even if in this case the public key reduction is less important because the instances recommended in [1] have very small code rate.

1.2 Paper organization

The paper is organized as follows. In Section 2 we introduce the notation we use in the paper and then recall background notions about coding theory and

[3] Instead of each $a_{i,i}$, we just need to send one bit in the worst case. Indeed, the anti self-orthogonality equation is quadratic: when there are two distinct roots, we use this bit to determine which one of the two solutions shall be considered.

self-orthogonal codes. In Section 3 we describe and analyze our basic procedure for representing self-orthogonal codes. The refinement of this procedure is given in Section 4, while the applications to cryptographic schemes are discussed in Section 5. Section 6 closes the paper with some concluding remarks.

All the code we have used for preparing this paper (simulations, together with proof of concept implementations of the algorithms devised in this paper) are publicly available and can be found at `https://github.com/secomms/self-orthogonal-codes`.

2 Notation and background

In this section we settle the notation we use throughout the paper and then recall useful background concepts.

2.1 Notation

We denote by $\mathbb{F}_q$ a finite field with q elements. Vectors (resp., matrices) are denoted with bold lowercase (resp., uppercase) letters. Given two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}_q^n$, their dot product is denoted as $\langle \mathbf{a} \; ; \; \mathbf{b} \rangle = \sum_{i=1}^n a_i \cdot b_i$.

The null matrix is indicated as $\mathbf{0}$ (its size will always be clear from the context); $\mathbf{I}_k$ denotes the identity matrix of side k. For a matrix $\mathbf{A} \in \mathbb{F}_q^{k \times n}$ and a set $J \subseteq \{1, \ldots, n\}$, we indicate as $\mathbf{A}_J$ the submatrix formed by the columns of $\mathbf{A}$ that are indexed by the elements of J. Given a matrix $\mathbf{A}$, we indicate its rank with $\mathrm{rk}(\mathbf{A})$.

The symmetric group on n elements is indicated as S_n and its elements are viewed as permutations. For a matrix $\mathbf{P} \in \mathbb{F}_q^{n \times n}$, we write $\mathbf{P} \in S_n$ if it is a permutation matrix (every row and column has only one entry equal to 1, while all other entries are null).

Given a matrix $\mathbf{A} \in \mathbb{F}_q^{k \times n}$, with $k \leqslant n$, for each $\ell \in \{1, \ldots, k\}$ we refer to its ℓ-th *leading submatrix*, and indicate it as $\mathbf{A}_{[\ell]}$, as the matrix obtained from $\mathbf{A}$ by deleting its last $k - \ell$ rows and its last $n - \ell$ columns. In other words, $\mathbf{A}_{[\ell]}$ is the top-left $\ell \times \ell$ submatrix of $\mathbf{A}$. Its determinant is called ℓ-th leading minor.[4] We extend this notation to the non-square case, by denoting with $\mathbf{A}_{[\ell,m]}$ the submatrix formed by the entries in the top-left ℓ rows and m columns of $\mathbf{A}$.

2.2 Linear codes

A linear code $\mathscr{C} \subseteq \mathbb{F}_q^n$ with length n and dimension k is a k-dimensional linear subspace of $\mathbb{F}_q^n$. A linear code can be represented using either a generator matrix or a parity-check matrix, that is, full rank matrices $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ and $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ such that

$$\mathscr{C} := \left\{ \mathbf{u}\mathbf{G} \,\middle|\, \mathbf{u} \in \mathbb{F}_q^k \right\} = \left\{ \mathbf{c} \in \mathbb{F}_q^n \,\middle|\, \mathbf{H}\mathbf{c}^\top = \mathbf{0} \right\}.$$

[4] Some authors refer to leading submatrices as first principal submatrices and, consequently, to their determinant as first principal minor.

The parity-check matrix serves as the generator matrix for the dual code $\mathscr{C}^\perp \subseteq \mathbb{F}_q^n$, which is the linear subspace of dimension $n-k$ formed by all vectors that are orthogonal to all codewords of $\mathscr{C}$.

For any code, there exist multiple generator matrices since, for any non singular $\mathbf{S} \in \mathbb{F}_q^{k\times k}$, one has that $\mathbf{G}$ and $\mathbf{SG}$ are generator matrices for the same code. We sometimes refer to this operation as change of basis. With obvious caveats, the same holds for parity-check matrices.

For a code $\mathscr{C} \subseteq \mathbb{F}_q^n$ with dimension k, an information set $J \subseteq \{1, \ldots, n\}$ is a non-ordered set of size k such that, for any two distinct codewords $\mathbf{c}, \mathbf{c}' \in \mathscr{C}$, the subvectors formed by the entries indexed by J are different. Put it differently, the submatrix $\mathbf{G}_J$ formed by the columns of $\mathbf{G}$ which are indexed by J form a non singular $k \times k$ matrix. This means that, for each information set J, one can define a "special" generator matrix, which we refer to as *standard form*, such that the columns indexed by J form the identity matrix of size k. Starting from any generator matrix $\mathbf{G}$ for $\mathscr{C}$, one can obtain the generator matrix in standard form as $\mathbf{G}_J^{-1} \cdot \mathbf{G}$ (i.e., using $\mathbf{G}_J^{-1}$ as a change of basis). When $J = \{1, \ldots, k\}$, a generator matrix in standard form has the form $\mathbf{G} = (\mathbf{I}_k,\ \mathbf{A})$, which we call *systematic.*

Self-orthogonal codes For a linear code $\mathscr{C} \subseteq \mathbb{F}_q^n$, we define its hull as the intersection between $\mathscr{C}$ and $\mathscr{C}^\perp$. We say that the hull is trivial whenever it contains only the null vector (i.e., the hull is a space with dimension 0). When instead the hull is equal to the code, then we the code is said to be self-orthogonal.

Definition 1 (Self-orthogonal and self-dual codes). *Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ be a linear code. If $\mathscr{C} \subseteq \mathscr{C}^\perp$, then we say that $\mathscr{C}$ is self-orthogonal (or weakly self-dual). If, moreover, $\mathscr{C} = \mathscr{C}^\perp$, then we say that the code is self-dual.*

For a self-orthogonal code $\mathscr{C}$, for every two codewords $\mathbf{c}, \mathbf{c}'$, one has $\langle \mathbf{c}\ ;\ \mathbf{c}' \rangle = \mathbf{c} \cdot \mathbf{c}'^\top = 0$. This implies that $\mathbf{G}\mathbf{G}^\top = \mathbf{0}$. For a given finite field with q elements, the number of self orthogonal codes can be counted precisely [16, Theorem 1].

Theorem 1 *For q a prime power and $k, n \in \mathbb{N}$ with $k \leqslant n/2$, let $\mathfrak{C}_{n,k,q}$ denote the set of all self-orthogonal codes with length n and dimension k over a finite field with q elements. The cardinality of $\mathfrak{C}_{n,k,q}$ is*

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

We also recall [16, Theorem 4] which, later on, will become rather handy to derive the asymptotics for the number of self-orthogonal codes.

Theorem 2 *Let $q \in \mathbb{N}$ be a (constant) prime power and $0 \leqslant k \leqslant n$. Let $\mathbb{F}_q$ be a finite field with q elements. For growing k and n,*

$$\lim_{\substack{k\to\infty \\ n\to\infty}} \frac{|\mathfrak{C}_{n,k,q}|}{\left[{n \atop k} \right]_q} = \frac{\gamma_q}{\prod_{i=1}^{k} q^i - 1},$$

where $\gamma_q = \prod_{i=1}^{\infty} \frac{1-q^{-i}}{1-q^{-2i}}$ and $\left[{n \atop k} \right]_q = \prod_{i=0}^{k-1} \frac{q^n - q^i}{q^k - q^i}$.

3 Representing self-orthogonal codes: the basic idea

In this section we describe the basic version of the algorithm we propose to compress the representation of the generator matrix of a self-orthogonal code. We first present the underlying main procedure which, however, is not guaranteed to work for all codes. Later on, we refine it to tackle all self-orthogonal codes. We also show that, for what concerns the output size, the techniques in this section converge asymptotically to the lower bound set by Shannon's source coding theorem.

3.1 Core idea

We first describe our idea at a very high-level. For simplicity, we start by considering only codes for which $\{1, \ldots, k\}$ forms an information set; later on, we will generalize the analysis to encompass all codes. Remember that, for such codes, there exists a systematic generator matrix $\mathbf{G} = (\mathbf{I}_k,\ \mathbf{A})$, with $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$. Since we are considering self-orthogonal codes we know that

$$\mathbf{G}\mathbf{G}^\top = \mathbf{0} \quad \implies \quad \mathbf{A}\mathbf{A}^\top = -\mathbf{I}_k.$$

$\mathbf{A}$ is said to be an *antiorthogonal*[5] matrix, and denoting by $\mathbf{a}_i$ its i-th row, we have that for each pair of indexes i, j:

$$\langle \mathbf{a}_i \ ; \ \mathbf{a}_j \rangle = \begin{cases} 0 & \text{if } i \neq j, \\ -1 & \text{if } i = j. \end{cases} \tag{1}$$

The equations originating from the antiorthogonality of $\mathbf{A}$ can be split in two sets: the $\binom{k}{2}$ pairwise orthogonality equations, which must hold for each pair of different rows of $\mathbf{A}$, and the k *self-antiorthogonality* equations, which is the property for which every row has a dot product with itself equal to -1.

Our idea is that of exploiting relations such as (1) to reconstruct some coefficients of $\mathbf{A}$ from its other entries. Let us view the $k(n-k)$ entries of $\mathbf{A}$ as unknowns, and let $\mathfrak{S}$ be the system that arises from (1), for each pair of indices $i \leqslant j$. The resulting system has $\binom{k+1}{2}$ equations: even if we assume that such

[5] Massey in [13] distinguishes between square and nonsquare matrices with this property, by calling the latter *row-antiorthogonal* matrices. For the sake of simplicity we will use the term antiorthogonal for both cases.

equations are algebraically independent, the system is underdetermined[6]. However, assuming that one knows the values of at least $k(n-k)-\binom{k+1}{2}$ unknowns, then the system may become determined and its solution should give the values of the remaining unknowns. In other words, one may represent the code using, say, only $k(n-k)-\binom{k+1}{2}$ entries of $\mathbf{A}$, since the other entries of $\mathbf{A}$ can be obtained by solving $\mathfrak{S}$.

As a technical caveat, one has to consider that $\mathfrak{S}$ is a system of quadratic equations which, generally, are hard to solve. However, we show that, with the knowledge of some specific entries of $\mathbf{A}$ (namely, all the entries in the upper triangular part), then the system can be solved by considering $\binom{k}{2}$ linear equations. For this purpose, we now focus on the $\binom{k}{2}$ pairwise orthogonality equations and do not consider the k self-antiorthogonality equations, which are necessarily quadratic. The latter will be reconsidered later, in Section 4.1.

Let us assume that, for each row $\mathbf{a}_i$, the elements $a_{i,i},\ldots,a_{i,n-k}$ are known. The remaining entries are treated as unknowns and are labeled as follows (rather frequently, we will highlight unknowns in red):

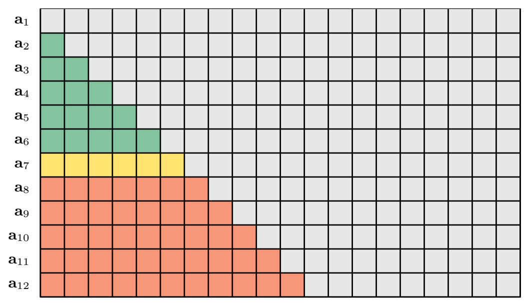
$$\mathbf{A}=\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,k-2} & a_{1,k-1} & a_{1,k} & \cdots & a_{1,n-k}\\
x_{2,1} & a_{2,2} & a_{3,3} & \cdots & a_{3,k-2} & a_{3,k-1} & a_{3,k} & \cdots & a_{2,n-k}\\
x_{3,1} & x_{3,2} & a_{3,3} & \cdots & a_{4,k-2} & a_{4,k-1} & a_{4,k} & \cdots & a_{3,n-k}\\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \\
x_{k-2,1} & x_{k-2,2} & x_{k-2,3} & \cdots & a_{k-2,k-2} & a_{k-2,k-1} & a_{k-2,k} & \cdots & a_{k-2,n-k}\\
x_{k-1,1} & x_{k-1,2} & x_{k-1,3} & \cdots & x_{k-1,k-2} & a_{k-1,k-1} & a_{k-1,k} & \cdots & a_{k-1,n-k}\\
x_{k,1} & x_{k,2} & x_{k,3} & \cdots & x_{k,k-2} & x_{k,k-1} & a_{k,k} & \cdots & a_{k,n-k}
\end{pmatrix}$$

Using all pairwise orthogonality equations, we obtain the following system:

$$\mathfrak{S}=\begin{cases}
\mathfrak{S}_2=\Big\{a_{1,1}\cdot x_{2,1}=-\sum_{j=2}^{n-k}a_{1,j}\cdot a_{2,j}\\[2ex]
\mathfrak{S}_3=\begin{cases}a_{1,1}\cdot x_{3,1}+a_{1,2}\cdot x_{3,2}=-\sum_{j=3}^{n-k}a_{1,j}\cdot a_{3,j}\\ x_{2,1}\cdot x_{3,1}+a_{2,2}\cdot x_{3,2}=-\sum_{j=3}^{n-k}a_{2,k}\cdot a_{3,j}\end{cases}\\
\quad\vdots\\
\mathfrak{S}_k=\begin{cases}a_{1,1}\cdot x_{k,1}+\ldots+a_{1,k-1}\cdot x_{k,k-1}=-\sum_{j=k}^{n-k}a_{1,j}\cdot a_{k,j}\\ x_{2,1}\cdot x_{k,1}+\ldots+a_{2,k-1}\cdot x_{k,k-1}=-\sum_{j=k}^{n-k}a_{2,j}\cdot a_{k,j}\\ \quad\vdots\\ x_{k-1,1}\cdot x_{k,1}+\ldots+a_{k-1,k-1}\cdot x_{k,k-1}=-\sum_{j=k}^{n-k}a_{k-1,k}\cdot a_{k,k}\end{cases}
\end{cases}$$

Observe that the system contains quadratic equations but, solving the subsystems in a precise order (namely, from $\mathfrak{S}_2$ to $\mathfrak{S}_k$), in practice we just need to consider linear equations.

[6] Since we are considering self-orthogonal codes, we have $k\leqslant n/2$: with simple manipulations, one can show that this always implies $\binom{k+1}{2}\leqslant k(n-k)$.



$$\mathfrak{S}_7 = \begin{cases} \langle \mathbf{a}_1 \,;\, \mathbf{a}_6 \rangle = 0 \\ \langle \mathbf{a}_2 \,;\, \mathbf{a}_6 \rangle = 0 \\ \langle \mathbf{a}_3 \,;\, \mathbf{a}_6 \rangle = 0 \\ \langle \mathbf{a}_4 \,;\, \mathbf{a}_6 \rangle = 0 \\ \langle \mathbf{a}_5 \,;\, \mathbf{a}_6 \rangle = 0 \end{cases}$$

Fig. 1: Representation of the solution of the system $\mathfrak{S}_7$ to recover row $\mathbf{a}_7$. Grey cells correspond to the elements of $\mathbf{A}$ that are known from the beginning, green cells correspond to the symbols that have been recovered in the previous steps $2, \ldots, 6$. Yellow cells represent the entries that are found by solving $\mathfrak{S}_7$, while red cells represent the symbols that will be found in subsequent steps.

For now, let us assume that each subsystem $\mathfrak{S}_i$ admits a unique solution (later on, we describe how to deal with the cases in which this is not true). We divide our procedure in k steps and consider that, at step $i \in \{1, \ldots, k\}$, we aim to recover $\mathbf{a}_i$ by finding its first $i - 1$ entries.

For step $i = 1$ there is no entry to recover[7]. We now consider the i-th step, with $i > 1$. We set up the system $\mathfrak{S}_i$ formed by the equations $\langle \mathbf{a}_j \,;\, \mathbf{a}_i \rangle = 0$, for each $j \in \{1, \ldots, i - 1\}$. At step i, the above rows $\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}$ are known since they have been determined by solving $\mathfrak{S}_2, \ \ldots, \ \mathfrak{S}_{i-1}$. For row $\mathbf{a}_i$ we only know the elements in columns $i, \ldots, n-k$, and thus need to determine $i-1$ unknowns. We do this by solving $\mathfrak{S}_i$ which, now, looks as follows

$$\mathfrak{S}_i = \begin{cases} a_{1,1} \cdot x_{i,1} + \cdots + a_{1,i-1} \cdot x_{i,i-1} = -\sum_{j=i}^{n-k} a_{1,j} \cdot a_{i,j} \\ a_{2,1} \cdot x_{i,1} + \cdots + a_{2,i-1} \cdot x_{i,i-1} = -\sum_{j=i}^{n-k} a_{2,j} \cdot a_{i,j} \\ \quad \vdots \\ a_{i-1,1} \cdot x_{i,1} + \cdots + a_{i-1,i-1} \cdot x_{i,i-1} = -\sum_{j=i}^{n-k} a_{i-1,j} \cdot a_{i,j} \end{cases}$$

Once we solve $\mathfrak{S}_i$ and fully determine $\mathbf{a}_i$, we proceed by solving $\mathfrak{S}_{i+1}$. See Figure 1 for a representation of this procedure.

3.2 Dealing with underdetermined subsystems

We now show how to deal with some subsystems that are underdetermined, and therefore admit more than one valid solution. In particular, we show that

[7] The reader may wonder why we are starting from step $i = 1$ even if we do nothing in this step. As we will show in Section 4.1, when self-antiorthogonality equations are reintroduced, step $i = 1$ is devoted to the reconstruction of $a_{1,1}$. Therefore, we are keeping it here as well, for consistency with the rest of the paper.

for any self-orthogonal code with generator matrix $\mathbf{G} = (\mathbf{I}_k,\ \mathbf{A})$, a reordering of the columns in $\mathbf{A}$ is actually enough to avoid that any of the subsystems $\mathfrak{S}_2, \ldots, \mathfrak{S}_{k-1}$ is underdetermined.

For $i = 2, \ldots, k$, let us express the i-th subsystem $\mathfrak{S}_i$ as

$$\mathfrak{S}_i : \quad \mathbf{A}_{[i-1]} \cdot \mathbf{x}^{(i)} = \mathbf{b}^{(i)},$$

where $\mathbf{A}_{[i]} \in \mathbb{F}_q^{(i-1)\times(i-1)}$ is the $(i-1) \times (i-1)$ leading submatrix, $\mathbf{b}^{(i)} \in \mathbb{F}_q^{i-1}$ and $\mathbf{x}^{(i)} = \begin{pmatrix} x_{i,1} \\ \vdots \\ x_{i,i-1} \end{pmatrix}$ collects the unknowns for this step.

First, we know that $\mathfrak{S}_i$ must admit at least a solution $\mathfrak{S}_i$ by construction. Whenever $\mathbf{A}_{[i-1]}$ has full rank, then the solution is unique. If, instead, $\mathbf{A}_{[i-1]}$ has some rank deficiency, then $\mathfrak{S}_i$ is undetermined and therefore admits also other solutions.

We start by observing that, because of self-orthogonality, $\mathbf{A}$ is guaranteed to have rank k.

Theorem 3 *Let $\mathbf{G} = (\mathbf{I}_k,\ \mathbf{A}) \in \mathbb{F}_q^{k\times n}$ be the generator matrix of a self-orthogonal code with dimension k. Then, $\mathbf{A}$ has full rank k.*

Proof. Since the code is self-orthogonal, we know that $\mathbf{A}\mathbf{A}^\top = -\mathbf{I}_k$. Remember that, for any two matrices $\mathbf{B}, \mathbf{C}$, one has $\mathrm{rk}(\mathbf{B}\cdot\mathbf{C}) \leqslant \min\{\mathrm{rk}(\mathbf{B})\ ;\ \mathrm{rk}(\mathbf{C})\}$. Thus

$$k = \mathrm{rk}(-\mathbf{I}_k) \leqslant \min\left\{\mathrm{rk}(\mathbf{A})\ ;\ \mathrm{rk}(\mathbf{A}^\top)\right\} = \mathrm{rk}(\mathbf{A}),$$

where the last equality comes from the fact that $\mathrm{rk}(\mathbf{A}) = \mathrm{rk}(\mathbf{A}^\top)$. Since for a self-orthogonal code $k \leqslant n-k$, we have that $\mathbf{A}$ has full rank k. □

We now show that, using just a permutation of columns, we can always transform $\mathbf{A}$ into a matrix so that the leading submatrices are non-singular.

Theorem 4 *Let $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$, with $k \leqslant n/2$, be a matrix with rank k. Then, there exist (at least) a permutation $\mathbf{P} \in S_{n-k}$ such that, for every $i \in \{1, \ldots, k\}$, the i-th leading submatrix of $\mathbf{A}\cdot\mathbf{P}$ is non-singular.*

Proof. We give a constructive proof that shows, also, how $\mathbf{P}$ can be computed. This can be done with a procedure such as Algorithm 1, on input $\mathbf{A}$. Notice that the algorithm is essentially computing the Row Reduced Echelon Form (RREF) of the input matrix, with only minor tweaks. For this reason, we refer to it as RREF*.

The algorithm can be split into k steps, with each step being an iteration of the loop in instructions 3–12. At every step, the algorithm updates $\mathbf{A}$ so that,

Algorithm 1: RREF*

```
Input: matrix $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$ with rank $k \leqslant n/2$
Output: permutation $\mathbf{P} \in S_{n-k}$ such that the leading submatrices of $\mathbf{A}\cdot\mathbf{P}$
        are non-singular
Set $\mathbf{u} = (1,\ 2,\ \ldots,\ n-k) \in \mathbb{N}^k$;
$i = 1$;// Number of pivoted columns
while $i \leqslant k$ do
    /* Find column that goes to position $i$ */
    $j = i$;
    while $a_{i,j} = 0$ do
        Update $j \leftarrow j+1$
    if $j \neq i$ then
        Swap columns $i$ and $j$ of $\mathbf{B}$;
        Swap entries $i$ and $j$ of $\mathbf{u}$;
    /* Do sums between rows to cancel all elements below $i$-th entry of $i$-th column.
       Notice that $\mathbf{a}_\ell = (a_{\ell,1},\ \ldots,\ a_{\ell,n-k})$ is the $\ell$-th row of $\mathbf{A}$ */
    for $\ell = i+1,\ldots,k$ do
        Update $\mathbf{a}_\ell \leftarrow \mathbf{a}_\ell - a_{i,i}^{-1}\cdot a_{\ell,i}\cdot\mathbf{a}_i$;
    Update $i \leftarrow i+1$;
Set $\mathbf{P} \in S_{n-k}$ as the permutation matrix so that $i$ goes to $u_i$;
return $\mathbf{P}$;
```

before starting step i, the matrix $\mathbf{A}$ looks as follows

$$\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,i-1} & a_{1,i} & \cdots & a_{1,k} & \cdots & a_{1,n-k}\\
0 & a_{2,2} & a_{2,3} & \cdots & a_{2,i-1} & a_{2,i} & \cdots & a_{2,k} & \cdots & a_{2,n-k}\\
0 & 0 & a_{3,3} & \cdots & a_{3,i-1} & a_{3,i} & \cdots & a_{3,k} & \cdots & a_{3,n-k}\\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots\\
0 & 0 & 0 & \ddots & a_{i-1,i-1} & a_{i-1,i} & \cdots & a_{i-1,k} & \cdots & a_{i-1,n-k}\\
0 & 0 & 0 & \cdots & 0 & a_{i,i} & \cdots & a_{i,k} & \cdots & a_{i,n-k}\\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots\\
0 & 0 & 0 & \cdots & 0 & a_{k,i} & \cdots & a_{k,k} & \cdots & a_{k,n-k}
\end{pmatrix},$$

where, for $\ell \in \{1,\ldots,i-1\}$, the entry $b_{\ell,\ell}$ is non-null. We first observe that, at instructions 4–6, the algorithm always finds a valid index j. Indeed, if this does not exist, then this means that the i-th row of the updated $\mathbf{A}$ is null, but this contradicts the fact that $\mathbf{A}$ has full rank. After a valid column is found, it is moved to position i and row operations are performed to eliminate all the entries which are below the entry in row i and column i.

We now show that the permutation provided by Algorithm 1 is valid. Let $\mathbf{A}$ indicate the matrix after the k-th step: observe that this matrix is upper triangular and, by construction, has non zero entries in the main diagonal. Let us denote it as $\mathbf{A}'$, and let $\mathbf{L} \in \mathbb{F}_q^{k\times k}$ be the lower triangular matrix so that $\mathbf{A}' = \mathbf{L}\cdot\mathbf{A}\cdot\mathbf{P}$. In particular, $\mathbf{L}$ has only ones in the main diagonal. Let $\mathbf{A}' = \mathbf{A}\cdot\mathbf{P}$ and observe that, for each $i \in \{1,\dots,k\}$, $\mathbf{A}_{[i]} = \mathbf{L}_{[i]}\cdot\mathbf{A}'_{[i]}$, hence $\mathbf{A}'_{[i]} = \mathbf{L}_{[i]}^{-1}\cdot\mathbf{A}_{[i]}$. Since both $\mathbf{L}_{[i]}$ and $\mathbf{A}_{[i]}$ are non-singular (both matrices are either lower or upper triangular, with all elements in the main diagonal different from 0), also $\mathbf{A}'_{[i]}$ is non singular. □

Putting together the two above theorems, we have a compression technique for any matrix $\mathbf{G} = (\mathbf{I}_k,\ \mathbf{A})$ generating a self-orthogonal code. Indeed, whenever $\mathbf{A}$ has some singular leading submatrix, we just need to find a permutation $\mathbf{P}$ according to Theorem 4 and set $\mathbf{A}' = \mathbf{A}\cdot\mathbf{P}$. Since $\mathbf{A}'\cdot\mathbf{A}'^\top = -\mathbf{I}_k$, we can fully represent $\mathbf{A}'$ using only the upper triangular part. After one reconstructs $\mathbf{A}'$ using the procedure in Section 3.1, the original matrix $\mathbf{A}$ can be easily obtained by permuting back columns with the inverse of $\mathbf{P}$.

Remark 1. We observe that $\mathbf{P}$ can be compactly represented as an ordered set of k distinct elements of $\{1,\ \dots,\ n-k\}$. Indeed, we do not need to care about the ordering of the last $n-k$ columns, we can just consider them as sorted in ascending order.

3.3 Compressing and decompressing: the general algorithm

We now put all the results from the previous section together and generalize to any self-orthogonal code, i.e., we also get rid of the constraint that $\{1,\dots,k\}$ must be an information set.

At a high level, our algorithm builds on the observation that for every self-orthogonal code $\mathscr{C} \subseteq \mathbb{F}_q^n$, there exists at least one permutation $\pi \in S_n$ such that $\pi(\mathscr{C})$ admits a generator matrix in systematic form, and can be compressed using the technique in Section 3.1. In other words, this permutation guarantees that $\pi(\mathscr{C})$ has $\{1,\ \dots,\ k\}$ as an information set and also that, in the non-systematic part of the (systematic) generator matrix, all leading submatrices are non singular. Existence of π is guaranteed by Theorems 3 and 4. Once we know $\pi \in S_n$, we can represent $\mathscr{C}$ by communicating π and compressing $\pi(\mathscr{C})$ using the technique from Section 3.1.

The resulting compression and decompression algorithms are detailed in Algorithms 2 and 3. Since π has a special form, in these algorithms we do not visualize π but, instead, break down how such a permutation is constructed. In particular, we split its action using two permutations, one describing the information set and the other one describing how columns outside of the information set shall be sorted, in order to allow reconstruction of the coefficients using the technique from Section 3.1. The first permutation is actually represented using only the set U representing the information set. The second permutation is computed using Algorithm 1.

Algorithm 2: Compress

```
Input: generator matrix $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ for self-orthogonal code $\mathscr{C} \subseteq \mathbb{F}_q^n$
       with dimension $k$
Output: permutation $\mathbf{P} \in S_{n-k}$, set $U \subseteq \{1,\ldots,n\}$ of size $k$, upper
        triangular matrix $\mathbf{V} \in \mathbb{F}_q^{k\times(n-k)}$
/* Find information set and use it for standard form */
1 Find an information set $U \subseteq \{1,\ldots,n\}$ for $\mathscr{C}$;
2 Set $\mathbf{A} = \mathbf{G}_U^{-1} \cdot \mathbf{G}_{\{1,\ldots,n\}\setminus U}$;
/* Permute columns of $\mathbf{A}$ so that all leading submatrices are non-singular */
3 Compute $\mathbf{P} = \mathsf{RREF}^*(\mathbf{A})$;
4 Set $\mathbf{A}' = \mathbf{A} \cdot \mathbf{P}$;
/* Encode coefficients for polynomial system */
5 Set $\mathbf{V}$ as the upper triangular part of $\mathbf{A}'$;
6 return $U$, $\mathbf{P}$ and $\mathbf{V}$;
```

Algorithm 3: Decompress

```
Input: permutation $\mathbf{P} \in S_{n-k}$, sets $U \subseteq \{1,\ldots,n\}$ of size $k$, upper
       triangular matrix $\mathbf{V} \in \mathbb{F}_q^{k\times(n-k)}$
Output: generator matrix $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ for self-orthogonal code $\mathscr{C} \subseteq \mathbb{F}_q^n$
        with dimension $k$
/* Reconstruct the full matrix $\mathbf{A}'$ from $\mathbf{V}$ */
1 for $i = 2,\ldots,k$ do
      /* Set up and solve subsystem $\mathfrak{S}_i$ */
2     Set $\mathbf{b}^{(i)} \in \mathbb{F}_q^{(i-1)}$ so that, for $j \in \{1,\ldots,i-1\}$, $b_j^{(i)} = -\sum_{\ell=i}^{n-k} v_{j,\ell} \cdot v_{i,\ell}$;
3     Set $(v_{i,1},\ldots,v_{i,i-1}) = \mathbf{V}_{[i-1]}^{-1} \cdot \mathbf{b}^{(i-1)}$;// Retrieve $i$-th row of $\mathbf{V}$
/* Undo column permutation $\mathbf{P}$ */
4 Set $\mathbf{A} = \mathbf{V} \cdot \mathbf{P}^{-1}$;
/* Put information set in positions indexed by $J$ */
5 Set $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ so that $\mathbf{G}_U = \mathbf{I}_k$, $\mathbf{G}_{\{1,\ldots,n\}\setminus U} = \mathbf{A}$;
6 return $\mathbf{G}$
```

The above algorithms clearly run in polynomial time, as they just require basic linear algebra operations. We now show that the above compression technique is asymptotically almost optimal, i.e., its output has binary length which differs only in low order terms from the binary length obtained with an optimal encoding scheme (assuming the code to be represented is picked randomly over $\mathfrak{C}_{n,k,q}$). First, through Theorem 5 we describe the asymptotic behavior of the number of self-orthogonal codes when both k and n grow; the proof of the theorem can be found in Appendix A.

Theorem 5 *For constant q, when both k and n go to infinity,*

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

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

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

According to Shannon's source coding theorem, a perfect encoding scheme for the set of self-orthogonal codes would take $L_{n,k,q} = \log_2\left(|\mathfrak{C}_{n,k,q}|\right)$ bits, that is,

$$\begin{aligned} L_{n,k,q} &= \left(k(n-k) - \frac{k(k+1)}{2}\right) \cdot \log_2(q) + \log_2(\omega_q) \\ &= \left(k(n-k) - \frac{k(k+1)}{2}\right) \cdot \log_2(q) \cdot \big(1+o(1)\big) \\ &= \frac{k}{2} \cdot (2n-3k-1) \cdot \big(1+o(1)\big). \end{aligned}$$

In the next theorem we show that Algorithm 2 outputs binary strings whose length converges asymptotically to $L_{n,k,q}$; the proof of the theorem is given in Appendix B.

Theorem 6 *Assuming perfect encoding schemes are used for all the outputs of Algorithm 2, then the algorithm has worst case output length which is*

$$L'_{n,k,q} = \left(k(n-k) - \frac{k(k-1)}{2}\right) \cdot \log_2(q) + \log_2\left(k! \cdot \binom{n}{k} \cdot \binom{n-k}{k}\right).$$

For constant q and both k and n going to infinity,

$$L'_{n,k,q} = L_{n,k,q} \cdot \big(1+o(1)\big) \quad \Longrightarrow \quad L'_{n,k,q} \sim L_{n,k,q}.$$

We remark that the worst case output length is derived assuming that $\mathbf{P}$ and U are uniformly distributed over the corresponding domains, as this maximizes their entropy. In practice, this may not be true. For instance, for large q, we have $U = \{1, \ldots, k\}$ with large probability so that, in practice, this set may not be communicated in many cases. As we show in the next section, with some observations and tricks, the non-dominant terms in $L'_{n,k,q}$ can be significantly reduced.

4 Optimizing the representation technique

We show how the compression and decompression algorithms given in the previous section can be tweaked to reduce the amount of bits required to represent self-orthogonal codes. Moreover, we provide a precise description of how such algorithms shall be implemented, optimizing also their computational cost.

4.1 Exploiting self-antiorthogonality equations

So far, given $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$ such that $\mathbf{A}\mathbf{A}^\top = -\mathbf{I}_k$, we used only the pairwise orthogonality relations between different rows. However, by considering also the self-antiorthogonality condition of each row, we can recover one more coefficient per row. Indeed, once $i-1$ rows of $\mathbf{A}$ have been fully determined, say, the first $i-1$ ones, then it is possible to recover exactly i coefficients of the i-th row. This is done by imposing orthogonality of the i-th row to the $i-1$ previous rows through $\mathfrak{S}_i$, and enriching $\mathfrak{S}_i$ with the equation resulting from self-antiorthogonality of the i-th row.

In other words, let us represent $\mathbf{A}$ as follows (again, red entries are unknowns):

$$\mathbf{A} = \begin{pmatrix} x_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,k-2} & a_{1,k-1} & a_{1,k} & \cdots & a_{1,n-k} \\ x_{2,1} & x_{2,2} & a_{2.3} & \cdots & a_{3,k-2} & a_{3,k-1} & a_{3,k} & \cdots & a_{2,n-k} \\ x_{3,1} & x_{3,2} & x_{3,3} & \cdots & a_{4,k-2} & a_{4,k-1} & a_{4,k} & \cdots & a_{3,n-k} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \\ x_{k-2,1} & x_{k-2,2} & x_{k-2,3} & \cdots & x_{k-2,k-2} & a_{k-2,k-1} & a_{k-2,k} & \cdots & a_{k-2,n-k} \\ x_{k-1,1} & x_{k-1,2} & x_{k-1,3} & \cdots & x_{k-1,k-2} & x_{k-1,k-1} & a_{k-1,k} & \cdots & a_{k-1,n-k} \\ x_{k,1} & x_{k,2} & x_{k,3} & \cdots & x_{k,k-2} & x_{k,k-1} & x_{k,k} & \cdots & a_{k,n-k} \end{pmatrix}.$$

At step i, we assume that rows $\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}$ have been fully determined (i.e., we have determined the values of all unknowns in the first $i-1$ rows) and we need to recover the first i coefficients of the i-th row. We set up $\mathfrak{S}_i$ with the $i-1$ equations $\langle \mathbf{a}_j \; ; \; \mathbf{a}_i \rangle = 0$, for each $j \in \{1, \ldots, i-1\}$, together with $\langle \mathbf{a}_i \; ; \; \mathbf{a}_i \rangle = -1$. The resulting system looks as follows:

$$\mathfrak{S}_i = \begin{cases} a_{1,1} \cdot x_{i,1} + \cdots + a_{1,i} \cdot x_{i,i} = -\sum_{j=i+1}^{n-k} a_{1,j} \cdot a_{i,j} \\ a_{2,1} \cdot x_{i,1} + \cdots + a_{2,i} \cdot x_{i,i} = -\sum_{j=i+1}^{n-k} a_{2,j} \cdot a_{i,j} \\ \vdots \\ a_{i-1,1} \cdot x_{i,1} + \cdots + a_{i-1,i} \cdot x_{i,i} = -\sum_{j=i+1}^{n-k} a_{i-1,j} \cdot a_{i,j} \\ {x_{i,1}}^2 + \cdots + {x_{i,i}}^2 = -1 - \sum_{j=i+1}^{n-k} a_{i,j}^2 \end{cases} \tag{2}$$

Observe that now, differently from before, we have a quadratic equation.

For $i = 1$, the system contains only the quadratic equation from which we find $x_{1,1}$. For each $i \geqslant 2$, instead, we also have $i-1$ linear equations in addition to the quadratic one. Let $\mathbf{b}^{(i)} \in \mathbb{F}_q^n$ be the vector collecting the terms in the right-hand side of (2). Then, we observe that the first $i-1$ equations of $\mathfrak{S}_i$ are linear and can be written as

$$\mathfrak{L}_i : \quad \mathbf{A}_{[i-1]} \cdot \begin{pmatrix} x_{i,1} \\ \vdots \\ x_{i,i-1} \end{pmatrix} + \begin{pmatrix} a_{1,i} \\ \vdots \\ a_{i-1,i} \end{pmatrix} \cdot x_{i,i} = \begin{pmatrix} b_1^{(i)} \\ \vdots \\ b_{i-1}^{(i)} \end{pmatrix}, \tag{3}$$

from which we can write each $x_{i,j}$ as an affine function of $x_{i,i}$. By substituting these expressions of $x_{i,j}$ into the last equation of $\mathfrak{S}_i$, we obtain a quadratic equation in which the only unknown is $x_{i,i}$. Thus, we can solve it and determine $x_{i,i}$, after which we can compute all the values of $x_{i,j}$, with $j \neq i$.

Notice that, when $\mathbf{A}_{[i-1]}$ is singular, this requires some caution since (3) admits more solutions. We address this case later on and, for now, assume that all leading submatrices of $\mathbf{A}$ are non singular.

Since there are two potentially distinct solutions to a quadratic equation, we need to specify which one shall be considered. This can be encoded using only one bit. So, using also self-antiorthogonality equations, providing at most k bits instead of $k \log_2(q)$ is enough to encode all the elements $a_{1,1}, \ldots, a_{k,k}$.

Remark 3. For $q = 2$, each subsystem $\mathfrak{S}_i$ contains only linear equations since, for each $x \in \mathbb{F}_2$, $x^2 = x$. So, all the equations in $\mathfrak{S}_i$ are linear.

4.2 Exploiting the Reduced Row Echelon Form

We observe that, to solve each subsystem $\mathfrak{S}_i$, one of the most computationally expensive operations consists in solving (3). Indeed, using Gaussian elimination, this comes with cost $O\big((i-1)^3\big)$. Considering all $i \in \{1, \ldots, k\}$, we get an overall cost growing as

$$O\left(\sum_{i=1}^{k-1} i^3\right) = O\left(\frac{k(k-1)}{2}\right)^2 = O(k^4).$$

When k is linear in n, for instance, this results in $O(n^4)$ operations and constitutes the computational bottleneck of the whole technique.

We now describe how the computational cost can be reduced. The idea here is that the coefficient matrix $\mathbf{A}_{[i-1]}$, which we need to invert for $\mathfrak{L}_i$, has the matrix $\mathbf{A}_{[i-2]}$ as leading $(i-2) \times (i-2)$ submatrix, whose inverse has been computed previously (when solving $\mathfrak{S}_{i-1}$). Leveraging this fact, we can reduce significantly the computational cost.

We first observe that $\mathfrak{L}_i$ can be set up by considering the scalar products between $\mathbf{a}_i$ and any set of $i-1$ linearly independent vectors $\{\mathbf{m}_1^{(i)}, \ldots, \mathbf{m}_{i-1}^{(i)}\} \in \mathrm{Span}(\mathbf{a}_1, \ldots, \mathbf{a}_{i-1})$, i.e.,

$$\mathfrak{L}_i : \begin{cases} \langle \mathbf{m}_1^{(i)} \ ; \ \mathbf{a}_i \rangle = 0 \\ \langle \mathbf{m}_2^{(i)} \ ; \ \mathbf{a}_i \rangle = 0 \\ \quad \vdots \\ \langle \mathbf{m}_{i-1}^{(i)} \ ; \ \mathbf{a}_i \rangle = 0 \end{cases}$$

We can use vectors having a convenient form, say, we choose them so that, if stacked to form an $(i-1) \times (n-k)$ matrix, the leftmost $i-1$ columns form an identity matrix. This would lead to

$$\mathfrak{L}_i : \begin{cases} x_{i,1} = -m_{1,i}^{(i)} \cdot x_{i,i} - \sum_{j=i+1}^{n} m_{1,j}^{(i)} \cdot a_{i,j} \\ x_{i,2} = -m_{2,i}^{(i)} \cdot x_{i,i} - \sum_{j=i+1}^{n} m_{2,j}^{(i)} \cdot a_{i,j} \\ \quad \vdots \\ x_{i,i-1} = -m_{i-1,i}^{(i)} \cdot x_{i,i} - \sum_{j=i+1}^{n} m_{i-1,j}^{(i)} \cdot a_{i,j} \end{cases} \tag{4}$$

Notice that these equations are exactly the same one would get from (3) with the same row operations that would pivot the first columns of $\mathbf{A}_{[i-1]}$. Moreover, the vectors $\mathbf{m}_1^{(i)}, \ldots, \mathbf{m}_{i-1}^{(i)}$ are exactly the same vectors we would obtain during computation of the RREF of $\mathbf{A}_{[i-1,i]}$.

The computational advantage we get may not seem obvious. Indeed, to compute the vectors $\mathbf{m}_1^{(i)}, \ldots, \mathbf{m}_{i-1}^{(i)}$ from scratch, we would need to do Gaussian elimination over $\mathbf{A}_{[i-1]}$. However, the computation of these vectors can be inherited from the previous step. This can be understood by considering which vectors $\mathbf{m}_1^{(i+1)}, \ldots, \mathbf{m}_i^{(i+1)}$ shall be used to set up $\mathfrak{L}_{i+1}$. We need that these vectors span the same space generated by the first i rows of $\mathbf{A}$ and, as before, we would like them to have the same form as those we used in (4). This can be done with, at most, $2i$ sums between the vectors $\mathbf{m}_1^{(i)}, \ldots, \mathbf{m}_{i-1}^{(i)}$ we used in the previous step and $\mathbf{a}_i$. Let us collect all vectors $\mathbf{m}_j^{(i)}$ in a matrix $\mathbf{M}^{(i)}$, and the vectors $\mathbf{m}_j^{(i+1)}$ in a matrix $\mathbf{M}^{(i+1)}$; see Figure 2 for a visualization of these matrices. It is easy to see that $\mathbf{M}^{(i+1)}$ is obtained from $\mathbf{M}^{(i)}$ with at most $2i$ sums between (scalar multiples of) rows. Thus, once we know $\mathbf{M}^{(i)}$, getting $\mathbf{M}^{(i+1)}$ requires at most $4i(n-k)$ sums and multiplications over $\mathbb{F}_q$.

Once we have the vectors $\mathbf{m}_1^{(i)}, \ldots, \mathbf{m}_{i-1}^{(i)}$, we need to set up $\mathfrak{L}_i$ which, in practice, consists only in computing the right-hand terms in the equations in eq. (4). For each equation, this requires $1 + (n-i)$ multiplications and $n - i - 1$ sums, i.e., a total of $2(n-i)$ operations sums and multiplications over $\mathbb{F}_q$. Considering all equations, we get an overall cost of $2(i-1)(n-i)$.

Summing over all steps, we get an overall cost of

$$\sum_{i=1}^{k} 4(i-1)(n-k) + 2(i-1)(n-i) = \frac{1}{3}k(k-1)(9n-8k-2)$$

operations. When k is linear in n, this grows as $O(n^3)$.

Remark 4. We observe that, in practice, one can save some more operations. Indeed, for $j \in \{1, \ldots, i-2\}$ each $\mathbf{m}_j^{(i)}$ is 0 in the first $i-2$ entries but 1 in j, moreover we already know that the sums between $\mathbf{a}_i$ and each $\mathbf{m}_j^{(i)}$ will produce a 0 in the first $i-1$ positions. Leveraging this fact, we can actually do sums considering only the rightmost $n-(i-2)$ entries of each row. We can make analogous considerations when when pivoting the i-th column. All in all, the

$$\begin{pmatrix}\mathbf{M}^{(i)}\\ \mathbf{a}_i\end{pmatrix} = \left(\begin{array}{ccccc|cccc}
\multicolumn{5}{c|}{\text{RREF of } \mathbf{A}_{[i-1,i]}} & & & & \\
\hline
1 & 0 & \dots & 0 & m^{(i)}_{1,i} & m^{(i)}_{1,i+1} & m^{(i)}_{1,i+2} & \dots & m^{(i)}_{1,n-k} \\
0 & 1 & \dots & 0 & m^{(i)}_{2,i} & m^{(i)}_{2,i+1} & m^{(i)}_{2,i+2} & \dots & m^{(i)}_{2,n-k} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & m^{(i)}_{i-1,i} & m^{(i)}_{i-1,i+1} & m^{(i)}_{i-1,i+2} & \dots & m^{(i)}_{i-1,n-k} \\
\hline
a_{i,1} & a_{i,2} & \cdots & a_{i,i-1} & a_{i,i} & a_{i,i+1} & a_{i,i+2} & \dots & a_{i,n-k}
\end{array}\right)$$

$$\mathbf{M}^{(i+1)} = \left(\begin{array}{cccccc|ccc}
\multicolumn{6}{c|}{\text{RREF of } \mathbf{A}_{[i,i+1]}} & & & \\
\hline
1 & 0 & \dots & 0 & 0 & m^{(i+1)}_{1,i+1} & m^{(i+1)}_{1,i+2} & \dots & m^{(i+1)}_{1,n-k} \\
0 & 1 & \dots & 0 & 0 & m^{(i+1)}_{2,i+1} & m^{(i+1)}_{2,i+2} & \dots & m^{(i+1)}_{2,n-k} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & 0 & m^{(i+1)}_{i-1,i+1} & m^{(i+1)}_{i-1,i+2} & \dots & m^{(i+1)}_{i-1,n-k} \\
0 & 0 & \cdots & 0 & 1 & m^{(i+1)}_{i,i+1} & m^{(i+1)}_{i,i+2} & \dots & m^{(i+1)}_{i,n-k}
\end{array}\right)$$

Fig. 2: Elements of $\mathbf{M}^{(i)}$ and $\mathbf{M}^{(i+1)}$ and relation with leading submatrices of $\mathbf{A}$.

cost to compute $\mathbf{M}^{(i)}$, in the i-th step, can be estimated as

$$2(i-2)(n-k-(i-2)) + 2(i-2)(n-k-(i-1))$$

That is slightly smaller than

$$4(i-2)(n-k-(i-2))$$

Summing over all steps, we get an overall cost of

$$\sum_{i=1}^{k} 4(i-2)(n-k-(i-2)) + 2(i-1)(n-i)$$

This is clearly a lower cost than the previous one. In particular, this allows to avoid a number of operations equal to

$$\frac{1}{3}\big(12kn + k(k-1)(4k-26)\big).$$

This means that we are achieving a reduction of approximately 40% in the number of operations.

4.3 Dealing with Rank Deficiency

It may happen that at the i-th step the matrix $\mathbf{A}_{[i-1,i]}$ has rank $z_i < i-1$. This means that the RREF of $\mathbf{A}_{[i-1,i]}$ leads to a matrix with z_i pivoted columns.

Let $D_i = \{d_1, \dots, d_{z_i}\} \subset \{1, \dots, i\}$, with $|D_i| = z_i$, be the set that indexes the pivoted columns. Then, we just need to communicate the values of the unknowns indexed by F_i, that is the set containing the first $i-z_i-1$ values of $\{1, \ldots, i\}\backslash D_i$. Then there exists a unique index $t \in \{1, \ldots, i\}$ that is neither in D_i, nor in F_i, i.e., $t = \max\left\{\{1, \ldots, i\}\backslash D_i\right\}$. Knowing the values of the unknowns indexed by F_i is enough to write each of the unknowns indexed by D_i as an affine function of $x_{i,t}$.

Indeed, after substitution of each a_{i,d_j} into x_{i,d_j}, the system $\mathfrak{L}_i$ now looks as

$$\mathfrak{L}_i : \begin{cases} x_{i,d_1} = -m^{(i)}_{1,t} \cdot x_{i,t} - \sum_{f \in F_i} m^{(i)}_{d_1,f} \cdot a_{i,f} - \sum_{j=i+1}^{n} m^{(i)}_{d_1,j} \cdot a_{i,j} \\ x_{i,d_2} = -m^{(i)}_{1,t} \cdot x_{i,t} - \sum_{f \in F_i} m^{(i)}_{d_2,f} \cdot a_{i,f} - \sum_{j=i+1}^{n} m^{(i)}_{d_2,j} \cdot a_{i,j} \\ \quad \vdots \\ x_{i,d_{z_i}} = -m^{(i)}_{d_{z_i},t} \cdot x_{i,t} - \sum_{f \in F_i} m^{(i)}_{d_{z_i},f} \cdot a_{i,f} - \sum_{j=i+1}^{n} m^{(i)}_{d_{z_i},j} \cdot a_{i,j} \end{cases}$$

We remark that the sets F_i and D_i can be defined uniquely starting from $\mathbf{A}_{[i-1,i]}$, hence we need to communicate only the values of the entries indexed by F_i and not the actual set. This is because the compression and decompression algorithms, that we will show below, are designed to perform the operations in such a way that F_i can be determined uniquely during each of the two procedures. Then, this requires to communicate only the values of the entries indexed by F_i; thus, the number of bits we need to communicate is

$$(i - 1 - z_i) \cdot \log_2(q).$$

As we shall see later on, in practice one has that $z_i = i - 1$ in the majority of cases since the matrices have full rank with large probability: when this happens, we do not need to communicate any entry.

Remark 5. Whenever we do not have rank deficiencies, the description of the procedure corresponds with what we have described in the previous description. Indeed, when the first $i-1$ columns of $\mathbf{A}_{[i-1,i]}$ are linearly independent, we have $z_i = i - 1$ and $D_i = \{1, \ldots, i-1\}$. Notice that in this case we do not need to communicate extra variables. We thus have $\{1, \ldots, i\}\backslash D_i = \{i\}$ and $t = i$: we write each unknown $x_{i,j}$, for $j \in \{1, \ldots, i-1\}$, as an affine function of $x_{i,i}$.

4.4 Compression and decompression algorithms

Putting all the techniques from this section together, we obtain algorithms 4 and 5, which can be thought of as optimized versions of the algorithms presented in Section 3.1.

Algorithm 4: Compress*

```
Input: $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$ such that $\mathbf{A}\mathbf{A}^\top = -\mathbf{I}_k$
Output: $\mathbf{V} \in \mathbb{F}_q^{k\times(n-k)}$ such that $v_{i,j} = 0$ for each $i \leqslant j$, string of symbols
        extra_vars from $\mathbb{F}_q$, binary string $\mathbf{b} \in \{0\ ;\ 1\}^k$
1  Set extra_vars = [ ];// Empty list
2  Set $\mathbf{b} = (0,\ 0,\ \ldots,\ 0)$;// Null vector of length $k$
   /* Set $\mathbf{V}$ as upper triangular part of $\mathbf{A}$, except the main diagonal */
3  Set $\mathbf{V} = \mathsf{Triang}(\mathbf{A})$;
   /* Set up and solve self-antiorthogonality equation for $i = 1$ */
4  Find $x_1$, $x_2$: roots of equation $x^2 + \sum_{j=2}^{n-k} a_{1,j}^2 = -1$;
5  Set $b_i = \mathsf{ChooseBit}(x_1,\ x_2)$;
   /* Start looping over all subsystems for $i \geqslant 2$ */
6  for $i = 1,\ \ldots,\ k$ do
       /* Compute vectors for orthogonality equations; for $i = 2$, $\mathbf{M}^{(i-1)}$ is not defined
          and we just consider the RREF of $\mathbf{a}_{i-1}$ */
7      Compute $\mathbf{M}^{(i)}$ by pivoting the first $i$ columns of $\begin{pmatrix}\mathbf{M}^{(i-1)}\\ \mathbf{a}_{i-1}\end{pmatrix}$;
       /* Determine extra variables that need to be communicated */
8      Set $D_i \subseteq \{1,\ \ldots,\ i\}$ as the set of indices for pivoted columns in $\mathbf{M}^{(i)}$;
9      Set $F_i$ as the set formed by the first $i-1-|D_i|$ elements of $\{1,\ \ldots,\ i\}\backslash D_i$;
10     Append to extra_vars the entries of $\mathbf{a}_i$ indexed by $F_i$;
11     Set $t = \{1,\ \ldots,\ i\}\backslash(D_i \cup F_i)$;
       /* Express each unknown indexed by $D_i$ as an affine function of $x_t$ */
12     for $d \in D_i$ do
13         Set $x_d = -m_{d,t}^{(i)} \cdot x_t - \sum_{f\in F_i} m_{d,f}^{(i)} \cdot a_{i,f} - \sum_{j=i+1}^{n} m_{d,j}^{(i)} \cdot a_{i,j}$;
       /* Determine $i$-th entry of $\mathbf{b}$ by solving for $x_t$ */
14     Find $x_1$, $x_2$: values of $x_t$ satisfying
       ${x_t}^2 + \sum_{d\in D_i} {x_d}^2 + \sum_{f\in F_i} x_f^2 + \sum_{j=i+1}^{n} a_{i,j}^2 = -1$;
15     Set $b_i = \mathsf{ChooseBit}(x_1,\ x_2)$;
16 return $\mathbf{V}$, extra_vars, $\mathbf{b}$;
```

Observe that the algorithms assume that the self-orthogonal code always admits a systematic generator matrix, i.e., always admits $\{1,\ \ldots,\ k\}$ as information set. This implies that some codes may not be represented but, again, this is not a big deal as the wide majority of codes (heuristically, $\approx 1 - 1/q - 1/q^2$ of all codes) has $\{1,\ \ldots,\ k\}$ as an information set. The algorithm can easily be adapted to overcome this by permuting the columns and communicating the original position of the information set, but for the sake of simplicity we do not consider this tweak.

Algorithm 5: Decompress*

```
Input: $\mathbf{V} \in \mathbb{F}_q^{k\times(n-k)}$ such that $v_{i,j} = 0$ for each $i \leqslant j$, string of symbols
       extra_vars from $\mathbb{F}_q$, binary string $\mathbf{b} \in \{0\ ;\ 1\}^k$
Output: $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$ such that $\mathbf{A}\mathbf{A}^\top = -\mathbf{I}_k$

1  Set $\mathbf{A} = \mathbf{V}$;
   /* Set up and solve self-antiorthogonality equation for $i = 1$ */
2  Find $x_1,\ x_2$: roots of equation $x^2 + \sum_{j=2}^{n-k} a_{1,j}^2 = -1$;
3  Set $a_{1,1} = \mathsf{ChooseRoot}(x_1,\ x_2,\ b_1)$;
   /* Start looping over all subsystems for $i \geqslant 2$ */
4  Set $u = 0$;// Counter for required extra variables
5  for $i = 2,\ \ldots,\ k$ do
       /* Compute vectors for orthogonality equations; for $i = 2$, $\mathbf{M}^{(i-1)}$ is not defined
          and we just consider the RREF of $\mathbf{a}_{i-1}$ */
6      Compute $\mathbf{M}^{(i)}$ by pivoting the first $i$ columns of $\begin{pmatrix}\mathbf{M}^{(i-1)}\\ \mathbf{a}_{i-1}\end{pmatrix}$;
       /* Determine extra variables that need to be communicated */
7      Set $D_i \subseteq \{1,\ \ldots,\ i\}$ as the set of indices for pivoted columns in $\mathbf{M}^{(i)}$;
8      Set $F_i$ as the set formed by the first $i - 1 - |D_i|$ elements of $\{1,\ \ldots,\ i\}\backslash D_i$;
       /* Assign extra variables to entries of $\mathbf{a}_i$ indexed by $F_i$ */
9      for $j = 1, \ldots, |F_i|$ do
10         Set $f$ as the $j$-th entry of $D_i$;
11         Read entry $u + j$ from extra_vars, assign it to $a_{i,f}$;
12     Update $u \leftarrow u + |F_i|$;
       /* Express each unknown indexed by $D_i$ as an affine function of $x_t$ */
13     Set $t = \{1,\ \ldots,\ i\}\backslash(D_i \cup F_i)$;
14     for $d \in D_i$ do
15         Set $x_d = -m_{d,t}^{(i)} \cdot x_t - \underbrace{\sum_{f\in F_i} m_{d,f}^{(i)} \cdot a_{i,f} - \sum_{j=i+1}^{n} m_{d,j}^{(i)} \cdot a_{i,j}}_{\alpha_d}$;
       /* Determine $x_t$ */
16     Find $x_1,\ x_2$: values of $x_t$ satisfying
       ${x_t}^2 + \sum_{d\in D_i} {x_d}^2 + \sum_{f\in F_i} x_f^2 + \sum_{j=i+1}^{n} a_{i,j}^2 = -1$;
17     Set $a_{i,t} = \mathsf{ChooseRoot}(x_1,\ x_2,\ b_i)$;
       /* Determine unknowns indexed by $D_i$ */
18     for $d \in D$ do
19         Set $a_{i,d} = -m_{d,t}^{(i)} \cdot x_t - \alpha_d$;
20 return $\mathbf{A}$;
```

As subroutines, we use the following functions:

- Triang: on input a $k \times (n-k)$ matrix, returns the upper triangular part of $\mathbf{A}$ except the main diagonal;
- ChooseBit: on input three finite field elements x_1, x_2 and a, returns 0 if a is equal to the minimum between x_1 and x_2 (assuming some lexicographical ordering for $\mathbb{F}_q$), 1 otherwise;
- ChooseRoot: on input two finite field elements x_1 and x_2 and a bit, returns the minimum between the two elements if the bit is 0, the maximum otherwise.

We observe that the compression and decompression algorithms are quite similar since, in the end, many operations are performed in the same way in each of the two procedures.

As we have already shown before, the time complexity of the algorithm in the worst case grows as $O(n^3)$. For what concerns the communication cost, that is, the cost in terms of the data needed to be transmitted to fully describe a code compressed through the procedure described, instead, the analysis is not trivial, as we need to study how many extra variables must be sent, on average. We do this in the remainder of this section.

4.5 Communication cost

The cost of communicating an antiorthogonal matrix $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$ can be characterized as the cost of communicating three different parts that compose it, i.e. the lower-triangular and upper-triangular parts of the leftmost k-dimensional square submatrix $\mathbf{L}$ and $\mathbf{U}$, and the rightmost $k \times (n-2k)$ submatrix $\mathbf{R}$; see Figure 3 to visualize these matrices and their sizes.

The compression algorithm we propose avoids the need to include in the code representation the $\frac{1}{2}k(k+1)$ elements of $\mathbf{L}$, making it sufficient, in order to recover $\mathbf{L}$, to know the k bits needed to solve the k quadratic self-antiorthogonality equations, and a number of additional variables needed to solve the non-uniquely

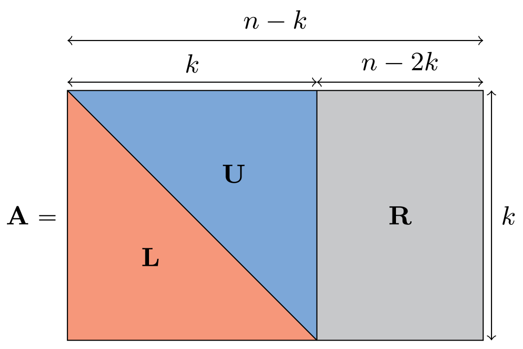


Fig. 3: Antiorthogonal matrix composition into matrices $\mathbf{L}$, $\mathbf{U}$ and $\mathbf{R}$.

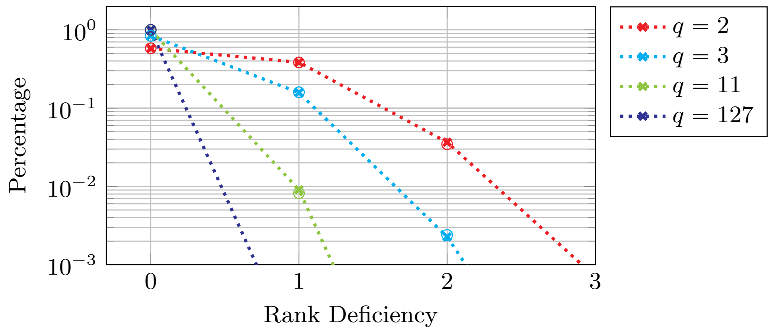


Fig. 4: Comparison of theoretical rank deficiency distribution for randomly sampled matrices and empirical rank deficiency distribution for submatrices of antiorthogonal matrices (10^4 submatrices sampled). Dotted curves represent theoretical values while circles represent empirical ones.

solvable subsystems encountered during the decompression process. It is easy to see that the number of additional variables is equal to the sum of the rank deficiencies of the encountered subsystems. In order to evaluate the expected output size of the compression algorithm, the rank distribution of such submatrices must be assessed. Deriving a precise estimate is a complex task, as it is difficult to characterize the effect of the relations that stem from the antiorthogonality of $\mathbf{A}$ on the rank of its submatrices. We do this by, heuristically, assuming that such matrices behave as random.

First, we recall [14] which counts the number of $k \times n$ matrices with rank z:

$$\left|\left\{\mathbf{T} \in \mathbb{F}_q^{k\times n} | \mathrm{rk}(\mathbf{T}) = z\right\}\right| = \begin{bmatrix} n \\ z \end{bmatrix}_q q \cdot \prod_{i=0}^{z-1}(q^k - q^i). \tag{5}$$

Then, Proposition 1 then follows naturally.

Proposition 1 *If $\mathbf{T} \in \mathbb{F}_q^{k\times n}$ is sampled uniformly at random, then*

$$\Pr[\mathrm{rk}(\mathbf{T}) = z] = \frac{\left[\begin{smallmatrix} n \\ z \end{smallmatrix}\right]_q \cdot \prod_{i=0}^{z-1}(q^k - q^i)}{q^{k\cdot n}}.$$

To evaluate the communication cost of the compression algorithm, we rely on the following heuristic argument, which has been validated through numerical simulations, as shown in fig. 4:

Heuristic 1 *If $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$ is antiorthogonal, then for any $\ell \leqslant k$, $m \leqslant n-k$,*

$$\Pr[\mathrm{rk}(\mathbf{A}_{[\ell,m]}) = z] = \Pr[\mathrm{rk}(\mathbf{T}) = z],$$

where $\mathbf{T} \in \mathbb{F}_q^{\ell\times m}$ is sampled uniformly at random.

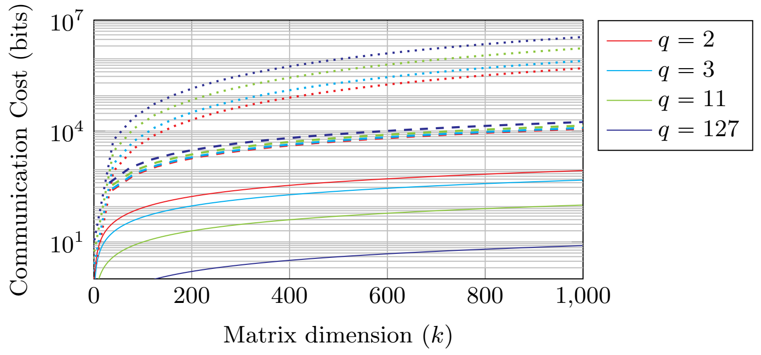


Fig. 5: Communication cost for sending $\mathbf{L}$ (dotted curves) compared with c_{add} (solid curves) and the worst case communication cost for algorithm 2 at fixed rate $\frac{1}{2}$ (dashed curves), for different values of k and q.

Let us denote with c_{add} the communication cost of the compression algorithm excluding the entries of $\mathbf{U}$ and $\mathbf{R}$. As stated previously, to recover the i-th row at the i-th step, the decompression algorithm requires 1 bit plus a number of variables equal to the rank deficiency of $\mathbf{A}_{[i-1,i]}$. Relying on Heuristic 1, denoting with $c_{\mathsf{add}}^{(i)}$ the communication cost for the i-th step, and with $\mathbf{M}^{(i)}$ the coefficient matrix at step i, as computed in algorithm 5, we can compute the expected value of c as follows:

$$\begin{aligned}
\mathbb{E}[c_{\mathsf{add}}] &= \sum_{i=1}^{k} \mathbb{E}\left[c_{\mathsf{add}}^{(i)}\right] \\
&= \sum_{i=1}^{k} 1 + \left(i - 1 - \mathbb{E}\left[\mathrm{rk}(\mathbf{M}^{(i)})\right]\right) \log_2(q) \\
&= k + \left(\sum_{i=2}^{k} \sum_{j=0}^{i-2} \Pr\left[\mathrm{rk}(\mathbf{M}^{(i)}) = j\right] (i-1-j)\right) \log_2(q). \\
&= k + \left(\sum_{i=2}^{k} \sum_{j=0}^{i-2} \frac{\begin{bmatrix} i \\ j \end{bmatrix}_q \cdot \prod_{t=0}^{j-1} (q^{(i-1)} - q^t)}{q^{(i-1)\cdot i}} (i-1-j)\right) \log_2(q). \qquad (6)
\end{aligned}$$

In fig. 5 the communication costs corresponding to sending the entries of $\mathbf{L}$ and to sending $\mathbb{E}[c_{\mathsf{add}}]$ are compared, showing a considerable cost decrease when the compression algorithm is used, especially for high values of k and q. In the same figure, the worst case communication cost for algorithm 2 is reported as well, highlighting how the improved algorithm is more efficient, particularly as q increases.

5 Applications to cryptography

We now describe how the techniques described in the previous sections can bring benefits when applied to some post-quantum cryptographic schemes based on the code equivalence problem. As concrete applications, we focus on LESS [7], SPECK [2] and the ABL scheme [1].

5.1 LESS

LESS is a signature scheme built from a Zero Knowledge (ZK) interactive proof for the code equivalence problem. In particular, LESS is based on LEP, which asks to find a monomial map $\mu \in M_n$, where M_n is the group of monomial transformations (transformations that permute and scale coordinates) between two linear codes. Observe that PEP is a special case of LEP, since a permutation is a special monomial transformation with all scaling coefficients set to 1.

In LESS, the public key is a set of codes $(\mathscr{C}_0, \mathscr{C}_1, \ldots, \mathscr{C}_{s-1})$ such that, for each $i \in \{1, \ldots, s-1\}$, it holds that

$$\mathscr{C}_i = \mu_i(\mathscr{C}_0), \quad \mu_i \in M_n.$$

The monomial maps $\mu_1, \ldots, \mu_{s-1}$ constitute the secret key. The codes in the public key are generated as follows:

- the code $\mathscr{C}_0$ is random code and is described by a generator matrix $\mathbf{G} = (\mathbf{I}_k, \mathbf{A})$; $\mathbf{A}$ is sampled using a PRNG fed with $\mathtt{seed} \xleftarrow{\$} \{0; 1\}^\lambda$;
- each code $\mathscr{C}_i$ is described by the standard form of $\mu_i(\mathbf{G})$. This requires to indicate the positions of the identity matrix columns (which takes $\log_2 \binom{n}{k}$ bits), as well as the values of the other columns (which require $k(n-k)\log_2(q)$ bits).

The resulting public key size is

$$|\mathsf{pk}| = \lambda + (s-1) \cdot \left(k(n-k)\log_2(q) + \log_2 \binom{n}{k}\right). \tag{7}$$

Observe that we are not taking the ceiling of logarithms, thus numbers may differ when comparing public key sizes reported elsewhere.

LESS based on PEP and self-orthogonal codes Without modifying the structure of LESS at a protocol level, one can rely on PEP instead of LEP. All relevant properties (ZK, correctness and soundness) would be trivially preserved, since permutations are a special case of monomial maps. Actually, the protocol would also get simplified as, for instance, the computation of canonical forms becomes easier [9].

To withstand attacks based on the hull of the code, such as the Support Splitting Algorithm [15] and the reduction to graph isomorphism [3], one needs to use codes with large hull. Using self-orthogonal codes, which have the maximum hull size, maximizes the cost of such attacks. In this case, the fastest attacks remain those based on canonical forms [9] and low-weight codewords [6], whose running time is the same regardless of the hull dimension. In other words, switching to PEP and self-orthogonal codes does not require any change either in the code parameters or in the security achieved by the scheme. On the other hand, this allows us to exploit the techniques described above to reduce the public key size, as they can be used to compress any code in the public key.

The code $\mathscr{C}_0$ can be represented using a seed: feeding a PRNG with such a seed, one samples $\mathscr{C}_0$ from $\mathfrak{C}_{n,k,q}$. For all the other codes in the public key, the techniques in this paper can be used to reduce the amount of bits required to represent each of them. By doing this, without modifying the code parameters, we can achieve approximately a halving of the size of public keys. In Table 2 we show the resulting public key size. With this modification, verification would start by recreating the codes in the public key. All the extra operations would take time $O(n^3)$, which matches the cost of Gaussian elimination, that represents the computational bottleneck in LESS. Consequently, we expect that this modification does not impact too much on the running times of LESS.

Table 2: Public key size reduction for the modified version of LESS based on self-orthogonal codes and PEP, using the techniques in this paper for compressing the public key size.

		pk size (kB)		
	Number of codes in pk	Original	This paper (worst case)	This paper (improved)
NIST Cat. 1 $n = 252$, $k = 126$	2	13.69	6.99	6.82
	4	40.74	20.93	20.44
	8	95.04	48.82	47.68
NIST Cat. 3 $n = 400$, $k = 200$	2	34.20	17.42	17.15
	4	102.54	52.21	51.39
NIST Cat. 5 $n = 548$, $k = 274$	2	64.15	32.53	32.14
	4	192.37	97.52	96.35

5.2 SPECK

SPECK is a recently proposed signature scheme whose security is based on PEP [2]. The public key is composed of a pair of permutation equivalent codes $\mathscr{C}, \mathscr{C}' \subseteq \mathbb{F}_q^n$ such that $\mathscr{C}' = \pi(\mathscr{C})$ for a $\pi \in S_n$, where π is the secret key.

SPECK is based on a protocol proving knowledge about a permutation sending a random codeword of $\mathscr{C}$ into $\mathscr{C}'$. Modulo some optimizations, the protocol in [2] works as follows: the prover first commits to $\mathbf{c}^* = \tau(\mathbf{c})$, with $\tau \xleftarrow{\$} S_n$ and

$\mathbf{c} \xleftarrow{\$} \mathscr{C}$, and then either shows that the commitment has been prepared honestly or discloses the permutation $\tau^{-1} \circ \pi$, which maps $\mathbf{c}^*$ to $\mathscr{C}'$.

With respect to LESS, SPECK has larger signatures but is much faster, because it does not require Gaussian elimination, which is required in LESS to verify the commitment. Indeed, regardless of the challenge, verification in SPECK just requires to perform a matrix-vector multiplication. Since SPECK requires permutation equivalent codes, one can exploit the techniques presented in the previous sections to compress the representation of $\mathscr{C}'$. By doing this, the public key size is reduced by about half again, as shown in Table 3.

Table 3: Public key size reduction for SPECK using the techniques in this paper for compressing the public key size.

		pk size (kB)		
	Number of codes in pk	Original	This paper (worst case)	This paper (improved)
NIST Cat. 1 $n = 252$, $k = 126$, $q = 127$	2	13.69	6.99	6.82
NIST Cat. 1 $n = 252$, $k = 126$, $q = 8861$	2	25.46	12.97	12.71

5.3 ABL

ABL is a lattice-based updatable public key encryption scheme which makes use of q-ary lattices. Security depends on LWE and a special version of LIP, sometimes called Signed PEP, in which the monomial transformation has only ± 1 as scalar coefficients. As shown in [1], all PEP attacks based on the hull work also for Signed PEP, thus the ABL scheme requires codes with large hull.

The instances recommended in [1] rely on weakly self-dual codes and have a hull dimension large enough to make all existing attacks running in time at least 2^λ. These instances, however, have been attacked in [5], exploiting the fact that whenever the hull dimension is less than $\sqrt{2n}$, then the squares of the hulls give a new PEP instance where the hull is trivial with large probability.

The attack in [5] can be easily avoided by increasing the hull dimension, with respect to the values recommended in [1]. For instance, using self-orthogonal codes, one can avoid all attacks based on the hull and, at the same time, unlock the possibility of compressing public keys using the techniques introduced in this paper. The public key for ABL is composed by the non-systematic part of a generator matrix (in systematic form), plus a vector of size n, both with elements in $\mathbb{F}_q$. Hence, the total resulting public key size is

$$|\mathsf{pk}| = (k(n-k) + n)\log_2(q). \tag{8}$$

In Table 4 we report the resulting instances and show that, although to a lesser extent than for LESS and SPECK, the techniques we have introduced also reduce the size of public keys for this system.

Table 4: Public key size reduction for the modified version of ABL based on self-orthogonal codes, using the techniques in this paper for compressing the public key size.

	pk size (kB)		
	Original	This paper (worst case)	This paper (improved)
$\lambda = 128$ $n = 7313$, $k = 450$, $q \approx 2^{13}$	4901.96	4742.63	4741.14
$\lambda = 128$ $n = 11000$, $k = 550$, $q \approx 2^{16}$	11227.04	10933.45	10931.36
$\lambda = 192$ $n = 20250$, $k = 900$, $q \approx 2^{18}$	38268.01	37381.31	37377.34
$\lambda = 256$ $n = 29688$, $k = 1250$, $q \approx 2^{19}$	82450.41	80643.04	80637.28

6 Conclusion

We have proposed a method to represent self-orthogonal codes in a compact way. We have first described a basic approach, based solely on the pairwise orthogonality of codewords. We have shown that the resulting compression algorithm yields an asymptotically optimal representation (according to Shannon's source coding theorem), while enabling efficient compression and decompression procedures. We have then presented a refined approach, which improves on the first one by using self-antiorthogonality equations and by dealing with rank deficiencies in a more efficient way, showing that such a refined algorithm reduces slightly the communication cost. Moreover, we have described how the compression and decompression techniques can be implemented in an efficient manner, achieving complexity $O(n^3)$. As examples of application, we have shown how the techniques we have introduced can be successfully exploited to reduce the public key size of cryptographic schemes such as LESS, SPECK and ABL. By applying our techniques, we were able to nearly halve the public key size of the former two schemes, and to produce a reduction in the size of public keys of the latter as well.

Acknowledgments

This work was partially supported by the Italian Ministry of University and Research (MUR) under the Projects of Significant National Interest (PRIN) 2022 call with the project "Mathematical Primitives for Post Quantum Digital Signatures" (CUP I53D23006580001), by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan, funded by the European Union - Next Generation EU, and by MUR under the Italian Fund for Applied Science (FISA) 2022, Call for tender No. 1405 published on 13/09/2022 - project title "Quantum-safe cryptographic tools for the protection of national data and information technology assets" (QSAFEIT) - No. FISA 2022-00618 (CUP I33C24000520001), Grant Assignment Decree no. 15461 adopted on 02/08/2024.

References

[1] M. R. Albrecht, B. Benčina, and R. W. Lai. "Hollow LWE: A New Spin: Unbounded Updatable Encryption from LWE and PCE". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2025, pp. 363–392.

[2] M. Baldi, M. Battagliola, R. El Mechri, P. Santini, R. Schiavoni, and D. De Zuane. "SPECK: Signatures from Permutation Equivalence of Codes and Kernels". In: *Cryptology ePrint Archive* (2025).

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

[4] D. Bartoli, M. Montanucci, and G. Zini. "On certain self-orthogonal AG codes with applications to Quantum error-correcting codes". In: *Des. Codes Cryptography* 89.6 (June 2021), pp. 1221–1239. DOI: 10.1007/s10623-021-00870-y. URL: https://doi.org/10.1007/s10623-021-00870-y.

[5] M. Battagliola, R. Mora, and P. Santini. "Using the Schur Product to Solve the Code Equivalence Problem". In: *Cryptology ePrint Archive* (2025).

[6] W. Beullens. "Not enough LESS: an improved algorithm for solving code equivalence problems over F q". In: *International Conference on Selected Areas in Cryptography*. Springer. 2020, pp. 387–403.

[7] J.-F. Biasse, G. Micheli, E. Persichetti, and P. Santini. "LESS is more: code-based signatures without syndromes". In: *International Conference on Cryptology in Africa*. Springer. 2020, pp. 45–65.

[8] A. R. Calderbank and P. W. Shor. "Good quantum error-correcting codes exist". In: *Phys. Rev. A* 54 (2 1996), pp. 1098–1105. DOI: 10.1103/PhysRevA.54.1098. URL: https://link.aps.org/doi/10.1103/PhysRevA.54.1098.

[9] T. Chou, E. Persichetti, and P. Santini. "On linear equivalence, canonical forms, and digital signatures". In: *Designs, Codes and Cryptography* (2025), pp. 1–43.

[10] A. Di Giusto and A. Ravagnani. "The asymptotic number of equivalence classes of linear codes with given dimension". In: *arXiv preprint arXiv:2510.14424* (2025).

[11] J. Fulman and L. Goldstein. "Stein's method and the rank distribution of random matrices over finite fields". In: *The Annals of Probability* (2015), pp. 1274–1314.

[12] X. Li and Z. Heng. "Self-Orthogonal Codes From p-Divisible Codes". In: *IEEE Transactions on Information Theory* 70.12 (2024), pp. 8562–8586. DOI: 10.1109/TIT.2024.3449921.

[13] J. L. Massey. "Orthogonal, antiorthogonal and self-orthogonal matrices and their codes". In: *Communications and coding* 2.3 (1998).

[14] K. Morrison. "Integer sequences and matrices over finite fields". In: *Journal of Integer Sequences* 9 (June 2006).

[15] N. Sendrier. "Finding the permutation between equivalent linear codes: The support splitting algorithm". In: *IEEE Transactions on Information Theory* 46.4 (2002), pp. 1193–1203.

[16] N. Sendrier. "On the dimension of the hull". In: *SIAM Journal on Discrete Mathematics* 10.2 (1997), pp. 282–293.

[17] A. Steane. "Multiple-particle interference and quantum error correction". In: *Proc. R. Soc. Lond. A* 452 (1954 1996), pp. 2551–2577. DOI: 10.1098/rspa.1996.0136.

[18] T. Yasusa, X. Dahan, Y.-J. Huang, T. Takagi, and K. Sakurai. "MQ Challenge: Hardness Evaluation of Solving MQ problems". In: *Proc. Workshop on Cybersecurity in a Post-Quantum World.* NIST. Gaithersburg, Maryalnd, Apr. 2015.

A Proof of Theorem 5

We first introduce the following well known constant

$$\zeta_q = \lim_{n\to\infty} \prod_{i=1}^{n} 1 - q^{-i}.$$

The value ζ_q corresponds to the probability that a random $n \times n$ matrix over $\mathbb{F}_q$, for n growing to infinity, is non singular: ζ_q is a constant lower bounded by $1 - 1/q - 1/q^2$ (see e.g. [11, Lemma 2.3]). First, we prove that γ_q is a constant.

Lemma 1 *For every $q \geqslant 2$ with q a prime power, γ_q is a positive constant, i.e.,*

$$\gamma_q = \lim_{n\to\infty} \prod_{i=1}^{n} \frac{1 - q^{-i}}{1 - q^{-2i}} \in \mathbb{R}_{\geqslant 0}.$$

Proof. We do this by showing that γ_q is bounded, both from below and from above, by a constant. We first derive the upper bound. To this end, we rewrite γ_q as

$$\prod_{i=1}^{\infty} q^i \cdot \frac{q^i - 1}{q^{2^i} - 1}.$$

For every $q > 1$ and every $i \geqslant 1$, it holds that $\frac{q^i-1}{q^{2i}-1} \leqslant \frac{q^i}{q^{2i}} = q^{-i}$ Then

$$\prod_{i=1}^{\infty} q^i \cdot \frac{q^i-1}{q^{2^i-1}} \leqslant \prod_{i=1}^{\infty} q^i \cdot q^{-i} = 1.$$

We now show that γ_q is also lower bounded by a constant. To this end, is it enough to argue that $\prod_{i=1}^{n} \frac{1-q^{-i}}{1-q^{-2i}} \geqslant \prod_{i=1}^{n} 1 - q^{-i}$ and, for $n \to \infty$, this corresponds to ζ_q. □

Now, we recall [10, Equation 10 and Corollary 4.1].

Lemma 2 *For constant q and every $0 \leqslant k \leqslant n$, it holds that*

$$1 \leqslant \frac{\left[{n \atop k}\right]_q}{q^{k(n-k)}} \leqslant \frac{1}{\prod_{i=1}^{\infty} 1 - q^{-i}}.$$

Moreover, for any k and for $n \to \infty$,

$$\begin{bmatrix} n \\ k \end{bmatrix}_q \sim \frac{1}{\prod_{i=1}^{k} 1 - q^{-i}} \cdot q^{k(n-k)}.$$

If also $k \to \infty$, then

$$\begin{bmatrix} n \\ k \end{bmatrix}_q \sim \frac{1}{\zeta_q} \cdot q^{k(n-k)}.$$

We recall Theorem 5 and have that, for growing k and n,

$$|\mathfrak{C}_{n,k,q}| \sim \gamma_q \cdot \begin{bmatrix} n \\ k \end{bmatrix}_q \cdot \frac{1}{\prod_{i=1}^{k} q^i - 1}.$$

We observe that

$$\frac{1}{\prod_{i=1}^{k} q^i - 1} = \frac{1}{\prod_{i=1}^{k} q^i \cdot (1 - q^{-i})} = \frac{q^{-k(k+1)/2}}{\prod_{i=1}^{k} 1 - q^{-i}}$$

which, for growing k, converges to $q^{-k(k+1)/2} \cdot \frac{1}{\zeta_q}$. Then, relying on Lemma 2, for growing k and n and putting everything together, we get

$$\begin{aligned} |\mathfrak{C}_{n,k,q}| &\sim \gamma_q \cdot \begin{bmatrix} n \\ k \end{bmatrix}_q \cdot \frac{1}{\prod_{i=1}^{k} q^i - 1} \\ &\sim \gamma_q \cdot \frac{q^{k(n-k)}}{\zeta_q} \cdot \frac{q^{-k\cdot(k+1)/2}}{\zeta_q} \\ &= q^{k(n-k)-\frac{k(k+1)}{2}} \cdot \underbrace{\frac{\gamma_q}{\zeta_q^2}}_{\omega_q}. \end{aligned}$$

B Proof of Theorem 6

To derive the worst case, we assume that all outputs are uniformly distributed over the corresponding domain. As it is well known, this is the worst case as the uniform distribution maximizes the entropy of the associated source.

First, the encoding of $\mathbf{V}$ has binary length

$$\left(k(n-k)-\frac{k(k-1)}{2}\right)\cdot\log_2(q)=\frac{k}{2}\cdot(2n-3k+1)\cdot\log_2(q). \tag{9}$$

Now, we just need to show that all the other terms contribute with lower order terms, so that they asymptotically vanish. We do this term by term.

- The set U is chosen out of $\binom{n}{k}$ possibilities. If $k=o(n)$, we have $\log_2\binom{n}{k}=k\cdot\log_2(k)\cdot\big(1+o(1)\big)$: if we divide by (9) and consider dominant terms, we obtain something that grows as $\log_2(n/k)/n<\log_2(n)/n$ which goes to 0 for growing n. If k is linear in n, we instead get $\log_2\binom{n}{k}=n\cdot h(k/n)\cdot\big(1+o(1)\big)$, where $h(x):=-x\log_2(x)-(1-x)\cdot\log_2(1-x)$ is the binary entropy function. Dividing by (9) and considering only dominant terms, we obtain a quantity that grows as $h(k/n)/k$: the numerator is constant while the denominator grows, hence this goes to 0.
- The permutation $\mathbf{P}$ belongs to S_{n-k} but has a special structure. In particular, the number of such permutations is

  $$\binom{n-k}{k}\cdot k!.$$

  Thus, taking its logarithm, we get

  $$\log_2\binom{n-k}{k}+\log_2(k!).$$

  To show that the first order term is vanishing, we can use arguments similar to the ones we used before. For the permutation part, we use Stirling's approximation, for growing k we have

  $$k!\sim\sqrt{2\pi k}\cdot\left(\frac{k}{e}\right)^k.$$

  If we divide $\log_2(k!)$ by the value in (9), we get

  $$\frac{\log_2(k!)}{\frac{k}{2}\cdot(2n-3k+1)\cdot\log_2(q)}\sim\frac{\log_2(k/e)+\frac{1}{2}\log_2(2\pi k)}{\frac{1}{2}\cdot(2n-3k+1)\cdot\log_2(q)}.$$

  The numerator grows as $\log_2(k)$ while the dominant term in the denominator is n. Since $\log_2(k)/n=o(1)$, the above quantity asymptotically goes to 0.

All in all, we have

$$L'_{n,k,q}=\frac{k}{2}\cdot(2n-3k+1)\cdot\log_2(q)\cdot\big(1+o(1)\big).$$

If we divide $L'_{n,k,q}$ by $L_{n,k,q}$, up to a multiplicative factor that converges to 1, we obtain $\frac{2n-3k+1}{2n-3k-1}$, which obviously converges to 1 for growing k and n.