Recursion Enabled: Improved Cryptanalysis of the Permuted Kernel Problem

Alessandro Budroni[1], Marco Defranceschi[1,2*], and Federico Pintore[2]

[1] Technology Innovation Institute, UAE
{alessandro.budroni,marco.defranceschi}@tii.ae
[2] University of Trento, Italy
federico.pintore@unitn.it

Abstract. The Permuted Kernel Problem (PKP) is a computational problem for linear codes over finite fields that has emerged as a promising hard problem for constructing post-quantum cryptographic schemes, with its main application found in the digital signature scheme PERK, submitted to the NIST standardization process for quantum-secure additional signatures. Upon reviewing the first version of PERK, NIST recommended further research on the concrete complexity of PKP. In this work, we follow this recommendation and investigate algorithmic improvements to the known methods for solving PKP. Specifically, we build upon the state-of-the-art work of Santini, Baldi, and Chiaraluce (IEEE Trans. Inf. Theory, 2024), and introduce a new algorithm that outperforms it over a wide range of parameters, yielding double-digit bit reductions in estimated complexity on representative instances. Nevertheless, our analysis shows that these improvements do not affect the parameter-set choices in PERK, thereby reinforcing confidence in its security.

Keywords: Permuted Kernel Problem · Code-based Cryptography · Post-quantum Cryptography · PERK

1 Introduction

Following Shor's seminal work [27], which introduced a quantum algorithm capable of solving the hard mathematical problems underlying much of classical cryptography in polynomial time, the cryptographic community began to seek alternative hardness assumptions that would withstand quantum adversaries. This effort gave birth to the field of Post-quantum Cryptography.

In 2017, the National Institute of Standards and Technology (NIST) launched a call for post-quantum schemes [18]. Three out of the five schemes selected

*This work was conducted during Marco Defranceschi's internship at the Technology Innovation Institute, Abu Dhabi.

for standardization are based on assumptions on algebraic lattices [25,17,21], reflecting the maturity reached by cryptosystems based on this family of problems. However, with the subsequent NIST "on-ramp" call for additional digital signature [19], NIST emphasized the importance of diversifying beyond lattices, encouraging the study of alternative quantum-resistant assumptions. In this context, the Permuted Kernel Problem (PKP), originally introduced by Shamir in 1989 [26] to construct an identification scheme, has resurfaced as a promising candidate for building digital signatures. Among the proposals submitted to the NIST "on-ramp" call for additional signatures, PERK [1] is a digital signature scheme built on the hardness of PKP. PERK successfully advanced through the first evaluation phase and is now in the second round of the NIST process for the selection of additional signatures to standardize.

Motivation and contribution. In its report on the first round of the additional signatures selection process, where candidates advancing to the second round are enumerated [3, Sec. 3.6], NIST explicitly highlighted the need for further research on the concrete complexity of PKP, underlining the importance of assessing the long-term security of the parameter choice in PERK. In this work, we take up this challenge and present new advances in the cryptanalysis of PKP, focusing on its concrete complexity. Specifically, we introduce a new algorithm that improves upon the state of the art across several parameter sets.

Related work. Shamir was the first one to construct a cryptographic scheme based on PKP [26]. More recently, a series of digital signature schemes relying on the hardness of PKP have been proposed [10,8,7,11,4]. However, the most prominent instantiation is represented by PERK [1]. Specifically, PERK version 1.0, submitted to the first round of the NIST process for additional signatures, relies on a variant of PKP that can nonetheless be solved by any algorithm solving the standard PKP. More recently, version 2.1 of PERK has been made public [2], this time relying on the pure PKP over a finite field of characteristic two.

Shamir was also the first one to provide some considerations on the security of PKP [26]. In particular, he observed that a reduction from the 3-Partition problem implies that the mono-dimensional version of PKP is NP-complete [12]. The NP-completeness of the multi-dimensional case was later proved in [6]. Since Shamir's original work, several works have examined the hardness of PKP [13,5,20,14,16], progressively refining the understanding of the computational complexity of the problem. More recently, Koussa, Macario-Rat and Patarin (KMP) [15] proposed a *meet-in-the-middle* algorithm, which was subsequently improved by Santini, Baldi, and Chiaraluce (SBC) [23,24] through the introduction of a preprocessing (*filtering*) step. The latter currently represents the state of the art in solving PKP.

1.1 Overview of the Contribution

The Permuted Kernel Problem. Informally, the Permuted Kernel Problem (PKP) consists in finding a permutation π such that

$$\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{0},$$

where $\boldsymbol{H} \in \mathbb{F}_q^{r\times n}$ and $\boldsymbol{V} \in \mathbb{F}_q^{\ell\times n}$ are public matrices. An equivalent inhomogeneous version of the problem, denoted by IPKP, asks for a permutation π satisfying

$$\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}, \tag{1}$$

where $\boldsymbol{E} \in \mathbb{F}_q^{\ell\times r}$ is also public. Typically, the parameters n, r, ℓ, and q are chosen so that the permutation π is unique with high probability.

Key contributions. In one of its subroutines, the SBC algorithm employs an Information Set Decoding (ISD) procedure to find a subcode of the code generated by $\boldsymbol{H}$ having dimension $d \leq r$ and small support $w \leq n$, that is, a subcode with a generator matrix with $n - w$ zero columns. The algorithm then performs a *meet-in-the-middle* style routine over this subcode to filter potential candidates for the permutation π.

Our first algorithmic improvement stems from the observation that, by disregarding the zero columns in the generator matrix of the subcode with support size w, one obtains a subinstance of IPKP where the corresponding public matrix $\bar{\boldsymbol{H}}$ has size $d \times w$. In other words, the *filtering* subroutine of SBC can be interpreted as solving smaller IPKP instances. This suggests that SBC could be applied recursively, since SBC solves IPKP (and not only PKP). However, this recursive approach is generally inefficient or inapplicable unless one considers another key observation of this work, that is, the dual form of Equation (1). Indeed, we notice that

$$\pi^{-1}(\boldsymbol{H})\boldsymbol{V}^\top = \boldsymbol{E}^\top,$$

can be seen as an instance of IPKP with secret permutation π^{-1} (the inverse of π), and we find that applying SBC recursively leads to improvements only when the subinstance is considered in its dual version. Moreover, enabling such recursion is non-trivial as the subinstance may admit no solution or multiple solutions. To address this, we revisited and extended SBC, introducing several subtle modifications to make recursive application feasible.

Building on the observations above, we introduce a new algorithm that either outperforms or matches SBC. Specifically, we find improvements for parameter sets where q is relatively small. Most importantly, depending on the code rate $R := r/n$, our algorithm is faster also for very small values of ℓ, that is the typical setting in cryptographic applications. As an example, for $(n, r, \ell, q) = (112, 54, 3, 16)$, our algorithm reduces the estimated complexity from 255 to 243 bits, corresponding to an improvement of 12 bits. Similarly, for $(n, r, \ell, q) = (118, 83, 1, 223)$, we observe a reduction from 199 to 189 bits, i.e., an improvement of 10 bits.

Implications for PERK. Despite significantly improving upon SBC for a wide range of parameters, our newly proposed algorithm does not affect the security estimates of either PERK 1.0 or PERK 2.1. In the former case, the parameter sets employ a very large field size q, which prevents our algorithmic improvements from being effective. In the latter, the combination of the code rate R and the choice $\ell = 1$ places the parameters in a regime where SBC remains the optimal solving algorithm.

Artifacts. We make our scripts used to compute the complexities publicly available [9].

Organization. In Section 2, we provide the necessary preliminaries and background. Section 3 reviews the KMP and SBC algorithms, serving as essential background for introducing our contribution in Section 4. Finally, we analyze our results and evaluate their impact on PERK in Section 5.

2 Preliminaries

Notation. Let $n \in \mathbb{N}^*$. We define $[n] := \{1, \dots, n\}$. If $I \subseteq [n]$, we denote by $\neg_n I$ its complementary $[n] \setminus I$. We use capital bold letters for matrices and lower-case bold letters for vectors. When the base field is clear from the context, the $n \times n$ identity matrix is denoted by $\boldsymbol{I}_n$. All vectors are row-vectors if not stated otherwise. Given a vector $\boldsymbol{v}$ we denote by $\boldsymbol{v}_i$ its i-th entry. Similarly, for an $m \times n$ matrix $\boldsymbol{A}$, with $m \in \mathbb{N}^*$, we denote its j-th column by $\boldsymbol{a}_j$ and its entry in the i-th row and j-th column by $\boldsymbol{a}_{i,j}$. Given a set of indices $I \subseteq [n]$, $\boldsymbol{v}_I$ is the projection of the length-n vector $\boldsymbol{v}$ onto the indices in I, i.e. $\boldsymbol{v}_I = (\boldsymbol{v}_i)_{i \in I}$. We extend this to matrices by letting $\boldsymbol{A}_I$ be the submatrix formed by the columns of $\boldsymbol{A}$ indexed by I. Similarly, we denote by $\boldsymbol{A}^J$ the submatrix obtained by taking the rows of $\boldsymbol{A}$ indexed by J, if $J \subseteq [m]$. The transpose of a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^\top$.
Let $\mathbb{F}_q$ be a finite field of size q, with q a prime power. We denote by $\mathsf{GL}_n(\mathbb{F}_q)$ the linear group of invertible $n \times n$ matrices over $\mathbb{F}_q$ and by $\mathsf{GL}_{k,n}(\mathbb{F}_q)$ the set of $k \times n$ (with $k \leq n$) matrices over $\mathbb{F}_q$ with full rank k.

We denote by $\mathsf{Sym}(n)$ the set of all permutations of $[n]$. For a permutation $\pi \in \mathsf{Sym}(n)$ and $\boldsymbol{A} \in \mathbb{F}_q^{m \times n}$, we write $\pi(\boldsymbol{A})$ to denote the matrix obtained by permuting the columns of $\boldsymbol{A}$ according to π, that is the matrix $\boldsymbol{AP}$ where $\boldsymbol{P} \in \mathbb{F}_q^{n \times n}$ is the permutation matrix associated to π.

Given a matrix $\boldsymbol{A} \in \mathsf{GL}_{k,n}(\mathbb{F}_q)$ we say that a set of indices $I \subseteq [n]$ with cardinality k is an information set for $\boldsymbol{A}$ if $\boldsymbol{A}_I$ is invertible. When this is the case, we denote by $\mathsf{RREF}_I(\boldsymbol{A})$ the *Row Reduced Echelon Form* of $\boldsymbol{A}$ w.r.t. to the set I, that is $\boldsymbol{A}_I^{-1}\boldsymbol{A}$. In addition, we denote by $\mathsf{RREF}^*{}_I(\boldsymbol{A})$ the pair composed of $\mathsf{RREF}_I(\boldsymbol{A})$ and the matrix $\boldsymbol{A}_I^{-1}$.

For a matrix $\boldsymbol{A}$ with n columns and a non-zero natural number $x \leq n$, we define $\mathcal{S}_x(\boldsymbol{A})$ as the set of all matrices with x columns, each of them picked from those of $\boldsymbol{A}$. It is easy to see that, if $\boldsymbol{A}$ has no repeated columns (we show in Section 4.2 that this might not be always the case), $|\mathcal{S}_x(\boldsymbol{A})| = \frac{n!}{(n-x)!}$. We call $\bar{\mathcal{S}}_x(\boldsymbol{A})$ the set obtained by identifying elements in $\mathcal{S}_x(\boldsymbol{A})$ up to column permutation, that is, the set of all the subsets of x columns of $\boldsymbol{A}$. Therefore, $|\bar{\mathcal{S}}_x(\boldsymbol{A})| = \frac{|\mathcal{S}_x(\boldsymbol{A})|}{x!} = \binom{n}{x}$.

With an abuse of notation, for two matrices $\boldsymbol{A} \in \mathbb{F}_q^{m \times n}$ and $\boldsymbol{B} \in \mathbb{F}_q^{n \times \ell}$ we say that $\boldsymbol{B} \in \mathsf{ker}(\boldsymbol{A})$ if $\boldsymbol{AB} = \boldsymbol{0}$.

Finally,we say that an event occurs with "high probability" if it happens with probability $1 - o(1)$.

Linear Codes. An $[n, k]$-linear code $\mathcal{C}$ over a finite field $\mathbb{F}_q$ is a subspace, of dimension k, of $\mathbb{F}_q^n$. $\mathcal{C}$ can be represented via a basis $\boldsymbol{G} \in \mathbb{F}_q^{k \times n}$, called *generator matrix*, or by a *parity-check matrix* $\boldsymbol{H} \in \mathbb{F}_q^{n-k \times n}$ satisfying $\boldsymbol{H}\boldsymbol{c}^\top = \boldsymbol{0} \Leftrightarrow \boldsymbol{c} \in \mathcal{C}$. The linear code generated by (the rows of) $\boldsymbol{H}$ is called the dual code of $\mathcal{C}$ and is denoted by $\mathcal{C}^\perp$. We refer to n as the *length* of $\mathcal{C}$ and to the quantity $R := k/n$ as the *code rate*. Given a codeword $\boldsymbol{c} \in \mathcal{C}$, we define its support as the set of indices of its nonzero entries, i.e. $\mathrm{Supp}(\boldsymbol{c}) := \{i \in [n] : \boldsymbol{c}_i \neq 0\}$. We also define the support of a linear code $\mathcal{C}$ as the union of the supports of its codewords, that is $\mathrm{Supp}(\mathcal{C}) := \{i \in [n] : \exists\ \boldsymbol{c} \in \mathcal{C} \text{ s.t. } \boldsymbol{c}_i \neq 0\}$. The Hamming weight $\mathrm{wt}(\boldsymbol{c}) := |\mathrm{Supp}(\boldsymbol{c})|$ of $\boldsymbol{c}$ is the size of its support.
A subspace $\mathcal{C}'$ of a linear code $\mathcal{C}$ over $\mathbb{F}_q$ is said to be a subcode of $\mathcal{C}$. If $\mathcal{C}$ has dimension k, the number of its subcodes having dimension $k' \leq k$ is given by $\begin{bmatrix} k \\ k' \end{bmatrix}_q = \prod_{i=0}^{k'-1} \frac{1-q^{k-1}}{1-q^{i+1}}$. In our complexity estimates, we make use of the approximation used in [24], i.e.

$$\frac{\begin{bmatrix} \omega n \\ d \end{bmatrix}_q}{\begin{bmatrix} n \\ d \end{bmatrix}_q} \approx q^{-d(1-\omega)n} \tag{2}$$

for $d \in \mathbb{N}$ and $\omega \in [0, 1]$.

Counting the number of operations. Coherently with the PKP literature, in our complexity analysis we consider as elementary operations both matrix multiplications and list operations. Therefore, when we say "operations", we refer to such operations and not to finite-field additions/multiplications.

Subcodes with small support. Given a $[n, k]$-linear code over $\mathbb{F}_q$, we follow [23, Thm. IV.1] and estimate the number of subcodes with support size $w \leq n$ and dimension $d \leq k$ with

$$N_{n,k}(w, d) = \binom{n}{w}(q^d - 1)^{w-d}\frac{\begin{bmatrix} k \\ d \end{bmatrix}_q}{\begin{bmatrix} n \\ d \end{bmatrix}_q}.$$

In [24, Prop. 2], it is shown the existence of an adaptation of Prange's ISD algorithm ([24, Alg. 3]) that takes in input a linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ of dimension k

and two other integers $w, d \in \mathbb{N}$ such that $w \leq n + d - k$ and returns a subcode of $\mathcal{C}$ with dimension d and support size w. The average running time of the algorithm is

$$T_{\mathsf{ISD}}^{(d)}(n,k,w) = \mathcal{O}\left(\frac{k^3 + \binom{k}{d}}{p^{(d)}(n,k,w)}\right) \tag{3}$$

with $p^{(d)}(n,k,w) = \min\left\{\frac{\binom{w}{d}\binom{n-w}{k-d}}{\binom{n}{k}} N_{n,k}(w,d),\ \ 1\right\}$.

2.1 The Permuted Kernel Problem

We now introduce the Permuted Kernel Problem (PKP), the central object of our study. We consider it directly in its search version, that is, under the assumption that a solution exists, as is the case in the setting of PERK ([1]).

Definition 1 (Permuted Kernel Problem). *Let $\boldsymbol{H} \in \mathbb{F}_q^{m \times n}$ be a random matrix and $\boldsymbol{V} \in \mathbb{F}_q^{\ell \times n}$, with $\ell \leq n - m$, be a random matrix for which there exists a permutation $\pi \in \mathsf{Sym}(n)$ such that $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{0}$. The* Permuted Kernel Problem (PKP) *is to recover π from $\boldsymbol{H}$ and $\boldsymbol{V}$.*

With this notation, we say that a PKP instance has parameters (n, m, q, ℓ). Note that in Definition 1 a solution exists by construction. However, in the following sections, we also consider instances that may not admit any solution. In such cases, we rely on an exhaustive search approach for solving them, which allows to detect whenever no solution exists. We nevertheless continue to refer to these as "PKP instances," even if they admit no solutions.

Remark 1. In contrast to [24], we restrict from the beginning to the case $\ell \leq n - m$. We show that this is actually not a restriction. First, notice that we can consider the matrices $\boldsymbol{V}$ and $\boldsymbol{H}$ to be full rank, since if that is not the case we can obtain an equivalent instance (i.e. they admit the same solutions) by removing the redundant rows. Such instance is equivalent to the starting one because the redundant rows introduce equations that are always verified (the other direction is easy). Under the assumption of full-rank matrices, the case $\ell > n - m$ is not possible. This can be easily seen considering the kernel dimension: consider $\boldsymbol{H} \in \mathsf{GL}_{m,n}(\mathbb{F}_q)$, $\boldsymbol{V} \in \mathsf{GL}_{\ell,n}(\mathbb{F}_q)$ and $\pi \in \mathsf{Sym}(n)$ such that

$$\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{0},$$

which is equivalent to requiring that $\boldsymbol{H}^\top \in \mathsf{ker}(\pi(\boldsymbol{V}))$. If the matrices are full-rank, we have that $\dim(\mathsf{ker}(\pi(\boldsymbol{V}))) = n - \mathsf{rank}(\pi(\boldsymbol{V})) = n - \ell$, which implies that $m \leq n - \ell$.

On the expected number of solutions. If π is a solution to the instance, its corresponding subspace must be orthogonal to the one generated by the rows of

$\boldsymbol{H}$. Thus, the space generated by the rows of $\pi(\boldsymbol{V})$ must lie in the orthogonal complement of the subspace generated by the rows of $\boldsymbol{H}$, which has dimension $n-m$. Therefore, the probability that a random ℓ-dimensional subspace of $\mathbb{F}_q^n$ is orthogonal to the space generated by $\boldsymbol{H}$ is $\frac{\begin{bmatrix} n-m \\ \ell \end{bmatrix}_q}{\begin{bmatrix} n \\ \ell \end{bmatrix}_q}$. The expected number of solutions is $n! \frac{\begin{bmatrix} n-m \\ \ell \end{bmatrix}_q}{\begin{bmatrix} n \\ \ell \end{bmatrix}_q}$. By putting $\omega = (1 - \frac{m}{n})$ and $d = \ell$ in Equation (2), we can approximate this quantity with $n! q^{-\ell m}$. Coherently with the literature on PKP [15,24], we consider instances where only one solution is expected, which is guaranteed by the following conditions (see [24]):

i) $\mathsf{rank}(\boldsymbol{V}) = \ell$ and $\mathsf{rank}(\boldsymbol{H}) = m$;
ii) $\boldsymbol{V}$ and $\boldsymbol{H}$ have no repeated columns;
iii) $q^\ell \geq n$ and $q^m \geq n$;
iv) $n! \frac{\begin{bmatrix} n-m \\ \ell \end{bmatrix}_q}{\begin{bmatrix} n \\ \ell \end{bmatrix}_q} < 1$.

Condition iii) is necessary because, for every column of $\boldsymbol{V}$ and $\boldsymbol{H}$, there are q^ℓ and q^m possible choices, respectively. If this condition does not hold, then either $\boldsymbol{V}$ or $\boldsymbol{H}$ must contain repeated columns. Condition iv) ensures that, in expectation, there exists only one solution—namely, the one introduced by construction. We remark that, from a purely brute-force approach point of view, instances admitting a unique solution are the hardest to solve. However, this is not necessarily the case for more advanced algorithms [15,24], whose complexity might grow with the expected number of solutions. Nevertheless, this choice is conservative in our case, since our algorithm introduced in Section 4 significantly benefits from having smaller field sizes q, which typically occurs when multiple solutions are expected. An in-depth analysis of the number of solutions to a random PKP instance was recently given in [22].

2.2 Basic approach to solve PKP

We show now how to exploit the kernel dimension to reduce the computational complexity of a brute force attack on PKP, which in principle is $n!$. Before doing that, we introduce the following variant of the problem.

Definition 2 (Inhomogeneous PKP). *Let $\boldsymbol{H} \in \mathbb{F}_q^{r \times n}$ be a random matrix. Let $\boldsymbol{V} \in \mathbb{F}_q^{\ell \times n}$ and $\boldsymbol{E} \in \mathbb{F}_q^{\ell \times r}$, with $\ell \leq n - r$, be random matrices for which there exists a permutation $\pi \in \mathsf{Sym}(n)$ such that $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$. The Inhomogeneous Permuted Kernel Problem* (IPKP) *is to recover π from $\boldsymbol{H}$, $\boldsymbol{V}$ and $\boldsymbol{E}$.*

Analogously as for PKP, we say that an IPKP instance has parameters (n, r, q, ℓ) and we refer to IPKP instances even in cases that may not admit any solution. For

the sake of easy presentation, we also denote by $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$ an IPKP instance defined by the matrices $\boldsymbol{H}$, $\boldsymbol{V}$ and $\boldsymbol{E}$. Moreover, we refer to IPKP instances even in cases in which the matrices are not completely random, as we will deal with instances in which the matrix $\boldsymbol{E}$ and the last row of $\boldsymbol{H}$ are fixed. We do this because our original goal is to solve a (homogeneous) PKP instance, which we show now to be equivalent to an IPKP instance with such properties.
If $\boldsymbol{H} \in \mathbb{F}_q^{m\times n}$, $\boldsymbol{V} \in \mathbb{F}_q^{\ell\times n}$ and $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{0}$ for some $\pi \in \mathsf{Sym}(n)$, then if we define $\boldsymbol{H}^{\mathsf{ext}} = \begin{bmatrix} \boldsymbol{H} \\ (1,\dots,1) \end{bmatrix}$, $\boldsymbol{E} = \left[\boldsymbol{0}_{\ell\times m} \sum_{i=1}^n \boldsymbol{v}_i\right]$ and $\tilde{\boldsymbol{V}} = \pi(\boldsymbol{V})$, we have that

$$\tilde{\boldsymbol{V}}\boldsymbol{H}^{\mathsf{ext}\top} = \left[\tilde{\boldsymbol{V}}\boldsymbol{H}^\top \; \tilde{\boldsymbol{V}} \cdot (1,\dots,1)^\top\right] = \left[\boldsymbol{0}_{\ell\times m} \sum_{i=1}^n \tilde{\boldsymbol{v}}_i\right] = \boldsymbol{E}. \tag{4}$$

This means that $\pi \in \mathsf{Sym}(n)$ also solves the IPKP instance defined by the matrices $\boldsymbol{H}^{\mathsf{ext}}$, $\boldsymbol{V}$ and $\boldsymbol{E}$. The other direction follows because, if $\pi \in \mathsf{Sym}(n)$ satisfies Equation (4), then it holds that $\tilde{\boldsymbol{V}}(\boldsymbol{H}^{\mathsf{ext}\{1,\dots,m\}})^\top = \boldsymbol{E}_{\{1,\dots,m\}} = \boldsymbol{0}_{\ell\times m}$, which is the equation defining the starting PKP instance. This is useful because we now show that to solve an $(n,r,q,\ell) - \mathsf{IPKP}$ instance it suffices to guess the action of π on $n-r$ coordinates. The same applies to PKP. However, if we want to solve an $(n,m,q,\ell) - \mathsf{PKP}$ instance, it is more convenient to solve the equivalent $(n,m+1,q,\ell) - \mathsf{IPKP}$ instance, since we need to guess the action of π on one coordinate less.
From now on, we refer to $\boldsymbol{H}$ as the extended parity check matrix.

Remark 2. In the same way as above, it is also possible to extend the parity check matrix in IPKP instances. In our newly-proposed algorithm we make use of this, see, for example, Line 7 and Line 15 in Algorithm 7. Notice, however, that we can not extend the same parity check matrix more than once, since doing this would not increase the rank.

Basic solver for IPKP. Let us consider now an (n,r,q,ℓ)-IPKP instance $(\boldsymbol{H},\boldsymbol{V},\boldsymbol{E})$ for which there exists $\pi \in \mathsf{Sym}(n)$ such that $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$. If we suppose that $J = \{n-r+1,\dots,n\}$ is an information set for $\boldsymbol{H}$ and let $\left[\boldsymbol{U}\ \boldsymbol{I}_r\right], \boldsymbol{S} = \mathsf{RREF}^*{}_J(\boldsymbol{H})$, then $\left[\tilde{\boldsymbol{V}}_{\neg_n J}\ \tilde{\boldsymbol{V}}_J\right](\boldsymbol{S}\boldsymbol{H})^\top = \boldsymbol{E}\boldsymbol{S}^\top$. This is equivalent to

$$\tilde{\boldsymbol{V}}_{\neg_n J}\boldsymbol{U}^\top + \tilde{\boldsymbol{V}}_J = \boldsymbol{E}\boldsymbol{S}^\top \quad \Leftrightarrow \quad \tilde{\boldsymbol{V}}_J = \boldsymbol{E}\boldsymbol{S}^\top - \tilde{\boldsymbol{V}}_{\neg_n J}\boldsymbol{U}^\top.$$

Finally, note that if J' is another information set for $\boldsymbol{H}$, it holds that

$$\tilde{\boldsymbol{V}}_{J'} = (\boldsymbol{E} - \tilde{\boldsymbol{V}}_{\neg_n J'}(\boldsymbol{H}_{\neg_n J'})^\top)(\boldsymbol{H}_{J'})^{-\top}, \tag{5}$$

where $^{-\top}$ denotes the inverse transposal operator. The relation above provides an improved brute-force solver: the columns with indices in $\neg_n J'$ are guessed, while the others are deduced from Equation (5). This method to solve IPKP is formalized in Algorithm 1 and has a time complexity of $\mathcal{O}(\frac{n!}{r!})$. This is obtained because the *Row-Reduced-Echelon-Form* can be computed with one elementary operation, since computing it has the same cost as a matrix multiplication (that

is $\mathcal{O}(n^3)$ field operations). Therefore the cost of computing the RREF is absorbed by the $\mathcal{O}$-notation. In Algorithm 1, for the sake of easy presentation, we assume that $\{n-r+1,\dots,n\}$ is an information set for $\boldsymbol{H}$, but the reasoning works for every information set J by using Equation (5).

Algorithm 1 $\mathsf{SolveIPKP}(\boldsymbol{H},\boldsymbol{V},\boldsymbol{E})$

Input: $\boldsymbol{H}\in\mathbb{F}_q^{r\times n}$, $\boldsymbol{V}\in\mathbb{F}_q^{\ell\times n}$, $\boldsymbol{E}\in\mathbb{F}_q^{\ell\times r}$
Output: $\pi\in\mathsf{Sym}(n)$ s.t. $\pi(\boldsymbol{V})\boldsymbol{H}^\top=\boldsymbol{E}$

$\left[\boldsymbol{U}\ \boldsymbol{I}_r\right]$, $\boldsymbol{S}\leftarrow\mathsf{RREF}^*{}_{\{n-r+1,\dots,n\}}(\boldsymbol{H})$
Initialize an empty list $\mathcal{L}$
for $\tilde{\boldsymbol{V}}_1\in\mathcal{S}_{n-r}(\boldsymbol{V})$ **do**
  $\tilde{\boldsymbol{V}}_2\leftarrow\boldsymbol{E}\boldsymbol{S}^\top-\tilde{\boldsymbol{V}}_1\boldsymbol{U}^\top$
  if $\left[\tilde{\boldsymbol{V}}_1\ \tilde{\boldsymbol{V}}_2\right]\in\mathcal{S}_n(\boldsymbol{V})$ **then**
    Compute $\pi\in\mathsf{Sym}(n)$ s.t. $\left[\tilde{\boldsymbol{V}}_1\ \tilde{\boldsymbol{V}}_2\right]=\pi(\boldsymbol{V})$
    Return: π

2.3 Working with the dual

Consider now the following (n,r,q,ℓ)-IPKP instance.

$$\pi(\boldsymbol{V})\boldsymbol{H}^\top=\boldsymbol{E}.$$

If we take the transpose we obtain the *dual* version of the problem (notice that, if $\boldsymbol{E}=\boldsymbol{0}$, the *dual* denomination becomes proper)

$$\pi^{-1}(\boldsymbol{H})\boldsymbol{V}^\top=\boldsymbol{E}^\top. \tag{6}$$

This can be seen as an IPKP instance with parameters (n,ℓ,q,r). By means of Algorithm 1, we can solve this problem instance by guessing the action of π^{-1} on $n-\ell$ coordinates, that is with $\mathcal{O}(\frac{n!}{\ell!})$ operations. Finally note that we can extend $\boldsymbol{V}$ by adding a row of ones in the same way as for $\boldsymbol{H}$ and so the number of operations needed is $\mathcal{O}(\frac{n!}{(\ell+1)!})$.

Remark 3. As explained above, Algorithm 1 terminates in either $\mathcal{O}(\frac{n!}{r!})$ (here we consider $\boldsymbol{H}$ to have already been extended) operations or in $\mathcal{O}(\frac{n!}{(\ell+1)!})$ operations when considering the dual problem. Considering the dual problem shows that IPKP (the same applies to PKP) becomes easier as ℓ grows, since Algorithm 1 becomes faster.
The existing algorithms, that are presented in the next section, also become faster as ℓ grows, but for different reasons (their time complexity depends on ℓ in a different way). In this paper we show how increasing ℓ can affect the hardness of the problem in a more impactful way than in the existing algorithms.

3 Review of the Known Attacks to PKP

In this section, we review the main known algorithms to solve PKP. We focus in particular on the algorithms by Koussa, Macario-Rat and Patarin (KMP) [15], and Santini, Baldi and Chiaraluce (SBC) [24]. More precisely, for the KMP algorithm, we review its generalization to the multi-dimensional (i.e. $\ell > 1$) setting given in [24].

3.1 KMP algorithm

We focus on solving an IPKP instance defined by the matrices $\boldsymbol{H} \in \mathbb{F}_q^{r\times n}$, $\boldsymbol{V} \in \mathbb{F}_q^{\ell\times n}$, $\boldsymbol{E} \in \mathbb{F}_q^{\ell\times r}$, that is, we want to find $\pi \in \mathsf{Sym}(n)$ such that $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$. As in Section 2, let $\tilde{\boldsymbol{V}} = \pi(\boldsymbol{V})$. In a nutshell, the KMP algorithm performs a brute-force attack to guess the action of π on $n-r$ columns, but it does it by adopting a *meet-in-the-middle* approach, which leads to significant time-complexity improvements. More specifically, the algorithm aims at guessing the action of π on v columns, with $v > n-r$, by guessing the action on u_1 and u_2 (with $v = u_1 + u_2$) columns separately and then matching the guessed columns. Notice that this matching can be seen as a sieving phase. The optimal values for u_1, u_2 depend on the instance and so they will be two parameters of the algorithm.

Outline of the algorithm. As in Algorithm 1, $\boldsymbol{H}$ is put in systematic form with the identity matrix on the right, that is, we are assuming that $\{n-r+1,\dots n\}$ is an information set for $\boldsymbol{H}$, but the reasoning works for every information set. Let $\boldsymbol{H}', \boldsymbol{M} = \mathsf{RREF}^*_{\{n-r+1,\dots n\}}(\boldsymbol{H})$. Then $\tilde{\boldsymbol{V}}\boldsymbol{H}^\top = \boldsymbol{E}$ if and only if $\tilde{\boldsymbol{V}}\boldsymbol{H}'^\top = \boldsymbol{E}\boldsymbol{M}^\top$. Let $\bar{\boldsymbol{H}} = \boldsymbol{H}'^{\{1,\dots,\tilde{r}\}}_{\{1,\dots,u_1+u_2\}}$ be the matrix obtained by taking the first $\tilde{r}$ rows and the first u_1+u_2 columns of $\boldsymbol{H}'$, with $\tilde{r} = u_1+u_2-(n-r) \geq 1$. Let $\boldsymbol{M}' = \boldsymbol{M}^{\{1,\dots,\tilde{r}\}}$ (notice that $\boldsymbol{M}' \in \mathsf{GL}_{\tilde{r},r}(\mathbb{F}_q)$), so that $\bar{\boldsymbol{H}} = \boldsymbol{M}'\boldsymbol{H}_{\{1,\dots,u_1+u_2\}}$. Then, it holds that $\tilde{\boldsymbol{V}}_{\{1,\dots,u_1+u_2\}}\bar{\boldsymbol{H}}^\top = \boldsymbol{E}\boldsymbol{M}'^\top$. Indeed, $\tilde{\boldsymbol{V}}\boldsymbol{H}'^\top = \boldsymbol{E}\boldsymbol{M}^\top$ and $\boldsymbol{H}'$ can be written as $\boldsymbol{H}' = \begin{bmatrix} \bar{\boldsymbol{H}} & \boldsymbol{0}_{\tilde{r}\times n-(u_1+u_2)} \\ \boldsymbol{A} & \boldsymbol{I}_{n-(u_1+u_2)} \end{bmatrix}$ with $\boldsymbol{A} \in \mathbb{F}_q^{(r-\tilde{r})\times(u_1+u_2)}$, so that

$$\begin{aligned}\tilde{\boldsymbol{V}}\boldsymbol{H}'^\top &= \begin{bmatrix}\tilde{\boldsymbol{V}}_{\{1,\dots,u_1+u_2\}} & \tilde{\boldsymbol{V}}_{\{u_1+u_2+1,\dots,n\}}\end{bmatrix}\begin{bmatrix} \bar{\boldsymbol{H}} & \boldsymbol{0}_{\tilde{r}\times n-(u_1+u_2)} \\ \boldsymbol{A} & \boldsymbol{I}_{n-(u_1+u_2)} \end{bmatrix}^\top = \\ &= \begin{bmatrix}\tilde{\boldsymbol{V}}_{\{1,\dots,u_1+u_2\}}\bar{\boldsymbol{H}}^\top & \tilde{\boldsymbol{V}}_{\{1,\dots,u_1+u_2\}}\boldsymbol{A}^\top + \tilde{\boldsymbol{V}}_{\{u_1+u_2+1,\dots,n\}}\end{bmatrix} = \boldsymbol{E}\boldsymbol{M}^\top.\end{aligned}$$

By definition of $\boldsymbol{M}'$, $(\boldsymbol{E}\boldsymbol{M}^\top)_{\{1,\dots,\tilde{r}\}} = \boldsymbol{E}\boldsymbol{M}'^\top$. We define $\boldsymbol{E} = \bar{\boldsymbol{E}}\boldsymbol{M}'^\top$ and so we have

$$\tilde{\boldsymbol{V}}_{\{1,\dots,u_1+u_2\}}\bar{\boldsymbol{H}}^\top = \bar{\boldsymbol{E}}. \tag{7}$$

Then, we call $\bar{\boldsymbol{H}}_1 = \bar{\boldsymbol{H}}_{\{1,\dots,u_1\}}$ and $\bar{\boldsymbol{H}}_2 = \bar{\boldsymbol{H}}_{\{u_1+1,\dots,u_1+u_2\}}$ and search for candidates for $\tilde{\boldsymbol{V}}_{\{1,\dots,u_1+u_2\}}$. For a visualization of these matrices see Fig. 1.

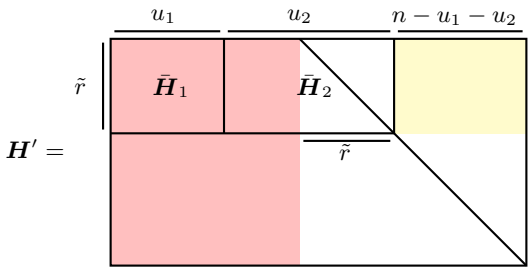

Fig. 1: Representation of the considered matrices, the yellow area is the zero matrix

Such candidates are found by building the following lists:

$$\begin{aligned} \mathcal{L}_1 &= \left\{ (\boldsymbol{X}, \boldsymbol{X}\bar{\boldsymbol{H}}_1^\top) \mid \boldsymbol{X} \in \mathcal{S}_{u_1}(\boldsymbol{V}) \right\}, \\ \mathcal{L}_2 &= \left\{ (\boldsymbol{Y}, \bar{\boldsymbol{E}} - \boldsymbol{Y}\bar{\boldsymbol{H}}_2^\top) \mid \boldsymbol{Y} \in \mathcal{S}_{u_2}(\boldsymbol{V}) \right\}. \end{aligned} \tag{8}$$

The algorithm proceeds by finding all *valid* collisions between $\mathcal{L}_1$ and $\mathcal{L}_2$, that is, finding pairs $(\boldsymbol{X}, \bar{\boldsymbol{X}}) \in \mathcal{L}_1$ and $(\boldsymbol{Y}, \bar{\boldsymbol{Y}}) \in \mathcal{L}_2$ s.t. $\bar{\boldsymbol{X}} = \bar{\boldsymbol{Y}}$ and $\begin{bmatrix} \boldsymbol{X} & \boldsymbol{Y} \end{bmatrix} \in \mathcal{S}_{u_1+u_2}(\boldsymbol{V})$ (therefore $\begin{bmatrix} \boldsymbol{X} & \boldsymbol{Y} \end{bmatrix}$ is a *valid* candidate for $\tilde{\boldsymbol{V}}_{\{1,\dots,u_1+u_2\}}$). We denote this operation of selecting all *valid* candidates by $\bowtie$, and call $\mathcal{L}$ the resulting list for $\tilde{\boldsymbol{V}}_{\{1,\dots,u_1+u_2\}}$, that is,

$$\mathcal{L} = \mathcal{L}_1 \bowtie \mathcal{L}_2 = \left\{ \boldsymbol{Z} \in \mathcal{S}_{u_1+u_2}(\boldsymbol{V}) \mid \boldsymbol{Z}\bar{\boldsymbol{H}}^\top = \bar{\boldsymbol{E}} \right\}.$$

From Equation (7) we have that $\tilde{\boldsymbol{V}}_{\{1,\dots,u_1+u_2\}} \in \mathcal{L}$ so, for every element in $\mathcal{L}$, we truncate it of $\tilde{r}$ columns and proceed as in Algorithm 1, and eventually find $\tilde{\boldsymbol{V}}$. Finally note that the requirement $u_1 + u_2 \geq n - r + 1$ is needed to have $\tilde{r} \geq 1$, so that it makes sense to look for collisions among matrices that have $\tilde{r}$ columns. The full algorithm is formalized in Algorithm 2.

Complexity of KMP. For the asymptotic analysis of the complexity of KMP we refer to [24]; here we are only interested in the finite regime, where its running time, measured in the number of matrix multiplications and list operations, is given by the following proposition.

Proposition 1 (Complexity of KMP [24, Prop. 5]). *Let $u_1, u_2 \in \mathbb{N}^*$ such that $n - r + 1 \leq u_1 + u_2 \leq n$. Then the KMP algorithm runs in average time*

$$T_{\mathsf{KMP}}(u_1, u_2) = |\mathcal{L}_1| + |\mathcal{L}_2| + N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} + |\mathcal{L}|$$

where

$$|\mathcal{L}_i| = \frac{n!}{(n-u_i)!}, \quad N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} = \frac{(n!)^2 q^{\ell(n-r-u_1-u_2)}}{(n-u_1)!(n-u_2)!},$$

Algorithm 2 $\mathsf{KMP}(\boldsymbol{H}, \boldsymbol{V}, \boldsymbol{E})$

Parameters: $u_1, u_2 \in \mathbb{N}^*$ s.t. $n - r + 1 \leq u_1 + u_2 \leq n$
Input: $\boldsymbol{H} \in \mathbb{F}_q^{r \times n}$, $\boldsymbol{V} \in \mathbb{F}_q^{\ell \times n}$, $\boldsymbol{E} \in \mathbb{F}_q^{\ell \times r}$
Output: $\pi \in \mathsf{Sym}(n)$ such that $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$

$\tilde{r} \leftarrow u_1 + u_2 - (n - r)$
$\boldsymbol{H}'$, $\boldsymbol{M} \leftarrow \mathsf{RREF}^*{}_{\{n-r+1,\ldots,n\}}(\boldsymbol{H})$, $\begin{bmatrix}\boldsymbol{U} & \boldsymbol{I}_r\end{bmatrix} \leftarrow \boldsymbol{H}'$
$\bar{\boldsymbol{H}} \leftarrow \boldsymbol{H}'^{\{1,\ldots,\tilde{r}\}}_{\{1,\ldots,u_1+u_2\}}$, $\quad \boldsymbol{M}' \leftarrow \boldsymbol{M}^{\{1,\ldots,\tilde{r}\}}$
$\bar{\boldsymbol{E}} \leftarrow \boldsymbol{E}\boldsymbol{M}'^\top$, $\bar{\boldsymbol{H}}_1 \leftarrow \bar{\boldsymbol{H}}_{\{1,\ldots,u_1\}}$, $\bar{\boldsymbol{H}}_2 \leftarrow \bar{\boldsymbol{H}}_{\{u_1+1,\ldots,u_1+u_2\}}$
$\mathcal{L}_1 \leftarrow \left\{(\boldsymbol{X}, \boldsymbol{X}\bar{\boldsymbol{H}}_1^\top) \mid \boldsymbol{X} \in \mathcal{S}_{u_1}(\boldsymbol{V})\right\}$,
$\mathcal{L}_2 \leftarrow \left\{(\boldsymbol{X}, \bar{\boldsymbol{E}} - \boldsymbol{Y}\bar{\boldsymbol{H}}_2^\top) \mid \boldsymbol{Y} \in \mathcal{S}_{u_2}(\boldsymbol{V})\right\}$
$\mathcal{L} \leftarrow \mathcal{L}_1 \bowtie \mathcal{L}_2$
for $\tilde{\boldsymbol{V}}_1 \in \mathcal{L}$ **do**
    $\tilde{\boldsymbol{V}}_2 \leftarrow \boldsymbol{E}\boldsymbol{M}^\top - \tilde{\boldsymbol{V}}_{1\{1\ldots,n-r\}}\boldsymbol{U}^\top$
    if $\begin{bmatrix}\tilde{\boldsymbol{V}}_1 & \tilde{\boldsymbol{V}}_2\end{bmatrix} \in \mathcal{S}_n(\boldsymbol{V})$ **then**
        Compute $\pi \in \mathsf{Sym}(n)$ s.t. $\begin{bmatrix}\tilde{\boldsymbol{V}}_1 & \tilde{\boldsymbol{V}}_2\end{bmatrix} = \pi(\boldsymbol{V})$
        Return: π

$$|\mathcal{L}| = \frac{n!}{(n-u_1-u_2)!} q^{\ell(n-r-u_1-u_2)}.$$

Remark 4. We note that $N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}$ is the estimated number of collisions between $\mathcal{L}_1$ and $\mathcal{L}_2$. Indeed we are searching for collisions between $\ell \times \tilde{r}$ matrices, so the average number of collisions can be estimated by the number of possible pairs (given by $|\mathcal{L}_1| \cdot |\mathcal{L}_2|$) multiplied by the probability of a pair to collide, that is $\frac{1}{q^{\ell\tilde{r}}}$. Since $|\mathcal{L}| \leq N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}$ by construction (here the inequality is considered between the estimates), the complexity of the algorithm is dominated by $\max\{|\mathcal{L}_1|, |\mathcal{L}_2|, N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}\}$ and the algorithm is optimized when $|\mathcal{L}_1| \approx |\mathcal{L}_2| \approx N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}$, which implies that either $u_1 = u_2$ if $u_1 + u_2$ is even or $u_1 = u_2 \pm 1$ otherwise. For the sake of simplicity, let us consider from now on $u_1 = u_2 = u$ so that $|\mathcal{L}_1| = |\mathcal{L}_2| = \frac{n!}{(n-u)!}$. Then to optimize the algorithm it is necessary to require that $N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} \leq \frac{n!}{(n-u)!}$, which is equivalent to

$$\frac{n!}{(n-u)!} \leq q^{\ell(2u+r-n)}.$$

Hence, u is chosen as the smallest integer which satisfies the above inequality.

Notice that increasing q allows us to choose a smaller u, which speeds up the algorithm; therefore, the hardest PKP instances are the ones in which q is chosen adaptively as the smallest prime or prime power that guarantees the uniqueness of the solution, that is, the smallest q, that satisfies Condition iv) in Section 2.1. Moreover, notice that choosing q as small as possible has other advantages when considering applications of PKP to cryptosystems, such as speeding up the encryption/decryption or sign/verify and reducing the bit-size of keys or witnesses

in ZK-proofs (see [24]).

Finally, in [24] it was also shown that the complexity of KMP does not asymptotically depend on the value of ℓ and, even in the finite regime, increasing ℓ does not significantly change the running time (this will be examined in Section 5).

3.2 SBC algorithm

The SBC algorithm works similarly to KMP, but with an additional *filtering step*. Instead of building the lists as in Equation (8), here the idea is to choose the elements of the second list from a smaller set of matrices (that is why the algorithm can be considered as a version of KMP with an additional filtering step). Suppose that we want to solve an (n, r, q, ℓ)-IPKP instance $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$. As usual, we denote $\pi(\boldsymbol{V})$ by $\tilde{\boldsymbol{V}}$. We now provide a description of the algorithm divided into 3 steps.

STEP 1: filtering step. The first step of the algorithm is to find a subcode of the linear code generated by $\boldsymbol{H}$ with support size w and dimension d, where w and d will be two parameters taken as input by the algorithm (actually, instead of w, the algorithm will take in input $w_1, w_2 \in \mathbb{N}^*$ with $w_1 + w_2 = w$ since, again, a *meet-in-the-middle* approach is used). Notice that it is not possible to find such subcode for every $w \in [n]$ and for every $d \in [r]$, therefore a necessary condition for w, d to be valid parameters is that $N_{n,r}(w, d) > 1$. Assuming one subcode with the desired properties has been found, let $\boldsymbol{H}^* \in \mathbb{F}_q^{d\times n}$ be its generator matrix. We have that $\boldsymbol{H}^* = \boldsymbol{S}\boldsymbol{H}$ for some $\boldsymbol{S} \in \mathsf{GL}_{d,r}(\mathbb{F}_q)$. We will use the relation between $\boldsymbol{H}^*$ and $\boldsymbol{E}^* = \boldsymbol{E}\boldsymbol{S}^\top$ to filter the candidates for $\tilde{\boldsymbol{V}}_L$ where $L \subseteq [n]$ has size w. As in KMP, the idea is to guess $n - r + \tilde{r}$ columns with $\tilde{r} \geq 1$; however, this time $\tilde{r}$ is not fixed as in KMP, but it is a parameter itself that is taken as input by the algorithm with the condition $d \leq \tilde{r} \leq r$.

Remark 5. We now show why in the algorithm it is useful to consider another auxiliary permutation of the columns of $\boldsymbol{V}$ and why the condition $\tilde{r} \geq d$ arises. As in [24], we consider another permutation $\sigma \in \mathsf{Sym}(n)$ such that $\sigma(\boldsymbol{H}^*)$ has its w non-null columns in positions indexed by $J = \{n-r+\tilde{r}-w+1, \ldots, n-r+\tilde{r}\}$. This choice simplifies the description of the algorithm because, at some point, $\sigma(\boldsymbol{H})$ will be brought into *Row Reduced Echelon Form* with respect to $\{n-r+1, \ldots, n\}$, and this arrangement of σ ensures that this can be done (i.e. it guarantees that $\sigma(\boldsymbol{H})_{\{n-r+1,\ldots,n\}}$ is non-singular). To see this, assume, for example, that σ moves the non-null columns of $\boldsymbol{H}^*$ to the first w positions. Now let $\boldsymbol{S} \in \mathsf{GL}_{d,r}(\mathbb{F}_q)$ be such that $\boldsymbol{S}\sigma(\boldsymbol{H}) = \sigma(\boldsymbol{H}^*)$ (notice that this is the same matrix $\boldsymbol{S}$ defined before, so $\boldsymbol{H}^* = \boldsymbol{S}\boldsymbol{H}$), and define $\boldsymbol{N} = \begin{bmatrix} \boldsymbol{S}' \\ \boldsymbol{S} \end{bmatrix}$, where $\boldsymbol{S}' \in \mathbb{F}_q^{(r-d)\times r}$ is chosen so that $\boldsymbol{N}$ is non singular. Then $\boldsymbol{N}\sigma(\boldsymbol{H})$ has, in its bottom-rightmost part, a $d\times(n-w)$ null matrix. To have that $\boldsymbol{N}\sigma(\boldsymbol{H})_{\{n-r+1,\ldots,n\}}$ is non-singular (which means that it is possible to compute $\mathsf{RREF}_{\{n-r+1,\ldots,n\}}(\sigma(\boldsymbol{H}))$, since $\boldsymbol{N}\sigma(\boldsymbol{H})$ and $\sigma(\boldsymbol{H})$ share the same *Row Reduced Echelon Form*), it is necessary that it does not have linearly

dependent rows. Since the last d of them have zeros in $n-w$ entries, this implies that $r-(n-w) \geq d$. This is equivalent to $w \geq n-r+d$. However, in this case, the *filtering step* would be as expensive as the KMP algorithm itself, since we are guessing $w \geq n-r+d \geq n-r+1$ columns and in KMP we were guessing $v \geq n-r+1$ columns. Instead, if $\sigma \in \mathsf{Sym}(n)$ moves the non-null columns of $\boldsymbol{H}^*$ in positions $\{n-r+\tilde{r}-w+1, \ldots, n-r+\tilde{r}\}$, we have that $\boldsymbol{N}\sigma(\boldsymbol{H})$ has a $d \times (r-\tilde{r})$ null matrix in its bottom-rightmost matrix. Thus, for $\boldsymbol{N}\sigma(\boldsymbol{H})_{\{n-r+1,\ldots,n\}}$ to be non-singular, it is necessary that $r-(r-\tilde{r}) \geq d$, which is set by hypothesis.

Consider now $w_1, w_2 \in \mathbb{N}^*$ with $w_1 + w_2 = w$ and set $J_1 = \{n-r+\tilde{r}-w+1, \ldots, n-r+\tilde{r}-w_2\}$ and $J_2 = \{n-r+\tilde{r}-w_2+1, \ldots, n-r+\tilde{r}\}$ (notice that $|J_1| = w - w_2 = w_1$ and $|J_2| = w_2$). Let $\boldsymbol{H}_1^* = \sigma(\boldsymbol{H}^*)_{J_1}$ and $\boldsymbol{H}_2^* = \sigma(\boldsymbol{H}^*)_{J_2}$. Then $\sigma(\boldsymbol{H}^*)$ can be visualized as follows.

$$\sigma(\boldsymbol{H}^*) = \quad \begin{array}{c} \\ d \end{array} \left| \begin{array}{c|c|c|c} \overbrace{\phantom{n-r+\tilde{r}-w}}^{n-r+\tilde{r}-w} & \overbrace{}^{w_1} & \overbrace{}^{w_2} & \\ \mathbf{0} & \boldsymbol{H}_1^* & \boldsymbol{H}_2^* & \mathbf{0} \end{array} \right.$$

We further define $J = J_1 \cup J_2 = \{n-r+\tilde{r}-w+1, \ldots, n-r+\tilde{r}\}$ and $\bar{\boldsymbol{H}} = \sigma(\boldsymbol{H}^*)_J = \left[\boldsymbol{H}_1^* \; \boldsymbol{H}_2^*\right]$. Then the algorithm proceeds by searching for candidates for $\sigma(\tilde{\boldsymbol{V}})_J$ in the list

$$\mathcal{K} = \left\{ \boldsymbol{X} \in \mathcal{S}_w(\boldsymbol{V}) \mid \boldsymbol{X}\bar{\boldsymbol{H}}^\top = \boldsymbol{E}^* \right\}. \tag{9}$$

We will show now that $\sigma(\tilde{\boldsymbol{V}})_J \in \mathcal{K}$. As $\tilde{\boldsymbol{V}}\boldsymbol{H}^\top = \boldsymbol{E}$, then it is easy to see that

$$\boldsymbol{E}\boldsymbol{A}^\top = \sigma(\tilde{\boldsymbol{V}})(\boldsymbol{A}\sigma(\boldsymbol{H}))^\top, \quad \forall \boldsymbol{A} \in \mathbb{F}_q^{r' \times r}, \; \forall r' \leq r, \; \forall \sigma \in \mathsf{Sym}(n). \tag{10}$$

Since $\boldsymbol{H}^* = \boldsymbol{S}\boldsymbol{H}$ we have that $\boldsymbol{E}^* = \boldsymbol{E}\boldsymbol{S}^\top = \sigma(\tilde{\boldsymbol{V}})\sigma(\boldsymbol{H}^*)^\top$ and $\sigma(\boldsymbol{H}^*)$ has the structure showed in the figure above. Therefore, if we call $J' = \{1, \ldots, n-r+\tilde{r}-w\}$ and $J'' = \{n-r+\tilde{r}+1, \ldots, n\}$, we have

$$\boldsymbol{E}^* = \sigma(\tilde{\boldsymbol{V}})\sigma(\boldsymbol{H}^*)^\top = \left[\sigma(\tilde{\boldsymbol{V}})_{J'} \; \sigma(\tilde{\boldsymbol{V}})_J \; \sigma(\tilde{\boldsymbol{V}})_{J''}\right] \begin{bmatrix} \mathbf{0} \\ \bar{\boldsymbol{H}}^\top \\ \mathbf{0} \end{bmatrix} = \sigma(\tilde{\boldsymbol{V}})_J \bar{\boldsymbol{H}}^\top.$$

To build the list $\mathcal{K}$, a *meet-in-the-middle* approach is used, so instead of building it like in Equation (9), we build $\mathcal{K}$ as $\mathcal{K}_1 \bowtie \mathcal{K}_2$, with

$$\mathcal{K}_1 = \left\{ (\boldsymbol{X}_1, \boldsymbol{X}_1 \boldsymbol{H}_1^{*\top}) \mid \boldsymbol{X}_1 \in \mathcal{S}_{w_1}(\boldsymbol{V}) \right\},$$
$$\mathcal{K}_2 = \left\{ (\boldsymbol{X}_2, \boldsymbol{E}^* - \boldsymbol{X}_2 \boldsymbol{H}_2^{*\top}) \mid \boldsymbol{X}_2 \in \mathcal{S}_{w_2}(\boldsymbol{V}) \right\}.$$

STEP 2: extra guessing. From now on, we assume that $\{n-r+1, \ldots, n\}$ is an information set for $\sigma(\boldsymbol{H})$, but the reasoning works for every information set.

After the *filtering step*, we obtain a list of candidates for $\sigma(\tilde{\boldsymbol{V}})_J$, with $|J| = w$. Because of the considerations made in Remark 5, if $w \geq n-r+d$ we know enough columns to reconstruct the whole solution by exploiting the kernel dimension (as in Lines 4-7 in Algorithm 1). As observed in the same remark, the interesting values of w are the ones for which $w < n - r + d$ (otherwise the complexity of the algorithm would be essentially the same as KMP). If this is the case, we need to guess $n - r - w + \tilde{r}$ more columns. Notice that guessing $n - r - w + d$ more columns would be sufficient, however since this guessing is done by using a *meet-in-the-middle* approach, the algorithm might be optimized when guessing more columns.
The algorithm proceeds by computing $\boldsymbol{H}', \boldsymbol{M} = \mathsf{RREF}^*_{\{n-r+1,\dots,n\}}(\sigma(\boldsymbol{H}))$. Now we call $\hat{\boldsymbol{H}} = \boldsymbol{H'}^{\{d+1,\dots\tilde{r}\}}$, $\hat{\boldsymbol{E}} = (\boldsymbol{E}\boldsymbol{M}^\top)_{\{d+1,\dots,\tilde{r}\}}$, $\hat{\boldsymbol{H}}_1 = \hat{\boldsymbol{H}}_{J'}$ and $\hat{\boldsymbol{H}}_2 = \hat{\boldsymbol{H}}_J$ and build the lists

$$\begin{aligned}
\mathcal{L}_1 &= \left\{ (\boldsymbol{Y}_1, \boldsymbol{Y}_1 \hat{\boldsymbol{H}}_1^\top) \mid \boldsymbol{Y}_1 \in \mathcal{S}_{n-r+\tilde{r}-w}(\boldsymbol{V}) \right\}, \\
\mathcal{L}_2 &= \left\{ (\boldsymbol{Y}_2, \hat{\boldsymbol{E}} - \boldsymbol{Y}_2 \hat{\boldsymbol{H}}_2^\top) \mid \boldsymbol{Y}_2 \in \mathcal{K} \right\}, \\
\mathcal{L} &= \mathcal{L}_1 \bowtie \mathcal{L}_2.
\end{aligned}$$

It is easy to see that $\sigma(\tilde{\boldsymbol{V}})_{J' \cup J} \in \mathcal{L}$. To visualize this, we refer to Figure 2.

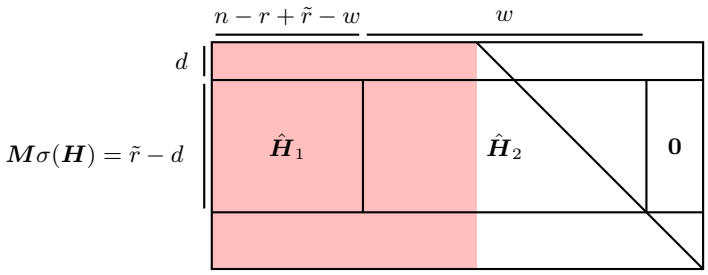

Fig. 2: Matrices employed in the SBC algorithm. The pink area is the matrix $\boldsymbol{U}$ defined in Algorithm 3.

Indeed, from Equation (10) we have that $\boldsymbol{E}\boldsymbol{M}^\top = \sigma(\tilde{\boldsymbol{V}})(\boldsymbol{M}\sigma(\boldsymbol{H}))^\top$. This implies

$$(\boldsymbol{E}\boldsymbol{M}^\top)_{\{d+1,\dots,\tilde{r}\}} = \left(\sigma(\tilde{\boldsymbol{V}})(\boldsymbol{M}\sigma(\boldsymbol{H}))^\top\right)_{\{d+1,\dots,\tilde{r}\}} = \sigma(\tilde{\boldsymbol{V}}) \left((\boldsymbol{M}\sigma(\boldsymbol{H}))^\top\right)_{\{d+1,\dots,\tilde{r}\}}.$$

Observing that selecting a set of columns of the transpose is the same as selecting that set of rows of the non-transpose, we obtain that:

$$\hat{\boldsymbol{E}} = \sigma(\tilde{\boldsymbol{V}})\hat{\boldsymbol{H}}^\top = \begin{bmatrix} \sigma(\tilde{\boldsymbol{V}})_{J'} & \sigma(\tilde{\boldsymbol{V}})_J & \sigma(\tilde{\boldsymbol{V}})_{J''} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{H}}_1^\top \\ \hat{\boldsymbol{H}}_2^\top \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \sigma(\tilde{\boldsymbol{V}})_{J'} & \sigma(\tilde{\boldsymbol{V}})_J \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{H}}_1^\top \\ \hat{\boldsymbol{H}}_2^\top \end{bmatrix}.$$

Notice that the last columns $r - \tilde{r}$ of $\hat{\boldsymbol{H}}$ are zero because they are obtained by taking a null submatrix of I_r. From the first step of the algorithm we have that $\sigma(\tilde{\boldsymbol{V}})_J \in \mathcal{K}$, so $\sigma(\tilde{\boldsymbol{V}})_{J' \cup J} \in \mathcal{L}$.

STEP 3: reconstruct the solution. As said before, at the end of **STEP 2** we know enough columns of $\sigma(\pi(\boldsymbol{V}))$ to reconstruct the solution working as in Section 2.2 and in the last part of KMP (that is, by exploiting the kernel dimension).

Remark 6. In Algorithm 3, in **STEP 3**, the solution is reconstructed by considering the last $r - \tilde{r}$ rows of $\boldsymbol{U}$ and consequently the last $r - \tilde{r}$ columns of $\boldsymbol{E}\boldsymbol{M}^\top$. This is done because we are looking for $\boldsymbol{X}'$ such that $\begin{bmatrix}\boldsymbol{X} & \boldsymbol{X}'\end{bmatrix} \in \mathcal{S}_n(\boldsymbol{V})$ and $\begin{bmatrix}\boldsymbol{X} & \boldsymbol{X}'\end{bmatrix} \begin{bmatrix}\boldsymbol{U}^\top \\ \boldsymbol{I}_r\end{bmatrix} = \boldsymbol{E}\boldsymbol{M}^\top$. However, we can just check the equality between the last $r - \tilde{r}$ columns of these matrices, because the equality between the first $n - r + \tilde{r}$ columns has already been checked in **STEP 2** (a similar reasoning also applies when discarding the first d columns after **STEP 1**).

Complexity of SBC. The next proposition provides the average running time of SBC.

Proposition 2 (Complexity of SBC [24, Prop. 9]). *Let d, w_1, w_2, $\tilde{r} \in \mathbb{N}^*$ be such that $w = w_1 + w_2 \leq n$, $d \leq \tilde{r} \leq r$ and $N_{n,r}(w, d) > 1$. Then Algorithm 3 runs in average time*

$$T_{\mathsf{SBC}}(n, r, q, \ell, w_1, w_2, d, \tilde{r}) = T^{(d)}_{\mathsf{ISD}}(n, r, w) + T_{\mathcal{K}} + T_{\mathcal{L}} + |\mathcal{L}| \tag{11}$$

where $T^{(d)}_{\mathsf{ISD}}(n, r, w)$ is defined as in Equation (3) and

$$T_{\mathcal{K}} = \frac{n!}{(n-w_1)!} + \frac{n!}{(n-w_2)!} + \frac{(n!)^2 q^{-d\ell}}{(n-w_1)!(n-w_2)!},$$

$$T_{\mathcal{L}} = \frac{n!}{(r+w-\tilde{r})!} + |\mathcal{K}| + \frac{n!|\mathcal{K}|q^{-\ell(\tilde{r}-d)}}{(r+w-\tilde{r})!}, \tag{12}$$

$$|\mathcal{L}| = \frac{(n-w)!|\mathcal{K}|q^{-\ell(\tilde{r}-d)}}{(r-\tilde{r})!}, |\mathcal{K}| = \max\left\{\frac{n!q^{-d\ell}}{(n-w)!}, 1\right\}. \tag{13}$$

Remark 7. In Proposition 2, $\tilde{r}$ is allowed to be equal to d. In this case, in **STEP 2**, one would build matrices with zero rows/columns. This is, in some sense, taken into account in Proposition 2, but in this case the complexity of the algorithm is slightly overestimated. If $\tilde{r} = d$, then for every candidate $\boldsymbol{X} \in \mathcal{K}$, one should try to reconstruct the solution for every possible way of choosing $n - r + \tilde{r} - w$ columns of $\boldsymbol{V}$ (without checking for collisions as in **STEP 2**) between the $n - w$ columns that do not appear in $\boldsymbol{X}$. This would lead to a slightly smaller complexity than the one showed in Proposition 2, because the complexity given by Equation (11) assumes that for every candidate the extra

Algorithm 3 $\mathsf{SBC}(\boldsymbol{H}, \boldsymbol{V}, \boldsymbol{E})$

Parameters: w_1, w_2, $\tilde{r}$, $d \in \mathbb{N}$ with $w = w_1 + w_2 \leq n$, $d \leq \tilde{r} \leq r$ and $N_{n,r}(w, d) > 1$

Input: $\boldsymbol{H} \in \mathsf{GL}_{r,n}(\mathbb{F}_q)$, $\boldsymbol{V} \in \mathsf{GL}_{\ell,n}(\mathbb{F}_q)$, $\boldsymbol{E} \in \mathbb{F}_q^{\ell \times r}$

Output: $\pi \in \mathsf{Sym}(n)$ such that $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$

STEP 1: filtering step

Use ISD to find a subcode of the code generated by $\boldsymbol{H}$ with dimension d and support size $w = w_1 + w_2$. Call $\mathcal{C}$ this subcode, $\boldsymbol{H}^*$ its generator matrix and $\boldsymbol{S} \in \mathsf{GL}_{d,r}(\mathbb{F}_q)$ the matrix s.t. $\boldsymbol{H}^* = \boldsymbol{S}\boldsymbol{H}$.

Compute $\sigma \in \mathsf{Sym}(n)$ such that $\mathrm{Supp}(\sigma(\mathcal{C})) = \{n-r+\tilde{r}-w+1, \ldots, n-r+\tilde{r}\}$

$J_1 \leftarrow \{n-r+\tilde{r}-w+1, \ldots, n-r+\tilde{r}-w_2\}$, $J_2 \leftarrow \{n-r+\tilde{r}-w_2+1, \ldots, n-r+\tilde{r}\}$

$\boldsymbol{E}^* \leftarrow \boldsymbol{E}\boldsymbol{S}^\top$, $\boldsymbol{H}_1^* \leftarrow \sigma(\boldsymbol{H}^*)_{J_1}$, $\boldsymbol{H}_2^* \leftarrow \sigma(\boldsymbol{H}^*)_{J_2}$

$\mathcal{K}_1 \leftarrow \left\{(\boldsymbol{X}_1, \boldsymbol{X}_1\boldsymbol{H}_1^{*\top}) \mid \boldsymbol{X}_1 \in \mathcal{S}_{w_1}(\boldsymbol{V})\right\}$,

$\mathcal{K}_2 \leftarrow \left\{(\boldsymbol{X}_2, \boldsymbol{E}^* - \boldsymbol{X}_2\boldsymbol{H}_2^{*\top}) \mid \boldsymbol{X}_2 \in \mathcal{S}_{w_2}(\boldsymbol{V})\right\}$

$\mathcal{K} \leftarrow \mathcal{K}_1 \bowtie \mathcal{K}_2$

STEP 2: extra guessing

$\boldsymbol{H}', \boldsymbol{M} \leftarrow \mathsf{RREF}^*_{\{n-r+1,\ldots,n\}}(\sigma(\boldsymbol{H}))$, $\begin{bmatrix}\boldsymbol{U} & \boldsymbol{I}_r\end{bmatrix} \leftarrow \boldsymbol{H}'$

$\hat{\boldsymbol{H}} \leftarrow \boldsymbol{H}'^{\{d+1,\ldots,\tilde{r}\}}$, $\hat{\boldsymbol{E}} \leftarrow (\boldsymbol{E}\boldsymbol{M}^\top)_{\{d+1,\ldots,\tilde{r}\}}$

$J' \leftarrow \{1, \ldots, n-r+\tilde{r}-w\}$, $J \leftarrow J_1 \cup J_2$, $\hat{\boldsymbol{H}}_1 \leftarrow \hat{\boldsymbol{H}}_{J'}$, $\hat{\boldsymbol{H}}_2 \leftarrow \hat{\boldsymbol{H}}_J$

$\mathcal{L}_1 = \left\{(\boldsymbol{Y}_1, \boldsymbol{Y}_1\hat{\boldsymbol{H}}_1^\top) \mid \boldsymbol{Y}_1 \in \mathcal{S}_{n-r+\tilde{r}-w}(\boldsymbol{V})\right\}$

$\mathcal{L}_2 = \left\{(\boldsymbol{Y}_2, \hat{\boldsymbol{E}} - \boldsymbol{Y}_2\hat{\boldsymbol{H}}_2^\top) \mid \boldsymbol{Y}_2 \in \mathcal{K}\right\}$

$\mathcal{L} = \mathcal{L}_1 \bowtie \mathcal{L}_2$

STEP 3: reconstruct the solution

$\bar{\boldsymbol{U}} \leftarrow \boldsymbol{U}^{\{\tilde{r}+1,\ldots,r\}}$, $\bar{\boldsymbol{E}} \leftarrow (\boldsymbol{E}\boldsymbol{M}^\top)_{\{\tilde{r}+1,\ldots,r\}}$

for $\boldsymbol{X} \in \mathcal{L}$ **do**

  $\boldsymbol{X}' \leftarrow \bar{\boldsymbol{E}} - \boldsymbol{X}_{\{1,\ldots,n-r\}}\bar{\boldsymbol{U}}^\top$

  if $\begin{bmatrix}\boldsymbol{X} & \boldsymbol{X}'\end{bmatrix} \in \mathcal{S}_n(\boldsymbol{V})$ **then**

    Compute $\pi \in \mathsf{Sym}(n)$ s.t. $\sigma^{-1}\left(\begin{bmatrix}\boldsymbol{X} & \boldsymbol{X}'\end{bmatrix}\right) = \pi(\boldsymbol{V})$

    Return: π

guessing is made between all the n columns of $\boldsymbol{V}$ instead of between the $n-w$ remaining ones. Therefore, the term $T_{\mathcal{L}} + |\mathcal{L}|$ in Equation (11) should be replaced by $\frac{(n-w)!|\mathcal{K}|}{(r-\tilde{r})!}$. We point out that, based on our analysis, this does not lead to significant changes in the running time, but we report it for completeness.

4 New Algorithmic Improvements for Solving PKP

In this section, we introduce several improvements to SBC. These improvements apply to KMP as well, but since SBC outperforms KMP in the finite regime (see [24, Sec. 6.2]), that is of primary interest in this work, we focus on SBC. We introduce variants both for the *filtering step* (**STEP 1** in Algorithm 3) and

for the *extra guessing* (**STEP 2** in Algorithm 3). Then, we outline our refined new algorithm which chooses the most convenient variant of these steps for each choice of parameters.

In the following, we consider an (n, r, q, ℓ)-IPKP instance $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$ with $\mathsf{rank}(\boldsymbol{E}) = 1$ and in which the last row of $\boldsymbol{H}$ is a row of ones (this is the case of an IPKP-instance obtained by de-homogenizing a PKP instance) and use the same notation used in the previous section. Although the algorithm applies to arbitrary IPKP instances, we focus on instances with specific properties arising from the dehomogenization of a (homogeneous) PKP instance. Such IPKP instances cannot be treated as random in the complexity analysis, as dehomogenization introduces equations that hold for every permutation, which would otherwise lead to an underestimation of the number of solutions and hence of the complexity (see Remark 14).

4.1 Variant of STEP 2: Using the dual to guess more columns

We begin introducing our proposed variant of **STEP 2** of SBC because this modification applies also to our variant of **STEP 1** that is presented in Section 4.2. In a nutshell, the variant exploits the RREF of $\boldsymbol{V}$ instead of the one of $\boldsymbol{H}$ by considering the dual version of the problem instance (see Section 2.3).
Recall that **STEP 2** starts with a list $\mathcal{K}$ containing all the candidates for $\sigma(\tilde{\boldsymbol{V}})_J$ (with $J = \{n - r + \tilde{r} - w + 1, \ldots, n - r + \tilde{r}\}$). By inverting σ, for each candidate $\boldsymbol{X}$ for $\sigma(\tilde{\boldsymbol{V}})_J$ (corresponding to a candidate permutation $\pi \in \mathsf{Sym}(n)$), it is possible to determine $i_1, \ldots, i_w$ s.t. $\pi(i_1) = j_1, \ldots, \pi(i_w) = j_w$ with $\sigma(j_1) = n - r + \tilde{r} - w + 1, \ldots, \sigma(j_w) = n - r + \tilde{r}$ and so we know $\pi^{-1}(j_1) = i_1, \ldots, \pi^{-1}(j_w) = i_w$. In the following, we call $\mathsf{InvCand}(\boldsymbol{X})$ the algorithm that takes in input a candidate $\boldsymbol{X}$ for $\sigma(\tilde{\boldsymbol{V}})_J$ and returns both the set $I_{\boldsymbol{X}} = \{i_1, \ldots, i_w\}$ and the corresponding candidate for $\pi^{-1}(\boldsymbol{H})_{I_{\boldsymbol{X}}}$. Notice that this candidate is unique only if $\boldsymbol{V}$ has no repeated columns, we clarify this in the next section but for now we assume this is the case.

Consider now the dual version of the problem

$$\pi^{-1}(\boldsymbol{H})\boldsymbol{V}^\top = \boldsymbol{E}^\top.$$

In this setting, for every $\boldsymbol{X} \in \mathcal{K}$, we know a candidate for $\pi^{-1}(\boldsymbol{H})_{I_{\boldsymbol{X}}}$, where in general the set $I_{\boldsymbol{X}}$ is different for every candidate $\boldsymbol{X}$. For this reason, it is not possible to build the list $\mathcal{L}$ with a *meet-in-the-middle* approach since for every candidate we need to compute the RREF w.r.t. a (possibly) different information set. In the original approach, we needed to guess the action of the permutation on $n - r - w + \tilde{r}$ more columns to have enough information to reconstruct the solution. Here, if we extend $\boldsymbol{V}$, we need to guess $n-(\ell+1)-w+\tilde{r}$ more columns of $\pi^{-1}(\boldsymbol{H})$. Thus, since we cannot use the *meet-in-the-middle* approach, for every element in $\mathcal{K}$ we try all the possible ways of choosing $n-(\ell+1)-w+\tilde{r}$ columns from the $n - w$ remaining ones.

Another difference with the original **STEP 2** in Algorithm 3 is that here we consider any $0 \leq \tilde{r} \leq \ell + 1$ instead of requiring $\tilde{r} \geq d$. The latter was required

because of the considerations made in Remark 5 which are no longer valid in our case. In particular, the constraint was to avoid having a sub-matrix with zero columns in the positions indexed by the set used to compute the RREF, but since now we are computing the RREF of $\boldsymbol{V}$ instead of $\boldsymbol{H}$, no null sub-matrices are involved.

On the probability of failure. If we denote with $\boldsymbol{V}^{\mathsf{ext}}$ the extended $\boldsymbol{V}$, we consider the probability of successfully computing $\mathsf{RREF}_{I'_{\boldsymbol{X}}}(\boldsymbol{V}^{\mathsf{ext}})$ w.r.t. a fixed set $I'_{\boldsymbol{X}} \subseteq [n]$ with $|I'_{\boldsymbol{X}}| = \ell + 1$ as the probability that a random $(\ell+1) \times (\ell+1)$ matrix is non-singular. However, we do not need to compute the RREF w.r.t. one fixed subset of indexes, as we usually have multiple possible choices. Indeed, we just need to compute the RREF w.r.t. a set $I'_{\boldsymbol{X}}$ that intersects the set $I_{\boldsymbol{X}}$ containing the indexes of the w known columns of $\pi^{-1}(\boldsymbol{H})$ in a set of cardinality $\tilde{r}$. This permits to use as much known columns as possible to reconstruct the solution. Since the possible choices of such $I'_{\boldsymbol{X}}$ are many, we can safely assume that the probability that every choice is not an information set for $\boldsymbol{V}^{\mathsf{ext}}$ is negligible. We call $\mathsf{RecoverInfSet}(\boldsymbol{X})$ the procedure that takes in input a candidate $\boldsymbol{X}$ for $\sigma(\tilde{\boldsymbol{V}})_J$ and returns a "valid" information set in the sense we just explained.

By joining the above variant of **STEP 2** with the corresponding adaptation of **STEP 3**, we obtain Algorithm 4. Note that it is not necessary (more precisely, it is not useful) to introduce the auxiliary permutation σ, since it is inverted before having played any role. Nevertheless, we still consider it in order to be consistent with the the original **STEP 2** in which σ is needed.

Remark 8. Notice that in Line 4 of Algorithm 4, we introduce an auxiliary permutation μ to simplify the description of the algorithm. This permutation is not required for the algorithm to function, but is introduced solely for clarity of exposition. Moreover, such a permutation exists because, by construction, $|\{i_1, \ldots, i_w\} \cap I'_{\boldsymbol{X}}| = \tilde{r}$.

Remark 9. Instead of taking $\boldsymbol{V}$ and $\boldsymbol{E}$ as inputs, Algorithm 4 takes the extended matrices, which we call $\boldsymbol{V}^{\mathsf{ext}}$ and $\boldsymbol{E}^{\mathsf{ext}}$ respectively. We explicitly build these matrices, for example, at Line 15 of Algorithm 7. Another observation is that Algorithm 4 could, in principle, terminate as soon as a solution is found. Nevertheless, we intentionally make it not terminate to account for instances with multiple solutions, which we encounter in the following section.

Proposition 3. *Let $\boldsymbol{H} \in \mathsf{GL}_{r,n}(\mathbb{F}_q)$, $\boldsymbol{V} \in \mathsf{GL}_{\ell+1,n}(\mathbb{F}_q)$ be matrices with no repeated columns and $\boldsymbol{E} \in \mathbb{F}_q^{r \times (\ell+1)}$. Let $w, d \in \mathbb{N}^*, \tilde{r} \in \mathbb{N}$ s.t. $w \leq n$, $0 \leq \tilde{r} \leq \ell+1$ and $N_{n,r}(w,d) > 1$. Let $\tilde{\boldsymbol{V}} = \pi(\boldsymbol{V})$ and let $\mathcal{K}$ be the list containing the candidates for $\sigma(\tilde{\boldsymbol{V}})_J$, with $J \subseteq [n]$ and $|J| = w$. Then the running time of Algorithm 4 is given by*

$$\bar{T}_{\mathcal{L}}(n, \ell, w, \tilde{r}, |\mathcal{K}|) = \begin{cases} |\mathcal{K}| \frac{(n-w)!}{(\ell+1-\tilde{r})!} & if \quad \ell + 1 - \tilde{r} \leq n - w, \\ |\mathcal{K}| & otherwise. \end{cases} \tag{14}$$

Algorithm 4 $\mathsf{DualGuessSolve}(\boldsymbol{H}, \boldsymbol{V}^{\mathsf{ext}}, \boldsymbol{E}^{\mathsf{ext}}, \mathcal{K})$

Parameters: w, $\tilde{r}$, $d \in \mathbb{N}^*$ with $0 \leq \tilde{r} \leq \ell + 1$ and $N_{n,r}(w, d) > 1$

Input: $\boldsymbol{H} \in \mathsf{GL}_{r,n}(\mathbb{F}_q)$, $\boldsymbol{V}^{\mathsf{ext}} \in \mathsf{GL}_{\ell+1,n}(\mathbb{F}_q)$, $\boldsymbol{E}^{\mathsf{ext}} \in \mathbb{F}_q^{r\times(\ell+1)}$, $\mathcal{K}$ containing all candidates for $\sigma(\tilde{\boldsymbol{V}})_J$

Output: A list $\mathcal{L}$ containing $\tilde{\boldsymbol{H}} \in \mathcal{S}_n(\boldsymbol{H})$ such that $\tilde{\boldsymbol{H}}(\boldsymbol{V}^{\mathsf{ext}})^\top = \boldsymbol{E}^{\mathsf{ext}}$

1: Initialize an empty list $\mathcal{L}$
2: **for** $\boldsymbol{X} \in \mathcal{K}$ **do**
3: $\quad \{i_1 \ldots, i_w\}, \boldsymbol{Y} \leftarrow \mathsf{InvCand}(\boldsymbol{X})$, $I'_{\boldsymbol{X}} \leftarrow \mathsf{RecoverInfSet}(\boldsymbol{X})$
4: $\quad$ Let $\mu \in \mathsf{Sym}(n)$ s.t. $\mu(I'_{\boldsymbol{X}}) = \{n-\ell, \ldots, n\}$ and

$$\mu(i_1) = n - (\ell+1) - w + \tilde{r} + 1, \ldots, \mu(i_w) = n - (\ell+1) + \tilde{r}$$

5: $\quad \boldsymbol{V}', \boldsymbol{M} \leftarrow \mathsf{RREF}^*_{\{n-\ell,\ldots,n\}}(\mu(\boldsymbol{V}^{\mathsf{ext}}))$, $\boldsymbol{U} \leftarrow \boldsymbol{V}'_{\{1,\ldots,n-(\ell+1)\}}$
6: $\quad \boldsymbol{E}' \leftarrow (\boldsymbol{E}^{\mathsf{ext}}\boldsymbol{M}^\top)_{\{1,\ldots,\tilde{r}\}}$
7: $\quad$ **for** $\boldsymbol{Z} \in \mathcal{S}_{n-(\ell+1)-w+\tilde{r}}(\boldsymbol{H}_{\neg_n\{i_1,\ldots,i_w\}})$ **do**
8: $\quad\quad \bar{\boldsymbol{V}} \leftarrow \boldsymbol{V}'^{\{1,\ldots,\tilde{r}\}}_{\{1,\ldots,n-(\ell+1)+\tilde{r}\}}$
9: $\quad\quad \boldsymbol{W} \leftarrow \begin{bmatrix}\boldsymbol{Z} & \boldsymbol{Y}\end{bmatrix}$
10: $\quad\quad$ **if** $\boldsymbol{W}\bar{\boldsymbol{V}}^\top = \boldsymbol{E}'$ **then**
11: $\quad\quad\quad \boldsymbol{W}' \leftarrow (\boldsymbol{E}^{\mathsf{ext}}\boldsymbol{M}^\top) - \boldsymbol{W}_{\{1,\ldots,n-(\ell+1)\}}\boldsymbol{U}^\top$
12: $\quad\quad\quad$ **if** $\begin{bmatrix}\boldsymbol{W}_{\{1,\ldots,n-(\ell+1)\}} & \boldsymbol{W}'\end{bmatrix} \in \mathcal{S}_n(\boldsymbol{H})$ **then**
13: $\quad\quad\quad\quad \mathcal{L} \leftarrow \mathcal{L} \cup \left\{ \mu^{-1}\left(\begin{bmatrix}\boldsymbol{W}_{\{1,\ldots,n-(\ell+1)\}} & \boldsymbol{W}'\end{bmatrix}\right)\right\}$
14: **Return:** $\mathcal{L}$

Proof. We first consider the case $\ell + 1 - \tilde{r} \leq n - w$. The cost of Line 3, because of the considerations above, can be assumed to be absorbed by the $\mathcal{O}$-notation. Therefore the overall complexity is given by $|\mathcal{K}| \cdot |\mathcal{S}_{n-(\ell+1)-w+\tilde{r}}(\boldsymbol{H}_{\neg_n\{i_1,\ldots,i_w\}})|$. Finally, if $\ell + 1 - \tilde{r} > n - w$, then, after the *filtering step*, we know enough columns to directly reconstruct the solution (which means that in the for loop at Line 7 it is not needed to compute $\boldsymbol{W}$ since we already know it). □

Remark 10. In Proposition 3 we allow $\tilde{r} = 0$, even if this means considering matrices with zero rows/columns. In this case, analogously to Remark 7, we try to reconstruct the solution directly from every candidate in $\mathcal{K}$. This observation does not affect the complexity of the algorithm given by Equation (14).

Remark 11. Note that our variant of **STEP 2** is not always an improvement w.r.t. the original version. Depending on the parameters, it might be more convenient to choose one version or the other. The two variants are compared in Table 1. Note that the last row in the table, corresponding to the dimensions of the colliding matrices, does not affect directly the running time of the algorithm as it is proposed now. However, as shown in this section, it affects the number of expected solutions. This affects the running time of our newly proposed Algorithm 7, see Lemma 2.

	Original STEP 2	Our variant
Amount of extra guesses	$n-r+\tilde{r}-w$	$n-(\ell+1)+\tilde{r}-w$
Constraint on $\tilde{r}$	$d \leq \tilde{r} \leq r$	$0 \leq \tilde{r} \leq \ell+1$
Meet-in-the-middle	✓	×
Dimensions of colliding matrices	$\ell \times (\tilde{r}-d)$	$\tilde{r} \times (r-1)$

Table 1: Comparison of the two variants. Red cells indicate the variant that results in higher complexity for each respective parameter.

4.2 Variant of STEP 1: Recursive Filtering

We introduce here a variant for the *filtering step* of SBC (**STEP 1** in Algorithm 3). The idea is to replace the *meet-in-the-middle* approach by solving multiple IPKP (sub)instances. Recall that the goal of the *filtering step* is to have a list $\mathcal{K} = \left\{\boldsymbol{X} \in \mathcal{S}_w(\boldsymbol{V}) \mid \boldsymbol{X}\bar{\boldsymbol{H}}^\top = \boldsymbol{E}^*\right\}$ containing candidates for $\sigma(\tilde{\boldsymbol{V}})_J$ (see Equation (9)). For every $W \subseteq [n]$, with $|W| = w$, we populate the list $\mathcal{K}$ with elements of the form $\tilde{\pi}(\boldsymbol{V}_W)$, for some $\tilde{\pi} \in \mathsf{Sym}(w)$, such that

$$\tilde{\pi}(\boldsymbol{V}_W)\bar{\boldsymbol{H}}^\top = \boldsymbol{E}^*, \tag{15}$$

which is a (w, d, q, ℓ)-IPKP instance. In total, that are $\binom{n}{w}$ possible choices for W. Notice that, for simplicity, we abuse the terminology of Definition 2 and refer to IPKP instances also in cases they have no solution. This can occur when the chosen set W is not the one containing the columns of $\sigma(\tilde{\boldsymbol{V}})_J$.

We only consider IPKP instances in their dual form (see Equation (6)), as our complexity analysis shows this to be consistently the best choice. To increase the row dimension of the instances by one, we extend $\boldsymbol{V}_W$, which in the considered dual instances plays the role of the parity-check, by adding an additional row of ones analogously to Section 2.2. This means that $\mathcal{K}$ collects solutions of IPKP instances with parameters $(w, \ell+1, q, d)$. We highlight that these subinstances might also have multiple solutions. In order to solve them, we use a slightly modified version of SBC.

Remark 12. To be precise, the subinstances to solve (after having extended $\boldsymbol{V}_W$) have parameters $(w, \mathsf{rank}(\boldsymbol{V}_W)+1, q, d)$. In the following, we assume that $\mathsf{rank}(\boldsymbol{V}_W) = \min\{\ell, w-d+1\}$. We motivate this in Appendix A.

Dealing with instances with multiple solutions. Let $\ell^* := \mathsf{rank}(\boldsymbol{V}_W)$. In most cases, the subinstances under consideration have multiple solutions that need to be found all. If $\bar{\boldsymbol{H}}$ has no repeated columns, we expect multiple solutions to occur whenever condition iv) in Section 2.1 does not hold (that is, when $\frac{w!}{q^{\ell^* d}} \geq 1$). In this case, Algorithm 3 can be easily adapted to return all solutions: it suffices to output all valid permutations instead of terminating as soon as one is found (as in Lines 13–14 of Algorithm 4). We define SBC* (Algorithm 5)

and $\mathsf{SBC}^*_{\mathsf{dual}}$ (Algorithm 6) as the variants of SBC (Algorithm 3) that take this into account, performing respectively the original **STEP 2-3** of Algorithm 3 and Algorithm 4. Suppose we are solving the instance $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$. Differently from Algorithm 3, Algorithm 5 and Algorithm 6 return lists containing, respectively, all $\tilde{\boldsymbol{V}} \in \mathcal{S}_n(\boldsymbol{V})$ such that $\tilde{\boldsymbol{V}}\boldsymbol{H}^\top = \boldsymbol{E}$, and all $\tilde{\boldsymbol{H}} \in \mathcal{S}_n(\boldsymbol{H})$ such that $\tilde{\boldsymbol{H}}(\boldsymbol{V}^{\mathsf{ext}})^\top = \boldsymbol{E}^{\mathsf{ext}}$, where $\boldsymbol{V}^{\mathsf{ext}}$ is the matrix obtained by extending $\boldsymbol{V}$ with a row of ones and $\boldsymbol{E}^{\mathsf{ext}}$ is the matrix obtained by extending $\boldsymbol{E}^\top$ accordingly. We explicit these algorithms in Algorithms 5 and 6.

Algorithm 5 $\mathsf{SBC}^*(\boldsymbol{H}, \boldsymbol{V}, \boldsymbol{E})$

Parameters: w_1, w_2, $\tilde{r}$, $d \in \mathbb{N}$ with $w = w_1 + w_2 \leq n$, $d \leq \tilde{r} \leq r$ and $N_{n,r}(w, d) > 1$
Input: $\boldsymbol{H} \in \mathsf{GL}_{r,n}(\mathbb{F}_q)$, $\boldsymbol{V} \in \mathsf{GL}_{\ell,n}(\mathbb{F}_q)$, $\boldsymbol{E} \in \mathbb{F}_q^{\ell \times r}$
Output: $\pi \in \mathsf{Sym}(n)$ such that $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$

Proceed as in Lines 1-14 of Algorithm 3
Initialize an empty list $\mathcal{T}$
for $\boldsymbol{X} \in \mathcal{L}$ **do**
  $\boldsymbol{X}' \leftarrow \bar{\boldsymbol{E}} - \boldsymbol{X}_{\{1,\dots,n-r\}}\bar{\boldsymbol{U}}^\top$
  if $\left[\boldsymbol{X}\ \boldsymbol{X}'\right] \in \mathcal{S}_n(\boldsymbol{V})$ **then**
    $\mathcal{T} \leftarrow \mathcal{T} \cup \left\{\sigma^{-1}\left(\left[\boldsymbol{X}\ \boldsymbol{X}'\right]\right)\right\}$
    Return: $\mathcal{T}$

Algorithm 6 $\mathsf{SBC}^*_{\mathsf{dual}}(\boldsymbol{H}, \boldsymbol{V}, \boldsymbol{E})$

Parameters: w_1, w_2, $\tilde{r}$, $d \in \mathbb{N}$ with $w = w_1 + w_2 \leq n$, $d \leq \tilde{r} \leq r$ and $N_{n,r}(w, d) > 1$
Input: $\boldsymbol{H} \in \mathsf{GL}_{r,n}(\mathbb{F}_q)$, $\boldsymbol{V} \in \mathsf{GL}_{\ell,n}(\mathbb{F}_q)$, $\boldsymbol{E} \in \mathbb{F}_q^{\ell \times r}$
Output: $\pi \in \mathsf{Sym}(n)$ such that $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$

Proceed as in Lines 1-7 of Algorithm 3
$\boldsymbol{V}^{\mathsf{ext}} \leftarrow \begin{bmatrix} \boldsymbol{V} \\ (1, \dots, 1) \end{bmatrix}$, $\boldsymbol{E}^{\mathsf{ext}} \leftarrow \left[\boldsymbol{E}^\top \sum_{i=1}^n \boldsymbol{h}_i\right]$
Return: $\mathsf{DualGuessSolve}(\boldsymbol{H}, \boldsymbol{V}^{\mathsf{ext}}, \boldsymbol{E}^{\mathsf{ext}}, \mathcal{K})$

Allowing matrices to have repeated columns. We must solve IPKP instances with parameters $(w, \ell^* + 1, q, d)$ obtained by considering the dual version of Equation (15), that is

$$\tilde{\pi}^{-1}(\bar{\boldsymbol{H}})\boldsymbol{V}_W^\top = \boldsymbol{E}^{*\top}.$$

We recall that $\bar{\boldsymbol{H}}$ is the matrix formed by the nonzero columns of the generator matrix of a random subcode of the code generated by $\boldsymbol{H}$, with dimension d and support size w (so $d < w$). We model $\bar{\boldsymbol{H}}$ as a random element of $\mathsf{GL}_{d,w}(\mathbb{F}_q)$

with no zero columns, meaning that there might be repeated columns. If this is the case, multiple solutions to the instance necessarily exist. Let us suppose, for example, that $\bar{\boldsymbol{H}}$ has 2 identical columns and all the others are distinct. If this happens and $\tilde{\pi}^{-1} \in \mathsf{Sym}(w)$ is a solution, another solution $\tilde{\mu} \in \mathsf{Sym}(w)$ can be found by swapping the images under $\tilde{\pi}^{-1}$ of the indexes corresponding to the repeating columns in $\bar{\boldsymbol{H}}$.

Definition 3. *If $\boldsymbol{H} \in \mathbb{F}_q^{r \times n}$ is a matrix with no zero columns, we define $\mathsf{COL}_{\mathsf{REP}}(\boldsymbol{H}) = (m_1, \ldots, m_{q^r-1}) \in \mathbb{N}^{q^r-1}$, where m_j is the number of times that the j-th element of $(\mathbb{F}_q^r)^*$ (ordered w.r.t. any ordering) appears among the columns of $\boldsymbol{H}$. We define the integer $N_{\mathsf{swaps}}(\boldsymbol{H})$ as*

$$N_{\mathsf{swaps}}(\boldsymbol{H}) = \prod_{i=1}^{q^r-1} m_i! \tag{16}$$

Proposition 4. *If $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$, then from $\pi \in \mathsf{Sym}(n)$ (or equivalently from $\pi(\boldsymbol{V})$) we can obtain, performing only swaps, $N_{\mathsf{swaps}}(\boldsymbol{H})$ solutions to the problem (including π).*

Proof. Every swap of the permutation indices corresponding to two repeating columns gives a new distinct solution. Equation (16) counts the number of all possible swaps combined. □

Proposition 5. *Let $\boldsymbol{H} \in \mathbb{F}_q^{r \times n}$ be a matrix with no zero columns and with $\mathsf{COL}_{\mathsf{REP}}(\boldsymbol{H}) = (m_1, \ldots, m_{q^r-1})$. Let $x \leq n$ be a positive integer and let*

$$J_{n,r,q}^{(x)} = \left\{ (\mu_1, \ldots, \mu_{q^r-1}) \in \mathbb{N}^{q^r-1} \;\mid\; \mu_1 + \cdots + \mu_{q^r-1} = x,\ 0 \leq \mu_i \leq m_i \right\}.$$

Then

$$|\mathcal{S}_x(\boldsymbol{H})| = \sum_{(\mu_1, \ldots, \mu_{q^r-1}) \in J_{n,r,q}^{(\ell)}} \frac{x!}{\mu_1! \cdots \mu_{q^r-1}!}. \tag{17}$$

Proof. $|\mathcal{S}_x(\boldsymbol{H})|$ is given by the possible ways of choosing ordered collections of x elements from n with no replacement, but some of them might be repeated. This follows the multinomial distribution, which gives us Equation (17). Notice that if $\boldsymbol{H}$ has no repeated columns, $\mathsf{COL}_{\mathsf{REP}}(\boldsymbol{H})$ has n entries equal to 1 and the rest of them equal to 0, so $|J_{n,r,q}^{(x)}| = \binom{n}{x}$, which implies

$$\sum_{(\mu_1, \ldots, \mu_{q^r-1}) \in J_{n,r,q}^{(x)}} \frac{x!}{\mu_1! \cdots \mu_{q^r-1}!} = \binom{n}{x} x! = \frac{n!}{(n-x)!},$$

which is what we expected. □

Remark 13. Since $\bar{\boldsymbol{H}}$ is modeled as random, we assume that its columns are uniformly distributed over $(\mathbb{F}_q^d)^*$, that is $\mathsf{COL}_{\mathsf{REP}}(\bar{\boldsymbol{H}}) = (u_1, \ldots, u_{q^d-1})$, where

$(u_1, \dots, u_{q^d-1}) \in \mathbb{N}^{q^d-1}$ is an element that minimizes the Euclidean distance from $(\frac{w}{q^d-1}, \dots, \frac{w}{q^d-1})$ in the set $\{(v_1, \dots, v_{q^d-1}) \in \mathbb{N}^{q^d-1} \mid \sum_{i=1}^{q^d-1} v_i = w\}$. Notice that this element is not unique: if $(u_1, \dots, u_{q^d-1})$ satisfies these requirements, then any permutation of it does as well. To justify this assumption, we model the distribution of each entry of $\mathsf{COL}_{\mathsf{REP}}(\bar{\boldsymbol{H}})$ as a binomial random variable with parameters (w, p), where $p = \frac{1}{q^d-1}$ is the probability of selecting a specific column (the probability of "success"). Hence, each entry has expected value $w \cdot p = \frac{w}{q^d-1}$, and by requiring every entry to be an integer, we obtain the above estimate.

Let $\mathsf{COL}_{\mathsf{REP}}{}^{(u)}(n, r, q)$ denote the vector $(u_1, \dots, u_{q^r-1}) \in \mathbb{N}^{q^r-1}$ as in Remark 13 and let $N^{(u)}_{\mathsf{swaps}}(n, r, q) \coloneqq \prod_{i=1}^{q^r-1} u_i!$. We also define $\mathsf{M}_{\mathsf{REP}}(n, r, q, x)$ as $|\mathcal{S}_x(\boldsymbol{H})|$, where $\boldsymbol{H} \in \mathbb{F}_q^{r \times n}$ is random with no zero columns, and therefore $\mathsf{M}_{\mathsf{REP}}(n, r, q, x)$ is computed using *Proposition* 5 with $\mathsf{COL}_{\mathsf{REP}}(\boldsymbol{H}) = \mathsf{COL}_{\mathsf{REP}}{}^{(u)}(n, r, q)$.

The next proposition presents the average running time of SBC^* (Algorithm 5) and $\mathsf{SBC}^*_{\mathsf{dual}}$ (Algorithm 6) in the case where $\boldsymbol{V}$ may contain repeated columns. Notice that, in our setting, the matrix that may have repeated columns is $\bar{\boldsymbol{H}}$; however, since we are solving the dual instance, it plays the role of $\boldsymbol{V}$ in Algorithm 5 and Algorithm 6.

Proposition 6. *Let $\boldsymbol{H} \in \mathsf{GL}_{r,n}(\mathbb{F}_q)$ with no repeated columns and s.t. its last row is a vector of all ones, $\boldsymbol{V} \in \mathsf{GL}_{\ell,n}(\mathbb{F}_q)$ be a matrix that might have repeated columns but no zero columns (and $\boldsymbol{H}$, $\boldsymbol{V}$ can be modeled as a random matrix with these properties) and $\boldsymbol{E} \in \mathbb{F}_q^{\ell \times r}$. Let now $w_1, w_2, d \in \mathbb{N}^*$ be valid parameters for SBC^*, that is $w = w_1 + w_2 \leq n$ and $N_{n,r}(w, d) > 1$. Then Algorithm 5 runs for $d \leq \tilde{r} \leq r$ in average time*

$$T_{\mathsf{SBC}^*}(n, r, q, \ell, w_1, w_2, d, \tilde{r}) = T^{(d)}_{\mathsf{ISD}}(n, r, w) + T^*_{\mathcal{K}} + T^*_{\mathcal{L}} + |\mathcal{L}^*|, \tag{18}$$

and Algorithm 6 runs for $0 \leq \tilde{r} \leq \ell + 1$ in average time

$$T_{\mathsf{SBC}^*_{\mathsf{dual}}}(n, r, q, \ell, w_1, w_2, d, \tilde{r}) = T^{(d)}_{\mathsf{ISD}}(n, r, w) + T_{\mathcal{K}} + \bar{T}_{\mathcal{L}}, \tag{19}$$

with

$$\begin{aligned}
T^*_{\mathcal{K}} &\coloneqq \sum_{\bar{w} \in \{w_1, w_2\}} \mathsf{M}_{\mathsf{REP}}(n, \ell, q, \bar{w}) + \frac{1}{q^{d\ell}} \cdot \prod_{\bar{w} \in \{w_1, w_2\}} \mathsf{M}_{\mathsf{REP}}(n, \ell, q, \bar{w}), \\
T^*_{\mathcal{L}} &\coloneqq \mathsf{M}_{\mathsf{REP}}(n, \ell, q, n - r - w + \tilde{r}) + |\mathcal{K}^*| + \frac{\mathsf{M}_{\mathsf{REP}}(n, \ell, q, n - r - w + \tilde{r}) \cdot |\mathcal{K}^*|}{q^{\ell(\tilde{r}-d)}}, \\
\bar{T}_{\mathcal{L}} &\coloneqq \bar{T}_{\mathcal{L}}(n, \ell, w, \tilde{r}, |\bar{\mathcal{K}}|) \text{ as in Equation (14)}, \\
|\mathcal{K}^*| &\coloneqq \frac{\mathsf{M}_{\mathsf{REP}}(n, \ell, q, w)}{q^{d\ell}}, \qquad |\bar{\mathcal{K}}| \coloneqq |\mathcal{K}^*| \cdot N^{(u)}_{\mathsf{swaps}}(w, \ell, q), \\
|\mathcal{L}^*| &\coloneqq \frac{\mathsf{M}_{\mathsf{REP}}(n - w, \ell, q, n - r - w + \tilde{r}) \cdot |\mathcal{K}^*|}{q^{\ell(\tilde{r}-d)}}.
\end{aligned}$$

Proof. The running time of Algorithm 5 is obtained in the same way as for Algorithm 3. Thus, we obtain Equation (18) by adapting Proposition 2 to the case in which $\boldsymbol{V}$ has repeated columns. Indeed, we obtain $T^*_{\mathcal{K}}$, $T^*_{\mathcal{L}}$, $|\mathcal{K}^*|$ and $|\mathcal{L}^*|$ as in Proposition 2 but we estimate $\mathcal{S}_x(\boldsymbol{V})$ with Equation (17) instead of $\frac{n!}{(n-x)!}$. The same reasoning also applies to Algorithm 6, but to prove Equation (19) we still need to clarify $|\bar{\mathcal{K}}|$. As already mentioned, if $\boldsymbol{V}$ has repeated columns, then $\mathsf{InvCand}$ returns more than one candidate. Since $\boldsymbol{V}$ is modeled as a random matrix, we have that every candidate $\boldsymbol{X} \in \mathcal{K}^*$ for $\sigma(\tilde{\boldsymbol{V}}_J)$ (with $|J| = w$), where $\mathcal{K}^*$ is the list obtained after the *filtering step*, generates $N^{(u)}_{\mathsf{swaps}}(w, l, q)$ candidates for $\pi^{-1}(\boldsymbol{H})_I$ (the set I is possibly different for every $\boldsymbol{X}$). Therefore, Lines 7-14 in Algorithm 4 are executed $|\mathcal{K}^*| \cdot N^{(u)}_{\mathsf{swaps}}(w, l, q)$ times, which gives the list cardinality. □

Proposition 7. *Under the same hypothesis and notation as in Proposition 6, with* $\boldsymbol{E} \in \mathbb{F}_q^{\ell \times r}$ *being of the form* $\boldsymbol{E} = \left[\boldsymbol{0}_{\ell \times (r-1)} \sum_{i=1}^{n} \boldsymbol{v}_i\right]$*, we have that the average number of solutions returned respectively by Algorithm 5 and Algorithm 6 is given by*

$$N_{\mathsf{SBC}^*}(n, r, q, \ell, w_1, w_2, d, \tilde{r}) = |\mathcal{L}^*| \cdot \min\left\{\frac{(r-\tilde{r})!}{q^{\ell(r-\tilde{r})} N^{(u)}_{\mathsf{swaps}}(r-\tilde{r}, \ell, q)}, 1\right\},$$
$$N_{\mathsf{SBC}^*_{\mathsf{dual}}}(n, r, q, \ell, w_1, w_2, d, \tilde{r}) = \frac{|\bar{\mathcal{K}}|(n-(w_1+w_2))!}{(\ell+1-\tilde{r})! q^{\tilde{r}(r-1)}} \cdot \min\left\{\frac{(\ell+1-\tilde{r})!}{q^{(r-1)(\ell+1-\tilde{r})}}, 1\right\}. \tag{20}$$

Proof. The number of solutions returned by Algorithm 5 is given by the number of elements in the list $\mathcal{L}$ that satisfy the condition at Line 5. Since, with the same notation as in Algorithm 5, $\boldsymbol{X}'$ can be modeled as a random $\ell \times (r-\tilde{r})$ matrix, then such condition is met with probability $\min\left\{\frac{|\mathcal{S}_{r-\tilde{r}}(\boldsymbol{V}_U)|}{q^{\ell(r-\tilde{r})}}, 1\right\}$ where U is the set of the indexes of the columns of $\boldsymbol{V}$ that are not columns of $\boldsymbol{X}$ (so $|U| = r - \tilde{r}$), therefore $|\mathcal{S}_{r-\tilde{r}}(\boldsymbol{V}_U)| = \frac{(r-\tilde{r})!}{N^{(u)}_{\mathsf{swaps}}(r-\tilde{r}, \ell, q)}$.
On the other hand, the number of solutions returned by $\mathsf{SBC}^*_{\mathsf{dual}}$ is given by the number of candidates $\boldsymbol{W}$ of the form given at Line 9 of Algorithm 4 that satisfy both the condition at Line 10 and the condition at Line 12. The probability of the first condition to be met is the probability of a collision between two random $(r-1) \times \tilde{r}$ matrices. Considering the notation of Algorithm 4, the equality at Line 10 is checked for the matrices $\boldsymbol{W}\boldsymbol{V}'^{\top}$ and $\boldsymbol{E}'$, which are actually $r \times \tilde{r}$. However, the last row of such matrices is not random at all and, indeed, it always collides. To see this, let us recall that the last row of $\boldsymbol{H}$, and so the last one of $\boldsymbol{W}$, is a row of ones. On the other hand, the matrix $\boldsymbol{E}'$ is of the form

$$\boldsymbol{E}' = \boldsymbol{E}^{\mathsf{ext}}(\boldsymbol{M}^{\top})_{\{1,\dots,\tilde{r}\}} = \begin{bmatrix} \boldsymbol{0}_{(r-1)\times\ell} & \sum_{i=1}^{n} \boldsymbol{h}_i \\ (\sum_{i=1}^{n} \boldsymbol{v}_i)^{\top} & \end{bmatrix} (\boldsymbol{M}^{\top})_{\{1,\dots,\tilde{r}\}},$$

since $\boldsymbol{E}^{\mathsf{ext}}$ is as in Line 2 of Algorithm 6. Notice that the last entry of the last row of $\boldsymbol{E}^{\mathsf{ext}}$ is given by $n \mod q$ (which we denote with n_q). If we denote

$\mu(\boldsymbol{V}^{\mathsf{ext}})$ with $\tilde{\boldsymbol{V}}_\mu^{\mathsf{ext}}$, we have that $\bar{\boldsymbol{V}}^\top = (\tilde{\boldsymbol{V}}_\mu^{\mathsf{ext}\top})^{\{1,\dots,n-(\ell+1)+\tilde{r}\}}(\boldsymbol{M}^\top)_{\{1,\dots,\tilde{r}\}}$ and, since the last row of $\boldsymbol{W}$ is a row of ones, the last row of $\boldsymbol{W}\bar{\boldsymbol{V}}^\top$ is given by $\left(\sum_{i=1}^{n-(\ell+1)+\tilde{r}}(\tilde{\boldsymbol{v}}_\mu^{\mathsf{ext}})_i\right)^\top (\boldsymbol{M}^\top)_{\{1,\dots,\tilde{r}\}}$. On the other hand, the last row of $\boldsymbol{E}'$ is given by

$$\begin{aligned}\left[\textstyle\sum_{i=1}^{n}\boldsymbol{v}_i\ n_q\right](\boldsymbol{M}^\top)_{\{1,\dots,\tilde{r}\}} &= \left(\sum_{i=1}^{n}(\tilde{\boldsymbol{v}}_\mu^{\mathsf{ext}})_i\right)^\top (\boldsymbol{M}^\top)_{\{1,\dots,\tilde{r}\}} \\ &= \left(\sum_{i=1}^{n-(\ell+1)+\tilde{r}}(\tilde{\boldsymbol{v}}_\mu^{\mathsf{ext}})_i\right)^\top (\boldsymbol{M}^\top)_{\{1,\dots,\tilde{r}\}},\end{aligned}$$

where the last equality follows from the fact that, by construction, $\left(\tilde{\boldsymbol{v}}_\mu^{\mathsf{ext}}\right)_i^\top (\boldsymbol{M}^\top)_{\{1,\dots,\tilde{r}\}} = \boldsymbol{0}$ for every i s.t. $n-(\ell+1)+\tilde{r}+1 \le i \le n$.
To prove Equation (20), it remains to estimate the probability of the condition at Line 12 to be met. This is the probability of the matrix $\boldsymbol{W}'$ having as columns those of $\boldsymbol{H}$ that do not appear in $\boldsymbol{W}_{\{1,\dots,n-(\ell+1)\}}$. This event happens for sure for $\tilde{r}$ columns (because $\boldsymbol{W}$ satisfies the condition at Line 10), while the other $\ell+1-\tilde{r}$ columns can be considered random. Therefore, the probability is given by $\min\left\{\frac{|\mathcal{S}_{\ell+1-\tilde{r}}(\boldsymbol{H}_U)|}{q^{(r-1)(\ell+1-\tilde{r})}}, 1\right\}$, where U is the set of columns of $\boldsymbol{H}$ that don't appear in $\boldsymbol{W}$. Since $\boldsymbol{H}$ has no repeated columns, $|\mathcal{S}_{\ell+1-\tilde{r}}(\boldsymbol{H}_U)| = (\ell+1-\tilde{r})!$. Notice that again we consider collision between random matrices with $r-1$ rows, since, analogously as above, one can show that the last row of $\boldsymbol{W}'$ is always a row of ones. □

Remark 14. We showed in Proposition 7 how extending the matrix $\boldsymbol{H}$ in Algorithm 6 does not help to reduce the number of collisions. The same reasoning applies to the matrix $\boldsymbol{V}$ in Algorithm 5, hence it is not useful to extend it. Moreover, one could observe that in Algorithm 5 and Algorithm 6 the matrices that are being checked for collisions have a column deriving from the extension of $\boldsymbol{H}$ and $\boldsymbol{V}$, respectively, but they are still considered random. Indeed, we consider the matrices at Line 5 in Algorithm 5 to be random $\ell \times (r-\tilde{r})$ matrices instead of $\ell \times (r-1-\tilde{r})$ (the same applies to Algorithm 6 with $(r-1)\times(\ell+1-\tilde{r})$ instead of $(r-1)\times(\ell-\tilde{r})$). We can do this because the "non-random" column appears in a matrix that is being multiplied on the right by another matrix, so the resulting product cannot be split as it was done Proposition 7 for the last row: roughly speaking, the "row-times-columns" multiplication scrambles the column-structure of the matrix on the left. Let us consider the matrix $\boldsymbol{X}'$ in Algorithm 5 (for Algorithm 6 the reasoning is analogous), which is obtained as $\bar{\boldsymbol{E}} - \boldsymbol{X}_{\{1,\dots,n-r\}}\bar{\boldsymbol{U}}^\top$. We have that

$$\begin{aligned}\boldsymbol{X}_{\{1,\dots,n-r\}}\bar{\boldsymbol{U}}^\top &= \boldsymbol{X}_{\{1,\dots,n-r\}}\left((\boldsymbol{M}\sigma(\boldsymbol{H}))_{\{1,\dots,n-r\}}^{\{\tilde{r}+1,\dots,r\}}\right)^\top \\ &= \boldsymbol{X}_{\{1,\dots,n-r\}}\left(\sigma(\boldsymbol{H})^\top\right)^{\{1,\dots,n-r\}}\boldsymbol{M}_{\{\tilde{r}+1,\dots r\}}^\top \\ &= \left[\boldsymbol{A} \quad \textstyle\sum_{i=1}^{n-r}\boldsymbol{x}_i\right]\boldsymbol{M}_{\{\tilde{r}+1,\dots r\}}^\top,\end{aligned}$$

for some $\boldsymbol{A} \in \mathbb{F}_q^{\ell \times (r-1)}$. Since $\bar{\boldsymbol{E}} = \boldsymbol{E}\boldsymbol{M}^\top_{\{\tilde{r}+1,\dots,r\}}$, we have that

$$\begin{aligned}\boldsymbol{X}' &= \bar{\boldsymbol{E}} - \boldsymbol{X}_{\{1,\dots,n-r\}}\bar{\boldsymbol{U}}^\top = \left(\begin{bmatrix}\boldsymbol{0} & \sum_{i-1}^{n} \boldsymbol{v}_i\end{bmatrix} - \begin{bmatrix}\boldsymbol{A} & \sum_{i=1}^{n-r} \boldsymbol{x}_i\end{bmatrix}\right)\boldsymbol{M}^\top_{\{\tilde{r}+1,\dots r\}} \\ &= \begin{bmatrix}-\boldsymbol{A} & \sum_{i=1}^{r} \bar{\boldsymbol{x}}_i\end{bmatrix}\boldsymbol{M}^\top_{\{\tilde{r}+1,\dots r\}}, \end{aligned} \tag{21}$$

where $\{\bar{\boldsymbol{x}}_i\}_{i=1,\dots r}$ is the set of the columns of $\boldsymbol{V}$ that do not appear in $\boldsymbol{X}$. Note that these are the columns of $\boldsymbol{X}'$ if $\boldsymbol{X}$ is the candidate that leads to the solution. However, we cannot express the matrix resulting from the product in Equation (21) in term of the last column of the matrix on the left, which is the one that we just showed to be not random. Therefore, for the sake of our analysis, we consider this matrix to look random.
Finally, notice that a similar reasoning can be applied when checking for collisions at Line 13 of Algorithm 3 (when using Algorithm 5) and at Line 10 of Algorithm 4 (when using Algorithm 6), for the case in which, respectively, $\tilde{r} = r$ and $\tilde{r} = \ell+1$. Indeed, in these cases we consider collisions between random $\ell \times (r-d)$ and $(r-1) \times (\ell+1)$ matrices (instead of between $\ell \times (r-1-d)$ and $(r-1) \times \ell$ matrices), respectively. Notice that the above considerations apply only when the IPKP instance arises from the dehomogenization of a PKP instance, which is why this assumption is required in Proposition 7. For generic IPKP instances, the algorithm behaves identically, but the number of solutions can be estimated without these considerations, resulting in slightly smaller complexity estimates.

Remark 15. In Proposition 6 we do not require the existence of a solution. We do this since we use this proposition to estimate the complexity of solving the subinstances, which are not required to have a solution, except for one of them. Therefore to estimate the running time of SBC^* and $\mathsf{SBC}^*_{\mathsf{dual}}$ applied on $\binom{n}{w} - 1$ subinstances we use respectively Equation (18) and Equation (19), while in one subinstance we use $T^{(1)}_{\mathsf{SBC}^*}$ and $T^{(1)}_{\mathsf{SBC}^*_{\mathsf{dual}}}$. These are obtained in the same way as in Proposition 7, but taking $|\mathcal{K}^*| = \max\left\{\frac{\mathsf{M}_{\mathsf{REP}}(n,\ell,q,w)}{q^{d\ell}}, 1\right\}$ instead of taking it as in Proposition 6. Analogously, we define $N^{(1)}_{\mathsf{SBC}^*}$ and $N^{(1)}_{\mathsf{SBC}^*_{\mathsf{dual}}}$.

4.3 New algorithm

We can now define our complete algorithm for solving IPKP, which incorporates all the modifications described above. The algorithm takes two additional Boolean parameters, $\mathsf{v}_{\mathsf{inn}}$ and $\mathsf{v}_{\mathsf{out}}$, which specify whether Algorithm 4 is used respectively in the subinstances and in the outer instance. Here, the algorithm $\mathsf{InvCand}^*(\boldsymbol{X})$ takes as input $\boldsymbol{X} \in \mathcal{S}_w(\bar{\boldsymbol{H}})$, a solution to the subinstance (that is, using the notation of the algorithm, $\boldsymbol{X}(\boldsymbol{W}^{\mathsf{ext}})^\top = \boldsymbol{E}^{*\mathsf{ext}}$), and returns all corresponding candidates for $\tilde{\boldsymbol{W}}^{\mathsf{ext}}$. In general, there is more than one such candidate, since $\bar{\boldsymbol{H}}$ typically contains repeated columns.

Lemma 1. *Let $\boldsymbol{H} \in \mathsf{GL}_{r,n}$, $\boldsymbol{V} \in \mathsf{GL}_{\ell,n}$ and $\boldsymbol{E} \in \mathbb{F}_q^{\ell \times r}$ be matrices that define an IPKP-instance obtained by de-homogenizing an $(n, r-1, q, \ell) - \mathsf{PKP}$ instance*

Algorithm 7 $\mathsf{NewAlgorithm}(\boldsymbol{H}, \boldsymbol{V}, \boldsymbol{E})$

Parameters: $\mathsf{v}_{\mathsf{inn}}$, $\mathsf{v}_{\mathsf{out}} \in \{0,1\}$, w, $\tilde{r}$, $d \in \mathbb{N}$ with $w \leq n$, $N_{n,r}(w,d) > 1$ and $0 \leq \tilde{r} \leq \ell + 1$ **if** $\mathsf{v}_{\mathsf{out}}$ **else** $d \leq \tilde{r} \leq r$.
Input: $\boldsymbol{H} \in \mathsf{GL}_{r,n}(\mathbb{F}_q)$, $\boldsymbol{V} \in \mathsf{GL}_{\ell,n}(\mathbb{F}_q)$, $\boldsymbol{E} \in \mathbb{F}_q^{\ell \times r}$,
Output: $\pi \in \mathsf{Sym}(n)$ such that $\pi(\boldsymbol{V})\boldsymbol{H}^\top = \boldsymbol{E}$

1: Use ISD to find a subcode of the code generated by $\boldsymbol{H}$ with dimension d and support size $w = w_1 + w_2$. Call $\mathcal{C}$ this subcode, $\boldsymbol{H}^*$ its generator matrix and $\boldsymbol{S} \in \mathsf{GL}_{d,r}(\mathbb{F}_q)$ the matrix s.t. $\boldsymbol{H}^* = \boldsymbol{S}\boldsymbol{H}$.
2: Compute $\sigma \in \mathsf{Sym}(n)$ such that $\mathrm{Supp}(\sigma(\mathcal{C})) = \{n-r+\tilde{r}-w+1, \ldots, n-r+\tilde{r}\}$
3: $J \leftarrow \{n-r+\tilde{r}-w+1, \ldots, n-r+\tilde{r}\}$
4: $\boldsymbol{E}^* \leftarrow \boldsymbol{E}\boldsymbol{S}^\top$, $\bar{\boldsymbol{H}} \leftarrow \sigma(\boldsymbol{H}^*)_J$
5: Initialize an empty list $\mathcal{K}$
6: **for** $\boldsymbol{W} \in \bar{\mathcal{S}}_w(\boldsymbol{V})$ **do**
7: $\quad \boldsymbol{W}^{\mathsf{ext}} \leftarrow \begin{bmatrix} \boldsymbol{W} \\ (1,\ldots,1) \end{bmatrix}$, $\boldsymbol{E}^{*\mathsf{ext}} \leftarrow \left[\boldsymbol{E}^{*\top} \sum_{i=1}^{w} \bar{\boldsymbol{h}}_i\right]$
8: $\quad$ **if** $\mathsf{v}_{\mathsf{inn}}$ **then**
9: $\quad\quad \mathcal{K} \leftarrow \mathcal{K} \cup \mathsf{SBC}^*_{\mathsf{dual}}(\boldsymbol{W}^{\mathsf{ext}}, \bar{\boldsymbol{H}}, \boldsymbol{E}^{*\mathsf{ext}})$
10: $\quad$ **else**
11: $\quad\quad \mathcal{K}^* \leftarrow \mathsf{SBC}^*(\boldsymbol{W}^{\mathsf{ext}}, \bar{\boldsymbol{H}}, \boldsymbol{E}^{*\mathsf{ext}})$
12: $\quad\quad$ **for** $\boldsymbol{X} \in \mathcal{K}^*$ **do**
13: $\quad\quad\quad \mathcal{K} \leftarrow \mathcal{K} \cup \mathsf{InvCand}^*(\boldsymbol{X})$
14: **if** $\mathsf{v}_{\mathsf{out}}$ **then**
15: $\quad \boldsymbol{V}^{\mathsf{ext}} \leftarrow \begin{bmatrix} \boldsymbol{V} \\ (1,\ldots,1) \end{bmatrix}$, $\boldsymbol{E}^{\mathsf{ext}} \leftarrow \left[\boldsymbol{E}^{\top} \sum_{i=1}^{n} \boldsymbol{h}_i\right]$
16: $\quad \tilde{\boldsymbol{H}} \leftarrow \mathsf{DualGuessSolve}(\boldsymbol{H}, \boldsymbol{V}^{\mathsf{ext}}, \boldsymbol{E}^{\mathsf{ext}}, \mathcal{K})$
17: $\quad$ Compute $\pi \in \mathsf{Sym}(n)$ s.t. $\pi(\boldsymbol{H}) = \tilde{\boldsymbol{H}}$
18: $\quad$ **Return:** π^{-1}
19: **else**
20: $\quad$ Proceed as in Lines 8-19 of Algorithm 3

that has a solution. Let $\mathsf{v}_{\mathsf{inn}} \in \{0,1\}$ *be a boolean parameter that specifies whether to use Algorithm 6 or Algorithm 5 in the (inner) subinstances. Let* $w, d \in \mathbb{N}^*$ *be valid parameters for the outer instance, that is* $w \leq n$ *and* $N_{n,r}(w,d) > 1$*. Let* $\ell^* = \min\{\ell, w-d+1\}$ *and let* $w_1^*, w_2^*, d^*, \tilde{r}^*$ *be valid parameters for the subinstances, that is* $w^* = w_1^* + w_2^* \leq w$*,* $N_{w,\ell^*+1}((w^*, d^*) > 1$ *and*

$$\begin{cases} 0 \leq \tilde{r}^* \leq d+1 & if \ \ \mathsf{v}_{\mathsf{inn}} \\ d^* \leq \tilde{r}^* \leq \ell^* + 1 & otherwise \end{cases}.$$

We denote the vector of the parameters used in the subinstances as $\mathsf{par}_{\mathsf{inn}} = (w, \ell^*+1, q, d, w_1^*, w_2^*, d^*, \tilde{r}^*)$*. Then the average running time of Lines 6-13 of Algorithm 7, that is the average running time of the modified* filtering step*, is*

given by

$$T_{\mathcal{K}}^{(\mathsf{v}_{\mathsf{inn}})} = \begin{cases} \left(\binom{n}{w} - 1\right) T_{\mathsf{SBC}^*_{\mathsf{dual}}}(\mathsf{par}_{\mathsf{inn}}) + T^{(1)}_{\mathsf{SBC}^*_{\mathsf{dual}}}(\mathsf{par}_{\mathsf{inn}}), & \textit{if } \mathsf{v}_{\mathsf{inn}}, \\ \left(\binom{n}{w} - 1\right) T_{\mathsf{SBC}^*}(\mathsf{par}_{\mathsf{inn}}) + T^{(1)}_{\mathsf{SBC}^*}(\mathsf{par}_{\mathsf{inn}}), & \textit{otherwise.} \end{cases}$$

Proof. $\underline{\mathsf{v}_{\mathsf{inn}} = 1}$: Since $\mathsf{v}_{\mathsf{inn}} = 1$, the cost of executing Lines 6-13, which is $T_{\mathcal{K}}^{(\mathsf{v}_{\mathsf{inn}})}$, is given by $|\bar{\mathcal{S}}_w(\boldsymbol{V})| = \binom{n}{w}$ multiplied by the running time of Algorithm 6 applied to the subinstances. However, as anticipated before, one of the subinstance is required to have one solution, therefore we estimate the running time of Algorithm 6 applied to this subinstance with $T^{(1)}_{\mathsf{SBC}^*_{\mathsf{dual}}}(\mathsf{par}_{\mathsf{inn}})$, while for all the other subinstances we use $T_{\mathsf{SBC}^*_{\mathsf{dual}}}(\mathsf{par}_{\mathsf{inn}})$.
$\underline{\mathsf{v}_{\mathsf{inn}} = 0}$: The proof is exactly the same as in the previous case with Algorithm 6 replaced by Algorithm 5. □

Lemma 2. *Under the same hypothesis and notation as in Lemma 1, then the cardinality of the list $\mathcal{K}$ obtained after Line 13 of Algorithm 7 is given on average by*

$$|\mathcal{K}^{(\mathsf{v}_{\mathsf{inn}})}| = \begin{cases} \left(\binom{n}{w} - 1\right) N_{\mathsf{SBC}^*_{\mathsf{dual}}}(\mathsf{par}_{\mathsf{inn}}) + \max\left\{N^{(1)}_{\mathsf{SBC}^*_{\mathsf{dual}}}(\mathsf{par}_{\mathsf{inn}}), 1\right\}, & \textit{if } \mathsf{v}_{\mathsf{inn}}, \\ \left(\left(\binom{n}{w} - 1\right) N_{\mathsf{SBC}^*}(\mathsf{par}_{\mathsf{inn}}) + \max\left\{N^{(1)}_{\mathsf{SBC}^*}(\mathsf{par}_{\mathsf{inn}}), 1\right\}\right) \cdot N^{(u)}_{\mathsf{swaps}}(w, d, q), & \textit{otherwise.} \end{cases}$$

Proof. $\underline{\mathsf{v}_{\mathsf{inn}} = 0}$: $|\mathcal{K}^{(\mathsf{v}_{\mathsf{inn}})}|$ is the cardinality of the list $\mathcal{K}$ containing all the candidates for $\sigma(\tilde{\boldsymbol{V}})_J$. This is given by the cardinality of the list $\mathcal{K}^*$ defined at Line 11 containing all the permuted $\bar{\boldsymbol{H}}$ that solve the subinstance multiplied by $N^{(u)}_{\mathsf{swaps}}(w, d, q)$, since, for every element $\boldsymbol{X} \in \mathcal{K}^*$, $\mathsf{InvCand}^*(\boldsymbol{X})$ returns $N^{(u)}_{\mathsf{swaps}}(w, d, q)$ candidates for $\sigma(\tilde{\boldsymbol{V}})_J$. Because of Proposition 7, we estimate $|\mathcal{K}^*|$ with $N_{\mathsf{SBC}^*}(\mathsf{par}_{\mathsf{inn}})$ in $\binom{n}{w} - 1$ subinstances, while we use $\max\{N^{(1)}_{\mathsf{SBC}^*}(\mathsf{par}_{\mathsf{inn}}), 1\}$ for one subinstance.
$\underline{\mathsf{v}_{\mathsf{inn}} = 1}$: The proof is obtained in the same way as in the previous case, but we don't have the factor $N^{(u)}_{\mathsf{swaps}}(w, d, q)$ because $\mathsf{SBC}^*_{\mathsf{dual}}$ already returns the candidates for $\sigma(\tilde{\boldsymbol{V}})_J$ (the swaps are handled internally). □

Lemma 3. *Under the same hypothesis and notation as in Lemma 1 and Lemma 2. Moreover, let $\mathsf{v}_{\mathsf{out}} \in \{0, 1\}$ be a boolean parameter that specifies whether to use Algorithm 4 or the original* **STEP 2-STEP 3** *in the outer instance and let $\tilde{r} \in \mathbb{N}$ be a valid parameter for the outer instance, that is*

$$\begin{cases} 0 \leq \tilde{r} \leq \ell + 1 & \textit{if } \mathsf{v}_{\mathsf{out}} \\ d \leq \tilde{r} \leq r & \textit{otherwise} \end{cases}.$$

Then the average running time of Lines 14-20 is given by

$$T_{\mathcal{L}}^{(\mathsf{v}_{\mathsf{inn}}, \mathsf{v}_{\mathsf{out}})} = \begin{cases} \bar{T}_{\mathcal{L}}(n, \ell, w, \tilde{r}, |\mathcal{K}^{(\mathsf{v}_{\mathsf{inn}})}|), & \textit{if } \mathsf{v}_{\mathsf{out}} \\ \frac{n!}{(r+w-\tilde{r})!} + |\mathcal{K}^{(\mathsf{v}_{\mathsf{inn}})}| + \frac{n!|\mathcal{K}^{(\mathsf{v}_{\mathsf{inn}})}|}{(r+w-\tilde{r})! q^{\ell(\tilde{r}-d)}} + \frac{(n-w)!|\mathcal{K}^{(\mathsf{v}_{\mathsf{inn}})}|}{(r-\tilde{r})! q^{\ell(\tilde{r}-d)}}, & \textit{otherwise.} \end{cases}$$

Proof. $\underline{\mathsf{v}_{\mathsf{out}} = 0}$: In these cases the proof follows as in Proposition 2 by using the estimates for $|\mathcal{K}|$ given by Lemma 2: in both cases, the first three terms of the sum are obtained as in Equation (12) by replacing $|\mathcal{K}|$ with $|\mathcal{K}^{(\mathsf{v}_{\mathsf{inn}})}|$. The last term of both sums is obtained in the same way from Equation (13).
$\underline{\mathsf{v}_{\mathsf{out}} = 1}$: The proof follows from Proposition 3 and Lemma 2. □

Theorem 1. *Under the same hypothesis and notation as in Lemma 1, Lemma 2 and Lemma 3, then the average running time of Algorithm 7 is given by*

$$T^{(4)}(n, r, q, \ell, w, d, \tilde{r}, w_1^*, w_2^*, d^*, \tilde{r}^*, \mathsf{v}_{\mathsf{inn}}, \mathsf{v}_{\mathsf{out}}) = T_{\mathsf{ISD}}^{(d)}(n, r, w) + T_{\mathcal{K}}^{(\mathsf{v}_{\mathsf{inn}})} + T_{\mathcal{L}}^{(\mathsf{v}_{\mathsf{inn}}, \mathsf{v}_{\mathsf{out}})}.$$

Proof. The proof follows by combining Equation (3), Lemma 1 and Lemma 3, since the cost of Lines 2-5 is negligible. □

Remark 16. Actually, in Theorem 1 we consider n and r to be valid values for respectively w and d, since the code generated by $\boldsymbol{H}$ is a subcode of itself. We make this explicit since n and r might not satisfy $N_{n,r}(n, r) > 1$. In this case Algorithm 7 becomes, if $\mathsf{v}_{\mathsf{out}} = 0$, the same as Algorithm 3, as we are applying SBC^* to a single instance (since $\binom{n}{n} = 1$) with parameters (n, r, q, ℓ), where $\boldsymbol{V}$ has no repeated columns. Consequently, in this setting, the running time of SBC^* coincides with that of SBC.

5 Discussion and Comparisons

In this section, we assess the impact of the algorithmic improvements introduced in Section 4 by comparing the complexity of Algorithm 7 with that of KMP and SBC.

Complexity for adaptive field size. Figure 3 reports the complexities of the three algorithms for $n \in \{64, 92, 118\}$, rates $R = \frac{r}{n} \in \{0.3, 0.5, 0.7\}$, and varying values of ℓ. For each parameter set (n, R, ℓ), the field size q is chosen as the smallest power of a prime satisfying conditions i)–iv) in Section 2.1, following the same approach adopted by Santini *et al.* [24]. Notice that, for given n, R and ℓ, a smaller field size q generally leads to a higher estimated time complexity of the problem instance when solving it with KMP or SBC. Hence, this choice of considering the smallest q admissible is motivated by the fact that it allows the construction of instances with smaller public key sizes. Moreover, notice that these conditions i)–iv) in Section 2.1 have to be met by the parameters of the homogeneous instance, since by adding a row of ones to the parity check matrix and extending the zero matrix accordingly, we introduce equations that are verified by every permutation. For every n and R, we consider all values of ℓ such that the corresponding q is at least 3. In addition, for each parameter set analyzed, we consider the minimum complexity between the problem and its dual version, as defined in Equation (6). As explained in Remark 16, Algorithm 7 automatically falls back to the SBC setting whenever this is the most convenient

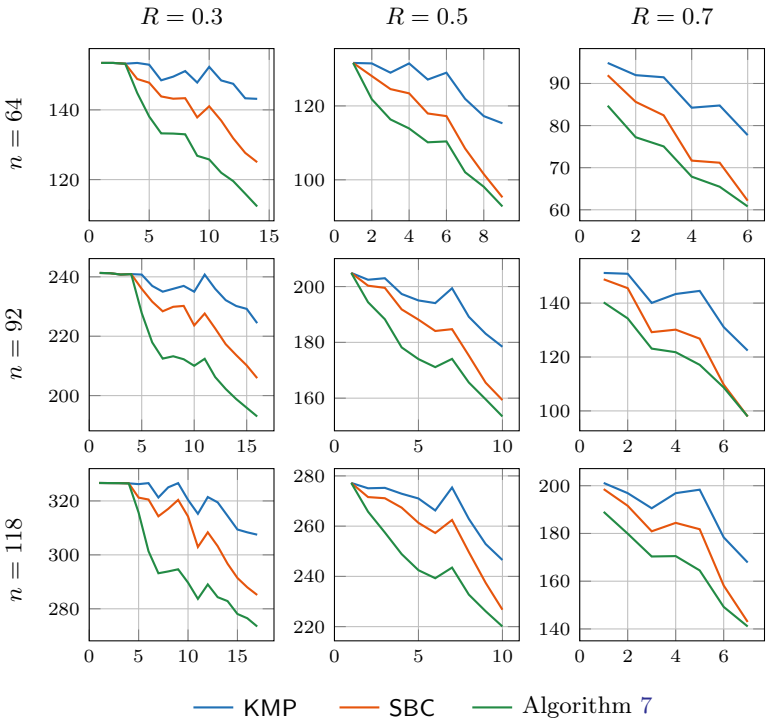

Fig. 3: Time complexity, expressed in $\log_2$ of the number of matrix and list operations, of KMP, SBC and Algorithm 7 for $n \in \{64,\ 92,\ 118\}$, $R \in \{0.3,\ 0.5,\ 0.7\}$ and different values of ℓ along the x-axis.

choice. Hence, as one can observe in the plots, its complexity is always smaller than or equal to that of SBC. In particular, for $R = 0.7$, we observe gains already from $\ell = 1$, whereas for $R = 0.5$ and $R = 0.3$, the improvement becomes evident only for $\ell = 2$ and $\ell = 4$, respectively. On the other hand, KMP is generally worse or equal than SBC and Algorithm 7.

Complexity for large fields and implications for PERK 1.0. When q is larger than the smallest suitable power of a prime, we observe that Algorithm 7 quickly converges to SBC. This behavior is expected, since larger field sizes q prevent the existence of subcodes with small support size that are useful when solving the IPKP subinstances, as explained in Section 4.2. Consequently, our variant of **STEP 1** in SBC becomes ineffective in this regime. This trend is clearly visible when comparing the complexities of Algorithm 7 and SBC for the PERK 1.0 [1] parameter sets, as reported in Table 2. We observe that Algorithm 7 achieves improvements for minimal field sizes q, while it coincides with SBC for the PERK field size $q = 1021$. This choice of q, despite leading to larger

public keys, was motivated by other constraints imposed by the MPC-in-the-Head framework used in PERK.

Discussion on PERK 2.1 parameter sets. We found that SBC remains optimal also for the parameter sets of the more recent PERK 2.1 [2]. Unlike in the previous version of PERK, the field size $q = 2048$ is close to the minimum admissible value according to conditions i)–iv) in Section 2.1. Specifically, the minimal field sizes satisfying these conditions are 1997, 2017, and 1993 for security levels I, III, and V, respectively. However, in this case, there is another reason supporting the optimality of SBC: across all parameter sets, we have $\ell = 1$ and $0.42 \leq R \leq 0.5$. From Figure 3, which reports the same values of n as in the PERK 2.1 parameter sets, we indeed observe that for this range of code rates and $\ell = 1$, Algorithm 7 matches SBC in complexity.

	(n, r, ℓ)	q	KMP	SBC	Algorithm 7
perk-i-3	$(79, 35, 3)$	1021	146.7	145.6	145.6
minimum q		13	180.1	175.4	166.0
perk-i-5	$(83, 36, 5)$	1021	153.8	147.7	147.7
minimum q		5	191.3	185.2	171.2
perk-iii-3	$(112, 54, 3)$	1021	212.7	210.8	210.8
minimum q		16	257.6	254.7	242.6
perk-iii-5	$(116, 55, 5)$	1021	219.0	212.6	212.6
minimum q		5	276.8	270.1	248.6
perk-v-3	$(146, 75, 3)$	1021	280.0	275.2	275.2
minimum q		16	342.5	339.2	325.5
perk-v-5	$(150, 76, 5)$	1021	280.9	274.1	274.1
minimum q		5	363.6	358.0	329.0

Table 2: Comparison of complexities for different PERK 1.0 parameter set and equivalent parameters but with minimal field size q.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and suggestions that helped improving the manuscript. The third author acknowledges support from Ripple's University Blockchain Research Initiative and from the INDAM group GNSAGA.

References

1. Aaraj, N., Bettaieb, S., Bidoux, L., Budroni, A., Dyseryn, V., Esser, A., Gaborit, P., Kulkarni, M., Mateu, V., Palumbi, M., Perin, L., Tillich, J.: PERK. Tech. rep., National Institute of Standards and Technology (2023), available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures
2. Aaraj, N., Bettaieb, S., Bidoux, L., Budroni, A., Dyseryn, V., Esser, A., Feneuil, T., Gaborit, P., Kulkarni, M., Mateu, V., Palumbi, M., Perin, L., Rivain, M., Tillich, J.P., Xagawa, K.: PERK: A post-quantum signature scheme (version 2.1.0) (Sep 2025), specification document; available at https://pqc-perk.org/assets/downloads/perk-v2.1.0.pdf
3. Alagic, G., Bros, M., Ciadoux, P., Cooper, D., Dang, Q., Dang, T., Kelsey, J.M., Lichtinger, J., Miller, C.A., Moody, D., Peralta, R., Perlner, R., Robinson, A., Silberg, H., Smith-Tone, D., Waller, N., Liu, Y.K.: Status report on the first round of the additional digital signature schemes for the nist post-quantum cryptography standardization process. NIST Interagency/Internal Report (NISTIR) 8528, National Institute of Standards and Technology (NIST), Gaithersburg, MD (Oct 2024). https://doi.org/10.6028/NIST.IR.8528, https://doi.org/10.6028/NIST.IR.8528
4. Baldi, M., Battagliola, M., Mechri, R.E., Santini, P., Schiavoni, R., Zuane, D.D.: SPECK: Signatures from permutation equivalence of codes and kernels. Cryptology ePrint Archive, Paper 2025/923 (2025), https://eprint.iacr.org/2025/923
5. Baritaud, T., Campana, M., Chauvaud, P., Gilbert, H.: On the security of the permuted kernel identification scheme. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 305–311. Springer, Berlin, Heidelberg (Aug 1993). https://doi.org/10.1007/3-540-48071-4_21
6. Berger, T.P., Gueye, C.T., Klamti, J.B.: A np-complete problem in coding theory with application to code based cryptography. In: El Hajji, S., Nitaj, A., Souidi, E.M. (eds.) Codes, Cryptology and Information Security. pp. 230–237. Springer International Publishing, Cham (2017)
7. Beullens, W.: Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 183–211. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45727-3_7
8. Bidoux, L., Gaborit, P.: Compact post-quantum signatures from proofs of knowledge leveraging structure for the pkp, sd and rsd problems. In: El Hajji, S., Mesnager, S., Souidi, E.M. (eds.) Codes, Cryptology and Information Security. pp. 10–42. Springer Nature Switzerland, Cham (2023)
9. Budroni, A., Defranceschi, M., Pintore, F.: Accompanying repository. https://github.com/Crypto-TII/pkp-complexity (2025), accessed: 2025-10-30
10. Faugère, J.C., Koussa, E., Macario-Rat, G., Patarin, J., Perret, L.: PKP-based signature scheme. Cryptology ePrint Archive, Report 2018/714 (2018), https://eprint.iacr.org/2018/714
11. Feneuil, T.: Building MPCitH-based signatures from MQ, MinRank, rank SD and PKP. Cryptology ePrint Archive, Report 2022/1512 (2022), https://eprint.iacr.org/2022/1512
12. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA (1990)
13. Georgiades, J.: Some remarks on the security of the identification scheme based on permuted kernels. Journal of Cryptology **5**(2), 133–137 (Jan 1992). https://doi.org/10.1007/BF00193565

14. Jaulmes, É., Joux, A.: Cryptanalysis of PKP: A new approach. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 165–172. Springer, Berlin, Heidelberg (Feb 2001). https://doi.org/10.1007/3-540-44586-2_12
15. Koussa, E., Macario-Rat, G., Patarin, J.: On the complexity of the permuted kernel problem. Cryptology ePrint Archive, Report 2019/412 (2019), https://eprint.iacr.org/2019/412
16. LAMPE, R., PATARIN, J.: Analysis of some natural variants of the PKP algorithm. Cryptology ePrint Archive, Report 2011/686 (2011), https://eprint.iacr.org/2011/686
17. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S.: CRYSTALS-DILITHIUM. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
18. National Institute of Standards and Technology: Post-Quantum Cryptography Standardization. https://csrc.nist.gov/projects/post-quantum-cryptography (2017)
19. National Institute of Standards and Technology: Post-quantum cryptography: Digital signature schemes. Round 2 Additional Signatures (2023), https://csrc.nist.gov/projects/pqc-dig-sig/round-2-additional-signatures
20. Patarin, J., Chauvaud, P.: Improved algorithms for the permuted kernel problem. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 391–402. Springer, Berlin, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_33
21. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
22. Sanna, C.: On the number of solutions to a random instance of the permuted kernel problem. Journal of Complexity **86**, 101898 (2025). https://doi.org/https://doi.org/10.1016/j.jco.2024.101898, https://www.sciencedirect.com/science/article/pii/S0885064X2400075X
23. Santini, P., Baldi, M., Chiaraluce, F.: A novel attack to the permuted kernel problem. In: 2022 IEEE International Symposium on Information Theory (ISIT). p. 1441–1446. IEEE Press (2022). https://doi.org/10.1109/ISIT50566.2022.9834867, https://doi.org/10.1109/ISIT50566.2022.9834867
24. Santini, P., Baldi, M., Chiaraluce, F.: Computational hardness of the permuted kernel and subcode equivalence problems. IEEE Trans. Inf. Theor. **70**(3), 2254–2270 (Mar 2024). https://doi.org/10.1109/TIT.2023.3323068, https://doi.org/10.1109/TIT.2023.3323068
25. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D., Ding, J.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
26. Shamir, A.: An efficient identification scheme based on permuted kernels (extended abstract) (rump session). In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 606–609. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34805-0_54
27. Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (1994). https://doi.org/10.1109/SFCS.1994.365700

A On the rank of the puncturings of V

We explain here why we can assume that $\ell^* = \mathsf{rank}(\boldsymbol{V}_W) = \min\{\ell, w-d+1\}$. With the same notation as Section 4.2, we have that for W corresponding to the solution, $\mathsf{rank}(\boldsymbol{V}_W) \leq \min\{\ell, w-d+1\}$. Indeed, $\boldsymbol{V}_W$ is $\ell \times w$ and there exists $\tilde{\pi} \in \mathsf{Sym}(w)$ such that

$$\tilde{\pi}(\boldsymbol{V}_W)\bar{\boldsymbol{H}}^\top = \boldsymbol{E}^*.$$

Then, by rank-nullity theorem and by noting that $\mathsf{rank}(\tilde{\pi}(\boldsymbol{V}_W)) = \mathsf{rank}(\boldsymbol{V}_W)$ for any $\tilde{\pi} \in \mathsf{Sym}(w)$, we have that

$$\mathsf{rank}(\boldsymbol{V}_W) = \mathsf{rank}(\tilde{\pi}(\boldsymbol{V}_W)) \leq w - \mathsf{rank}(\bar{\boldsymbol{H}}) + \mathsf{rank}(\boldsymbol{E}^*) = w - d + 1. \tag{22}$$

We can safely assume the equality in Equation (22), that is $\mathsf{rank}(\boldsymbol{V}_W) = \min\{\ell, w-d+1\}$. Indeed, for ℓ small enough (which corresponds to the cases of cryptographic interest), we have $\ell < w-d+1 \leq w$. Then, Equation (22) is satisfied, and $\boldsymbol{V}_W$ is a non-square submatrix of the full-rank matrix $\boldsymbol{V}$, and it is full rank with high probability. If, on the other hand, $\ell > w-d+1$, we can obtain an equivalent instance after removing the redundant rows of $\boldsymbol{V}_W$, which are $\ell - (w-d+1)$, and the resulting matrix is full-rank with high probability. It remains to consider the case $\ell = w-d+1$. If $d > 1$, $\boldsymbol{V}_W$ is again non-square, so the previous considerations apply. Otherwise, if $d = 1$ and $\ell = w$, $\boldsymbol{V}_W$ is square, and the probability that it is not full rank is approximately $1/q$. In our estimates, however, we can safely assume that this never occurs. Indeed, when a square $\boldsymbol{V}_W$ is not full rank, solving such a subinstance requires an overhead of guessing $\tilde{\pi}$ on $w - \mathsf{rank}(\boldsymbol{V}_W)$ columns. With high probability, this overhead appears in few cases and, when it appears, it is absorbed by the overall complexity, so we assume it to be negligible.