Cryptanalysis of the Binary Permuted Kernel Problem[⋆]

Thales Bandiera Paiva and Routo Terada

Universidade de São Paulo
{tpaiva, rt}@ime.usp.br

Abstract. In 1989, Shamir presented an efficient identification scheme (IDS) based on the permuted kernel problem (PKP). After 21 years, PKP was generalized by Lampe and Patarin, who were able to build an IDS similar to Shamir's one, but using the binary field. This binary variant presented some interesting advantages over Shamir's original IDS, such as reduced number of operations and inherently resistance against side-channel attacks. In the security analysis, considering the best attacks against the original PKP, the authors concluded that none of these existing attacks appeared to have a significant advantage when attacking the binary variant. In this paper, we propose the first attack that targets the binary PKP. The attack is analyzed in detail, and its practical performance is compared with our theoretical models. For the proposed parameters originally targeting 79 and 98 bits of security, our attack can recover about 100% of all keys using less than 2^{63} and 2^{77} operations, respectively.

Keywords: permuted kernel problem, cryptanalysis, post-quantum cryptography

1 Introduction

With the engineering progress on building larger quantum computers, the main cryptographic schemes used today become more and more vulnerable. Since 2016, the National Institute of Standards and Technology (NIST), is running a standardization process for post-quantum cryptography [4]. A similar initiative is conducted by the Chinese Association for Cryptographic Research (CACR).

One of the candidate for CACR's competition is PKP-DSS [3], a digital signature scheme based on the hardness of the permuted kernel problem (PKP). This signature scheme is obtained by applying the Fiat-Shamir [6] transform on Shamir's PKP-based identification scheme [20], which dates back from 1989. Given a matrix **A** and a vector **v** with elements in a finite field, PKP asks to find a permutation of the entries of **v** that is in the kernel of **A**. PKP is NP-hard

⋆ T. B. Paiva is supported by CAPES. R. Terada is supported by CNPq grant number 442014/2014-7.

and there is no known quantum algorithm which have a significant advantage over classical algorithms when solving the problem.

In 2010, Lampe and Patarin proposed a generalized version of PKP, in which vector $\mathbf{v}$ is substituted by a matrix $\mathbf{V}$. This enabled them to instantiate PKP in the binary field, without an apparent security loss. At the time, this binary variant presented some interesting advantages such as a reduction in the number of operations and an inherently resistance to side-channel attacks. To estimate the security of binary PKP, the authors considered the best attacks against the original PKP, with minor adjustments to make they work against the binary variant. They noted that none of the available attacks was significantly faster against binary PKP.

However, the use of binary coefficients for matrix $\mathbf{A}$ comes with a security risk. We observed that that low weight binary words occur with non-negligible probability in two public spaces: one is generated by the matrix $\mathbf{A}$ while the other is generated by the kernel of $\mathbf{V}$. It is then possible to devise an attack against binary PKP by matching these low weight codewords using subgraph isomorphism algorithms, and recovering the secret permutation from these matchings.

Contribution. In this paper, we present the first attack that specifically targets the binary PKP. Unlike previous attacks, which need a very large amount of memory to run efficiently, our attack uses only a negligible amount of memory. This allows us to provide a concrete implementation of the attack. We provide a detailed analysis of the attack, and then compare these results with the attack's performance in practice. As an example of the power of the attack: for binary PKP parameters originally targeting 80 bits of security, it uses about 2^{63} CPU cycles to fully recover the key, while the best previously known attack [12] needs about 2^{76} matrix-vector multiplications and 2^{50} bytes of memory.

Paper organization. In Section 2, we introduce our notation and review basic concepts of Coding Theory. Then, PKP and its binary variant are presented in Section 3, where we also review previous attacks against PKP. The attack is described in Section 4 and its performance is analyzed in Section 5. The asymptotic analysis of the attack is given in Section 6. In Section 7 we briefly describe how to choose secure parameters for binary PKP. In Section 8 we conclude and provide directions for future work.

2 Background

This section introduces the notation and reviews important concepts in Coding Theory.

Notation. Vectors and matrices are denoted by lower and upper case bold letters, respectively. In general, vectors are rows, except when explicitly mentioning specific columns of matrices. If ϕ is a permutation of n elements and $\mathbf{M}$ is an $n \times n$ matrix, then $\mathbf{M}_\phi$ and $\phi(\mathbf{M})$ correspond to the action of permutation ϕ

over rows and columns of $\mathbf{M}$, respectively. For any matrix $\mathbf{X}$, we denote its i–th column as $(\mathbf{X})^i$. We denote the finite field of q elements as $\mathbb{F}_q$.

We abuse the factorial notation to avoid overloading expressions in the analysis of the attack. For any $x \in [0, +\infty)$, we let

$$x! = \begin{cases} \Gamma(x+1), & \text{if } x \geq 1, \text{ and} \\ 1, & \text{otherwise.} \end{cases}$$

Clearly it does not affect the definition of factorials of integers. Furthermore, it allows us to evaluate upper bounds of products of factorials of real numbers without having to worry about the interval $x \in (0, 1)$, where $\Gamma(x+1) < 1$, which could make the product vanish rapidly. Using this notation, we can then write $\binom{x}{y} = x!/((x-y)!y!)$, for $x, y \in [0, +\infty)$ with $x > y$. These will make for a more clear description of our approximations in Section 5.

Coding Theory. A binary $[n, k]$-linear code is a k-dimensional linear subspace of $\mathbb{F}_2^n$, where $\mathbb{F}_2$ denotes the binary field. Let $\mathcal{C}$ be a binary $[n, k]$-linear code. If $\mathcal{C}$ is the linear subspace spanned by the rows of a matrix $\mathbf{G}$ in $\mathbb{F}_2^{k \times n}$, we say that $\mathbf{G}$ is a generator matrix of $\mathcal{C}$. The Hamming weight of a vector $\mathbf{v}$, denoted by $\mathrm{w}(\mathbf{v})$, is the number of its non-null entries. The support of a vector $\mathbf{v}$, denoted by $\mathrm{supp}(\mathbf{v})$, is the set of indexes of its non-null entries.

3 The Permuted Kernel Problem

Let us begin by formally defining the permuted kernel problem. Let $\mathbf{A}$ be an $m \times n$ matrix and $\mathbf{v}$ be a vector of n entries whose coordinates are taken from a finite field $\mathbb{F}_p$. Then, the permuted kernel problem asks to find some permutation π of the coordinates of $\mathbf{v}$ such that $\mathbf{A}\mathbf{v}_\pi^\top = \mathbf{0}$.

PKP is well-known to be NP-hard [8], and it is conjectured to be hard on the average case. The naive approach to solve this problem would be to test all permutations of the entries of $\mathbf{v}$. Intuitively, there are two components which make the problem hard. The first is the large number of possible permutations, which is close to $n!$, when $\mathbf{v}$ does not have a large number of equal entries. The second is the small number of permutations of $\mathbf{v}$ which are in the kernel of $\mathbf{A}$.

In 2011, Lampe and Patarin [13] considered a PKP variant with $p = 2$. The authors pointed a few problems when transitioning to the binary setting that need to be taken into account. One is that the number of different permutations is significantly reduced, since every two binary vectors of the same weight are equal, up to some permutation. Furthermore, for a fixed matrix $\mathbf{A}$, there are effectively only n possibilities for $\mathbf{v}$, corresponding to one for each possible value of $\mathrm{w}(\mathbf{v})$. To avoid these problems, they proposed the use of an $n \times \ell$ matrix $\mathbf{V}$ instead of the vector $\mathbf{v}$, obtaining the following PKP variant.

Definition 1 (Binary PKP [13]). *Let $\mathbf{A}$ be an $m \times n$ binary matrix and $\mathbf{V}$ be an $n \times \ell$ binary matrix. Then, the permuted kernel problem asks to find some permutation π of the rows of $\mathbf{V}$ such that $\mathbf{A}\mathbf{V}_\pi = \mathbf{0}$.*

Notice that the original PKP can be seen as an instance of this generalized variant, by taking p instead of 2, and $\ell = 1$.

Even though the main interest on PKP is for the construction of signature schemes, we will not review details of Shamir's protocol [20] or PKP-DSS [3] this construction because they are not relevant for our attack.

3.1 Previous Attacks on PKP

After Shamir introduced the PKP-based IDS [20], there has been some effort to find efficient algorithms to solve the problem. In 1990, Georgiades [9] discussed how one can use symmetric equations, such as the sum of the entries of $\mathbf{v}$ or the sum of their squares, can help in lowering the number of permutations one needs to test. This, combined with the linear relations among the coordinates of kernel elements, can reduce the number of permutations to test in a brute force attack from $n!$ to $n!/(m+2)!$ permutations.

Soon after, in 1992, Baritaud et al. [1] proposed a time-memory tradeoff, where one first precompute a large table of partial solutions, which is then used to speed up a bruteforce search. In particular, for attack parameters (k, k'), their algorithm searches for solutions of a set of $k \leq m$ equations, after precomputing partial values of the equations when some set of k' variables are fixed by some arrangement of the entries in $\mathbf{v}$.

In 1993, Patarin and Chauvaud [17] showed a significant improvement on the cryptanalysis of PKP, which is also based on a time-memory tradeoff. Their idea was to partition the variables of the linear equation $\mathbf{A}\mathbf{v}_\pi^\top = \mathbf{0}$ into two sets. For one set, all the possible values for their linear combination is computed and stored in a file. Then, a brute-force search, which is sped-up by the precomputed values, is used to find the values of the other set of variables. Furthermore, in 1997, Poupard [18] provided a careful and realistic extension on the analysis of Patarin and Chauvaud's algorithm by considering the impact of reasonable memory limitations on the time-memory trade-off.

In 2001, Jaulmes and Joux [10] proposed a new attack against PKP, which is also based on a time-memory tradeoff, but used a very different strategy. Their attack consists in adapting an algorithm for counting points in an elliptic curve [11] to solve a new problem, called 4SET, to which PKP can be reduced. Interestingly, this approach resulted in an algorithm somewhat similar to the one by Patarin and Chavaud [17], but, for years after the attack was published, it appeared to be more efficient.

More recently, in 2019, Koussa, Macario-Rat and Patarin [12] presented two important contributions on the hardness of PKP. Their first contribution is to provide a detailed analysis of the attack proposed by Jaulmes and Joux [10], which was considered to be the most efficient attack against PKP. They concluded that Jaulmes and Joux's attack may not be as efficient as previously thought for the current PKP security parameters. Koussa, Macario-Rat and Patarin's second contribution is a combination of the ideas of Patarin and Chauvaud [17] with the ones by Poupard [18] to obtain a new algorithm to solve PKP, together with a detailed analysis on their time and space complexity.

The main drawback of Koussa's et al. attack is that they use a significant amount of memory, and their implementation may not be efficient in practice. Moreover, all of the published attacks against PKP target the original version of the problem. And even though they all can be adapted to attack the binary PKP, as done by Lampe and Patarin [13] for their analysis, it appears that none of the attacks are significantly more efficient in the binary case.

3.2 Instantiation

We now present the parameter sets for PKP, in which the security level is estimated based on the best attacks available by Koussa et al. [12]. Table 1 shows these parameter sets for different security levels. The focus of this work is in the parameter sets given in the first two rows, corresponding to the binary PKP. It is important to notice that what we now consider to be the parameter sets BPKP–76 and BPKP–89, originally targeted security levels 79 and 98, respectively. However, these had to be revised after Koussa et al's [12] attack.

Table 1: Parameter sets for different security levels. The security level is estimated based on the attack by Koussa et al. [12].

Parameter set	Security level	Targeted security level when proposed	p	n	m	ℓ
BPKP–76 [13]	76	79	2	38	15	10
BPKP–89 [13]	89	98	2	42	15	11
PKP–128 [3]	128	128	251	69	41	1
PKP–192 [3]	192	192	509	94	54	1
PKP-256 [3]	256	256	4093	106	47	1

4 A Novel Attack Against Binary PKP

We are given the public matrices $\mathbf{A}$ and $\mathbf{V}$ and we want to find the secret permutation π such that $\mathbf{AV}_\pi = \mathbf{0}$. Let $\mathcal{C}_\mathbf{A}$ and $\mathcal{C}_\mathbf{K}$ be the binary codes generated by $\mathbf{A}$ and $\mathbf{K}$, respectively, where $\mathbf{K}$ is the left kernel matrix of $\mathbf{V}$. Fix an integer w small enough so that we can build the sets $\mathcal{L}^w_\mathbf{A}$ and $\mathcal{L}^w_\mathbf{K}$ consisting of all the codewords of weight w in $\mathcal{C}_\mathbf{A}$ and $\mathcal{C}_\mathbf{K}$, correspondingly. Notice that, since $\mathbf{AV}_\pi = \mathbf{0}$, then $\mathcal{L}^w_\mathbf{A} \subset \mathcal{L}^w_{\pi(\mathbf{K})} = \{\mathbf{u}_\pi : \mathbf{u} \in \mathcal{L}^w_\mathbf{K}\}$.

This idea gives the following simple algorithm to find the secret permutation π. First find a subset S of $\mathcal{L}^w_\mathbf{K}$, such that, for some permutation τ, $\mathcal{L}^w_\mathbf{A} = \{\mathbf{u}_\tau : \mathbf{u} \in S\}$. Then, test if the corresponding column permutation τ is valid, that is, if $\mathbf{AV}_\tau = \mathbf{0}$. If τ is valid, return it as π. Otherwise, restart the search. Figure 1 can be useful for visualizing the relationship between the two sets of codewords, which is the core of the attack.

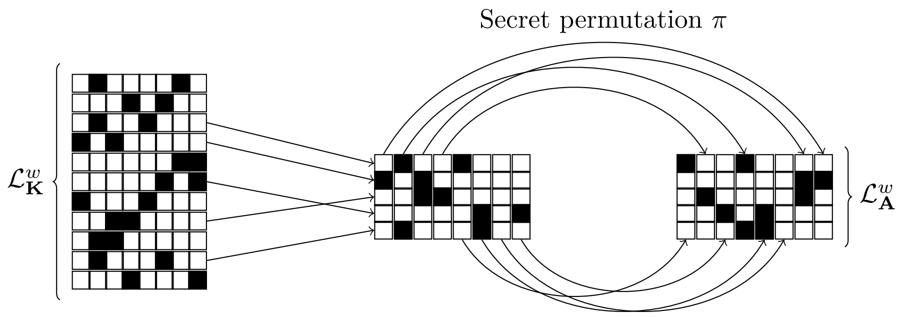

Fig. 1: Illustration of the relationship between $\mathcal{L}^w_{\mathbf{A}}$ and $\mathcal{L}^w_{\mathbf{K}}$ with respect to the secret column permutation π for codewords of weight $w = 2$. White and black squares represent null and non-null entries, respectively.

Even though it has a rather simple description, we need to carefully deal with the following two problems. The first one is that matching vectors in $\mathcal{L}^w_{\mathbf{A}}$ and a subset of $\mathcal{L}^w_{\mathbf{K}}$ is closely related to the subgraph isomorphism problem, which is NP-hard [8]. The second problem is that, since we are dealing with sparse codewords, there may be a large number of repeated columns in $\mathcal{L}^w_{\mathbf{A}}$. This could potentially make it infeasible to find the secret permutation π because of the combinatorial explosion on the number of possible permutations between columns.

In the following sections, we formally describe the algorithms for the attack against the binary PKP. Then, after this initial exposition, each component of the algorithm is analyzed in Section 5.

4.1 Searching for codewords of small weight

The problem of finding codewords of small weight is hard in general, with the security of some well known cryptographic schemes, such as McEliece's one [15], depend on this problem's hardness. However, in the binary PKP setting, the length n of the codes in question, namely $\mathcal{C}_{\mathbf{A}}$ and $\mathcal{C}_{\mathbf{K}}$, is typically very small, which makes it even possible to use brute force. Using brute force, one has to test exactly if $\binom{n}{w}$ words are elements of each of the codes.

A better approach would be to use specialized algorithms from Coding Theory such as Stern's algorithm [22], which we used in our attack implementation, or its improved variants [2, 7]. All of these are are well-known probabilistic algorithms that can be used to find low weight codewords in binary codes.

4.2 Searching for matchings

Aiming to simplify the description of the attack, we identify sets $\mathcal{L}^w_{\mathbf{A}}$ and $\mathcal{L}^w_{\mathbf{K}}$ as matrices where each row is one vector in the corresponding set. This is arguably a natural identification when we consider a real implementation of the algorithm in a programming language such as C.

We now focus on the problem of finding a submatrix of $\mathcal{L}_{\mathbf{K}}^{w}$ which is equal to matrix $\mathcal{L}_{\mathbf{A}}^{w}$ when its coordinates are permuted by some permutation τ. Notice that, if we let $\mathcal{G}(\mathbf{X})$ be the bipartite graph built using matrix $\mathbf{X}$ as a biadjacency matrix, then this problem is exactly the subgraph isomorphism problem for the bipartite graphs $\mathcal{G}(\mathcal{L}_{\mathbf{A}}^{w})$ and $\mathcal{G}(\mathcal{L}_{\mathbf{K}}^{w})$.

Even though subgraph isomorphism is NP-hard [8], for small enough inputs, the problem has been widely studied because of its importance in Pattern Recognition. It is well-known that, for sufficiently small instances, the problem can be solved efficiently using algorithms such as the one by Ullman [23] or the ones from the VF family [5,19]. The main problem with these widely used algorithms is that they use heuristics that make it hard to perform a sound average case complexity analysis for our case. Since such analysis is critical for estimating the concrete security of the scheme, we propose a different algorithm with two remarkable advantages. The first one is that it runs faster than other generic subgraph isomorphism algorithms for our specific case of bipartite graphs. The second is that it is simpler to analyze and give realistic estimates on its performance.

The algorithm we propose is based on a simple depth-first search strategy. In each level α of the search, a node represents a matrix built using a set of α rows of $\mathcal{L}_{\mathbf{K}}^{w}$ which is equal to the first α rows of $\mathcal{L}_{\mathbf{A}}^{w}$, when its columns are permuted by some permutation. Whenever a matching is found, the searching algorithm calls a procedure that tries to extract the secret permutation from the matching. In the following sections, we describe each component of the algorithm in more detail.

Signature of a matrix. It is crucial for the subgraph isomorphism algorithms to efficiently determine whether a matrix $\mathbf{S}$ is equal to a submatrix of $\mathcal{L}_{\mathbf{A}}^{w}$ up to some column permutation. For this task, we can use a function σ such that, if $\sigma(\mathbf{S}_1) = \sigma(\mathbf{S}_2)$, then with high probability $\mathbf{S}_1 = \tau(\mathbf{S}_2)$ for some permutation τ, for any two matrices $\mathbf{S}_1$ and $\mathbf{S}_2$ with equal dimensions.

One easy way to build such a function is to sort the columns of $\mathbf{S}$ using a lexicographical ordering obtaining $\mathbf{S}^{\text{Sorted}}$. Then, the signature of $\mathbf{S}$ is simply $\sigma(\mathbf{S}) = h\left(\mathbf{S}^{\text{Sorted}}\right)$, for some cryptographic hash function h. It is clear, by this construction, that σ is invariant with respect to column permutations.

The problem with sorting is that, since this function will be executed a very large number of times, it can become expensive. One alternative is to use the following approximation $\sigma(\mathbf{S}) = \sum_{\mathbf{c} \text{ column of } \mathbf{S}} h(\mathbf{c})$, for some hash function h.

Precomputation of signatures. This step consists in building the $|\mathcal{L}_{\mathbf{A}}^{w}| \times |\mathcal{L}_{\mathbf{A}}^{w}|$ matrix $\mathbf{H}$ containing signatures of submatrices of $\mathcal{L}_{\mathbf{A}}^{w}$ that are used for pruning the possible child nodes in each level of the search. Let $\mathbf{a}_1, \ldots, \mathbf{a}_{|\mathcal{L}_{\mathbf{A}}^{w}|}$ be the vectors in $\mathcal{L}^w_{\mathbf{A}}$, and let $\mathbf{L}_j$ denote the matrix formed by the first j rows of $\mathcal{L}^w_{\mathbf{A}}$. Then, we let

$$\mathbf{H}_{i,j} = \begin{cases} \sigma\left(\begin{bmatrix} \mathbf{L}_j \\ \hline \mathbf{a}_i \end{bmatrix}\right) & \text{if } i > j \text{, and} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Key recovery algorithm. In this step, the algorithm effectively tries to build a submatrix $\mathbf{S}$ of $\mathcal{L}^w_{\mathbf{K}}$ such that $\tau(\mathbf{S}) = \mathcal{L}^w_{\mathbf{A}}$, for some column permutation τ. The algorithm is formally described as Algorithm 1 but we give a brief description next.

Algorithm 1: KeySearch: Key search algorithm using depth-first search

```
Data: $\mathbf{A}$ and $\mathbf{V}$: the PKP public parameters
      $\mathcal{L}^w_{\mathbf{A}}$: a set of codewords in $\mathcal{C}_{\mathbf{A}}$ of weight $w$
      $\mathcal{L}^w_{\mathbf{K}}$: the set of all codewords in $\mathcal{C}_{\mathbf{K}}$ of weight $w$
      $\mathbf{H}$: the precomputed matrix of signatures
      $\alpha$: the level in the search tree (initially, $\alpha = 0$)
      $\mathbf{S}$: an $\alpha \times n$ matrix (initially, $\mathbf{S}$ is the empty $0 \times n$ matrix)
      $\mathcal{P} = \left(P_1, \ldots, P_{|\mathcal{L}^w_{\mathbf{A}}|}\right)$: the sets of children (initially, each $P_i = \mathcal{L}^w_{\mathbf{K}}$)
Result: $\pi$: a permutation such that $\mathbf{A}\mathbf{V}_\pi = \mathbf{0}$ or $\perp$ if none exists
1  begin
2      if $\alpha = |\mathcal{L}^w_{\mathbf{A}}|$ then
3          return ExtractPermutationFromMatching($\mathbf{A}, \mathbf{V}, \mathbf{S}$)
       /* Updates the possible children for each level not yet defined: */
4      for $i = \alpha + 1$ to $|\mathcal{L}^w_{\mathbf{A}}|$ do
5          $\hat{P}_i \leftarrow \left\{\mathbf{p} \in P_i : \sigma\left(\begin{bmatrix} \mathbf{S} \\ \hline \mathbf{p} \end{bmatrix}\right) = \mathbf{H}_{i,\alpha}\right\}$
6      $\mathcal{P} \leftarrow \left(P_1, \ldots, P_\alpha, \hat{P}_{\alpha+1}, \ldots, \hat{P}_{|\mathcal{L}^w_{\mathbf{A}}|}\right)$
7      for each $\mathbf{p}$ in $\hat{P}_{\alpha+1}$ do
8          Update $\mathbf{S}$ by inserting $\mathbf{p}$ as its last row
           /* Recursive call: */
9          $\pi \leftarrow$ KeySearch($\mathbf{A}, \mathbf{V}, \mathcal{L}^w_{\mathbf{A}}, \mathcal{L}^w_{\mathbf{K}}, \mathbf{H}, \alpha + 1, \mathbf{S}, \mathcal{P}$)
10         if $\pi \neq \perp$ then
11             return $\pi$
12         Update $\mathbf{S}$ by removing its last row $\mathbf{p}$
13     return $\perp$
```

The search starts at level $\alpha = 0$, with $\mathbf{S}$ being a $0 \times n$ empty matrix. At each level α in the search tree, the algorithm runs a pruning procedure, that updates the lists of possible vectors for each level greater than α using the precomputed matrix of signatures $\mathbf{H}$. This ensures that the main invariant of the recursive algorithm is that, at level α, the algorithm holds an $\alpha \times n$ submatrix $\mathbf{S}$ of $\mathcal{L}^w_{\mathbf{K}}$ which is equal to the matrix formed by the first α rows of $\mathcal{L}^w_{\mathbf{A}}$, up to some column permutation. The search proceeds by selecting a vector from set $P_{\alpha+1} \subset \mathcal{L}^w_{\mathbf{K}}$ of vectors which can be safely added to the next level without breaking the invariant. Each time the algorithm successfully gets to a leaf, that is, it adds a vector to level $\alpha = |\mathcal{L}^w_{\mathbf{A}}|$, then a full matching $\mathbf{S}$ is found and the procedure that tries to extract the permutation π from matching $\mathbf{S}$ is executed. If the permutation π is successfully extracted, then π is returned. Otherwise, the depth-first search proceeds.

The procedure for extracting the secret permutation from the matching, if possible, is described in the next section.

4.3 Extracting permutations from matchings

After each each matching found in the previous step, we are given a matrix $\mathbf{S}$ such that $\tau(\mathbf{S}) = \mathcal{L}^w_{\mathbf{A}}$ for at least one permutation of columns τ. In this section we describe how to efficiently extract the secret permutation π from this matching, if possible.

We first consider the brute force solution. Let T be the set of permutations that match equal columns in $\mathbf{S}$ and $\mathcal{L}^w_{\mathbf{A}}$. Then, we can just test, for each permutation τ in T, if $\mathbf{A}\mathbf{V}_\tau = \mathbf{0}$. If one such τ is found, then the algorithm returns $\pi \leftarrow \tau$. Suppose that there are β unique columns $\mathbf{c}_1, \ldots, \mathbf{c}_\beta$ of matrix $\mathcal{L}^w_{\mathbf{A}}$, and let c_i denote the number of times column $\mathbf{c}_i$ appears in $\mathcal{L}^w_{\mathbf{A}}$. This implies that the number of candidate permutations is given by $|T| = \prod_{i=1}^{\beta} (c_i!)$. The brute force approach may be efficient when $\mathbf{S}$ has a large number of unique columns. However, due to the combinatorial nature of this problem, even a small increase in the number of equal columns can make the algorithm very inefficient.

To reduce the number of permutations to test we can use the fact that $\dim(\ker \mathbf{A}) = n - m$. Therefore, there are $n - m$ rows of $\mathbf{V}_\pi$ which, together with the m equations defined by $\mathbf{A}$, completely determine the other m rows of $\mathbf{V}_\pi$. Intuitively, this means we can focus on partial permutations in T corresponding to these $n - m$ indexes.

More formally, let I_1 and I_2 be a partition of the set of possible n indexes such that $|I_1| = m$ and the $m \times m$ matrix $\mathbf{A}_1$ built using the columns from $\mathbf{A}$ whose indexes are in I_1 is invertible. Similarly, let $\mathbf{A}_2$ be the $m \times (n-m)$ matrix whose columns are taken from $\mathbf{A}$, but with indexes in I_2. Let ϕ be the permutation of n elements such that $\phi(\mathbf{A}) = [\mathbf{A}_1 | \mathbf{A}_2]$, and define as $\mathbf{U}_1$ and $\mathbf{U}_2$ the matrices such that $\phi\left((\mathbf{V}_\pi)^\top\right) = (\mathbf{V}_{\pi\phi})^\top = \left[\, \mathbf{U}_1^\top \,|\, \mathbf{U}_2^\top \,\right]$. Then, we have

$$\mathbf{A}\mathbf{V}_\pi = \phi(\mathbf{A})\mathbf{V}_{\pi\phi} = [\mathbf{A}_1 | \mathbf{A}_2] \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix} = \mathbf{A}_1\mathbf{U}_1 + \mathbf{A}_2\mathbf{U}_2 = \mathbf{0},$$

which implies that $\mathbf{U}_1 = \left(\mathbf{A}_1^{-1}\mathbf{A}_2\right)\mathbf{U}_2$.

Therefore, one can reduce the number of permutations in T to test by using the following procedure. Let I be a sequence of n column indexes sorted, in decreasing order, by the number of times in which the corresponding column of matrix $\mathcal{L}_{\mathbf{A}}^{w}$ occurs in this same matrix.[1] Now let I_1 to be composed by the first m indexes in I whose corresponding columns of $\mathbf{A}$ are linearly independent, and let $I_2 = I - I_1 = \{i_1, \ldots, i_{n-m}\}$. Consider the set

$$\mathcal{J} = \Big\{(j_1, \ldots, j_{n-m}) : (\mathbf{S})^{j_k} = (\mathcal{L}_{\mathbf{A}}^{w})^{i_k} \text{ for all } k = 1, \ldots, n-m\Big\},$$

where $(\mathbf{X})^{y}$ denotes the y–th column of matrix $\mathbf{X}$. Intuitively, set $\mathcal{J}$ captures the parts of the permutations in T corresponding only to the $n-m$ indexes in I_2, and therefore $|\mathcal{J}|$ may be much smaller than $|T|$. For each sequence J of $\mathcal{J}$, we let $\mathbf{U}_2$ be the $(n-r) \times \ell$ matrix built from rows of $\mathbf{V}$ whose indexes are in J. For each of these possible values of $\mathbf{U}_2$, we compute the matrix $\mathbf{U}_1 = \left(\mathbf{A}_1^{-1}\mathbf{A}_2\right)\mathbf{U}_2$, and test if $\begin{bmatrix}\mathbf{U}_1 \\ \hline \mathbf{U}_2\end{bmatrix}$ corresponds to a permutation of the rows of $\mathbf{V}$. If this is indeed the case, then the secret matrix $\mathbf{V}_\pi$ is simply $\mathbf{V}_\pi = \begin{bmatrix}\mathbf{U}_1 \\ \hline \mathbf{U}_2\end{bmatrix}_{\phi^{-1}}$.

It is important to notice that, since we want to make $|\mathcal{J}|$ as low as possible, we sorted the set of indexes I so that, when defining I_1 and I_2, the columns of $\mathcal{L}_{\mathbf{A}}^{w}$ whose indexes are in I_2 tend to appear a lower number of times. In Section 5.3, we show how to estimate the size of $\mathcal{J}$.

5 Concrete Analysis of the Attack

In this section we estimate the attack complexity. We begin by analyzing, in the first three subsections, the work factor of the three components of the attack algorithm: building sets $\mathcal{L}_{\mathbf{A}}^{w}$ and $\mathcal{L}_{\mathbf{K}}^{w}$, matching the low weight vectors in these sets, and extracting the secret permutation from matchings. Then, we put these components together to give the complexity of the attack in Section 5.4. Finally, in Section 5.5, we show the performance of the attack in practice.

The work factor of attacks against PKP is typically stated in number of matrix-vector products, as it is the basic operation to test if a vector is in the kernel of a matrix. Even though binary PKP uses two matrices, we can see the rows of $\mathbf{V}$ as elements of $\mathbb{F}_{2^\ell}$ and, since ℓ is typically small, then the product $\mathbf{AV}$ can be seen as a matrix-vector multiplication where sum is replaced by a XOR.

5.1 Searching for codewords of small weight

Let us analyze the first step of the attack: the construction of sets $\mathcal{L}_{\mathbf{A}}^{w}$ and $\mathcal{L}_{\mathbf{K}}^{w}$. Each of these sets can be computed by searching exhaustively the whole set of $\binom{n}{w}$ possible vectors of n bits of weight w, and testing if they belong to $\mathcal{C}_{\mathbf{A}}$ and $\mathcal{C}_{\mathbf{K}}$. However, as we pointed in Section 4.1, we can do a lot better by using Stern's [22] algorithm. Consider a random $[n, k]$–linear code generated by matrix

[1] The reason why it is interesting to sort the indexes in this way is explained in the last paragraph of this section.

$\mathbf{G}$. Given parameters (p, q), Stern's algorithm first permutes the columns of $\mathbf{G}$ hoping to obtain a matrix $\hat{\mathbf{G}} = \phi(\mathbf{G})$, called a good permutation, for which there is a linear combination of its rows that has the form $\mathbf{c} = [\mathbf{c}_1|\mathbf{c}_2|\mathbf{c}_3|\mathbf{c}_4]$, such that $\mathrm{w}(\mathbf{c}_1) = \mathrm{w}(\mathbf{c}_2) = p$, component $\mathbf{c}_3$ is the zero vector of length q, and $\mathbf{c}_4$ has weight $\mathrm{w}(\mathbf{c}_4) = w - 2p$.

When such conditions are met, Stern's algorithm finds a vector $\mathbf{c}$ with such properties, which can then be permuted to give a vector $\mathbf{c}_{\phi^{-1}}$ of weight w in the code generated by $\mathbf{G}$. To compute the work factor of Stern's algorithm then, we have to take into account the average number of iterations until it chooses a good permutation $\hat{\mathbf{G}}$ and the average number of operations performed by the algorithm each time. Considering Finiasz and Sendrier's [7] approximation, which takes parameter $q \approx \log\binom{k/2}{p}$, the work factor of Stern's algorithm, considering the number of binary operations, until it gives us a random codeword of weight w in a random $[n, k]$–linear code is

$$\mathbf{BinOpsWF}^{(n,k,w)}_{\text{Stern}} \approx \min_{p} \frac{2q\binom{n}{w}}{\binom{n-k-q}{w-2p}\binom{k/2}{p}}.$$

Each time Stern's algorithm runs successfully, it finds a random codeword of weight w. Therefore we can model the expected number of iterations until all codewords are found as an instance of the coupon collector problem. Let us consider the time to build $\mathcal{L}^w_{\mathbf{A}}$. Each low weight vectors is modeled as a coupon, and we need to collect all $\ell_{\mathbf{A}}$ of them. Let C be the random variable that counts the number of low weight vectors we need to find before obtaining $\ell_{\mathbf{A}}$ different vectors. Then, it is well known that $\mathbb{E}(C) = \Theta(\ell_{\mathbf{A}} \log \ell_{\mathbf{A}})$. Furthermore, the upper tail estimate for the coupon collector problem ensures that

$$\Pr(C \leq \gamma_{\mathbf{A}} \ell_{\mathbf{A}} \log \ell_{\mathbf{A}}) \leq \ell_{\mathbf{A}}^{-\gamma_{\mathbf{A}}+1}.$$

Let $\mathbf{WF}_{\mathcal{L}^w_{\mathbf{A}}}$ be the work factor of building the set $\mathcal{L}^w_{\mathbf{A}}$, counted in number of binary matrix-vector multiplications. Since $\dim \mathbf{A} = m$, we can get an upper bound on $\mathbf{WF}_{\mathcal{L}^w_{\mathbf{A}}}$ as

$$\mathbf{WF}^{(n,m,w,\ell_{\mathbf{A}})}_{\mathcal{L}^w_{\mathbf{A}}} \leq \mathbf{BinOpsWF}^{(n,m,w,\ell_{\mathbf{A}})}_{\mathcal{L}^w_{\mathbf{A}}} = \gamma_{\mathbf{A}} (\ell_{\mathbf{A}} \log \ell_{\mathbf{A}}) \, \mathbf{BinOpsWF}^{(n,m,w)}_{\text{STERN}},$$

where $\gamma_{\mathbf{A}} > 1$ is chosen so that $\ell_{\mathbf{A}}^{-\gamma_{\mathbf{A}}+1}$ gives a small error probability.

Now we want do do the same thing for the construction of $\mathcal{L}^w_{\mathbf{K}}$. Let $\ell_{\mathbf{K}} = |\mathcal{L}^w_{\mathbf{K}}|$. and let us estimate $\ell_{\mathbf{K}}$. As usual in coding theory, to count elements of a given weight, we approximate the number of elements of weight w in a random code as a binomial distribution. Thus, out of the $\binom{n}{w}$ possible vectors of weight w, we expect that a fraction of $2^{\dim \mathbf{K}}/2^n$ belong to $\mathcal{C}_{\mathbf{K}}$. Since $\mathbf{K}$ is the left kernel matrix of $\mathbf{V}$, then $\dim \mathbf{K} = (n - \dim \mathbf{V}) = (n - \ell)$, and we can approximate the expected value of $\ell_{\mathbf{K}}$ as

$$\hat{\ell}_{\mathbf{K}} = \mathbb{E}(\ell_{\mathbf{K}}) \approx \frac{2^{n-l}}{2^n}\binom{n}{w} = 2^{-l}\binom{n}{w}. \tag{2}$$

Therefore, for some factor $\gamma_{\mathbf{K}} > 1$ we can define an upper bound on the work factor of building $\mathcal{L}_{\mathbf{K}}^{w}$ as

$$\mathbf{WF}_{\mathcal{L}_{\mathbf{K}}^{w}}^{(n,\ell,w,\ell_{\mathbf{A}})} \leq \gamma_{\mathbf{K}} \left(\hat{\ell}_{\mathbf{K}} \log \hat{\ell}_{\mathbf{K}}\right) \mathbf{BinOpsWF}_{\textsc{Stern}}^{(n,n-\ell,w)}.$$

Notice that if $\mathcal{L}_{\mathbf{K}}^{w}$ does not contain the permutations of all vectors of $\mathcal{L}_{\mathbf{A}}^{w}$, then the search will fail. Thus, factor $\gamma_{\mathbf{K}}$ must be chosen conservatively, but since $\hat{\ell}_{\mathbf{K}}$ is typically very large, the probability $\hat{\ell}_{\mathbf{K}}^{(1-\gamma_{\mathbf{K}})}$ of not collecting all vectors can be made negligible even for relatively small $\gamma_{\mathbf{K}}$.

5.2 Searching for matchings

In this section, we evaluate the number of paths that will be tested by the subgraph isomorphism algorithm. For this evaluation, we need to estimate the number of possible child nodes in each level.

Consider the case when the search is holding matrix $\mathbf{S}$ at level α. We want to estimate the size of set $\hat{P}_{\alpha+1}$ of possible rows to add to $\mathbf{S}$ in the next level of the search. In other words, we want to compute the number of vectors that survive the filter imposed by the line 5 of Algorithm 1. The first thing to notice is that the result of the filtering is exactly the same if we filter from $\mathbf{p} \in \mathcal{L}_{\mathbf{K}}^{w}$ instead of $\mathbf{p} \in P_i$, that is

$$\hat{P}_i = \left\{ \mathbf{p} \in P_i : \sigma\left(\left[\frac{\mathbf{S}}{\mathbf{p}}\right]\right) = \mathbf{H}_{i,\alpha} \right\} = \left\{ \mathbf{p} \in \mathcal{L}_{\mathbf{K}}^{w} : \sigma\left(\left[\frac{\mathbf{S}}{\mathbf{p}}\right]\right) = \mathbf{H}_{i,\alpha} \right\}.$$

The reason why the algorithm keeps updating the list $\mathcal{P} = \left(P_1, \ldots, P_{|\mathcal{L}_{\mathbf{A}}^{w}|}\right)$ of possible vectors in all levels, is solely for efficiency. Without it, the filtering would be very inefficient for nodes in lower levels down the search because it would have to run, every time, through set $\mathcal{L}_{\mathbf{K}}^{w}$, which may be very large.

Let $\mathbf{L}_{\alpha}$ be the matrix formed by the first α rows of $\mathcal{L}_{\mathbf{A}}^{w}$, and let $\mathbf{r}$ be the $(\alpha+1)$–th row of $\mathcal{L}_{\mathbf{A}}^{w}$. Now, using the definition of $\mathbf{H}_{i,\alpha}$, we want to estimate how many vectors $\mathbf{p}$ in $\mathcal{L}_{\mathbf{K}}^{w}$ satisfy $\sigma\left(\left[\frac{\mathbf{S}}{\mathbf{p}}\right]\right) = \sigma\left(\left[\frac{\mathbf{L}_{\alpha}}{\mathbf{r}}\right]\right)$.

One problem that makes estimating the number of child nodes difficult is that, since vectors in $\mathcal{L}_{\mathbf{K}}^{w}$ are low-weight codewords of a fixed linear code, the vectors in $\mathcal{L}_{\mathbf{K}}^{w}$ are not independently distributed. This is a common problem when analyzing bounds on weight distribution in coding theory, and as usual in the field, we overcome this problem by assuming that the set $\mathcal{L}_{\mathbf{K}}^{w}$ consists of vectors chosen uniformly at random over the vectors of length n and weight w.

Now, under our model, let us fix $\mathbf{L}_{\alpha}$ and estimate the probability $q_{\alpha+1}(\mathbf{L}_{\alpha})$ that vector $\hat{\mathbf{p}}$ of $\mathcal{L}_{\mathbf{K}}^{w}$ is a possible child node in $\hat{P}_{\alpha+1}$. Because of the way that the algorithm builds $\mathbf{S}$, its columns are the same as the ones of $\mathbf{L}_{\alpha}$, up to some permutation, and therefore

$$\begin{aligned} q_{\alpha+1}(\mathbf{L}_\alpha) &= \Pr\left(\sigma\left(\left[\frac{\mathbf{S}}{\hat{\mathbf{p}}}\right]\right) = \sigma\left(\left[\frac{\mathbf{L}_\alpha}{\mathbf{r}}\right]\right)\right) \\ &= \Pr\left(\sigma\left(\left[\frac{\mathbf{L}_\alpha}{\mathbf{p}}\right]\right) = \sigma\left(\left[\frac{\mathbf{L}_\alpha}{\mathbf{r}}\right]\right)\right), \end{aligned}$$

where $\mathbf{p}$ is a random n–bit vector of weight w.

The signatures of the two matrices will be the same if the columns above the non-null entries of $\mathbf{p}$ and $\mathbf{r}$ are equal, up to some permutation. Therefore, $q_{\alpha+1}(\mathbf{L}_\alpha)$ is simply the probability that two subsets of w columns drawn from $\mathbf{L}_\alpha$ are the same, up to some permutation. In the simple case when all columns of $\mathbf{L}_\alpha$ are unique, then $q_{\alpha+1}(\mathbf{L}_\alpha) = 1/\binom{n}{w}$. However, in general, $\mathbf{L}_\alpha$ may have non-unique columns, which occur with high probability for small values of α, since $\mathbf{L}_\alpha$ is sparse.

Let $\mathbf{R}$ be the $\alpha \times w$ matrix built by taking columns of $\mathbf{L}_\alpha$ whose indexes are in $\operatorname{supp}(\mathbf{r})$. Define two counting functions N and $N_\mathbf{R}$ that, given a column $\mathbf{c}$, output the number of times column $\mathbf{c}$ appears in matrices $\mathbf{L}_\alpha$ and $\mathbf{R}$, respectively. For each column $\mathbf{c}$, which should appear $N_\mathbf{R}(\mathbf{c})$ times in the columns above the non-null entries of $\mathbf{p}$, there are $\binom{N(\mathbf{c})}{N_\mathbf{R}(\mathbf{c})}$ ways in which different column indexes of $\mathbf{R}$ can be chosen. Therefore

$$q_{\alpha+1}(\mathbf{L}_\alpha) = \frac{1}{\binom{n}{w}} \prod_{\mathbf{c}\in\mathbb{F}_2^\alpha} \binom{N(\mathbf{c})}{N_\mathbf{R}(\mathbf{c})}.$$

To estimate the average attack performance, we want to compute the expected value $\overline{q}_{\alpha+1} = \mathbb{E}\left(q_{\alpha+1}(\mathbf{L}_\alpha)\right)$ when $\mathbf{L}_\alpha$ is obtained from a randomly generated key. This value can be easily estimated using simulations by sampling $\mathbf{L}_\alpha$ from the set of $\alpha \times n$ matrices in which each row has weight w. However, to give an analytic approximation, we face the problem of computing the expected value of the binomial coefficients over the random variables $N(\mathbf{c})$ and $N_\mathbf{R}(\mathbf{c})$ for each possible column $\mathbf{c}$. To deal with this problem, we use of the following rough approximation

$$\overline{q}_{\alpha+1} \approx \frac{1}{\binom{n}{w}} \prod_{\mathbf{c}\in\mathbb{F}_2^\alpha} \binom{\mathbb{E}\left(N(\mathbf{c})\right)}{\mathbb{E}\left(N_\mathbf{R}(\mathbf{c})\right)}.$$

To compute the expected values $\mathbb{E}\left(N(\mathbf{c})\right)$ and $\mathbb{E}\left(N_\mathbf{R}(\mathbf{c})\right)$, we consider $\mathbf{L}_\alpha$ as a random sparse matrix of density w/n as an approximation of the real case where each of its rows have a fixed weight w. Under this model, the probability that a random column of $\mathbf{L}_\alpha$ is equal to $\mathbf{c}$ depends only on its the weight $k = \mathrm{w}(\mathbf{c})$ and and the number α of rows in $\mathbf{L}_\alpha$. This probability is given by

$$p(k,\alpha) = \left(\frac{w}{n}\right)^k \left(1 - \frac{w}{n}\right)^{\alpha-k}. \tag{3}$$

Thus both $N(\mathbf{c})$ and $N_{\mathbf{R}}(\mathbf{c})$ follow binomial distributions with parameters $(n, p(\mathrm{w}(\mathbf{c}), \alpha))$ and $(w, p(\mathrm{w}(\mathbf{c}), \alpha))$, respectively. Therefore[2]

$$\begin{aligned}\overline{q}_{\alpha+1} &\approx \frac{1}{\binom{n}{w}} \prod_{\mathbf{c}\in\mathbb{F}_2^{\alpha}} \binom{np(\mathrm{w}(\mathbf{c}), \alpha)}{wp(\mathrm{w}(\mathbf{c}), \alpha)} \\ &\approx \frac{1}{\binom{n}{w}} \prod_{k=0}^{\alpha} \binom{np(k, \alpha)}{wp(k, \alpha)}^{\binom{\alpha}{k}}.\end{aligned}$$

One can then use this analytic approximation or simulations for $\overline{q}_{\alpha}$ to obtain the number of possible nodes in each level as $\left|\hat{P}_{\alpha}\right| = \overline{q}_{\alpha}\hat{\ell}_{\mathbf{K}}$, where $\hat{\ell}_{\mathbf{K}} = \mathbb{E}\left(|\mathcal{L}_{\mathbf{K}}^{w}|\right)$ is given approximately by Equation 2. Figure 2 shows how the analytic approximation and the value obtained by simulations compare with what is observed during a real attack. We can see that simulations can accurately be used to estimate $\overline{q}_{\alpha}$ and that the analytic estimate tends to overestimate the number of possible nodes in each level.

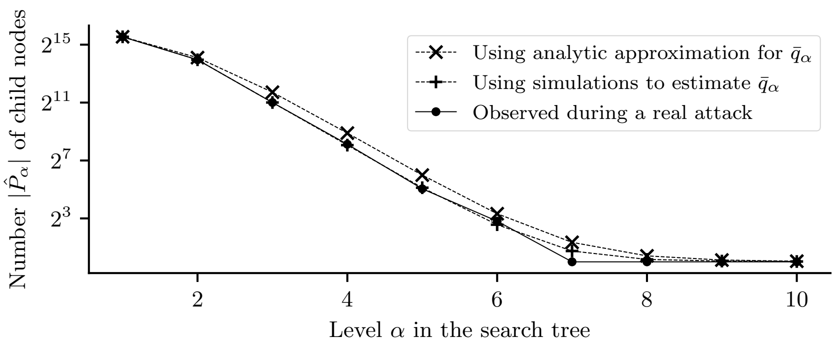

Fig. 2: Comparison of estimates on the average number of possible vectors to add in each level of the search. The attack parameters ($w = 8, \ell_{\mathbf{A}} = 10$) were used against the BPKP–76 parameter set.

The work factor of the search procedure, denoted by $\mathbf{WF}_{\text{SEARCH}}$, consists of the expected number of possible paths, which is given by

$$\mathbf{WF}_{\text{SEARCH}}^{(n,w,\ell_{\mathbf{A}})} = \prod_{\alpha=1}^{\ell_{\mathbf{A}}} \left|\hat{P}_{\alpha}\right| \approx \left(\hat{\ell}_{\mathbf{K}}\right)^{\ell_{\mathbf{A}}} \prod_{\alpha=1}^{\ell_{\mathbf{A}}} \overline{q}_{\alpha}.$$

[2] Recall, from Section 2, that binomials are defined over non-negative real numbers to allow our approximations.

5.3 Extracting permutations from matchings

We now analyze the complexity of the procedure that tries to extract the secret permutation after a matching is found. The main quantity we need to estimate is the number of permutations that the procedure needs to test each time it is called. Formally, we need to estimate the average size of set $\mathcal{J}$ for each parameter set (n, m, ℓ) when the scheme is attacked with attack parameters $(w, \ell_{\mathbf{A}})$.

Let I_1 and $I_2 = \{i_1, \ldots, i_{n-m}\}$ be the sets constructed from $\mathcal{L}_{\mathbf{A}}^{w}$ and $\mathbf{A}$ as described in Section 4.3. The first thing to notice is that $|\mathcal{J}|$ can be computed directly from matrix $\mathcal{L}_{\mathbf{A}}^{w}$, that is, it does not depend on a each $\mathbf{S}$. This is a consequence of the fact that, by construction, $\mathbf{S} = \tau(\mathcal{L}_{\mathbf{A}}^{w})$, for some column permutation τ. Formally, what we mean is that, since[3]

$$\begin{aligned}\mathcal{J} &= \Big\{(j_1, \ldots, j_{n-m}) : (\tau(\mathcal{L}_{\mathbf{A}}^{w}))^{j_k} = (\mathcal{L}_{\mathbf{A}}^{w})^{i_k} \text{ for all } k = 1, \ldots, n-m\Big\} \\ &= \Big\{(j_1, \ldots, j_{n-m})_{\tau} : (\mathcal{L}_{\mathbf{A}}^{w})^{j_k} = (\mathcal{L}_{\mathbf{A}}^{w})^{i_k} \text{ for all } k = 1, \ldots, n-m\Big\},\end{aligned}$$

then $|\mathcal{J}| = \Big|\Big\{(j_1, \ldots, j_{n-m}) : (\mathcal{L}_{\mathbf{A}}^{w})^{j_k} = (\mathcal{L}_{\mathbf{A}}^{w})^{i_k} \text{ for all } k = 1, \ldots, n-m\Big\}\Big|$, which does not depend on τ.

Thus we can model $|\mathcal{J}|$ as the number of arrangements of $n - m$ different balls, which may come from different boxes, under the restriction that each box will be sampled a fixed number of times. In this analogy, each box represents a set of indexes that correspond to equal columns in $\mathcal{L}_{\mathbf{A}}^{w}$. More formally, let $\mathbf{L}_2$ be the $\ell_{\mathbf{A}} \times (n-m)$ matrix formed by taking columns of $\mathcal{L}_{\mathbf{A}}^{w}$ whose indexes are in I_2. Define two counting functions N and N_2 that, given a column $\mathbf{c}$, output the number of times column $\mathbf{c}$ appears in matrices $\mathcal{L}_{\mathbf{A}}^{w}$ and $\mathbf{L}_2$, respectively. Then, we have

$$|\mathcal{J}| = \prod_{\mathbf{c} \in \mathcal{C}_2} \frac{N(\mathbf{c})!}{(N(\mathbf{c}) - N_2(\mathbf{c}))!}.$$

Now, let us consider the expected value of $\mathcal{J}$ when $\mathbf{A}$ is a random matrix such that $\mathcal{L}_{\mathbf{A}}^{w}$ contains $\ell_{\mathbf{A}}$ vectors of weight w. This number can easily be estimated by simulations, which perfectly correspond to what is observed in a real attack since, up to this point no simplification has been made. Furthermore, we can also give an analytic estimate using the very same ideas from the previous section. First we approximate this case by modeling $\mathcal{L}_{\mathbf{A}}^{w}$ as a random $\ell_{\mathbf{A}} \times n$ sparse matrix with density w/n, and let $p(k, \ell_{\mathbf{A}})$ denote the probability that a given column of $\mathcal{L}_{\mathbf{A}}^{w}$ is equal to a fixed column of weight k and height $\ell_{\mathbf{A}}$, as defined by Equation 3.

[3] Recall that $(\mathbf{X})^{i}$ denotes the i–th column of matrix $\mathbf{X}$.

Then, the rough approximation on $\mathbb{E}(|\mathcal{J}|)$ is given by

$$\begin{aligned}\mathbb{E}(|\mathcal{J}|) &\approx \prod_{\mathbf{c}\in\mathbb{F}_2^{\ell_{\mathbf{A}}}} \frac{\mathbb{E}(N(\mathbf{c}))!}{(\mathbb{E}(N(\mathbf{c})) - \mathbb{E}(N_2(\mathbf{c})))!} \\ &= \prod_{k=0}^{\ell_{\mathbf{A}}} \left(\frac{(np(k,\ell_{\mathbf{A}}))!}{(np(k,\ell_{\mathbf{A}}) - (n-m)p(k,\ell_{\mathbf{A}}))!}\right)^{\binom{\ell_{\mathbf{A}}}{k}} \\ &= \prod_{k=0}^{\ell_{\mathbf{A}}} \left(\frac{(np(k,\ell_{\mathbf{A}}))!}{(mp(k,\ell_{\mathbf{A}}))!}\right)^{\binom{\ell_{\mathbf{A}}}{k}}.\end{aligned}$$

Figure 3 shows how $|\mathcal{J}|$ rapidly decreases as larger values of $\ell_{\mathbf{A}}$ are used. It also provides a comparison between our analytic estimate on $\mathbb{E}(|\mathcal{J}|)$ and the observed values in our simulations. Notice that, for small values of $\ell_{\mathbf{A}}$, the analytic estimate tends to overestimate the real values of $|\mathcal{J}|$, but for sufficiently large $\ell_{\mathbf{A}}$, the estimate converges to the observed values. Now, since each sequence $\mathbb{E}(|J|)$ needs one matrix multiplication to be tested, we define the work factor of the permutation extraction procedure, as $\mathbf{WF}_{\text{PERMS}}^{(n,m,w,\ell_{\mathbf{A}})} = \mathbb{E}(|J|)$.

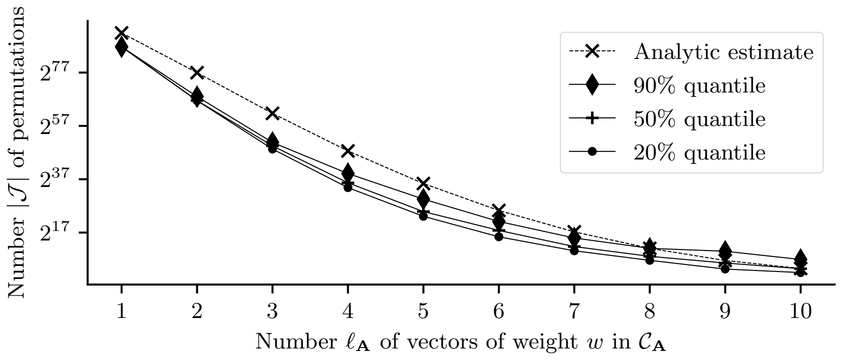

Fig. 3: The number of permutations to test after each matching considering the BPKP–76 parameter set. The attack parameter $w = 8$ was fixed, and the simulations were run for increasing values of parameter $\ell_{\mathbf{A}}$.

5.4 Attack Complexity

This section builds upon the three previous sections to explicitly state the attack complexity and the fraction of keys that can be attacked for different attack parameters $(w, \ell_{\mathbf{A}})$.

The full complexity of the attack is given by the following lemma.

Lemma 1. *Let (n, m, ℓ) be a binary PKP parameter set. Then, the work factor of the attack with parameters $(w, \ell_{\mathbf{A}})$ is given as*

$$\mathbf{WF}^{(n,m,\ell,w,\ell_{\mathbf{A}})}_{\text{Attack}} = \mathbf{WF}^{(n,m,\ell,w,\ell_{\mathbf{A}})}_{\text{LowWeightSets}} + \left(\mathbf{WF}^{(n,w,\ell_{\mathbf{A}})}_{\text{Search}}\right)\left(\mathbf{WF}^{(n,m,w,\ell_{\mathbf{A}})}_{\text{Perms}}\right).$$

Proof. The complexity of the attack is given by summing the costs of building the sets of vectors of small weight $\mathcal{L}^w_{\mathbf{A}}$ and $\mathcal{L}^w_{\mathbf{K}}$, and the complexity of the key recovery algorithm. The cost of the key recovery algorithm is computed as follows. Remember that, for each path, from the root to one leaf, the number of permutations we have to test is given as $\mathbf{WF}^{(n,m,w,\ell_{\mathbf{A}})}_{\text{Perms}}$. Since the average number of paths is $\mathbf{WF}^{(n,w,\ell_{\mathbf{A}})}_{\text{Search}}$, the complexity of the key recovery algorithm is simply the product $\left(\mathbf{WF}^{(n,w,\ell_{\mathbf{A}})}_{\text{Search}}\right)\left(\mathbf{WF}^{(n,m,w,\ell_{\mathbf{A}})}_{\text{Perms}}\right)$. $\square$

Figure 4 shows how $\mathbf{WF}^{(n,m,\ell,w,\ell_{\mathbf{A}})}_{\text{Attack}}$ varies with respect to the attack parameters used, when attacking BPKP–76 parameter set. To estimate the work factor of the attack, we used simulations[4] for $\mathbf{WF}^{(n,w,\ell_{\mathbf{A}})}_{\text{Search}}$ and analytic estimation for $\mathbf{WF}^{(n,m,w,\ell_{\mathbf{A}})}_{\text{Perms}}$. Notice how, as $\ell_{\mathbf{A}}$ gets larger, the work factor stabilizes. This happens because the number of permutations to test gets closer to 1. Furthermore, it is clear that when w is smaller, the attack is more efficient, which happens because, in this case, $|\mathcal{L}^w_{\mathbf{K}}|$ is smaller, which makes the search much faster. The problem however, is that the attack parameters for which the attack is most efficient occur with lower probability, as we elaborate next.

Lemma 1 does not say anything about the fraction of keys that one can attack using parameters $(w, \ell_{\mathbf{A}})$. To compute this fraction, we have to take into account the probability that a public matrix $\mathbf{A}$, selected at random, generates a code with at least $\ell_{\mathbf{A}}$ codewords of weight w. This is considered in the following lemma.

Lemma 2. *Let (n, m, ℓ) be a binary PKP parameter set. Then, the fraction of keys against which the attack is effective when using parameters $(w, \ell_{\mathbf{A}})$ is given as*

$$\mathbf{KF}^{n,m,\ell,w,\ell_{\mathbf{A}}}_{\text{Attack}} \approx 1 - e^{-\lambda} \sum_{k=0}^{\ell_{\mathbf{A}}-1} \frac{\lambda^k}{k!}, \tag{4}$$

where $\lambda = \binom{n}{w} 2^{m-n}$.

Proof. Take a random matrix $\mathbf{A}$, generated with parameters (n, m, ℓ). Let L_w be the random variable that represents the number of vectors of weight w in the code generated by matrix $\mathbf{A}$. Since the code generated by $\mathbf{A}$ is a random code, we can approximate L_w by a binomial distribution which samples $\binom{n}{w}$ vectors and each one of them is in the code with probability 2^{m-n}.

[4] Even though the analytic approach is useful to estimate the number of nodes in each level, the errors would accumulate exponentially in the product necessary to compute the work factor of the search.

The probability that $L_w \geq \ell_{\mathbf{A}}$ would be then simply $\left(1 - \sum_{k=0}^{\ell_{\mathbf{A}}-1} \Pr(L_w = k)\right)$. However, the probability mass function of the binomial can be costly to compute for some values of k, since N may be very large, and $N-k$ appears as an exponent. But, for large N and small probability 2^{m-n}, the binomial may be approximated as a Poisson distribution with parameter $\lambda = \binom{n}{w}2^{m-n}$. Then, the approximation given as Equation 4 is easily achieved by considering the cumulative distribution function of the Poisson distribution, instead of the binomial. □

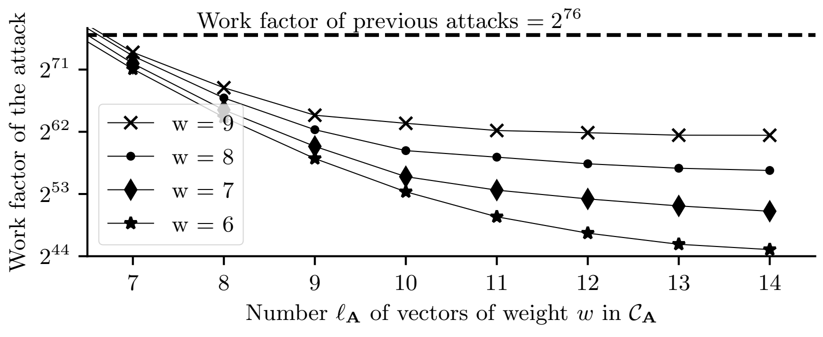

Fig. 4: Work factor of the attack against BPKP–76 parameter set using different attack parameters $(w, \ell_{\mathbf{A}})$.

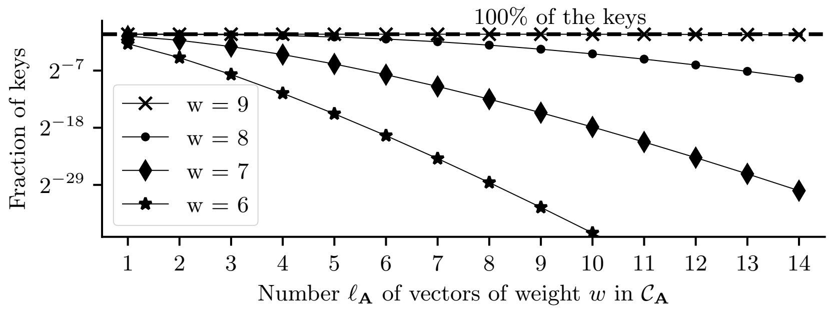

Fig. 5: Fraction of the keys generated with BPKP–76 parameter set against which the attack is successful using different attack parameters $(w, \ell_{\mathbf{A}})$.

Figure 5 shows the effect of parameters $(w, \ell_{\mathbf{A}})$ in the fraction of keys that we can attack. The first thing to notice is that large $\ell_{\mathbf{A}}$ and small w tend to occur with smaller probability. Now we can combine both Figures 4 and 5 to understand the power of the attack. For example, considering parameters $(w = 7, \ell_{\mathbf{A}} = 10)$, we can attack about 1 in each 150.000 keys of BPKP–76 with less than 2^{55} operations, and about 100% of all keys can be recovered using 2^{62} operations.

5.5 Experimental Results

To validate our proposed attack, we implemented it in SageMath and in C language, using M4RI [14] library for efficient binary linear algebra computations. The source code is publicly available at `www.ime.usp.br/~tpaiva`.

Table 2 shows the performance of the attack against BPKP-76. To obtain empirical estimates on its performance, we considered the average number of clock cycles for the smallest level $\hat{\alpha}$ in the search for which we can get a significant number of samples. Then, the empirical estimate is given by the product between this average number of clock cycles and the average number of total nodes in level $\hat{\alpha}$. Thus smallest values of α give more accurate results. The values of $\ell_{\mathbf{A}}$ are chosen as to guarantee that the number of permutations to test is within reasonable computational limits and so that $\hat{\alpha} \leq 3$.

Notice how, in general, the estimates on the work factor of the attack tend to overestimate the real complexity of the attack. The main explanation for this fact seems to be that, for sufficiently large $\ell_{\mathbf{A}}$, the algorithm rarely enters in a leaf node, which is where most of the matrix product operations occur. This is exemplified by the decay, shown in Figure 2, of the curve representing the observed number of nodes in each level during a real attack, where, after level $\alpha = 7$, a node rarely has more than one child.

Table 2: Estimates on the number of clock cycles necessary for a successful attack.

w	$\ell_{\mathbf{A}}$	$\hat{\alpha}$	Fraction of keys	Predicted work factor (matrix-vector products)	Empirical estimate (clock cycles)
5	14	1	0	$2^{39.46}$	$2^{34.39}$
6	11	2	$2^{-43.32}$	$2^{49.75}$	$2^{47.58}$
7	10	2	$2^{-17.86}$	$2^{55.84}$	$2^{48.62}$
8	9	3	$2^{-2.88}$	$2^{62.28}$	$2^{60.54}$
9	9	3	$2^{-0.00}$	$2^{64.16}$	$2^{62.31}$

6 Asymptotic Analysis

In the previous section, a detailed analysis of the attack is presented. However, the concrete analysis fails to provide a general idea of how the complexity grows, as the complexity of the components are not easy to simplify and must be computed using iterative procedures for products of binomial coefficients. Therefore we aim, in this section, to give simpler and closed expressions for the asymptotic attack complexity, but without compromising the reliability of the analysis.

6.1 Asymptotic growth of the attack parameters

First let us recall the growth of parameters m and ℓ with respect to n. To ensure that the binary PKP instances are difficult to solve on average, we need that, out of the $n!$ possible permutations of the rows of $\mathbf{V}$, about only one of them is in the kernel of $\mathbf{A}$. The dimension of $\mathbf{A}$ is m, which means the probability that a random vector belongs to the kernel is $2^{(n-m)}/2^n = 2^{-m}$. Therefore, since the binary PKP is solved only when all ℓ column vectors of $\mathbf{V}_\pi$ are in the kernel of $\mathbf{A}$, then $n!2^{-m\ell} \approx 1$. As suggested by Lampe and Patarin [13], we consider that m and l should be roughly the same size

$$m \approx l \approx \sqrt{\log n!} = O(\sqrt{n \log n}). \tag{5}$$

From Equation 5, we see that the dimension m of the code generated by $\mathbf{A}$ grows much slower than its size n. Intuitively then, as n gets larger, it gets harder to obtain codewords of weight much smaller than $n/2$, because of the small dimension. Since we need to deal with values of w close to $n/2$, we are interested in using the following lemma that gives approximations on binomial coefficients $\binom{n}{w}$ under this regime.

Lemma 3 (Eq. 5.41 [21]). *Let n and w be positive integers such that $|n/2 - w| = o(n^{2/3})$. Then*

$$\binom{n}{w} \sim 2^n \sqrt{\frac{2}{n\pi}} e^{\frac{-(n-2w)^2}{2n}}.$$

□

We are now ready to show, in the following lemma, how to carefully choose values of w such that Lemma 3 ensures us that $\mathcal{L}^w_{\mathbf{A}}$ has a reasonable number of vectors.

Lemma 4. *Take the attack parameter w as*

$$w = \left\lfloor \frac{n}{2} - \sqrt{\frac{mn^{4/5}}{2\log e}} \right\rceil. \tag{6}$$

Then, on average, the attack can effectively use parameters $(w, \ell_{\mathbf{A}})$ when $\ell_{\mathbf{A}}$ is smaller than

$$\ell_{\mathbf{A}} \leq \left(\sqrt{\frac{2}{\pi}}\right) 2^{\left(m\left(1-n^{(-1/5)}\right)-(\log n)/2\right)}.$$

Proof. Let $\mathbf{A}$ be an $m \times n$ random binary PKP public matrix. The attack parameters $(w, \ell_{\mathbf{A}})$ are effective when $\ell_{\mathbf{A}}$ is smaller than or equal to the number of vectors of weight w in the code generated by $\mathbf{A}$. Therefore, on average, the attack works when

$$\ell_{\mathbf{A}} \leq 2^{m-n}\binom{n}{w}.$$

Now take w as defined by Equation 6, and notice that, since $m = O\left(\sqrt{n \log n}\right)$, then

$$\begin{aligned}|n/2 - w| &= \sqrt{\frac{mn^{4/5}}{2\log e}} = O\left(\sqrt{n^{4/5}\sqrt{n\log n}}\right)\\ &= O\left(n^{13/20}\left(\log n\right)^{1/4}\right) = o\left(n^{13/20+\epsilon}\right)\\ &= o\left(n^{2/3}\right).\end{aligned}$$

Therefore we can use Lemma 3 to obtain the approximation

$$2^{m-n}\binom{n}{w} \approx 2^{m-n}\left(2^n\sqrt{\frac{2}{n\pi}}e^{\frac{-(n-2w)^2}{2n}}\right) = 2^m\left(\sqrt{\frac{2}{n\pi}}e^{\frac{-(n-2w)^2}{2n}}\right).$$

But notice that

$$e^{\frac{-(n-2w)^2}{2n}} = e^{\frac{-(n/2-w)^2}{n/2}} = \exp\left(-\frac{1}{n/2}\sqrt{\frac{mn^{4/5}}{2\log e}}^{2}\right) = \exp\left(-\frac{mn^{-1/5}}{\log e}\right).$$

That is

$$e^{\frac{-(n-2w)^2}{2n}} = 2^{-mn^{-1/5}}. \tag{7}$$

Therefore the attack is effective for

$$\ell_{\mathbf{A}} \leq 2^m\left(\sqrt{\frac{2}{n\pi}}2^{-mn^{-1/5}}\right) = \left(\sqrt{\frac{2}{\pi}}\right)2^{\left(m\left(1-n^{(-1/5)}\right)-(\log n)/2\right)}.$$

□

Notice that when w is chosen according to the lemma above, then w/n approaches $1/2$ when n gets larger. This motivates the following corollary, which has an important role in simplifying the analysis.

Corollary 1. *As n gets larger and w is taken as in Lemma 4, the values of $p(k,\alpha)$ stop depending on k, and we have*

$$p(k,\alpha) = \left(\frac{1}{2}\right)^k\left(1-\frac{1}{2}\right)^{\alpha-k} = 2^{-\alpha}.$$

□

As a first application of Corollary 1, we show that, for sufficiently large n, we do not need $\ell_{\mathbf{A}}$ to be very large. With roughly $\ell_{\mathbf{A}} \approx \log n$, the number $\mathbf{WF}_{\text{PERMS}}$ of permutations to test after each matching is very close to 1.

Lemma 5. *Consider binary PKP parameters (n, m, ℓ). Take attack parameters w as in Lemma 4 and $\ell_{\mathbf{A}} \geq \lceil \log n \rceil$. Then, for sufficiently large values of n, the average number of permutations to test after each matching is*

$$\mathbf{WF}_{\text{PERMS}} = 1.$$

Proof. From our concrete analysis, we know that

$$\mathbf{WF}_{\text{PERMS}} = \prod_{k=0}^{\ell_{\mathbf{A}}} \left(\frac{(np(k, \ell_{\mathbf{A}}))!}{(mp(k, \ell_{\mathbf{A}}))!} \right)^{\binom{\ell_{\mathbf{A}}}{k}}.$$

But Corollary 1 tells us that $p(k, \ell_{\mathbf{A}}) \approx 2^{-\ell_{\mathbf{A}}}$ when n is large. Therefore,

$$\mathbf{WF}_{\text{PERMS}} = \prod_{k=0}^{\ell_{\mathbf{A}}} \left(\frac{(n2^{-\ell_{\mathbf{A}}})!}{(m2^{-\ell_{\mathbf{A}}})!} \right)^{\binom{\ell_{\mathbf{A}}}{k}} = \prod_{k=0}^{\ell_{\mathbf{A}}} \left(\frac{(n2^{-\lceil \log n \rceil})!}{(m2^{-\lceil \log n \rceil})!} \right)^{\binom{\lceil \log n \rceil}{k}} = 1.$$

□

It is important to understand that the lemma above needs a relatively large n, because it uses Corollary 1. Therefore, to lower the number of permutations to test after each matching when attacking small values of n, we typically want to use $\ell_{\mathbf{A}}$ near the maximum provided by Lemma 4. Notice that even for relatively small values of n, there are usually more than $\log n$ vectors of weight w in the code generated by $\mathbf{A}$. For example, when $n = 38$ we have

$$\log(n) \approx 5.25 < 5.99 \approx \sqrt{\tfrac{2}{\pi}} 2^{m(1-n^{-1/5})-(\log n)/2}.$$

We are now ready to derive the asymptotic complexity of $\mathbf{WF}_{\text{SEARCH}}$, which is the most critical step of the attack.

6.2 Searching for matchings

Let us begin by deriving an asymptotic bound on the number of child nodes in each level of the search tree.

Lemma 6. *Take the attack parameter w as in Lemma 4. Then, for sufficiently large values of n, the number of child nodes in each level α of the search is given as*

$$\left|\hat{P}_{\alpha+1}\right| = \begin{cases} 2^{n-l-mn^{-1/5}} \left(2^{\alpha 2^{\alpha}/2}\right) \sqrt{\frac{2}{n\pi}}^{2^{\alpha}} & \text{if } \alpha \leq (\lceil \log n \rceil - 2); \\ 1 & \text{otherwise.} \end{cases}$$

Proof. By our concrete analysis, we know that $\left|\hat{P}_{\alpha+1}\right|$ is given as

$$\begin{aligned} \left|\hat{P}_{\alpha+1}\right| = \hat{\ell}_{\mathbf{K}} \overline{q}_{\alpha+1} &= \left(2^{-l}\binom{n}{k}\right) \frac{1}{\binom{n}{w}} \prod_{k=0}^{\alpha} \binom{np(k,\alpha)}{wp(k,\alpha)}^{\binom{\alpha}{k}} \\ &= 2^{-l} \prod_{k=0}^{\alpha} \binom{np(k,\alpha)}{wp(k,\alpha)}^{\binom{\alpha}{k}}. \end{aligned}$$

Using Corollary 1, we can simplify the above expression as

$$\begin{aligned}\left|\hat{P}_{\alpha+1}\right| &= 2^{-l}\prod_{k=0}^{\alpha}\binom{n2^{-\alpha}}{w2^{-\alpha}}^{\binom{\alpha}{k}} = 2^{-l}\binom{n2^{-\alpha}}{w2^{-\alpha}}^{\left(\sum_{k=0}^{\alpha}\binom{\alpha}{k}\right)}\\ &= 2^{-l}\binom{n2^{-\alpha}}{w2^{-\alpha}}^{2^{\alpha}}.\end{aligned}$$

Now, if $\alpha \geq (\lceil \log n \rceil - 1)$, then $w2^{-\alpha} < 1$, and

$$\binom{n2^{-\alpha}}{w2^{-\alpha}}^{2^{\alpha}} \leq \binom{n2^{-\alpha}}{\frac{n}{2}2^{-\alpha}}^{2^{\alpha}} \approx 1.$$

Therefore, we can focus on approximating the case when $\alpha \leq (\lceil \log n \rceil - 2)$. Remember that w is close to $n/2$, thus we can use Lemma 3 to get the approximation

$$\begin{aligned}\binom{n2^{-\alpha}}{w2^{-\alpha}} &\approx 2^{n2^{-\alpha}}\sqrt{\frac{2}{n2^{-\alpha}\pi}}e^{\frac{-(n2^{-\alpha}-2w2^{-\alpha})^2}{2n2^{-\alpha}}}\\ &= 2^{n2^{-\alpha}}2^{\alpha/2}\sqrt{\frac{2}{n\pi}}e^{\frac{-(n-w)^2}{2n}2^{-\alpha}}.\end{aligned}$$

Recall Equation 7, which lets us further simplify the expression above as

$$\binom{n2^{-\alpha}}{w2^{-\alpha}} = 2^{n2^{-\alpha}}2^{\alpha/2}\sqrt{\frac{2}{n\pi}}\left(2^{-mn^{-1/5}}\right)^{2^{-\alpha}}.$$

Now, getting back to $\left|\hat{P}_{\alpha+1}\right|$, we have

$$\begin{aligned}\left|\hat{P}_{\alpha+1}\right| &= 2^{-l}\binom{n2^{-\alpha}}{w2^{-\alpha}}^{2^{\alpha}}\\ &= 2^{-l}\left(2^{n2^{-\alpha}}2^{\alpha/2}\sqrt{\frac{2}{n\pi}}\left(2^{-mn^{-1/5}}\right)^{2^{-\alpha}}\right)^{2^{\alpha}}\\ &= 2^{n-l-mn^{-1/5}}\left(2^{\alpha 2^{\alpha}/2}\right)\sqrt{\frac{2}{n\pi}}^{2^{\alpha}}.\end{aligned}$$

□

Now that we have bounded the number of nodes in each level, we are ready to give the asymptotic bound on the search procedure.

Lemma 7. *Take attack parameters w and $\ell_{\mathbf{A}} \geq \lceil \log n \rceil$ as in Lemma 4. Then, the asymptotic work factor of the search is given as*

$$\mathbf{WF}_{\textsc{Search}} \approx 2^{\left(n-l-mn^{-1/5}\right)(\lceil \log n \rceil - 1) - 0.91n + \frac{1}{2}\log n + 1.33}.$$

Proof. From our concrete analysis, we know that the complexity of the search is the product of the number of nodes in each level of the search. Furthermore, Lemma 6 says that we only need to compute these values for $\alpha \leq \lceil \log n \rceil - 2$, because after this point, typically there is at most one possible child node. Therefore, the complexity of the search is given as

$$\begin{aligned}
\mathbf{WF}_{\textsc{Search}} &= \prod_{\alpha=0}^{\ell_{\mathbf{A}}-1} \left|\hat{P}_{\alpha+1}\right| = \prod_{\alpha=0}^{\lceil \log n \rceil - 2} \left|\hat{P}_{\alpha+1}\right| \\
&= \prod_{\alpha=0}^{\lceil \log n \rceil - 2} \left(2^{n-l-mn^{-1/5}} \left(2^{\alpha 2^{\alpha}/2}\right) \sqrt{\frac{2}{n\pi}}^{2^{\alpha}} \right) \\
&= 2^{\left(n-l-mn^{-1/5}\right)(\lceil \log n \rceil - 1)} 2^{\left(\sum_{\alpha=0}^{\lceil \log n \rceil - 2} \alpha 2^{\alpha}/2\right)} \left(\frac{2}{n\pi}\right)^{\left(\sum_{\alpha=0}^{\lceil \log n \rceil - 2} 2^{\alpha}/2\right)} \\
&= 2^{\left(n-l-mn^{-1/5}\right)(\lceil \log n \rceil - 1)} 2^{\left(1+\frac{n}{4}(\lceil \log n \rceil - 3)\right)} 2^{\left(\log \frac{2}{n\pi}\right)(n/4-1/2)} \\
&\approx 2^{\left(n-l-mn^{-1/5}\right)(\lceil \log n \rceil - 1) - 0.91n + \frac{1}{2}\log n + 1.33}.
\end{aligned}$$

□

6.3 Asymptotic Complexity of the Attack

We are almost ready to complete the asymptotic analysis of the attack. The only missing component to consider is $\mathbf{WF}_{\textsc{LowWeightSets}}$. Using the bruteforce algorithm, one needs to test, for all $\binom{n}{w} = O(2^n)$ possible vectors of weight w, if they are in the space generated by $\mathbf{A}$ or in the left kernel of $\mathbf{V}$. Therefore, the complexity of building sets $\mathcal{L}^w_{\mathbf{A}}$ and $\mathcal{L}^w_{\mathbf{K}}$ is

$$\mathbf{WF}_{\textsc{LowWeightSets}} = O(2^n).$$

We can then combine the result above with Lemmas 5 and 7 to obtain the complexity of the attack, as given next.

Theorem 1. *Take attack parameters w and $\ell_{\mathbf{A}} \geq \lceil \log n \rceil$ as in Lemma 4. Then, the asymptotic work factor of the attack is given as*

$$\begin{aligned}
\mathbf{WF}_{\textsc{Attack}} &= \mathbf{WF}_{\textsc{LowWeightSets}} + (\mathbf{WF}_{\textsc{Search}})(\mathbf{WF}_{\textsc{Perms}}) \\
&= O\left(2^{\left(n-l-mn^{-1/5}\right)(\lceil \log n \rceil - 1) - 0.91n + \frac{1}{2}\log n}\right).
\end{aligned}$$

□

Figure 6 shows how the asymptotic complexity presented above compares with the simulations based on the concrete analysis. We can see that the asymptotic estimate appears to be realistic, even though the ceiling operation used for $\lceil \log n \rceil$ makes the function rapidly increase when $n-1$ is a power of 2, and then decrease until the next power of 2 is found.

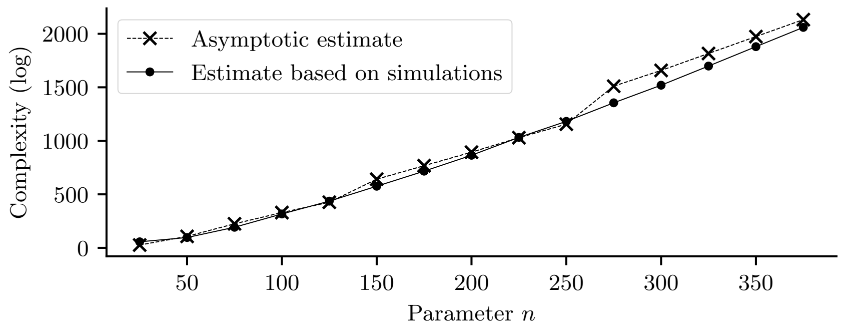

Fig. 6: Asymptotic complexity of the attack.

Figure 7 shows an asymptotic comparison between our algorithm and the one by Koussa et al. [12]. Even though their algorithm is currently the best generic algorithm for solving PKP in every field, we can see that our algorithm has a considerable advantage in the binary case. To help us visualize the asymptotic growth of our attack, we consider a smooth version of the estimate that consists in using $\log n$ instead of $\lceil \log n \rceil$ in the expression provided in Theorem 1.

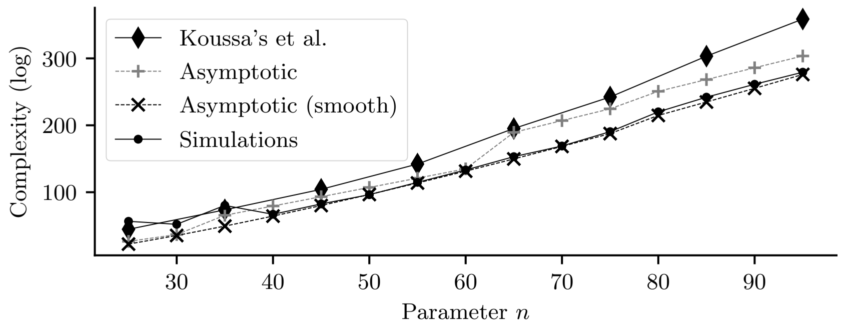

Fig. 7: Comparison between our attack and the one by Koussa et al. [12].

7 On Secure Parameters for Binary PKP

A conservative approach to select parameters for binary PKP, considering security level λ, would be to choose them in such a way that no class of keys that occurs with probability greater than $2^{-\lambda}$ should be attacked with less 2^{λ} operations. Furthermore, the choice of parameters should consider the use of binary PKP when building a signature scheme, and, as such, should aim to minimize not only the key sizes, but also signature sizes and the computational cost to sign and verify each signature.

The safest possible choice of parameters would be the ones that make it difficult to even build sets $\mathcal{L}_{\mathbf{A}}^{w}$ and $\mathcal{L}_{\mathbf{K}}^{w}$. If we take schemes that rely on the difficulty of finding small weight codewords such as MDPC [16], this would result in a very large matrix $\mathbf{A}$. This, however, would have a very negative impact on performance, key sizes and signature length.

A less conservative approach is to scale parameters (n, m, ℓ) and compute $\mathbf{WF}_{\text{Attack}}^{(n,m,\ell,w,\ell_{\mathbf{A}})}$ and $\mathbf{KF}_{\text{Attack}}^{(n,m,\ell,w,\ell_{\mathbf{A}})}$ for different attack parameters $(w, \ell_{\mathbf{A}})$. The search is efficient and can be done with the code that we provide. However, it is important to notice that it seems to be early to state sets of parameters for BPKP, as there may be some opportunities to improve this attack, which could thwart the security of parameters suggested without careful consideration. Our recommendation therefore is to avoid the Binary PKP, and more generally, the PKP using small fields for matrix $\mathbf{A}$, for which the search for low weight codewords can be done efficiently.

8 Conclusion and Future Work

In this paper, we present the first attack that targets binary PKP and provide a detailed analysis on the attack's components. The attack is practical and we provide an implementation of the attack in SageMath and C. Furthermore, the attack shows an inherently weakness of PKP using small fields, and we recommend that binary PKP be avoided while its security is not well understood against this new type of attack.

For future work, we plan to extend this attack to the original PKP, hoping to better understand what is the minimum finite field size p that can used securely. Furthermore, we believe that there are some opportunities to improve this attack. For example, it may be possible to increase the fraction of keys that one can attack by considering different parameters w simultaneously, or one can try to reduce the complexity of matching by introducing heuristics.

Acknowledgments

The authors would like to thank Augusto C. Ferrari for his helpful comments on earlier drafts of this paper. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance

Code 001. This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

References

1. Baritaud, T., Campana, M., Chauvaud, P., Gilbert, H.: On the security of the permuted kernel identification scheme. In: Annual International Cryptology Conference. pp. 305–311. Springer (1992)
2. Bernstein, D.J., Lange, T., Peters, C.: Smaller decoding exponents: ball-collision decoding. In: Annual Cryptology Conference. pp. 743–760. Springer (2011)
3. Beullens, W., Faugère, J.C., Koussa, E., Macario-Rat, G., Patarin, J., Perret, L.: PKP-based signature scheme. In: International Conference on Cryptology in India. pp. 3–22. Springer (2019)
4. Chen, L., Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on post-quantum cryptography. US Department of Commerce, National Institute of Standards and Technology (2016)
5. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE transactions on pattern analysis and machine intelligence **26**(10), 1367–1372 (2004)
6. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 186–194. Springer (1986)
7. Finiasz, M., Sendrier, N.: Security bounds for the design of code-based cryptosystems. In: Matsui, M. (ed.) Advances in Cryptology – ASIACRYPT 2009. pp. 88–105. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York (1979)
9. Georgiades, J.: Some remarks on the security of the identification scheme based on permuted kernels. Journal of Cryptology **5**(2), 133–137 (1992)
10. Jaulmes, É., Joux, A.: Cryptanalysis of PKP: a new approach. In: International Workshop on Public Key Cryptography. pp. 165–172. Springer (2001)
11. Joux, A., Lercier, R.: "Chinese & Match", an alternative to Atkin's "Match and Sort" method used in the SEA algorithm. Mathematics of computation **70**(234), 827–836 (2001)
12. Koussa, E., Macario-Rat, G., Patarin, J.: On the complexity of the Permuted Kernel Problem. IACR Cryptology ePrint Archive **2019**, 412 (2019)
13. Lampe., R., Patarin., J.: Analysis of some natural variants of the PKP algorithm. In: Proceedings of the International Conference on Security and Cryptography - Volume 1: SECRYPT, (ICETE 2012). pp. 209–214. INSTICC, SciTePress (2012). https://doi.org/10.5220/0004012202090214
14. M. Albrecht and G. Bard: The M4RI Library – Version 20121224. The M4RI Team (2012)
15. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. Deep Space Network Progress Report **44**, 114–116 (1978)
16. Misoczki, R., Tillich, J.P., Sendrier, N., Barreto, P.S.: MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In: Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on. pp. 2069–2073. IEEE (2013)

17. Patarin, J., Chauvaud, P.: Improved algorithms for the permuted kernel problem. In: Annual International Cryptology Conference. pp. 391–402. Springer (1993)
18. Poupard, G.: A realistic security analysis of identification schemes based on combinatorial problems. European transactions on telecommunications **8**(5), 471–480 (1997)
19. Sansone, P.F.C., Vento, M.: An improved algorithm for matching large graphs. In: Proc. of the 3rd IAPR-TC-15 International Workshop on Graph-based Representations (2001)
20. Shamir, A.: An efficient identification scheme based on permuted kernels. In: Conference on the Theory and Application of Cryptology. pp. 606–609. Springer (1989)
21. Spencer, J.: Asymptopia, vol. 71. American Mathematical Soc. (2014)
22. Stern, J.: A method for finding codewords of small weight. In: International Colloquium on Coding Theory and Applications. pp. 106–113. Springer (1988)
23. Ullmann, J.R.: An algorithm for subgraph isomorphism. Journal of the ACM (JACM) **23**(1), 31–42 (1976)