On the Complexity of the Permuted Kernel Problem

Abstract. In 1989, A. Shamir [1] introduced an interesting public-key scheme of a new nature, a Zero-Knowledge (ZK) Identification scheme, based on PKP: the Permuted Kernel Problem. PKP is an NP-hard [2] algebraic problem which has been extensively studied [7,8,9,5,10]. Among all the attacks, the problem PKP is in spite of the research effort, still exponential. This problem was used to develop an Identification Scheme (IDS) which has a very efficient implementation on low-cost smart cards [1].
There has been recently a renewed interest in PKP-based cryptography due to post quantum security considerations, simple security proofs, and the design of new PKP-based signature algorithm [12]. In 2018 and through the Fiat-Shamir transform [3], the PKP-IDS was used to construct a post-quantum signature scheme [12] which was submitted to a Chinese competition for the design of post-quantum cryptographic algorithms (organized by the Chinese Association CACR). This latter was improved in [13].
The aim of this document is two-fold. First, we investigate the complexity of the combinatorial problem - namely PKP. We also present a summary of previously known algorithms devoted to solve this problem. Contrary to what is shown in [10], and after a thorough analysis of the State-of-the-art attacks of PKP, we claim that the Joux-Jaulmes attack [10] is not the most efficient algorithm for solving PKP. In fact, the complexity of the Joux-Jaulmes attack underestimate the amount of certain important phase of the algorithm.
Second, we examine the complexity given by various algorithms, specifically the ones introduced by Patarin-Chauvaud [9] and Poupard [5]. It is relatively complex to obtain a general complexity formula due to the very numerous variants. However, we have been able to develop a program and provide its approximate space and time complexities which allow us to identify hard instances and secure sets of parameters of this problem with respect to the best attack currently known.

Keywords: Cryptanalysis · Identification scheme · Public-key signature · Complexity · Post-quantum cryptography · Permuted Kernel Problem.

1 Introduction

In this paper we focus on the analysis of the Permuted Kernel Problem (PKP). PKP is the problem of finding a permutation of a known vector such that the resulting vector is in the kernel of a given matrix. It was proved to be an NP-hard combinatorial problem [1].
PKP requires simple operations which involve basic linear algebra computations. Due to its simplicity, the problem has received significant attentions from theory and application in cryptography. Here, we study the theoretical analysis behind the PKP problem over a finite field. There are no new reported attacks on PKP which makes the construction of schemes based on hard instances of this problem more applicable.

We are essentially concerned about this problem because it can be used to build a post-quantum signature scheme based on the hardness of solving random instances of PKP. In fact and due to the call for post-quantum standards of the NIST, there has been renewed interest in the transformed Zero-Knowledge Identification Schemes into Digital Signatures Schemes (DSS) via the Fiat-Shamir paradigm [3]. This transformation method is important since it yields to efficient signature schemes in terms of minimal and sufficient security assumptions. Hence, a post-quantum signature scheme based on PKP (PKP-DSS) was proposed in [12]. This scheme takes part of the chinese competition for the standarization of new post-quantum schemes. (PKP-DSS) [12] is therefore a potential candidate for the design of new cryptographic standards.
Therefore, it is important to reconsider some NP-hard problems (for example PKP) whose security relies on the fact that there is no quantum algorithms known to solve such problems [4].
Previous works and Main results. Since quantum computers are expected to be incapable to solve *NP*-hard problems [4], algebraic problems such PKP, are very interesting nowadays.
The main contribution of this paper is to present an updated complexity analysis of the most efficient algorithm for solving instances of the Permuted Kernel Problem.
In [7], J. Georgiades proposed the first algorithm to solve PKP. The author presents symmetric polynomials equations which will be utilized by all the other attacks.
The authors of [8] investigate also the security of PKP, where a time-memory trade-off was introduced. Moreover, J. Patarin and P. Chauvaud improve algorithms for the Permuted Kernel Problem[9]. Also, in [10], a new time-memory trade-off was proposed and believed to be the most efficient attack against PKP.
All the suggested attacks combine exhaustive search with some form of time-memory tradeoff.
In this paper we perform a complexity analysis of all the existing attacks for solving the Permuted Kernel Problem. Interestingly, it appears that the complexity bound given in Joux-Jaulmes paper [10] is not quite precise. Therefore, we show that Joux-Jaulmes attack is not the best algorithm for solving PKP.
In order to estimate a concrete security of PKP, we review and compare the best known attack's effeciency, in terms of the number of operations performed, for different finite fields.
After all, we have been able to bring together Patarin-Chauvaud attack [9] and Poupard algorithm [5] to provide an accurate program. This latter yields better security estimates that contribute for secure parameters sets of the Permuted Kernel Problem.
After updating the complexity bounds of PKP's solving tools, we use a Magma implementation for our software to compare all the attacks in order to define several sets of parameters for different security levels.
Our results have been used in the post-quantum signature scheme based on PKP: PKP-DSS [12]. Moreover, these results may be very useful for the improved scheme of PKP-DSS proposed in [13].

2 The Permuted Kernel Problem

In this section, we first present the PKP problem [1] over a finite field $\mathbb{F}_p$.

2.1 Description of PKP

PKP [1,2] is the problem on which the security of the Identification scheme proposed by A. SHAMIR is based. PKP is a linear algebra problem which asks to find a kernel vector of given matrix under a vector-entries constraint. It's a generalization of the Partition problem [2, pg.224]. More precisely, it is defined as follows:

Input. A finite field $\mathbb{F}_p$, a matrix $A \in \mathcal{M}_{m\times n}(\mathbb{F}_p)$ and a n-vector $V \in {\mathbb{F}_p}^n$.
Question. Find a permutation π over $(1,\ldots,n)$ *such that* $A \times V_\pi = 0$, *where* $V_\pi = (V_{\pi(1)},\ldots,V_{\pi(n)})$.

2.2 Practical complexity considerations

First, for the problem to be hard, $n-m$ must be large enough so that the kernel of A has sufficiently enough elements. Then, since the problem is unchanged by manipulation on lines of A, one may assume that the matrix A is of rank exactly m. Otherwise an equivalent matrix of A could be expressed with fewer significant lines.
By denoting $A_\sigma = (a_{i\sigma(j)})$, the effect of a permutation σ over the columns of A, it's easy to see that $A_\sigma V_\sigma = AV$. Also, up to a reordering of the columns of A, we may assume that it is given in a systematic form:

$$A = (a_{ij})_{1\le i\le m, 1\le j\le n} \;=\; \big[A'|I\big],$$

where $A' = (a'_{ij})_{1\le i\le m, 1\le j\le n-m} \in \mathcal{M}_{m\times n-m}(\mathbb{F}_p)$ and I is the identity matrix of size m.
Note that, to reach higher security levels, it's more desirable that the n-vector V has distinct coordinates, since it reduce the number of solutions, or otherwise said, it enlarges the search space. If the vector V is drawn at random in the kernel of A and is expected to have this property, then one should draw at random the first distinct $n-m$ variables, computes the m last ones and checks if suitable. So one should have $\frac{(p-n+m)!}{p^m(p-n)!}$ high.
Also, it is most suitable that $p^m \approx n!$ so that the number of solutions is close to 1.
A reduction of the 3-Partition problem proves PKP to be NP-hard [2] in the good reasoning (*i.e.*its hardness grows exponentially with p). The solidity of PKP comes from, on one hand, the big number of permutations, on the other hand, from the small number of possible permutations which may suit the kernel equations. More precisely, PKP is hard because it obligates the choice of a vector, with already fixed set of entries, from the kernel of the matrix A.

3 Solving PKP: best known algorithms

The implementation's efficiency of the first IDS, proposed by A. SHAMIR [1], based on PKP problem has led to several solving tools, which are all exponential.

As a reference, we mention the exhaustive search consisting in examining all the possible permutations. Its complexity is obviously $n!$.
We detail here the previously well-known attacks for the so-called PKP problem over a finite field. As well, we discuss the complexity analysis of solving this problem.

3.1 J. GEORGIADES method

J. GEORGIADES [7] was the first to publish an improvement of the exhaustive search. The basic idea is to find some new equations in order to reduce the set of suitable permutations. Since the rank of A is equal to m, then $dim\big(ker(A)\big) = n-m$. There exists in the kernel of the considered matrix $n-m$ vectors which are linearly independent. So, we can fix the first $n-m$ coordinates of each vector and the last m coordinates are constants depending on A. Consequently, it is possible to give the kernel of A the following form:

$$Ker(A) = \lambda_1 \begin{pmatrix} u_{1,1} \\ u_{1,2} \\ \vdots \\ u_{1,m} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \lambda_2 \begin{pmatrix} u_{2,1} \\ u_{2,2} \\ \vdots \\ u_{2,m} \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} + \cdots + \lambda_{n-m} \begin{pmatrix} u_{n-m,1} \\ u_{n-m,2} \\ \vdots \\ u_{n-m,m} \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}, \tag{1}$$

where $u_{1,1}, \ldots, u_{n-m,m}, \ldots$ belong to $\mathbb{F}_p$, and so does the $\lambda_i s$.
Thus, we can conclude that the kernel is the set of vectors:

$$\Big(f_1, f_2, \ldots, f_m, \lambda_1, \lambda_2, \ldots, \lambda_{n-m}\Big), \tag{2}$$

where $\quad f_j = \sum_{i=1}^{n-m} u_{i,j}\lambda_i, \quad j \in \{1, \ldots, m\}$.
Thus, to find the find the secret permutation π, it suffices to exactly place $n-m$ coordinates of the corresponding vector V_π, and then, the other m coordinates will be deduced from the kernel equations 1 and 2. It is equivalent to pick $(n-m)$ values out of n and correctly place them. This will decrease the number of permutations to be considered from $n!$ to:

$$\frac{n!}{\big(n-(n-m)\big)!} = \frac{n!}{m!}.$$

Moreover, since the coordinates of V_π are known, all their symmetrical expressions are also known, for example their sum, the sum of their squares, etc... Then, this cost may be diminished if we successfully use relations between the $\lambda_i s$. As a matter of fact, $V = (v_1, \ldots, v_n)$ is known, and its permutation $V_\pi = (x_1, \ldots, x_n) \in Ker(A)$ has the form

given in 2. Hence, it's feasible to get the following relations in $\mathbb{F}_p$:

$$\sum_{i=1}^{n} v_i^r \mod p \quad = \quad \sum_{i=1}^{n} x_i^r \mod p \quad = \quad \sum_{i=1}^{m} f_i^r + \sum_{i=1}^{n-m} \lambda_i^r \mod p, \qquad (G_r)$$

where r is a positive integer. Such equations G_r are very useful and, for small values of r (for example $r = 1,2$) , are simple to calculate. For $r = 1$ (*resp.* $r = 2$), it is easy to represent some λ_i (*resp.* $\lambda_{j\neq i}$) as a linear combination (*resp.* quadratic equation) of the other $n-m-1$ (*resp.* $n-m-2$) parameters. This will reduce, by taking $r = 2$, the possible permutations to:

$$\frac{n!}{\big(n-(n-m-2)\big)!} = \frac{n!}{(m+2)!}.$$

So far, it is not obvious how to use symmetric equations with higher degree, and decrease more the complexity.

3.2 A time-memory trade-off

Another attack on PKP uses the time-memory trade-off. It was introduced by T. BARITAUD, M. CAMPANA, P. CHAUVAUD and H. GILBERT in [8]. The proposed scheme reduces the time and space required to solve the PKP problem.
Recall that, solving PKP is equivalent to find a permutation π of a vector V such that $A \times V_\pi = 0$. Thus, using the reduced form of A, we can represent PKP as:

$$\begin{pmatrix} a'_{1,1} & \dots & a'_{1,n-m} & 1 & & \\ \vdots & & \vdots & & \ddots & \\ a'_{m,1} & \dots & a'_{m,n-m} & & & 1 \end{pmatrix} \begin{pmatrix} V_{\pi(1)} \\ \vdots \\ V_{\pi(n)} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

Consequently, solving PKP is equivalent to solve a system $\mathcal{S}$ of m equations in n variables given by the entries of the matrix product given above. The algorithm is accomplished by considering k equations of $\mathcal{S}$, where $0 \leq k \leq m$ is a parameter of the algorithm. Due to the block form of A, one can easily see that only $n-m+k$ variables, namely $V_{\pi(1)}, \dots, V_{\pi(n-m+k)}$, are involved in the sub-system of $\mathcal{S}$ formed by k relations. Another parameter $0 \leq k' \leq n-m+k$ is used to indicate the amount of storage to be computed in the first step of the algorithm.
This method is composed by two essential steps:
Step1: precomputation. Recall that the aim is to solve k relations of $\mathcal{S}$. Hence, for each $k'-uple$ $(V_{\pi(1)}, \dots, V_{\pi(k')})$, the corresponding values are computed as follows:

$$b_1 = \sum_{j=1}^{k'} a'_{1,j} V_{\pi(j)}$$

$$\vdots$$

$$b_k = \sum_{j=1}^{k'} a'_{k,j} V_{\pi(j)}$$

Then, the $\frac{n!}{(n-k')!}$ possible values of the $k'-uples$ and its corresponding results $(b_1, \cdots, b_k)$ are stored. Note that, for each of the p^k possible value of the vector $(b_1, \cdots, b_k)$ the $k'-uple$ $(V_{\pi(1)}, \ldots, V_{\pi(k')})$ are quickly accessed.
This step costs $\frac{n!}{(n-k')!}$ matrix-vector product. The memory required is about $\frac{n!}{(n-k')!}$ $k'-uples$. Also, for each $(b_1, \cdots, b_k)$ value corresponds *approx.* $p^{-k}\frac{n!}{(n-k')!}$ $k'-uples$.
Step2: exhaustive trial. The exhaustive search is performed over the remaining components $(V_{\pi(k'+1)}, \ldots, V_{\pi(n-m+k)})$. There is $\frac{n!}{(m+k'-k)!}$ possible value of such vector.
For each tested vector, the corresponding values are computed from the k equations:

$$c_1 = \sum_{j=k'+1}^{n-m+k} a'_{1,j} V_{\pi(j)}$$

$$\vdots$$

$$c_k = \sum_{j=k+1}^{n-m+k} a'_{k,j} V_{\pi(j)}$$

Now, using the precomputation step, a list of probable $(V_{\pi(1)}, \ldots, V_{\pi(k')})$ is obtained. Obviously, the k relations can be represented as:

$$b_1 + c_1 = 0$$

$$\vdots$$

$$b_k + c_k = 0.$$

Moreover, the $k'-uple$ $(V_{\pi(1)}, \ldots, V_{\pi(k')})$ is certainly one of the possible $k'-uples$ for the $(-c_1, \ldots, -c_k)$ value of $(b_1, \ldots, b_k)$.
For every vector $(V_{\pi(k'+1)}, \ldots, V_{\pi(n-m+k)})$ generated, there are in average $p^{-k}\frac{n!}{(n-k')!}$ $(V_{\pi(1)}, \ldots, V_{\pi(k')})$ values. For each probable solution $(V_{\pi(1)}, \ldots, V_{\pi(n-m+k)})$, the remaining unsolved equations from $(k'+1)$ to (m) give successively only one possible value for the last components $V_{\pi(n-m+k+1)}, \ldots, V_{\pi(n)}$.
The required space of this step is negligible. In contrast, the required time is about $\sup(\frac{n!}{(m+k'-k)!}\frac{n!}{(n-k')!}p^{-k}, \frac{n!}{(m+k'-k)})$ matrix vector product.
Thus, for every pair (k, k'), the total time complexity of solving PKP, using this time-memory attack is about:

$$\frac{n!}{(n-k')!} + \sup\left(\frac{n!}{(m+k'-k)!}\frac{n!}{(n-k')!}p^{-k}, \frac{n!}{(m+k'-k)}\right),$$

and the total space required is about:

$$\frac{n!}{(n-k')!} \text{ k'-vectors.}$$

3.3 Joux-Jaulmes algorithm

In [10], A. Joux and E. Jaulmes introduce a new time-memory trade-off algorithm which is an application of the algorithm described in [11] to the Permuted Kernel Problem. This technique includes the so-called 4SET problem (see [14,10] for more details) which is defined as follows:

Input. An n-vector $P = (p_1, \ldots, p_n)$ where the p_is are primes, four sets S_i of $n-$vectors such that $|S_i| = N_i$ for $i = 1 \ldots 4$, and n sets $D_1, \ldots, D_n$.
Question. Find $v^{(1)} \in S_1, \ldots, v^{(4)} \in S_4$, $d_1 \in D_1, \ldots, d_n \in D_n$ such that:

$$\forall i \in [1, \ldots, n], \quad v_i^{(1)} + v_i^{(2)} + v_i^{(3)} + v_i^{(4)} \equiv d_i \pmod{p_i}$$

The solving strategy of 4SET is composed of two phases: the A-Phase which is a pre-computation step, and the B-Phase which is a main loop consisting two enumeration steps (detailed in [10]). The authors of [10] specify reasonable choice of parameters for the solving technique of 4SET. Thus, the its time complexity is given by:

$$\mathcal{O}\Big((n-k)\psi N_1 N_2 N_3 N_4\Big),$$

Where $\psi = \prod_{i=1}^{k} \frac{|D_i|}{p_i}$ for suitable choices of $1 \leq k \leq n$. As shown in [10], we can reduce an instance of PKP to the 4SET problem. According to [7], it is useful to add the G_r equations. The linear equation G_r, for $r = 1$, represents the fact that the sum σ of the coordinates of the vector V is independent of the secret permutation π. By considering this linear equation the kernel vector V_π verify:

$$(A'_0 | I_{m+1}) V_\pi = \begin{pmatrix} \sigma \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

as said in 2.1, $A = \big[A'|I\big]$. Thus, $A'_0 \in \mathcal{M}_{(m+1)\times(n-m-1)}(\mathbb{F}_p)$, and I_{m+1} is the identity matrix of order $(m+1)$.
Now, A'_0 is divided into four roughly equal parts, so:

$$(A'_1 A'_2 A'_3 A'_4 | I_{m+1}) V_\pi = \begin{pmatrix} \sigma \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{3}$$

where A'_i is a $(m+1) \times (n_i)$ matrix and $n_1 + n_2 + n_3 + n_4 = n - m - 1$.
Recall that V is known and V_π is its permutation vector. In order to apply the 4SET problem, we need to construct the sets S_i. Since A'_i is an $(m+1) \times (n_i)$ matrix, S_i is the set of $m+1$-vectors: the product resulting of A'_i by all the possible n_i combinations of

the coordinates of V. So, the size of S_i is equal to $N_i = \frac{n!}{(n-n_i)!}$.
In this case, all the primes p_is are equal to the prime number p given by the PKP instance. Now, to determine the $m+1$ sets D_i, lets decompose, similarly to the matrix A'_0, the vector $V_\pi = (V_1^{(\pi)} V_2^{(\pi)} V_3^{(\pi)} V_4^{(\pi)} V_5^{(\pi)})$ such that for $i \in [1,\dots,4]$, $V_i^{(\pi)}$ is an n_i-vector where, as we quoted before, $n_1+n_2+n_3+n_4 = n-m-1$. So, $V_5^{(\pi)}$ is the vector formed by the last $(m+1)$ coordinates of V_π. Hence, we can reformulate the matrix-vector product (3) as follow:

$$A'_1 V_1^{(\pi)} + A'_2 V_2^{(\pi)} + A'_3 V_3^{(\pi)} + A'_4 V_4^{(\pi)} = \begin{pmatrix} \sigma \\ 0 \\ \vdots \\ 0 \end{pmatrix} - V_5^{(\pi)} = \begin{pmatrix} \sigma - v_{n-m}^{(\pi)} \\ -v_{n-m+1}^{(\pi)} \\ \vdots \\ -v_n^{(\pi)} \end{pmatrix} = \mathcal{D}$$

It's obvious that the value of $V_5^{(\pi)}$ depends on the V_i^{π}s , for $i \in [1,\dots,4]$, so does the $m+1$-vector $\mathcal{D}$. The first component of $\mathcal{D}$ depends on $v_{n-m}^{(\pi)}$ which has n possible values. Thus, D_1 is the set of these n possible values. Since V has no double, the set D_2 is formed by $n-1$ elements, and so on. In this way, the sets $D_1,\dots,D_{m+1}$ are built such that each one has in average:

$$\frac{n+(n-1)+,\ \dots\ ,+(n-m)}{m+1} \quad elements.$$

Note that we are dealing with bit operations. So, in order to define the solving time complexity, it is indispensable to pack 32 or even $64 = 2^6$ bit operations in one word operation. It's equivalent to divide the complexity by 2^6. In summary, the authors of [10] gives the following time complexity for their algorithm (see [10] for more details):

$$\mathcal{O}\Big((m+1-k) \times \psi \times \frac{n!^2}{(n-n_1-n_2)!(n-n_3-n_4)!} \times 2^{-6}\Big),$$

Where $k = \log_{\frac{|D_i|}{p}}(\psi)$.
It appears in [10] that this new approach is the most efficient to solve PKP.
But, we show in the next section that the performance of Joux-Jaulmes attack on PKP had been misjudged. Joux-Jaulmes attack is much more complicated than expected.
Thus, in the next section, we detail the choice of the most efficient solving tool for PKP.

4 Concrete security of the Permuted Kernel Problem

In this section, we present the main contributions of this paper. The aim here is double: correcting the complexity bound of Joux-Jaulmes algorithm, and providing the best method for solving PKP.

4.1 Complexity Analysis of Joux-Jaulmes algorithm

Note that, in this section we use the notations of Section 3.3.
Recall that we can reduce an instance of PKP to the 4SET problem, so we can apply its

solving strategy to PKP. Then as well, the algorithm of JOUX-JAULMES consists of a main loop containing two enumerations phases: A-Phase and B-Phase. The authors of [10] assume that the B-Phase controls the time complexity of this approach. Without going too far into the analysis of this technique, we found that the overall complexity is wrongly estimated. By considering a reasonable choices of parameters, it turns out that the time complexity of the algorithm is dominated most cases by the A-Phase. Recall from [10],the analysis of JOUX-JAULMES algortihm:
The number of operations needed to execute the A-Phase is in:

$$\mathcal{C}_{A-Phase} \;=\; \mathcal{O}\Big(\; \max\big\{N_1 \log(N_2)\psi\phi,\; (m+1-k)\psi\frac{n!}{(n-n_1-n_2)!}\big\}\;\Big).$$

While the B-phase requires:

$$\mathcal{C}_{B-Phase} \;=\; \mathcal{O}\Big(\; \max\big\{N_3 \log(N_4), (m+1-k)\psi\frac{n!^2}{(n-n_1-n_2)!(n-n_3-n_4)!}\phi^{-1}\big\}\;\Big),$$

Where, $N_i \;=\; \frac{n!}{(n-n_i)!}$. Hence, by considering the effect of the main loop which contains the 2 Phases, the total time complexity can be expressed as:

$$2^{-6} \times \phi \times \big(\mathcal{C}_{A-Phase} + \mathcal{C}_{B-Phase}\big),$$

Where $\phi = p^k$ is the cost of the main loop containing the two phases A and B, and 2^{-6} is for packing bit operations.
The following table confirms what we claimed earlier, that the overall complexity is dominated by the A-phase.
Note that, we use here the same parameters sets, of the form $\big(\mathrm{PKP}_p(m,n)\big)$, given in [10].

Parameters Sets	A-phase Complexity	B-phase Complexity	Overall Complexity
$\big(\mathrm{PKP}_{251}(16,32)\big)$	$2^{45.72}$	$2^{18.02}$	$2^{93.55}$
$\big(\mathrm{PKP}_{251}(15,32)\big)$	$2^{46.13}$	$2^{18.02}$	$2^{93.96}$
$\big(\mathrm{PKP}_{251}(24,48)\big)$	$2^{94.45}$	$2^{32.09}$	$2^{190.1}$
$\big(\mathrm{PKP}_{251}(34,64)\big)$	$2^{135.67}$	$2^{40.67}$	$2^{270.19}$

Table 1. The A/B-Phase complexity of JOUX-JAULMES algorithm

Experimental results show that for the set of parameter that minimize the total time complexity, the phase A dominates, and the total is much higher as announced. Therefore, the JOUX-JAULMES algorithm is not efficient for solving PKP.

4.2 Simplest and most efficient algorithm

Here, we try to estimate a lower complexity bound of the algorithms that serve for solving PKP. As a simplification, we assume that the elementary operation is the computation of a tuple, namely a vector-matrix product in $\mathbb{F}_p$ or whatever should be more

efficient to accomplish it, as for instance a result can be deduced from another one, where few changes have to be made. For the same reason, we assume also that the memory unit is the space needed to store a tuple.

Improvement and Generalisation of already existing attacks. As already recalled in Section 3.2, the time-memory trade-off method [8] reduces the time required to solve the PKP problem at the cost of use of significantly large memory. The idea for speeding up the solving time is to perform a pre-computation on a smaller search space involving an A-sub-matrix and the corresponding subsystem.
Then naturally in [9], J. Patarin and P. Chauvaud combine this method with the idea of J. Georgiades. Hence, adding the "free" linear symmetric equation leads to a reduction in the time required to attack PKP. They present also some new ideas in order to reduce the memory usage. Due to the different variants presented, we will ony cite the following (See [9] for further details):

- "Set introduction" : make an exhaustive search on a sub-set of values, instead of ordered tuples of values. This leads to diminish the size of the initial pre-computation.
- "Middle values" : make an exhaustive search on the value of a sub-system,
- "Pre-computation on A": search if a sub-system could be expressed with less variables. This leads to probabilistic algorithms.

G. Poupard, in [5] gives a nice generalization of the "Middle values" method, and a corresponding complexity analysis, but it seems to be flawed since the details of the complexity are not clearly given.
Thus, in the next section, we consider all the existing attacks and their improvements in order to obtain a fresher look at the algorithm for effectively solving PKP.

Our method : Extension of the most efficient attacks. In this section, we bring together most of the previously known methods to solve PKP. We provide a new software which leads to an efficient complexity measurement, and we establish its approximate space and time complexity.
We will use the following notation: k-V is a tuple of k values, where k notes conveniently at the same time a subset of indexes and the number of elements of this subset. In the same spirit, $k.i$-A is a sub-matrix of A with the given subsets of indexes.
Mainly, we build upon the work of Patarin-Chauvaud [9] and Poupard [5], combining their techniques, and pushing further the implementation to hold higher security levels. More specifically, four variations are essentials:

1. Use the symmetric polynomial equations G_r of small degrees.
2. Reduce the time complexity: Perform a precomputation step and an exhaustive search on a sub-set of variables, instead of ordered tuples of variables [8,9].
3. Reduce the memory: Introduce some middle values leading to solve a simple system of equations [9].
4. Carry out a pre-computation on the matrix A which leads to probabilstic algorithms (See Section 4.2).

Our starting point will be to combine, in Alg. 1, the first two ideas given above (Section 4.2) for solving PKP. This method exploits usefully the special form of the matrix $A = (A'\|I)$.

Algorithm 1 A1 : Solve PKP (n,m,p)

Require: $0 \leq k \leq m$ and $l+r+m-k=n$
1: select k equations, and split their variables into two sets l and r
2: **for all** l-tuple l-V **do**
3: compute $C \leftarrow k.l\text{-}A \times l\text{-}V$
4: store efficiently $(C, l\text{-}V)$ in a file $F0$ so that given a value C, one can retrieve efficiently all the associated l-tuples
5: **end for**
6: **for all** r-tuple r-V **do**
7: compute $C \leftarrow -k.r\text{-}A \times r\text{-}V$
8: retrieve from $F0$ a list L of l-tuples associated with C
9: filter the list L keeping only the l-tuples compatible with r-V
10: **for all** l-tuple l-V in L **do**
11: compute the last variables s-$V \leftarrow (m-k).(l+r)\text{-}A \times (l+r)\text{-}V$
12: **if** the values s are compatible with $(l+r)$ **then**
13: $(l+r+s)$ is a solution
14: **end if**
15: **end for**
16: **end for**

For this algorithm, we can summarize the time and space complexities by these formulas :

$$\text{space} = \frac{n!}{(n-l)!} \tag{4}$$

$$\text{time} = \frac{n!}{(n-l)!} + \frac{n!}{(n-r)!} + \frac{n!}{p^k(n-(l+r))!} \tag{5}$$

In order to provide a fisrt clear image on the parameter sets, we have collected, using Alg. 1, several tests on prime fields for different values of n. So, in Fig. 1, we show the time complexity for solving PKP by Alg. 1.
Note that, the number m of equations is implicitly estimated as the closest integer of $\log(n!)/\log(p)$.

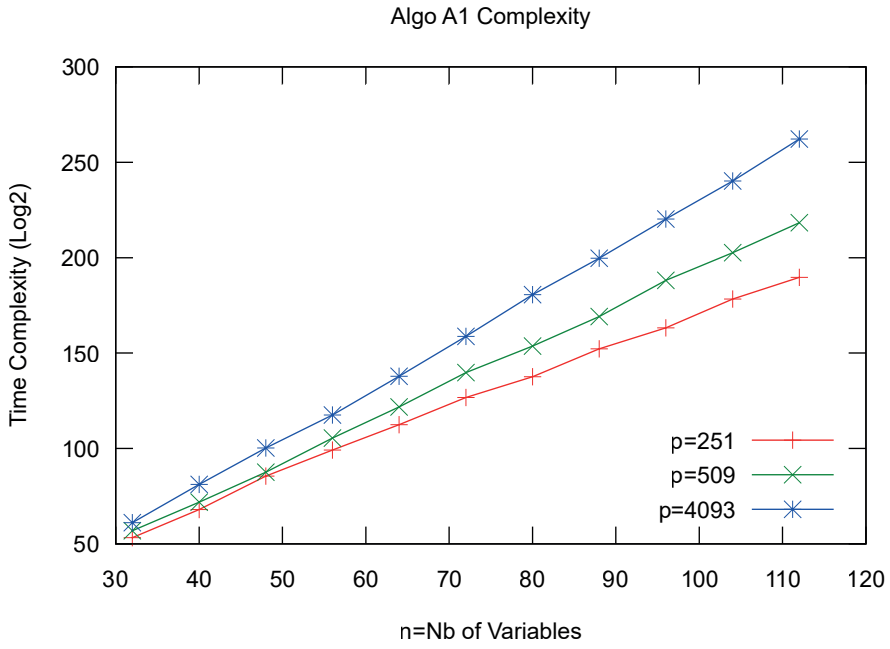

Fig. 1. Time complexity of Alg. 1 for various values of p

Our second point is to combine almost all of the variations (Section 4.2). Thus, in Alg. 2 we give a more synthetic description of Poupard's algorithm mixed with the techniques of precomputed files and "Middle values". Moreover, we provide its detailed complexity which was not estimated before (cf. equations 6 and 7 below).
Notations are the ones in [5, Fig. 3].

Algorithm 2 A2 : Solve PKP (n,m,p)

Require: $i+j+r+m=n$ and $c+c'\leq m$ and $d\leq c$ and $l+k=r+d$

for all j-tuple j-V **do**
  compute $C0 \leftarrow c.j\text{-}A \times j\text{-}V$
  store efficiently $(C0, j\text{-}V)$ in a file $F0$ so that given a value $C0$, one can retrieve efficiently all the associated j-tuples
end for
for all l-tuple l-V **do**
  compute $C2 \leftarrow d.l\text{-}A \times l\text{-}V$
  store efficiently $(C2, l\text{-}V)$ in a file $F2$ so that given a value $C2$, one can retrieve efficiently all the associated l-tuples
end for
for all c-tuple of "Middle values" c-C **do**
  for all i-tuple i-V **do**
    compute $C0 \leftarrow c\text{-}C - c.i\text{-}A \times i\text{-}V$
    retrieve from $F0$ a list L of j-tuples associated with $C0$
    filter the list $L0$ keeping only the j-tuples compatible with i-V and get $(j+i)$-tuples
    for all $(j+i)$-tuple $j+i$-V of $L0$ **do**
      compute $C1 \leftarrow c'.(j+i)\text{-}A \times (j+i)\text{-}V$
      store efficiently $(C1, (j+i)\text{-}V)$ in a file $F1$ so that given a value $C1$, one can retrieve efficiently all the associated $(j+i)$-tuples
    end for
    for all k-tuple k-V **do**
      compute $C2 \leftarrow d\text{-}C - d.k\text{-}A \times k\text{-}V$
      retrieve from $F2$ a list $L2$ of l-tuples associated with $C2$
      filter the list $L2$ keeping only the l-tuples compatible with k-$V$and get $(l+k)$-tuples
      for all $(l+k)$-tuple $(l+k)$-V of $L2$ **do**
        compute $(c-d)\text{-}V \leftarrow (c-d)\text{-}C - (c-d).(l+k)\text{-}A \times (l+k)\text{-}V$ and keep only values compatible with $(l+k)$-V and get $(r+c)$-tuples
        for all c'-tuple c'-V compatible with $(r+c)$-V **do**
          compute $C1 \leftarrow -(c').(r+c)\text{-}A \times (r+c)\text{-}V$
          retrieve from $F1$ a list $L1$ of $j+i$-tuples associated with $C1$
          filter the list $L1$ keeping only the $(j+i)$-tuples compatible with $r+c+c'$-V and get $(j+i+r+c+c')$-tuples
          for all $(j+i+r+c+c')$-tuple $(j+i+r+c+c')$-V in $L1$ **do**
            compute the last variables s-$V \leftarrow (m-c-c').(j+i+r+c+c')\text{-}A \times (j+i+r+c+c')\text{-}V$
            if the values s are compatible with $(j+i+r+c+c')$ **then**
              $(j+i+r+c+c'+s)$ is a solution
            end if
          end for
        end for
      end for
    end for
  end for
end for

For this algorithm, we can approximately estimate the time and space complexities as follows:

$$\text{space} = \frac{n!}{(n-j)!} + \frac{n!}{(n-l)!} + \frac{n!}{p^c(n-(i+j))!} \tag{6}$$

$$\begin{aligned}\text{time} = {} & \frac{n!}{(n-j)!} + \frac{n!}{(n-l)!} + \frac{n!}{(n-(i+j))!} + \\ & \frac{p^c n!}{(n-k)!} + \frac{p^{c-d} n!}{(n-(k+l))!} + \frac{p^{c-d}(n-(c-d))!}{(n-(r+c+c'))!} \\ & + \frac{(n-(c-d))!}{p^d(n-(i+j+r+c+c'))!}\end{aligned} \tag{7}$$

As said in [5], the interest of this algorithm is for realistic attacks, where memory is limited. However from a theoretical point of view, the previous algorithm is the most efficient.

Probabilistic method We here discuss the method named "Pre-computation on the A matrix" of [9]. Like previous methods, this new method aims at decreasing the complexity. Its specificity is to search for subsets of equations with fewer variables than expected. We first give an estimation of the probability that among a given set of m random equations in n variables over $\mathbb{F}_p$, there exists a subset of k equations in only r variables. Such a probability was given in [9], but here we explain how to compute it more accurately. Lets start with the following results.

Claim. The probability that a random linear equation in n variables over $\mathbb{F}_p$ is indeed in r variables is $\binom{n}{r}(\frac{1}{p})^{n-r}(\frac{p-1}{p})^r$.

Proof. Let a random linear equation in n variables over $\mathbb{F}_p$. Each of its coefficients is assumed to be randomly and uniformly distributed if $\mathbb{F}_p$. Therefore the probability that a given variable does not occur in the equation is the probability that its coefficient is 0, or so $\frac{1}{p}$. All the coefficients are assumed to be independently drawn at random. Therefore, the number of variables that appears in the equation follows the binomial distribution with parameters n and $\frac{1}{p}$; hence the result.

Claim. The probability that a set of k random linear equations in n variables over $\mathbb{F}_p$ is indeed in r variables is $\binom{n}{r}(\frac{1}{p^k})^{n-r}(1-\frac{1}{p^k})^r$.

Proof. The probability that a given variable does not occur in the k equations $\frac{1}{p^k}$. Therefore, the number of variables that appears in the k equations follows the binomial distribution with parameters n and $\frac{1}{p^k}$; hence the result.

Claim. In a linear space of m linear equations over $\mathbb{F}_p$, the number of distinct linear subspaces of k equations is

$$\prod_{i=0}^{k-1} \frac{p^m - p^i}{p^k - p^i}$$

Proof. The number of k-tuples of linearly independent equations is given by :

$$\prod_{i=0}^{k-1} p^m - p^i.$$

All the tuples that are equivalent bases of the same subspace are related by a given isomorphism over $\mathbb{F}_p^k$. These isomorphisms amount to $\prod_{i=0}^{k-1} p^k - p^i$. Hence the result.

Claim. In a linear space of m linear equations over $\mathbb{F}_p$, the number of distinct linear subspaces of k equations that can be expressed in r variables is approx.

$$\binom{n}{r}(\frac{1}{p^k})^{n-r}(1-\frac{1}{p^k})^r \prod_{i=0}^{k-1} \frac{p^m - p^i}{p^k - p^i}.$$

Proof. Although the subspaces of dimension k in the linear space are not uniformly distributed over all possibilities, a good approximation is the product of the number of subspaces by the probability that one subspace has the required property. This result has been experimentally verified.

Although the method of using subspaces of equations with less variables speeds indeed the searching algorithm, it appears that the probability of finding such subsets is overwhelming small. Example: For PKP(64,37,251), the optimum value for k is 19. Therefore, finding k equations expressed in $r = n - m + k = 45$ is easy. The expected number of subspaces expressed in only 44 variables is $\approx 2^{-98.5}$.

4.3 Secure Parameters Sets And PKP application.

As said before, PKP was used in 1989 to build a Zero-Knowledge Identification scheme (IDS) [1]. There has been renewed interest in the PKP-based IDS. In fact, the authors of [12] convert the PKP-based IDS to a post-quatum digital signature scheme. This latter takes part in the chinese competition for the design of post-quantum cryptographic algorithms (organized by the Chinese Association CACR). Thus, it is important to define the complexity bound of each solving tool for the Permuted Kernel Problem.
Since we have now a realistic picture of the efficiency and the complexity bound of nearly all of the known methods for solving PKP, it is possible to compare the performance of each technique. Consequently, we can define by now secure parameters sets for the PKP instances.
The following table shows that the most efficient attack to solve PKP is the extended version of Patarin-Chauvaud [9]and Poupard [5]. Our complexity bound for this version which is established in Section 4.2, is computed using a Magma code given in Appendix A.
Note that, we will recall the same parameters sets, of the form $\big(\mathrm{PKP}_p(m,n)\big)$, used in [12]

Parameters Sets	$(\mathrm{PKP}_{251}(41,69))$	$(\mathrm{PKP}_{509}(54,94))$	$(\mathrm{PKP}_{4093}(47,106))$
Security level	2^{128}	2^{192}	2^{256}
Brute force attack	2^{326}	2^{485}	2^{565}
J. Georgiades attack	2^{151}	2^{236}	2^{356}
Time-memory trade-off	2^{131}	2^{196}	2^{262}
Joux-Jaulmes attack	2^{286}	2^{413}	2^{432}
Patarin-Chauvaud / Poupard	$\mathbf{2^{130}}$	$\mathbf{2^{193}}$	$\mathbf{2^{257}}$

Table 2. Complexity bounds for PKP's best known algorithms

5 Conclusion

In this paper, we have seen the best known methods for solving PKP. We presented briefly each one and updated some results that were not accurate or genuine. More specifically, we have found that the Joux-Jaulmes algorithm [10] is not the most efficient technique to solve the Permuted Kernel Problem.
Combining methods, namely the approach of Patarin-Chauvaud and Poupard [5], we have been able to provide an explicit complexity formula (cf. equations 6 and 7 above) of the best algorithm for solving hard instances of PKP. Also, we have built a program which gives a realistic picture on the security level of PKP instances. This program is very useful to establish secure sets of parameters in order to arise hard instances of PKP.

References

1. Shamir, A. (1989, August). An efficient identification scheme based on permuted kernels. In Conference on the Theory and Application of Cryptology (pp. 606-609). Springer, New York, NY.
2. Gary, M., Johnson, D. (1979). Computers and Intractability: A Guide to NP-Completeness. New York: W H.
3. Fiat, A., Shamir, A. (1986, August). How to prove yourself: Practical solutions to identification and signature problems. In Advances in Cryptology—CRYPTO'86 (pp. 186-194). Springer, Berlin, Heidelberg.
4. Bennett, C. H., Bernstein, E., Brassard, G., Vazirani, U. (1997). Strengths and weaknesses of quantum computing. SIAM journal on Computing, 26(5), 1510-1523.
5. Poupard, G., (1997). A realistic security analysis of identification schemes based on combinatorial problems. European Transactions on Telecommunications.
6. Lyubashevsky, V. (2009, December). Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 598-616). Springer, Berlin, Heidelberg.
7. Georgiades, J. (1992). Some remarks on the security of the identification scheme based on permuted kernels. Journal of Cryptology, 5(2), 133-137.
8. 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). (1992, August), Springer, Berlin, Heidelberg.

9. Patarin, J., Chauvaud, P. Improved algorithms for the permuted kernel problem. In Annual International Cryptology Conference (pp. 391-402). (1993, August) Springer, Berlin, Heidelberg.
10. Jaulmes, É., Joux, A. Cryptanalysis of pkp: a new approach. In International Workshop on Public Key Cryptography (pp. 165-172). (2001, February) Springer, Berlin, Heidelberg.
11. Joux, A., Lercier, R. "Chinese and Match", an alternative to Atkin's "Match and Sort" method used in the SEA algorithm. Mathematics of computation, 70(234), 827-836.
12. Faugère, J-C., Koussa,E., Macario-rat, G., Patarin, J., Perret, L., PKP-Based Signature Scheme. In IACR Cryptology ePrint Archive (714). (2018).
13. Beullens, W., On sigma protocols with helper for MQ and PKP, fishy signature schemes and more. In IACR Cryptology ePrint Archive (490). (2019).
14. Joux, A., Lercier, R. Chinese and Match, an alternative to Atkin's Match and Sort method used in the SEA algorithm. (1999). Preprint.

Appendix A

Magma code for the complexity of PKP

```
//The first algorithm A1
m:=Round(Log(p,Factorial(n)));
m+:=1; //georgA1:=function(n,p);iades
SpaceMax:=8*10^15;
TimeMin:=2^1000;

for k in [1..m] do;
 l:=(n-m+k) div 2;
 r:=n-m+k-r;
 F0:=1.0*Factorial(n) / Factorial(n-l);
 size0:=F0;
  if size0 gt SpaceMax then continue; end if;
 nb1:=1.0*Factorial(n)/Factorial(n-r);
 nb2:=1.0*Factorial(n)/Factorial(n-(l+r))/p^k;
 tps0:= F0;//*(k*l);
 tps1:= nb1;//*(r-k)*k;
 tps2:= nb2;//*(l+r-k)*(m-k);
 tps:=tps0+tps1+tps2;
  if tps lt TimeMin then
    TimeMin:=tps;
    U:=<Log(2,tps),Log(10,size0), Log(2,tps0), Log(2,tps1),
    Log(2,tps2), Log(2,nb1),Log(2,nb2),l, r, k>;
  end if;
end for;

return U;
end function;

// The second algorithm : A2
//Here we use sets instead of tuples
for given n,p;
m:=Round(Log(p,Factorial(n)));
m+:=1; //georgiades
SpaceMax:=8*10^15;
TimeMin:=2^1000;
 for k in [1..m], l in [1..n-m] do;
  r:=n-m-l+k;
  nb0:=Binomial(n,r);
```

```
  F0:=1.0*Factorial(n-r) / Factorial(n-l-r);
  size0:=F0*l;
   if size0 gt SpaceMax then continue; end if;
  nb1:=1.0*Factorial(r);
  nb2:=1.0*nb1*F0/p^k;
  tps0:= F0;//*(k*l);
  tps1:= nb1;//*(r-k)*k;
  tps2:= nb2;//*(l+r-k)*(m-k);
  tps:=nb0*(tps0+tps1+tps2);
   if tps lt TimeMin then
     TimeMin:=tps;
     U:=<Log(2,tps),Log(10,size0), Log(2,tps0), Log(2,tps1),
     Log(2,tps2), Log(2,nb1),Log(2,nb2),l, r, k>;
   end if;
  end for;
 U; exit;

poupard:= function(n,p);    //poupard
m:=Round(Log(p,Factorial(n)));
m+:=1; //georgiades
SpaceMax:=8*10^150;
TimeMin:=2^1000;
 for j in [0..n-m], i in [0..n-m-j],
 l in [0..n-m-j-i], c in [0..m], d in [0..c], cc in [0..m-c] do;
  r:=n-m-i-j;
  k:=r+d-l;
  mm:=c-d;
  nf0:=Factorial(n)/Factorial(n-j);
  size_f0:=nf0*j;
  time_f0:=nf0*(j*c);
  nf2:=Factorial(n)/Factorial(n-l);
  size_f2:=nf2*l;
  time_f2:=nf2*(l*d);
  nf1:=Ceiling(Factorial(n)/Factorial(n-(i+j))/p^c);
  size_f1:=nf1*(i+j);
  time_f1:=Factorial(n)/Factorial(n-i)*(c*i) // -> compute candidate F0
  +nf1*((i+j)*cc);
  size0:=size_f0+size_f1+size_f2;
   if size0 gt SpaceMax then continue; end if;
  //choice of k-tuples
   nb1:=Factorial(n)/Factorial(n-k); // k-tuples_V
   time_1:=nb1*((r-l)*d); //calcul entr e F2
  //solution F2 in (l+k)
   nb2:=1.0*Factorial(n)/Factorial(n-(k+l))/p^d;
   time_2:=1+nb2*(r*mm); //-> donne candidat M
```

```
   //M (compatible KL)
    nb3_:=1.0*nb2*Factorial(n-mm)*Factorial(n-(k+l))
     /Factorial(n)/Factorial(n-(k+l+mm));
   nb3:=1.0* Factorial(n-mm)/Factorial(n-(k+l+mm))/p^d;
  //We complete with N
   nb4:=nb3*Factorial(n-(k+l+mm))/Factorial(n-(k+l+mm+cc));
   time_4:=nb4*(cc); //-> candidat F1
  //We read F1
   nb5:=Ceiling(nb4*nf1/(p^cc)*Factorial(n-(i+j))*Factorial(n-(l+k+mm+cc))
   /Factorial(n)/Factorial(n-(i+j+l+k+mm+cc)));
  time_5:=nb5*((m+1)-(c+cc))*(i+j+r);
  tps:=time_f0+time_f2+ p^c *
  ( time_f1 + time_1 + time_2
   + time_4 + time_5);
  if tps lt TimeMin then
    TimeMin:=tps;
    U:=<Log(2,tps),Log(10,size0),
    j,i,l, r, k>;
   end if;
end for;
return U;
end function;
U;
exit;
```