Improved Algorithms for the Permuted Kernel Problem

Jaques Patarin
Bull CP8
BP 45
68 Route de Versailles
78430 Louveciennes
France

Pascal Chauvaud
CNET-France Telecom
PAA-TSA-SRC
38-40 Rue du Général Leclerc
92131 Issy-les-Moulineaux
France

Abstract

In 1989, Adi Shamir published a new asymmetric identification scheme, based on the intractability of the Permuted Kernel Problem (PKP) [3]. In 1992, an algorithm to solve the PKP problem was suggested by J. Georgiades [2], and also in 1992 T. Baritaud, M. Campana, P. Chauvaud and H. Gilbert [1] have independently found another algorithm for this problem. These algorithms still need huge amount of time and/or memory in order to solve the PKP problem with the values suggested by A. Shamir.

In this paper, we will see that it is possible to solve the PKP problem using less time that which was needed in [1] and [2], and much less memory than that needed in [1].

First we will investigate how the ideas of [1] and [2] can be combined. This will enable us to obtain a little reduction in the time needed. Then, some new ideas will enable us to obtain a considerable reduction in the memory required, and another small reduction in time.

Since our new algorithms are quicker and more practical than previous algorithms they confirm the idea stated in [1] that for strong security requirements, the smallest values ($n = 32$, $m = 16$, $p = 251$) mentioned in [3] are not recommended.

1 Recall of the algorithms of [1].

In this section, we will briefly recall the attack given in [1] for the PKP Problem (see [1] for more details). Then in the next sections, we will study how to improve these algorithms. The PKP problem is the following :

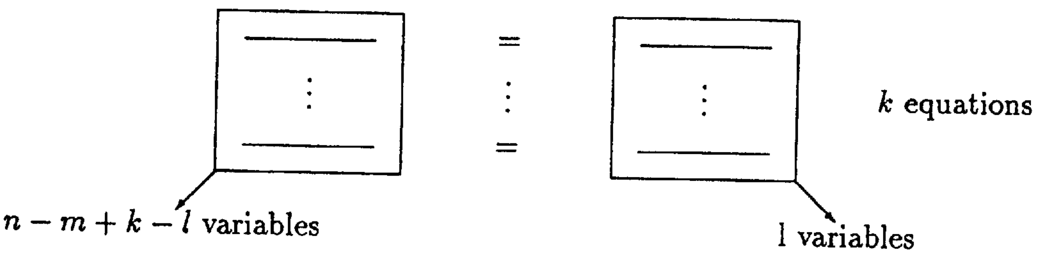
<u>Given</u> : a prime number p,
a $m \times n$ matrix $A = (a_{ij})$, $i = 1 \dots m$, $j = 1 \dots n$, over $\mathrm{Z_p}$,
a n-vector $V = (V_j)$, $j = 1 \dots n$, over $\mathrm{Z_p}$,
<u>Find</u> : a permutation π over $(1, \dots, n)$ such that $A * V_\pi = 0$, where $V_\pi = (V_{\pi(j)})$, $j = 1, \dots, n$.

We will assume that A is of rank m and is generated under the form $[\ A'|I\ ]$ where A' is a fixed $m*(m-n)$ matrix and I the $m*m$ identity matrix. As mentioned in [3] this is not restrictive because both the prover and the verifier can apply Gaussian elimination. We will use, as far as possible, the notations of [1], and we will denote by $(x_1,\dots,x_n)$ the components of the V_π vector. So

$$A * V_\pi = 0 \Longleftrightarrow \left\{ \begin{array}{lcll} \sum\limits_{i=1}^{n-m} a'_{1i}x_i + x_{n-m+1} & = & 0 & (1) \\ \vdots & & & \vdots \\ \sum\limits_{i=1}^{n-m} a'_{mi}x_i + x_n & = & 0 & (m) \end{array} \right.$$

In the algorithms described in [1], one first tries to solve the equations (1) to (k), where k is a parameter of the algorithms. In these k equations, there are $n-m+k$ distinct variables x_i. These variables are divided into two groups : one group of l variables (l is another parameter of the algorithm) will be written on the right-hand side of the equations, and the other $n-m+k-l$ variables will be on the left-hand side of the equations. (We will denote by $(x_1,\dots,x_l)$ the l variables on the right-hand side).

So the equations (1) to (k) will be represented in a scheme like this :



Then the algorithm of [1] proceeds in the following way :

Step 1 : Precomputation. For each of the $\frac{n!}{(n-l)!}$ possible values for the l variables on the right-hand side, the right-hand side value (of k equations) is calculated. These values are stored in a file $F1$ in such a way that for each of the p^k possible values for the right-hand side, the list of the corresponding $(x_1,\dots,x_l)$ can be accessed by very few elementary operations.

Step 2 : Generation and test of candidates. For each of the $\frac{n!}{(n-k+l)!}$ possible values for the $(n-k+m-l)$ variables on the left-hand side, the left-hand side value (of k equations) is calculated. Then the file $F1$ of step 1 is used to obtain a list of possible $(x_1,\dots,x_l)$ such that the value on the left-hand side is the same as the value on the right-hand side. Then these candidates are tested, by using also the equations $(k+1)$ to (n). (One also has to check that the variables used for the right-hand side are not used on the left-hand side).

Example 1 : Algorithm $A0$.

For a PKP (16,32), that is to say with $n = 32$, $m = 16$, $p = 251$, let us choose (as suggested in [1]) $k = 6$ and $l = 10$. Then step 1 needs $\frac{32!}{22!} \approx 2^{47.7}$ 10-uples of memory. Since $2^{47.7} < 251^6$, in step 2 on average less than one candidate for the right-hand side will have to be checked for each candidate for the left-hand side. But there are 12 variables on the left-hand side, so the time for the step 2 will be about $\frac{32!}{20!} \approx 2^{56.6}$ multiplications by A. We will call this algorithm $A0$. More generally, in this paper we will call by Ai, i=0,1,.. some algorithms with $k = 6$. $A0$ was the quickest algorithm described in [1] but it needs a huge amount of memory(about 1300 Terabytes).

Remarks.

1. To memorize a 10-uple one can use 10 bytes (since each value is modulo 251) or 10 5-bits (since there are only 32 possible values for each x_i).

2. The unit of time is, in first evaluation, the time for a multiplication by A. But it is possible to show that on average it will be appreciably less. This is because when the value of only one x_i changes (or a few x_i), the new value of $A * V_\pi$ can be quickly calculated from the old value of $A * V_\pi$.

These two remarks will be true for all the algorithms we are studying.

Example 2 : Algorithm $B0$.

If we choose $k = 3$ and $l = 5$ then we will need about 2^{24} 5-uples to memorize and $2^{65.1}$ in time for calculation. We will call this algorithm $B0$. More generally, in this paper we will call by Bi, i=0, 1,.. some algorithms with $k = 3$.

In the following paragraphs we will show how to modify $A0$ and $B0$ in order to improve these algorithms.

2 How to combine the ideas of [1] and [2].

In [2], J. Georgiades pointed out that we can add some equations in the system (1) to (n) of equations such that $A * V_\pi = 0$. As a matter of fact, $V = (v_1, \ldots, v_n)$ is known, and $V_\pi = (x_1, \ldots, x_n)$ is a permutation of V, so every symmetrical function of the x_i is known.

So we can add :

$$\begin{cases} \sum_{i=1}^{n} x_i & = & \alpha_1 \quad (G1) \\ \sum_{i=1}^{n} x_i^2 & = & \alpha_2 \quad (G2) \\ \sum_{i=1}^{n} x_i^3 & = & \alpha_3 \quad (G3) \quad etc\ldots \end{cases}$$

where $\alpha_1 = \sum_{i=1}^{n} v_i \ mod\ p$, $\alpha_2 = \sum_{i=1}^{n} v_i^2 \ mod\ p$ etc ...

For us, equation $(G1)$ will be very useful due to the fact that $(G1)$ is of the first degree. So with $(G1)$ we can apply Gaussian elimination with one more equation. Then we will be able to use the algorithms of [1] but with one variable less. (We will obtain in this way new algorithms which combine the ideas of [1] and [2] in a straightforward way!) Let us call "Algorithm A1" the modified version of algorithm $A0$ where the Gaussian reduction takes ($G1$) into account, and "Algorithm B1" the modified version of algorithm $B0$ where the Gaussian reduction takes ($G1$) into account. It is easy to see that algorithm $A1$ ($k = 6, l = 10$), needs $2^{52.2}$ in time (instead of 2^{56} for A0), for the same memory as A0 ($2^{47.7}$ 10-uples). The algorithm $B1$ ($k = 3, l = 5$), needs $2^{60.9}$ in time (instead of $2^{65.1}$) for the same memory as B0 (2^{24} 5-uples).

So we have obtained a small reduction of the time needed.

Remark. In order to defend PKP, one may consider to choose the A matrix and the vector V in such a way that the equation ($G1$) will be just a consequence of the equations (1) to (m). But this idea doesn't work because the number m of equations has been chosen in such a way that there is about one solution π for $A * V_\pi = 0$. So m is chosen such that $n! \approx p^m$ as explained in [3]. But for all the permutations π, (G1) is satisfied. So (G1) does not restrict the permutations solutions. So (G1) is really a "free" equation for the attack.

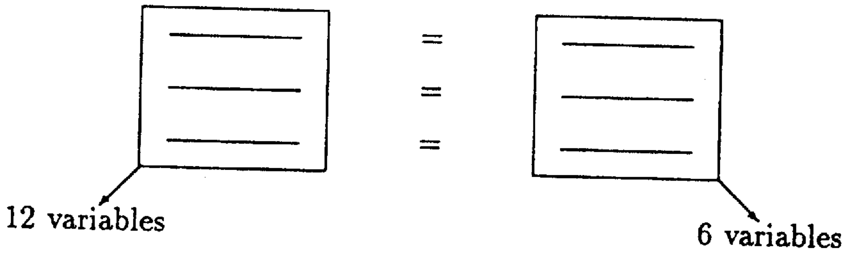
How can we use equation (G2)? In [2], J Georgiades was able to use the equation (G2) in his algorithm. Nevertheless we didn't see how to use it in a combined algorithm with the algorithms of [1]. This is because (G2) uses all the x_i and we eliminate some of them with the other equations (as we did for($G1$)). Thus we will obtain, in general, an equation of the second degree which seems impossible to split in two groups of distinct variables (some variables on the right-hand side of the equation and the others on the left-hand side), because we will have all the double products. So we do not use ($G2$), ($G3$) etc.., in our algorithms.

3 Introducing a set E on the left (or right) hand side.

In this section, we describe a new algorithm, which we will call : algorithm "B2". Like $B1$ and $B0$, our algorithm B2 will first be used for k=3 equations. But here, the new idea is that we will take into account the fact that the variables on the left-hand side cannot be used on the right-hand side.

After the Gaussian reduction with (G1) (as described in Section 2), there are 15+3=18 variables left in our equations. Here we split these variables with l=6.

This is represented schematically as follows :



The new idea is that we will distinguish C_{32}^{12} cases : one for each possible value for the set E of the 12 variables on the left-hand side. For every such case we do :

Step 1 : Precomputation compatible with E.

For each of the $\frac{20!}{14!} \approx 2^{24.7}$ possible values for the 6 variables on the right-hand side (when E is given), the right-hand side value is calculated and stored in a file $F1$ with the corresponding 6 variables.

Step 2 : Generation and test of a candidate.

For each of the $12! \approx 2^{28.8}$ possible values for the 12 variables on the left-hand side (when E is given) the left-hand side value is calculated. Then the file $F1$ of step 1 is used to obtain a list of possible $(x_1, ..., x_6)$ values. On average 1.76 such candidates will be given by $F1$ (because $\frac{20!}{14!} \approx 1.76 * (251)^3$). All these candidates will be immmediately tested (as usual) with more equations.

Total time for this algorithm B2 : $C_{32}^{12} * (2^{24.7} + 2^{28.8} * 1.76) \approx 2^{57.4}$.
Total memory : $2^{24.7}$ 6-uples.

For each new set E the new file $F1$ can use the memory of the old file $F1$ because we can forget the old file $F1$. So this algorithm B2 is quicker than B1 and needs about the same amount of memory.

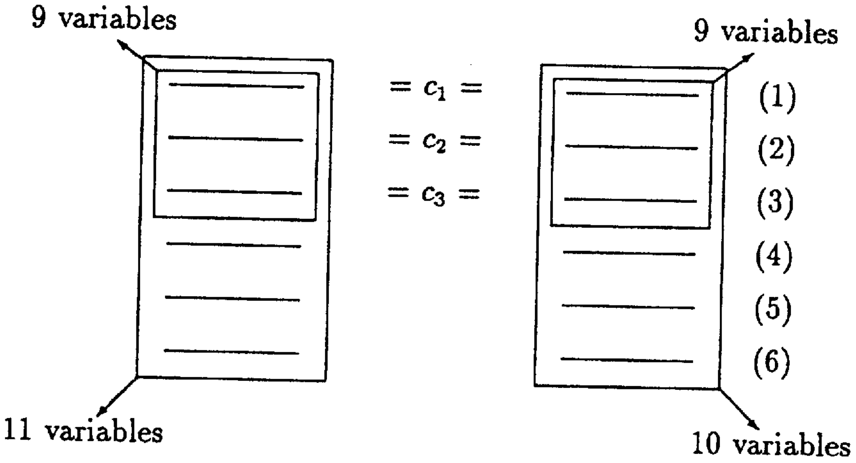
Remark : This idea to fix the set E of the variables used on the left-hand side can be generalized for values of the parameter other than the $k = 3$ and $l = 6$ that we have given. By convention, in our diagrams, the values that we will store during the precomputation phase are the values of the right-hand side. However, we can choose to fix the set E of these variables (with a higher value of l), or we can choose to fix the set E of the variables of the left-hand side (as we did in algorithm $B2$). In this paper, we will only describe some of the best algorithms that we have obtained.

4 Introducing some "middle values".

The idea given in paragraph 3 (fixing a set E in order to take into account the fact that the variables on the left-hand side cannot be used on the right-hand side) was useful to improve algorithm $B1$, because we were able to split the 18 variables in a group with much more variables (12) than in the other group (6). However, it is not possible to do this in order to improve algorithm $A1$ because here to minimize the time we have to split the 21 variables in two groups of about the same number of variables. So the idea of paragraph 3 does not work for algorithm $A1$.

Nevertheless, we will describe here another idea that will allow us to reduce the memory needed in $A1$ by $(251)^3$ without significative loss of time! We will call this algorithm "A2". Here k=6 and l=10 (as for A1), we have 15+6=21 variables after reduction with G1, but we will introduce the middle values c_1, c_2, c_3 of the first three equations.

This is represented schematically as follows :

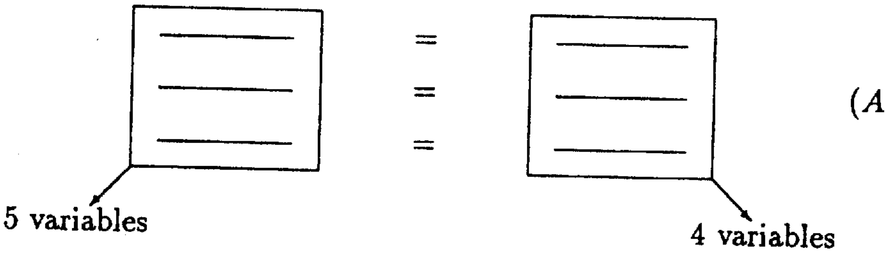


Here the new idea is that, in order to need less memory, we will try to find the solutions with a given (c_1, c_2, c_3). So, our algorithm A2 will proceed in $(251)^3$ cases : one for each possible value for (c_1, c_2, c_3). For each such cases we do :

Step 1 : Precomputation compatible with (c_1, c_2, c_3).

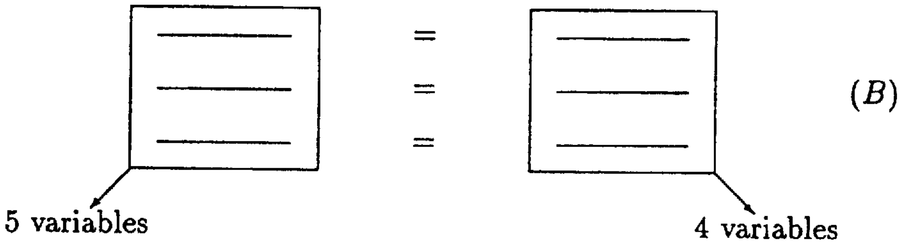
The aim of this step is to calculate and store in a file F1 all the possible values for the right-hand side of equations (4), (5), (6), (with the corresponding 10 variables of the right-hand side) when the right-hand side of equations (1), (2), (3) is (c_1, c_2, c_3). The problem is that we do not want to do that in time $\frac{32!}{22!}$ but in time $\frac{32!}{22!*(251)^3}$ (because we will have to do that $(251)^3$ times). For that purpose, we first notice that the right-hand side of (1), (2), (3) gives us three equations with 9 variables, when (c_1, c_2, c_3) is fixed. We will write these equations with 5 variables on the left and 4 variables on the right. We denote by (A) this system of three equations. (A) depends on c_1, c_2, c_3.

This is represented schematically as follows :



All the possible values for the right-hand side of (A) are calculated and stored in a file $F0$. So $F0$ contains $32 * 31 * 30 * 29$ values $\approx 0.055 * (251)^3$. We can notice that if c_1, c_2, c_3 are put on the left-hand side, then the file $F0$ can be precomputed once and for all : it does not depend on (c_1, c_2, c_3). Then for each of the 5 possible variables on the left-hand side of (A), the value of the left-hand side of (A) is calculated (this value depends on c_1, c_2, c_3). Then file $F0$ is used to obtain suitable values of the right-hand side variables (if any). If values are formed, and if all the 9 variables are distinct then the variable number 10 is introduced (there are 32-9=23 possible values for this variable) and the 23 corresponding values for the right-hand side of (4), (5), (6) are stored in $F1$ with the corresponding 10 variables. So, the time to generate $F1$ will be about $\frac{32!}{27!} \approx 2^{24.5}$ plus the number of values stored in $F1$, that is to say about $2^{24.5} + \frac{32!}{22! * (251)^3} \approx 2^{25.2}$. This will be done $(251)^3$ times. The memory needed is about $2^{19.7}$ 4-uples for $F0$ and $2^{23.8}$ 10-uples for $F1$.

Step 2 : Generation of a candidate compatible with (c_1, c_2, c_3).
When (c_1, c_2, c_3) is given, the left-hand side of equations (1), (2), (3) gives three equations with 9 variables. We will denote by (B) this system of equations. We will write (as in step 1) these equations with 5 variables on the left and 4 on the right.
This is represented schematically as follows :



All the possible values for the right-hand side of (B) are calculated and stored in a file $F'0$. Then a candidate compatible with (c_1, c_2, c_3) values is quickly generated as follows :
(i). Choose values for the 5 variables on the left-hand side of (B).
(ii). Then look in the file $F'0$ to see whether there are any 4 variables compatible with these 5 variables.
(iii). If (ii) provides 9 variables which are all distinct, then introduce variables 10 and 11 of the left-hand side of equations (4), (5), (6) and for each possible value calculate the left-hand side of (4), (5), (6).
(iv). Then look in the file $F1$ to see whether there are any 10 variables compatible with these 11 variables.
(v). If (iv) provides 21 variables which are all distincts then quickly test this candidate as usual with extra equations.
(vi). If the test proves negative then retry (i) with a new choice for the 5 variables.

Finally, there are $\frac{32!}{27!} \approx 2^{24.5}$ values for the 5 variables on the left-hand side of (B).
We will generate about $\frac{32!}{21! * (251)^3} \approx 2^{28.3}$ candidates for the 11 variables compatible with (c_1, c_2, c_3). So the total time for step 2 is about $2^{24.5} + 2^{28.4}$. The memory needed for the file $F'0$ is $\approx 2^{19.7}$ 4-uples.

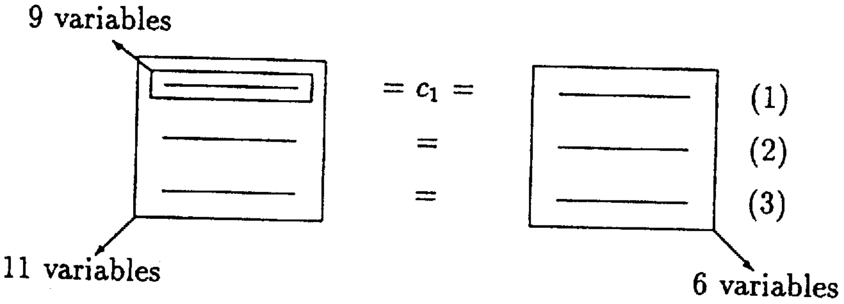
Conclusion : The total time for this algorithm A2 is : $(251)^3.(2^{25.2} + 2^{28.4}) \approx 2^{52.4}$. The total memory is about 2^{24} 10-uples (files $F0 + F'0 + F1$). So A2 is a great improvement on A1: with about the same time we need $(251)^3$ times less memory.
This idea to fix some "middle values" is very useful in order to improve the algorithms of [1]. We can use it also to improve the algorithm B2 of section 3 : it is possible to reduce the memory needed by making $251.C_{32}^{12}$ cases ; one for each value of c_1 and E. We will

call this algorithm $B3$. We will not go into details for $B3$ because we will now introduce another idea to improve $B3$ and we will explain in details the algorithm $B4$ obtained.

5 Introducing precomputation on the A matrix.

In algorithms $B2$ and $B3$ we had $k = 3$, and in the first three equations we had $15+3 = 18$ variables (after Gaussian reduction with G1). We will now see that if the variables are carefully chosen we can do even better and have only 17 variables in three independent equations. For this, the idea is that we can choose a particularly convenient Gaussian reduction. When the 17 variables that we want to keep in the equations are chosen, we will try to eliminate the other variables by Gaussian reduction. For three equations there are on average about $C_{32}^{17}/(251)^3 \approx 36$ non equivalent possible choices for the set of variables providing three independent equations.
Note : For six equations the probability that we could have only 20 variables instead of 21 is only about $C_{32}^{20}/(251)^6$ which is less than 10^{-6}. We have to be careful in the evaluation of this probability : when the 20 variables that we want to have in the six equations are fixed, there are a lot of ways to eliminate the 12 other variables. But if one succeeds they all succeed, and if one fails they all fail. So for six equations the probability to eliminate one variable in this way is very small. If it happens, it gives us an algorithm in time 2^{48} instead of 2^{52}. However for $k = 3$ equations we can easily eliminate one variable. The probability of success is close to 100% , and we can quickly find a good system of three equations : we will have to do at most $C_{32}^{17} \approx 2^{29}$ Gaussian reductions (this is negligible as compared with the time of the algorithm). With a program of Gaussian elimination on a PC we have made simulations, and we were able to find such equations quite easily. We will now describe an "algorithm B4", which will use such equations.
Algorithm B4 :

Step 0 : **Find three independent equations with only 17 variables.** (This is done as described above with Gaussian reduction with G1 and a good choice of the 17 variables). Then these are written with l=6, that is to say with 11 variables on the left and 6 on the right-hand side. Now the algorithm proceeds in $C_{32}^{11} \approx 2^{26.9}$ cases, one case for each possible value in the set E of the 11 variables on the left-hand side. Each cases is separated in 251 subcases : one for each possible value of the "middle value" c_1, as described in the following diagram :

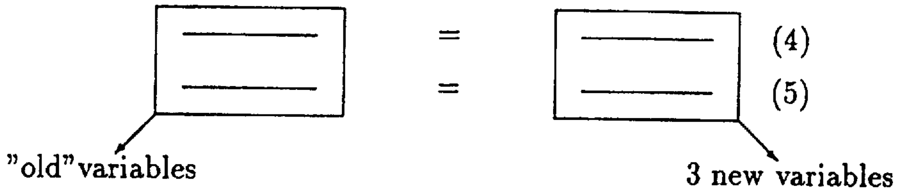


For each of the $251 * C_{32}^{11} \approx 2^{34.9}$ subcases we perform the following steps 1, 2 and 3.

Step 1 : Precomputation compatible with E and c_1. The aim of this step 1 is to calculate and store in a file $F1$ all the possible values for the right-hand side of equations (2) and (3) (with the corresponding 6 variables) when the right-hand side of equation (1) is c_1 and when the set E of the 11 variables on the left-hand side is known. The problem is that we want to do that not in $\frac{21!}{15!}$ time but in about $\frac{21!}{15!*251}$ time. For that purpose, we first notice that the right-hand side of (1) gives us one equation with six variables when c_1 is fixed. We will call this equation (A) and we will write (A) with 4 variables on the left and 2 variables on the right. A file F0 of the possible value for the right-hand side of (A) is calculated, so F0 contains $21 * 20 \approx 1.67 * 251$ values. Now for all possible values for the 4 left variables of (A), this file $F0$ will be used in order to find the corresponding 2 variables and to store in $F1$ all the solutions such that the 6 variables are distinct. The time to generate $F1$ will be then about $1.67 * 21 * 20 * 19 * 18 \approx 2^{17.9}$. $F1$ will contain about $\frac{21!}{15!*251} \approx 2.47 * 251^2 \approx 2^{17.3}$ values.

Step 2 : Generation of a candidate compatible with E and c_1.
The left-hand side of (1) gives us one equation with nine variables when c_1 is fixed. We will call this equation (B) and we will write (B) with 5 variables on the right-hand side and 4 variables on the left-hand side. Now we will distinguish $C_{11}^5 = 462$ more subcases (for each cases for E and c_1), one for each possible value for the 5 variables on the right-hand side of (B). As usual, a file $F'0$ of the 5! possible values for the right-hand side of (B) is introduced. Then for all possible values for the 4 left variables of (B) this file $F'0$ will be used in order to find whether there is a corresponding set of 5 variables (the probability is about $5!/251 \approx 0.48$ for each attempt). If this is the case, then variables number 10 and 11 of the left-hand side are introduced (E is fixed so we will have only 2 possible values for these variables when the first 9 variables are known). Then the left-hand side of equations (2) and (3) are calculated for this 11-variable candidate. Then the file $F1$ of step 1 is used in order to add 6 variables to our candidate. (In average, we will have 2.47 solutions for these 6 more variables). Then these 17-variable candidate is quickly tested as we will describe in step 3.

Step 3 : test of a 17-variable candidate.
At the end of step 2 we have obtained a 17-variable solution of equations (1), (2), (3). As usual we will now test this candidate with more equations. But the problem here is that when introducing one more equation, we introduce two more variables (because of the special equations (1), (2), (3) chosen with 17 variables instead of 18 variables). We want to test a candidate very quickly. In order to do this, we will first add only two equations that we call equations (4) and(5). These equations are written with the variables of (1), (2), (3) on the left and the three new variables on the right.



Once and for all at the beginning of this algorithm, a file $F"0$ with the possible values for the right-hand side of (4) and (5) has been stored. So $F"0$ contains $32 * 31 * 30 \approx 0.47(251)^2$ values. Now with $F"0$ a 17-variable candidate is immediately transformed to a 20-variable candidate which will be tested as usual with more equations (each extra equation needs only one additional variable).

Conclusion : The total time of this algorithm $B4$ is about $C_{32}^{11} * 251 * (2^{17.9} + 462 * (5! + 6 * 5 * 4 * 3 * 0.48 * 2 * 2.47)) \approx 2^{54}$. The total memory is about 2^{17} 6-uples (files $F'0$ and $F"0$ requiring negligible memory in comparison with $F1$).

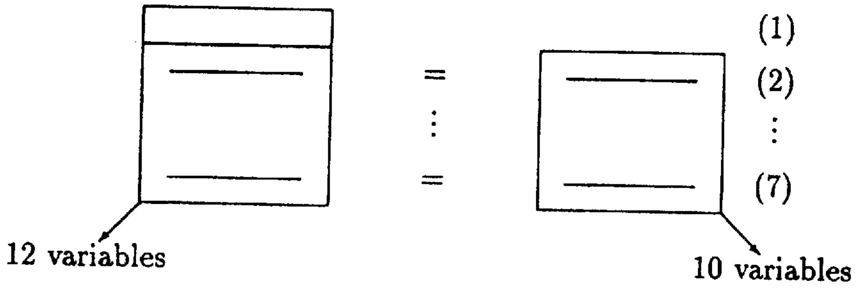
Note : As an algorithm with $k = 3$, the total time of $B4$ is near from optimal because there are about $\frac{32!}{15!} \cdot \frac{1}{251^3} \approx 2^{53.5}$ solutions for 3 equations and 17 variables. So an algorithm with $k = 3$ will need at least this time.

6 Introducing a special equation (1).

In the first note of paragraph 5 we have seen that with a small probability (less than 10^{-6}) we can have an algorithm in time 2^{48} instead of 2^{52} and it is possible to know if this algorithm will be convenient after only $C_{32}^{20} \approx 2^{28}$ gaussian reductions. Here we will see that there is also an algorithm in time 2^{48} with a probability of about 0.057 of success (or about 1/18). It is possible to know if this algorithm will be convenient after only $C_{32}^{12} \approx 2^{28}$ gaussian reductions.

The algorithm.

Step 0 : First, we have to find an equation with only 12 variables (instead of 16). The probability to find such an equation is about $C_{32}^{12} \cdot \frac{1}{251^4} \approx 0.0569$. If it exists, then we will find it after at most C_{32}^{12} gaussian reductions. If we find such an equation, call it equation (1). Then we introduce 6 more equations. The 12 variables of (1) are written on the left-hand side, the 10 more variables are written on the right-hand side. This is represented schematically as follows.



Step1 : precomputation. For each possible values for the 10 variables of the right hand side, the right-hand side value is calculated. By introducing middle values, like in paragraph 4, it is possible to reduce the memory needed. We do not give details since we are mainly interested in the time here.

Step2 : Equation (1) is written with 10 variables on the left-hand side, and 2 variables on the right-hand side. Now the algorithm proceeds in $C_{32}^{10} \approx 2^{26}$ steps : one for each possible value for the set E of the 10 variables on the left side of (1). At the beginning of each such step, a file F2 of the possible values for the right-hand side of (1) is stored. So there is 22.21 = 462 values in $F2$. Then, each possible value for the left-hand 10 variables is tested, $F2$ gives then in average 462/251 = 1.84 solutions for variables 11 and 12. Then $F1$ gives variables 13 to 20 and this 20-variable candidate is tested as usual with more equation.

Conclusion: The total time is only $C_{32}^{10}.10!.(1.84) \approx 2^{48.6}$. But the probability that this algorithm works is only about 0.057.

For PKP (37,64) a similar algorithm with $k = 15$ and $l = 21$ works. In this case the probability to find an equation (1) with 21 variables instead of 27 is closed to 100 per cent, because $C_{64}^{21}/251^6 \approx 164$. It gives the fastest algorithm we found for PKP(37,64) with 2^{116} in time.

7 Conclusion

We have investigated different ideas in order to improve the algorithms for the PKP problem. Let us recall the numerical results that we have obtained for the PKP (16,32), with a probability of success of about 100 per cent.

Previous algorithms for PKP(16,32) :

Memory	Time	Name
Negligible	2^{65}	J. Georgiades (cf[2])
2^{24} $(5 - uples)$	2^{65}	see [1] with k=3
2^{38} $(8 - uples)$	2^{60}	see [1] with k=5
2^{48} $(10 - uples)$	2^{56}	see [1] with k=6

Our algorithms for PKP(16,32) :

Memory	Time	Name
2^{17} $(6 - uples)$	2^{54}	Algorithm B4
2^{24} $(10 - uples)$	2^{52}	Algorithm A2

Note : Numeric values show that algorithm B4 needs only about 600 kbytes of memory and algorithm A2 about 100 Megabytes of memory, instead of about 1300 Terabytes for [1] with k=6, for example.

Now we summarize some results for PKP(37,64) (with a probability of success of about 100 per cent).

Previous algorithms for PKP(37,64) :

Memory	Time	Name
Negligible	2^{142}	J. Georgiades (cf[2])
2^{64}	2^{137}	see [1] with k=8 and l=11
2^{120}	2^{122}	see [1] with k=15 and l=21

Our algorithms for PKP(37,64) :

Memory	Time	Name
2^{27}	2^{123}	k=7, l=19, one variable eliminated, E on the right, c_1 to c_4 fixed
2^{52}	2^{119}	k=14, l=20, c_1 to c_8 fixed
2^{65}	2^{116}	k=15, l=21, equation (1) with 21 variables, c_2 to c_7 fixed

Note : The times given above are the times for finding all the solutions. If there is only one solution, this solution will be found in average after half the time given. Moreover, for PKP(37,64), there will be about three solutions: the secret solution plus about two solutions because $64! \approx 2 * 251^{37}$. So, for PKP(37,64), the average time to find one solution will really be a little less than the time given.

As we can see, our algorithms are still impracticable for PKP (37,64). However, although they need a lot of time, they are not completely unrealistic for PKP(16,32). Furthermore, the times given above are for sequential algorithms. Nevertheless, all our algorithms are very easy to implement in parallel, since they are not only designed in independent cases, but the memory needed in each of these cases is moderate.

Acknowledgements

We want to thank Thierry Baritaud for doing the Latex version of this paper.

References

[1] T. Baritaud, M. Campana, P. Chauvaud and H. Gilbert, "On the security of the Permuted Kernel Identification Scheme", Crypto'92, Springer Verlag.

[2] J. Georgiades, "Some Remarks on the security of the Identification Scheme Based on Permuted Kernels", Journal of Cryptology Vol. 5, n 2, 1992.

[3] A. Shamir, "An Efficient Identification Scheme Based on Permuted Kernels", Crypto'89, Springer Verlag.