IMPROVED ATTACKS ON KNAPSACK PROBLEM WITH THEIR VARIANTS AND A KNAPSACK TYPE ID-SCHEME

K.A. DRAZIOTIS[1] AND A. PAPADOPOULOU[2]

Abstract. In the present study we consider two variants of Schnorr-Shevchenko method (SS) for solving hard knapsack problems, which are on average faster than the SS method. Furthermore, we study the compact knapsack problem i.e. the solution space is not $\{0,1\}$ as in knapsack problem but some largest set, and we present an algorithm to attack this problem. Finally, we provide a sound three move id-scheme based on the compact knapsack problem.

1. Introduction-Statement of results

In the present study we consider the subset sum or knapsack problem. Given a list of n positive integers $\{a_1, ..., a_n\}$ and an integer s, find a binary vector $\mathbf{x}$,

$$\mathbf{x} = (x_i)_i \text{ with } \sum_{i=1}^{n} x_i a_i = s, \tag{1.1}$$

if such a binary vector exists. We define the density of the knapsack to be,

$$d = \frac{n}{\log_2 \max_i \{a_i\}_i}.$$

The decision version of the problem is known to be NP-complete [17]. The hard knapsack problems, as we shall see, have density close to 1. In this case, with high probability, there is only one solution e.g. [10, 20]. For low density knapsack problems, $d < 0.94$, we can apply [5]. If $d > 1$, then there are more solutions of the knapsack problem. There are some cryptographic schemes with $d > 1$ [33], but there are attacks against them [21, 34]. In our study we are interested in knapsacks with density close to one.

We shall optimize the algorithm of Schnorr-Shevchenko (SS) [30] and apply it to solve efficiently knapsack problems. We further address the problem of finding integers solutions of a linear Diophantine equation, where the solutions are in specific intervals (this problem is called compact knapsack problem). The solution to this problem heavily depends on the choice of the coefficients and on how to choose the solution space.

Finally, we provide a three move id-scheme based on compact knapsack problem. We shall prove that this system is sound, which is the minimal notion of security for id-schemes.

2010 *Mathematics Subject Classification.* 11D04, 94A60, 11Y16.

Key words and phrases. Subset Sum problem, Lattice, BKZ reduction, Id-scheme, Compact knapsack problem, Cryptography.

[1] Aristotle University of Thessaloniki, Department of Informatics, drazioti@csd.auth.gr.

[2] Aristotle University of Thessaloniki, Department of Mathematics, anastasia3g@hotmail.com.

Roadmap. The paper is organized as follows. In the next section we present some preliminaries for lattices. In section 3 we present relevant work that has been done. Section 4 presents two variants of the SS method and we provide some experimental results. In section 5 we address the problem of finding constraint solutions in linear Diophantine equations. Section 6 is dedicated to a construction of an id-scheme based on compact knapsack problem and we use the results of section 5 to provide a possible secure selection of the parameters of our id-scheme. Finally, in the last section we provide some concluding remarks.

2. Background on

See [16, 24] for a recent account on lattices.

Definition 1. *Let* $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k$ *linearly independent vectors of* $\mathbf{R}^n$. *The set of vectors*

$$L = \Big\{ \sum_{j=1}^{k} \alpha_j \mathbf{b}_j : \alpha_j \in \mathbb{Z}, 1 \leq j \leq k \Big\})$$

is called a lattice generated by $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k$. *The set* $B = \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k\}$ *is called a lattice basis of* L.

All the bases have the same number of elements, and this common number is called *rank* of the lattice. Also, we call the number n dimension of the lattice.

The most famous problem in lattices is the Shortest Vector Problem (SVP), which is the task of finding a smallest vector in $L(B)$. This problem is proved to be NP-hard under randomized reductions [3]. Also the SVP_γ is an approximation SVP with factor γ. That is, we are looking for lattice vectors $\mathbf{x} \neq \mathbf{0}$ with, $||\mathbf{x}|| < \gamma(n)||\mathbf{y}||$ for every $\mathbf{y} \in L(B) - \{\mathbf{0}\}$. A famous conjecture in lattices is the following (see [9, Section 4, p.87]) and [24, Conjecture 1]):
Conjecture. There is no polynomial time algorithm which solves SVP_γ with $\gamma(n) = Poly(n)$.
A similar problem is the Closest Vector Problem (CVP). In this case given a target vector $\mathbf{t} \in span(B) = \Big\{ \sum_{j=1}^{k} c_j \mathbf{b}_j : c_j \in \mathbf{R} \Big\}$, we are looking for a vector $\mathbf{x} \in L(B)$, such that $||\mathbf{x} - \mathbf{t}|| \leq ||\mathbf{y} - \mathbf{t}||$ for every $\mathbf{y} \in L$.

Having a lattice we need to work with *good* bases. That is, the basis vectors have small lengths and are almost orthogonal. Such a basis is called reduced. A widely known algorithm that provides such bases is the LLL algorithm, which was developed in 1982 by A. Lenstra, H. Lenstra, and L. Lovász [22]. Furthermore, LLL solves SVP_γ in polynomial time for $\gamma = 2^{k/2}$. In fact, LLL behaves better in practice than in theory. Experiments showed [14] that LLL behaves as an SVP-oracle for dimensions ≤ 35 (i.e., solves SVP with high probability in polynomial time).

Another reduction in lattices is the BKZ_β (Block Korkin-Zolotarev) introduced by Schnorr and Euchner [29]. For $\beta = 2$, BKZ coincides with LLL. This algorithm provides better quality of the vector basis, but it is exponential with respect to the blocksize.

3. Previous Work

Shroeppel-Shamir Algorithm [31]. This algorithm was the best for solving hard knapsacks until 2009, with time complexity $\tilde{O}(2^{n/2})$ and memory requirement $O(2^{n/4})$. For simplicity assume that n is divided by 4.

First, the set $S = \{\sum_{i=1}^{n/2} a_i x_i : x_i \in \{0,1\}\}$ with all possible values of this sum is constructed. We decompose the sum $s = \sum_{i=1}^{n} a_i x_i$ as:

$$s = \sigma_1 + \sigma_2 + \sigma_3 + \sigma_4,$$

where each σ_i is a smaller knapsack of $n/4$ elements:

$$\sigma_1 = \sum_{i=1}^{n/4} a_i x_i,\ \sigma_2 = \sum_{i=n/4+1}^{n/2} a_i x_i,\ \sigma_3 = \sum_{i=n/2+1}^{3n/4} a_i x_i,\ \sigma_4 = \sum_{i=3n/4+1}^{n} a_i x_i.$$

We construct the following lists for a value $\sigma \in S$ which is chosen appropriately:

- Σ_2: all possible $2^{n/4}$ values of σ_2. We find these values and then we sort the list. For example, if $n = 40$ we must compute a list σ_2 of 2^{10} elements, where each vector $\mathbf{x}$ of the list, contains the binary digits of a number in $[0, 2^{10} - 1]$.
- Σ_1: all possible values of σ_1 such that

$$\sigma_{12} = \sigma_1 + \sigma_2 = \sigma \pmod{2^{n/4}}$$

- Σ_4: all possible $2^{n/4}$ values of σ_4 (we sort this list as before).
- Σ_3: all possible values of σ_3 such that

$$\sigma_{34} = \sigma_3 + \sigma_4 = s - \sigma \pmod{2^{n/4}}$$

Thus we get $\sigma_{12} = \sigma \pmod{2^{n/4}}$ and $\sigma_{34} = s - \sigma \pmod{2^{n/4}}$. We search for a collision between the lists $\{\sigma_{12}\}$ and $\{s - \sigma_{34}\}$. For the right choice of σ we have found elements such that $\sigma_{12} + \sigma_{34} = s$, thereby we have solved the initial knapsack problem. The two lists $\{\sigma_{12}\}$ and $\{\sigma_{34}\}$ require $\tilde{O}(2^{n/4})$ time, as well the collision after sorting. Besides, we must find the right value of $n/4$-bit number σ. The total running time is $\tilde{O}(2^{n/2})$ and the memory requirement $\tilde{O}(2^{n/4})$.

The Howgrave-Graham Joux Algorithm. In 2010, Howgrave-Graham and Joux [19] improved the Schroeppel-Shamir algorithm. They claimed that the new algorithm can solve hard knapsack problems with heuristic running time $\tilde{O}(2^{0.3113n})$ and memory $\tilde{O}(2^{0.256n})$. But this running time was proven wrong by May and Meurer who estimated that the running time of this improved algorithm is $\tilde{O}(2^{0.337n})$ [4].

This improvement depends on the more degrees of freedom. Here, the two lists we construct can overlap and that was not allowed in the previous algorithm. We consider the knapsack $s = \sum_{i=1}^{n} a_i x_i$ and for simplicity assume that n is divided by 4 again and $\sum_{i=1}^{n} x_i = n/2$. There exist pairs (σ_1, σ_2) with Hamming weight $n/4$ such that $\sigma_1 + \sigma_2 = s$. This decomposition is not unique and we define these pairs as following:

$$\sum_{i=1}^{n} a_i y_i = \sigma_1, \qquad \sum_{i=1}^{n} a_i z_i = \sigma_2$$

where $y_i, z_i \in \{0,1\}$.

After choosing a value $M \approx 2^{n/2}$ and a random R modulo M we construct two lists such that:

$$\sigma_1 = R \pmod M \text{ and } \sigma_2 = s - R \pmod M$$

and then using all possible vectors $\mathbf{y}$ and $\mathbf{z}$, we search for collisions, in order to find a solution of the initial knapsack. Since the Hamming weight is $n/4$ for each sub-knapsack, the number of solutions we expect to find is $k = \frac{\binom{n}{n/4}}{M}$. The required time for this procedure is $O(k)$.

New improvement by Becker, Coron and Joux. An other improvement of the previous algorithm, that was presented in Eurocrypt 2011 [4], reduces the (heuristic) running time down to $\tilde{O}(2^{0.291n})$. The basic idea of Becker, Coron and Joux algorithm is adding a bit more degrees of freedom. The solutions of the two sub-knapsacks consist of coefficients from $\{-1, 0, 1\}$. As a result, there are more representations of the solution of the original knapsack.

3.1. **The method of Schnorr-Shevchenko.** In this subsection we present Schnorr and Shevchenko method for knapsack problems with density close to 1. This method is clearly faster (in practice), than the one that Becker, Coron and Joux presented. There is some theoretical evidence for this method, which we shall provide after presenting the method.

Here BKZ-reduction is used into a specific basis to find the solution. We use the lattice $L(B)$ generated by the rows of the matrix B.

$$B = \begin{bmatrix} 2 & 0 & \dots & 0 & Na_1 & 0 & N \\ 0 & 2 & \dots & 0 & Na_2 & 0 & N \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \dots & 2 & Na_n & 0 & N \\ 1 & 1 & \dots & 1 & Ns & 1 & \frac{n}{2}N \end{bmatrix} \in \mathbb{Z}^{(n+1)\times(n+3)}. \tag{3.1}$$

The basis of the lattice $L(B)$ consists of these $n+1$ row-vectors, with $n+3$ elements each one. In the following examples we consider that $n = 80$ and $(a_i)_i$ are random integers in the set $\mathcal{A} = (1, 2^n + 1] \cap \mathbb{Z}$. This choice provides densities very close to 1, since the denominator of the fraction $d = \frac{n}{\log_2 \max_i\{a_i\}_i}$ will be close enough to n. To see this, we write the set $\mathcal{A}$ as

$$\mathcal{A} = \{2, 3, \dots, 2^n + 1\} = \{2, \dots, 2^{n-1} + 1\} \cup \{2^{n-1} + 2, \dots, 2^n + 1\}.$$

We denote the first subset of $\mathcal{A}$ by $\mathcal{A}_1$ and the second by $\mathcal{A}_2$. Then,

$$Pr(x \in \mathcal{A}_2 : x \text{ is randomly chosen from } \mathcal{A}) = \frac{1}{2}.$$

Thus, $1 < \frac{n}{\log_2 x} < \frac{n}{n-1}$, with probability $\approx 1/2$, since

$$\left\{ x \in \mathcal{A} : 1 < \frac{n}{\log_2 x} < \frac{n}{n-1} \right\} = \{2^{n-1} + 1, \dots, 2^n - 1\},$$

whose cardinality is strictly less than that of $\mathcal{A}_2$.

Since our sample is n, in our case $n \geq 80$, with probability $1 - \frac{1}{2^n}$, we shall get a coefficient a_i in $\mathcal{A}_2$. So the density with probability very close to 1, will be greater than 1 and at most $\frac{n}{n-1} \approx 1$.

Moreover, N must be larger than $\sqrt{n}$. For $n = 80$, we choose $N = 16$. Let $\mathbf{b} = (b_1, b_2, \dots, b_{n+3})$ in $L(B)$, that satisfies:

$$|b_1| = |b_2| = \cdots = |b_n| = 1, \ |b_{n+2}| = 1 \ \text{ and } \ b_{n+1} = b_{n+3} = 0.$$

Then, we can reveal the solution $\mathbf{x} = (x_1, \ldots, x_n)$ from

$$x_i = \frac{|b_i - b_{n+2}|}{2}, \quad \text{for } i = 1, 2, \ldots, n,$$

with the property $\sum_{i=1}^{n} x_i = n/2$ (here we considered totally balanced solutions, i.e. with Hamming weight $H = n/2$). The inverse fact is that every solution is written as previous. The integer $n/2$ (assume that n is even) can be an integer $H \in \{1, ..., n-1\}$. More precisely the algorithm works by the following way:

- In the first 5 steps we apply BKZ-reduction to the basis B without pruning and with blocksizes 2^k for $k = 1, 2, 3, 4, 5$. Before the reduction we permute the rows of the matrix in order to:
  - first rows have a nonzero element in column $n + 2$
  - sort in ascending order the other rows according to their Euclidean norm as row-vectors.

  Every step takes as input the matrix B from the previous iteration. If the solution is found, the algorithm stops.

- In case the algorithm did not find the solution in the first 5 steps, BKZ-reduce the basis independently with blocksizes: $bs = 30, 31, 32, \ldots, 60$. The pruning parameter is 10 for $bs = 30$, 11 for $bs = 31$, 12 for $bs = 32$ and then 10 again for $bs = 33$ and so on. Always terminate if the solution has been found.

We provide the following auxiliary Lemma. Subsequently, we substitute $n/2$ with an integer $H \in \{1, 2, ..., n-1\}$.

Lemma 3.1. *A solution $\mathbf{x} = (x_1, \ldots, x_n)$ of the knapsack problem (1.1), with Hamming weight H, is given by a vector $\mathbf{b} = (b_1, b_2, \ldots, b_{n+3}) \in L(B)$, with*

$$|b_1| = |b_2| = \cdots = |b_n| = 1, \ |b_{n+2}| = 1 \quad \textit{and} \quad b_{n+1} = b_{n+3} = 0 \tag{3.2}$$

by the relations

$$x_i = \frac{|b_i - b_{n+2}|}{2}, \quad \textit{for } i = 1, 2, \ldots, n.$$

Also, the inverse is true.

Proof. There are integers λ_j $(j = 1, 2, ..., n+3)$ such that, every vector $\mathbf{b} \in L(B)$ is written as $\mathbf{b} = \sum_{j=1}^{n+3} \lambda_j B_j$, where B_j denotes the j-th row vector of the matrix B (3.1). Then,

$$b_i = 2\lambda_i + \lambda_{n+1} \ (i = 1, 2, ..., n)$$

$$b_{n+1} = N \sum_{i=1}^{n} \lambda_i a_i + \lambda_{n+1} Ns, \ b_{n+2} = \lambda_{n+1}$$

$$b_{n+3} = N \sum_{i=1}^{n} \lambda_i + \lambda_{n+1} NH,$$

where the b_i's satisfy the conditions,

$$|b_1| = |b_2| = \cdots = |b_n| = 1,$$

$$|b_{n+2}| = 1, \tag{3.3}$$

$$b_{n+1} = 0, \tag{3.4}$$

$$b_{n+3} = 0. \tag{3.5}$$

From (3.3) we get $\lambda_{n+1} = \pm 1$, from (3.4) $\sum_{i=1}^{n} \lambda_i a_i = -\lambda_{n+1} s$, and from relation (3.5) we get $\sum_{i=1}^{n} \lambda_i = -\lambda_{n+1} H$. To simplify the exposition, assume that $\lambda_{n+1} = 1$. The other option leads to similar results. If $b_i = 1$, then $\lambda_i = 0$. Otherwise, $\lambda_i = -1$. Therefore, $\lambda_i \leq 0$. The solution is revealed by, $x_i = \frac{|b_i - b_{n+2}|}{2} = |\lambda_i| = -\lambda_i \geq 0$. As a result λ_i's are 0 or -1, and satisfy $\sum_{i=1}^{n} \lambda_i a_i = -s$, $\sum_{i=1}^{n} \lambda_i = -H$. The second equality is required because of the Hamming weight.

(Inverse). Now we shall prove that every knapsack solution $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with $\sum_{i=1}^{n} x_i = H$ corresponds to some $\mathbf{b} \in L(B)$, that satisfies the relations (3.2).

Consider $\mathbf{b} = (b_1, \ldots, b_n, 0, 1, 0)$, with $b_i = 1 - 2x_i$. Since $x_i \in \{0, 1\}$ we get $|b_i| = 1$ $(1 \leq i \leq n)$. It is enough to prove that $\mathbf{b} \in L(B)$. But

$$\mathbf{b} = -x_1 B_1 - \cdots - x_n B_n + B_{n+1}.$$

So, the inverse is also true. □

3.2. **Theoretical Evidences of the method.** First we provide the following theorem.

Theorem 3.2. [20] *Assume that there is an SVP-oracle and the knapsack problem has a solution. Then with high probability we can solve all knapsack problems with density* < 0.6463.

With SVP-oracle we mean a probabilistic polynomial algorithm which given a lattice L, it provides a shortest vector of L with high probability. Unfortunately, in practice we do not have SVP-oracles (at least for lattices with large rank). Experiments made by Nguyen and Gama [14] suggest that LLL behaves as an SVP-oracle for dimensions ≤ 35 and BKZ-20 algorithm [29] (i.e. BKZ with blocksize 20), for dimensions ≤ 60. Furthermore, two more simplified proofs of the previous theorem were also given in [5, 13]. Also in [27] the algorithm was tested experimentally providing some improvements.

The previous Theorem was improved by Coster *et al.* [5]. The new density bound was improved to 0.9408. Their approach is in the same spirit of [20]. So the assumption of the existence of an SVP-oracle remained (this assumption is a serious drawback of these methods, when we try to apply them in practice). They applied LLL reduction algorithm to the lattice generated by the rows of the matrix:

$$B = \begin{bmatrix} 1 & 0 & \cdots & 0 & Na_1 \\ 0 & 1 & \cdots & 0 & Na_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & Na_n \\ \frac{1}{2} & \frac{1}{2} & \cdots & \frac{1}{2} & Ns \end{bmatrix},$$

where N is a positive integer $> \sqrt{n}/2$.

The SS-method uses a similar matrix B. In fact, experimentally we get that SS-method provides a very good approximation of an SVP-oracle for lattices of dimension ≈ 80.

4. A new strategy for hard knapsack problems

We consider two variants of SS's method. In our experiments we used fpLLL interface for python [11, 12] and in some cases we used Sagemath [32]. The code we used is online in github (see [8]). In our experiments we used a PC with 8Gb ram and i3 3.5GHz cpu.

Our first observation is that the SS's method is influenced by the density of the knapsack problem. For instance, table 1 provides the relation for 40 random instances of dimensions $\{72, 76, 80\}$ and densities in $\{1, 0.975, 0.95, 0.92\}$, with the mean number of rounds until SS-method terminates successfully.

	density	1	0.975	0.95	0.92
dim = 80	success round (average)	12.57	10.38	9.69	8.35
dim = 76	success round (average)	9.1	8.83	7.95	5.6
dim = 72	success round (average)	8.15	7.9	7.7	5.5

Table 1. Relation of the density of random knapsack problems and the average number of rounds until SS-method terminates successfully.

Also, after some critical round say R, the SS method is really slow for rounds $\geq R$. This is because the execution of SS's method is dominated by the running time of BKZ, which heavily depends on the blocksize and pruning. For dimension 80, experimentally this value is $R \approx 18$. For instance see figure 1.

Fig. 1 We considered 50 knapsack problems of dimension 80 and density ≈ 1 and we measured the average time for all the instances terminated in round 5 or 6,...,or 23.

Finally, if we reduce the original knapsack problem to some easier problems, i.e. having smaller dimension and density, then it would be faster to solve the reduced knapsack problems instead of the first initial knapsack.

But there is also a theoretical reason for this. As we already wrote in subsection 3.2, there is strong evidence that lower density knapsack problems, i.e. $d < 0.94$ are easier than hard knapsack problems, i.e. with density close to 1. So our first variant is supported from the previous theoretical evidence.

We summarize our first variant in two simple steps.

First Variant

1. Execute SS algorithm until round = 5 (the first stage of SS's method)
2. Brute force on the b initial bits of the solution and for each value (of the choice of the b initial bits) reduce the initial knapsack to some with smaller densities and dimension. Then apply SS's method until round R (overall rounds: $5 + R$).

This method depends on two parameters : b which is the number of the initial bits, where we shall apply brute force, and the parameter R which is the maximum round for the second stage of SS method.

After enough experiments, for dimension $n = 80$ and 84 we concluded that the pair $(b, R) = (4, 11)$ provides the best results when we compare the variant with the original method.

If the first step fails, then we start a brute force on the four initial bits of the candidate solution. Say we start with $[0, 0, 0, 0]$, i.e. we assume now that the solution is of the form $[0, 0, 0, 0, ...]$. Then we consider the reduced knapsack problem $\sum_{j=5}^{n} a_j x_j = s$. This has dimension 76 (if $n = 80$) and density ≈ 0.95. We run SS's method until the 11th round. If it fails, then we continue to the next quadruple of bits, say $[0, 0, 0, 1]$, then the reduced knapsack is $\sum_{j=5}^{n} a_j x_j = s - a_4$. We execute again SS and so on, until the solution is found. We summarize the results in figure 5. To collect the data for this diagram we executed the following experiment : we chose a random knapsack of density 1 with a random solution having Hamming weight $n/2$, and then we run SS's method and the SS-variant. Finally, we measured the time for each experiment.

Also, note that the distance from round 10 to round 16 is shorter than the distance from round 16 to round 20. This is because after some rounds BKZ is very slow.

Fig. 2 We compare SS's Method, for dimension 80, with the first variant for some randomly chosen instances of knapsack problem. The $x-$ axis is the number of instances and the $y-$axis is the cpu time. Moreover, we sorted the results with respect the times of the original method.

Finally, we run examples with dim = 84. The results are summarized in figure 3. This method can be easily parallelized, since each reduced knapsack problem corresponding to some possible solution can be run in one CPU Core. If we have a cluster with 16 PC's, we dedicate each one to solve one such reduced problem. In this way we can optimize the method.

Fig. 3 We compare SS's Method with the first variant, for 32 randomly chosen instances of density 1 and dim = 84.

Second Variant

The second variant is even simpler than the previous. In the first variant we decreased the numerator of the density $d = \frac{n}{\log_2 \max_i\{a_i\}_i}$. Now we shall increase the denominator. The idea is seemingly very simple and works in practice. We fix a number, say $B_n = 2^{n+b}$, for some positive integer b. Also, assume that $x_1 = 1$. If we substitute $a_1 \leftarrow a_1 + B_n$ and $s \leftarrow s + B_n$, then the seemingly new knapsack will have density

$$d' = \frac{n}{\log_2 \max_i\{a_i\}_i} = \frac{n}{\log_2 a_1} = \frac{n}{n+b} = \frac{1}{1 + b/n} < 1.$$

This is because all a_i's, for $i \geq 2$ are smaller than $2^n + 1$.

Someone, may suggest to solve the new knapsack of dimension $n - 1$ since we know the first bit. This would have negligible impact to the new density. In fact the new density would be $\frac{n-1}{\log_2 \max_i\{a_i\}_{2 \leq i \leq n}}$, which is again very close to 1.

In order to choose the most suitable value of b we made enough experiments in dimension 80. We found that for $b \in \{5, 6\}$ we get the best results. Remark that for b large enough the new density is close to 0.5 so we expect the variant to be very fast. Unfortunately, for $b \geq 7$ this method failed for the majority of instances we executed. Furthermore, for $b < 5$ the average times of success were almost the same.

If $n = 80$ and $b = 6$, then we get $d' \approx 0.93$. Someone would expect that BKZ, fast enough would remove B_n from the matrix and then again the situation would be the same as before the substitution. But it seems that this is not the case. For instance, in figure 4 we compare the two methods, the original and the previous variant (for $b = 6$). The vertical axis is the time. The only drawback of this variant is the assumption that we know the first bit. In other words this variant succeeds with probability $1/2$. There is a small number of instances($\approx 10\%$) where the original method is faster.

Fig. 4 We compare the SS's Method with the second variant for randomly chosen instances of knapsack problem with dimension 80, Hamming weight $n/2$ and density close to 1. The two horizontal axes for 16 and 10, are (on average) the time that the original method takes until round 16 and 10 (resp.). Furthermore, we sorted the results with respect the times of the original method.

4.1. **Final Remarks.** We provide one more figure, that contains data when we consider both the variants. On average the second variant is slightly faster. The drawback of the second variant is that it is probabilistic i.e. we need two know the first bit and even then, the 10% of the random instances fails.

Fig. 5 We compare SS's Method, for dimension 80, with the first variant and second variant for some randomly chosen instances of knapsack problem. The $x-$ axis is the number of instances and the $y-$axis is the cpu time. Furthermore, we sorted the results with respect to the times of the original method.

The asymptotic complexity of SS attack (and its variants) is dominated by the time complexity of BKZ. Furthermore, for the specific lattices, the density of the knapsack contributes to the time complexity. That is, for low density the SS attack is faster than for density close to 1. If we restrict to density close to 1, then the complexity of the SS attack is dominated only by the complexity of BKZ. The asymptotic complexity of BKZ is exponential with respect to the blocksize, see [18, section 3]. So, for large dimensions and density close to 1, the time complexity of the original SS-attack is that of BKZ. On the other hand, in the variants we managed to decrease the density, therefore, except the running time of BKZ also the density has a positive impact, and we expect the overall algorithm to run faster. Since the analysis of BKZ is not well studied, we can not provide specific formula for the complexity.

4.2. **Theoretical Evidences of the variants.** Knapsack problems with density close to 1, are considered hard since there is no efficient lattice attack. For low density knapsack was shown that can be attacked via lattices attack. In fact in [10, Proposition 1.2], was proved that density close 1 is the hardest case for knapsack problems. So reducing a hard knapsack problem to a small number of knapsack problems that have lower density will efficiently attack hard knapsack problems.

This is the idea behind the two variants and is theoretically based on the result [10] (see also [19, Introdution]). In the first variant this is achieved by solving in the worst case $2^4 = 16$ knapsack problems with density close to 0.95 and dimension 76. This has a major impact on the problem since such knapsack problems are easier than the original. So we expect to get faster a solution. Furthermore, this variant can be easily parallelized. In the second variant we artificially lower the density by substituting the constant with some larger. The dimension remains the same but the new density is ≈ 0.93.

5. Compact knapsack

In this section we study a more general problem, the compact knapsack problem, in which we allow solutions in a specific set.

5.1. **A CVP attack to compact knapsack.** An approach to attack compact knapsack is to reduce it, to a suitable closest Vector Problem (CVP). The idea is the following. We denote with I_α the set of integers having $\alpha-$ bits and with t_α the integer $2^{\alpha-1} + 2^{\alpha-2}$. Let $\sum_{i=1}^n a_i x_i = a_0$, and we restrict our solutions to a set $\mathcal{S} \subset \mathbf{Z}^n$. Let also $\mathbf{y}$ be a solution (for instance we can use Euclidean algorithm). Let L be the lattice generated by the integer solutions of $\sum_{i=1}^n a_i x_i = 0$ and choose a suitable target vector $\mathbf{t} \in span(L)$ (with $\mathbf{t} \in \mathcal{S}$). Then we solve the CVP instance $CVP(L, \mathbf{t}, |\cdot|_\infty)$ and say $\mathbf{b}$ its output. We expect the solution $\mathbf{x} = \mathbf{y} + j\mathbf{b}$ (for some small integer j) to be in $\mathcal{S}$. In fact, using the previous attack, for dimension $n \leq 200$, $R \leq 200$, $\mathbf{a} \in I_R^n$, and with $\mathbf{x} \in \mathcal{S} = I_R^n$ we always get a solution in I_R^n using the target vector $\mathbf{t} = (t_R, ..., t_R)$. The situation becomes harder if we consider groups of $\{x_j\}_j$'s having different bits. Assume that, R and n are even integers. For instance, if the first $n/2$ entries of $\mathbf{x} = (x_j)_j$ have $R-$ bits and the other half have $R/2-$ bits, we get a solution having (on average) the half of entries in $I_R \cup I_{R/2}$. We used the target vector, $\mathbf{t} = (t_R, .., t_R, t_{R/2}, ..., t_{R/2})$, where the first $n/2$ entries are equal to t_R and the rest to $t_{R/2}$.

We provide the pseudocode of this attack.

CVP-attack:
`INPUT:` $\mathcal{S} \subset \mathbf{Z}^n$, $\mathbf{a} \in \mathbf{Z}^n_{>0}$, a_0 `such that the equation` $\sum_{i=1}^n a_i x_i = a_0$ `has a solution in` $\mathcal{S}$, `and a target vector` $\mathbf{t} \in \mathbf{R}^n \cap \mathcal{S}$.
`OUTPUT: A solution` $\mathbf{x} \in \mathcal{S}$ `or in the worst case returns a solution that satisfies some constraints.`

`01. compute a solution` $\mathbf{y}$ `of` $\sum_{i=1}^n a_i x_i = a_0$ (using either [1] or [7])
`02. compute a basis` B `of the lattice` $L = \{\mathbf{x} \in \mathbf{Z}^n : \sum_{j=1}^n a_j x_j = 0\}$
`03.` $B \leftarrow LLL(L(B))$
`04.` $\mathbf{b} \leftarrow CVP(L, \mathbf{t}, |\cdot|_\infty)$
`05. return the best vector (i.e. the one that meets more constraints) from the set` $\{\mathbf{y} + j\mathbf{b} : j = -10, \dots 10\}$

In practice, in line 04 we can use Babai algorithm [16, Chapter 18] (which has polynomial running time) with the usual Euclidean norm $||\cdot||$. Since, $||\cdot||_\infty < ||\cdot|| < \sqrt{n}||\cdot||_\infty$ in $\mathbf{R}^n$, we get a very good approximation of $CVP(L, \mathbf{t}, ||\cdot||_\infty)$. We also tried the exact CVP algorithm (using enumeration) of FpyLLL [12]. For $R > 80$ the exact-CVP algorithm was very slow and we did not manage to get a solution. But, for $R \leq 80$ we got exactly the same results as provided by Babai.

In tables 2 and 3 we provide some results using the previous algorithm.

n	R	right entries(on average)
$\mathcal{S}_1 : 30$	40	100%
$\mathcal{S}_2 : 30$	40	50%
$\mathcal{S}_3 : 30$	40	50%
$\mathcal{S}_4 : 30$	40	62.8%

TABLE 2. Here we assume that $\mathbf{x} \in \mathcal{S}_i$, where $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4$ are I_R^n, $I_R^{n/2} \times I_{R/2}^{n/2}$, $I_{R/2}^{n/2} \times I_{R/4}^{n/2}$ and $I_R^{n/3} \times I_{R/2}^{n/3} \times I_{R/4}^{n/3}$, respectively. Further $a_j \xleftarrow{\$} I_{R/2}$. We executed 80 random instances for each row. Similar results we got for $(n, R) = (60, 80), (103, 100)$.

n	R	right entries(on average)
$\mathcal{S}_1 : 60$	320	100%
$\mathcal{S}_2 : 60$	320	50%
$\mathcal{S}_3 : 60$	320	50%
$\mathcal{S}_4 : 60$	320	62.4%

TABLE 3. Here $a_j \xleftarrow{\$} I_{R/8}$. We executed 80 random instances for each row.

We can improve on this attack if we apply the following strategy. We explain for the case $\mathbf{x} \in \mathcal{S}_2$. We choose K randomly from the set $I_{R/2}$. Then we apply CVP-attack to $\sum_{j=1}^{n/2} a_j x_j = a_0 - K$. Almost always we get a solution $\mathbf{x}_1 = (x_1, ..., x_{n/2}) \in I_R^{n/2}$. Then, we check if the equation $\sum_{j=n/2+1}^{n} a_j x_j = K$ has a solution in $I_{R/2}^{n/2}$. In this way, we improved the 50% to 64% in the second row of table 2 and 3. The same if the solution set is $\mathcal{S}_3$.

5.2. **A branch and bound algorithm.** Since the previous method does not find all the right entries, but find a vector $\mathbf{x}$ that meets enough constraints, we can search in the neighborhood of $\mathbf{x}$ for a better vector.

Let $\{\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}\}$ be a basis of the lattice $\{\mathbf{y} \in \mathbb{Z}^n : \sum_{j=1}^{n} a_j y_j = 0\}$. We apply the CVP-attack of the previous subsection and let $\mathbf{x}$ the output of the CVP-attack. The method we provide uses this basis to get a new better solution $\mathbf{x}'$, that satisfies, all or some, of our bound constraints. Starting from the solution $\mathbf{x}$ that does not meet the constraints, we update it using the relation $\mathbf{x}' \leftarrow \mathbf{x} + k\mathbf{b}_j$ for $1 \leq j \leq n-1$ and for some $k \in \mathbb{Z} - \{0\}$ with $|k| \leq rad$ for some positive integer rad. In every step

we calculate the number of right entries $V = V_{\mathcal{S}}(\mathbf{x})$ of the update vector $\mathbf{x}$. If the new V' satisfy $V' \geq V$, then we update the solution $\mathbf{x}$ and increase (or decrease) k for a better solution, else we use another vector $\mathbf{b}_j$ from the basis. We provide the pseudocode of this algorithm.

The function UPDATE:
INPUT: $L = \{\mathbf{b}_j\}_j, \mathbf{x}, n, \mathcal{S}, V = V_{\mathcal{S}}(\mathbf{x}), rad, flag \in \{-1, 1\}$
OUTPUT: A solution $\mathbf{x} \in \mathcal{S}$ that satisfies more constraints than the initial solution $\mathbf{x}$ or in the worst case returns a solution that satisfies the same constraints as the initial solution.

01. $k \leftarrow flag$
02. $i \leftarrow 1$
03. **While** $i \leq n-1$ and $|k| \leq rad$
04. $\quad \mathbf{x}' \leftarrow \mathbf{x} + k\mathbf{b}_i$
05. $\quad$ find the new $V' = V_{\mathcal{S}}(\mathbf{x}')$
06. $\quad$ **If** $V \leq V'$
07. $\quad\quad \mathbf{x} \leftarrow \mathbf{x}'$
08. $\quad\quad k \leftarrow k + flag$
09. $\quad\quad V \leftarrow V'$
10. $\quad$ **else**
11. $\quad\quad i \leftarrow i + 1$
12. $\quad\quad k \leftarrow flag$
13. $\quad$ **end if**
14. **end while**
15. return $\mathbf{x}$, V

Once we have the update function, we apply the pseudocode below.

Branch and bound algorithm for compact knapsack:
INPUT: $L = \{\mathbf{b_j}\}_j, \mathbf{x}, n, \mathcal{S}, rad, K$
OUTPUT: A solution $\mathbf{x} \in \mathcal{S}$ or a better solution which satisfies more constraints or in the worst case returns a solution that satisfies the same constraints as the initial solution.

01. $h = 0$
02. **While** $V = V_{\mathcal{S}}(\mathbf{x}) < n$ and $h < K$
03. $\quad h \leftarrow h + 1$
04. $\quad (\mathbf{x}, V) \leftarrow$UPDATE$(L, \mathbf{x}, n, \mathcal{S}, V, rad, 1)$
05. $\quad$ **If** $V < n$
06. $\quad\quad (\mathbf{x}, V) \leftarrow$UPDATE$(L, \mathbf{x}, n, \mathcal{S}, V, rad, -1)$
07. $\quad$ **end if**
08. **end while**
09. return $\mathbf{x}$

The efficiency and the success rate of the algorithm depends on how many nodes we consider i.e. the value of K, and how we choose the coefficients $\{a_j\}_j$ and the set $\mathcal{A}$.

This algorithm can be easily parallelized. For each value of

$$k \in \{-rad, ..., rad\} - \{0\}$$

(in the UPDATE algorithm) we consider a different node.

Using the previous algorithm we did not get any solution for the case $(n, R) = (60, 100)$ and $\mathbf{x} \in \mathcal{S}_2$ for $(K, rad) = (20, 2^{15})$. We have an advantage, on average 6 bits, in the case $(n, R) = (30, 40)$ for $\mathbf{x} \in \mathcal{S}_2$. Although we did not find any solution we do not consider such values as safe for the hardness of the problem.

6. An Id system based on compact knapsack

In public key Id (Identification) protocols an entity (the prover) holding a secret key wants to prove its identity to an entity (the verifier) holding only the public key. The minimal notion of security in this scheme, is that of a passive attacker, knowing only the public key. This notion, as we shall see below coincides with a property called soundness of the Id-scheme. In this case the adversary is not capable to impersonate the prover, knowing only the public key. In a real world situation, the adversary is allowed to communicate with the prover, hoping to extract some information and then impersonate the prover. The usual model of security that we use is the second one. In this case we say that the Id-scheme is secure in active attacks. See [6] for a recent account in Id-schemes. Most of public key Id-schemes are based on specific problems from number theory such as factorization and discrete logarithm problem. Thus are quite expensive because of the exponentiations they use. That is they are not lightweight. A second possible problem is that they are not quantum resistant, since in the factorization and discrete logarithm problem, Shor's algorithm can be applied, assuming the existence of a (large) quantum computer.

We are interested in three moves Id-schemes. Here Alice (the prover) holds a secret key and sends to Bob (the verifier) a message which we call *commitment*. Bob responds with a random string which we call it *challenge* (or *exam*). Alice provides a *response*. Finally, Bob applies a verification algorithm which has as input, the public key of Alice and the previous conversation, in order to decide if he will accept or reject the id of Alice. The length of the challenge is the security parameter. We shall provide an Id scheme which is not based on discrete logarithm or factorization problem, but on the compact knapsack problem i.e. we provide a proof of knowledge for the compact knapsack problem. Let,

$$\sum_{j=1}^{n} a_j x_j = b, \tag{6.1}$$

with $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in S_{n,R,\ell} = I_{R+\ell}^{n/2} \times I_{R/2+\ell}^{n/2}$, where I_α is the set of integers having $\alpha-$bits (R, ℓ are positive integers and assume for simplicity that R, n are even) and $(a_j)_j$ positive integers with at most $R-$bits. If n is odd we define $S_{n,R,\ell} = I_{R+\ell}^{(n-1)/2} \times I_{R/2+\ell}^{(n+1)/2}$.

We describe our Id system. Alice picks a vector $\mathbf{a} = (a_j)_j$, with entries to be positive integers of at most $R-$ bits as entries and a vector $\mathbf{x} \in S_{n,R,\ell}$, such that $\mathbf{a} \cdot \mathbf{x} = b$. She publishes $(\mathbf{a}, b, R, \ell)$. The private key is $\mathbf{x}$.

The following scheme is repeated $t-$times.
• Alice picks a random vector $\mathbf{k} \in S_{n,R,0}$. Then she computes

$$\mathbf{a} \cdot \mathbf{k} = r$$

and sends it to Bob (*commitment*).
$\diamondsuit$ Bob picks a random integer $e \in \{0, 1\}$ and sends it to Alice (*challenge*).
• Alice computes

$$\mathbf{s} = \mathbf{k} + e\mathbf{x}$$

and sends $\mathbf{s}$ to Bob (*response*).
$\diamondsuit$ Bob verifies the equality $\mathbf{a} \cdot \mathbf{s} = r + eb$ and that $\mathbf{s} \in S_{n,R,0}$ if $e = 0$. Now, if $e = 1$ Alice can choose from the beginning R, ℓ such that $\mathbf{s} \in S_{n,R,\ell}$ with large probability (≈ 1). See corollary 6.5.

Proof of correctness.

$$\mathbf{a} \cdot \mathbf{s} = \mathbf{a} \cdot \mathbf{k} + e\mathbf{a} \cdot \mathbf{x} = r + eb.$$

Also, $\mathbf{s}$ satisfies the constraints of the scheme. □

Remark 6.1. To be precise the previous scheme is a probabilistic Id-scheme, since Bob is convinced with high probability.

The first basic requirement for an Id-scheme is the soundness. Let Eve be an adversary. The scheme is sound if Eve knowing only the public key, can pass the verification test with only negligible probability. The soundness of the scheme depends on the number of iterations t. Assume that, for simplicity $t = 1$. Say that Eve, by tossing up a fair coin, picks the right $e' \in \{0, 1\}$. Then, she computes a random vector $\mathbf{s} \in S_{n,R,0}$ if $e = 0$, else she chooses $\mathbf{s} \in S_{n,R,\ell}$. Then, she sends to Bob the pair $(r = \mathbf{a} \cdot \mathbf{s}, \mathbf{s})$ if $e' = 0$ and $(r = \mathbf{a} \cdot \mathbf{s} - b, \mathbf{s})$ if $e' = 1$. The pair passes the verification test since $\mathbf{a} \cdot \mathbf{s} = r + e'b$. The success rate is $1/2$. In general is 2^{-t}. So for $t = 80$ the success rate is negligible. To prove that the scheme is sound we have to show that this success rate can not be improved unless the compact knapsack problem is easy. So assume that Eve has a Monte Carlo algorithm $\mathcal{A}$ with inputs the public key and a random $\mathbf{e} \in \{0, 1\}^t$, that provides $t-$passing pairs $(r_i, \mathbf{s_i})$ for the verification test. In fact, as we shall see, $\mathcal{A}$ is a parallel Monte Carlo algorithm e.g. [23, chapter 12]. Also, the time complexity for algorithm $\mathcal{A}$ is $|\mathcal{A}|$. Furthermore, suppose that the probability of success is ε. We shall show that using the probabilistic algorithm $\mathcal{A}$ we can find with constant probability, a solution $\mathbf{x}' \in S_{n,R,\ell}$, such that $\mathbf{a} \cdot \mathbf{x}' = b$. This is enough for the adversary to impersonate Alice. Thus, we consider all the class of the solutions $\Sigma_b = \{\mathbf{x} \in S_{n,R,\ell} : \mathbf{a} \cdot \mathbf{x} = b\}$ as *one* solution. Also, assume that it is difficult to find elements in Σ_b knowing only the vector $\mathbf{a}$ and the integer b. We shall now state our Theorem.

Theorem 6.2. *(soundness). Let $\mathcal{A}$ be a probabilistic algorithm having as inputs the public key of the Id-scheme and a random vector $\mathbf{e} \in \{0, 1\}^t$, and output $t-$passing pairs $(\mathbf{r}, (\mathbf{s}_i)_i) = ((r_i)_i, (\mathbf{s}_i)_i)$, with probability $\varepsilon > 2^{-t+2}$. Suppose that, $\ell = \lfloor 0.58 - \log_2(1 - 0.99^{1/n}) \rfloor$. Then, with constant probability and running time $O(|\mathcal{A}|/\varepsilon)$ we can find an element of Σ_b (which is equivalent to knowing the secret key $\mathbf{x}$).*

We need some auxiliary lemmas.

Lemma 6.3. *Let $0 < a < b < c < d$ integers and $N_1 = [a, b] \cap \mathbb{Z},\ N_2 = [c, d] \cap \mathbb{Z}$. Suppose that $b \leq d - c + 1$. Let also x_1, x_2 are chosen uniformly from N_1, N_2, respectively. Then,*

$$Pr(x_2 - x_1 \notin N_2) = \frac{(a+b)}{2(d-c+1)} = Pr(x_1 + x_2 \notin N_2).$$

Proof. We count the pairs $(x_1, x_2) \in N_1 \times N_2$ such that, $x_2 - x_1 < c$. We fix $x_1 = b$. Then, the maximum x_2, such that, $x_2 - b < c$, is $x_2 = c + b - 1 \in N_2$. So we start counting from $c + b - 1$ until $x_2 = c$. Thus for $x_1 = b$, we get $(c + b - 1) - c + 1 = b$ possible x_2's. Similar for $x_1 = b - 1$, we get $b - 1$ possible x_2's and so on. Finally, we get $\frac{(a+b)(b-a+1)}{2}$ pairs $(x_1, x_2) \in N_1 \times N_2$ such that, $x_2 - x_1 < c$. So

$$Pr(x_2 - x_1 \notin N_2) = \frac{(a+b)}{2(d-c+1)}.$$

The number of pairs $(x_1, x_2) \in N_1 \times N_2$ such that, $x_2 + x_1 > d$, is equal with the number of pairs $(x_1, x_2) \in N_1 \times N_2$ such that, $x_2 - x_1 < c$. To see this we repeat the previous arguments, but starting from $x_1 = a$ and $x_2 = d - a + 1 \in N_2$ and continue as previous. The lemma follows. □

Corollary 6.4. *Let $N_1 = I_R,\ N_2 = I_{R+\ell}$ $(\ell > 1)$, and x_1, x_2 are chosen uniformly from N_1, N_2, respectively. Then,*

$$Pr(x_2 - x_1 \notin N_2) = Pr(x_2 + x_1 \notin N_2) \approx \frac{3}{2^{\ell+1}}.$$

Proof. We set $a = 2^{R-1}, b = 2^R - 1, c = 2^{R+\ell-1}, d = 2^{R+\ell} - 1$. Since $b \leq d - c + 1$, the corollary follows from lemma 6.3. □

Remark that if $N_1 = I_{R/2}$ and $N_2 = I_{R/2+\ell}$, the corollary is still valid.

Corollary 6.5. *Let $(\mathbf{k}, \mathbf{x})$ is randomly chosen from $\mathcal{S}_{n,R,0} \times \mathcal{S}_{n,R,\ell}$. Let also $\ell = \lfloor \log_2 3 - 1 - \log_2(1 - p^{1/n}) \rfloor$ and $p \approx 1$, but $p < 1$. It turns out*

$$Pr\big((\mathbf{k}, \mathbf{x}) \xleftarrow{\$} \mathcal{S}_{n,R,0} \times \mathcal{S}_{n,R,\ell} : \mathbf{x} \pm \mathbf{k} \in \mathcal{S}_{n,R,\ell}\big) \approx p.$$

Proof. We set $N_1 = I_R,\ N_2 = I_{R+\ell}$. Then, from corollary 6.4, $Pr(x_j - k_j \in N_2) \approx 1 - \frac{3}{2^{\ell+1}}$. Let $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2})$ where $\mathbf{x_1}$ contains the first half entries of $\mathbf{x}$ and $\mathbf{x_2}$ contains the rest entries. Similar for $\mathbf{k} = (\mathbf{k}_1, \mathbf{k}_2)$. Let also the events

$$\mathbf{A}_1 = \big\{(\mathbf{k}_1, \mathbf{x}_1) \xleftarrow{\$} I_R^{n/2} \times I_{R+\ell}^{n/2} : \mathbf{x}_1 - \mathbf{k}_1 \in I_{R+\ell}^{n/2}\big\},$$

$$\mathbf{A}_2 = \big\{(\mathbf{k}_2, \mathbf{x}_2) \xleftarrow{\$} I_{R/2}^{n/2} \times I_{R/2+\ell}^{n/2} : \mathbf{x}_2 - \mathbf{k}_2 \in I_{R/2+\ell}^{n/2}\big\}.$$

Since $\mathbf{A}_1, \mathbf{A}_2$ are independent, corollary 6.4 provides,

$$Pr(\mathbf{x} - \mathbf{k} \in \mathcal{S}_{n,R,\ell}) = Pr((\mathbf{k}_1, \mathbf{x}_1) \in \mathbf{A}_1 \text{ and } (\mathbf{k}_2, \mathbf{x}_2) \in \mathbf{A}_2) =$$

$$Pr((\mathbf{k}_1, \mathbf{x}_1) \in \mathbf{A}_1) \cdot Pr((\mathbf{k}_2, \mathbf{x}_2) \in \mathbf{A}_2) \approx$$

$$\left(1 - \frac{3}{2^{\ell+1}}\right)^{n/2} \left(1 - \frac{3}{2^{\ell+1}}\right)^{n/2} = \left(1 - \frac{3}{2^{\ell+1}}\right)^{n} \approx p.$$

Substituting $\ell' = \log_2 3 - 1 - \log_2(1 - p^{1/n})$ we get $(1 - \frac{3}{2^{\ell'+1}})^n = p$. Setting $\ell = \lfloor \ell' \rfloor$, the previous equality is no longer true. But for $n \geq 2$ and $p \to 1^-$, then $(1 - \frac{3}{2^{\ell+1}})^n \to 1$. So the result follows. Similar for $Pr(\mathbf{x} + \mathbf{k} \in \mathcal{S}_{n,R,\ell})$. This completes the proof. □

Remark 6.6. In practice for $p \in (0.99, 1)$ and $n \geq 2$, we get $(1 - \frac{3}{2^{\ell+1}})^n \approx p$.

Lemma 6.7. *(special soundness). If we have two queries* $(\mathbf{r}, \mathbf{e}, (\mathbf{s}_i)_i)$, $(\mathbf{r}, \mathbf{e}', (\mathbf{s}'_i)_i)$ $(i = 1, 2, ..., t)$ *and* $\mathbf{e} \neq \mathbf{e}'$, *then for suitable choice of* ℓ *we can efficiently find a solution in* $\Sigma_b = \{\mathbf{x} \in S_{n,R,\ell} : \mathbf{a} \cdot \mathbf{x} = b\}$ *with high probability.*

Proof. Eve has two queries $\mathbf{e} = (e_i)_i, \mathbf{e}' = (e'_i)_i \in \{0,1\}^t$ $(\mathbf{e} \neq \mathbf{e}')$ such that, the corresponding values of $(r_i, \mathbf{s}_i)$,$(r'_i, \mathbf{s}'_i)$ $(i = 1, 2, ..., t)$ give positive result and $r_i = r'_i$ for every i. Thus without loss of generality, there are $e_j = 1$, $e'_j = 0$, for some $j \in \{1, 2, .., t\}$ such that

$$\mathbf{a} \cdot \mathbf{s}_j = r_j + b, \ \mathbf{a} \cdot \mathbf{s}'_j = r_j.$$

Then,

$$\mathbf{a} \cdot (\mathbf{s}_j - \mathbf{s}'_j) = b.$$

We set $\mathbf{S} = (\mathbf{s}_j - \mathbf{s}'_j)$. By corollary 6.5, for $\ell = \lfloor 0.58 - \log_2(1 - 0.99^{1/n}) \rfloor$ with $p = 0.99$, and $\mathbf{s}_j \in \mathcal{S}_{n,R,\ell}$, $\mathbf{s}'_j \in \mathcal{S}_{n,R,0}$, we get $\mathbf{S} \in \mathcal{S}_{n,R,\ell}$ with high probability. So, $\mathbf{S} \in \Sigma_b$, with high probability. □

Remark 6.8. From the previous lemma we see that, if Eve can impersonate Alice (i.e. the system is not passive secure), then she can construct two passing queries $(\mathbf{r}, \mathbf{e}, \mathbf{s}_i)$, $(\mathbf{r}, \mathbf{e}', \mathbf{s}'_i)$. Thus she can compute the private key. Now, if Eve knows the private key, she can easily impersonate Alice. Such systems where impersonation is equivalent with knowing the secret key are called *proof of knowledge* systems.

For the proof of Theorem 6.2 we follow [6].

Proof of the theorem 6.2. Assume that $\mathcal{A}$ is a probabilistic algorithm with input the public key of prover, $pk = (\mathbf{a}, b, R, l)$ and a random $\mathbf{e} = (e_1, ..., e_t) \in \{0,1\}^t$. It works like a black box and depends on an internal state, which is a random string. It gives with probability ε, as output a pair $\mathbf{out} = (\mathbf{r}, (\mathbf{s}_i)_i)$ such that, $(\mathbf{out}, \mathbf{e})$ be a query that passes the verification test. That is $\mathbf{a} \cdot \mathbf{s}_i = r_i + e_i b$, for $i = 1, 2, ..., t$.

We model $\mathcal{A}$ as follows. We fix a matrix H with entries 0 and 1. H has a column for each different vector $\mathbf{e}$, which is the value of the *challenge* in every round of our system. Therefore, it can be from 1 to 2^t, which represents binary vectors between $(0, \ldots, 0, 1)$ and $(1, \ldots, 1, 1)$. For each internal state we get a row of the matrix. The vector $\mathbf{r}$ that the algorithm produces depends on the pk and the choice of the row, i.e. $\mathbf{r} = \mathbf{r}(pk, \text{Internal State})$ and $\mathbf{s}$ depends on the row and the column, that is $(\mathbf{s}_i)_i = (\mathbf{s}_i)_i(pk, \text{Internal State}, \mathbf{e})$.

With input $(pk, \text{Internal State}, \mathbf{e})$, the output of algorithm $\mathcal{A}$ is a passing pair $\big((\mathbf{r}, (\mathbf{s}_i)_i); \mathbf{e}\big)$. If we hit 0, then we do not get a passing pair for the specific input $(pk, \text{Internal State}, \mathbf{e})$. Our goal is to found a row with two 1's at least. If we have this row, we show using Lemma 6.7 how to reveal an element of the set Σ_b. Furthermore, we shall show that this can be done with constant probability. First, we examine the distribution of 1's in the rows. A row is called *heavy* if it contains a fraction of at least $\frac{\varepsilon}{2}$, 1's. We also define:
h: the number of H's entries, so $\varepsilon \cdot h$ is the number of 1's in H
h_1: the number of H's entries in non heavy rows, so the number of 1's in these rows is less than $h_1\varepsilon/2$. Therefore, heavy rows have at least h_2 1's, which are:

$$h_2 > h\varepsilon - h_1\varepsilon/2 > h\varepsilon - h\varepsilon/2 = h\varepsilon/2.$$

That is at least half of the rows are heavy. Furthermore, from the assumption that $\varepsilon \geq 2^{-t+2}$ and the fact that $h > 2^t$, it follows that a heavy row contains at least two 1's.

After $1/\varepsilon$ tries we can find the first 1, if we probe H randomly (i.e. if we choose a random internal state and a random $\mathbf{e}$). The probability of that is more than $1/2$. If this 1 lies in a heavy row, then we can find a second 1 in the same row with probability $\frac{\frac{\varepsilon}{2}2^t-1}{2^t}$. As a result, we need $\frac{2^t}{\frac{\varepsilon}{2}2^t-1}$ tries to succeed. Since $\frac{2^t}{\frac{\varepsilon}{2}2^t-1} < \frac{2^t}{\frac{\varepsilon}{2}2^{t-1}} = 4/\varepsilon$, this means that with less than $\frac{4}{\varepsilon}$ tries we get the second 1, with probability $1/2$. Otherwise, if the first hit is in a non heavy row, we could spend too much time searching for a second 1. To avoid this, we apply an algorithm, say $\mathcal{A}_1$, which stops the procedure after a specific number of tries. The algorithm consists of two steps, that run in parallel:
St1: Probe random in the same row until a second 1 is found.
St2: Repeatedly, probing a random entry in H and choosing a random number among $1, 2, \ldots, d$ (d will be chosen later). This step stops, if the entry is 1 and the number d is 1.

Algorithm $\mathcal{A}_1$ runs in expected time $\mathrm{O}(|\mathcal{A}|/\varepsilon)$. But, we want St1 to stop first (with high probability), in order to have two 1's in one row. The probability of St2 finishes after k attempts is $\varepsilon/d(1-\varepsilon/d)^{k-1}$. Using the assumption for ε as before, we get that $(1-\varepsilon/d)^{k-1} \leq 1$ and that means the probability of finishing after k or fewer attempts is at most $k\varepsilon/d$. We consider that $k = \lfloor d/(2\varepsilon) \rfloor$, in order the success probability for St2 be $1/2$. Furthermore, if we choose $d = 16$, we have that St2 finishes after more than $8/\varepsilon = k$ tries with probability at least $1/2$. As a result, St1 finishes before St2 with probability greater than $\frac{1}{2}\frac{1}{2} = \frac{1}{4}$. So, if this occurred, then with probability at least $1/8$ we shall get the second 1.

$$\mathcal{A} \xrightarrow[\Pr>1/2]{\mathrm{O}(|\mathcal{A}|/\varepsilon)} \text{1st hit} \longrightarrow \mathcal{A}_1 \xrightarrow[\Pr>1/8]{\mathrm{O}(|\mathcal{A}|/\varepsilon)} \text{2nd hit}$$

So, we shall find two 1's in a heavy row after $12/\varepsilon$ tries and with constant probability at least $\frac{1}{2} \cdot \frac{1}{8} = \frac{1}{16}$. That is, the algorithm runs in $O(|\mathcal{A}|/\varepsilon)$ and succeeds to find two 1's in the same row with constant probability $> 1/16$. Since now, we have two 1's (in the same row), we get two queries $(\mathbf{r}, (\mathbf{s}_i)_i, \mathbf{e}), (\mathbf{r}, (\mathbf{s}'_i)_i, \mathbf{e}')$, with $\mathbf{e} \neq \mathbf{e}'$. Applying Lemma 6.7, the Theorem follows. □

Since we assumed that finding elements of Σ_b is difficult, we can not improve the success probability to be $> 2^{-t+2}$.

6.1. **The parameters.** We have to choose R, ℓ in order to be difficult to find even one $n-$tuple $(x_1, ..., x_n)$ of equation (6.1), if Eve picks randomly a solution from $\mathcal{S}_{n,R,\ell}$. Also, to consider secure parameters we have to consider the attack given in section 5.1. Let Pr be the probability to find a solution $\mathbf{x}$ of equation (6.1) with $\mathbf{x} \in \mathcal{S}_{n,R,\ell}$, if we pick $\mathbf{x}$ randomly from the set $\mathcal{S}_{n,R,\ell}$. Also, we set Pr' the same probability if we choose $\mathbf{x}$ from the set of $\mathcal{S}_{n,R,0}$. Let N_b be the number of solutions in $\mathcal{B}_n = \mathcal{S}_{n,R,\ell}$ of the Diophantine equation $\sum_{i=1}^{n} a_i x_i = b$. We remark that, there is a hyperplane that meets $n-$ vertices of the box $\mathcal{B}_n$ (e.g. a face of $\mathcal{B}_n$) and contains the maximum number of integer points (in $\mathcal{B}_n$) of all hyperplanes that passes from

at least one point of $\mathcal{B}_n$. Let $\mathcal{F}_{n-1}$ be that hyperplane. So, $N_b \leq |\mathcal{F}_{n-1}|$, thus

$$Pr \leq \frac{|\mathcal{F}_{n-1}|}{|\mathcal{S}_{n,R,\ell}|} = \frac{|I^{n/2}_{R+\ell}| \cdot |I^{n/2-1}_{R/2+\ell}|}{|I^{n/2}_{R+\ell}| \cdot |I^{n/2}_{R/2+\ell}|} =$$

$$= \frac{1}{2^{R/2+\ell}-1}.$$

Similarly,

$$Pr' \leq \frac{1}{2^{R/2}-1}.$$

We want both the probabilities to be negligible. It is enough to choose, $R/2 \geq 80$. Then, the probabilities $Pr,\ Pr'$ are $\leq 2^{-80}$. So, if we pick $R \geq 160$, then the probabilities $Pr,\ Pr'$ are negligible. Furthermore, R must be large enough, since choosing two times the same ephemeral key, will reveal the secret key with probability $1/2$.

For instance, as a minimal choice we suggest $n = 70,\ R = 192,\ p = 0.99$ so $\ell = 12$, and $\mathbf{a} \overset{\$}{\leftarrow} I^n_{R/8}$, then we get a public key $(\mathbf{a}, b, R, \ell)$ of length

$$|\mathbf{a}| + |b| + |R| + |\ell| \approx \left(\frac{n}{2}\frac{R}{8} + \frac{n}{2}\frac{R}{8}\right) + |b| + |R| + |\ell| = 1927 \text{ bits}.$$

Note that the length $|b| \approx 9R/8 + \ell + |n| = 235$ bits (here with $|.|$ we denote the binary length).

The system has relatively small public keys and is very fast since it uses only additions and multiplications. Finally, the previous selection of parameters shall resist the attack given in subsection 5.1. Since may exist (or found) other better attacks, someone has to adjust the parameters considering the new attack. In general, the hardness of the compact knapsack among others, depends on the topology of the set of solutions. So someone instead of $\mathcal{S}_{n,R,\ell}$ can use another set $\mathcal{S}$ which minimize the length of the public key and achieve the same security. The only thing that needs some care is to check that lemma 6.2 is still valid for the new set $\mathcal{S}$.

On the whole, we need three sets, the parameter set P where we choose $\mathbf{a}$ and two sets $\mathcal{S}_1, \mathcal{S}_2$, with $\mathcal{S}_2 \subset \mathcal{S}_1$, where we randomly choose the private key $\mathbf{x}$ from $\mathcal{S}_1$ and the ephemeral key $\mathbf{k}$ from $\mathcal{S}_2$. The sets $\mathcal{S}_1, \mathcal{S}_2$ are chosen such that $\mathbf{x} - \mathbf{k} \in \mathcal{S}_1$ with large probability. Then, we can prove the soundness of the id-scheme. Furthermore, we want the compact knapsack problem with solution sets $\mathcal{S}_1, \mathcal{S}_2$ and parameter set P to resist the CVP-attack of subsection 5.1.

Finally, in [26] an id-scheme was presented based on Short Integers Problem (SIS). Some basic differences between our scheme and [26] are the following:

• In [26] the system is over $\mathbf{Z}_p$ and ours is over the integers.

• System [26] is provable secure under active attacks. He applied theorem 2 of [26]. In this specific theorem, we can not consider one equation (as in our system). Because in this case the reduction of the theorem does not work (i.e. SIS can not be reduced to a hard lattice problem).

• Our id-scheme security is based on compact knapsack (the solutions here may be large and not necessarily small as in SIS). Finally, the two problems SIS and compact knapsack are different in the sense that the latter is NP-complete.

7. Conclusions and Future Work

In this work we addressed the knapsack problem and some of its variants. We optimized the algorithm presented in [30] using some heuristics that work in practice. We provided two variants, which in average are better than the original method. One of them can be easily parallelized. Furthermore, we considered the compact knapsack problem and we apply a CVP-reduction and we combine it with a branch and bound algorithm. We used the results of our algorithm to provide some minimal security conditions of the parameters for our Id-scheme which base its security on this problem. Unfortunately, we managed to prove its security only under passive attacks. One other disadvantage is the relatively large information complexity. For instance, if $(n, R, \ell) = (70, 192, 12)$, then the prover sends ≈ 145 KBytes using 80 rounds, to the verifier. On the other hand, this scheme is lightweight in the sense that does not use exponentiations. Also, it is potentially quantum resistant since does not base its security in factorization or the discrete logarithm problem. We did not address here the problem of choosing the right parameters, but we provide some minimal requirements for them. Furthermore, other choices of the parameters probable lead to smaller public keys and information complexity. This last remark, can be used to implement the system in constrained devices e.g. smart cards.

The next step of this work is to consider the transformation of the three-move Id -scheme to a digital signature using the Fiat-Shamir transformation. This can be done if we replace the role of Bob with a random hash function. Then, many issues need to be carefully studied. For instance, the security under random oracle model. Also, the selection of the parameters.

One other possible extension as far as the experimental part of this work, is to consider more advanced type of reductions instead of classical BKZ with pruning. For instance in [15] the authors suggested a new type of pruning, they called it extreme pruning, which behaves better than the classical linear pruning. So one may try this new improved method to solve hard knapsack problems using the method of Schnorr-Shevchenko.

Acknowledgment. The authors are indebted to the anonymous referees for their helpful suggestions.

References

[1] K. Aardal, C. Hurkens, A. Lenstra, Solving a linear Diophantine equation with lower and upper bounds on the variables. Integer programming and combinatorial optimization LNCS **1412**, p.229–242, 1998.

[2] K. Aardal, F. Eisenbrand, The LLL Algorithm and Integer Programming, The LLL Algorithm (Survey and Applications) p.293–314, Springer, 2010.

[3] M. Ajtai, The shortest vector problem in L_2 is NP-hard for randomized reduction, Proc. 30th ACM Symposium on Theory of Computing (STOC), 1998. Combinatorica, **6** p.1-13, 1986.

[4] A. Becker, J.-S Coron and A. Joux, Improved generic algorithm for hard knapsacks. Eurocrypt 2011, LNCS **6632**, p. 364–385, 2011.

[5] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. Computational Complexity, **2**, p.111–128, 1992.

[6] I. Damgård, On Sigma-protocols, `http://www.cs.au.dk/~ivan/Sigma.pdf`, Cource material (2010).

[7] K. A. Draziotis, Balanced Integer solutions of linear equations, AMIMS 2013(Greece), Optimization and its Applications (SOIA), vol.**91**, p. 173–188, Springer 2014.

[8] K.A. Draziotis and A. Papadopoulou, `https://github.com/drazioti/python_scripts/tree/master/paper_knapsack`.

[9] D. Micciancio,, S. Goldwasser, Complexity of Lattice Problems A cryptographic Perspective, Springer 2002.
[10] R. Impagliazzo, M. Naor, Efficient Cryptographic schemes Provably as secure as subset sum. Journal of Cryptology Vol.**9(4)**, 1996, p. 199–216.
[11] The FPLLL development team, fplll, Available at `https://github.com/fplll/fplll`.
[12] The FPyLLL development team, fpylll : A python interface for fplll, Available at `https://github.com/fplll/fpylll`.
[13] A. M. Freize, On the Lagarias-Odlyzko algorithm for the subset sum problem. SIAM J.Comput. **15(2)**, p.536–539, 1986.
[14] Nicolas Gama and Phong Q. Nguyen, Predicting Lattice Reduction, LNCS **4965**, p.31–51, Springer 2008.
[15] Nicolas Gama, Phong Q. Nguyen and Oded Regev, Lattice Enumeration using Extreme Pruning, Eurocrypt 2010.
[16] S. Galbraith, Mathematics of Public Key Cryptography, Cambridge University Press, 2012.
[17] M. R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.
[18] Guillaume Hanrot, Xavier Pujol and Damien Stehlé. Analyzing Blockwise Lattice Algorithms Using Dynamical Systems, p.447–464, LNCS **6841**, Springer 2011.
[19] Nick Howgrave-Graham, Antoine Joux, New generic algorithms for hard knapsacks. Eurocrypt 2010, LNCS **6110**, p.235–256, 2010.
[20] J. C. Lagarias, and A. M. Odlyzko, Solving low-density subset sum problems. J. Assoc. Comput. Mach. **32**, p. 229–246, 1985.
[21] M. S. Lee, Improved cryptanalysis of a knapsack-based probabilistic encryption scheme, Information Sciences **222** (10), 2013.
[22] A. K. Lenstra, H. W. Lenstra and L. Lovász, Factoring polynomials with rational coefficients. Math. Ann., **261(4)**, p. 515–534, 1982.
[23] R. Motwani and P. Raghavan, Randomized Algorithms. Cambridge University Press (1995), (Chapter 12.)
[24] D. Micciancio, O. Regev, Lattice-Based Cryptography, In Post Quantum Cryptography, p. 147–191, Springer 2009.
[25] A. May, M.Herrmann, Solving linear equations modulo divisors : on factoring given any bits. In Advances in Cryptology (Asiacrypt 2008).
[26] Vadim Lyubashevsky, Lattice-Based Identification Schemes Secure Under Active Attacks, PKC 2008.
[27] Stanislaw Radziszowski, Donald Kreher, Solving subset sum problems with the L^3 algorithm, Journal of Combinatorial Mathematics and Combinatorial Computing **3** p.49–63, 1988.
[28] Claus-Peter Schnorr. Efficient Signature Generation by Smart Cards, J.Cryptology, vol.**4** (1991).
[29] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. Mathematical programming, **66**, pp.181–199, 1994.
[30] Claus-Peter Schnorr and Taras Shevchenko, Solving Subset Sum Problems of Density close to 1 by "randomized" BKZ-reduction. Cryptology ePrint Archive: Report **2012/620**.
[31] Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. SIAM J. Comput., **10(3)**, p.456–464, 1981.
[32] The Sage Developers, SageMath, the Sage Mathematics Software System (Ver. 6.9) `http://www.sagemath.org`.
[33] B. Wang, Q. Wu and Y. Hu, A knapsack-based probabilistic encryption scheme, Information Science bf 177 (19), 2007.
[34] A. M. Youssef, Cryptanalysis of a knapsack-based probabilistic encryption scheme, Information Sciences **179 (18)**, 2009.