Improved Generic Algorithms for Hard Knapsacks

Anja Becker[⋆,1], Jean-Sébastien Coron[3], and Antoine Joux[1,2]

[1] University of Versailles Saint-Quentin-en-Yvelines
[2] DGA
[3] University of Luxembourg

Abstract. At Eurocrypt 2010, Howgrave-Graham and Joux described an algorithm for solving hard knapsacks of density close to 1 in time $\tilde{\mathcal{O}}(2^{0.337n})$ and memory $\tilde{\mathcal{O}}(2^{0.256n})$, thereby improving a 30-year old algorithm by Shamir and Schroeppel. In this paper we extend the Howgrave-Graham–Joux technique to get an algorithm with running time down to $\tilde{\mathcal{O}}(2^{0.291n})$. An implementation shows the practicability of the technique. Another challenge is to reduce the memory requirement. We describe a constant memory algorithm based on cycle finding with running time $\tilde{\mathcal{O}}(2^{0.72n})$; we also show a time-memory tradeoff.

1 Introduction

The Knapsack Problem. Given a list of n positive integers $(a_1, a_2, \ldots, a_n)$ and another positive integer S such that:

$$S = \sum_{i=1}^{n} \epsilon_i \cdot a_i \ \ , \tag{1}$$

where $\epsilon_i \in \{0,1\}$, the knapsack problem consists in recovering the coefficients ϵ_i. The vector $\epsilon = (\epsilon_1, .., \epsilon_n)$ is called the solution of the knapsack problem. It is well known that the decisional version of the knapsack problem is NP-complete [4].

The first cryptosystem based on the knapsack problem was introduced by Merkle and Hellmann [10] in 1978, and subsequently broken by Shamir [14] using lattice reduction. For random knapsack problems the Lagarias-Odlyzko attack [7] can solve knapsacks with density $d < 0.64$, given an oracle solving the shortest vector problem (SVP) in lattices; the density of a knapsack is defined as:

$$d := \tfrac{n}{\log_2 \max_i a_i} \ \ .$$

The Lagarias-Odlyzko attack was further improved by Coster *et al.* [3] to knapsack densities up to $d < 0.94$. Since solving SVP is known to be NP-hard [1], in practice, the shortest vector oracle is replaced by a lattice reduction algorithm such as LLL [8] or BKZ [12].

⋆ The first author was mainly funded by a scholarship of the Gottlieb Daimler- und Karl Benz-Stiftung.

The Schroeppel-Shamir Algorithm. For a knapsack of density close to 1 lattice reduction algorithms do not seem to apply. Until 2009, the best algorithm for such hard knapsacks was due to Schroeppel and Shamir [13] with time complexity $\tilde{\mathcal{O}}(2^{n/2})$ and memory $\tilde{\mathcal{O}}(2^{n/4})$. This is the same running time as the straightforward meet-in-the-middle algorithm but with a lower memory requirement of $\tilde{\mathcal{O}}(2^{n/4})$ instead of $\tilde{\mathcal{O}}(2^{n/2})$. A drawback is that the Schroeppel-Shamir algorithm requires sophisticated data structure such as balanced trees which can be difficult to implement in practice. A simpler but heuristic variant of Schroeppel-Shamir was described in [5] with the same time and memory complexity; we recall this variant in Sect. 2.1. We also recall how to solve *unbalanced* knapsack problems, where the Hamming weight of the coefficient vector $\epsilon = (\epsilon_1, \dots, \epsilon_n)$ can be much smaller than n.

The Howgrave-Graham–Joux Algorithm. At Eurocrypt 2010, Howgrave-Graham and Joux introduced a more efficient algorithm [5] for hard knapsacks. While in Schroeppel-Shamir's algorithm the knapsack instance is divided into two halves with no overlap, the new algorithm allows for overlaps, which induces more degrees of freedom. This enables to reduce the running time down to $\tilde{\mathcal{O}}(2^{0.337n})$ while keeping the memory requirement reasonably low at $\tilde{\mathcal{O}}(2^{0.256n})$. We recall the Howgrave-Graham–Joux algorithm in Sect. 2.2.

Our Contributions. The main contribution of our paper is to extend the Howgrave-Graham–Joux technique to get a new algorithm with running time down to $\tilde{\mathcal{O}}(2^{0.291n})$. The knapsack instance is divided in two halves with possible overlap, as in the Howgrave-Graham–Joux algorithm, but the set of possible coefficients is extended from $\{0, 1\}$ to $\{-1, 0, +1\}$. This means that a coefficient $\epsilon_i^{(1)} = -1$ in the first half can be compensated with a coefficient $\epsilon_i^{(2)} = +1$ in the second half, the resulting coefficient $\epsilon_i = 0$ of the golden solution being $\epsilon_i = \epsilon_i^{(1)} + \epsilon_i^{(2)} = (-1) + (+1) = 0$. Adding (a few) -1 coefficients brings an additional degree of freedom that enables to again decrease the running time; we describe our new algorithm in Sect. 3. We show the practicality of the technique with an implementation.

Another challenge in solving knapsack problems is to reduce the memory requirement. We first describe a simple constant memory algorithm based on cycle finding with running time $\tilde{\mathcal{O}}(2^{0.75n})$. We show how to improve this algorithm down to $\tilde{\mathcal{O}}(2^{0.72n})$ running time still requiring constant memory, by using the Howgrave-Graham–Joux technique. Eventually, we present a time-memory tradeoff for the Schroeppel-Shamir algorithm downto $\tilde{\mathcal{O}}(2^{n/16})$ memory.

2 Existing Algorithms

2.1 The Schroeppel-Shamir Algorithm

We present the Schroeppel-Shamir algorithm [13] under the simpler heuristic variant described in [5]. We consider a knapsack as in (1) and for simplicity we assume that n is a multiple of 4. We write the knapsack sum S as:

$$S = \sigma_1 + \sigma_2 + \sigma_3 + \sigma_4$$

where each σ_i is a knapsack of $n/4$ elements, that is,

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

We guess a middle value σ_M of $n/4$ bits which leads to the equations:

$$\sigma_1 + \sigma_2 = \sigma_M \mod 2^{n/4} \text{ and } \sigma_3 + \sigma_4 = S - \sigma_M \mod 2^{n/4} \ .$$

We solve the two equations separately and merge the result. More precisely, we first construct a sorted list $\{\sigma_2\}$ of all $2^{n/4}$ possible values for σ_2. Then for each possible σ_1, we use the sorted list $\{\sigma_2\}$ to find all σ_2 such that $\sigma_1 + \sigma_2 = \sigma_M \mod 2^{n/4}$. This gives a list $\{\sigma_{12}\}$ of knapsack values $\sigma_{12} = \sigma_1 + \sigma_2$ such that $\sigma_{12} = \sigma_M \mod 2^{n/4}$; the size of the list $\{\sigma_{12}\}$ is heuristically $\tilde{\mathcal{O}}(2^{n/4})$ and it can be built in time $\tilde{\mathcal{O}}(2^{n/4})$. We build the list $\{\sigma_{34}\}$ of knapsack values $\sigma_{34} = \sigma_3 + \sigma_4$ such that $\sigma_{34} = S - \sigma_M \mod 2^{n/4}$ in an analogue way. Eventually, we find a collision between the two lists $\{\sigma_{12}\}$ and $\{S - \sigma_{34}\}$ of two elements σ_{12} and σ_{34}, respectively. For the right guess of σ_M we have found elements such that $\sigma_{12} + \sigma_{34} = S$, thereby solving the knapsack problem.

The time required to build the two lists $\{\sigma_{12}\}$ and $\{\sigma_{34}\}$ is $\tilde{\mathcal{O}}(2^{n/4})$. Then by sorting those two lists the collision can be found in time $\tilde{\mathcal{O}}(2^{n/4})$. Since we have to guess σ_M which is a $n/4$-bit value, the total running time is $\tilde{\mathcal{O}}(2^{n/2})$ and the required memory is $\tilde{\mathcal{O}}(2^{n/4})$.

Unbalanced Case. We say that a knapsack is unbalanced when the Hamming weight of the coefficient vector $\epsilon = (\epsilon_1, \ldots, \epsilon_n)$ is known and equal to ℓ where ℓ significantly differs from $n/2$. As shown in [5], one can adapt the previous algorithm as follows: instead of taking all possible knapsacks of $n/4$ elements we only consider knapsacks of Hamming weight exactly $\ell/4$ (assuming that ℓ is divisible by 4). Note that if the correct solution is not perfectly balanced between the four quarters, then such solution will be missed. This problem is easily solved by permuting the order of the elements in the knapsacks until the Hamming weight of each quarter is equal to $\ell/4$. As explained in [5], the expected number of required repetitions is polynomial in n. Thus, this change does not modify the value of the exponent in the running time.

For $\ell = \tau \cdot n$ the size of the lists $\{\sigma_2\}$ and $\{\sigma_4\}$ becomes $\binom{n/4}{\ell/4} \approx 2^{h(\tau)n/4}$ where:

$$h(x) := -x \cdot \log_2 x - (1-x) \cdot \log_2 (1-x) \ .$$

Again, we can guess a middle value σ_M modulo $2^{h(\tau)n/4}$; as previously the two lists $\{\sigma_{12}\}$ and $\{\sigma_{34}\}$ can be built in time $\tilde{\mathcal{O}}(2^{h(\tau)n/4})$ and a collision is found in time $\tilde{\mathcal{O}}(2^{h(\tau)n/4})$. Therefore, the total time complexity is $\tilde{\mathcal{O}}(2^{h(\tau)n/2})$ and the memory complexity is $\tilde{\mathcal{O}}(2^{h(\tau)n/4})$.

2.2 The Howgrave-Graham–Joux Algorithm

We consider the knapsack (1). For simplicity we assume again that n is a multiple of four and additionally that the Hamming weight of the coefficients ϵ_i is equal to $n/2$. To find a solution $x \in \{0,1\}^n$, the basic idea of Howgrave-Graham and Joux [5] is to split the knapsack into two subknapsacks of size n and of Hamming weight $n/4$. In other words, we write S as the sum $\sigma_1 + \sigma_2$ of two subknapsacks with Hamming weight $n/4$ chosen among the n knapsack elements,

$$\underbrace{\sum_{i=1}^{n} a_i y_i}_{\sigma_1} + \underbrace{\sum_{i=1}^{n} a_i z_i}_{\sigma_2} = S \tag{3}$$

where $y_i, z_i \in \{0,1\}$. Clearly, the combination of two solutions $y \in \{0,1\}^n$ and $z \in \{0,1\}^n$ gives a solution to the original knapsack when the two solutions do not overlap. In other words, we represent any x_i by a binary tuple (y_i, z_i), replacing 0 by $(0,0)$ and 1 by $(1,0)$ or $(0,1)$, respectively. As a consequence, a single solution of the original knapsack problem decomposes into many different representations. This is used to reduce the overall running time as described in the following. We choose a modulus M, a random element $R \in \mathbb{Z}_M$ and we only consider decompositions such that:

$$\sigma_1 = \sum_{i=1}^{n} a_i y_i \equiv R \pmod{M} \text{ and } \sigma_2 = \sum_{i=1}^{n} a_i z_i \equiv S - R \pmod{M} .$$

Since both σ_1 and σ_2 are knapsacks of Hamming weight $n/4$ over n elements, the expected number of solutions to each of these two modular subknapsacks is

$$L = \frac{\binom{n}{n/4}}{M} .$$

Assuming that the lists of solutions of the two subknapsacks can be obtained very efficiently (in time $\tilde{\mathcal{O}}(L)$), it remains to paste the partial solutions together to obtain a solution to the original knapsack. We therefore search a collision between the values σ_1 and $S - \sigma_2$, for all y and z in the two lists of solutions. Since the expected number of such collisions is small, this can be done in $\tilde{\mathcal{O}}(L)$. To minimize the overall running time, M is chosen to be as large as possible. More precisely, one chooses M as a number close to the number of decompositions of the original solution into two solutions of the two subknapsacks, i.e. $M \approx 2^{n/2}$. Under these assumptions, the running time would be reduced down to $\tilde{\mathcal{O}}(2^{h(1/4)n}/2^{n/2}) = \tilde{\mathcal{O}}(2^{0.3113})$.

However, there are several technical difficulties with this approach. First, there is an exponentially small number of bad weights $(a_1, .., a_n)$ where the algorithm fails. Second, the assumption that the list of solutions of each subknapsack can be obtained in time $\tilde{\mathcal{O}}(L)$ is quite strong and difficult to achieve. [5] describes a heuristic algorithm, supported by an implementation, and claims that it achieves the $\tilde{\mathcal{O}}(2^{0.3113n})$ running time. However, May and Meurer recently discovered a mistake in the analysis of this algorithm [9]; they showed that the asymptotic running time of the Howgrave-Graham–Joux algorithm is actually $\tilde{\mathcal{O}}(2^{0.337n})$; see [2] for more details.

3 New Algorithm with Better Time Complexity

3.1 Theoretical Improvement

Our basic idea is to enhance the algorithm of [5] by allowing more representations of the solution of the initial knapsack. Instead of decomposing the original solution into two binary coefficient vectors of weight $n/4$, we consider decompositions that contain 0s, 1s and -1s. More precisely, we choose a parameter α and search for decompositions containing $(1/4+\alpha)n$ 1s and αn -1s. Put differently, we split the 1s of the original solution into pairs $(0,1)$ or $(1,0)$ as before and the 0s into pairs $(0,0)$, $(1,-1)$ or $(-1,1)$. The number of such decompositions is

$$N_D = \binom{n/2}{n/4}\binom{n/2}{\alpha n, \alpha n, (1/2-2\alpha)n} \ .$$

As in Sect. 2.2, we choose a modulus $M \approx N_D$, a random value R modulo M and search for solutions of the two subknapsacks

$$\sigma_1 = \sum_{i=1}^{n} a_i y_i \equiv R \pmod{M} \text{ and } \sigma_2 = \sum_{i=1}^{n} a_i z_i \equiv S - R \pmod{M} \ ,$$

where y and z contain $(1/4+\alpha)n$ 1s and αn -1s each. The expected number of solutions to each of these new modular subknapsacks is

$$L = \frac{\binom{n}{(1/4+\alpha)n, \alpha n, (3/4-2\alpha)n}}{M} \ .$$

Using:

$$\binom{n}{xn, yn, (1-x-y)n} = \tilde{\mathcal{O}}(2^{g(x,y)n})$$

where:

$$g(x,y) := -x\log_2 x - y\log_2 y - (1-x-y)\log_2(1-x-y)$$

we obtain:

$$\log_2 L \approx n \cdot \left(g(1/4+\alpha, \alpha) - \frac{1}{2} - \frac{g(2\alpha, 2\alpha)}{2} \right) \ .$$

Assuming that creating the lists and searching for collisions can be done in time $\tilde{\mathcal{O}}(L)$ and minimizing on α, we obtain a time complexity $\tilde{\mathcal{O}}(L) = \tilde{\mathcal{O}}(2^{0.151\,n})$ for $\alpha \approx 0.103$.

This analysis shows that adding more representations of the original solution has the potential to give better algorithms. However, there are many obstacles to achieve such a good algorithm. A first obstacle is that the size of the modulus M should never be larger than the size of the knapsack elements. Indeed, we want the knapsack after reduction modulo M to behave like a random knapsack, which is not the case if M is larger than the original knapsack elements. Thus, we want to ensure $M < 2^n$. Optimizing for α under this condition, we get $\alpha = 0.05677$ and $L \approx 2^{0.173\,n}$.

3.2 The Basic Building Block

Before describing our algorithm, we recall a classical basic building block that we extensively use. This building block performs the following task: given two lists of numbers $\mathbb{L}_a$ and $\mathbb{L}_b$ of respective sizes $|\mathbb{L}_a|$ and $|\mathbb{L}_b|$, together with two integers M and R, the algorithm computes the list $\mathbb{L}_R$ such that:

$$\mathbb{L}_R = \{x + y \mid x \in \mathbb{L}_a, y \in \mathbb{L}_b \text{ s.t. } x + y \equiv R \pmod{M}\} \ .$$

To solve this problem, we use a classical algorithm [16] whose description is given in pseudo-code by Algorithm 1.

Algorithm 1. Compute list $\mathbb{L}_R$

```
Sort the lists $\mathbb{L}_a$ and $\mathbb{L}_b$ (by increasing order of the values modulo $M$);
Let Target $\leftarrow R$;
Let $i \leftarrow 0$ and $j \leftarrow |\mathbb{L}_b| - 1$;
while $i < |\mathbb{L}_a|$ and $j \geq 0$ do
    Let Sum $\leftarrow (\mathbb{L}_a[i] \pmod{M}) + (\mathbb{L}_b[j] \pmod{M})$;
    if Sum $<$ Target then Increment $i$;
    if Sum $>$ Target then Decrement $j$;
    if Sum $=$ Target then
        Let $i_0, i_1 \leftarrow i$;
        while $i_1 < |\mathbb{L}_a|$ and $\mathbb{L}_a[i_1] \equiv \mathbb{L}_a[i_0] \pmod{M}$ do Increment $i_1$;
        Let $j_0, j_1 \leftarrow j$;
        while $j_1 \geq 0$ and $\mathbb{L}_b[j_1] \equiv \mathbb{L}_b[j_0] \pmod{M}$ do Decrement $j_1$;
        for $i \leftarrow i_0$ to $i_1 - 1$ do
            for $j \leftarrow j_1 + 1$ to $j_0$ do Append $\mathbb{L}_a[i] + \mathbb{L}_b[j]$ to $\mathbb{L}_R$
        end
        Let $i \leftarrow i_1$ and $j \leftarrow j_1$;
    end
end
Let Target $\leftarrow R + M$;
Let $i \leftarrow 0$ and $j \leftarrow |\mathbb{L}_b| - 1$;
Repeat the above loop with the new target;
```

The complexity of Algorithm 1 is $\tilde{\mathcal{O}}(\max(|\mathbb{L}_a|, |\mathbb{L}_b|, |\mathbb{L}_R|))$. Moreover, assuming that the values of the initial lists modulo M are randomly distributed, the expected size of $\mathbb{L}_R$ is $|\mathbb{L}_a| \cdot |\mathbb{L}_b|/M$. However, this cannot be guaranteed in general.

Using a slight variation of Algorithm 1, it is also possible given $\mathbb{L}_a$ and $\mathbb{L}_b$ together with a target integer R to construct the set:

$$\mathbb{L}_R = \{x + y \mid x \in \mathbb{L}_a, y \in \mathbb{L}_b \text{ s.t. } x + y = R\} \ .$$

The only differences are that we sort the lists by value (not by modular values) and then run the loop with a single target value R (instead of 2).

3.3 Devising a Concrete Algorithm

In order to achieve a concrete algorithm along the lines of the theoretical analysis from Sect. 3.1, we must be able to solve the subknapsacks that arise after decomposing the original knapsack problem in a reasonably efficient manner. The difficulty here is that a direct use of an adapted Schroeppel-Shamir algorithm is too costly.

Instead, we use the idea of decomposing a knapsack into two subknapsacks several times. More precisely, we introduce three levels of decomposition; see Fig. 1 for an illustration. The first decomposition follows the method described in Sect. 3.1, with a different (smaller) choice for the value α denoting the proportion of -1s added on each side. At the second or middle level, we decompose each subknapsack from the first level into two. We also add some new -1s in the decompositions. The number of additional -1s for each of the four subknapsacks at the middle level is controlled by a new parameter β. In the last level, we finally decompose into a total of eight different subknapsacks. At this level, we use a parameter γ to denote the proportion of extra -1s in the subknapsacks.

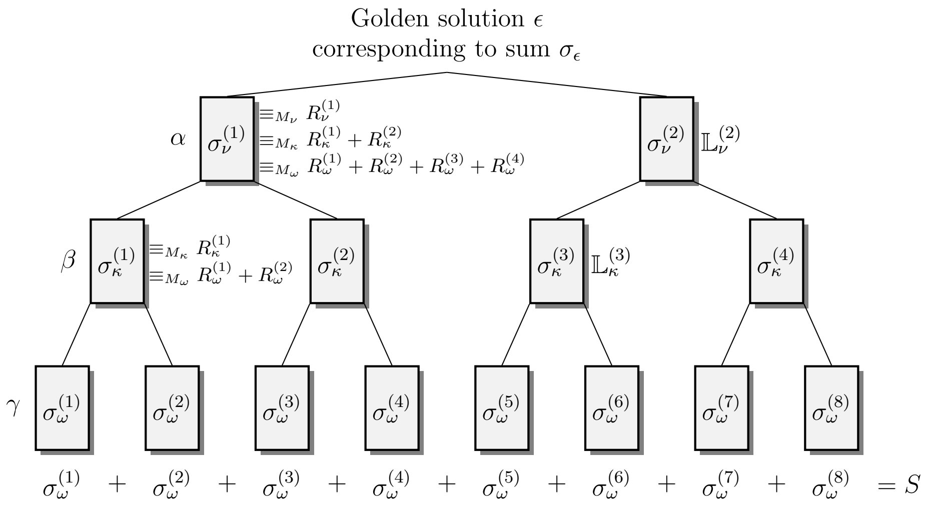
Golden solution ϵ corresponding to sum σ_ϵ
α
$\sigma_\nu^{(1)}$
$\equiv_{M_\nu} R_\nu^{(1)}$
$\equiv_{M_\kappa} R_\kappa^{(1)} + R_\kappa^{(2)}$
$\equiv_{M_\omega} R_\omega^{(1)} + R_\omega^{(2)} + R_\omega^{(3)} + R_\omega^{(4)}$
$\sigma_\nu^{(2)}$
$\mathbb{L}_\nu^{(2)}$
β
$\sigma_\kappa^{(1)}$
$\equiv_{M_\kappa} R_\kappa^{(1)}$
$\equiv_{M_\omega} R_\omega^{(1)} + R_\omega^{(2)}$
$\sigma_\kappa^{(2)}$
$\sigma_\kappa^{(3)}$
$\mathbb{L}_\kappa^{(3)}$
$\sigma_\kappa^{(4)}$
γ
$\sigma_\omega^{(1)}$ $\sigma_\omega^{(2)}$ $\sigma_\omega^{(3)}$ $\sigma_\omega^{(4)}$ $\sigma_\omega^{(5)}$ $\sigma_\omega^{(6)}$ $\sigma_\omega^{(7)}$ $\sigma_\omega^{(8)}$
$\sigma_\omega^{(1)} + \sigma_\omega^{(2)} + \sigma_\omega^{(3)} + \sigma_\omega^{(4)} + \sigma_\omega^{(5)} + \sigma_\omega^{(6)} + \sigma_\omega^{(7)} + \sigma_\omega^{(8)} = S$

Fig. 1. Iterative decomposition in three steps. $\sigma_\chi^{(j)}$: partial sum, $R_\chi^{(j)}$: target value, M_χ : modulus, α, β and γ : proportion of additional -1s.

Notation. We use a different Greek letter (ϵ, κ, ω or ν) to denote the coefficient vectors of each subknapsack. In the original knapsack, we carry on using the letter ϵ. At the first level of decomposition, we now use $\nu^{(1)}$ and $\nu^{(2)}$ for the coefficient vectors of the two subknapsacks. At the middle level, we choose the notation $\kappa^{(1)}$ to $\kappa^{(4)}$. At the bottom level, we use the letters $\omega^{(1)}$ to $\omega^{(8)}$. We then let $N_\chi(x)$ denote the number of occurrences of $x \in \{-1, 0, 1\}$ in the coefficient vector χ. For a knapsack of n elements, we have:

$$\begin{array}{ll} N_\epsilon(1) = n/2, & N_\epsilon(-1) = 0, \\ N_\nu(1) \approx (1/4 + \alpha)n, & N_\nu(-1) \approx \alpha n, \\ N_\kappa(1) \approx (1/8 + \alpha/2 + \beta)n, & N_\kappa(-1) \approx (\alpha/2 + \beta)n, \\ N_\omega(1) \approx (1/16 + \alpha/4 + \beta/2 + \gamma)n, & N_\omega(-1) \approx (\alpha/4 + \beta/2 + \gamma)n \ . \end{array}$$

We always have $N_\chi(0) = n - N_\chi(1) - N_\chi(-1)$. Since all these numbers need to be rounded to integers for a concrete knapsack instance, we write $\approx$ instead of $=$ above. For each of the coefficient vectors $\chi^{(j)}$ we introduce the corresponding partial sum:

$$\sigma_\chi^{(j)} = \sum_{i=1}^{n} \chi_i^{(j)} a_i \ .$$

To control the size of the lists of solutions that arise at each level of decomposition, we introduce a modulus and target values for each of the subknapsacks. We denote the modulus corresponding to the bottom level by M_ω, we introduce 7 random values $R_\omega^{(j)}$ (for $1 \le j \le 7$) and let $R_\omega^{(8)} = S - \sum_{j=1}^{7} R_\omega^{(j)}$. We solve the eight modular subknapsacks:

$$\sigma_\omega^{(j)} \equiv R_\omega^{(j)} \pmod{M_\omega} \quad \text{for } 1 \le j \le 8 \ .$$

We denote by $\mathbb{L}_\omega^{(j)}$, the list of solutions of each of these subknapsacks.

Basic Principle and Modular Constraints. To build solutions at the middle level κ, we consider sums of two partial solutions from two neighboring lists $\mathbb{L}_\omega^{(2j-1)}$ and $\mathbb{L}_\omega^{(2j)}$ containing solutions of the last level. By construction, we see that:

$$\sigma_\kappa^{(j)} = \sigma_\omega^{(2j-1)} + \sigma_\omega^{(2j)} \equiv R_\omega^{(2j-1)} + R_\omega^{(2j)} \pmod{M_\omega}$$

which means that all these partial sums already have some fixed value modulo M_ω. To prune the size of the lists of solutions at this level, we add an extra constraint modulo M_κ (chosen coprime to M_ω). Thus, we introduce three random values $R_\kappa^{(j)}$ (for $1 \le j \le 3$) and let $R_\kappa^{(4)} = S - \sum_{j=1}^{3} R_\kappa^{(j)}$. The new lists of solutions are denoted by $\mathbb{L}_\kappa^{(j)}$.

For the first level, we proceed similarly, adding partial solutions from $\mathbb{L}_\kappa^{(2j-1)}$ and $\mathbb{L}_\kappa^{(2j)}$. Clearly, the resulting sums already have fixed values modulo M_κ and M_ω. Again, we introduce a modulus M_ν, a random value $R_\nu^{(1)}$ and we let $R_\nu^{(2)} = S - R_\nu^{(1)}$ to reduce the size of the lists.

Finally, the (presumably unique) solution of the original knapsack is found by searching for a collision of the form $\sigma_\nu^{(1)} + \sigma_\nu^{(2)} = S$ with $\sigma_\nu^{(1)} \in \mathbb{L}_\nu^{(1)}$ and $\sigma_\nu^{(2)} \in \mathbb{L}_\nu^{(2)}$. Figure 1 illustrates the technique.

To transform this informal description into a formal algorithm and to analyze its complexity, we need to specify how the lists $\mathbb{L}_\omega^{(j)}$ are constructed. We also explain how to merge solutions from one level to solutions at the next level and specify the choices of the moduli M_ω, M_κ and M_ν in the next paragraph.

Algorithmic Details. The eight lists $\mathbb{L}_\omega^{(j)}$ can be constructed using a straightforward adaptation of the simple birthday paradox algorithm. It suffices to split the n elements into two random subsets of size $n/2$ and to assume that the 1s and -1s are evenly[1] distributed between the two halves. As with the case of binary coefficient vectors, the probability of this event is the inverse of a polynomial in n. Thus by repeating polynomially many times, we recover all of $\mathbb{L}_\omega^{(j)}$ with overwhelming probability. Assuming that the elements in $\mathbb{L}_\omega^{(j)}$ are random modulo M_ω, the expected size of $\mathbb{L}_\omega^{(j)}$ is:

$$L_\omega = \frac{\mathcal{L}_\omega}{M_\omega} = \frac{\binom{n}{N_\omega(1), N_\omega(-1), N_\omega(0)}}{M_\omega} ,$$

where $\mathcal{L}_\omega$ is the multinomial coefficient that counts the number of ways to choose $N_\omega(1)$ 1s, $N_\omega(-1)$ -1s and $N_\omega(0)$ 0s among n elements. Since the number of ways to choose $N_\omega(1)/2$ 1s, $N_\omega(-1)/2$ -1s and $N_\omega(0)/2$ 0s among $n/2$ elements is $\approx \mathcal{L}_\omega^{1/2}$ for large n, the running time of the construction of each $\mathbb{L}_\omega^{(j)}$ is $\max(|\mathbb{L}_\omega^{(j)}|, \mathcal{L}_\omega^{1/2})$.

At the middle level, the expected size of $\mathbb{L}_\kappa^{(j)}$ is upper bounded by

$$L_\kappa = \frac{\mathcal{L}_\kappa}{M_\omega \cdot M_\kappa} = \frac{\binom{n}{N_\kappa(1), N_\kappa(-1), N_\kappa(0)}}{M_\omega \cdot M_\kappa} .$$

This is only an upper bound on the expected size since the definition of L_κ ignores the fact that we discard solutions that cannot be decomposed with the modular constraints of the lower level.

To construct these lists, we match values from $\mathbb{L}_\omega^{(2j-1)}$ and $\mathbb{L}_\omega^{(2j)}$ modulo M_κ using Algorithm 1 from Sect. 3.2. We let $\mathbb{K}_\kappa^{(j)}$ denote the resulting list. We then remove inconsistent solutions from $\mathbb{K}_\kappa^{(j)}$ in order to produce $\mathbb{L}_\kappa^{(j)}$. We say that a solution is inconsistent when the vector $\omega^{(2j-1)} + \omega^{(2j)}$ contains 2s or -2s and/or does not have the number of 1s, -1s and 0s specified by $N_\kappa(1)$, $N_\kappa(-1)$ and $N_\kappa(0)$. According to Sect. 3.2, the cost of this step is $\max(|\mathbb{L}_\omega^{(2j-1)}|, |\mathbb{L}_\omega^{(2j)}|, |\mathbb{K}_\kappa^{(j)}|)$.

Proceeding in the same way, we give an upper bound on the expected size of $\mathbb{L}_\nu^{(j)}$ by

$$L_\nu = \frac{\mathcal{L}_\nu}{M_\omega \cdot M_\kappa \cdot M_\nu} = \frac{\binom{n}{N_\nu(1), N_\nu(-1), N_\nu(0)}}{M_\omega \cdot M_\kappa \cdot M_\nu} .$$

[1] Or almost evenly when the number of 1s and/or -1s are odd.

Using the same notation as above, the cost to construct the two lists $\mathbb{L}_\nu^{(j)}$ is $\max(|\mathbb{L}_\kappa^{(2j-1)}|, |\mathbb{L}_\kappa^{(2j)}|, |\mathbb{K}_\nu^{(j)}|)$.

Finally, the last step is to apply the integer variant of Algorithm 1 to the two integer lists $\mathbb{L}_\nu^{(1)}$ and $\mathbb{L}_\nu^{(2)}$, obtaining a list $\mathbb{K}_0$ of (possibly inconsistent) solutions. The cost of this step is $\max(|\mathbb{L}_\nu^{(1)}|, |\mathbb{L}_\nu^{(2)}|, |\mathbb{K}_0|)$.

To estimate the size of $\mathbb{K}_0$, we count the number of expected solutions in a modular merge modulo the multiple of $M_\omega \cdot M_\kappa \cdot M_\nu$ closest to 2^n. This overestimates the size of $\mathbb{K}_0$ since it is slightly easier to find a knapsack solution modulo this value than a knapsack solution over the integers. This yields an estimate equal to:

$$L_\nu^2 \cdot \frac{M_\nu \cdot M_\kappa \cdot M_\omega}{2^n} \enspace .$$

If $\mathbb{K}_0$ contains at least one *consistent* solution, we obtain a solution of the initial knapsack problem.

To conclude the description of the algorithm, we need to specify the values of the moduli M_ω, M_κ and M_ν. The key idea at this point is to choose each modulus to ensure that each solution appearing at a given level is represented (on average) by a single decomposition at the previous level. Indeed, if we add a larger modular constraint, we lose solutions from one level to the next and if we choose a smaller constraints, we construct each solution many times which increases the overall cost. Using binomials and multinomials to compute the number of decompositions we obtain the following conditions for the values of the moduli:

$$M_\omega \approx \binom{N_\kappa(1)}{N_\kappa(1)/2} \cdot \binom{N_\kappa(-1)}{N_\kappa(-1)/2} \cdot \binom{N_\kappa(0)}{N_\omega(1)-N_\kappa(1)/2,\, N_\omega(-1)-N_\kappa(-1)/2,\ \star} \approx$$
$$2^{(1/8+\alpha+2\,\beta-2\gamma\log_2\gamma-(7/8-\alpha-2\,\beta-2\gamma)\log_2(7/8-\alpha-2\,\beta-2\gamma)+(7/8-\alpha-2\,\beta)\log_2(7/8-\alpha-2\,\beta)} \enspace ,$$

$$M_\kappa \cdot M_\omega \approx \binom{N_\nu(1)}{N_\nu(1)/2} \cdot \binom{N_\nu(-1)}{N_\nu(-1)/2} \cdot \binom{N_\nu(0)}{N_\kappa(1)-N_\nu(1)/2,\, N_\kappa(-1)-N_\nu(-1)/2,\ \star} \approx$$
$$2^{(1/4+2\alpha-2\beta\log_2\beta-(3/4-2\alpha-2\beta)\log_2(3/4-2\alpha-2\beta)+(3/4-2\alpha)\log_2(3/4-2\alpha))n} \enspace ,$$

$$\begin{aligned} M_\nu \cdot M_\kappa \cdot M_\omega &\approx \binom{n/2}{n/4} \cdot \binom{n/2}{N_\nu(-1),\, N_\nu(-1),\ \star} \\ &\approx 2^{(1/2-2\alpha\log_2\alpha-(1/2-2\alpha)\log_2(1/2-2\alpha)+(1/2)\log_2(1/2))n} \\ &\approx 2^{(-2\alpha\log_2\alpha-(1/2-2\alpha)\log_2(1/2-2\alpha))n} \enspace . \end{aligned}$$

The $\star$ symbol in the above multinomials denotes the number of remaining elements (corresponding to 0s) after specifying the number of 1s and -1s introduced to decompose the set of 0s from the lower level.

The overall running time of the algorithm is the maximum of the individual costs to run Algorithm 1 and the construction of the eight lists, which gives:

$$\tilde{\mathcal{O}}(\max(\max_j |\mathbb{L}_\omega^{(j)}|, \max_j \mathcal{L}_\omega^{1/2}, \max_j |\mathbb{K}_\kappa^{(j)}|, \max_j |\mathbb{L}_\kappa^{(j)}|, \max_j |\mathbb{K}_\nu^{(j)}|, \max_j |\mathbb{L}_\nu^{(j)}|, |\mathbb{K}_0|)) \enspace .$$

Assuming that each list has a size close to its expected value (see Sect. 3.5), the expected running time is:

$$T(\alpha,\beta,\gamma)=\tilde{\mathcal{O}}(\max(L_\omega,\mathcal{L}_\omega^{1/2},\frac{L_\omega^2}{M_\kappa},L_\kappa,\frac{L_\kappa^2}{M_\nu},L_\nu,L_\nu^2\cdot\frac{M_\nu\cdot M_\kappa\cdot M_\omega}{2^n}))\ .$$

Since none of the $\mathbb{K}_\chi$ lists need to be stored, the amount of memory required is:

$$\tilde{\mathcal{O}}(\max(L_\omega,\mathcal{L}_\omega^{1/2},L_\kappa,L_\nu))\ .$$

Finally, there is an additional, very important, parameter to consider, the probability of success p_{succ} taken over the possible random choices of the $R_\chi^{(j)}$ values. This parameter is quite tricky to estimate because it varies depending on the initial knapsack that we are solving. As an illustration, consider the knapsack whose elements are all equal to 0. It is clear that unless all the random $R_\chi^{(j)}$ are chosen equal to 0 then the algorithm cannot succeed. As a consequence, in this case the probability of success is very low. There are many other bad knapsacks; however, for a random knapsack, the expected probability of success is not too small (see Sect. 3.4 for a discussion).

Numerical Results for the Complexity Analysis. Minimizing the expected running time $T(\alpha,\beta,\gamma)$ results in:

$$\alpha=0.0267,\quad\beta=0.0168,\quad\gamma=0.0029\ .$$

With these values, we obtain:

$$\mathcal{L}_\omega\approx 2^{0.532\,n},\ L_\omega\approx 2^{0.291\,n},\ L_\kappa\approx 2^{0.279\,n},\ L_\nu\approx 2^{0.217\,n}\text{ and}$$
$$M_\omega\approx 2^{0.241\,n},\ M_\kappa\approx 2^{0.291\,n},\ M_\nu\approx 2^{0.267\,n}\ .$$

As a consequence, we find that both the time and memory complexity are equal to $\tilde{\mathcal{O}}(2^{0.291\,n})$. We can also check that the product of the three moduli $M_\omega\cdot M_\kappa\cdot M_\nu$ is smaller than the size of the numbers in the initial knapsack, i.e. 2^n.

However, we remark that γ is so small that for any achievable knapsack size n, the number of -1s added at the last level is 0 in practice. Thus, in order to improve the practical choices of the number of -1s at the higher levels, it is better to adjust the minimization with the added constraint $\gamma=0$. This leads to the alternative values:

$$\alpha=0.0194,\quad\beta=0.0119,\quad\gamma=0\ .$$

With these values, we obtain:

$$\mathcal{L}_\omega\approx 2^{0.463\,n},\ L_\omega\approx 2^{0.295\,n},\ L_\kappa\approx 2^{0.284\,n},\ L_\nu\approx 2^{0.234\,n}\text{ and}$$
$$M_\omega\approx 2^{0.168\,n},\ M_\kappa\approx 2^{0.295\,n},\ M_\nu\approx 2^{0.272\,n}\ .$$

We can also remark that by choosing $\alpha=\beta=\gamma=0$, we recover the time complexity $\tilde{\mathcal{O}}(2^{0.337\,n})$ given by May and Meurer [9] for the algorithm of [5]. However, in our case, the memory complexity is also $\tilde{\mathcal{O}}(2^{0.337\,n})$, which indicates that our algorithm can probably be improved in this respect. In the full version of the paper [2], we also consider the unbalanced case and possible extensions.

3.4 Analysis of the Probability of Success

In order to analyze the probability of success, it is convenient to bear in mind Fig. 1. We are starting from an unknown but fixed golden solution of the knapsack and we wish to decompose it seven times. (At each step we represent the 0s, 1s and -1s of the current coefficient vector by a tuple (i, j) where $i, j \in \{0, 1, -1\}$.) For each of the seven splits, we add a modular constraint modulo a number very close to the total number of decompositions. For example, during the top level split, we are specifying that the sum of the left hand-side after the splitting should be congruent to $R_\nu^{(1)}$ modulo M_ν, to $R_\kappa^{(1)} + R_\kappa^{(2)}$ modulo M_κ and to $R_\omega^{(1)} + R_\omega^{(2)} + R_\omega^{(3)} + R_\omega^{(4)}$ modulo M_ω. Since the three moduli are coprime, this is equivalent to simply specifying a value modulo $M_\nu \cdot M_\kappa \cdot M_\omega$. Each of the decompositions is considered successful if the current golden solution admits at least one way of splitting which satisfies the modular constraint. In this case, we focus on one of the admissible solutions for which we search for a decomposition in the level below. Fixing the solution on the left-hand side also determines the solution of the right-hand side.

Clearly, if each of the seven decompositions succeeds, the initial solution can be found by the algorithm. Assuming independence, the overall probability of success is at least equal[2] to the product of the probability of success of the individual decompositions. If we do not assume independence, we can still say that the overall probability of failure is smaller than the individual probabilities of failure.

Purely Random Heuristic Model. One approach to the analysis of the probability of an individual decomposition succeeding is to assume that for each of the possible decompositions, the resulting modular sum is a random value. We already know that there are knapsacks for which this assumption does not hold, as illustrated by the all-zero example. This is true for a large number of random, however, and is a very useful benchmark for the following analysis. For simplicity, we assume here that the number of possible decompositions is equal to the modulus M for a large set of random knapsacks.

In this case, it is well-known that for large values of M, the proportion of modular values which are not attained after picking M random values is close to $e^{-1} \simeq 0.36$.

Experimental Behavior of Decompositions. In Section 5, we describe an implementation of our algorithm on a 80-bit knapsack. To better understand the behavior of this implementation, it is useful to determine the probability of success of each decomposition. Three levels of decomposition occur. At the top level, a balanced golden solution with 40 zeros and 40 ones needs to be split into two partial solutions with 22 ones and two -1s each. At the middle level, a golden solution with 22 ones and two -1s is to be split into two partial solutions

[2] The probability can be larger, since we ignore multiple correct splits when they occur.

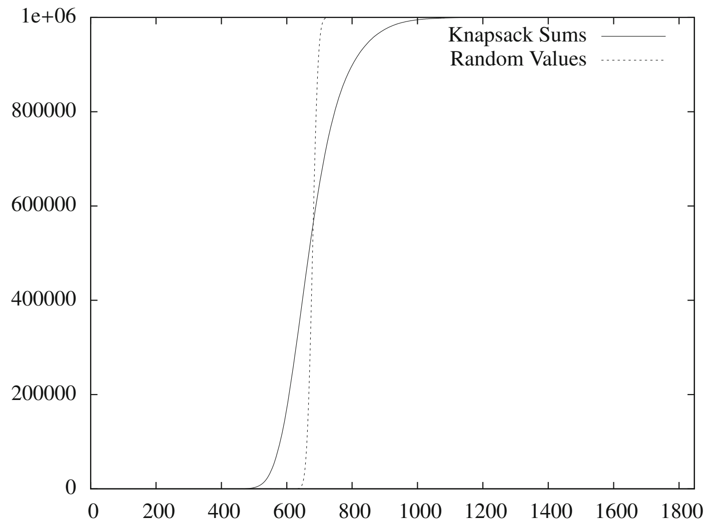

Fig. 2. Cumulative number of knapsacks (in a million) with less than a given number of not obtained values

with 12 ones and two -1s. Finally, at the bottom level, we split 12 ones and two -1s into twice 6 ones and one -1.

At the top level, the number of possible decompositions of a golden solution is larger than $\binom{40}{22}\binom{40}{2,2,36} \approx 2^{56}$. As a consequence, it is not possible to perform experimental statistics of the modular values of such a large set. At the middle level, the number of decompositions is larger than $\binom{22}{11}\binom{2}{1}\binom{46}{1,1,44} \approx 2^{32}$. Thus, it is possible to perform some experiments, but doing a large number of tests to perform a statistical analysis of the modular values is very cumbersome. At the bottom level, the number of decompositions of a golden solution is $\binom{12}{6}\binom{2}{1} =$ 1848. This is small enough to perform significant statistics and, in particular, to study the fraction of modular values which are not obtained (depending on a random choice of 14 knapsack elements, 12 1s and two -1s, to be split). The value of the modulus used in this experiment is 1847, the closest prime to 1848.

During our experimental study, we created one million modular subknapsacks from 14 randomly selected values modulo 1847. Among these values 12 elements correspond to additions and 2 to subtractions. From this set we computed (in $\mathbb{Z}_{1847}$) all of the 1848 values that can be obtained by summing 6 out of the 12 addition elements and subtracting one of the subtraction elements. In each experiment, we counted the number of values which were not obtained; the results are presented in Fig. 2. On the vertical axis we display the cumulative number of knapsacks which result in x or less unobtained values. To allow comparison with the purely random model, we display the same curve computed for one million of experiments where 1848 random numbers modulo 1847 are chosen. In particular, we see on this graph that for 99.99% of the random knapsacks we have constructed the fraction of unobtained value stays below 2/3. This means that experimentally, the probability of success of a decomposition at the bottom level is, at least, 1/3 for a very large fraction of knapsacks. Assuming independence between the probability of success of the seven splits and a similar behavior of three levels[3], we conclude that for 99.93% of random knapsacks an average number of $3^7 = 2187$ repetitions suffices to solve the initial problem.

Distribution of Modular Sums. When considering the decomposition of a given golden solution (at any level), we can construct the set $\mathcal{B}$ of all left-hand sides which can appear. For this set $\mathcal{B}$ we wish to study the distribution of the scalar product $\mathbf{a} \cdot \mathbf{x} = \sum_{i=1}^{n} a_i x_i \pmod{M}$, for given knapsack weights a_i. Let $P_{a_1,..,a_n}(\mathcal{B}, c)$ denote the probability that a knapsack of elements $\mathbf{a} = (a_1,..,a_n) \in \mathbb{Z}_M^n$ results in the value c modulo M for a uniformly at random chosen solution $(x_1,..,x_n)$ from $\mathcal{B}$,

$$P_{a_1,\cdots,a_n}(\mathcal{B}, c) = \frac{1}{|\mathcal{B}|}\left|\left\{(x_1,\cdots,x_n) \in \mathcal{B} \text{ such that } \sum_{i=1}^{n} a_i x_i \equiv c \pmod{M}\right\}\right| .$$

Our main tool to theoretically study the distribution of the scalar products is the following theorem [11, Theorem 3.2]:

Theorem 1. *For any set $\mathcal{B} \subset \mathbb{Z}_M^n$, the identity:*

$$\frac{1}{M^n} \sum_{(a_1,\cdots,a_n)\in\mathbb{Z}_M^n} \sum_{c\in\mathbb{Z}_M} \left(P_{a_1,\cdots,a_n}(\mathcal{B}, c) - \frac{1}{M}\right)^2 = \frac{M-1}{M|\mathcal{B}|}$$

holds.

With this equation we can prove a weak but sufficient result about the proportion of missed values during a decomposition. Let $\Lambda > 0$ be an arbitrary integer. We want to find an upper bound for the fraction f_Λ of "bad" knapsacks modulo M with less than M/Λ obtained values. First, we remark that for a knapsack $(a_1,\cdots,a_n)$ that reaches less than M/Λ values, at least $(\Lambda-1)M/\Lambda$ values modulo M are obtained 0 times. Since

$$\sum_{c\in\mathbb{Z}_M} P_{a_1,\cdots,a_n}(\mathcal{B}, c) = 1$$

some values c need to be obtained many times. As a consequence, we find that

$$\sum_{c\in\mathbb{Z}_M} \left(P_{a_1,\cdots,a_n}(\mathcal{B}, c) - \frac{1}{M}\right)^2 \geq \frac{\Lambda-1}{\Lambda} \cdot M \cdot \frac{1}{M^2} + \frac{1}{\Lambda} \cdot M \cdot \frac{(\Lambda-1)^2}{M^2} = \frac{\Lambda-1}{M} .$$

[3] The limited number of experiments we have performed for the middle level seem to indicate a comparable behavior. We performed 100 experiments and the number of not obtained values remained in the range 42% – 43.1%.

This implies that the number N_{bad} of “bad” knapsacks satisfies:

$$N_{\text{bad}} \leq M^n \cdot \frac{M-1}{(\Lambda-1)|\mathcal{B}|} \ .$$

With this bound, it is possible to construct a variation of our algorithm with a provable probability of success. Given $\Lambda \geq 10$ as a function of n, we repeat each split for $2\,\Lambda$ random and independently picked values. The probability of failure of such a repeated split is at most $e^{-2} \approx 0.135$, except for a “bad” knapsack. Thus, the global probability of failure on the seven splits is smaller than 95%. By choosing M smaller than $|\mathcal{B}|$ (but close to it), we ensure that the total fraction of bad knapsacks is at most:

$$\frac{7}{\Lambda-1} \ .$$

This fraction becomes arbitrarily small by choosing a large enough value of Λ. Note that the running time is multiplied by $(2\,\Lambda)^3$, since there are three nested levels of decompositions. If a probability of success of 5% is not sufficient, it is possible to increase the probability by repeating the complete algorithm with independent random numbers. A polynomial number of repetition leads to a probability of success exponentially close to 1 (with the exception of the “bad” knapsacks).

3.5 Analysis of the Size of the Lists

Concerning the size of the lists that occur during the algorithm, both the simple heuristic model and the experimental results (see Section 5) predict that the size of the lists are always very close to the theoretical values at the bottom level and smaller (due to the overestimation) at the levels above. It remains to use Theorem 1 to give an upper bound on the size of the various lists.

For the sizes of the lists $\mathbb{L}_\chi$, we can use a direct application of the theorem. The set of concern, $\mathcal{B}$, is the set of all repartitions of 1s, 0s and -1s fulfilling the conditions of $\mathbb{L}_\chi$. The modulus M is the product of all active moduli at the current and preceding levels. That is, for level ω we have $M = M_\omega$; for level κ, $M = M_\omega \cdot M_\kappa$, and for level ν we take $M = M_\omega \cdot M_\kappa \cdot M_\nu$.

Once again, we fix an integer Λ and consider the number F_Λ of knapsacks for which more than $M/(2\,\Lambda)$ values c have a probability that satisfies:

$$P_{a_1,\cdots,a_n}(\mathcal{B}, c) \geq \Lambda/M \ .$$

Due to Theorem 1, we find:

$$\frac{F_\Lambda}{M^n} \cdot \frac{M}{2\,\Lambda} \cdot \frac{(\Lambda-1)^2}{M^2} \leq \frac{M-1}{M|\mathcal{B}|} \leq \frac{1}{|\mathcal{B}|} \ .$$

As a consequence:

$$F_\Lambda \leq \frac{2\,\Lambda}{(\Lambda-1)^2} \cdot \frac{M}{|\mathcal{B}|} \cdot M^n \leq \frac{2\,\Lambda}{(\Lambda-1)^2} \cdot M^n \ .$$

The key point is that for a knapsack which is not one of the F_Λ knapsacks above and for most values of c (all but at most $M/(2\,\Lambda)$), the size of $\mathbb{L}_\chi$ is smaller than Λ times the expected value $|\mathcal{B}|/M$, that is,

$$|\mathbb{L}_\chi| \leq \frac{\Lambda\,|\mathcal{B}|}{M} \ .$$

To bound the size of the lists $\mathbb{K}_\chi$, we proceed slightly differently. The set $\mathcal{B}$ consists of 1s, 0s and -1s that are allowed in the $\mathbb{L}$ lists and are matched to construct $\mathbb{K}_\chi$. We write $M = M_1 \cdot M_2$, where M_1 is the product of the active moduli for the $\mathbb{L}$ list and M_2 is the modulus that is added when constructing $\mathbb{K}_\chi$. Let $\sigma \pmod{M}$ denote the target sum as a new modulo constraint for elements in $\mathbb{K}_\chi$. Let $\sigma_L \pmod{M_1}$ and $\sigma_R \pmod{M_1}$ respectively denote the values of the sums in the left-side and right-side lists $\mathbb{L}$. Of course, we have $\sigma_L + \sigma_R \equiv \sigma \pmod{M_1}$. We can write:

$$\begin{aligned}
|\mathbb{K}_\chi| &= \sum_{\substack{c \in \mathbb{Z}_M \\ c \equiv \sigma_L \pmod{M_1}}} (|\mathcal{B}| \cdot P_{a_1,\cdots,a_n}(\mathcal{B},c)) \cdot (|\mathcal{B}| \cdot P_{a_1,\cdots,a_n}(\mathcal{B},\sigma - c)) \\
&\leq \left[\sum_{\substack{c \in \mathbb{Z}_M \\ c \equiv \sigma_L}} (|\mathcal{B}| \cdot P_{a_1,\cdots,a_n}(\mathcal{B},c))^2 \times \sum_{\substack{c \in \mathbb{Z}_M \\ c \equiv \sigma_R}} (|\mathcal{B}| \cdot P_{a_1,\cdots,a_n}(\mathcal{B},c))^2 \right]^{1/2} . \quad (4)
\end{aligned}$$

Thus to estimate the size of the lists $\mathbb{K}_\chi$, we need to find an upper bound for the value of sums of the form:

$$\sum_{\substack{c \in \mathbb{Z}_M \\ c \equiv c_1 \pmod{M_1}}} P_{a_1,\cdots,a_n}(\mathcal{B},c)^2 \ .$$

To do this, it is useful to rewrite the relation from Theorem 1 as:

$$\frac{1}{M^n} \sum_{(a_1,\cdots,a_n)\in\mathbb{Z}_M^n} \sum_{c\in\mathbb{Z}_M} P_{a_1,\cdots,a_n}(\mathcal{B},c)^2 = \frac{M + |\mathcal{B}| - 1}{M|\mathcal{B}|} \ .$$

Given Λ, we let G_Λ denote the number of knapsacks for which more than $M_1/(8\,\Lambda)$ values c_1 have a sum of squared probabilities that satisfy:

$$\sum_{\substack{c \in \mathbb{Z}_M \\ c \equiv c_1 \pmod{M_1}}} P_{a_1,\cdots,a_n}(\mathcal{B},c)^2 \geq \frac{\Lambda^2}{M_1^2 M_2} \ .$$

We find that

$$\frac{G_\Lambda}{M^n} \cdot \frac{M_1}{8\,\Lambda} \cdot \frac{\Lambda^2}{M_1^2 M_2} \leq \frac{M + |\mathcal{B}| - 1}{M|\mathcal{B}|} \ ;$$

and as a consequence

$$G_\Lambda \leq \frac{8}{\Lambda} \cdot \frac{M + |\mathcal{B}|}{|\mathcal{B}|} \cdot M^n \ .$$

Moreover, we can check with our concrete algorithm that we always have $|\mathcal{B}| \geq M$ for the construction of the lists $\mathbb{K}_\chi$. Thus we have $G_\Lambda \leq (16/\Lambda)M^n$. For a knapsack which is not one of the G_Λ knapsacks above and for most values[4] of $\sigma_L \bmod M_1$, the size of $\mathbb{K}_\chi$ is smaller than Λ^2 times the expected value $|\mathcal{B}|^2/(M_1^2\, M_2)$, that is,

$$|\mathbb{K}_\chi| \leq \frac{\Lambda^2\, |\mathcal{B}|^2}{M_1^2 \cdot M_2} \ .$$

Note, that this bound includes the case $|\mathbb{K}_0|$.

3.6 Provable Variant of the Concrete Algorithm

Following the ideas presented in Sect. 3.4, we can now describe a variant of our concrete algorithm with provable probabilistic run-time and space requirements. First, fix a large enough value of Λ. We redefine the notion of a "bad" knapsack in this section, by saying that a knapsack is bad if it fails to fulfill one of the three criteria developed in Sect. 3.4 and Sect. 3.5. That is, if there are too many values that yield incorrect splits or lists of type $\mathbb{L}$ or $\mathbb{K}$ which are too large. We find that the total fraction of bad knapsacks is smaller than

$$7\left(\frac{1}{\Lambda-1} + \frac{2\,\Lambda}{(\Lambda-1)^2} + \frac{16}{\Lambda}\right) \leq \frac{140}{\Lambda} \quad \text{for } \Lambda \geq 7 \ .$$

By choosing a large enough value for Λ, this fraction can become arbitrarily small.

Once again, we consider a variation of the concrete algorithm where at each level we repeat the choice of random numbers often enough to be successful. For a "good" knapsack there are three ways a decomposition can fail . Firstly, we could choose a random value which does not permit a decomposition of the golden solution; Secondly, we could choose a random value which makes $\mathbb{L}_\chi$ overflow; Thirdly, we could choose a random value which makes $\mathbb{K}_\chi$ overflow. Note that the last two events can be detected, in which case we erase the lists that have been constructed for this random value and turn to the next. For each modulus, the proportion of random values which are incorrect with respect to at least one criteria is smaller than

$$\frac{\Lambda-1}{\Lambda} + \frac{1}{2\,\Lambda} + \frac{2}{8\,\Lambda} = 1 - \frac{1}{4\,\Lambda} \ .$$

Thus by repeating each split 8Λ times, we make sure that the probability of failure of a given split is at most e^{-2}. Once again, this yields a global probability of success of 5%, which becomes exponentially close to 1 by repeating polynomially many times. Given a real $\varepsilon > 0$, by setting $\Lambda = 2^{\varepsilon\, n}$ we obtain the following theorem:

[4] For all but at most $2M_1/(8\,\Lambda)$ – the factor 2 in the numerator comes from the fact that there are two terms to bound in ((4)).

Theorem 2. *For any real $\varepsilon > 0$ and for a fraction of at least $1 - 140 \cdot 2^{-\varepsilon n}$ of equibalanced knapsacks with density $D < 1$ given by an n-tuple $(a_1, \cdots, a_n)$ and a target value S, if $\epsilon = (\epsilon_1, \cdots, \epsilon_n)$ is a solution of the knapsack then the algorithm described in Sect. 3.3 modified as above finds the solution ϵ sought after in time $\tilde{\mathcal{O}}(2^{(0.291+3\varepsilon)\, n})$.*

We recall that in the theorem, the term *equibalanced* means that the solution ϵ contains exactly the same number of 0s and 1s.

4 Memory Complexity Improvement

In this section we first show a new algorithm of constant memory requirement and running time $\tilde{\mathcal{O}}(2^{3n/4})$. We then show how to decrease its time complexity down to $\tilde{\mathcal{O}}(2^{0.72n})$ using a technique similar to Howgrave-Graham and Joux [5]. Finally, we show a time memory tradeoff for Schroeppel-Shamir's algorithm down to $\tilde{\mathcal{O}}(2^{n/16})$ memory.

4.1 An Algorithm with Running Time $\tilde{\mathcal{O}}(2^{3n/4})$ and Memory $\tilde{\mathcal{O}}(1)$

We describe a simple algorithm that solves the knapsack problem in time $\tilde{\mathcal{O}}(2^{3n/4})$ and constant memory, using a meet-in-the-middle attack. This is done by formulating the meet-in-the-middle attack as a collision search problem (see [15]); then a constant memory cycle-finding algorithm can be used.

We define two functions $f_1, f_2 : \{0,1\}^{n/2} \to \{0,1\}^{n/2}$:

$$f_1(x) = \sum_{i=1}^{n/2} a_i x_i \mod 2^{n/2}, \quad f_2(y) = S - \sum_{i=n/2+1}^{n} a_i y_i \mod 2^{n/2}$$

where x_i denotes the i-th bit of x, and similarly for y_i. If we can find $x, y \in \{0,1\}^{n/2}$ such that $f_1(x) = f_2(y)$, then we get:

$$\sum_{i=1}^{n/2} a_i x_i + \sum_{i=n/2+1}^{n} a_i y_i = S \mod 2^{n/2} \ .$$

This gives a solution of the knapsack problem that is only valid modulo $2^{n/2}$. Since there are heuristically $\tilde{\mathcal{O}}(2^{n/2})$ such solutions holding modulo $2^{n/2}$, and only a single one that holds over $\mathbb{Z}$, a random (x, y) such that $f_1(x) = f_2(y)$ leads to the correct knapsack solution with probability roughly $2^{-n/2}$. Below we show that we can generate such random solution in time $\tilde{\mathcal{O}}(2^{n/4})$ and constant memory. This gives an algorithm with total running time $\tilde{\mathcal{O}}(2^{3n/4})$ and constant memory.

From the two functions f_1, f_2 we define the function $f : \{0,1\}^{n/2} \to \{0,1\}^{n/2}$ where:

$$f(x) = \begin{cases} f_1(x) & \text{if } g(x) = 0 \\ f_2(x) & \text{if } g(x) = 1 \end{cases}$$

where $g : \{0,1\}^{n/2} \to \{0,1\}$ is a random bit function. Then a collision $f(x) = f(y)$ for f gives a desired collision $f_1(x) = f_2(y)$ with probability 1/2. The function $f : \{0,1\}^{n/2} \to \{0,1\}^{n/2}$ is a random function, therefore using Floyd's cycle finding algorithm [6] we can find a collision for f in time $2^{n/4}$ and constant memory.

However we need to obtain a random collision whereas Floyd's cycle finding algorithm is likely to produce always the same collision. A simple solution is to further randomize the function f; more precisely we apply Floyd' cycle-finding algorithm to $f' : \{0,1\}^{n/2} \to \{0,1\}^{n/2}$ with $f'(x) = P(f(x))$, where P is a random permutation in $\{0,1\}^{n/2}$. Then a new permutation P is used every time a new collision (x, y) is required for f.

4.2 An Algorithm with Running Time $\tilde{\mathcal{O}}(2^{0.72n})$ and Memory $\tilde{\mathcal{O}}(1)$

In this section we show how to slightly decrease the running time down to $\tilde{\mathcal{O}}(2^{0.72n})$, still with constant memory; for this we use the Howgrave-Graham–Joux technique recalled in Sect. 2.1. Again for simplicity we assume that n is a multiple of 4, and that the Hamming weight of the knapsack solution ϵ is exactly $n/2$. As in (3) we write S as the sum $\sigma_1 + \sigma_2$ of two subknapsacks with Hamming weight $n/4$ chosen among the n knapsack elements.

We let W be the set of n-bit strings of Hamming weight $n/4$. We have $\#W = 2^{h(1/4)} \simeq 2^{0.81n}$. We define the two functions $f_1, f_2 : W \to \{0,1\}^{h(1/4)n}$:

$$f_1(y) = \sum_{i=1}^{n} a_i y_i \mod 2^{h(1/4)n}, \quad f_2(z) = S - \sum_{i=1}^{n} a_i z_i \mod 2^{h(1/4)n}$$

where y_i denotes the i-th bit of y, and similarly for z_i. We consider $y, z \in W$ such that:

$$f_1(y) = f_2(z) \ . \tag{5}$$

Since f_1 and f_2 are random functions heuristically there are $2^{h(1/4)n}$ solutions to (5). Moreover given the correct solution ϵ of the knapsack, as in Sect. 2.1 there are $\binom{n/2}{n/4} \simeq 2^{n/2}$ ways of writing this correct solution as $\sigma_1 + \sigma_2 = S$ (see (3)). All these $2^{n/2}$ solutions are solutions of (5). Therefore the probability p that a random solution of (5) leads to the correct knapsack solution is:

$$p = \frac{2^{n/2}}{2^{h(1/4)n}} \simeq 2^{-.31n} \ .$$

The input space of f_1, f_2 has size $2^{h(1/4)n}$. Therefore using the same cycle-finding algorithm as in the previous section, a random solution of (5) can be found in time $\tilde{\mathcal{O}}(2^{h(1/4)n/2})$. The total time complexity is therefore:

$$\begin{aligned}\tilde{\mathcal{O}}(2^{h(1/4)n/2})/p &= \tilde{\mathcal{O}}(2^{h(1/4)n/2}) \cdot 2^{(h(1/4)-1/2)n} \\ &= \tilde{\mathcal{O}}(2^{(3h(1/4)/2-1/2)n}) = \tilde{\mathcal{O}}(2^{.72n}) \ .\end{aligned}$$

Finally, we note that it is possible to further improve this complexity by adding -1s in the decomposition (as in Sect. 3) but the time complexity improvement is almost negligible.

4.3 A Time-Memory Tradeoff on Schroeppel-Shamir Down to $2^{n/16}$ Memory

The original Schroeppel-Shamir algorithm works in time $\tilde{\mathcal{O}}(2^{n/2})$ and memory $\tilde{\mathcal{O}}(2^{n/4})$. In this section we describe a continuous time-memory tradeoff down to $\tilde{\mathcal{O}}(2^{n/16})$ memory. That is we describe a variant of Schroeppel-Shamir with:

$$\text{Running time: } \tilde{\mathcal{O}}(2^{(11/16-\varepsilon)n}), \quad \text{Memory: } \tilde{\mathcal{O}}(2^{(1/16+\varepsilon)n})$$

for any $0 \leq \epsilon \leq 3/16$. For simplicity we first describe the algorithm with exactly $\tilde{\mathcal{O}}(2^{n/16})$ memory. We write the knapsack as $S = \sigma_1 + \sigma_2 + \sigma_3 + \sigma_4$ as in (2) where each σ_i is a knapsack of $n/4$ elements.

We guess three values R_1, R_2 and R_3 of $3n/16$-bit each and we let R_4 such that $R_1 + R_2 + R_3 + R_4 = S \mod 2^{3n/16}$. We consider the four subknapsack equations

$$\sigma_i = R_i \mod 2^{3n/16} \ .$$

We solve these four equations independently by using the original Schroeppel-Shamir algorithm. Therefore in time $\tilde{\mathcal{O}}(2^{n/8})$ and memory $\tilde{\mathcal{O}}(2^{n/16})$ we obtain four lists $\{\sigma_1\}$, $\{\sigma_2\}$, $\{\sigma_3\}$ and $\{\sigma_4\}$ satisfying the four equations. Eventually to recover the knapsack solution we merge these four lists using the same merging procedure as in the original Schroeppel-Shamir algorithm; since each list has size $\tilde{\mathcal{O}}(2^{n/16})$, the merging procedure runs in time $\tilde{\mathcal{O}}(2^{n/8})$ and memory $\tilde{\mathcal{O}}(2^{n/16})$. Since we have guessed three values of $3n/16$-bit each, the total running time is:

$$\tilde{\mathcal{O}}(2^{3n/16})^3 \cdot \left(\tilde{\mathcal{O}}(2^{n/8}) + \tilde{\mathcal{O}}(2^{n/8})\right) = \tilde{\mathcal{O}}(2^{11n/16})$$

as required, and the memory consumption is $\tilde{\mathcal{O}}(2^{n/16})$.

It is easy to generalize the previous algorithm to memory $\tilde{\mathcal{O}}(2^{(1/16+\varepsilon)n})$ for any $0 \leq \epsilon < 3/16$. For this we take the R_i's of size $(3/16 - \varepsilon)n$-bit each. We can still build the four lists $\{\sigma_i\}$ in time $\tilde{\mathcal{O}}(2^{n/8})$ using Schroeppel-Shamir, but this time the size of the lists is $\tilde{\mathcal{O}}(2^{(1/16+\varepsilon)n})$, therefore it requires $\tilde{\mathcal{O}}(2^{(1/16+\varepsilon)n})$ memory. The merging procedure now runs in time $\tilde{\mathcal{O}}(2^{(1/8+2\varepsilon)n})$, still with memory $\tilde{\mathcal{O}}(2^{(1/16+\varepsilon)n})$. Therefore the total running time is:

$$\tilde{\mathcal{O}}(2^{(3/16-\varepsilon)n})^3 \cdot \left(\tilde{\mathcal{O}}(2^{n/8}) + \tilde{\mathcal{O}}(2^{(1/8+2\varepsilon)n})\right) = \tilde{\mathcal{O}}(2^{(11/16-\varepsilon)n})$$

as required, for a memory consumption $\tilde{\mathcal{O}}(2^{(1/16+\varepsilon)n})$.

Surprisingly there remains a gap between our variant of Schroeppel-Shamir with $\tilde{\mathcal{O}}(2^{n/16})$ memory and our constant memory algorithm from Sect. 4.1. Namely we were unable to find a variant of Schroeppel-Shamir requiring less than $\tilde{\mathcal{O}}(2^{n/16})$ memory, nor a cycle-based algorithm requiring more than $\tilde{\mathcal{O}}(1)$ memory.

5 Implementation and Experimental Evidence

In order to verify the correctness of the algorithm presented in Sect. 3.3, we have implemented it. We ran our implementation on 50 random knapsacks containing

Table 1. Experimental versus theoretical sizes of the intermediate lists

List type	Min. size	Max. size	Theoretical estimate
$\mathbb{L}_\omega$	12 024 816	12 056 576	$L_\omega = \frac{\binom{80}{6,1,73}}{1\,847} \approx 12\,039\,532$
$\mathbb{K}_\kappa$	61 487 864	61 725 556	$\frac{L_\omega^2}{2\,352\,689} \approx 61\,610\,489$
$\mathbb{L}_\kappa$	12 473 460	20 224 325	$L_\kappa = \frac{\binom{80}{12,2,66}}{1\,847 \cdot 2\,352\,689} \approx 31\,583\,129$
$\mathbb{K}_\nu$	14 409 247	23 453 644	$\frac{L_\kappa^2}{17\,394\,593} \approx 57\,345\,064$
$\mathbb{L}_\nu$	183 447	268 964	$L_\nu = \frac{\binom{80}{22,2,56}}{1\,847 \cdot 2\,352\,689 \cdot 17\,394\,593} \approx 592\,402$
$\mathbb{K}_0$	178	1 090	$\frac{L_\nu^2 \cdot 1\,847 \cdot 2\,352\,689 \cdot 17\,394\,593}{2^{80}} \approx 21\,942$

80 elements on 80 bits. The target sum was constructed in each case as a sum of 40 knapsack elements. For each of these knapsacks, we ran our implementation several times, choosing new random modular constraints for each execution, until a solution was found. We added two -1s at the first level, one -1 at the second and none at the third level. At the same time, we collected some statistics about the behavior of our code.

The total running time to solve the 50 knapsacks was 14 hours and 50 minutes on a Intel® Core$^{\text{TM}}$ i7 CPU M 620 at 2.67GHz. The total number of repetitions of the basic algorithm was equal to 280. We observed that a maximum of 16 repetitions (choice of a different random value in level ν) was sufficient to find the solution. Also, 53% of the 50 random knapsacks needed only up to 4 repetitions. On average, each knapsack required 5.6 repetitions.

During the execution of the 280 repetitions of the basic algorithm, we also noted the length of the lists $\mathbb{L}$ and $\mathbb{K}$ (still containing inconsistent solutions) that occurred at each level. The moduli were chosen as primes of size as discussed in Sect. 3.3: $M_\omega = 1\,847$, $M_\kappa = 2\,353\,689$, and $M_\nu = 17\,394\,593$. The experimental and theoretical list sizes are given in Table 1.

We see in Table 1 that the sizes of $\mathbb{L}_\omega$ and $\mathbb{K}_\kappa$ are very close to the predicted values and do not vary a lot. We refer to the full version of our paper [2] for a more detailed discussion; see also [2] for implementation results on 96 bits.

Acknowledgments. We would like to thank Alexander May and Alexander Meurer for pointing out the inconsistency issue in Howgrave-Graham–Joux algorithm. We also thank Dan Bernstein for helpful comments on a preliminary version of this work.

References

1. Ajtai, M.: The shortest vector problem in L_2 is NP-hard for randomized reductions (extended abstract). In: STOC 1998, pp. 10–19 (1998)
2. Becker, A., Coron, J.-S., Joux, A.: Improved generic algorithms for hard knapsacks. Eprint archive (2011)
3. Coster, M.J., Joux, A., LaMacchia, B.A., Odlyzko, A.M., Schnorr, C.-P., Stern, J.: Improved low-density subset sum algorithms. Computational Complexity 2, 111–128 (1992)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
5. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 235–256. Springer, Heidelberg (2010)
6. Knuth, D.E.: The Art of Computer Programming, 2nd edn. Seminumerical Algorithms, vol. II. Addison-Wesley, Reading (1981)
7. Lagarias, J.C., Odlyzko, A.M.: Solving low-density subset sum problems. J. ACM 32(1), 229–246 (1985)
8. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen 261, 515–534 (1982)
9. May, A., Meurer, A.: Personal communication
10. Merkle, R.C., Hellman, M.E.: Hiding information and signatures in trapdoor knapsacks. IEEE Transactions on Information Theory 24, 525–530 (1978)
11. Nguyen, P.Q., Shparlinski, I.E., Stern, J.: Distribution of modular sums and the security of the server aided exponentiation. In: Progress in Computer Science and Applied Logic, Final Proceedings of Cryptography and Computational Number Theory Workshop, Singapore, vol. 20, pp. 331–224 (2001)
12. Schnorr, C.-P.: A hierarchy of polynomial time lattice basis reduction algorithms. Theor. Comput. Sci. 53, 201–224 (1987)
13. Schroeppel, R., Shamir, A.: A $T = O(2^{n/2}), S = O(2^{n/4})$ algorithm for certain NP-complete problems. SIAM J. Comput. 10(3), 456–464 (1981)
14. Shamir, A.: A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. In: CRYPTO 1982, pp. 279–288 (1982)
15. van Oorschot, P.C., Wiener, M.J.: Improving implementable meet-in-the-middle attacks by orders of magnitude. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 229–236. Springer, Heidelberg (1996)
16. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–303. Springer, Heidelberg (2002)