Low Weight Discrete Logarithm and Subset Sum in $2^{0.65n}$ with Polynomial Memory

Andre Esser and Alexander May[*]

Ruhr University Bochum, Germany
{andre.esser, alex.may}@rub.de

Abstract. We propose two heuristic polynomial memory collision finding algorithms for the low Hamming weight discrete logarithm problem in any abelian group G. The first one is a direct adaptation of the Becker-Coron-Joux (BCJ) algorithm for subset sum to the discrete logarithm setting. The second one significantly improves on this adaptation for all possible weights using a more involved application of the representation technique together with some new Markov chain analysis. In contrast to other low weight discrete logarithm algorithms, our second algorithm's time complexity interpolates to Pollard's $|G|^{\frac{1}{2}}$ bound for general discrete logarithm instances.
We also introduce a new heuristic subset sum algorithm with polynomial memory that improves on BCJ's $2^{0.72n}$ time bound for random subset sum instances $a_1, \dots, a_n, t \in \mathbb{Z}_{2^n}$. Technically, we introduce a novel nested collision finding for subset sum – inspired by the NestedRho algorithm from Crypto '16 – that recursively produces collisions. We first show how to instantiate our algorithm with run time $2^{0.649n}$. Using further tricks, we are then able to improve its complexity down to $2^{0.645n}$.

Keywords: Low weight dlog, subset sum, representations, Nested Rho.

1 Introduction

The subset sum problem is one of the most fundamental problems in cryptography. It was early used in the 80's for the construction of cryptosystems [8, 21], suffered from lattice-based attacks [18, 23], and found a revival in the last two decades [11,19] since even LWE(SIS) [25] and LPN [2] instances can be formulated as (vectorial) versions of subset sum.

In this paper, we study subset sum instances $a_1, \dots, a_n, t \in \mathbb{Z}_{2^n}$. These are known as density-1 instances in the cryptographic literature, and they enjoy some useful hardness properties [16]. The invention of the polynomial memory $2^{0.72n}$-algorithm by Becker, Coron and Joux (BCJ) [4] initiated a renewed interest in the subset sum problem itself [1,3] and its vectorial versions [5,15]. Polynomial memory algorithms are of crucial importance to cryptanalysis, since they allow for very

[*] Funded by DFG under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

efficient implementations, and therefore are often used for record computations [7, 10]. Moreover, if we aim at implementing cryptanalytic quantum algorithms [6] in the near future, we have to focus on the development of low memory algorithms.

Our contributions

Discrete logarithms. It is not hard to see that also the famous *discrete logarithm problem* (DLP) reduces to the subset sum problem. Let $g^\alpha = \beta$ be a discrete logarithm instance in some abelian group G generated by g with order $2^{n-1} \leq |G| < 2^n$. Then we can easily compute the values $a_i := g^{2^{i-1}}$. Any subset $I \subset \{1,\ldots,n\}$ of all a_i that combines in G to β immediately reveals the bits of the discrete logarithm α. Moreover, if we know a priori that α has low Hamming weight ωn, $\omega \leq \frac{1}{2}$, then we have to find some I of small size $|I| = \omega n$, a fact from which subset sum algorithms usually benefit.

While the security of discrete log-based schemes is usually not directly based on the hardness of low weight DLP, their side channel resistance is linked to low weight DLP. Assume that we obtain via some side channel a faulty version $\tilde{\alpha}$ of a discrete logarithm α. Further assume that $\tilde{\alpha}$ is obtained from α by flipping ωn one bits to zero, but not flipping any zero bits to one (a quite usual setting for e.g. cold boot attacks [14]). Then $\frac{\beta}{g^{\tilde{\alpha}}} = g^{\alpha - \tilde{\alpha}} =: g^{\alpha'}$ forms a low weight DLP with $\mathrm{wt}(\alpha') = \omega n$. In this setting, any low weight DLP algorithm serves as an error correction algorithm for reconstructing α from $\tilde{\alpha}$.

For both DLP settings – the general discrete logarithm problem as well as its low weight variant – there exist algorithms matching the square root time lower bound for generic algorithms [26, 27]. But only for the general DLP we have Pollard's Rho algorithm with polynomial memory that matches the time lower bound $|G|^{\frac{1}{2}}$. Whether there exists a low memory algorithm for the low weight DLP was left as an open question by Galbraith and Gaudry [12, 13]. We do a significant step towards answering this question in the affirmative by giving a heuristic algorithm that achieves the time lower bound $|G|^{\frac{H(\omega)}{2}}$, where H is the binary entropy function. While there is a variant of the Baby-Step Giant Step algorithm, which also achieves time $|G|^{\frac{H(\omega)}{2}}$, our algorithm consumes way less memory. To quantify, our algorithm consumes always less than $|G|^{0.23}$ memory, while Baby-Step Giant-Step for $\omega = \frac{1}{2}$ consumes as much as $|G|^{\frac{1}{2}}$ memory. Additionally, we are able to instantiate our algorithm with polynomial memory only, where it outperforms all other known low-weight DLP algorithms in that memory regime for the whole range of weights $0 \leq \omega \leq \frac{1}{2}$.

In more detail, we show the following discrete log results.

– The BCJ subset sum algorithm works in a more general setting that we call group subset sum, from which we directly obtain a BCJ adaptation to the low weight discrete logarithm setting via the above reduction. This adaptation already improves on the best known folklore low weight discrete log algorithm (see e.g. Chapter 14.8.1 in [12]), which is an application of van Oorschot-Wiener's collision finding [28].

– We introduce an improved low weight discrete logarithm algorithm, inspired by the BCJ adaptation, that makes use of the special form $a_i = g^{2^{i-1}}$ of our subset sum instance. Our idea is to represent weight-ωn discrete logarithms α as the sum of two integers of smaller weight $\phi(\omega)$ by exploiting the fact of carry propagation. Technically, we introduce a Markov chain analysis for finding the optimal weight $\phi(\omega)$.
– By not insisting on polynomial memory, we tune our algorithm via Parallel Collision search [29] to reach the time complexity $|G|^{\frac{H(\omega)}{2}}$ of the low weight DLP variant of Baby-Step Giant-Step, while consuming only $|G|^{H(\omega)-H(\phi(\omega))}$ memory.

Our results for polynomial space algorithms are illustrated in Figure 1.

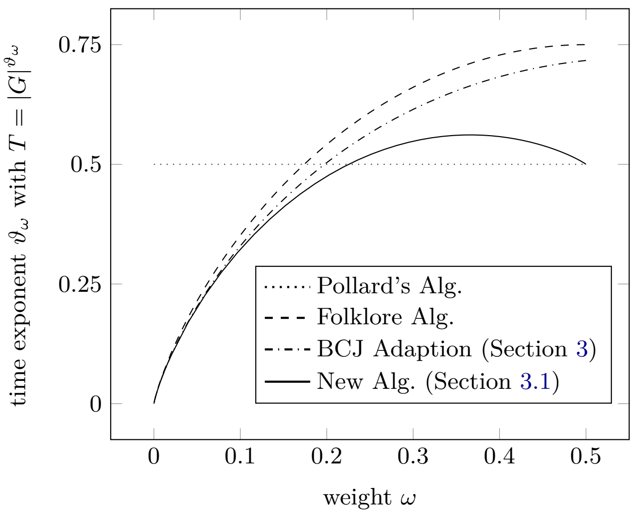


Fig. 1: Comparison of runtime exponents ϑ_ω for low weight DLP algorithms

Subset Sums. Previous polynomial memory subset sum algorithms based on collision finding, such as the folklore algorithm or the BCJ algorithm, are non-optimal in the following sense. In a first step, these algorithms output collisions that correspond to potential solutions $\mathbf{e}' = (e'_1, \ldots, e'_n) \in \{0,1\}^n$ such that the subset sum identity $\sum_i e'_i a_i = t$ holds only for a constant fraction of all n bits. In a second step, the algorithms brute-force potential solutions $\mathbf{e}'$ until by chance the identity holds on all bits.

We replace this collision-and-brute-force approach by a two-layer nested collision finding that is inspired by the NestedRho algorithm from Dinur, Dunkelman, Keller and Shamir [9], which was introduced in the context of finding the mode of a distribution. More precisely, we find in layer-1 potential solutions $\mathbf{e}'$ that satisfy the subset sum identity on $n/2$ bits, where in layer-2 our algorithm produces only candidates that also satisfy the subset sum identity on the remaining $n/2$ bits. Therefore, our algorithm always returns some $\mathbf{e}'$ satisfying $\sum_i e'_i a_i = t \bmod 2^n$. Moreover, each iteration costs us time roughly $2^{n/2}$.

As a collision finding technique this is optimal, since for a search space of size 2^n we need to perform $\Omega(2^{n/2})$ operations to obtain collisions at all (with good probability). Unfortunately, our collision-finding algorithm does not solve the subset sum problem in time $2^{n/2}$, since it produces potential solutions $\mathbf{e}'_i \in \{0,1,2,3,4\}^n$.

However, we show that after $2^{0.149n}$ iterations of our algorithm we expect to find some $\mathbf{e}'_i \in \{0,1\}^n$ that solves subset sum. This leads to a subset sum algorithm with complexity $2^{0.649n}$. Using additional tricks, we further improve to $2^{0.645n}$.

One might hope that our improvements for subset sum then in turn lead to improvements for the low Hamming weight discrete logarithm problem. However, for instantiating our two-layer subset sum collision finding we make use of the canonical group homomorphism $(\mathbb{Z}_{2^n}, +) \to (\mathbb{Z}_{2^{n/2}}, +)$, a subgroup structure that usually does not exist for discrete logarithm groups $(G, \cdot)$ (see [20] for results in composite groups).

Our paper is organized as follows. In Section 3 we introduce the general group subset sum problem, which we solve via the BCJ algorithm, and derive our first low weight discrete logarithm algorithm. Our second low weight DLP algorithm is described, analyzed, and experimentally validated in Section 3.1. Our $2^{0.649n}$ subset sum algorithm is given in Section 4 & 4.1, and experimentally verified in Section 4.3, the improvement to $2^{0.645n}$ can be found in Section 4.2.

2 Preliminaries

It is well-known that a collision in an n-to-n bit function f can be computed using roughly $2^{\frac{n}{2}}$ function evaluations and only a polynomial amount of memory. Common collision search algorithms [17, 22, 24] achieve this by computing a chain of invocations of f from a random starting point x, that is the iteration $f(x), f^2(x) := f(f(x)), f^3(x), \ldots$, until a repetition occurs, which in turn is found via some cycle finding algorithm (see also Figure 2). Let $f^\ell(x)$ be the first repeated value in the chain and $f^{\ell+\mu}$ its second appearance. We denote by $\texttt{Rho}(f, x)$ the output of some collision search procedure on f started at point x, that is the pair of colliding inputs. In other words

$$\texttt{Rho}(f, x) := (f^{\ell-1}(x), f^{\ell+\mu-1}(x)).$$

The name $\texttt{Rho}$ stems from the usual illustration from Figure 2 of a colliding chain's shape for iterated collision search.

In [28] van Oorschot and Wiener extended this idea of collision search to collisions between two functions f_1 and f_2. The van Oorschot-Wiener construction defines a new function $\tilde{f}$ that alternates between applications of f_1 and f_2 depending on the input. The output of a collision search in $\tilde{f}$ yields then a collision between f_1 and f_2 with probability $\frac{1}{2}$, whereas with probability $\frac{1}{2}$ it

$$\begin{tikzcd}
 & x_1 \arrow[r, "f"] & x_2 \arrow[r, "f"] & \cdots \arrow[r, "f"] & x_{\ell-1} \arrow[r, "f"] & x_\ell \arrow[dr, "f"] & \\
x \arrow[ur, "f"] & & & & x_{\ell+\mu-1} \arrow[ur, "f"] & & x_{\ell+1} \arrow[d, "f"] \\
 & & & & \cdots \arrow[u, "f"] & & x_{\ell+2} \arrow[dl, "f"] \\
 & & & & & x_{\ell+3} \arrow[ul, "f"] &
\end{tikzcd}$$

Fig. 2: Application of `Rho` to function f with starting point x.

produces a collision between the same functions f_1 and f_1, or f_2 and f_2. If we obtain a collision between the same functions, we may (in a deterministic fashion) manipulate the start point x, until we obtain a collision between f_1 and f_2. On expectation, this only doubles the run time (in case that collisions are independent for different start points, which we address below). Therefore, in the following we assume without loss of generality that we always obtain collisions for f_1, f_2 and define

$$\texttt{Rho}(f_1, f_2, x) := (x_1, x_2) \text{ with } f_1(x_1) = f_2(x_2).$$

Obviously, by restricting the collision search to the function $\tilde{f}$ not all collisions between f_1 and f_2 can be found anymore. However, on expectation this concerns only a constant fraction of all collision and hence we safely ignore this in our analysis.

All the algorithms considered in this work perform exponentially many invocations of the `Rho` algorithm on different starting points using the same function f. This causes some technical dependency problems. For instance in Figure 2 $\texttt{Rho}(f, \cdot)$ produces the same output collision for any start point $x, x_1, \ldots, x_{\ell-1}$. This problem was already identified in the work of [4, 9], who both introduced similar notions to break dependencies, called flavours in [9]. We adapt this notion to our purpose.

Definition 2.1 (Flavour of a function). *Let f be a function with $f\colon \mathcal{T} \to \mathcal{T}$, where $\mathcal{T} \subseteq \{0,1\}^n$. Let $P_k : \mathcal{T} \to \mathcal{T}$ be a family of bijective functions addressed by k. Then the k^{th} flavour of f is defined as*

$$f^{[k]}(\mathbf{x}) := P_k(f(\mathbf{x})).$$

Notice that for all k, a collision $(\mathbf{x}_1, \mathbf{x}_2)$ in $f^{[k]}$ satisfies

$$f^{[k]}(\mathbf{x}_1) = f^{[k]}(\mathbf{x}_2) \;\Leftrightarrow\; P_k(f(\mathbf{x}_1)) = P_k(f(\mathbf{x}_2)) \;\Leftrightarrow\; f(\mathbf{x}_1) = f(\mathbf{x}_2).$$

Thus, $(\mathbf{x}_1, \mathbf{x}_2)$ is also a collision in f itself. However, different flavours $f^{[k]}$ invoke different function graphs. We use flavours of f to heuristically obtain independence of the $\texttt{Rho}(f, \cdot)$ invocations. Namely, we assume that different $\texttt{Rho}(f, \cdot)$ invocations independently produce uniformly distributed collisions in f. A similar heuristic was also used in [9], and the authors verified their heuristic experimentally.

We analyze our algorithms in $\tilde{\Theta}$-notation, where $\tilde{\Theta}(2^n)$ suppresses polynomial factors in n. By $H(x)$ we refer to the binary entropy function defined as $H(x) := -x\log(x) - (1-x)\log(1-x)$, where all logarithms are base 2. Using Stirling's formula, we estimate binomial coefficients as

$$\binom{n}{m} = \tilde{\Theta}\left(2^{nH\left(\frac{m}{n}\right)}\right) \ .$$

For $a, b \in \mathbb{N}$ with $1 \leq a < b$ we let $[a, b] := \{a, a+1, \ldots, b\}$ and conveniently write $[b] := [1, b]$. For a vector $\mathbf{y} \in \{0,1\}^n$ we denote by $\mathrm{wt}(\mathbf{y}) := |\{i \in [n] \mid y_i = 1\}|$ the Hamming weight of $\mathbf{y}$ while for an integer $a \in \mathbb{N}$, $\mathrm{wt}(a)$ denotes the Hamming weight of the binary representation of a.

We denote by $\mathbb{Z}_N$ the ring of integers modulo N. We call $(\mathbf{x}_1, \ldots, \mathbf{x}_k) \in (\mathbb{Z}^n)^k$ a representation of $\mathbf{x} = \mathbf{x}_1 + \ldots + \mathbf{x}_k$ over $\mathbb{Z}^n$.

3 A Generalized View on the BCJ subset sum algorithm

In this section we define a generalized *group subset sum problem* and show that the BCJ algorithm also succeeds on this generalization. This abstraction contains the usual subset sum problem in $\mathbb{Z}_{2^n}$ as well as our new application, the low weight *discrete logarithm problem* (DLP), as special cases. As a first result we obtain a BCJ-type algorithm solving the low weight DLP using only a polynomial amount of memory — in any group, generically.

Definition 3.1 (Group Subset Sum). *Let $(G, \cdot)$ be an abelian group with order $|G|$ satisfying $2^{n-1} \leq |G| < 2^n$. In the* group subset sum problem *we are given $a_1, \ldots, a_n, t \in G$ together with $\omega, 0 < \omega \leq \frac{1}{2}$ such that there exists some solution $\mathbf{e} = (e_1, \ldots, e_n) \in \{0,1\}^n$ satisfying*

$$\prod_{i=1}^{n} a_i^{e_i} = t \text{ in } G \text{ with } \mathrm{wt}(\mathbf{e}) = \omega n.$$

Our goal is to recover $\mathbf{e}$ (or some other weight-ωn solution $\mathbf{e}'$).

Notice that by Definition 3.1 our group subset sum problem has some desired solution $\mathbf{e}$. In cryptographic applications, such a solution exists by construction and is usually unique. Moreover, the weight ω is generally known. If ω is unknown, one may iterate over all $\mathcal{O}(n)$ choices. As the change to target $t' = t^{-1} \cdot \prod_{i=1}^{n} a_i$ with complimentary solution $(1, \ldots, 1) - \mathbf{e}$ yields an instance with solution weight $(1-\omega)n$, there is no loss in generality assuming $\omega \leq \frac{1}{2}$. Additionally, one usually knows the generators of G such that one can define the a_i via generators.

Let us now look at two interesting special cases of group subset sum.

Subset Sum. Let $(a_1,\dots,a_n,t)$ be a subset sum instance. By considering $G = (\mathbb{Z}_{2^n},+)$ the product $\prod_{i=1}^n a_i^{e_i} = t$ in G rewrites directly to the usual subset sum identity

$$\sum_{i=1}^n e_i a_i = t \bmod 2^n.$$

Low weight DLP. Let $(G,\cdot)$ be a cyclic group generated by g. Let $g^\alpha = \beta$ be a discrete logarithm instance in G. Let us define $a_i = g^{2^{i-1}}$ for $1 \le i \le n$ and $t = \beta$. Let $\mathbf{e}$ be a solution of the group subset sum problem, that is

$$t = \prod_{i=1}^n a_i^{e_i} = \prod_{i=1}^n g^{e_i 2^{i-1}} = g^{\sum_{i=1}^n e_i 2^{i-1}}. \tag{1}$$

Thus, $\mathbf{e} = (e_1,\dots,e_n)$ immediately implies a solution $\alpha = \sum_{i=1}^n e_i 2^{i-1}$ to the DLP. Moreover, low weight group subset sum solutions $\mathbf{e}$ imply low weight discrete logarithms α.

Folklore algorithm. Let us first translate the folklore algorithm for low weight DLP – as for example described in [12] – into the notion of the group subset sum problem. We take $\mathcal{T} := \{\mathbf{x} \in \{0,1\}^{\frac{n}{2}} \mid \mathrm{wt}(\mathbf{x}) = \frac{\omega n}{2}\}$ with $|\mathcal{T}| = \tilde{\Theta}(|G|^{\frac{1}{2}H(\omega)})$. Let us define a hash function $\pi : G \to \mathcal{T}$. Further, we define functions $f,\ f_t$ as

$$f, f_t \colon \mathcal{T} \to \mathcal{T} \ \ \text{, where}$$
$$f(\mathbf{x}) = \pi\left(\prod_{i=1}^{\frac{n}{2}} a_i^{x_i}\right) \text{ and } f_t(\mathbf{x}) = \pi\left(t \cdot \prod_{i=1}^{\frac{n}{2}} a_{\frac{n}{2}+i}^{-x_i}\right) \ .$$

Now we search for collisions $(\mathbf{x}_1,\mathbf{x}_2)$ between f and f_t until $\mathbf{x} = \mathbf{x}_1||\mathbf{x}_2$ solves the group subset sum problem. Note, that there is a unique decomposition of the desired solution $\mathbf{e} = \mathbf{x}_1||\mathbf{x}_2$ and hence a single collision $(\mathbf{x}_1,\mathbf{x}_2)$ giving rise to $\mathbf{e}$. This in turn requires us to find all collisions between f and f_t. However, we expect f, f_t to have $|\mathcal{T}|$ collisions, where finding each collision costs $\tilde{\Theta}(|\mathcal{T}|^{\frac{1}{2}})$ function evaluations. In total, we achieve expected runtime

$$T = \tilde{\Theta}(|\mathcal{T}|^{\frac{3}{2}}) = \tilde{\Theta}(|G|^{\frac{3}{4}H(\omega)}) \ . \tag{2}$$

The runtime exponent $\frac{3}{4}H(\omega)$ is depicted in Figure 3.

A pseudocode description of the folklore algorithm in the group subset sum setting is given by Algorithm 1, where we instantiate f_1, f_2 via their function definitions f, f_t.

Algorithm 1 Group Subset Sum Solver

```
Input: functions $f_1, f_2 \colon \mathcal{D} \to \mathcal{D}$, group subset sum instance $(a_1, \dots, a_n, t) \in G^{n+1}$
Output: solution $\mathbf{e} \in \{0,1\}^n$ satisfying $\prod_{i=1}^n a_i^{e_i} = t$
1: repeat
2:     choose random flavour $k$
3:     choose random starting point $\mathbf{s} \in \mathcal{D}$
4:     $(\mathbf{x}, \mathbf{y}) \leftarrow \texttt{Rho}(f_1^{[k]}, f_2^{[k]}, \mathbf{s})$
5:     $\mathbf{e}' \leftarrow \mathbf{x} + \mathbf{y}$
6: until $\mathbf{e}' \in \{0,1\}^n$ and $\prod_{i=1}^n a_i^{e'_i} = t$
7: return $\mathbf{e}'$
```

Remark 3.1. We find collisions via the `Rho` procedure defined in Section 2 (see Algorithm 1). To (heuristically) guarantee independence of the collisions, we choose a random flavour k (see Definition 2.1) each time we invoke `Rho`. This means instead of searching for collisions between f and f_t themselves, we search for collisions between their flavoured versions $f^{[k]}$ and $f_t^{[k]}$. Analogously, we have to proceed for the subsequently described algorithms, our BCJ adaptation and our new algorithm in Section 3.1. However, for ease of notation we skip the flavours in our descriptions.

BCJ algorithm. The idea of the memoryless BCJ algorithm is to split the solution vector $\mathbf{e}$ with $\mathrm{wt}(\mathbf{e}) = \omega n$ in two vectors $\mathbf{e}_1, \mathbf{e}_2 \in \{0,1\}^n$ each of weight $\frac{\omega n}{2}$, which add up to $\mathbf{e}$. Let $(\mathbf{a}, t)$ be a group subset sum instance and $\mathcal{T} := \{\mathbf{x} \in \{0,1\}^n \mid \mathrm{wt}(\mathbf{x}) = \frac{\omega n}{2}\}$, where $|\mathcal{T}| = \tilde{\Theta}(2^{H(\frac{\omega}{2})n}) = \tilde{\Theta}(|G|^{H(\frac{\omega}{2})})$.

Let us define a hash function $\pi : G \to \mathcal{T}$ and the two functions

$$f, f_t \colon \mathcal{T} \to \mathcal{T} \ , \text{ where}$$
$$f(\mathbf{x}) = \pi\left(\prod_{i=1}^n a_i^{x_i}\right) \text{ and } f_t(\mathbf{x}) = \pi\left(t \cdot \prod_{i=1}^n a_i^{-x_i}\right) \ .$$

Note that any representation $(\mathbf{e}_1, \mathbf{e}_2)$ of our desired solution $\mathbf{e}$, i.e. $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$, satisfies

$$\prod_{i=1}^n a_i^{(\mathbf{e}_1)_i} = t \cdot \prod_{i=1}^n a_i^{-(\mathbf{e}_2)_i} \ , \tag{3}$$

and therefore also $f(\mathbf{e}_1) = f_t(\mathbf{e}_2)$. The algorithm now simply searches for collisions $(\mathbf{e}'_1, \mathbf{e}'_2)$ between f and f_t, until $\mathbf{e}'_1 + \mathbf{e}'_2$ yields a solution to the subset sum instance. Algorithm 1 provides a pseudocode description of our BCJ adaptation by using the function definitions of f and f_t to instantiate f_1 and f_2.

Runtime analysis. Notice that while our hash function $\pi : G \to \mathcal{T}$ allows us to iterate the functions f, f_t, it also introduces many useless collisions $(\mathbf{x}, \mathbf{y})$ with

$$\pi\left(\prod_{i=1}^n a_i^{x_i}\right) = \pi\left(t \cdot \prod_{i=1}^n a_i^{-y_i}\right) \text{ but } \prod_{i=1}^n a_i^{x_i} \neq t \cdot \prod_{i=1}^n a_i^{-y_i} \ .$$

However, we already know that any representation $(\mathbf{e}_1, \mathbf{e}_2)$ of the desired solution $\mathbf{e}$ satisfies Equation (3) and thus defines a useful collision. Hence, we can simply compute the probability p that a randomly drawn collision is useful. We expect f, f_t to have $|\mathcal{T}|$ collisions, while the number of representations of $\mathbf{e}$ as weight-$\frac{\omega}{2}n$ vectors $\mathbf{e}_1, \mathbf{e}_2 \in \{0,1\}^n$ is $\binom{\omega n}{\frac{\omega}{2}n} = \tilde{\Theta}(2^{\omega n}) = \tilde{\Theta}(|G|^{\omega})$. This implies

$$p = \tilde{\Theta}\left(\frac{|G|^{\omega}}{|\mathcal{T}|}\right) \ .$$

Hence, after an expected number of p^{-1} iterations we find our desired solution $\mathbf{e}$. Since finding a single collision takes on expectation $\tilde{\Theta}(|\mathcal{T}|^{\frac{1}{2}})$ function evaluations and $|\mathcal{T}| = \tilde{\Theta}(|G|^{H(\frac{\omega}{2})})$, we obtain a total runtime of

$$T = \tilde{\Theta}(|\mathcal{T}|^{\frac{1}{2}} \cdot p^{-1}) = \tilde{\Theta}(|\mathcal{T}|^{\frac{3}{2}} \cdot |G|^{-w}) = \tilde{\Theta}(|G|^{\frac{3}{2}H(\omega/2)-\omega}) \ . \tag{4}$$

In Figure 3 we show our new runtime exponent $\frac{3}{2}H(\omega/2) - \omega$ (called BCJ Adaption), which always improves on the folklore algorithm over the whole range of ω.

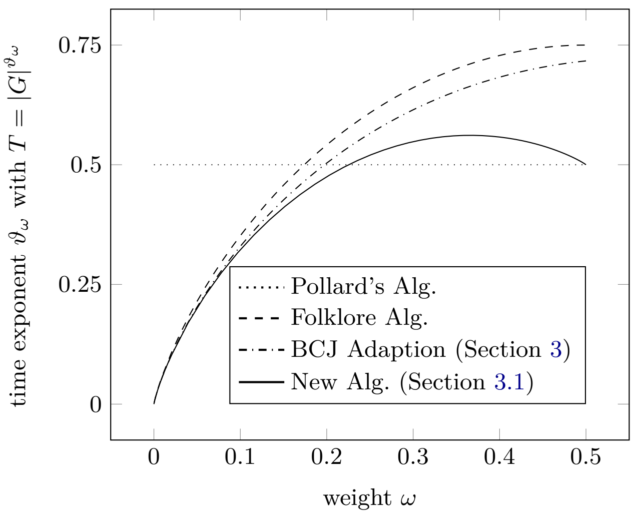


Fig. 3: Comparison of runtime exponents ϑ_ω for low weight DLP algorithms

For group subset sum problems of weight $\frac{n}{2}$, we achieve runtime $|G|^{\frac{3}{2}H(\frac{1}{4})-\frac{1}{2}} = |G|^{0.72}$. This implies a polynomial-space subset algorithm with runtime $2^{0.72n}$, the remarkable result of Becker, Coron and Joux. For the discrete logarithm setting, the result $|G|^{0.72}$ is less remarkable, since Pollard's algorithm already achieves runtime $|G|^{\frac{1}{2}}$. However, for weights $\omega \leq 0.197$ our BCJ adaption improves on Pollard's runtime.

In the subsequent Section 3.1 we further improve on the BCJ adaptation by using the special form $a_i = g^{2^{i-1}}$ in the low weight DLP. Notice that so far representations $(\mathbf{e}_1, \mathbf{e}_2) \in \{0,1\}^n$ of $\mathbf{e}$ in the analysis of the BCJ algorithm fulfill $\mathbf{e}_1 + \mathbf{e}_2 \in \{0,1\}^n$. In other words, in $\mathbf{e}_1, \mathbf{e}_2$ we never have 1-coordinates that add up. However, we also know by Equation (1) that the vectors $\mathbf{e}_1, \mathbf{e}_2$ can be treated as bit-representations of numbers. Hence, if in the i^{th} position, $i < n$, we have a 1-entry in both vectors, then we obtain

$$a_i^2 = (g^{2^{i-1}})^2 = g^{2^i} = a_{i+1}.$$

That is, we can make good use of carry bits to further increase the number of representations of our solution by looking at the total number of sums $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ not only over $\{0,1\}^n$ but over the integers (modulo $|G|$).

3.1 Improved Low Weight DLP Algorithm

Let G be a cyclic group generated by g with order $2^{n-1} \leq |G| < 2^n$. Let $g^\alpha = \beta$ with $\mathrm{wt}(\alpha) = \omega n \leq \frac{n}{2}$ be a low weight DLP in G. Our idea is to represent $\alpha = \alpha_1 + \alpha_2 \mod |G|$ with $\alpha_1, \alpha_2 \in \mathbb{Z}_{|G|}$ both of a certain weight. For ease of notation, we will only concentrate on representations (α_1, α_2) with $\alpha_1 + \alpha_2 < |G|$ such that we can ignore reductions modulo $|G|$. With this simplification, we only lose a constant (namely $\frac{1}{2}$) fraction of all representations, which does not affect asymptotics.

Thus, we first have to determine for which weight of $\phi n = \mathrm{wt}(x_1) = \mathrm{wt}(x_2)$, $x_1, x_2 \in \mathbb{Z}_{|G|}$ we expect that $\mathrm{wt}(x_1 + x_2) = \omega n$. This seems to be a rather natural and fundamental problem to study, but to our surprise we could not find any treatment in the literature. In the following we provide a (heuristic) Markov chain analysis for computing ϕ as a function of ω, which we experimentally validate.

Computation of ϕ via Markov chain. Let us model the bitwise summation $x_1 + x_2$ as a Markov chain. Since in every bit position we also have to take into account a carry bit, every state of our Markov chain contains three bits (b_1, b_2, b_3). Here we denote by b_1 the carry bit and by b_2, b_3 the corresponding bit from x_1 respectively x_2.

To make ourselves a bit more familiar with the notion, let us look at the following example from Figure 4. Let us start in state $(0,0,0)$. As $0+0+0 = 0$, this state produces carry bit 0 and depending on the subsequent bits of x_1, x_2 we may enter one of the four states $(0,0,0), (0,1,0), (0,0,1)$ or $(0,1,1)$.

As x_1, x_2 are uniformly drawn from the set of vectors with weight ϕn, a single bit position in these vectors takes value 1 with probability ϕ. In our analysis, we ignore the effect that the random variables for the bit positions are not independent (since they have to sum to ϕn). This heuristic should only insignificantly affect our asymptotic treatment.

Hence, in our example we stay in state $(0,0,0)$ with probability $(1-\phi)^2$, move to either $(0,0,1)$ or $(0,1,0)$ with probability $\phi(1-\phi)$, and move to $(0,1,1)$ with probability ϕ^2.

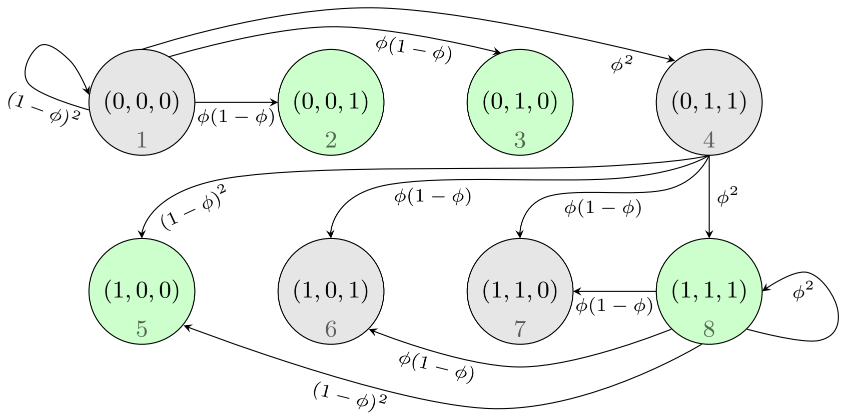


Fig. 4: Possible state transitions from states $(0,0,0)$, $(0,1,1)$ and $(1,1,1)$, where edge labels are transition probabilities and the first bit indicates the carry. Further state transitions are omitted for the sake of clarity.

The complete Markov process is defined by the following transition matrix $M = (m_{i,j})_{1\le i,j\le 8}$, where $m_{i,j}$ is the transition probability from state i to j, where the labels i, j are defined in Figure 4.

$$M := \begin{pmatrix} (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 & 0 & 0 & 0 & 0 \\ (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 & 0 & 0 & 0 & 0 \\ (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 \\ (1-\phi)^2 & \phi(1-\phi) & (\phi)(1-\phi) & \phi^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 \\ 0 & 0 & 0 & 0 & (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 \\ 0 & 0 & 0 & 0 & (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 \end{pmatrix}$$

Note that the states $2,3,5$ and 8 produce a 1 in the corresponding bit of the sum $x_1 + x_2$, while the other states produce a 0. Since $\mathrm{wt}(x_1 + x_2) = \omega n$, we should (asymptotically) produce a 1 with probability ω. Thus, we should be in either of the states $2,3,5$ or 8 with probability ω.

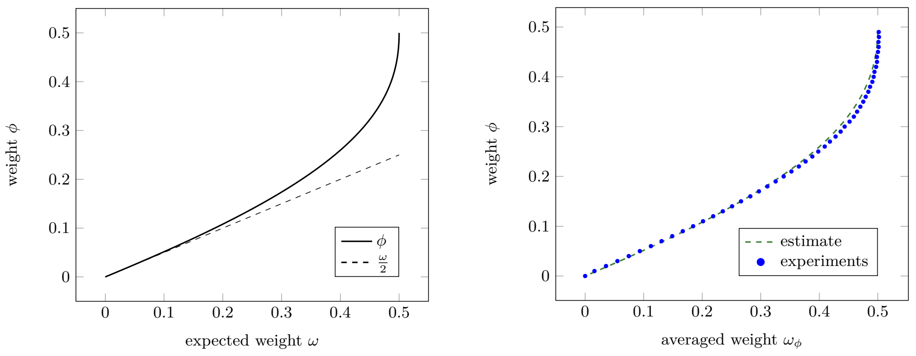
Markov chain theory tells us that M reaches a *stationary distribution* $\pi = (\pi_1, \ldots, \pi_8)$ satisfying $\pi M = \pi$. For each $1 \le i \le 8$, the Markov process (asymptotically) reaches state i with probability π_i. Thus, from the linear equations

$$\pi M = \pi, \quad \pi_1 + \ldots + \pi_8 = 1 \text{ and } \pi_2 + \pi_3 + \pi_5 + \pi_8 = \omega$$

we obtain an expression for ϕ as a function of ω. Computing the stationary distribution yields

$$\begin{aligned}\pi = \quad &(-\phi^4 + 2\phi^3 - 2\phi + 1, \quad \phi^4 - \phi^3 - \phi^2 + \phi, \quad \phi^4 - \phi^3 - \phi^2 + \phi, \quad -\phi^4 + \phi^2, \\ &\phi^4 - 2\phi^3 + \phi^2, \quad -\phi^4 + \phi^3, \quad -\phi^4 + \phi^3, \quad \phi^4) \ .\end{aligned}$$

Hence, it follows that $\omega = 4\phi^4 - 4\phi^3 - \phi^2 + 2\phi$. Solving for ϕ yields our desired function that we illustrate in Figure 5a. In Figure 5b we experimentally verify the accuracy of our asymptotic analysis for concrete sums of 500-bit integers x_1, x_2.



(a) Asymptotic estimate of the expected weight ωn

(b) Experimentally averaged value of ω for $n = 500$ (sample size per ϕ is 1000)

Fig. 5: Asymptotic estimate and experimentally obtained values for the weight ωn of the sum of two n-bit numbers with weight ϕn

Our Low Weight Discrete Logarithm Algorithm. Recall that $g^\alpha = \beta$ with $\mathrm{wt}(\alpha) = \omega \leq \frac{1}{2}$, where g generates a group G of order $2^{n-1} \leq |G| < 2^n$. We represent $\alpha = \alpha_1 + \alpha_2$ with $\mathrm{wt}(\alpha_1) = \mathrm{wt}(\alpha_2) = \phi$, where we compute ϕ as a function of ω as described in the previous paragraph.

Let us define $\mathcal{T} := \{x \in \mathbb{Z}_{2^n} \mid \mathrm{wt}(x) = \phi(\omega)n\}$, where $|\mathcal{T}| = \tilde{\Theta}(2^{H(\phi(\omega))n}) = \tilde{\Theta}(|G|^{H(\phi(\omega))})$. Further, we define a hash function $\pi : G \to \mathcal{T}$ and the two functions

$$f, f_\beta \colon \mathcal{T} \to \mathcal{T} \ \text{, where}$$
$$f(x) = \pi\left(g^x\right) \text{ and } f_t(x) = \pi\left(\beta g^{-x}\right) \ .$$

Any representation (α_1, α_2) of our discrete logarithm α, i.e. $\alpha = \alpha_1 + \alpha_2$, satisfies

$$g^{\alpha_1} = \beta g^{-\alpha_2}, \tag{5}$$

and therefore also $f(\alpha_1) = f_\beta(\alpha_2)$. Our algorithm searches for collisions (x_1, x_2) between f and f_β, until $x_1 + x_2$ yields a solution to the discrete logarithm problem. Algorithm 2 gives a pseudocode description of our new algorithm.

Algorithm 2 Discrete Logarithm Solver

```
Input: functions $f, f_\beta : \mathcal{T} \to \mathcal{T}$, generator $g$ of $G$, $\beta \in G$
Output: $\alpha = \mathrm{dlog}_g \beta$ satisfying $g^\alpha = \beta$
1: repeat
2:     choose random flavour $k$
3:     choose random starting point $s \in \mathcal{T}$
4:     $(x, y) \leftarrow$ Rho$(f^{[k]}, f_\beta^{[k]}, s)$
5:     $\alpha' \leftarrow x + y$
6: until $g^{\alpha'} = \beta$
7: return $\alpha'$
```

Heuristic Analysis of our Algorithm. The hash function $\pi : G \to \mathcal{T}$ produces a lot of useless collisions $f(x_1) = f_\beta(x_2)$ for which $g^{x_1} \neq \beta g^{-x_2}$. However, for any representation (α_1, α_2) of α Equation (5) holds. In order to determine the probability p of a collision (x_1, x_2) being useful – which means $g^{x_1} = \beta g^{-x_2}$ – we compute the number of representations.

We search for our weight-ωn discrete logarithm α by computing sums of weight-$\phi(\omega)n$ numbers $(x_1, x_2) \in \mathcal{T}^2$. Let us heuristically assume that the weights of the resulting sums $x_1 + x_2$ concentrate around weight ωn. Namely, we assume that a polynomial fraction of all sums attains weight ωn. Such concentration results hold for similar distributions like the binomial distribution, and we validate our concentration heuristic experimentally. From Figure 6 we conclude that quite sharply a $\frac{1}{\sqrt{n}}$-fraction of our experiments hits their expectation, exactly the same concentration result that we obtain from the binomial distribution.

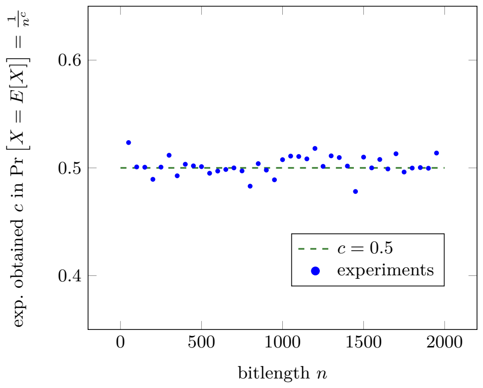


Fig. 6: Experimentally averaged value c in $\Pr\left[X = \mathbb{E}[X]\right] = \frac{1}{n^c}$, when adding two n-bit numbers of weight $\lfloor 0.3n \rfloor$, where X is a random variable for the weight of their sum (sample size per n is 5000).

We further assume that the polynomial fraction of sums $x_1 + x_2$ with weight ωn takes uniformly distributed values among all numbers of weight ωn. Therefore, any random sum $x_1 + x_2$ of weight ωn equals α with probability $\binom{n}{\omega n}^{-1} = \tilde{\Theta}(G^{-H(\omega)})$. This implies that heuristically we obtain $\tilde{\Theta}(|\mathcal{T}|^2 |G|^{-H(\omega)})$ many representations. As we expect a total of $|\mathcal{T}|$ collisions, the probability p of a useful collision is

$$p = \tilde{\Theta}\left(\frac{|\mathcal{T}|^2 |G|^{-H(\omega)}}{|\mathcal{T}|}\right) = \tilde{\Theta}(|\mathcal{T}| \cdot |G|^{-H(\omega)}) \ .$$

Finding each collision heuristically takes time $|\mathcal{T}|^{\frac{1}{2}}$. Since $|\mathcal{T}| = \tilde{\Theta}(|G|^{H(\phi(\omega))})$, we expect to find the low weight discrete logarithm α in time

$$T = \tilde{\Theta}(|\mathcal{T}|^{\frac{1}{2}} \cdot p^{-1}) = \tilde{\Theta}(|\mathcal{T}|^{-\frac{1}{2}} |G|^{H(\omega)}) = \tilde{\Theta}(|G|^{H(\omega) - \frac{1}{2}H(\phi(\omega))}) \ . \qquad (6)$$

The run time exponent of Equation (6) is depicted in Figure 3 (as New Alg.). While our low weight DLP algorithm significantly improves over the folklore algorithm and the BCJ adaptation, it does not yet achieve the square root of the search space S. Namely, if S denotes the set of all weight-ωn numbers, i.e. $|S| = \binom{n}{\omega n} = \tilde{\Theta}(|G|^{H(\omega)})$, then we might hope for a polynomial space algorithm with time complexity $|S|^{\frac{1}{2}}$, as a possible generalization of Pollard's algorithm to the low weight discrete logarithm regime.

In Figure 7 we illustrate all runtime exponents to the base $|S|$. That is for Pollard's algorithm, the folklore algorithm (Equation (2)), the BCJ algorithm (Equation (4)), and our new algorithm (Equation (6)) we depict their exponents

$$\frac{1}{2H(\omega)}, \ \frac{3}{4}, \ \frac{\frac{3}{2}H(\omega/2) - \omega}{H(\omega)}, \ \frac{H(\omega) - \frac{1}{2}H(\phi(\omega))}{H(\omega)}.$$

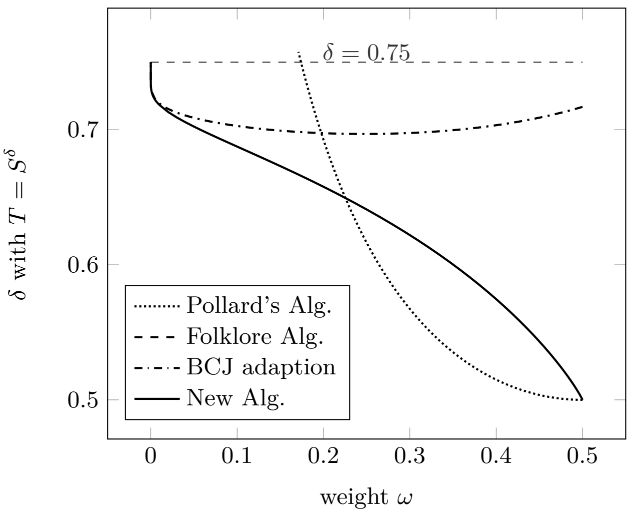


Fig. 7: Runtime exponent of the search space for low hamming weight DLP

The folklore algorithm has a constant exponent $\frac{3}{4}$ in the search space. Pollard's algorithm is superior for large weights $\omega \geq 0.174$, but also gets arbitrarily bad in the search space for small ω. Our BCJ adaptation achieves the first improvement over $\frac{3}{4}$ for *arbitrary* weights ω. However, BCJ only reaches a minimal exponent of $\delta = 0.697$ (achieved at $\omega = 0.248$). Eventually, our new algorithm improves on the BCJ algorithm for all weights, where we obtain an interpolation between $\frac{3}{4}$ for arbitrary small weights and the desired optimum $\frac{1}{2}$ for $\omega = \frac{1}{2}$. Additionally, our new algorithm is superior to Pollard's algorithm for all weights $\omega \leq 0.225$.

Experimental verification of our algorithm. For experimental convenience, we implemented our algorithm in the multiplicative group $G = \mathbb{Z}_p^*$. We chose bit-length $n = 40$, and p as the largest prime smaller than 2^{40}. We generated 40 random small weight DLP instances for each weight $\omega n \in \{2, 3, \dots, 20\}$. We measured T_f, the amount of calls to our function f, and averaged T_f over all 40 instances.[1]

Figure 8 shows the results of our computations. Here the dots are the values obtained in our experiments, while the dashed line is our asymptotic prediction $40(H(\omega) - \frac{1}{2}H(\phi(\omega)))$, shifted by 2.92 to take some (in the analysis neglected) polynomial run time factor into account.

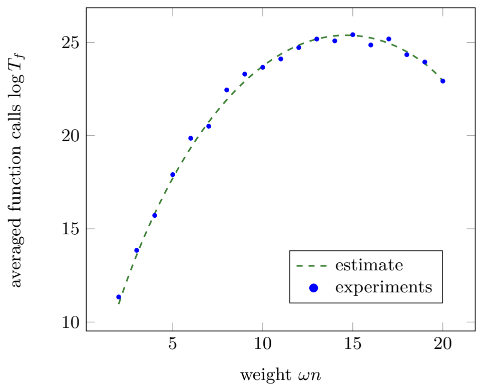


Fig. 8: Experimentally averaged number of function calls (in logarithmic scale) needed to solve a 40-bit discrete logarithm problem of weight ωn. Sample size per ωn is 40.

Time-Memory Tradeoff for Reaching the Square Root Bound. Let $|S| = \binom{n}{\omega n} = \tilde{\Theta}(|G|^{H(\omega)})$ be the low weight DLP search space as defined before. As we have seen, it remains open to reach square root complexity $|S|^{\frac{1}{2}}$ with a polynomial space algorithm, but our new algorithm makes a significant step towards this goal.

[1] implementation available at https://github.com/LwDLPandSubsetSum/lwDLP-and-NestedSubsetSum

Let us at this point forget about our polynomial space restriction (for the first and only time in this paper). Then Coppersmith's Baby-Step Giant-Step variant for low weight DLP(see [12], Chapter 13) achieves both time and space complexity $T_{\text{BSGS}} = M_{\text{BSGS}} = \tilde{\Theta}(|S|^{\frac{1}{2}})$.

Fortunately, our BCJ adaption as well as our new algorithm also allow for a time-memory tradeoff using van Oorschot-Wiener's Parallel Collision Search (PCS) [29]. Let C be the time complexity to find a collision with polynomial memory, then PCS finds 2^m collisions in time $\tilde{\Theta}(2^{\frac{m}{2}}C)$ using $\tilde{\Theta}(2^m)$ memory.

In the following, we minimize the run time of BCJ and our new algorithm by applying PCS.

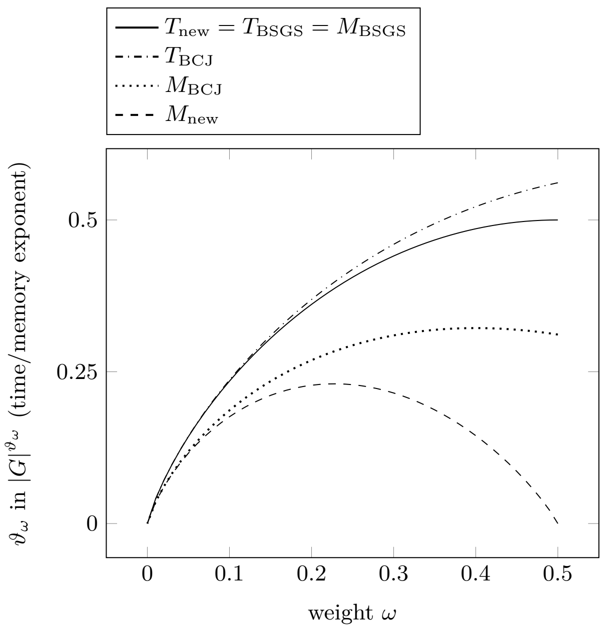


Fig. 9: Time-Memory tradeoffs when applying PCS to our algorithms.

BCJ Tradeoff. From the analysis in Section 3, our BCJ adaptation requires to find an expected number of $\tilde{\Theta}(|G|^{H(\omega/2)-\omega})$ collisions, each at the cost of $\tilde{\Theta}(|G|^{\frac{H(\omega/2)}{2}})$. Thus, using memory $M_{\text{BCJ}} = \tilde{\Theta}(|G|^{H(\omega/2)-\omega})$ BCJ's time complexity decreases to

$$T_{\text{BCJ}} = \tilde{\Theta}(|G|^{\frac{H(\omega/2)-\omega}{2}} \cdot |G|^{\frac{H(\omega/2)}{2}}) = \tilde{\Theta}(|G|^{H(\frac{\omega}{2})-\frac{\omega}{2}}) \ .$$

The time exponent is illustrated in Figure 9 as dash-dotted line, the memory exponent as dotted line, both as a function of ω.

New Algorithm Tradeoff. Following the analysis in Section 3.1 our new algorithm requires to find $\tilde{\Theta}(|G|^{H(\omega)-H(\phi(\omega))})$ collisions, each at the cost of $\tilde{\Theta}(|G|^{\frac{H(\phi(\omega))}{2}})$. Hence, using memory $M_{\text{new}} = \tilde{\Theta}(|G|^{H(\omega)-H(\phi(\omega))})$ via PCS yields a runtime of

$$T_{\text{new}} = \tilde{\Theta}(|G|^{\frac{H(\omega)-H(\phi(\omega))}{2}} \cdot |G|^{\frac{H(\phi(\omega))}{2}}) = \tilde{\Theta}(|G|^{\frac{H(\omega)}{2}}) = \tilde{\Theta}(|S|^{\frac{1}{2}}) \ .$$

Therefore – as opposed to the BCJ adaptation – our new low weight DLP algorithm achieves the optimal time bound $\tilde{\Theta}(|S|^{\frac{1}{2}})$ for collision search algorithms. In comparison to Coppersmith's Baby-Step Giant-Step algorithm it does not require full memory $\tilde{\Theta}(|S|^{\frac{1}{2}})$, but only memory $\tilde{\Theta}(|G|^{H(\omega)-H(\phi(\omega))})$. We illustrate the exponent $H(\omega) - H(\phi(\omega))$ as a dashed line in Figure 9.

4 Subset Sum in $2^{0.65n}$ with Polynomial Space

Motivation. Let $(\mathbf{a}, t) = (a_1, \ldots, a_n, t) \in (\mathbb{Z}_{2^n})^{n+1}$ be a weight-$\frac{1}{2}$ instance of the subset sum problem, i.e., there exists some solution $\mathbf{e} = (e_1, \ldots, e_n)$ with $\text{wt}(\mathbf{e}) = \frac{n}{2}$ satisfying $\langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n$. Our algorithm extends to all weights ω, but for simplicity we analyze in the following only the worst-case $\omega = \frac{1}{2}$.

The folklore algorithm from Section 3 (also stated explicitly for subset sum in [4]) has runtime $2^{\frac{3}{4}n}$. This is achieved by choosing $\mathcal{T} := \{\mathbf{x} \in \{0,1\}^{\frac{n}{2}} \mid \text{wt}(\mathbf{x}) = \frac{n}{4}\}$ with $|\mathcal{T}| = \tilde{\Theta}(2^{\frac{n}{2}})$, defining an injective function $h \colon \mathbb{Z}_{2^{n/2}} \to \mathcal{T}$ and searching for collisions between

$$\begin{aligned} f(\mathbf{x}) &= h(\langle (a_1, \ldots, a_{n/2}), \mathbf{x} \rangle \bmod 2^{\frac{n}{2}}) \text{ and} \\ f_t(\mathbf{x}) &= h(t - \langle (a_{n/2+1}, \ldots, a_n), \mathbf{x} \rangle \bmod 2^{\frac{n}{2}}). \end{aligned}$$

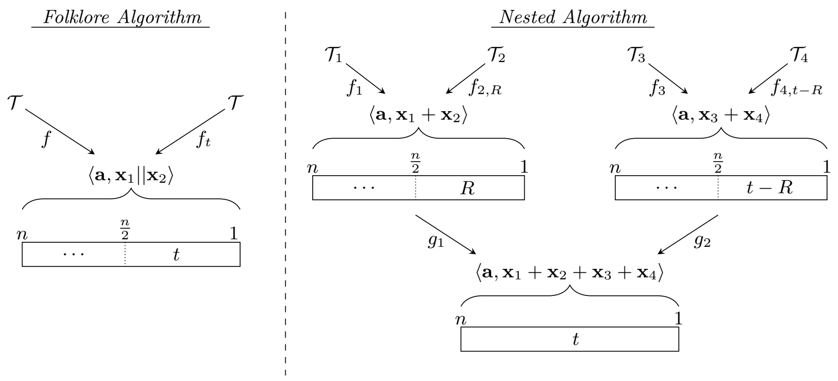
With the notation from Section 3 our hash function π first applies the canonical group homomorphism $(\mathbb{Z}_{2^n}, +) \to (\mathbb{Z}_{2^{n/2}}, +)$, followed by an application of h.

Any collision $(\mathbf{x}_1, \mathbf{x}_2)$ satisfies

$$h(\langle (a_1, \ldots, a_{n/2}), \mathbf{x}_1 \rangle \bmod 2^{\frac{n}{2}}) = h(t - \langle (a_{n/2+1}, \ldots, a_n), \mathbf{x}_2 \rangle \bmod 2^{\frac{n}{2}}) \ .$$

Since h is injective, we conclude that $\langle \mathbf{a}, \mathbf{x}_1 || \mathbf{x}_2 \rangle = t \bmod 2^{\frac{n}{2}}$. Thus $\mathbf{x} = \mathbf{x}_1 || \mathbf{x}_2$ is a *potential solution* that matches t already on $\frac{n}{2}$ bits, see also Figure 10. Any potential solution can be found in time $2^{\frac{n}{4}}$ via collision search. However, it costs us on expectation $2^{\frac{n}{2}}$ iterations to find a useful solution that also matches t on the remaining $\frac{n}{2}$ bits. Put differently, we use a square-root cycle finding algorithm to find potential solutions, whereas we use a naive brute-force routine to identify a useful solution. The conflicting problem is that π hashes down to $\frac{n}{2}$ bits to allow for an iterative function application, but thereby inherently introduces $2^{\frac{n}{2}}$ useless collisions.

Our high-level idea. Our goal is to use a nested collision search to first find potential solutions that match on $\frac{n}{2}$ bits, and then among these collisions use another collision search for identifying some useful solution. This introduces a



Folklore Algorithm
$\mathcal{T}$
f
$\mathcal{T}$
f_t
$\langle \mathbf{a}, \mathbf{x}_1 || \mathbf{x}_2 \rangle$
n
$\frac{n}{2}$
1
$\cdots$
t
Nested Algorithm
$\mathcal{T}_1$
f_1
$\mathcal{T}_2$
$f_{2,R}$
$\mathcal{T}_3$
f_3
$\mathcal{T}_4$
$f_{4,t-R}$
$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 \rangle$
$\langle \mathbf{a}, \mathbf{x}_3 + \mathbf{x}_4 \rangle$
R
$t - R$
g_1
g_2
$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 \rangle$
t

Fig. 10: Basic Structure of the folklore and our new algorithm for solving subset sum.

two-layer approach, see also Figure 10, for which we need to split our solution $\mathbf{e}$ into four summands $\mathbf{e} = \mathbf{x}_1 + \ldots + \mathbf{x}_4$. Obviously, $\langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n$ implies

$$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 \rangle = t - \langle \mathbf{a}, \mathbf{x}_3 + \mathbf{x}_4 \rangle \bmod 2^{\frac{n}{2}}.$$

On layer 1, our algorithm fixes some $R \in \mathbb{Z}_{2^{n/2}}$ and finds collisions $(\mathbf{x}_1, \mathbf{x}_2)$ satisfying $\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 \rangle = R \bmod 2^{\frac{n}{2}}$ as well as collisions $(\mathbf{x}_3, \mathbf{x}_4)$ satisfying $\langle \mathbf{a}, \mathbf{x}_3 + \mathbf{x}_4 \rangle = t - R \bmod 2^{\frac{n}{2}}$.

On layer 2, we search among the collisions $(\mathbf{x}_1, \mathbf{x}_2)$ and $(\mathbf{x}_3, \mathbf{x}_4)$ via some nested collision search for some collision that satisfies the identity $\langle \mathbf{a}, \mathbf{x}_1 + \ldots + \mathbf{x}_4 \rangle$ also on the remaining $\frac{n}{2}$ bits.

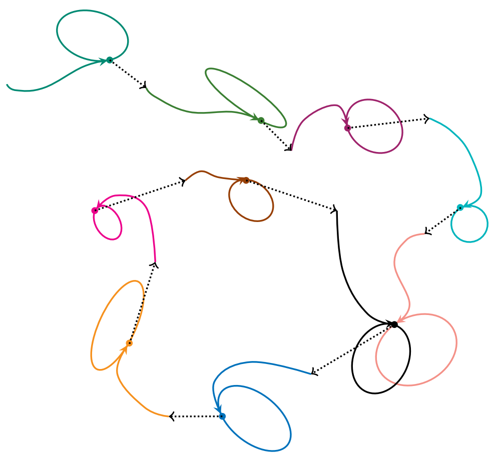


Fig. 11: Structure of the nested Rho algorithm, where different flavours of the function are represented by different colours

Our technique for subset sum induces a giant Rho structure (layer 2 collision search) over smaller Rho structures (layer 1 collision search), as illustrated in Figure 11. Here different colours represent a collision search on different function flavours, as defined in Section 2. The dotted arrows depict the transition from a collision to a new starting point.

4.1 Nested Collision Subset Sum in $2^{0.649n}$

To explain our algorithm from Figure 10 in more detail, we have to first specify the domains $\mathcal{T}_1, \ldots, \mathcal{T}_4$ of the layer-1 functions $f_1, f_{2,R}, f_3, f_{4,t-R}$. We illustrate the domains in Figure 12.
Let us denote by

$$\mathcal{B}(n, \beta) := \{\mathbf{x} \in \{0,1\}^n \mid \mathrm{wt}(\mathbf{x}) = \beta n\}$$

the set of n dimensional vectors with relative (to n) weight β. For some $\gamma \in [0,1]$ we define

$$\begin{aligned}
\mathcal{T}_1 &= 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\frac{1}{4}(1-\gamma)n, \frac{1}{2}\right) \times \mathcal{B}\left(\gamma n, \frac{1}{8}\right) \\
\mathcal{T}_2 &= 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\frac{1}{4}(1-\gamma)n, \frac{1}{2}\right) \times 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\gamma n, \frac{1}{8}\right) \\
\mathcal{T}_3 &= 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\frac{1}{4}(1-\gamma)n, \frac{1}{2}\right) \times 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\gamma n, \frac{1}{8}\right) \\
\mathcal{T}_4 &= \mathcal{B}\left(\frac{1}{4}(1-\gamma)n, \frac{1}{2}\right) \times 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\gamma n, \frac{1}{8}\right) \quad .
\end{aligned} \tag{7}$$

Notice that $|\mathcal{B}(n,\beta)| = \binom{n}{\beta n} = \tilde{\Theta}(2^{H(\beta)n})$. Therefore, all $\mathcal{T}_i$ satisfy

$$|\mathcal{T}_i| = \tilde{\Theta}(2^{(\frac{1-\gamma}{4} + H(\frac{1}{8})\gamma)n}).$$

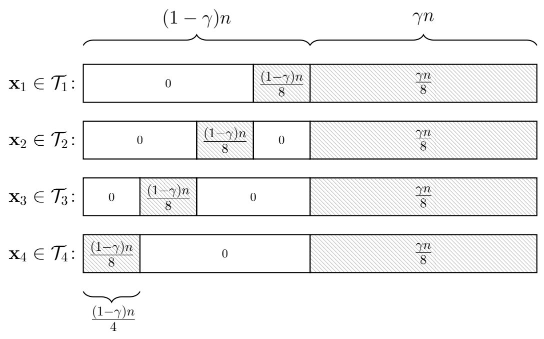
Since we want to have function domain $\frac{n}{2}$ for both layers, we set γ as the solution of $\frac{1-\gamma}{4} + H(\frac{1}{8})\gamma = \frac{1}{2}$, that is

$$\gamma \approx 0.8516. \tag{8}$$

Recall that we represent our subset sum solution $\mathbf{e}$ as $\mathbf{e} = \mathbf{e}_1 + \ldots + \mathbf{e}_4$ with $(\mathbf{e}_1, \ldots, \mathbf{e}_4) \in \mathcal{T}_1 \times \ldots \times \mathcal{T}_4$. By our definition of the $\mathcal{T}_i$ we may write $\mathbf{e} \in \left(\{0,1\}^{\frac{(1-\gamma)n}{4}}\right)^4 \times \{0,1\}^{\gamma n}$, where each of its 5 parts has relative weight $\frac{1}{2}$. Such a weight distribution of $\mathbf{e}$ can be enforced by an initial permutation on the a_i.

Let us fix some $R \in \mathbb{Z}_{2^{n/2}}$, and let $h_i : \mathbb{Z}_{2^{n/2}} \to \mathcal{T}_i$, $i = 1, \ldots, 4$, be injective functions. Layer-1 collisions are defined as elements $(\mathbf{x}_1, \mathbf{x}_2) \in \mathcal{T}_1 \times \mathcal{T}_2$ and $(\mathbf{x}_3, \mathbf{x}_4) \in \mathcal{T}_3 \times \mathcal{T}_4$ satisfying

$$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 \rangle = R \bmod 2^{\frac{n}{2}} \text{ and } \langle \mathbf{a}, \mathbf{x}_3 + \mathbf{x}_4 \rangle = t - R \bmod 2^{\frac{n}{2}} \quad . \tag{9}$$



$(1-\gamma)n$
γn
$\mathbf{x}_1 \in \mathcal{T}_1$:
$\mathbf{x}_2 \in \mathcal{T}_2$:
$\mathbf{x}_3 \in \mathcal{T}_3$:
$\mathbf{x}_4 \in \mathcal{T}_4$:
0
$\frac{(1-\gamma)n}{8}$
$\frac{\gamma n}{8}$
$\frac{(1-\gamma)n}{4}$

Fig. 12: Weight distribution of vectors from first level domains. Shaded areas contain weight, white areas are all zero.

Therefore, we define the following four first layer functions $f_{i,\cdot} : \mathcal{T}_i \to \mathbb{Z}_{2^{n/2}}$ with

$$f_1(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \bmod 2^{\frac{n}{2}}, \; f_{2,R}(\mathbf{x}) = R - \langle \mathbf{a}, \mathbf{x} \rangle \bmod 2^{\frac{n}{2}} \text{ and}$$
$$f_3(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \bmod 2^{\frac{n}{2}}, \; f_{4,t-R}(\mathbf{x}) = t - R - \langle \mathbf{a}, \mathbf{x} \rangle \bmod 2^{\frac{n}{2}} \quad .$$

As a consequence, layer-1 collisions $f_1(\mathbf{x}_1) = f_{2,R}(\mathbf{x}_2)$ and $f_3(\mathbf{x}_3) = f_{4,t-R}(\mathbf{x}_4)$ satisfy Equation (9). Also notice that by Equation (9) any pair of layer-1 collisions $(\mathbf{x}_1, \mathbf{x}_2) \in \mathcal{T}_1 \times \mathcal{T}_2$, $(\mathbf{x}_3, \mathbf{x}_4) \in \mathcal{T}_3 \times \mathcal{T}_4$ satisfies

$$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 \rangle = t \bmod 2^{\frac{n}{2}} \quad .$$

Layer-2 collisions are now defined as pairs of layer-1 collisions $(\mathbf{x}_1, \mathbf{x}_2) \in \mathcal{T}_1 \times \mathcal{T}_2$ and $(\mathbf{x}_3, \mathbf{x}_4) \in \mathcal{T}_3 \times \mathcal{T}_4$ satisfying

$$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 \rangle = t \bmod 2^n \quad .$$

Since we already know that by construction layer-1 collisions satisfy the identity $\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 \rangle = t$ on the lower $n/2$ bits, it suffices to check for layer-2 collisions the identity on the upper $n/2$ bits, which we denote by

$$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 \rangle_{[n/2+1,n]} = t_{[n/2+1,n]} \quad . \tag{10}$$

Recall from Section 2 that $(\mathbf{x}_1, \mathbf{x}_2) = \texttt{Rho}(f_1, f_{2,R}, x)$ denotes the application of a collision finding algorithm on f_1, f_2 with starting point x. The starting point $x \in \mathbb{Z}_{2^{n/2}}$ fully determines the collision $(\mathbf{x}_1, \mathbf{x}_2) \in \mathcal{T}_1 \times \mathcal{T}_2$ found by Rho. Before we apply functions f_1 respectively $f_{2,R}$ iteratively on x, we map x (and all function outputs) via h_1 respectively h_2 to their domains $\mathcal{T}_1$ respectively $\mathcal{T}_2$. An analogous mapping is done for the collision search between f_3 and $f_{4,t-R}$.

Let us define the layer-2 functions $g_1, g_2 \colon \mathbb{Z}_{2^{n/2}} \to \mathbb{Z}_{2^{n/2}}$ as

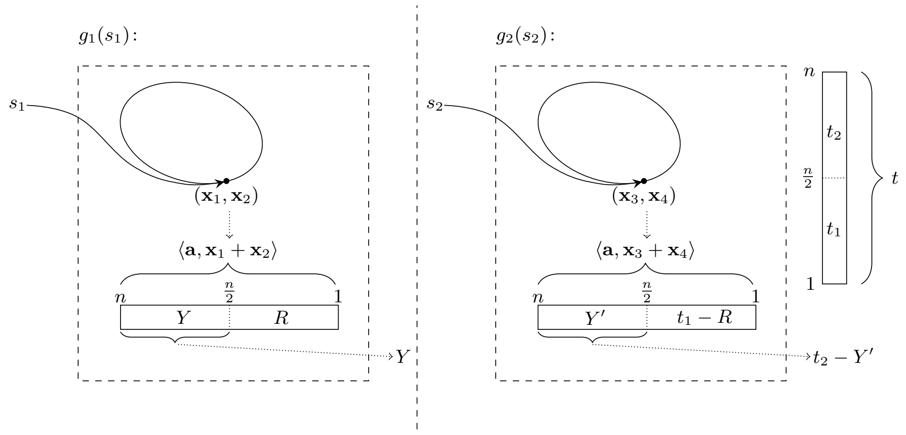
$$g_1(x) := \langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 \rangle_{[n/2+1,n]} \qquad , \text{ where } (\mathbf{x}_1, \mathbf{x}_2) = \texttt{Rho}(f_1, f_{2,R}, x) \text{ and}$$
$$g_2(x) := (t - \langle \mathbf{a}, \mathbf{x}_3 + \mathbf{x}_4 \rangle)_{[n/2+1,n]}, \text{ where } (\mathbf{x}_3, \mathbf{x}_4) = \texttt{Rho}(f_3, f_{4,t-R}, x) \quad .$$

Assume that we found a layer-2 collision (s_1, s_2). We compute from (s_1, s_2) the values $(\mathbf{x}_1, \mathbf{x}_2) = \texttt{Rho}(f_1, f_{2,R}, s_1)$ and $(\mathbf{x}_3, \mathbf{x}_4) = \texttt{Rho}(f_3, f_{4,t-R}, s_2)$. Since (s_1, s_2) is a layer-2 collision we have $g_1(s_1) = g_2(s_2)$ and therefore

$$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 \rangle_{[n/2+1,n]} = (t - \langle \mathbf{a}, \mathbf{x}_3 + \mathbf{x}_4 \rangle)_{[n/2+1,n]} \ .$$

This identity implies Equation (10). Thus, $\mathbf{e} = \mathbf{x}_1 + \ldots + \mathbf{x}_4$ is a solution to the subset sum problem if $\mathbf{e} \in \{0,1\}^n$.

The computation of our layer-2 functions is illustrated in Figure 13. Our whole algorithm Nested Collision Subset Sum is summarized in Algorithm 3.



$g_1(s_1)$:
s_1
$(\mathbf{x}_1, \mathbf{x}_2)$
$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 \rangle$
n
$\frac{n}{2}$
1
Y
R
Y
$g_2(s_2)$:
s_2
$(\mathbf{x}_3, \mathbf{x}_4)$
$\langle \mathbf{a}, \mathbf{x}_3 + \mathbf{x}_4 \rangle$
n
$\frac{n}{2}$
1
Y'
$t_1 - R$
$t_2 - Y'$
n
t_2
$\frac{n}{2}$
t_1
1
t

Fig. 13: Computation of layer-2 functions $g_1(s_1)$ and $g_2(s_2)$, where $(\mathbf{x}_1, \mathbf{x}_2) = \texttt{Rho}(f_1, f_{2,R}, s_1)$ and $(\mathbf{x}_3, \mathbf{x}_4) = \texttt{Rho}(f_3, f_{4,t-R}, s_2)$.

Algorithm 3 Nested Collision Subset Sum

```
Input: subset sum instance $(\mathbf{a}, t) = (a_1, \ldots, a_n, t) \in \mathbb{Z}_{2^n}^{n+1}$
Output: solution $\mathbf{e} \in \{0,1\}^n$ satisfying $\langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n$
1: repeat
2:     Randomly permute the $a_i$.
3:     Choose $R, z \in \mathbb{Z}_{2^{n/2}}$ randomly.
4:     $(s_1, s_2) \leftarrow \texttt{Rho}(g_1, g_2, z)$
5:     Compute $(\mathbf{x}_1, \mathbf{x}_2) = \texttt{Rho}(f_1, f_{2,R}, s_1)$.
6:     Compute $(\mathbf{x}_3, \mathbf{x}_4) = \texttt{Rho}(f_3, f_{4,t-R}, s_2)$.
7:     Set $\mathbf{e} = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4$.
8: until $\mathbf{e} \in \{0,1\}^n$
```

Remark 4.1. We have to guarantee independence of the collisions returned by `Rho` on input g_1, g_2 for different starting points. This can be done (heuristically) by using flavoured inner functions (see Section 2). More formally, we have to change the definitions to

$$g_1(x) := \langle \mathbf{a}, \mathbf{x}_1 + \mathbf{x}_2 \rangle_{[n/2+1,n]} \quad , \text{ where } (\mathbf{x}_1, \mathbf{x}_2) = \texttt{Rho}(f_1^{[x]}, f_{2,R}^{[x]}, x) \text{ and}$$
$$g_2(x) := (t - \langle \mathbf{a}, \mathbf{x}_3 + \mathbf{x}_4 \rangle)_{[n/2+1,n]}, \text{ where } (\mathbf{x}_3, \mathbf{x}_4) = \texttt{Rho}(f_3^{[x]}, f_{4,t-R}^{[x]}, x) \ .$$

In the following, we omit flavours for ease of notation.

Run Time Analysis of Nested Collision Subset Sum. The cost of any iteration of the **repeat**-loop in Nested Collision Subset Sum is dominated by the function call to $\texttt{Rho}(g_1, g_2, z)$, which itself recursively calls $\texttt{Rho}(f_1, f_{2,R}, \cdot)$ and $\texttt{Rho}(f_3, f_{4,t-R}, \cdot)$. Each invocation of collision finding in the layer-1 functions costs time $|\mathcal{T}_i|^{\frac{1}{2}} = \tilde{\Theta}(2^{\frac{n}{4}})$. Since $g_i : \mathbb{Z}_{2^{\frac{n}{2}}} \rightarrow \mathbb{Z}_{2^{\frac{n}{2}}}$, a collision search in the layer-2 functions requires on expectation $\tilde{\Theta}(2^{\frac{n}{4}})$ function evaluations of the g_i. Hence in total each iteration in Nested Collision Subset Sum requires time $\tilde{\Theta}(2^{\frac{n}{2}})$.

Notice that each iteration computes some potential solution $\mathbf{e}'$ satisfying $\langle \mathbf{a}, \mathbf{e}' \rangle = t$, no matter whether the permutation of the a_i induced the correct weight distribution on $\mathbf{e}$. However, such an $\mathbf{e}'$ is usually not in $\{0,1\}^n$, and therefore does not solve our subset sum instance.

Let us look at some fixed iteration of Nested Collision Subset Sum. Let E_1 be the event that our initial permutation yields the correct weight distribution in this iteration. Then

$$p_1 := \Pr[E_1] = \frac{\binom{(1-\gamma)n/4}{(1-\gamma)n/8}^4 \binom{\gamma n}{\gamma n/2}}{\binom{n}{n/2}} = \frac{1}{\text{poly}(n)} = \tilde{\Theta}(1) \ . \tag{11}$$

Let $(\mathbf{e}_1, \ldots, \mathbf{e}_4)$ be a representation of our subset sum solution $\mathbf{e}$ satisfying $\langle \mathbf{a}, \mathbf{e}_1 + \mathbf{e}_2 \rangle = R \mod 2^{\frac{n}{2}}$ (for the choice of R in line 3 of Algorithm 3). Then we call $(\mathbf{e}_1, \mathbf{e}_2)$ a *useful collision* of $f_1, f_{2,R}$. By construction, $(\mathbf{e}_3, \mathbf{e}_4)$ is automatically a useful collision of $f_3, f_{4,t-R}$ satisfying $\langle \mathbf{a}, \mathbf{e}_3 + \mathbf{e}_4 \rangle = t - R \mod 2^{n/2}$. Now, for all useful collisions $(\mathbf{e}_1, \mathbf{e}_2)$ of $f_1, f_{2,R}$ and $(\mathbf{e}_3, \mathbf{e}_4)$ of $f_3, f_{4,t-R}$ there exists some collision (s_1', s_2') of g_1, g_2 satisying $(\mathbf{e}_1, \mathbf{e}_2) = \texttt{Rho}(f_1, f_{2,R}, s_1')$ and $(\mathbf{e}_3, \mathbf{e}_4) = \texttt{Rho}(f_3, f_{4,t-R}, s_2')$. Thus, useful collisions of $f_1, f_{2,R}$ and $f_3, f_{4,t-R}$ are in 1:1-correspondence with the collisions of g_1, g_2 that yield a representation of the solution. Hence, we can compute the probability of success in one iteration given E_1 as the fraction of useful collisions with respect to R among all collisions of g_1, g_2, where the latter is $\tilde{\Theta}(2^{\frac{n}{2}})$.

Let E_2 be the event that there exist useful collisions for our choice of R. Let E_3 be the event that our collision finding yields a representation $(\mathbf{x}_1, \ldots, \mathbf{x}_4)$ of the solution $\mathbf{e}$. Then we succeed in this iteration with probability $p := \Pr[E_1 \cap E_2 \cap E_3] = \Pr[E_1] \cdot \Pr[E_2 \mid E_1] \cdot \Pr[E_3 \mid E_2 \cap E_1]$. It remains to compute $p_2 := \Pr[E_2 \mid E_1]$ and $p_3 := \Pr[E_3 \mid E_2 \cap E_1]$. Let us start with probability p_2.

Let us calculate the number of different R values for which we obtain useful collisions. We first observe that different representations $(\mathbf{e}_1, \ldots, \mathbf{e}_4)$ might share the same value $\mathbf{e}_1 + \mathbf{e}_2$ and hence the same inner product $\langle \mathbf{a}, \mathbf{e}_1 + \mathbf{e}_2 \rangle$. Thus, we have to count the number of distinct representations $(\mathbf{e}'_1, \mathbf{e}'_2) = (\mathbf{e}_1 + \mathbf{e}_2, \mathbf{e}_3 + \mathbf{e}_4)$ of $\mathbf{e}$. By the definition of our function domains in Equation (7) and in Figure 12 this number is

$$\binom{\frac{\gamma n}{2}}{\frac{\gamma n}{4}} = \tilde{\Theta}\left(2^{\frac{\gamma n}{2}}\right) .$$

Hence, the probability of choosing an $R \in \mathbb{Z}_{2^{n/2}}$ for which useful collisions exist is

$$p_2 := \tilde{\Theta}\left(\frac{2^{\frac{\gamma n}{2}}}{2^{\frac{n}{2}}}\right) = \tilde{\Theta}\left(2^{\frac{(\gamma-1)n}{2}}\right) .$$

Note that for a good choice of R there directly exist several useful collisions, since any fixed $(\mathbf{e}'_1, \mathbf{e}'_2) = (\mathbf{e}_1 + \mathbf{e}_2, \mathbf{e}_3 + \mathbf{e}_4)$ is represented via multiple $(\mathbf{e}_1, \ldots, \mathbf{e}_4)$. More precisely every $\mathbf{e}'_1$ (resp. $\mathbf{e}'_2$) is represented by

$$\binom{\frac{\gamma n}{4}}{\frac{\gamma n}{8}} = \tilde{\Theta}\left(2^{\frac{\gamma n}{4}}\right)$$

different $(\mathbf{e}_1, \mathbf{e}_2)$ (resp. $(\mathbf{e}_3, \mathbf{e}_4)$). Note that any of these $(\mathbf{e}_1, \mathbf{e}_2)$ and $(\mathbf{e}_3, \mathbf{e}_4)$ form useful collisions of $f_1, f_{2,R}$ and $f_3, f_{4,t-R}$. Furthermore, any of the $2^{\frac{\gamma n}{2}}$ combinations of $(\mathbf{e}_1, \mathbf{e}_2)$ and $(\mathbf{e}_3, \mathbf{e}_4)$ is a representation of $\mathbf{e}$. Therefore, we obtain a total of $2^{\frac{\gamma n}{2}}$ distinct collisions in g_1, g_2 that represent $\mathbf{e}$. Thus in case that we made a good choice of R, a random collision is a representation of the solution with probability

$$p_3 := \tilde{\Theta}\left(\frac{2^{\frac{\gamma n}{2}}}{2^{\frac{n}{2}}}\right) = \tilde{\Theta}\left(2^{\frac{(\gamma-1)n}{2}}\right) .$$

Eventually, we expect $p^{-1} = (p_1 p_2 p_3)^{-1} = 2^{(1-\gamma)n} = 2^{0.149n}$ iterations with cost each $\tilde{\Theta}(2^{\frac{n}{2}})$, resulting in total expected runtime

$$T = \tilde{\Theta}(2^{(\frac{3}{2}-\gamma)n}) = 2^{0.649n} .$$

Alternative Run Time Analysis of Nested Collision Subset Sum. We already saw that each iteration of Nested Collision Subset Sum takes time $\tilde{\Theta}(2^{n/2})$ and we have to iterate until we find some $\mathbf{e} \in \{0,1\}^n$. We call $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \in \mathcal{T}_1 \times \ldots \times \mathcal{T}_4$ *consistent* iff $\mathbf{e} = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 \in \{0,1\}^n$.

Now observe that a random tuple $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \in \mathcal{T}_1 \times \ldots \times \mathcal{T}_4$ is consistent with probability

$$\begin{aligned} p &= \frac{\binom{\gamma n}{\gamma n/8}\binom{7\gamma n/8}{\gamma n/8}\binom{6\gamma n/8}{\gamma n/8}\binom{5\gamma n/8}{\gamma n/8}}{\binom{\gamma n}{\gamma n/8}^4} \\ &= \tilde{\Theta}\left(2^{\left(\frac{7}{8}H\left(\frac{1}{7}\right)+\frac{3}{4}H\left(\frac{1}{6}\right)+\frac{5}{8}H\left(\frac{1}{5}\right)-3H\left(\frac{1}{8}\right)\right)\gamma n}\right) \geq 2^{-0.149n} . \end{aligned}$$

Let us assume that the representations of $\mathbf{e}$ distribute uniformly in $\mathcal{T}_1 \times \ldots \times \mathcal{T}_4$ and that Nested Collision Subset Sum finds random collisions. Then we need on expection p^{-1} iterations until we find $\mathbf{e} \in \{0,1\}^n$, resulting in a total runtime of

$$T = p^{-1} \cdot \tilde{\Theta}(2^{n/2}) = 2^{0.649n}.$$

This view on the runtime of Nested Collision Subset Sum motivates the improved algorithm in the subsequent section that increases the probability of obtaining a consistent vector $\mathbf{e}$ at the cost of an initial exponential time permutation step.

4.2 Improved Nested Collision Subset Sum in $2^{0.645n}$

Recall from Equation (7) and Figure 12 that in $\mathbf{x}_1 + \ldots + \mathbf{x}_4$ the left-most coordinates are always in $\{0,1\}^{(1-\gamma)n}$. In other words, inconsistencies are always due to the last γn coordinates. Therefore, our goal is to shift less weight in the last γn coordinates. Namely, we modify the weight distribution of $\mathbf{e}$ such that the last γn coordinates have relative weight $\frac{\beta}{2}$ for some $2 - \frac{1}{\gamma} \leq \beta \leq 1$. We depict our new weight distribution in Figure 14. Since we cannot shift arbitrary weight into the left-most coordinates, the lower bound on β guarantees $\frac{(1-\gamma)n}{4} \geq \frac{(1-\gamma\beta)n}{8}$.

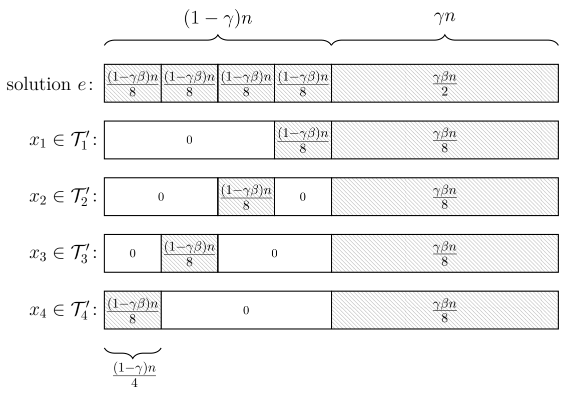

$(1-\gamma)n$
γn
solution e:
$\frac{(1-\gamma\beta)n}{8}$
$\frac{\gamma\beta n}{2}$
$x_1 \in \mathcal{T}'_1$:
$x_2 \in \mathcal{T}'_2$:
$x_3 \in \mathcal{T}'_3$:
$x_4 \in \mathcal{T}'_4$:
0
$\frac{\gamma\beta n}{8}$
$\frac{(1-\gamma)n}{4}$

Fig. 14: Weight distribution of vectors from new layer-1 domains. Shaded areas contain weight, white areas are all zero.

More formally, we change the layer-1 domains to

$$\mathcal{T}_1' = 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\frac{1}{4}(1-\gamma)n, \frac{1-\gamma\beta}{2(1-\gamma)}\right) \times \mathcal{B}\left(\gamma n, \frac{\beta}{8}\right)$$
$$\mathcal{T}_2' = 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\frac{1}{4}(1-\gamma)n, \frac{1-\gamma\beta}{2(1-\gamma)}\right) \times 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\gamma n, \frac{\beta}{8}\right)$$
$$\mathcal{T}_3' = 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\frac{1}{4}(1-\gamma)n, \frac{1-\gamma\beta}{2(1-\gamma)}\right) \times 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\gamma n, \frac{\beta}{8}\right)$$
$$\mathcal{T}_4' = \mathcal{B}\left(\frac{1}{4}(1-\gamma)n, \frac{1-\gamma\beta}{2(1-\gamma)}\right) \times 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times 0^{\frac{1}{4}(1-\gamma)n} \times \mathcal{B}\left(\gamma n, \frac{\beta}{8}\right) \quad .$$

This changes the domain sizes to

$$|\mathcal{T}_i'| = \binom{(1-\gamma)n/4}{(1-\gamma\beta)n/8}\binom{\gamma n}{\frac{\gamma\beta n}{8}} \quad .$$

In the analysis from Section 4.1, we set γ such that the search space of 2^n splits into $2^{\frac{n}{2}}$ for both layer-1 and layer-2 collisions. However, observe that our new *weight-shifted subset sum problem* has no longer search space of size 2^n, but only of size

$$\mathcal{S} = \binom{(1-\gamma)n/4}{(1-\gamma\beta)n/8}^4 \binom{\gamma n}{\frac{\gamma\beta n}{2}} = \tilde{\Theta}\left(2^{\left((1-\gamma)H\left(\frac{1-\gamma\beta}{2(1-\gamma)}\right)+\gamma H\left(\frac{\beta}{2}\right)\right)n}\right) \quad .$$

Let $\delta := (1-\gamma)H\left(\frac{1-\gamma\beta}{2(1-\gamma)}\right) + \gamma H\left(\frac{\beta}{2}\right)$ be the exponent of $\mathcal{S}$. Thus, computing $\langle \mathbf{a}, \mathbf{e}\rangle = t \bmod 2^{\delta n}$ is already sufficient for uniquely determining $\mathbf{e}$. Analogous to Section 4.1 we set $|\mathcal{T}_i'| = 2^{\frac{\delta}{2}n}$. Hence, each iteration of Nested Collision Subset Sum costs time $\tilde{\Theta}(2^{\frac{\delta}{2}n})$.

The probability to obtain the correct weight distribution for $\mathbf{e}$ is

$$p_1 := \frac{S}{\binom{n}{\frac{n}{2}}} = \tilde{\Theta}(2^{(\delta-1)n}) \quad .$$

Let us look at a fixed iteration of Nested Collision Subset Sum with some choice of R. Assume that in this iteration $\mathbf{e}$ has the correct weight distribution. Any representation $(\mathbf{e}_1, \ldots, \mathbf{e}_4)$ of $\mathbf{e}$ is useful in this iteration if $\langle \mathbf{a}, \mathbf{e}_1 + \mathbf{e}_2\rangle = R \bmod 2^{\frac{\delta}{2}n}$. Since we shift less weight into the γn right-most coordinates, the amount of distinct representations $(\mathbf{e}_1', \mathbf{e}_2') = (\mathbf{e}_1 + \mathbf{e}_2, \mathbf{e}_3 + \mathbf{e}_4)$ decreases to

$$\binom{\frac{\gamma\beta n}{2}}{\frac{\gamma\beta n}{4}} = \tilde{\Theta}\left(2^{\frac{\gamma\beta n}{2}}\right) \quad .$$

Hence the probability of choosing an $R \in \mathbb{Z}_{2^{\delta n/2}}$ for which useful representations exist becomes

$$p_2 := \tilde{\Theta}\left(\frac{2^{\frac{\gamma\beta n}{2}}}{2^{\frac{\delta n}{2}}}\right) = \tilde{\Theta}\left(2^{\frac{(\gamma\beta-\delta)n}{2}}\right) \quad .$$

For a good choice of R there exists at least one representation $(\mathbf{e}_1', \mathbf{e}_2') = (\mathbf{e}_1 + \mathbf{e}_2, \mathbf{e}_3 + \mathbf{e}_4)$ of the solution with $\langle \mathbf{a}, \mathbf{e}_1' \rangle = R \bmod 2^{\frac{\delta n}{2}}$, and the number of ways we can represent $(\mathbf{e}_1', \mathbf{e}_2') = (\mathbf{e}_1 + \mathbf{e}_2, \mathbf{e}_3 + \mathbf{e}_4)$ is

$$\binom{\frac{\gamma\beta n}{4}}{\frac{\gamma\beta n}{8}}^2 = \tilde{\Theta}\left(2^{\frac{\gamma\beta n}{2}}\right) \ .$$

Since g has a total of $\tilde{\Theta}(2^{\frac{\delta n}{2}})$ collisions, a random collision is a representation of the solution with probability

$$p_3 := \tilde{\Theta}\left(\frac{2^{\frac{\gamma\beta n}{2}}}{2^{\frac{\delta n}{2}}}\right) = \tilde{\Theta}\left(2^{\frac{(\gamma\beta - \delta)n}{2}}\right) \ .$$

With probability $p = p_1 p_2 p_3$ we have the correct weight distribution, choose a good R, and find a useful representation. Thus, we need on expectation p^{-1} iterations with running time $\tilde{\Theta}(2^{\frac{\delta}{2}n})$ each. This results in a total run time of

$$T = \tilde{\Theta}\left(2^{(1-\delta+\delta-\gamma\beta+\frac{\delta}{2})n}\right) = \tilde{\Theta}\left(2^{(1-\gamma\beta+\frac{\delta}{2})n}\right).$$

Optimization yields $\beta = 0.964$, from which we obtain $\gamma = 0.8832$ and $\delta = 0.9928$. This gives us $p_1 = 2^{-0.0072n}$, $p_2 = p_3 = 2^{-0.0707n}$ and a total expected run time of

$$T = 2^{0.645n}.$$

4.3 Experiments for our $2^{0.649n}$ Subset Sum Algorithm

We implemented our Nested Collision Subset Sum algorithm (Algorithm 3).[2]

n	16	24	32	40	48
$\log T_f$	16.80	21.91	26.97	32.01	37.25

Table 1: Amount of function calls T_f in logarithmic scale to solve a random subset sum instance in dimension n using our Nested Collision Subset Sum algorithm. Sample size per n is 30.

The computed regression line in Figure 15 is $\log T_f(n) = 0.637n + 6.678$. The parameter 6.678 shows that the implementation of our algorithm incorporates some quite large polynomial run time overhead. However, more importantly the experimental slope 0.637 demonstrates that our asymptotic run time exponent of 0.649 is already achieved in small dimension.

[2] implementation available at
https://github.com/LwDLPandSubsetSum/lwDLP-and-NestedSubsetSum

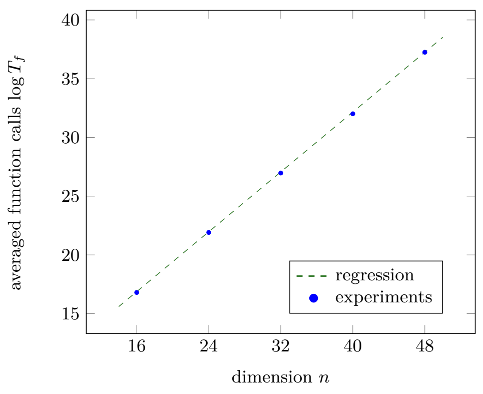


Fig. 15: Experimentally averaged number of function calls (in logarithmic scale) needed to solve a subset sum instance in dimension n of weight $n/2$. Sample size per n is 30.

References

1. Abboud, A., Bringmann, K., Hermelin, D., Shabtay, D.: Seth-based lower bounds for subset sum and bicriteria path. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 41–57. Society for Industrial and Applied Mathematics (2019)
2. Alekhnovich, M.: More on average case vs approximation complexity. In: 44th Annual Symposium on Foundations of Computer Science. pp. 298–307. IEEE Computer Society Press, Cambridge, MA, USA (Oct 11–14, 2003)
3. Bansal, N., Garg, S., Nederlof, J., Vyas, N.: Faster space-efficient algorithms for subset sum and k-sum. In: Hatami, H., McKenzie, P., King, V. (eds.) 49th Annual ACM Symposium on Theory of Computing. pp. 198–209. ACM Press, Montreal, QC, Canada (Jun 19–23, 2017)
4. Becker, A., Coron, J.S., Joux, A.: Improved generic algorithms for hard knapsacks. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 364–385. Springer, Heidelberg, Germany, Tallinn, Estonia (May 15–19, 2011)
5. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 520–536. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012)
6. Bernstein, D.J., Jeffery, S., Lange, T., Meurer, A.: Quantum algorithms for the subset-sum problem. In: Gaborit, P. (ed.) Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013. pp. 16–33. Springer, Heidelberg, Germany, Limoges, France (Jun 4–7 2013)
7. Bos, J.W., Kaihara, M., Kleinjung, T., Lenstra, A.K., Montgomery, P.L.: Solving 112-bit prime ecdlp on game consoles using sloppy reduction. International Journal of Applied Cryptography 2(ARTICLE), 212–228 (2012)
8. Chor, B., Rivest, R.L.: A Knapsack type public key cryptosystem based on arithmetic in finite fields. In: Blakley, G.R., Chaum, D. (eds.) Advances in Cryptology – CRYPTO'84. Lecture Notes in Computer Science, vol. 196, pp. 54–65. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 1984)

9. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Memory-efficient algorithms for finding needles in haystacks. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 185–206. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016)
10. Esser, A., Kübler, R., May, A.: LPN decoded. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 486–514. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)
11. Faust, S., Masny, D., Venturi, D.: Chosen-ciphertext security from subset sum. In: Cheng, C.M., Chung, K.M., Persiano, G., Yang, B.Y. (eds.) PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 9614, pp. 35–46. Springer, Heidelberg, Germany, Taipei, Taiwan (Mar 6–9, 2016)
12. Galbraith, S.D.: Mathematics of public key cryptography. Cambridge University Press (2012)
13. Galbraith, S.D., Gaudry, P.: Recent progress on the elliptic curve discrete logarithm problem. Designs, Codes and Cryptography 78(1), 51–72 (2016)
14. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: cold-boot attacks on encryption keys. Communications of the ACM 52(5), 91–98 (2009)
15. Herold, G., May, A.: LP solutions of vectorial integer subset sums — cryptanalysis of Galbraith's binary matrix LWE. In: Fehr, S. (ed.) PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 10174, pp. 3–15. Springer, Heidelberg, Germany, Amsterdam, The Netherlands (Mar 28–31, 2017)
16. Impagliazzo, R., Naor, M.: Efficient cryptographic schemes provably as secure as subset sum. Journal of Cryptology 9(4), 199–216 (Sep 1996)
17. Knuth, D.: The art of computer programming, 2 (seminumerical algorithms) (1981)
18. Lagarias, J.C., Odlyzko, A.M.: Solving low-density subset sum problems. Journal of the ACM (JACM) 32(1), 229–246 (1985)
19. Lyubashevsky, V., Palacio, A., Segev, G.: Public-key cryptographic primitives provably as secure as subset sum. In: Micciancio, D. (ed.) TCC 2010: 7th Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 5978, pp. 382–400. Springer, Heidelberg, Germany, Zurich, Switzerland (Feb 9–11, 2010)
20. May, A., Ozerov, I.: A generic algorithm for small weight discrete logarithms in composite groups. In: Joux, A., Youssef, A.M. (eds.) SAC 2014: 21st Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 8781, pp. 278–289. Springer, Heidelberg, Germany, Montreal, QC, Canada (Aug 14–15, 2014)
21. Merkle, R., Hellman, M.: Hiding information and signatures in trapdoor knapsacks. IEEE transactions on Information Theory 24(5), 525–530 (1978)
22. Nivasch, G.: Cycle detection using a stack. Information Processing Letters 90(3), 135–140 (2004)
23. Odlyzko, A.M.: The rise and fall of knapsack cryptosystems. Cryptology and computational number theory 42, 75–88 (1990)
24. Pollard, J.M.: A monte carlo method for factorization. BIT Numerical Mathematics 15(3), 331–334 (1975)
25. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing. pp. 84–93. ACM Press, Baltimore, MA, USA (May 22–24, 2005)

26. Shanks, D.: Five number-theoretic algorithms. In: Proceedings of the Second Manitoba Conference on Numerical Mathematics (Winnipeg), 1973 (1973)
27. Stinson, D.: Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem. Mathematics of Computation 71(237), 379–391 (2002)
28. Van Oorschot, P.C., Wiener, M.J.: Parallel collision search with application to hash functions and discrete logarithms. In: Proceedings of the 2nd ACM Conference on Computer and Communications Security. pp. 210–218. ACM (1994)
29. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. Journal of Cryptology 12(1), 1–28 (Jan 1999)