Improved Low-Memory Subset Sum and LPN Algorithms via Multiple Collisions

Claire Delaplace, Andre Esser and Alexander May

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

Abstract. For enabling post-quantum cryptanalytic experiments on a meaningful scale, there is a strong need for low-memory algorithms. We show that the combination of techniques from representations, multiple collision finding, and the Schroeppel-Shamir algorithm leeds to improved low-memory algorithms.
For random subset sum instances $(a_1, \ldots, a_n, t)$ defined modulo 2^n, our algorithms improve over the Dissection technique for small memory $M < 2^{0.02n}$ and in the mid-memory regime $2^{0.13n} < M < 2^{0.2n}$.
An application of our technique to LPN of dimension k and constant error p yields significant time complexity improvements over the Dissection-BKW algorithm from Crypto 2018 for all memory parameters $M < 2^{0.35 \frac{k}{\log k}}$.

Keywords: time-memory trade-off, representations, parallel collision search

1 Introduction

We are now in a transition phase to post-quantum cryptography, where we have to determine reliably strong parameters for prospective schemes (e.g. for the 2nd round candidates of NIST's post-quantum standardization process [1]). This requires mid-scaled cryptanalytic experiments from which we can safely extrapolate to the desired security levels. However, a major drawback for cryptanalytic analysis of most post-quantum systems, for example in comparison to their number-theoretic counterparts, is the large memory consumption of today's best attack algorithms.

For instance, the famous BKW algorithm [10] for attacking coding/lattice-based schemes [3, 15, 17, 22] as well as lattice sieving [2, 8] require huge memory, which prevents their application even for medium-sized parameters. Thus, there is a strong need for developing general techniques that sacrifices a bit of run time at the sake of having a manageable memory consumption. These time-memory trade-offs are well-studied for the subset sum problem [4, 6, 7, 13], which usually serves as a meta-problem to sharpen our tools and techniques. Then, these techniques are often transferred to the coding and lattice world [5, 8, 9, 24], where we solve similar vectorial versions of the subset sum problem.

In the subset sum context, the best memory-saving techniques are Schroeppel-Shamir [27] and the elegant Dissection technique from Dinur, Dunkelman, Keller, Shamir [13]. The Schroeppel-Shamir algorithm is a remarkable technique that allows to save memory without sacrificing time at all. Namely, solving subset sum via the usual Meet-in-the-Middle technique using the Horowitz-Sahni algorithm [19] requires time and space $\tilde{\mathcal{O}}(2^{n/2})$, whereas the Schroeppel-Shamir algorithm needs the same time but only $\tilde{\mathcal{O}}(2^{n/4})$ memory. The Dissection technique can be seen as a natural generalization of Schroeppel-Shamir, where Schroeppel-Shamir is the special case of a 4-Dissection. Indeed, one of the original applications of Dissection in [13] is the today's best time-memory trade-off for subset sum. More recently, Esser et al. [14] used Dissection for also designing time-memory trade-offs for the LPN (and LWE) problem.

Without memory restrictions, the currently best algorithm for solving random subset sum instances $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_{2^n}^n, t \in \mathbb{Z}_{2^n}$ — with some weight-$\frac{n}{2}$ solution $\mathbf{e} \in \{0,1\}^n$ satisfying $\langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n$ — is the Becker-Coron-Joux algorithm [7] with time and space complexity $2^{0.291n}$. The core idea of this algorithm is the so-called representation technique, where the search space for $\mathbf{e}$ is enhanced by R redundant representations of $\mathbf{e}$. Then one enumerates a $\frac{1}{R}$-fraction of the search space such that on expectation one representation survives.

Moreover, Becker, Coron and Joux also provide a polynomial memory algorithm that solves random subset sum instances in time $2^{0.72n}$. This low-memory algorithm represents $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ with weight-$\frac{n}{4}$ vectors and samples (via a random walk) weight-$\frac{n}{4}$ vectors $\mathbf{e}'_1, \mathbf{e}'_2$ until one finds a collision $\langle \mathbf{a}, \mathbf{e}'_1 \rangle = t - \langle \mathbf{a}, \mathbf{e}'_2 \rangle$. Thus, the low-memory algorithm uses collision finding to recover a 2-sum representation $\mathbf{e}'_1 + \mathbf{e}'_2 = \mathbf{e}$ of the solution.

Our contribution. First, we show that multiple collision finding easily provides a time-memory trade-off for the low-memory 2-sum Becker-Coron-Joux algorithm. We benefit basically from the well-known fact that multiple collision finding algorithms like Parallel Collision Search (PCS) [29] provide 2^m collisions for an r-bit function $f : \mathbb{F}_2^r \to \mathbb{F}_2^r$ in time only $2^{\frac{r+m}{2}}$ instead of the trivial $2^m \cdot 2^{\frac{r}{2}}$ for 2^m applications of simple collision finding.

Second, we develop a more involved 4-sum subset sum algorithm that represents $\mathbf{e} = \mathbf{e}_1 + \ldots + \mathbf{e}_4$ with weight-$\frac{n}{8}$ vectors $\mathbf{e}_i$, thereby profiting from the increased amount of representations. We sample all candidates $\mathbf{e}'_i$ for $\mathbf{e}_i$ via PCS as the sum of two weight-$\frac{n}{16}$ vectors, again exploiting the benefits of representations. When having sampled sufficiently many candidate tuple $(\mathbf{e}'_1, \ldots, \mathbf{e}'_4)$, one of them is a representation of $\mathbf{e}$ with high probability. We then efficiently construct this solution $\mathbf{e}$ using the Schroeppel-Shamir algorithm.

Our improved algorithms time-memory behaviours are depicted in Figure 1. Our 2-sum subset sum algorithm provides some improvement in the small memory regime $M < 2^{0.02n}$, whereas our 4-sum algorithm improves on the Dissection technique in the mid-size memory regime $2^{0.13n} < M < 2^{0.2n}$.

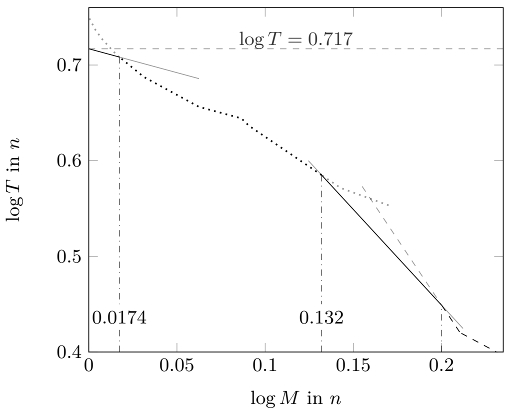


Fig. 1: Comparison of our trade-offs and the previous best trade-offs. The dotted line represent the trade-off obtained via Dissection [13], while the dashed one was obtained in [20] using representations. Our new trade-offs are depicted as solid lines.

Third, for the LPN problem with dimension k and constant error probability p, we build on the Dissection-BKW algorithm proposed by Esser et al. [14]. The authors of [14] show that any algorithm for a certain c-sum problem can be turned into an LPN algorithm. The c-sum problem is solved in [14] via the Dissection framework to obtain efficient time-memory trade-offs.

In this work, we express the c-sum problem from [14] as a multiple collision problem of two $\frac{c}{2}$-sums, which again is solved via sampling collisions with PCS. This results in quite significant time improvements for the whole memory region

$M < 2^{0.35 \frac{k}{\log k}}$ when compared to the Dissection technique (see Figure 2). For small memory regimes, the time complexity of our new `PCS-BKW` algorithm even comes close to the best quantum algorithm `Quantum-BKW`, which makes use of a highly memory-efficient Grover search.

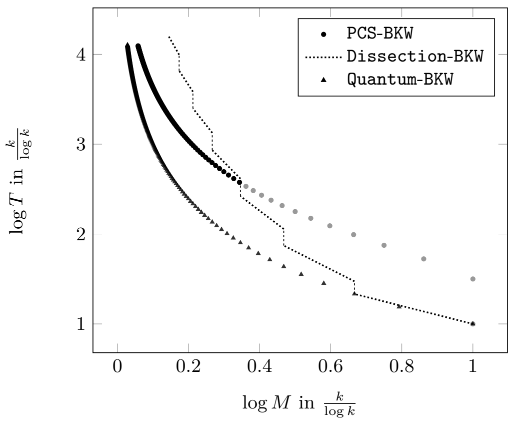


Fig. 2: The dotted marks depict our trade-off `PCS-BKW`, which improves for memory $M < 2^{0.35 \frac{k}{\log k}}$ on the so far best classical algorithm `Dissection-BKW` from [14]. The triangle marks depict the best known quantum trade-off [14].

In the commonly used time-memory notion, we achieve the results from Table 1. Notice that our `PCS-BKW` algorithm has the same linear dependency on c as `Quantum-BKW`.

Tradeoff	$2^{\log c \cdot \frac{k}{\log k}} =$
`PCS-BKW`	$T \cdot M^{\frac{c-2}{4}}$
`Dissection-BKW` [14]	$T \cdot M^{\sqrt{c}}$
`Quantum-BKW` [14]	$T \cdot M^{\frac{c-2}{2}}$

Table 1: Our LPN-trade-off `PCS-BKW` in comparison to [14].

Related work. Parallel Collision Search (PCS), introduced by van Oorschot and Wiener [29], is a widely applied tool in cryptanalysis for achieving good time-memory trade-offs [12, 16, 21, 25, 26]. It has been thoroughly analyzed in the context of finding multiple collisions [16, 23, 28].

Nikolić and Sasaki [25] applied PCS sampling in a similar scenario to ours, namely for constructing improved time-memory trade-offs for the Generalized Birthday Problem based on Wagners k-tree algorithm [30]. Dinur [12] later generalized the Nikolić-Sasaki approach using the Dissection technique.

2 Preliminaries

Let us denote by $\mathbb{Z}_q$ the ring of integers modulo q. $\mathcal{U}_S$ is the uniform distribution over a finite set S. Ber_p is the Bernoulli distribution with parameter p, i.e. for $X \sim \mathrm{Ber}_p$ we have $\Pr[X = 1] = p$ and $\Pr[X = 0] = 1 - p$. We denote by Geo_p the geometric distribution with parameter p, which is the amount of independent Bernoulli trials needed for the first success.

We define lists $L = \{\ell_1, \ell_2, \ldots, \ell_n\}$ as multisets over some universe S. For $\mathbf{x} \in \{0,1\}^n$ we refer to the i-th coordinate of $\mathbf{x}$ by x_i. By $\mathrm{wt}(\mathbf{x})$ we refer to the Hamming weight of $\mathbf{x}$.

For complexity statements we use soft-Oh notation, where $\tilde{\mathcal{O}}(f(k))$ is a shorthand for $\mathcal{O}(\log(f(k))^i \cdot f(k))$ for some constant i. An algorithm succeeds with high probability $p(n)$ if $p(n) \geq 1 - \frac{1}{\mathrm{poly}(n)}$.

We refer to the binary entropy function by $H(\alpha)$, $\alpha \in [0,1]$. We also make use of the approximation of binomial coefficients derived from Stirlings formula, which is $\binom{n}{m} \simeq 2^{n \cdot H(\frac{m}{n})}$, where $\simeq$ is used to suppress polynomial factors in n. More precisely we have

$$\frac{2^{n \cdot H(\frac{m}{n})}}{n+1} \leq \binom{n}{m} \leq 2^{n \cdot H(\frac{m}{n})}.$$

For finding multiple collisions between functions $f, g : \mathbb{F}_2^r \to \mathbb{F}_2^r$ we apply the Parallel Collision Search algorithm [29], denoted `PCS`. We call procedure `PCS`$(f, g, \alpha r)$ for finding $2^{\alpha r}, 0 \leq \alpha \leq 1$ distinct collisions. The following theorem states `PCS`'s complexities.

Theorem 2.1 (Parallel Collision Search). *Let $r \in \mathbb{N}$, $0 \leq \alpha \leq 1$, and $m := \alpha r$. Given two independent random functions $f\colon \{0,1\}^r \to \{0,1\}^r$ and $g\colon \{0,1\}^r \to \{0,1\}^r$,* Parallel Collision Search *returns 2^m distinct collisions between f and g using expected time $T = \tilde{\mathcal{O}}\left(2^{\frac{r+m}{2}}\right)$ and memory $M = \tilde{\mathcal{O}}(2^m)$.*

For more details on `PCS` the reader is referred to [28,29]. A complexity analysis for multiple collisions is given in [16, 23, 28]. We apply `PCS` on random subset sum, defined as follows.

Definition 2.1 (Random Subset Sum). *Let $\mathbf{a} \in (\mathbb{Z}_{2^n})^n$ be chosen uniformly at random. For a random $\mathbf{e} \in \mathbb{F}_2^n$ with $\mathrm{wt}(\mathbf{e}) = \frac{n}{2}$ compute $t = \langle \mathbf{a}, \mathbf{e} \rangle \bmod 2^n$. Then $(\mathbf{a}, t) \in (\mathbb{Z}_{2^n})^{n+1}$ is called a random subset sum instance, while each $\mathbf{e}' \in \mathbb{F}_2^n$ with $\langle \mathbf{a}, \mathbf{e}' \rangle = t$ is called a solution.*

Following Howgrave-Graham and Joux [20], we represent a subset sum solution in a redundant manner as sums of vectors, called *representations*.

Definition 2.2 (Representation). *Let $\mathbf{e} \in \{0,1\}^n$ with $\mathrm{wt}(\mathbf{e}) = \beta n$. Any tuple $(\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_k) \in \left(\{0,1\}^n\right)^k$ with $\mathrm{wt}(\mathbf{e}_i) = \frac{\beta n}{k}$ for all $i = 1, \ldots, k$ is called a* representation *of $\mathbf{e}$ if $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2 + \ldots + \mathbf{e}_k$.*

The Schroeppel-Shamir Algorithm [27]. We use the algorithm by Schroeppel and Shamir to solve the following problem: Given four lists $L_1, \ldots, L_4$ of equal size 2^m containing uniformly at random drawn elements from $\mathbb{Z}_{2^n}$ together with a target $t \in \mathbb{Z}_{2^n}$, compute the solution set

$$\mathcal{C} = \{(x_{i_1}, \ldots, x_{i_4}) \in L_1 \times \ldots \times L_4 \mid x_{i_1} + \ldots + x_{i_4} = t \mod 2^n\} \ .$$

While the original algorithm uses involved data structures to guarantee worst case complexities, we use a heuristic simplification by Howgrave-Graham and Joux [20] at the cost of obtaining only expected complexities. However, we still refer to the algorithm as `Schroeppel-Shamir`.

`Schroeppel-Shamir` merges two lists at a time. The lists L_1 and L_2 are merged into a new list

$$L_{12} = \{x + y \mid x \in L_1, y \in L_2, x + y = R \mod 2^m\} \text{ for some } R \in \mathbb{Z}_{2^m} \ .$$

The constraint R enforces expected size $|L_{12}| = 2^m$. Similarly, we merge L_3, L_4 into L_{34} with the constraint $t - R \bmod 2^m$. Eventually, we merge L_{12}, L_{34} such that their elements sum up to $t \bmod 2^n$. To compute the complete solution set $\mathcal{C}$ the algorithm iterates over all $R \in \mathbb{Z}_{2^m}$.

Since the elements in $L_1, \ldots, L_4$ are uniformly distributed, we have $\mathbb{E}[|C|] = 2^{4m-n}$. Thus $\mathbb{E}[|C|] \leq 2^m$ if $n \geq 3m$. Since each merge can be performed in expected time and memory $\tilde{\mathcal{O}}(2^m)$ the total expected time complexity by iterating over all constraints is $\tilde{\mathcal{O}}(2^{2m})$, while the expected memory complexity is $\tilde{\mathcal{O}}(2^m)$.

Lemma 2.1 (Schroeppel-Shamir). *Let $m, n \in \mathbb{N}$ with $n \geq 3m$. Given four lists $L_1, \ldots, L_4$ of equal size 2^m containing uniformly at random drawn elements from $\mathbb{Z}_{2^n}$ together with a target $t \in \mathbb{Z}_{2^n}$,* `Schroeppel-Shamir` *returns the solution set $\mathcal{C}$ in expected time $\tilde{\mathcal{O}}\left(2^{2m}\right)$ and memory $\tilde{\mathcal{O}}\left(2^m\right)$.*

We also apply multiple collision search to the LPN problem, which is defined as follows.

Definition 2.3 (Search LPN Problem). *Let $k \in \mathbb{N}$, $\mathbf{s} \in \mathbb{F}_2^k$ and $p \in [0, \frac{1}{2})$ be a constant. Let* `Sample` *denote an oracle that, when queried, samples $\mathbf{a} \sim \mathcal{U}_{\mathbb{F}_2^k}$, $e \sim Ber_p$ and outputs a* sample *of the form $(\mathbf{a}, b) := (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$. The* $\texttt{LPN}_k$ problem *consists of recovering* $\mathbf{s}$ *given access to* `Sample`.

3 New Subset-Sum Trade-Offs Using PCS

We introduce two new trade-offs for the random subset sum problem from Definition 2.1. The first one, `SS-PCS`, uses the representation technique together with the PCS algorithm and provides time improvements in the sparse memory area with memory $M < 2^{0.02n}$. The second one, $\texttt{SS-PCS}_4$, combines representations with Schroeppel-Shamir and PCS to achieve time improvements in the (quite large) memory regime $2^{0.13n} < M < 2^{0.2n}$. Notice that a memory $M = 2^{0.291n}$ is sufficient to run the best algorithm by Becker-Coron-Joux [7] with time $T = M$. All in all, we achieve improvements in a large parameter range of M (for roughly a third of the meaningful exponents).

3.1 Algorithm SS-PCS

Our algorithm SS-PCS builds on the memoryless BCJ-algorithm [7] for which we replace its simple collision search procedure by PCS.

The idea of the BCJ-algorithm is to split the solution vector $\mathbf{e}$ with $\mathrm{wt}(\mathbf{e}) = \frac{n}{2}$ in two vectors $\mathbf{e}_1, \mathbf{e}_2 \in \mathbb{F}_2^n$ each of weight $\frac{n}{4}$. Let $(\mathbf{a}, t)$ be a random subset sum instance and $\mathcal{T} := \{\mathbf{x} \in \mathbb{F}_2^n \mid \mathrm{wt}(\mathbf{x}) = \frac{n}{4}\}$, where $|\mathcal{T}| \simeq 2^{H\left(\frac{1}{4}\right)n}$ and define the two functions

$$g, g_t \colon \mathcal{T} \to \mathbb{Z}_{2^{H(1/4)n}} \quad \text{, where}$$
$$g(\mathbf{x}) = \sum_{i=1}^{n} x_i a_i \mod 2^{H\left(\frac{1}{4}\right)n} \text{ and } g_t(\mathbf{x}) = t - \sum_{i=1}^{n} x_i a_i \mod 2^{H\left(\frac{1}{4}\right)n} \ .$$

Note that any representation $(\mathbf{e}_1, \mathbf{e}_2)$ of our solution $\mathbf{e}$ satisfies $g(\mathbf{e}_1) = g_t(\mathbf{e}_2)$. The algorithm now simply searches for collisions $(\mathbf{e}_1', \mathbf{e}_2')$ between g and g_t until $\mathbf{e}_1' + \mathbf{e}_2'$ yields a solution to the subset sum instance.

We expect to have $2^{H(1/4)n}$ collisions between g and g_t, whereas the number of representations $(\mathbf{e}_1, \mathbf{e}_2)$ is $\binom{n/2}{n/4} \simeq 2^{\frac{n}{2}}$. Thus, a random collision solves the subset sum instance with probability $p = 2^{1/2 - H(1/4)n}$. Equivalently, we need to compute on expectation $2^{(H(1/4)-1/2)n} = 2^{0.31n}$ collisions before we find the solution. The BCJ-algorithm uses standard cycle-finding for computing a collision in $2^{H(1/4)\frac{n}{2}}$, resulting in total run time $2^{\frac{3H(1/4)-1}{2}n} = 2^{0.717n}$.

Since we have to compute an exponential amount of collisions, obviously PCS can be utilized to improve on run time if we are willing to spend some memory. In the following, we show that (under some mild heuristic assumption) algorithm SS-PCS (Algorithm 1) solves random subset sum instances in time $2^{(0.717 - \frac{\gamma}{2})n}$ using memory $\tilde{\mathcal{O}}(2^{\gamma n})$.

Algorithm 1 SS-PCS$((\mathbf{a}, t), \gamma)$

```
Input: subset sum instance $(\mathbf{a}, t)$, memory parameter $\gamma$
Output: solution $\mathbf{e} \in \mathbb{F}_2^n$ to instance $(\mathbf{a}, t)$ or $\perp$
1: for $i = 1$ to $n^3 \cdot 2^{(H(\frac{1}{4}) - \frac{1}{2} - \gamma)n}$ do
2:     $L \leftarrow$ PCS$(g, g_t, \gamma n)$
3:     if $\exists(\mathbf{e}_1, \mathbf{e}_2) \in L$, such that $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2 \in \{0,1\}^n$ and $\langle \mathbf{a}, \mathbf{e} \rangle = t$ then
4:         return $\mathbf{e}$
5: return $\perp$
```

Notice that for rigorously analyzing the time complexity of SS-PCS, we have to lower bound the probability p for success in each iteration. This in turn requires an upper bound on the number of collisions between g and g_t, which is shown in the following lemma.

Lemma 3.1 (Number of collisions between g and g_t). *Let $n \in \mathbb{N}$ and $(\mathbf{a}, t)$ be a random subset sum instance. Then with high probability the number of collisions between g and g_t is at most $n \cdot 2^{H(1/4)n}$.*

Proof. Given in Appendix A.

Assumptions of the analysis. For the proof of the complexity of `SS-PCS` we rely on commonly used heuristics in the context of `PCS` [13, 29] and collision search [7].

Heuristic 1.

1) `PCS` returns uniformly random collisions.
2) `PCS` behaves on input functions g and g_t as on independent random functions.

We validate Heuristic 1 experimentally in Appendix C.

Theorem 3.1 (Complexity of `SS-PCS`). *Let $(\mathbf{a}, t)$ be a random subset sum instance and let $0 \leq \gamma \leq 0.31$. Then under Heuristic 1 with high probability* `SS-PCS` *finds a solution to $(\mathbf{a}, t)$ in expected time $2^{\left(0.717 - \frac{\gamma}{2}\right)n}$ and memory $\tilde{\mathcal{O}}(2^{\gamma n})$.*

Proof. The time complexity of one iteration of the loop is dominated by the execution of `PCS`. By Theorem 2.1 `PCS` finds $2^{\gamma n}$ collisions between g and g_t in expected time $\tilde{\mathcal{O}}\left(2^{\frac{H(1/4)+\gamma}{2}n}\right)$. Therefore the overall expected time complexity is

$$T = n^3 \cdot 2^{\frac{2H(1/4)-1-2\gamma}{2}n} \cdot \tilde{\mathcal{O}}\left(2^{\frac{H(1/4)+\gamma}{2}n}\right)$$
$$= \tilde{\mathcal{O}}\left(2^{\frac{3H\left(\frac{1}{4}\right)-1-\gamma}{2}n}\right) = 2^{(0.717-\frac{\gamma}{2})n} \quad .$$

Here, the restriction $\gamma \leq 0.31$ ensures that the exponent $H(\frac{1}{4}) - \frac{1}{2} - \gamma$ of the number of iterations of the **for**-loop is positive. Regarding memory, we need to store $|L| = 2^{\gamma n}$ elements.

It remains to show that `SS-PCS` succeeds with high probability. An iteration is successful whenever list L contains a representation $(\mathbf{e}_1, \mathbf{e}_2)$ of the desired solution $\mathbf{e}$. Let us denote by $\mathcal{R}_{\frac{n}{2},2}$ the number of representations of $\mathbf{e}$ with weight $\frac{n}{2}$ as a decomposition in two vectors $(\mathbf{e}_1, \mathbf{e}_2)$. Via Definition 2.2 and Stirling's formula we obtain

$$\mathcal{R}_{\frac{n}{2},2} = \binom{n/2}{n/4} \geq \frac{2^{\frac{n}{2}}}{n} \quad .$$

By construction, each representation $(\mathbf{e}_1, \mathbf{e}_2)$ forms a collision between g and g_t. Thus, a collision sampled uniformly at random from the whole set of collisions C_{g,g_t} is a solution with probability

$$q := \frac{\mathcal{R}_{\frac{n}{2},2}}{|C_{g,g_t}|} \quad .$$

Since by Lemma 3.1 with high probability the total amount of collisions is $|C_{g,g_t}| \leq n \cdot 2^{H(\frac{1}{4})n}$, we obtain

$$q \geq \frac{1}{n^2} \cdot 2^{-\left(H(\frac{1}{4})-\frac{1}{2}\right)n} \quad .$$

Let Y denote the amount of sampled collisions until we hit a solution, where $Y \sim \mathrm{Geo}_q$. Then

$$\mathbb{E}[Y] = \frac{1}{q} \leq n^2 \cdot 2^{\left(H(\frac{1}{4})-\frac{1}{2}\right)n} \quad .$$

In total, `SS-PCS` samples $2^{\gamma n} \cdot n^3 \cdot 2^{\left(H(\frac{1}{4})-\frac{1}{2}-\gamma\right)n} = n^3 \cdot 2^{\left(H(\frac{1}{4})-\frac{1}{2}\right)n}$ collisions. Under Heuristic 1 these are independently and uniformly at random drawn from the whole set of collisions.

Thus, by Markov's inequality `SS-PCS` does not recover the solution with probability at most

$$\Pr\left[Y > n^3 \cdot 2^{\left(H(\frac{1}{4})-\frac{1}{2}\right)n}\right] \leq \frac{\mathbb{E}[Y]}{n^3 \cdot 2^{\left(H(\frac{1}{4})-\frac{1}{2}\right)n}} \leq \frac{1}{n}.$$

□

The achieved trade-off is illustrated in Figure 3. As discussed before, it contains as special case the memoryless algorithm by Becker et al. [7] and is superior to the previously best trade-off based on a combination of PCS and the Dissection framework [13] for any memory $M \leq 2^{0.0174n}$.

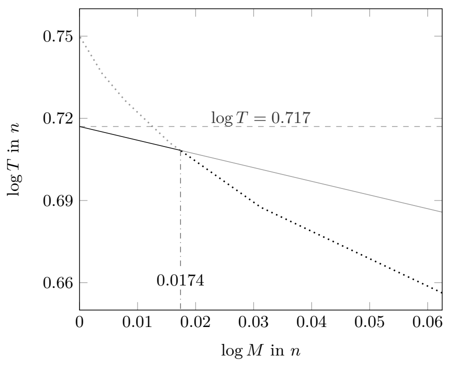


Fig. 3: Comparison of our trade-off and the previous best trade-off for small available memory. The dotted line represents the trade-off obtained in [13], while the solid line is our trade-off achieved by using `PCS`.

3.2 Algorithm `SS-PCS`$_4$

In this section we introduce a more involved trade-off based on a combination of PCS, the representation technique and the Schroeppel-Shamir algorithm that achieves time improvements for an available memory of $2^{0.13n} < M < 2^{0.2n}$.

Idea of SS-PCS$_4$. We represent our solution $\mathbf{e}$ as a sum of four vectors $\mathbf{e} = \mathbf{e}_1 + \cdots + \mathbf{e}_4$, such that $\mathrm{wt}(\mathbf{e}_j) = \frac{n}{8}$ for all j. We choose random restrictions $R_1, R_2, R_3 \in \mathbb{Z}_{2^{\lambda n}}$ and $R_4 = t - (R_1 + R_2 + R_3) \mod 2^{\lambda n}$, for some $0 < \lambda < 1$.

Using PCS, we compute four lists $L_i \subseteq Z_i$, $i = 1, \dots, 4$, where

$$Z_i = \left\{ u_i = \langle \mathbf{a}, \mathbf{e}_i' \rangle \mid \mathrm{wt}(\mathbf{e}_i') = \frac{n}{8},\ u_i = R_i \bmod 2^{\lambda n} \right\} .$$

Note that by construction every $(u_1, u_2, u_3, u_4) \in L_1 \times L_2 \times L_3 \times L_4$ satisfies $u_1 + u_2 + u_3 + u_4 = t \bmod 2^{\lambda n}$. We now use the Schroeppel-Shamir algorithm to search for $(u_1, u_2, u_3, u_4) \in L_1 \times L_2 \times L_3 \times L_4$, such that also $u_1 + u_2 + u_3 + u_4 = t \bmod 2^n$, which yields

$$\langle \mathbf{a}, \mathbf{e}_1' \rangle + \langle \mathbf{a}, \mathbf{e}_2' \rangle + \langle \mathbf{a}, \mathbf{e}_3' \rangle + \langle \mathbf{a}, \mathbf{e}_4' \rangle = \langle \mathbf{a}, \mathbf{e}_1' + \mathbf{e}_2' + \mathbf{e}_3' + \mathbf{e}_4' \rangle = t \bmod 2^n.$$

This implies that $\mathbf{e}_1' + \mathbf{e}_2' + \mathbf{e}_3' + \mathbf{e}_4' \in \{0,1,2,3,4\}^n$ solves our subset sum instance iff it defines a vector in $\{0,1\}^n$. We iterate our process until we find a solution.

Remark 3.1. In order to be able to reconstruct the solution, the Schroeppel-Shamir algorithm has to keep track of the vectors $\mathbf{e}_i'$ that were used to produce the corresponding list elements $u_i \in L_i$. For ease of notation, we ignore this in the following.

Let us elaborate a bit on how we construct $L_1, \dots, L_4$ using PCS. We denote by $\mathcal{S}$ the set $\mathcal{S} := \{\mathbf{x} \in \mathbb{F}_2^n \mid \mathrm{wt}(\mathbf{x}) = n/16\}$ with $|\mathcal{S}| = \binom{n}{n/16} \simeq 2^{H(1/16)n}$ and define the function f as

$$\begin{aligned} f \colon \mathcal{S} &\to \mathbb{Z}_{2^{H(1/16)n}} \\ \mathbf{x} &\mapsto \sum_{i=1}^{n} x_i a_i \mod 2^{H(1/16)n} = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^{H(1/16)n} . \end{aligned}$$

Analogously, given an arbitrary value $R \in \mathbb{Z}_{2^{H(1/16)n}}$ we define

$$\begin{aligned} f_R \colon \mathcal{S} &\to \mathbb{Z}_{2^{H(1/16)n}} \\ \mathbf{x} &\mapsto R - \sum_{i=1}^{n} x_i a_i \mod 2^{H(1/16)n} = R - \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^{H(1/16)n} . \end{aligned}$$

Thus, any collision $(\mathbf{x}, \mathbf{y}) \in \mathcal{S}^2$ between f and f_R satisfies

$$\begin{aligned} f(\mathbf{x}) = f_R(\mathbf{y}) \Leftrightarrow \ & \sum_{i=1}^{n} x_i a_i = R - \sum_{i=1}^{n} y_i a_i \mod 2^{H(1/16)n} \\ \Leftrightarrow \ & \sum_{i=1}^{n} (x_i + y_i) a_i = R \qquad \mod 2^{H(1/16)n} . \end{aligned}$$

Since eventually we look for a solution $\mathbf{e} \in \{0,1\}^n$, we filter out non-binary vectors $\mathbf{x} + \mathbf{y}$ in every iteration of our algorithm.

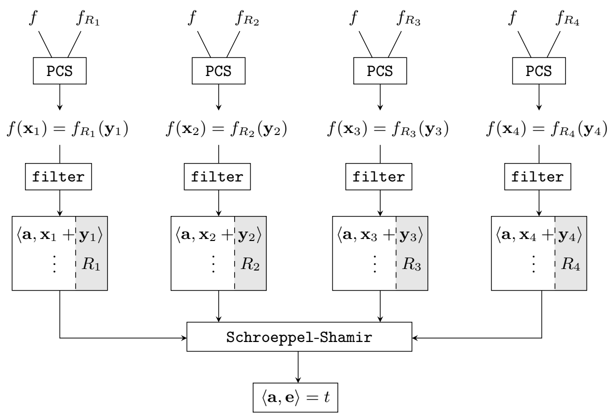
Algorithm 2 SS-PCS$_4$($(\mathbf{a}, t), \gamma$)

```
Input: subset sum instance $(\mathbf{a}, t)$, memory parameter $\gamma$
Output: solution $\mathbf{e} \in \mathbb{F}_2^n$ to instance $(\mathbf{a}, t)$ or $\perp$
 1: for $i = 1$ to $n^4 \cdot 2^{(3 \cdot H(1/16)-1)n}$ do
 2:     choose $R_1, R_2, R_3 \in \mathbb{Z}_{2^{H(1/16)n}}$ uniformly at random
 3:     $R_4 \leftarrow t - R_1 - R_2 - R_3 \mod 2^{H(1/16)n}$
 4:     for $i = 1$ to $n^9 \cdot 2^{(4H(1/16)-\frac{1}{2}-4\gamma)n}$ do
 5:         $L_i \leftarrow$ PCS$(f, f_{R_i}, \gamma n)$ for $i = 1, 2, 3, 4$
 6:         Filter$(L_i)$ for $i = 1, 2, 3, 4$          ▷ Filter out inconsistent vectors.
 7:         $L \leftarrow$ Schroeppel-Shamir$(L_1, L_2, L_3, L_4, t)$
 8:         if $\exists (u_1, u_2, u_3, u_4) \in L$, such that $\mathbf{e} = \mathbf{e}_1' + \mathbf{e}_2' + \mathbf{e}_3' + \mathbf{e}_4'$ is in $\{0,1\}^n$ then
 9:             return e
10: return $\perp$
```

Definition 3.1. *Let* $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$. *We call* $(\mathbf{x}, \mathbf{y})$ consistent *if* $\mathbf{x} + \mathbf{y} \in \{0,1\}^n$, *otherwise we call* $(\mathbf{x}, \mathbf{y})$ inconsistent.

Notice that for consistent $(\mathbf{x}, \mathbf{y})$ we have $\text{wt}(\mathbf{x} + \mathbf{y}) = \text{wt}(\mathbf{x}) + \text{wt}(\mathbf{y})$.

Now, our algorithm proceeds as follows (see also Figure 4). We fill all lists L_i with collisions provided by PCS, where we filter out inconsistent collisions immediately. Notice that this filter does not discard any representation of the desired solution, since representations are consistent by definition. The whole algorithm is summarized in Algorithm 2.



f
f_{R_1}
f
f_{R_2}
f
f_{R_3}
f
f_{R_4}
PCS
$f(\mathbf{x}_1) = f_{R_1}(\mathbf{y}_1)$
$f(\mathbf{x}_2) = f_{R_2}(\mathbf{y}_2)$
$f(\mathbf{x}_3) = f_{R_3}(\mathbf{y}_3)$
$f(\mathbf{x}_4) = f_{R_4}(\mathbf{y}_4)$
filter
$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{y}_1 \rangle$ R_1
$\langle \mathbf{a}, \mathbf{x}_2 + \mathbf{y}_2 \rangle$ R_2
$\langle \mathbf{a}, \mathbf{x}_3 + \mathbf{y}_3 \rangle$ R_3
$\langle \mathbf{a}, \mathbf{x}_4 + \mathbf{y}_4 \rangle$ R_4
Schroeppel-Shamir
$\langle \mathbf{a}, \mathbf{e} \rangle = t$

Fig. 4: One iteration of the SS-PCS$_4$ Algorithm.

For the analysis, we use a heuristic similar to Heuristic 1, where we additionally assume that representations of a solution are sufficiently uniform, as commonly done in the context of the representation technique [7, 18, 20].

Heuristic 2.
1) PCS returns uniformly random collisions.
2) PCS behaves on input functions f and f_R as on independent random functions.
3) Let $(\mathbf{e}_1, \ldots, \mathbf{e}_4)$ be a representation of the solution $\mathbf{e}$ of a random subset sum instance $(\mathbf{a}, t)$. Then the values of $\langle \mathbf{a}, \mathbf{e}_i \rangle \mod 2^{H(1/16)n}$, $i = 1, 2, 3$, are independently and uniformly distributed.

Theorem 3.2 (Complexity of SS-PCS$_4$). *Let $(\mathbf{a}, t)$ be a random subset sum instance and let $\frac{1}{8} \leq \gamma \leq 0.21$. Then under Heuristic 2 with high probability* SS-PCS$_4$ *finds a solution to $(\mathbf{a}, t)$ in expected time $2^{(0.849-2\gamma)n}$ and memory $\tilde{\mathcal{O}}(2^{\gamma n})$.*

Proof. Given in Appendix B.

Our new trade-offs are illustrated in Figure 5. By Theorem 3.2, SS-PCS$_4$ gives us for memory $2^{\gamma n}$ within the interval $\frac{1}{8} \leq \gamma \leq 0.21$ a line with slope -2 (solid line in Figure 5). For $\gamma \geq 0.132$ our algorithm improves on the trade-off based on the Dissection framework obtained in Dinur et al. [13] (dotted lines in Figure 5). Moreover, for $\gamma \leq 0.2$, our algorithm improves on a trade-off by Howgrave-Graham and Joux [20] (dashed line in Figure 5).

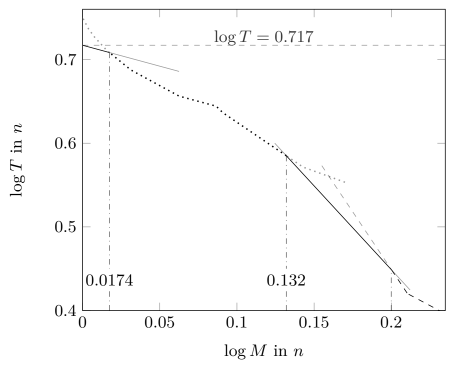


Fig. 5: Comparison of our trade-offs and the previous best trade-offs. The dotted line represent the trade-off obtained in [13], while the dashed one was obtained in [20]. Our new trade-offs are depicted as solid lines.

Remark 3.2. We also generalized our 4-sum algorithm SS-PCS$_4$ to a 7- and 11-sum algorithm in a natural way, thereby replacing Schroeppel-Shamir with a 7- respectively 11-Dissection. While our 7-sum algorithm gave us an additional (tiny) improvement, the 11-sum algorithm could no longer provide any improvements.

4 Application to LPN

The results from Section 3 show that the subset sum problem formulated as a 4-sum problem in combination with PCS leads to improved time-memory trade-offs.

This technique is even superior to the quite involved time-memory Dissection framework for many parameter sets.

Since c-sum applications appear quite often in cryptanalysis, it is natural to ask whether other problems enjoy similar advantages. At Crypto '18, Esser et al. [14] proposed time-memory trade-offs for the LPN problem of dimension k using c-sums in combination with Dissection. In the following, we show that the combination of c-sums with PCS also for LPN provides significant improvements for the whole memory region $M < 2^{0.35\frac{k}{\log k}}$.

The BKW algorithm [10] achieves for $\mathtt{LPN}_k$ time and memory complexity of $2^{\frac{k}{\log k}(1+o(1))}$ using 2-sums. Esser et al. [14] achieved time-memory trade-offs for $\mathtt{LPN}_k$ by using BKW in combination with c-sums for $c > 2$ and the Dissection technique. The resulting algorithm is called Dissection-BKW in [14]. Let us define the c-sum problem underlying $\mathtt{LPN}_k$ more formally.

Definition 4.1 (The c-Sum-Problem ($\mathtt{c}$-$\mathtt{SP}$)). *Let $b, c, N \in \mathbb{N}$ with $c \geq 2$. Let $L := \{\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_N\}$, where each list element $\boldsymbol{\ell_i} \sim \mathcal{U}_{\mathbb{F}_2^b}$. A* single-solution *of the $\mathtt{c}$-$\mathtt{SP}_b$ is a size-c set $\mathcal{L} \subset \{1, \ldots, N\}$ such that*

$$\sum_{j \in \mathcal{L}} \boldsymbol{\ell}_j = 0^b \quad .$$

A solution *is a set of at least N distinct single-solutions. The* c-sum-problem *$\mathtt{c}$-$\mathtt{SP}_b$ consists in finding a solution when given L.*

Hence, we have to find N different combinations of c out of all N b-bit vectors in L such that each of them sums up to the all-zero vector. In the BKW algorithm applied to $\mathtt{LPN}_k$, the block-size b is chosen as $b := \log c \cdot \frac{k}{\log k}$. Esser et al [14] showed that the running time of their Dissection-BKW on $\mathtt{LPN}_k$ is dominated by the time to solve the $\mathtt{c}$-$\mathtt{SP}_b$, as formulated in the next theorem.

Here, we use the following heuristic from Esser et al [14] for analyzing c-sum algorithms. This heuristic is backed up theoretically for $c = 2$ by results in [11] and experimentally for $c > 2$ in [14].

Independence Heuristic. [14] We treat c-sums as independent in the run time analysis.

Theorem 4.1. *[14, Theorem 3.2] Let $k, c \in \mathbb{N}$ and $0 < \varepsilon < 1$. Let us define $b := \frac{\log c}{1-\varepsilon} \cdot \frac{k}{\log k}$. Under the Independence Heuristic the following holds: If there is an algorithm solving the $\mathtt{c}$-$\mathtt{SP}_b$ for an input list of size N in expected time T and memory M, then it is possible to solve the $\mathtt{LPN}_k$ problem with high probability in time $T^{1+o(1)}$ and memory $M^{1+o(1)}$ using $N^{1+o(1)}$ samples, as long as $\log(N) \geq \frac{b+c\log c+1}{c}$.*

Theorem 4.1 states that N (roughly) denotes the number of samples from our $\mathtt{LPN}_k$ oracle. By Definition 2.3, N can be freely chosen, since we are given full access to $\mathtt{Sample}$. However, Theorem 4.1 provides a lower bound on N, which basically guarantees the existence of a solution to the $\mathtt{c}$-$\mathtt{SP}_b$ as specified in Definition 4.1.

Algorithm 3 c-sum-PCS(L)

```
Input: list $L = \{\boldsymbol{\ell}_1, \ldots \boldsymbol{\ell}_N\}$ with $\boldsymbol{\ell}_i \in \mathbb{F}_2^b$, where $N = c \cdot 2^{\frac{2b}{c}}$
Output: solution $S$ to the c-SP$_b$ instance $L$
1: split $L$ in $c$ lists $L_i$ of equal size $2^{\frac{2b}{c}}$
2: $S \leftarrow$ PCS$(f_0, f_1, \log N)$
3: return $S$
```

4.1 Computing c-sums with PCS

We choose list size $N = |L| = c \cdot 2^{\frac{2b}{c}}$, which satisfies the constraint from Theorem 4.1. For simplicity of notation, we assume in the following that $c \in \mathbb{N}$ is even. We first split L in c lists of equal size $|L_i| = 2^{\frac{2b}{c}}$, $i = 1, \ldots, c$. Let us denote by $L_i[k]$ the k^{th} element in list L_i. For $j = 0, 1$ we define the functions

$$\begin{aligned} f_j \colon \left(\mathbb{F}_2^{\frac{2b}{c}}\right)^{\frac{c}{2}} &\to \mathbb{F}_2^b, \\ (x_1, \ldots, x_{\frac{c}{2}}) &\mapsto L_{1+j\frac{c}{2}}[x_1] + L_{2+j\frac{c}{2}}[x_2] \ldots + L_{(1+j)\frac{c}{2}}[x_{\frac{c}{2}}]. \end{aligned}$$

Hence, f_0 maps $\frac{c}{2}$ indices to a $\frac{c}{2}$-sum over the first half of all lists, where f_1 computes a $\frac{c}{2}$-sum over the second half. Therefore a collision $f_0(x_1, \ldots, x_{\frac{c}{2}}) = f_1(x_{\frac{c}{2}+1}, \ldots, x_c)$ yields a c-sum satisfying

$$L_1[x_1] + \ldots + L_c[x_c] = 0^b \ ,$$

as desired for a single-solution in Definition 4.1. A solution to the c-SP$_b$ requires according to Definition 4.1 N distinct single-solutions. Thus, we apply PCS to find $N = c \cdot 2^{\frac{2b}{c}}$ different collisions between f_0 and f_1. The resulting procedure is given in Algorithm 3.

Lemma 4.1 (Complexity of c-sum-PCS). *Let c be even, and let L be a* c-SP$_b$*-instance with $|L| = N := c \cdot 2^{\frac{2b}{c}}$. Under the Independence Heuristic* c-sum-PCS *solves the* c-SP$_b$ *in expected time $T = \tilde{\mathcal{O}}\left(2^{(\frac{1}{2}+\frac{1}{c})b}\right)$ and memory $M = \tilde{\mathcal{O}}(2^{\frac{2b}{c}})$.*

Proof. The time complexity of c-sum-PCS is dominated by the application of PCS. Since list elements from L are from $\mathcal{U}_{\mathbb{F}_2^b}$, under the Independence Heuristic the functions f_0 and f_1 behave like independent random functions. By Theorem 2.1 the PCS algorithm finds $N = c \cdot 2^{\frac{2b}{c}}$ collisions between f_0 and f_1 with range $\mathbb{F}_2^b$ using memory $M = \tilde{\mathcal{O}}(N) = \tilde{\mathcal{O}}(2^{\frac{2b}{c}})$ and expected time

$$T = \tilde{\mathcal{O}}(N^{\frac{1}{2}} \cdot 2^{\frac{b}{2}}) = \tilde{\mathcal{O}}(2^{\frac{b}{c}} \cdot 2^{\frac{b}{2}}) = \tilde{\mathcal{O}}(2^{(\frac{1}{2}+\frac{1}{c})b}) \ .$$

Since PCS by definition returns N distinct collisions, this solves the c-SP$_b$. □

Putting Theorem 4.1 with its choice $b := \frac{\log c}{1-\varepsilon} \cdot \frac{k}{\log k}$ and Lemma 4.1 together, we immediately obtain the following LPN trade-off.

Theorem 4.2 (PCS-BKW)**.** *Let $\varepsilon > 0$, $c \in \mathbb{N}$ be even and $k \in \mathbb{N}$ be sufficiently large. Under the Independence Heuristic* $\mathtt{LPN}_k$ *can be solved with high probability in time*

$$T = 2^{\left(\frac{1}{2}+\frac{1}{c}\right)\cdot \log c \cdot \frac{k}{\log k}(1+\varepsilon)} \text{ using } M = 2^{\frac{2}{c}\cdot \log c \cdot \frac{k}{\log k}(1+\varepsilon)}$$

memory and samples.

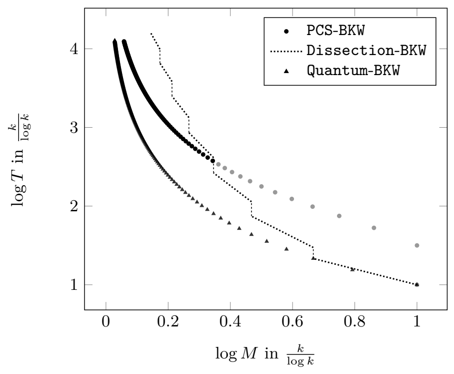


Fig. 6: The dotted marks depict our trade-off PCS-BKW, which improves for memory $M < 2^{0.35\frac{k}{\log k}}$ on the so far best classical algorithm Dissection-BKW from [14]. The triangle marks depict the best known quantum trade-off [14].

In Figure 6, we compare our new trade-off, called PCS-BKW, to Dissection-BKW and Quantum-BKW from Esser et al. [14]. In comparison to the so far best classical trade-off Dissection-BKW, based on the Dissection technique, our PCS-BKW improves on the run-time for any memory less than $2^{0.35\frac{k}{\log k}}$, or in other words it improves over any Dissection larger than an 11-Dissection. For very small memory we even come close to the time requirement of the quantum version Quantum-BKW with its highly memory-efficient Grover search.

Tradeoff	$2^{\log c \cdot \frac{k}{\log k}} =$
PCS-BKW	$T \cdot M^{\frac{c-2}{4}}$
Dissection-BKW [14]	$T \cdot M^{\sqrt{c}}$
Quantum-BKW [14]	$T \cdot M^{\frac{c-2}{2}}$

Table 2: Our LPN-trade-off PCS-BKW in comparison to [14].

In the commonly used time-memory trade-off notation, we obtain Table 2. We see that PCS-BKW shares with Quantum-BKW the linear dependency on c, whereas the previously best classical trade-off Dissection-BKW had only a square root dependency on c. In comparison with Quantum-BKW, for fixed T our algorithm needs only a square of the space requirement.

References

1. http://csrc.nist.gov/groups/ST/post-quantum-crypto/
2. Aggarwal, D., Dadush, D., Regev, O., Stephens-Davidowitz, N.: Solving the shortest vector problem in 2^n time using discrete Gaussian sampling: Extended abstract. In: Servedio, R.A., Rubinfeld, R. (eds.) 47th Annual ACM Symposium on Theory of Computing. pp. 733–742. ACM Press, Portland, OR, USA (Jun 14–17, 2015)
3. Albrecht, M.R., Cid, C., Faugere, J.C., Fitzpatrick, R., Perret, L.: On the complexity of the bkw algorithm on lwe. Designs, Codes and Cryptography 74(2), 325–354 (2015)
4. Austrin, P., Kaski, P., Koivisto, M., Määttä, J.: Space-time tradeoffs for subset sum: An improved worst case algorithm. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) ICALP 2013: 40th International Colloquium on Automata, Languages and Programming, Part I. Lecture Notes in Computer Science, vol. 7965, pp. 45–56. Springer, Heidelberg, Germany, Riga, Latvia (Jul 8–12, 2013)
5. Bai, S., Laarhoven, T., Stehlé, D.: Tuple lattice sieving. LMS Journal of Computation and Mathematics 19(A), 146–162 (2016)
6. 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)
7. 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)
8. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Krauthgamer, R. (ed.) 27th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 10–24. ACM-SIAM, Arlington, VA, USA (Jan 10–12, 2016)
9. 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)
10. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. In: 32nd Annual ACM Symposium on Theory of Computing. pp. 435–440. ACM Press, Portland, OR, USA (May 21–23, 2000)
11. Devadas, S., Ren, L., Xiao, H.: On iterative collision search for LPN and subset sum. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017: 15th Theory of Cryptography Conference, Part II. Lecture Notes in Computer Science, vol. 10678, pp. 729–746. Springer, Heidelberg, Germany, Baltimore, MD, USA (Nov 12–15, 2017)
12. Dinur, I.: An algorithmic framework for the generalized birthday problem. Designs, Codes and Cryptography pp. 1–30 (2018)
13. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 719–740. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
14. Esser, A., Heuer, F., Kübler, R., May, A., Sohler, C.: Dissection-BKW. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture

Notes in Computer Science, vol. 10992, pp. 638–666. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)

15. 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)
16. Fouque, P.A., Joux, A., Mavromati, C.: Multi-user collisions: Applications to discrete logarithm, Even-Mansour and PRINCE. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014, Part I. Lecture Notes in Computer Science, vol. 8873, pp. 420–438. Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014)
17. Guo, Q., Johansson, T., Stankovski, P.: Coded-BKW: Solving LWE using lattice codes. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 23–42. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)
18. Helm, A., May, A.: Subset sum quantumly in 1.17ˆ n. In: 13th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
19. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. Journal of the ACM (JACM) 21(2), 277–292 (1974)
20. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: Gilbert, H. (ed.) Advances in Cryptology – EUROCRYPT 2010. Lecture Notes in Computer Science, vol. 6110, pp. 235–256. Springer, Heidelberg, Germany, French Riviera (May 30 – Jun 3, 2010)
21. Joux, A., Lucks, S.: Improved generic algorithms for 3-collisions. In: Matsui, M. (ed.) Advances in Cryptology – ASIACRYPT 2009. Lecture Notes in Computer Science, vol. 5912, pp. 347–363. Springer, Heidelberg, Germany, Tokyo, Japan (Dec 6–10, 2009)
22. Kirchner, P., Fouque, P.A.: An improved BKW algorithm for LWE with applications to cryptography and lattices. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 43–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)
23. Kuhn, F., Struik, R.: Random walks revisited: Extensions of Pollard's rho algorithm for computing multiple discrete logarithms. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001: 8th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 2259, pp. 212–229. Springer, Heidelberg, Germany, Toronto, Ontario, Canada (Aug 16–17, 2001)
24. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology – ASIACRYPT 2011. Lecture Notes in Computer Science, vol. 7073, pp. 107–124. Springer, Heidelberg, Germany, Seoul, South Korea (Dec 4–8, 2011)
25. Nikolic, I., Sasaki, Y.: Refinements of the k-tree algorithm for the generalized birthday problem. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology – ASIACRYPT 2015, Part II. Lecture Notes in Computer Science, vol. 9453, pp. 683–703. Springer, Heidelberg, Germany, Auckland, New Zealand (Nov 30 – Dec 3, 2015)
26. Nikolic, I., Sasaki, Y.: A new algorithm for the unbalanced meet-in-the-middle problem. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 627–647. Springer, Heidelberg, Germany, Hanoi, Vietnam (Dec 4–8, 2016)

27. 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), https://doi.org/10.1137/0210033
28. Trimoska, M., Ionica, S., Dequen, G.: Time-memory trade-offs for parallel collision search algorithms. Cryptology ePrint Archive, Report 2017/581 (2017), https://eprint.iacr.org/2017/581
29. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. Journal of Cryptology 12(1), 1–28 (Jan 1999)
30. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) Advances in Cryptology – CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 288–303. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2002)

A Proof of Lemma 3.1

Lemma 3.1 (Number of collisions between g and g_t) *Let $n \in \mathbb{N}$ and $(\mathbf{a}, t)$ be a random subset sum instance. Then with high probability the number of collisions between g and g_t is at most $n \cdot 2^{H(1/4)n}$.*

Proof. By definition a collision between g and g_t is a tuple $(\mathbf{x}_1, \mathbf{x}_2) \in \mathcal{T}^2$ with $g(\mathbf{x}_1) + g(\mathbf{x}_2) = t \bmod 2^{H(1/4)n}$. Let us define indicator variables $X_{i,j}$ with $X_{i,j} = 1$ iff $g(\mathbf{x}_i) + g(\mathbf{x}_j) = t \bmod 2^{H(1/4)n}$ and let $X = \sum_{1 \le i,j \le |\mathcal{T}|} X_{i,j}$.

Let $i \neq j$ and $\mathbf{c} = \mathbf{x}_i + \mathbf{x}_j \in \{0, 1, 2\}^n$. Then $\mathbf{c}$ contains at least one 1-coefficient, wlog $c_1 = 1$. By the randomness of $\mathbf{a}$ in our subset sum instance we have

$$\begin{aligned}\Pr\left[X_{i,j} = 1 \mid i \neq j\right] &= \Pr\left[\langle \mathbf{a}, \mathbf{c}\rangle = t \bmod 2^{H(1/4)n}\right] \\ &= \Pr\left[a_1 = t - \sum_{i=2}^{n} c_i a_i \bmod 2^{H(1/4)n}\right] = \frac{1}{2^{H(1/4)n}} \quad .\end{aligned}$$

Thus, $X_{i,j} \sim \mathrm{Ber}_{2^{-\mathrm{H}(1/4)n}}$ for $i \neq j$. Using $|\mathcal{T}| = \binom{n}{n/4} \le 2^{H(1/4)n}$, we obtain

$$\mathbb{E}[X] \le (|\mathcal{T}|^2 - |\mathcal{T}|) \cdot 2^{-\mathrm{H}(1/4)n} + |\mathcal{T}| \le 2^{H(1/4)n+1} \quad .$$

An application of Markov's inequality yields

$$\Pr\left[X > n2^{H(1/4)n}\right] \le \frac{\mathbb{E}[X]}{n2^{H(1/4)n}} \le \frac{2}{n} \quad .$$

$\square$

B Proof of Theorem 3.2

Theorem 3.2 (Complexities of SS-PCS$_4$) *Let $(\mathbf{a}, t)$ be a random subset sum instance and let $\frac{1}{8} \le \gamma \le 0.21$. Then under Heuristic 2 with high probability* `SS-PCS`$_4$ *finds a solution to $(\mathbf{a}, t)$ in expected time $2^{(0.849-2\gamma)n}$ and memory $\tilde{\mathcal{O}}(2^{\gamma n})$.*

Proof. We start by analyzing the time complexity of the algorithm. The running time $T_{\texttt{it}}$ of each iteration of the second **for**-loop of Algorithm 2 (steps 5–9) is dominated by creating $L_1, \dots, L_4$ via `PCS` and checking for a solution with `Schroeppel-Shamir`. In the following we show that by our choice of γ the run time $T_{\texttt{it}}$ is solely dominated by `Schroeppel-Shamir`.

According to Theorem 2.1, computing $2^{\gamma n}$ collisions between f and f_{R_i} can be done in expected time

$$T_{\texttt{PCS}} = \tilde{\mathcal{O}}(2^{\frac{(H(1/16)+\gamma)n}{2}}) \quad .$$

By Heuristic 2, PCS gives us random collisions $(\mathbf{x}, \mathbf{y}) \in_R \mathcal{S}^2$, where $\mathcal{S} := \{\mathbf{x} \in \mathbb{F}_2^n \mid \mathrm{wt}(\mathbf{x}) = n/16\}$. We obtain

$$\Pr\left[(\mathbf{x}, \mathbf{y}) \in_R \mathcal{S}^2 \text{ is consistent}\right] = \frac{\binom{\frac{15}{16}n}{\frac{1}{16}n}}{\binom{n}{\frac{1}{16}n}} = \tilde{\Theta}\left(2^{(\frac{15}{16}H(1/15) - H(1/16))n}\right) \quad .$$

Let $\delta = (H(1/16) - \frac{15}{16}H(1/15))n$. Thus, the input lists for `Schroeppel-Shamir` have expected size $\tilde{\mathcal{O}}(2^{(\gamma-\delta)n})$. An easy calculation shows that the prerequisite of Lemma 2.1 is met. Therefore, an application of Lemma 2.1 yields that `Schroeppel-Shamir` takes expected time $T_{\mathtt{SS}} = \tilde{\mathcal{O}}(2^{2(\gamma-\delta)n})$. One also easily verifies that our restriction $\gamma \geq \frac{1}{8}$ from Theorem 3.2 implies $T_{\mathtt{SS}} \geq T_{\mathtt{PCS}}$, which entails that $T_{\mathtt{SS}}$ dominates $T_{\mathtt{it}}$. Moreover, the prerequisite $\gamma \leq 0.21$ guarantees that the exponent $4H(1/16) - \frac{1}{2} - 4\gamma$ of the number of iterations of the second **for**-loop is positive.
Therefore, the total expected time complexity is

$$T = \tilde{\mathcal{O}}(2^{(7\cdot H(1/16) - 3/2 - 4\gamma)n} \cdot T_{\mathtt{it}}) = \tilde{\mathcal{O}}(2^{(7\cdot H(1/16) - 3/2 - 2\gamma - 2\delta)n}) = 2^{(0.849 - 2\gamma)n}.$$

Concerning memory, we require to store $|L_i| = 2^{\gamma n}$ elements.

It remains to show that Algorithm 2 succeeds with high probability. Let us define the event E_1 that we find within the first **for**-loop a choice of $R_1, \ldots, R_3$ for which a representation $(\mathbf{e}_1, \ldots, \mathbf{e}_4)$ of our solution $\mathbf{e}$ exists. Further, let E_2 be the event that $(\mathbf{e}_1, \ldots, \mathbf{e}_4)$ is found within the second **for**-loop. We show that $E_1 \cap E_2$ happens with high probability in at least one of the iterations of the algorithm.

Let us start with event E_1. Let $\mathcal{R}_{\frac{n}{2},4}$ denote the number of representations of a vector with weight $\frac{n}{2}$ into four vectors, i.e.

$$\mathcal{R}_{\frac{n}{2},4} = \binom{n/2}{n/8, n/8, n/8, n/8} \geq \frac{2^n}{n^3} \quad .$$

By Heuristic 2 for a representation $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4)$ the values $\langle \mathbf{a}, \mathbf{e}_i \rangle \mod 2^{H(1/16)n}$, for $i = 1, 2, 3$ are independently and uniformly distributed over $\mathbb{Z}_{2^{H(1/16)n}}$. Thus, a fixed choice of $R_1, R_2, R_3 \in \mathbb{Z}_{2^{H(1/16)n}}$ does not yield a representation with probability

$$\Pr\left[R_1, R_2, R_3 \text{ bad}\right] = (1 - 2^{-3H(1/16)n})^{\mathcal{R}_{\frac{n}{2},4}} \leq (1 - 2^{-3H(1/16)n})^{\frac{2^n}{n^3}} \quad .$$

Using $1 - x \leq e^{-x}$, in all $n^4 \cdot 2^{(3\cdot H(1/16) - 1)n}$ iterations of the first **for**-loop we find no representation with probability

$$\Pr\left[\bar{E_1}\right] \leq (1 - 2^{-3H(1/16)n})^{n\cdot 2^{3H(1/16)n}} \leq e^{-n} \quad .$$

It remains to show that $E_2|E_1$ happens with high probability. As we condition on E_1, we already fixed R_1, R_2, R_3 for which there exist a representation $(\mathbf{e}_1, \ldots, \mathbf{e}_4)$ with $\langle \mathbf{a}, \mathbf{e}_i \rangle = R_i \mod 2^{H(1/16)n}$ for $i = 1, 2, 3$. We now have to lower bound the probability that $(\mathbf{e}_1, \ldots, \mathbf{e}_4) \in L_1 \times \ldots \times L_4$.

Each element from L_i is constructed via PCS, where by Heuristic 2 PCS returns independently and uniformly at random drawn collisions. Let us represent $\mathbf{e}_i$ as $(\mathbf{e}_i^{(1)}, \mathbf{e}_i^{(2)})$, which can be done in $R_{\frac{n}{8},2} = \binom{n/8}{n/16} \geq \frac{1}{n} \cdot 2^{\frac{n}{8}}$ many ways.

Moreover, similar to the proof of Lemma 3.1 we know that with high probability the number of collisions between f and f_{R_i} is upper-bounded by $n \cdot 2^{\mathrm{H}(1/16)n}$. Hence, a random collision from PCS yields $\mathbf{e}_i$ with probability

$$q \geq \frac{\mathcal{R}_{\frac{n}{8},2}}{n \cdot 2^{\mathrm{H}(1/16)n}} \geq \frac{1}{n^2} \cdot 2^{(-\mathrm{H}(1/16)+1/8)n} \quad .$$

As an execution of PCS provides us $2^{\gamma n}$ collisions we obtain

$$p := \Pr\left[L_i \text{ contains } \mathbf{e}_i\right] = 1 - (1-q)^{2^{\gamma n}} \quad .$$

It follows that

$$\Pr\left[(\mathbf{e}_1, \ldots, \mathbf{e}_4) \in L_1 \times \ldots \times L_4\right] = p^4 = \left(1 - (1-q)^{2^{\gamma n}}\right)^4 \quad .$$

Let $Y \sim \mathrm{Geo}_{p^4}$ be a random variable for the number of iterations until $(\mathbf{e}_1, \ldots, \mathbf{e}_4) \in L_1 \times \ldots \times L_4$. Using Bernoulli's inequality $(1-x)^n \geq 1 - xn$ we obtain

$$\mathbb{E}[Y] = \frac{1}{\left((1-q)^{2^{\gamma n}} - 1\right)^4} \leq \frac{1}{(1 - q \cdot 2^{\gamma n} - 1)^4} \leq n^8 \cdot 2^{(4H(1/16) - \frac{1}{2} - 4\gamma)n} \quad .$$

Using Markov's inequality, SS-PCS$_4$ does not succeed to find a solution to the random subset sum instance $(\mathbf{a}, t)$ in its $n^9 \cdot 2^{(4H(1/16)-\frac{1}{2}-4\gamma)n}$ iterations of the second **for**-loop with

$$\Pr\left[\bar{E}_2 \mid E_1\right] = \Pr\left[Y > n^9 \cdot 2^{(4H(1/16)-\frac{1}{2}-4\gamma)n}\right] \leq \frac{\mathbb{E}[Y]}{n^9 \cdot 2^{(4H(1/16)-\frac{1}{2}-4\gamma)n}} \leq \frac{1}{n} \quad .$$

□

C Experimental Verification of Heuristic 1

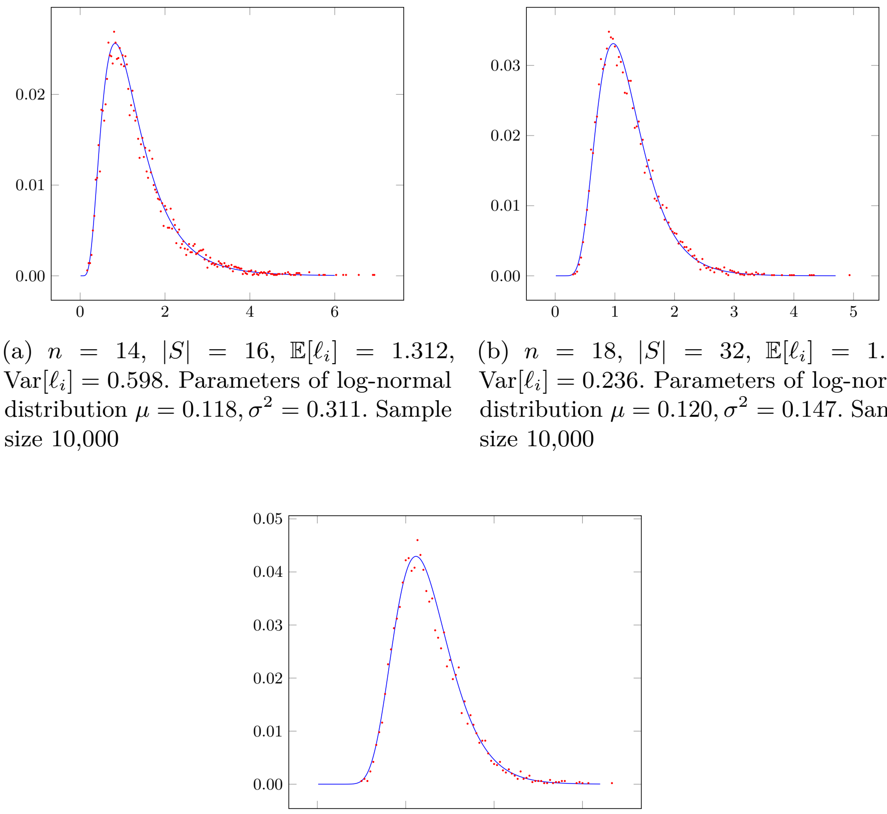
In this section we present experimental results that verify the used heuristic assumptions.

Distribution of Collisions. Our analyses assume that collision sampling via PCS yields independently and uniformly distributed collisions. Let C be the set of all collisions of some function g and let $S \subseteq C$ be a distinguished subset. By our assumption we hit S with probability $p = \frac{|S|}{|C|}$.

We tested this heuristic for functions with domain size 2^n, where $n \in \{14, 18, 22\}$ by measuring the amount of collisions until we hit S for the first time, which exactly determines the running time of our algorithms in Section 3 and should be geometrically distributed with parameter p. To this end we generated a random subset sum instance $(\mathbf{a}, t)$ and constructed a function g mapping $\mathbf{x} \in \mathbb{F}_2^n$

to $\langle \mathbf{a}, \mathbf{x} \rangle \bmod 2^n$. We then enumerated all collisions C of g and randomly chose $S \subseteq C$.

We experimentally observed that the distribution of required collisions until we first hit S is indeed geometric. Moreover, for various different functions g_i we compared the experimentally observed geometric parameters p_i to the expected p. Let $\ell_i = \frac{p}{p_i}$ denote their quotient, so ℓ_i should be close to 1. In Figure 7 we show the distribution of the ℓ_i in our experiments, where the dots represent the relative frequencies of the ℓ_i. The ℓ_i closely follow a logarithmic-normal distribution – depicted as a solid line – centered around the desired value of one. Moreover, we see that for increasing n the variance of the distribution decreases. Thus, ℓ_i becomes sharply centred around one.



(a) $n = 14$, $|S| = 16$, $\mathbb{E}[\ell_i] = 1.312$, $\mathrm{Var}[\ell_i] = 0.598$. Parameters of log-normal distribution $\mu = 0.118, \sigma^2 = 0.311$. Sample size 10,000

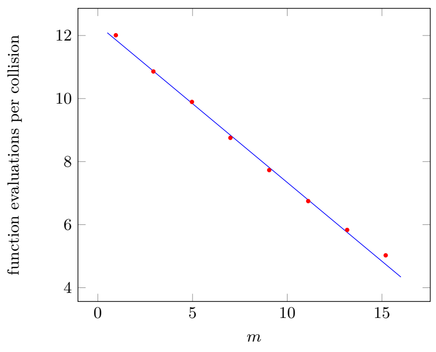
(b) $n = 18$, $|S| = 32$, $\mathbb{E}[\ell_i] = 1.214$, $\mathrm{Var}[\ell_i] = 0.236$. Parameters of log-normal distribution $\mu = 0.120, \sigma^2 = 0.147$. Sample size 10,000

(c) $n = 22$, $|S| = 64$, $\mathbb{E}[\ell_i] = 1.252$, $\mathrm{Var}[\ell_i] = 0.128$. Parameters of log-normal distribution $\mu = 0.186, \sigma^2 = 0.076$. Sample size 5,000

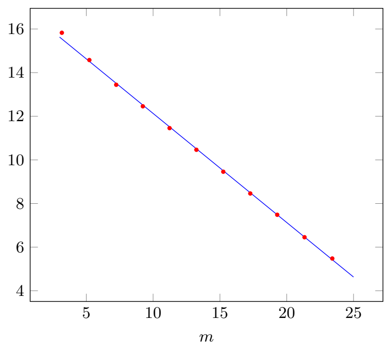
Fig. 7: Distribution of the ℓ_i for $(n, \log|S|) \in \{(14, 4), (18, 5), (22, 6)\}$.

Complexity of PCS Algorithm Applied to Subset Sum Functions. According to Theorem 2.1, the PCS algorithm performs on independent random functions (roughly) $2^{\frac{r+m}{2}}$ evaluations for finding 2^m collisions. This implies on average $2^{\frac{r-m}{2}}$ evaluations per collision.

We verify this asymptotic behaviour experimentally for our subset sum functions g and g_t from Section 3.1. We implemented g, g_t for $n \in \{28, 40\}$ and measured the average amount of function evaluations to obtain a specific number of collisions via PCS. Figure 8 shows the results in logarithmic scale, where the dots represent the experimental data averaged over multiple executions. The solid line represents the asymptotic prediction of $\frac{r-m}{2}$ (shifted by a small additive constant that stems from the $\tilde{\mathcal{O}}$-notion). We see that the average cost of multiple collision finding in g and g_t closely matches the prediction from Theorem 2.1, and thus g, g_t behave with respect to multiple collision finding like independent random functions.



(a) $n = 28$, $r = \log\binom{28}{7} \approx 20$. Each data point is averaged over 30 executions



(b) $n = 40$, $r = \log\binom{40}{10} \approx 30$. Each data point is averaged over 15 executions

Fig. 8: Average number of function evaluations per collision (in logarithmic scale, y-axis) for generating 2^m collisions (x-axis) for $n \in \{28, 40\}$.