New Time-Memory Trade-Offs for Subset Sum – Improving ISD in Theory and Practice

Andre Esser[1] and Floyd Zweydinger[2*]

[1] Technology Innovation Institute, UAE
andre.esser@tii.ae
[2] Ruhr University Bochum, Germany
floyd.zweydinger@rub.de

Abstract. We propose new time-memory trade-offs for the random subset sum problem defined on $(a_1, \ldots, a_n, t)$ over $\mathbb{Z}_{2^n}$.
Our trade-offs yield significant running time improvements for every fixed memory limit $M \geq 2^{0.091n}$. Furthermore, we interpolate to the running times of the fastest known algorithms when memory is not limited. Technically, our design introduces a pruning strategy to the construction by Becker-Coron-Joux (BCJ) that allows for an exponentially small success probability. We compensate for this reduced probability by multiple randomized executions. Our main improvement stems from the clever reuse of parts of the computation in subsequent executions to reduce the time complexity per iteration.
As an application of our construction, we derive the first non-trivial time-memory trade-offs for Information Set Decoding (ISD) algorithms. Our new algorithms improve on previous (implicit) trade-offs asymptotically as well as practically. Moreover, our optimized implementation also improves on *running time*, due to reduced memory access costs. We demonstrate this by obtaining a new record computation in decoding quasi-cyclic codes (QC-3138). Using our newly obtained data points we then extrapolate the hardness of suggested parameter sets for the NIST PQC fourth round candidates McEliece, BIKE and HQC, lowering previous estimates by up to 6 bits and further increasing their reliability.

Keywords: representation technique · information set decoding · code-based cryptography · record computation · security estimates · NIST PQC

1 Introduction

For the ongoing NIST PQC standardisation process to be successful, large cryptanalytic efforts analysing the involved primitives are required. This includes theoretical studies of the asymptotically best attacks as well as experiments on a meaningful scale to safely extrapolate the hardness of cryptographic-sized instances. This methodology, combining theory and practice, is well established

* Funded by BMBF under Industrial Blockchain – iBlockchain

for conventional (number-theoretic) cryptographic systems and has found its adaptation to post-quantum secure schemes in recent years [1, 20, 25, 28, 41].

The best attacks on post-quantum schemes often suffer from high memory demands [6,8,9,11,18]. This either leads to an immense slowdown of the algorithm due to physical access times or, in the worst case, prevents its application entirely. In practice, both cases usually lead to a fallback to more memory-efficient but asymptotically inferior procedures. In these cases *time-memory trade-offs* for the best algorithms are needed which allow to tailor their memory consumption to any given amount while (only slightly) increasing their running time.

For post-quantum secure candidates, especially from code- and lattice-families, several of the known attacks are built on techniques initially introduced in the context of the (random) subset sum problem [8, 15, 32, 33]. This is because the underlying problems can usually be formulated as (vectorial) variants of subset sum, as it is the case for LPN / LWE, SIS or the syndrome decoding problem.

The subset sum problem defined on $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{2^n}$ asks to find a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = t \bmod 2^n$. For this problem time-memory trade-offs are actually well studied [4,17,19,30]. However, the translations of those trade-offs to the aforementioned applications are mostly missing. The reason is the very diverse landscape of optimal trade-offs for subset sum, i.e., for different memory limitations there exist different optimal trade-offs. Furthermore, these trade-offs often do not match the design of the fastest subset sum algorithm used in the original application, which implies a separate translation effort for each algorithm.

In this work we construct new improved time-memory trade-offs for the subset sum problem. In contrast to previous works, our constructions follow the design by Becker-Coron-Joux (BCJ) [7], which is the basis for the fastest known algorithms. This allows for an easy adaptation of our trade-off to known applications of the BCJ algorithm. Further, our trade-offs reduce the running time of previous approaches for any fixed memory significantly. Only for very small available memory a trade-off based on a memory-less algorithm by Esser and May [27] becomes favourable. In total this reduces the trade-off landscape to only two algorithms.

We illustrate the potential of our trade-off by formalizing its application to the syndrome decoding problem, whose hardness forms the basis of code-based cryptography. Informally, the problem asks to find a low Hamming weight solution $\mathbf{e} \in \mathbb{F}_2^n$ to the matrix-vector equation $\mathbf{He} = \mathbf{s}$, where $\mathbf{H} \in \mathbb{F}_2^{r \times n}$ and $\mathbf{s} \in \mathbb{F}_2^r$. Moreover, it allows for a direct translation to a vectorial subset sum variant. Denote by $\mathbf{h}_i$ the columns of $\mathbf{H}$, then $(\mathbf{h}_1, \ldots, \mathbf{h}_n, \mathbf{s})$ defines a subset sum instance over $\mathbb{F}_2^n$, i.e., we are looking for a small subset of the $\mathbf{h}_i$ that sums to $\mathbf{s}$ over $\mathbb{F}_2^n$.

Information Set Decoding (ISD) algorithms now solve this problem by first applying a dimension reduction technique, which yields an instance with decreased n, r and smaller solution weight. Then an adaptation of the BCJ subset sum algorithm over $\mathbb{F}_2$ is applied to solve this reduced instance. Since the dimension reduction technique, in contrast to the subset sum algorithm, does not require any memory, every ISD algorithm inherits a naive time-memory trade-off. That is, reduce the instance size sufficiently so that the latter applied BCJ algorithm does not exceed the given memory. So far this simple interpolation to a full dimension-reduction based ISD algorithm proposed by Prange in 1962 [37], was the best known trade-off strategy. Our adaptation now yields the first time-memory trade-offs for advanced ISD algorithms improving their performance asymptotically as well as in practice.

1.1 Related Work

Subset Sum. Any subset sum instance can be solved in time and memory $\mathrm{poly}(n) \cdot 2^{\frac{n}{2}}$ via a meet-in-the-middle algorithm [29]. Schroeppel and Shamir [38] then showed how to reduce the memory complexity to $\mathrm{poly}(n) \cdot 2^{\frac{n}{4}}$. Later, their technique formed the basis for a series of advanced time-memory trade-offs [16, 17, 19].

The second key-ingredient for most subset sum trade-offs [7, 16, 27, 30] is the so-called *representation technique* introduced by Howgrave-Graham and Joux (HGJ) in [30]. In their work they constructed the first algorithm breaking the $2^{\frac{n}{2}}$ time bound for *random* subset sum instances by achieving running time $2^{0.337n}$. In the cryptographic setting we usually encounter random instances, i.e., the vector $\mathbf{a} := (a_1, \ldots, a_n)$ is chosen uniformly at random and the target is set to $t = \langle \mathbf{a}, \mathbf{e} \rangle$ for a randomly chosen solution vector $\mathbf{e} \in \{0, 1\}^n$ of Hamming weight $\frac{n}{2}$. Howgrave-Graham and Joux then split the solution $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ with $\mathbf{e}_i \in \{0, 1\}^n$ of weight $n/4$. Now, there exist multiple, namely $\binom{n/2}{n/4}$, such *representations* of $\mathbf{e}$, i.e., different combinations $\mathbf{e}_1, \mathbf{e}_2$ that sum to $\mathbf{e}$. The core observation is that it suffices to find a single of these representations to recover the solution. This representation is then constructed using a search-tree imposing restrictions on the exact form of the solution (similar to Wagners k-tree algorithm [43]) so that in expectation one representation satisfies all restrictions. Becker, Coron and Joux (BCJ) [7] improved the running time to $2^{0.291n}$ by choosing $\mathbf{e}_i \in \{-1, 0, 1\}^n$ to increase the amount of representations. Later Bonnetain, Bricout, Schrottenloher and Shen (BBSS) [12] further extended the digit set to $\mathbf{e}_i \in \{-1, 0, 1, 2\}^n$ yielding a time and memory complexity of $2^{0.283n}$.

As mentioned, the time-memory trade-off landscape for subset sum is diverse [7, 16, 19, 22, 27, 30]. Additionally, there are several techniques [17, 19, 36] improving the time-memory behaviour of the k-tree algorithm, which forms the foundation of the fastest known subset sum algorithms. However, since these techniques usually introduce asymmetries in the matching algorithm, which are inherently difficult to combine with the representation technique, they did not find a broad adaptation in trade-offs for the subset sum problem yet.

Information Set Decoding. ISD algorithms are the fastest known algorithms to solve general instances of the syndrome decoding problem and form the basis in assessing the security of code-based schemes. Introduced originally by Prange [37], the class was extended by several improved algorithms over the years [9, 21, 33, 34, 39]. All these works improve the running time by using more advanced subset sum techniques to solve the reduced instance after dimension reduction, which simultaneously increases the memory requirements. Surprisingly, there has been very limited work on time-memory trade-offs for ISD algorithms. Karpman and Lefevre [31] recently constructed advanced time-memory trade-offs for the special case of decoding ternary codes based on a subset sum trade-off strategy known as Dissection [19]. Further, a work by Wang et al. [44] extends an early ISD algorithm from Stern [39] by the Dissection approach. However, this trade-off is entirely outperformed by the previously mentioned implicit trade-offs of more advanced ISD procedures.

1.2 Our Contribution

Subset Sum. As a first contribution we give a generalized description of the BCJ algorithm, that combines previous interpretations from [12, 26]. This description then forms the basis for one of our main contributions which are new time-memory trade-offs for the random subset sum problem. Our constructions yield significantly improved running times for every fixed memory $M \geq 2^{0.091n}$, which corresponds to more than two-thirds of the meaningful memory parameters. Recall that $M = 2^{0.283n}$ memory is sufficient to instantiate the fastest known algorithm with time complexity $T = M$. In Figure 1 we illustrate the performance of our new trade-offs in comparison to previous works. For example, if the memory is limited to $2^{0.17n}$, we improve the running time from $2^{0.51n}$ down to $2^{0.4n}$, corresponding to an improvement by a factor of $2^{0.11n}$.

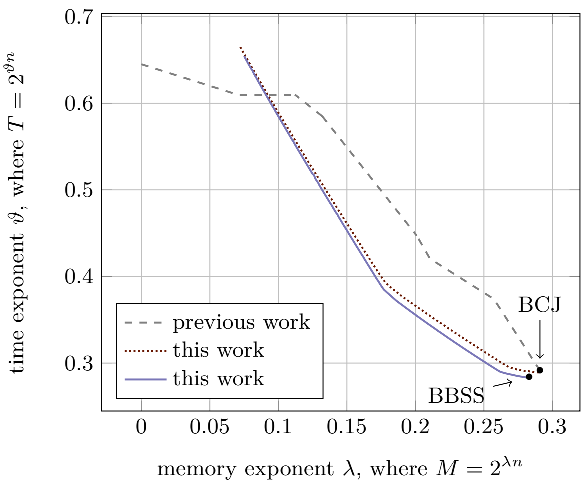

Fig. 1: Our new subset sum trade-offs in comparison to the previously best known time-memory trade-offs. The dashed line illustrates the minimum running time over the algorithms given in [7, 16, 19, 22, 27, 30]. The dotted and solid lines are obtained via our trade-off Algorithm 2 (see Section 4). For a memory larger than $2^{0.091n}$ ($2^{0.093n}$ resp.) our new trade-offs are superior to previous approaches.

From a technical side we allow the BCJ and BBSS construction to impose larger restrictions on the representation-space, yielding an exponentially small success probability. We then perform multiple randomized executions to compensate for the reduced probability. In this context we introduce a novel strategy of reusing lower levels of the search-tree in subsequent randomized executions to reduce the time complexity per iteration. Note that while Dinur in [17] also reuses the first level of his list construction in later repetitions, this is motivated by the use of different algorithms to construct the first and later levels. An asymmetry that makes the incorporation of representations even more difficult. In contrast our technique is symmetric, allows for easy incorporation of representations and, moreover, precisely exploits this embedding of representations when reusing lists in later stages. Also our technique extends well to every level of the construction.

Furthermore, to obtain instantiations for small memory parameters and to further reduce the time complexity, we then integrate the Dissection framework [19] in our construction, inspired by the combination of Wagners k-tree and Dissection in [17].

Information Set Decoding. We give the first non-trivial time-memory trade-offs for advanced ISD algorithms by combining our trade-offs with the ISD algorithms by May-Meuer-Thomae (MMT) [33] and Becker-Joux-May-Meurer (BJMM) [9]. Overall this yields asymptotic improved running times for every fixed memory. Moreover, for the MMT algorithm we are able to improve the memory, while maintaining its running time.

On the practical side, we extend the fastest implementation of the MMT / BJMM algorithm from [28] by our trade-off strategy observing memory and *time* improvements. Using our optimized implementation we obtain a new record computation in decoding quasi-cyclic codes (QC-3138) [3]. Further we re-break several old records, consuming less resources, i.e., time and memory. Hence, our trade-off is the first asymptotic improvement of the MMT algorithm that transfers to the implementation level. Eventually, using our newly obtained data-points in combination with an estimation script we extrapolate the hardness of suggested parameter sets for code-based NIST PQC fourth round candidates McEliece, BIKE and HQC, resulting in reduced security estimates by up to 6 bits compared to previous works. This improvement is even more significant considering that the bit-complexity estimates of code-based schemes have essentially been stable over the past decades, which is especially true for quasi-cyclic schemes. In this context, we provide estimates following two different methodologies, a conventional approximation of the bit complexity and an extrapolation method based on our practical experiments, recently suggested in [28]. Overall we find that both methods paint a comparable picture regarding the security claims of proposed parameter sets, invalidating claims that the latter method would lead to drastically decreased estimates [42].

All our used estimation and optimization scripts are available at https://github.com/FloydZ/Improving-ISD-in-Theory-and-Practice.[3] Our adapted implementation of the BJMM algorithm can be found at https://github.com/FloydZ/Decoding.

Outline. In Section 2 we set up necessary notation and cover some basics on the Dissection technique. Subsequently, in Section 3 we give the generalized description of the BCJ algorithm, which is then used as a basis to build our new trade-offs in Section 4. Eventually, in Section 5 we give the asymptotic and practical results of our decoding application including security estimates for all NIST PQC candidates of the ongoing forth round.

2 Preliminaries

All logarithms are base 2. We define $H(x) := -x\log(x) - (1-x)\log(1-x)$ to be the binary entropy function with H^{-1} its inverse on $[0, \frac{1}{2}]$. Extending this definition, we also use the 2-way entropy function defined as $g(x, y) := -x\log(x) - y\log(y) - (1-x-y)\log(1-x-y)$. We simplify binomial and multinomial coefficients via Sterling's formula as

$$\binom{n}{\alpha n} \simeq 2^{nH(\alpha)} \text{ and } \binom{n}{\alpha n, \beta n, \cdot} \simeq 2^{ng(\alpha,\beta)},$$

where $\binom{n}{\alpha n, \beta n, \cdot} := \binom{n}{\alpha n, \beta n, (1-\alpha-\beta)n}$. We use standard landau notation, with $\tilde{\mathcal{O}}$-notation suppressing poly-logarithmic factors and write $A = \tilde{\mathcal{O}}(B)$ as $A \simeq B$. Our asymptotic complexity statements are all to be understood up to poly-logarithmic factors, even though we sometimes drop the $\tilde{\mathcal{O}}$ for convenience.

For a vector $\mathbf{x} \in \mathbb{F}_2^n$ we denote by $\mathrm{wt}(\mathbf{x})$ its Hamming weight. Additionally we denote by $\langle \mathbf{x}, \mathbf{y} \rangle$ the inner product of two vectors $\mathbf{x}, \mathbf{y}$.

All our algorithms target the random subset sum problem defined as follows, even if we might omit the term *random* sometimes.

Definition 2.1 (Random Subset Sum Problem). *Let* $\mathbf{a} := (a_1, \ldots, a_n) \in \mathbb{Z}_{2^n}^n$ *be drawn uniformly at random. For a random* $\mathbf{e} \in \{0,1\}^n$ *with* $\mathrm{wt}(\mathbf{e}) = \frac{n}{2}$*, let* $t := \langle \mathbf{a}, \mathbf{e} \rangle$*. The* random subset sum problem *is given* $(\mathbf{a}, t)$ *find any* $\mathbf{e}' \in \{0,1\}^n$ *satisfying* $\langle \mathbf{a}, \mathbf{e}' \rangle = t$*. We call any such* $\mathbf{e}'$ *a* solution *and* $(\mathbf{a}, t)$ *an* instance.

Our definition of the subset sum problem asks for a solution in $\{0,1\}^n$. However, algorithms like the BCJ algorithm approach the problem in a divide-and-conquer manner, which requires solving sub-instances with solutions in a different domain D. These sub-instances are usually solved via a meet-in-the-middle strategy, which we later exchange by a more memory efficient strategy known as Dissection.

[3] Our numerical optimization scripts are based on a code by Bonnetain et. al [12] accessible at https://github.com/xbonnetain/optimization-subset-sum.

Schroeppel-Shamir and Dissection A standard meet-in-the-middle solves a subset sum instance with solution in a set D in time and memory $|D|^{\frac{1}{2}}$ [29]. Therefore it first splits $D = D_1 \times D_2$, with $|D_i| = |D|^{\frac{1}{2}}$, enumerates all possible elements of D_1 and D_2 separately in lists L_1 and L_2 and then searches for a solution in D by combining elements from L_1 and L_2. The algorithm by Schroeppel and Shamir [38] now achieves the same time complexity of $|D|^{\frac{1}{2}}$ while improving the memory complexity to $|D|^{\frac{1}{4}}$. It works similarly by first splitting $D = D_1 \times D_2 \times D_3 \times D_4$ with $|D_i| = |D|^{\frac{1}{4}}$ and then enumerating all elements of D_i in lists L_i. Next an artificial constraint is introduced restricting the search to solutions which lie in a specific subset $(D_{12} \times D_{34}) \subseteq D$. This constraint is used to combine elements from L_1 and L_2 to obtain only elements from D_{12} in a new list L_{12} and analogously elements from D_{34} in a new list L_{34} by combining L_3 and L_4. From there the two lists L_{12} and L_{34} are combined as in the usual meet-in-the-middle case to search for a solution in D. As a priori it is not known in which subset the solution is located the algorithm partitions D in multiple subsets and re-applies the procedure for each of them. The Dissection framework introduced in [19] offers instantiations with less memory in form of a continues time-memory trade-off starting from the Schroeppel-Shamir algorithm. Besides the Schroeppel-Shamir algorithm our constructions make use of another instantiation of this framework, a so-called 7-Dissection. A 7-Dissection runs in time $|D|^{4/7}$ and uses memory $|D|^{1/7}$. Moreover, with more memory its time complexity can be gradually decreased until it reaches the complexity of the Schroeppel-Shamir algorithm. Technically a 7-Dissection works similar to the Schroeppel-Shamir technique by initially splitting $D = D_1 \times \ldots \times D_7$ and creating seven corresponding lists. Then multiple times artificial constraints are introduced to combine the lists most effectively, while, eventually, the algorithms is iterated for each possible choice of the constraints.

We summarize the time and memory complexity of the 7-Dissection in the following lemma. For more details on the dissection framework the reader is referred to [19].

Lemma 2.1 (7-Dissection, [19]). *Let $\frac{1}{7} \leq \lambda \leq \frac{1}{4}$. The 7-Dissection algorithm finds all solutions $\mathbf{e} \in D$ to a random subset sum instance in expected time $|D|^{\frac{2(1-\lambda)}{3}}$ and expected memory $|D|^{\lambda}$.*

3 The generalized BCJ Algorithm

In this section we give a general description of the BCJ algorithm [7] for solving the random subset sum problem. This description forms the basis for our new trade-offs presented in the following section. We advise the reader to follow Figure 2. In our exposition we assume a certain familiarity of the reader with the representation technique, otherwise we refer to [7, 30] for an introduction.

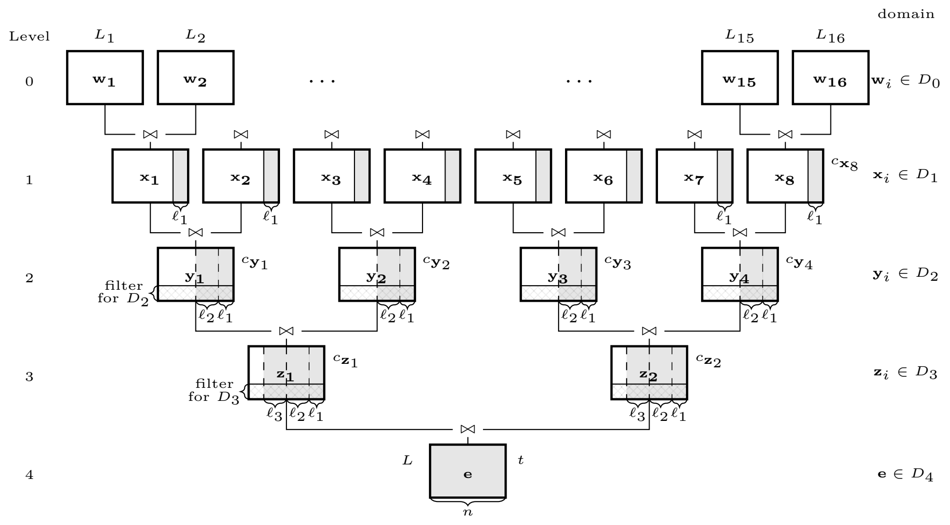

Fig. 2: Generalized tree construction of the BCJ Algorithm in depth 4. Shaded areas on the right of a list L indicate that for all elements $\mathbf{v} \in L$ the inner product $\langle \mathbf{a}, \mathbf{v} \rangle$ matches a predefined value $c_{\mathbf{v}}$ (resp. t) on those bits.

Basic idea To construct a solution $\mathbf{e}$ of the subset sum problem the BCJ algorithm splits $\mathbf{e}$ in the sum of two addends, i.e.,

$$\mathbf{e} = \mathbf{z}_1 + \mathbf{z}_2 \ .$$

Here the $\mathbf{z}_i$ are chosen from a set, such that there exist multiple different *representations* of the solution, i.e., different tuples that sum to $\mathbf{e}$. The goal is then to examine a respective fraction of the space of the $\mathbf{z}_1, \mathbf{z}_2$ to find one of these representations.

From $\langle \mathbf{a}, \mathbf{e} \rangle = \langle \mathbf{a}, \mathbf{z}_1 + \mathbf{z}_2 \rangle = t \mod 2^n$ we have by linearity

$$\langle \mathbf{a}, \mathbf{z}_1 \rangle = t - \langle \mathbf{a}, \mathbf{z}_2 \rangle \mod 2^n. \tag{1}$$

Note that the value of $\langle \mathbf{a}, \mathbf{z}_1 \rangle$ is not known. However, by considering only those $\mathbf{z}_1$ which fulfill $\langle \mathbf{a}, \mathbf{z}_1 \rangle = c_{\mathbf{z}_1} \mod 2^\ell$ for some fixed integer $c_{\mathbf{z}_1}$ we are able to impose a constraint on the search space. Here $\ell := \ell_1 + \ell_2 + \ell_3$ is an optimization parameter of the algorithm, with the ℓ_i's being positive integers. Moreover, since each representation of $\mathbf{e}$ fulfills Equation (1) the value of $c_{\mathbf{z}_2} := \langle \mathbf{a}, \mathbf{z}_2 \rangle = t - c_{\mathbf{z}_1} \mod 2^\ell$ is fully determined once $c_{\mathbf{z}_1}$ is fixed.

The construction of the $\mathbf{z}_1$ and $\mathbf{z}_2$ then works recursively. Therefore, they are split again in the sum of two addends

$$\mathbf{z}_1 = \mathbf{y}_1 + \mathbf{y}_2 \text{ and } \mathbf{z}_2 = \mathbf{y}_3 + \mathbf{y}_4 \ ,$$

and we fix the values $\langle \mathbf{a}, \mathbf{y}_1 \rangle$ and $\langle \mathbf{a}, \mathbf{y}_3 \rangle$ to some constraints $c_{\mathbf{y}_1} \mod 2^{\ell_1+\ell_2}$ and $c_{\mathbf{y}_3} \mod 2^{\ell_1+\ell_2}$. Note that this again determines the inner product of the remaining addends for any representation $(\mathbf{y}_1, \mathbf{y}_2)$ of $\mathbf{z}_1$ and $(\mathbf{y}_3, \mathbf{y}_4)$ of $\mathbf{z}_2$ as

$$c_{\mathbf{y}_2} := \langle \mathbf{a}, \mathbf{y}_2 \rangle = c_{\mathbf{z}_1} - c_{\mathbf{y}_1} \bmod 2^{\ell_1+\ell_2} \text{ and } c_{\mathbf{y}_4} := \langle \mathbf{a}, \mathbf{y}_4 \rangle = c_{\mathbf{z}_2} - c_{\mathbf{y}_3} \bmod 2^{\ell_1+\ell_2}$$

The recursion continues once more by splitting the $\mathbf{y}_i = \mathbf{x}_{2i-1} + \mathbf{x}_{2i}$ and introducing four additional modular constraints $c_{\mathbf{x}_{2i-1}} \bmod 2^{\ell_1}$. These modular constraints together with the $c_{\mathbf{y}_i}$'s determine the values of inner products $c_{\mathbf{x}_{2i}} := \langle \mathbf{a}, \mathbf{x}_{2i} \rangle \bmod 2^{\ell_1}$, since we have

$$\begin{aligned} c_{\mathbf{z}_2} &:= t - c_{\mathbf{z}_1} && \bmod 2^{\ell} \\ c_{\mathbf{y}_{2i}} &:= c_{\mathbf{z}_i} - c_{\mathbf{y}_{2i-1}} && \bmod 2^{\ell_1+\ell_2} \quad, i = 1, 2 \\ c_{\mathbf{x}_{2i}} &:= c_{\mathbf{y}_i} - c_{\mathbf{x}_{2i-1}} && \bmod 2^{\ell_1} \quad, i = 1, 2, 3, 4 \end{aligned} \tag{2}$$

Eventually, the $\mathbf{x}_i$'s are split in a meet-in-the-middle fashion, i.e.,

$$\mathbf{x}_i = (\mathbf{w}_{2i-1}, 0^{n/2}) + (0^{n/2}, \mathbf{w}_{2i}),$$

giving only a single representation of each $\mathbf{x}_i$.

The algorithm now starts by enumerating all possible values for the $\mathbf{w}_i$ in the base lists L_i. Then two lists are merged at a time in a new list by only considering those elements which fulfill the current constraint modulo 2^{ℓ_1}, $2^{\ell_1+\ell_2}$, 2^{ℓ} or 2^n respectively (compare to Figure 2). After the level-i list construction only those elements are kept whose coordinates follow a predefined distribution D_i, while all others are discarded. The choice of these distributions mainly determines the existing amount of representations and ultimately the performance of the algorithm. We give the pseudocode of the procedure in Algorithm 1.

Complexity Let the expected list sizes *before* filtering on level i be $\mathcal{L}_i$ and let the probability of any element of a level-i list surviving the filter be q_i. Since the level-1 lists are constructed from the Cartesian product of the level-0 lists by enforcing a modular constrained on ℓ_1 bits we have

$$\mathcal{L}_1 = \frac{(\mathcal{L}_0)^2}{2^{\ell_1}}.$$

Analogously the level-2 lists are constructed from the *filtered* level-1 lists by enforcing a modular constrained on $\ell_1 + \ell_2$ bits. However, since the last ℓ_1 bits are already fixed to some value in the previous step we only enforce a new constraint on ℓ_2 bits, which results in

$$\mathcal{L}_2 = \frac{(q_1 \cdot \mathcal{L}_1)^2}{2^{\ell_2}}.$$

Analogously we obtain

$$\mathcal{L}_3 = \frac{(q_2 \cdot \mathcal{L}_2)^2}{2^{\ell_3}} \text{ and } \mathcal{L}_4 = \frac{(q_3 \cdot \mathcal{L}_3)^2}{2^{n-\ell}}.$$

Algorithm 1: BCJ ALGORITHM

```
Input  : $\mathbf{a} \in (\mathbb{Z}_{2^n})^n, t \in \mathbb{Z}_{2^n}$
Output : $\mathbf{e} \in \mathbb{F}_2^n$ with $\langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n$
1 Choose optimal $\ell_1, \ell_2, \ell_3$ and $D_i,\ i = 0, 1, 2, 3$
2 Enumerate
      $L_{2i-1} = \{\mathbf{w}_{2i-1} \mid \mathbf{w}_{2i-1} \in D_0 \times 0^{n/2}\}$
      $L_{2i} = \{\mathbf{w}_{2i} \mid \mathbf{w}_{2i} \in 0^{n/2} \times D_0\}, i = 1, \ldots, 8$
3 Choose random $c_{\mathbf{z}_1} \in \mathbb{F}_2^{\ell}, c_{\mathbf{y}_1}, c_{\mathbf{y}_3} \in \mathbb{F}_2^{\ell_1+\ell_2}, c_{\mathbf{x}_1}, c_{\mathbf{x}_3}, c_{\mathbf{x}_5}, c_{\mathbf{x}_7} \in \mathbb{F}_2^{\ell_1}$
4 Set remaining constraints according to Equation (2)
5 Compute (and filter)
      $L_i^{(1)} = \{\mathbf{x}_i \mid \langle \mathbf{a}, \mathbf{x}_i \rangle = c_{\mathbf{x}_i} \bmod 2^{\ell_1}, \mathbf{x}_i = \mathbf{w}_{2i-1} + \mathbf{w}_{2i}\}$
      from $L_{2i-1}, L_{2i},\ i = 1, \ldots, 8$    , then filter such that $L_i^{(1)} \subseteq D_1$
      $L_i^{(2)} = \{\mathbf{y}_i \mid \langle \mathbf{a}, \mathbf{y}_i \rangle = c_{\mathbf{y}_i} \bmod 2^{\ell_1+\ell_2}, \mathbf{y}_i = \mathbf{x}_{2i-1} + \mathbf{x}_{2i}\}$,
      from $L_{2i-1}^{(1)}, L_{2i}^{(1)},\ i = 1, \ldots, 4$    , then filter such that $L_i^{(2)} \subseteq D_2$
      $L_i^{(3)} = \{\mathbf{z}_i \mid \langle \mathbf{a}, \mathbf{z}_i \rangle = c_{\mathbf{z}_i} \bmod 2^{\ell}, \mathbf{z}_i = \mathbf{y}_{2i-1} + \mathbf{y}_{2i}\}$,
      from $L_{2i-1}^{(2)}, L_{2i}^{(2)},\ i = 1, 2$    , then filter such that $L_i^{(3)} \subseteq D_3$
      $L = \{\mathbf{e} \mid \langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n, \mathbf{e} = \mathbf{z}_{2i-1} + \mathbf{z}_{2i}\}$
      from $L_1^{(3)}, L_2^{(3)}$    , then filter such that $L \subseteq \{0, 1\}^n$
  return $\mathbf{e} \in L$
```

The construction of each unfiltered list can be performed via hashing in time linear in the list's sizes giving an expected time complexity of

$$T = \max_i(\mathcal{L}_i) \ .$$

Since we need to store only filtered lists and the filtering can be performed on-the-fly the memory complexity becomes $M = \max_i(q_i \cdot \mathcal{L}_i)$.

Correctness Obviously the constraint's sizes ℓ_1, ℓ_2 and ℓ_3 cannot be chosen arbitrarily large if one representation of the solution should survive all imposed constraints. On the other hand we need to ensure that multiple representations do not lead to the construction of duplicate elements in intermediate lists to ensure a proper list distribution. This leads to further restrictions on the size of ℓ_1, ℓ_2 and ℓ_3, called *saturation constraints* in [12] or simply lower bounds in [26].

In [12] this is formalized by ensuring that each list after filtering at every level is not larger than the size of the set filtered for, reduced by the total enforced constraint. Since by the randomness of the instance the elements distribute uniformly, it follows that the lists will not contain duplicate elements with high probability. The sets for which we filter on level i are D_i, $i = 1, 2, 3, 4$. Note that the choice of the sets D_i, $i \neq 4$ can be optimized, while the set D_4 has to describe the valid set of solutions, which is the set of binary vectors of length n.

Hence, to guarantee that there are no duplicates present in the level-1, level-2 and level-3 lists we need to ensure that

$$q_1 \cdot \mathcal{L}_1 \leq \frac{|D_1|}{2^{\ell_1}} \text{ and } q_2 \cdot \mathcal{L}_2 \leq \frac{|D_2|}{2^{\ell_1+\ell_2}} \text{ and } q_3 \cdot \mathcal{L}_3 \leq \frac{|D_3|}{2^{\ell}} \tag{3}$$

Next let us write the probabilities q_i in terms of the D_i and the corresponding representations. Therefore, let 2^{r_i} denote the amount of different representations of any element from D_{i+1} as the sum of two elements from D_i. Then we have

$$q_{i+1} = \frac{|D_{i+1}| \cdot 2^{r_i}}{|D_i|^2}, \tag{4}$$

describing the probability that a random sum of two elements from D_i forms a representation of any element from D_{i+1}.

Recall that we construct level-1 elements $\mathbf{x}_i = (\mathbf{w}_{2i-1}, \mathbf{w}_{2i}) \in D_0 \times D_0 = D_1$ in a meet-in-the-middle fashion from level-0 elements $\mathbf{w}_j$, which implies $\mathcal{L}_0 = \sqrt{|D_1|}$. As this gives only a single representation of any level-1 element, we have $r_0 = 0$, which leads to $q_1 = 1$, i.e., for this choice of D_0 there is no filtering on level one. It follows that the first saturation constraint from Equation (3) is always fulfilled since

$$q_1 \cdot \mathcal{L}_1 = \frac{(\mathcal{L}_0)^2}{2^{\ell_1}} = \frac{|D_1|}{2^{\ell_1}}.$$

The second constraint of Equation (3) gives

$$q_2 \cdot \mathcal{L}_2 = \frac{|D_2| \cdot 2^{r_1}}{|D_1|^2} \cdot \frac{(\mathcal{L}_0)^4}{2^{2\ell_1+\ell_2}} \overset{!}{\leq} \frac{|D_2|}{2^{\ell_1+\ell_2}} \Leftrightarrow r_1 \leq \ell_1.$$

Analogously we get from the last saturation constraint

$$q_3 \cdot \mathcal{L}_3 = q_3 \cdot \frac{(q_2)^2 \cdot (\mathcal{L}_1)^4}{2^{4\ell_1+2\ell_2+\ell_3}} = \frac{2^{2r_1+r_2} \cdot |D_3|}{2^{4\ell_1+2\ell_2+\ell_3}} \overset{!}{\leq} \frac{|D_3|}{2^{\ell}} \Leftrightarrow 2r_1 + r_2 \leq 3\ell_1 + \ell_2.$$

Eventually, to find exactly one representation of the solution in the final list we need to ensure that $q_4 \cdot \mathcal{L}_4 = 1$, which yields

$$\begin{aligned} q_4 \cdot \mathcal{L}_4 &= \frac{q_4 \cdot (q_3)^2 \cdot (q_2)^4 (\mathcal{L}_1)^8}{2^{n+7\ell_1+3\ell_2+\ell_3}} = \frac{2^{4r_1+2r_2+r_3} \cdot |D_4|}{2^{n+7\ell_1+3\ell_2+\ell_3}} \overset{!}{=} 1 \\ \Leftrightarrow \quad & 4r_1 + 2r_2 + r_3 = 7\ell_1 + 3\ell_2 + \ell_3, \end{aligned} \tag{5}$$

since we have $|D_4| = 2^n$, as D_4 is the set of binary vectors of length n.

Instantiation The description of the general BCJ algorithm gives several degrees of freedom, including the choice of sets D_i, $i = 1, 2, 3$ and the size of the constraints ℓ_1, ℓ_2, ℓ_3. The original BCJ algorithm restricts all D_i's to include only vectors with coordinates in $\{0, \pm 1\}$. The purpose of including -1's is simply to increase the number of representations. Since the final goal is to construct a binary vector, minus one entries are supposed to cancel out with one entries in the addition. Thus, the distribution D_3 is chosen as vectors of length n with exactly $\omega_3 := n/4 + \alpha_3$ one entries and $m_3 := \alpha_3$ minus one entries for some small α_3, which has to be optimized. The distribution D_2 is then composed similarly as vectors of length n with $\omega_2 := \omega_3/2 + \alpha_2$ one entries and $m_2 := m_3/2 + \alpha_2$ minus one entries, where again α_2 minus ones are supposed to cancel out. Analogously the level-1 distribution is chosen as vectors of length n with $\omega_1 := \omega_2/2 + \alpha_1$ one entries and $m_1 := m_2/2 + \alpha_1$ minus one entries, expecting α_1 cancellations. An overview of this choice of distributions is given in Table 1. The size of these sets is

$$|D_i| = \binom{n}{\omega_i, m_i, \cdot} \simeq 2^{g(\frac{\omega_i}{n}, \frac{m_i}{n})n},$$

while the number of representations is given as

$$2^{r_{i-1}} = \binom{\omega_i}{\omega_i/2}\binom{m_i}{m_i/2}\binom{n - \omega_i - m_i}{\alpha_{i-1}, \alpha_{i-1}, \cdot} \simeq 2^{\omega_i + m_i + \rho_i},$$

where $\rho_i := g\left(\frac{\alpha_i}{n-\omega_i-m_i}, \frac{\alpha_i}{n-\omega_i-m_i}\right)(n - \omega_i - m_i)$.

		D_4	D_3	D_2	D_1
BCJ	ω_i	$\frac{1}{2}$	$\frac{1}{4} + \alpha_3$	$\frac{1}{8} + \frac{\alpha_3}{2} + \alpha_2$	$\frac{1}{16} + \frac{\alpha_3}{4} + \frac{\alpha_2}{2} + \alpha_1$
	m_i	0	α_3	$\frac{\alpha_3}{2} + \alpha_2$	$\frac{\alpha_3}{4} + \frac{\alpha_2}{2} + \alpha_1$
BBSS	ω_i	$\frac{1}{2}$	$\frac{1}{4} + \alpha_3 - \gamma_3$	$\frac{1}{8} + \frac{\alpha_3 - \gamma_3}{2} + \alpha_2 - \gamma_2$	$\frac{1}{16} + \frac{\alpha_3 - \gamma_3}{4} + \frac{\alpha_2 - \gamma_2}{2} + \alpha_1 - \gamma_1$
	m_i	0	α_3	$\frac{\alpha_3}{2} + \alpha_2$	$\frac{\alpha_3}{4} + \frac{\alpha_2}{2} + \alpha_1$
	c_i	0	γ_3	$\frac{\gamma_3}{2} + \gamma_2$	$\frac{\gamma_3}{4} + \frac{\gamma_2}{2} + \gamma_1$

Table 1: Choices of D_i made by BCJ and BBSS algorithm. The table states the proportion of coordinates equal to 1 (ω_i), -1 (m_i) and 2 (c_i). The proportion of zeros is $1 - \omega_i - m_i - c_i$. Set D_0 has half the proportions of D_1.

Here the two binomial coefficients count the number of possibilities how to distribute the one and minus one entries of an element from D_i equally over a sum of two elements. The multinomial coefficient then counts the number of possibilities how the remaining minus one and one entries can cancel out.

Note that the algorithm splits D_1 into $D_0 \times D_0$, where D_0 is the set of vectors of length $n/2$ containing exactly $\omega_1/2$ ones and $m_1/2$ minus ones. This leads to all D_i only including balanced elements, i.e., elements which contain an equal amount of ones (resp. minus ones) on their first and second half of the coordinates. However, this affects the sizes of the D_i and the amount of representations only by a polynomial factor, which is subsumed in the Landau notation.

Eventually, the BCJ algorithm chooses $\ell_1 = r_1$ and $\ell_2 = r_2 - r_1$, which yields $\ell_3 = r_3 - r_2$. A numerical optimization of the α_i results in a time complexity of $2^{0.291n}$ for the BCJ configuration.

Bonnetain et al. [12] then showed that a more flexible choice of ℓ_1 and ℓ_2 and correspondingly adapted ℓ_3 allows to decrease the time complexity to $2^{0.289n}$. They also showed that extending the digit set of the D_i to $\{0, \pm 1, 2\}$ allows to further decrease the time complexity to $2^{0.283n}$, yielding the best known time complexity for the random subset sum problem.

The BCJ algorithm achieves optimal time complexity for a depth of the search-tree of four. However, in general the optimal depth varies with the application. We therefore give for completeness and later reference the complexity and saturation constraints for variable depth in Appendix A.

4 New Subset Sum Trade-Off

The (generalized) BCJ algorithm from the previous section already inherits some time-memory trade-off potential. That is, one can try to optimize the choice of the ℓ_i with respect to the memory usage, since the larger the ℓ_i the smaller the list's sizes. However, the overall size of the ℓ_i's is bounded by the restriction that the last list should contain a representation of the solution.

On a high level our new trade-off works by relaxing this restriction, i.e. we do not require the last list to contain a solution. This allows to balance the lists more memory-friendly. We then perform multiple randomized executions of the algorithm to ensure that we find a solution overall. However, let alone this is not sufficient to obtain our improvements. The main runtime advantage of our improved trade-off comes from our observation that we can reuse parts of the tree in subsequent randomized executions, reducing the cost per iteration. A second improvement stems from our use of the Dissection framework [19] for the construction of the level-1 lists.

Note that if we change some bit-constraints in the tree (the values of $c_{\mathbf{v}}$ in Figure 2) not necessarily all levels are affected. That means we do not need to re-compute all lists of the tree, but only those which depend on the changed constraints. Now, if the computation of each list had the same complexity, this strategy would only yield a constant factor improvement since at least one list needs to be recomputed. However, by adapting parameters accordingly and exploiting the involved filtering, we can guarantee that the creation of frequently reconstructed lists (from already existing lists) is much cheaper than a reconstruction of the whole tree. This partial reconstruction strategy in combination with relaxing the correctness constraint from Equation (5) allows us to obtain significant improvements for rather high memory parameters $M \geq 2^{0.169n}$.

From there on the base lists, which are so far a meet-in-the-middle split of the first level domains start dominating the memory. The only possibility for the algorithm to decrease the size of those lists is to choose a set D_1 with smaller size on level 1. For the BCJ algorithm this means including less -1 entries, until ultimately no -1 entries are included in the enumeration. In this case the base lists require a memory of $\binom{n/2}{n/32} \simeq 2^{0.169n}$. From there the list sizes are as small as possible and we can not obtain instantiations for less memory. We circumvent this problem by exchanging the meet-in-the-middle strategy for exhaustive examination of the level-1 domain by the 7-Dissection algorithm. We find that apart from offering instantiations for memory parameters $M < 2^{0.169n}$, this gives also (slight) time improvements in the high memory regime $M \geq 2^{0.169n}$ as the optimization can choose a more optimal, usually larger set D_1 (implying larger D_0) without exceeding the memory limit.

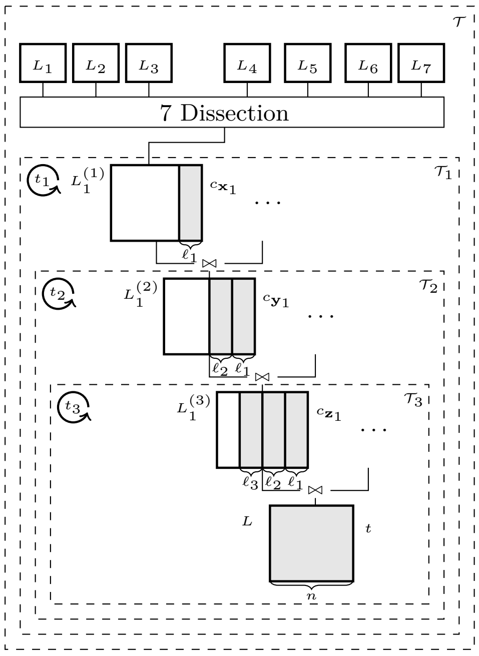

Fig. 3: Our new trade-off in depth 4. Dashed boxes frame different subtrees $\mathcal{T}_i$, which are rebuild 2^{t_i} times. The level-1 lists are constructed using the 7-Dissection algorithm.

Note that the 7-Dissection in our setting requires a memory of at least $\binom{n}{n/16}^{1/7} \simeq 2^{0.049n}$. To obtain instantiations for every $M > 0$ we could exchange the 7-Dissection by a c-Dissection for $c > 7$. However, since for a memory of $M \leq 2^{0.091n}$ a trade-off based on an algorithm by Esser-May offers a better time complexity anyway, we stick with a 7-Dissection for simplicity.

Adaptation of the BCJ Algorithm We advise the reader to follow Figure 3. Let $\mathcal{T}$ be the full tree and $\mathcal{T}_i, i = 1, 2, 3$ the subtrees only including the lists from level i onwards. We denote by 2^{t_i} the number of times we rebuild the subtree $\mathcal{T}_i$ from the (already existing) lists of the previous level.

We start by changing only the upper ℓ_3 bits of the modular constraint $c_{\mathbf{z}_1}$ which requires recomputing only the subtree $\mathcal{T}_3$, since the level-i lists for $i \leq 2$ do not depend on these bits. Since there are only 2^{ℓ_3} choices for those bits we have $t_3 \leq \ell_3$. If 2^{ℓ_3} iterations are not sufficient to find the solution we start modifying the upper ℓ_2 bits of the modular constraints $c_{\mathbf{y}_1}$, $c_{\mathbf{y}_2}, c_{\mathbf{z}_1} \bmod 2^{\ell_2}$. This implies again that $t_2 \leq 3\ell_2$. Still, for every different choice of those bits we recompute the subtree $\mathcal{T}_3$ another 2^{t_3} times for different choices of the upper ℓ_3 bits. If $2^{3\ell_2+\ell_3}$ iterations are still not sufficient to find a solution, we eventually start modifying the lower ℓ_1 bits of the chosen modular constraints. Again for each choice of lower bits we reconstruct the tree $\mathcal{T}_2$ and $\mathcal{T}_3$ several times. Furthermore, as there are seven constraints that can be freely chosen we have $t_1 \leq 7\ell_1$

Finally, instead of computing the level-1 lists via a meet-in-the-middle algorithm we now use the 7-Dissection algorithm.

The pseudocode of our modified BCJ trade-off is given in Algorithm 2.

Complexity. The memory complexity stays as before with the only difference that the memory requirement of the base lists is now substituted by the memory requirement M_{7D} of the 7-Dissection algorithm, i.e.,

$$M = \max(M_{7D}, q_1\mathcal{L}_1, q_2\mathcal{L}_2, q_3\mathcal{L}_3, q_4\mathcal{L}_4).$$

To balance the memory requirement we instantiate the 7-Dissection algorithm with $M_{7D} = |D_1|^{\max\left(\frac{1}{7},\lambda'\right)}$ memory where $|D_1|^{\lambda'} = \max_i(q_i L_i)$.

The analysis of the time complexity also follows along the lines of the previous analysis, with the essential difference that the three subtrees are now computed differently many times.

A single construction of subtree $\mathcal{T}_1$ can be performed in time

$$T_1 = \max(T_{7D}, \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4),$$

where T_{7D} is the time it takes to compute the level-1 lists via the 7-Dissection algorithm. Recall that instantiated with $|D_1|^{\delta}$ memory, the 7-Dissection runs in time $T_{7D} = |D_1|^{\max\left(\frac{2(1-\delta)}{3},\frac{1}{2}\right)}$ (compare to Lemma 2.1). The subtrees $\mathcal{T}_2$ and $\mathcal{T}_3$ can then be computed in time

$$T_2 = \max(q_1 \cdot \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4) \text{ and } T_3 = \max(q_2 \cdot \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4),$$

as they can be computed from the stored and already filtered level-1 respectively level-2 lists. Now the total time complexity becomes

$$T = \max(2^{t_1} \cdot T_1, 2^{t_1+t_2} \cdot T_2, 2^{t_1+t_2+t_3} \cdot T_3),$$

as subtree $\mathcal{T}_i$ is rebuild $2^{t_1+\dots+t_i}$ many times.

Correctness. Most of the correctness follows from the correctness of the BCJ algorithm and the 7-dissection algorithm. Note that we instantiate the 7-Dissection

Algorithm 2: BCJ Trade-Off

```
    Input   : $\mathbf{a} \in (\mathbb{Z}_{2^n})^n, t \in \mathbb{Z}_{2^n}$
    Output : $\mathbf{e} \in \{0,1\}^n$ with $\langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n$
1   Choose optimal $\ell_1, \ell_2, \ell_3$ and $D_i, i = 1,2,3$, let $r := r_3 + 2r_2 + 4r_1$
2   repeat $2^{t_1} := 2^{\max(7\ell_1 - r, 0)}$ times
3       Choose random $c_{\mathbf{z}_1} \in \mathbb{F}_2^{\ell}, c_{\mathbf{y}_1}, c_{\mathbf{y}_3} \in \mathbb{F}_2^{\ell_1+\ell_2}, c_{\mathbf{x}_1}, c_{\mathbf{x}_3}, c_{\mathbf{x}_5}, c_{\mathbf{x}_7} \in \mathbb{F}_2^{\ell_1}$
4       Set remaining constraints according to Equation (2)
5       Compute
            $L_i^{(1)} = \{\mathbf{x}_i \mid \langle \mathbf{a}, \mathbf{x}_i \rangle = c_{\mathbf{x}_i} \bmod 2^{\ell_1}, \mathbf{x}_i \in D_1, \},$
                via 7-Dissection, $\quad i = 1, \dots, 8$
6       repeat $2^{t_2} := 2^{\max(7\ell_1 + 3\ell_2 - r, 0) - t_1}$ times
7           Choose randomly the upper $\ell_2$ bits of $c_{\mathbf{z}_1}, c_{\mathbf{y}_1}, c_{\mathbf{y}_3} \bmod 2^{\ell_1+\ell_2}$
8           Update $c_{\mathbf{z}_2}, c_{\mathbf{y}_2}, c_{\mathbf{y}_4}$ according to Equation (2)
9           Compute
                $L_i^{(2)} = \{\mathbf{y}_i \mid \langle \mathbf{a}, \mathbf{y}_i \rangle = c_{\mathbf{y}_i} \bmod 2^{\ell_1+\ell_2}, \mathbf{y}_i \in D_2, \mathbf{y}_i = \mathbf{x}_{2i-1} + \mathbf{x}_{2i}\},$
                    from $L_{2i-1}^{(1)}, L_{2i}^{(1)}, \quad i = 1, \dots, 4$
10          repeat $2^{t_3} := 2^{\max(7\ell_1 + 3\ell_2 + \ell_3 - r, 0) - t_1 - t_2}$ times
11              Choose randomly the upper $\ell_3$ bits of $c_{\mathbf{z}_1}$
12              Update $c_{\mathbf{z}_2}$ according to Equation (2)
13              Compute
                    $L_i^{(3)} = \{\mathbf{z}_i \mid \langle \mathbf{a}, \mathbf{z}_i \rangle = c_{\mathbf{z}_i} \bmod 2^{\ell}, \mathbf{z}_i \in D_3, \mathbf{z}_i = \mathbf{y}_{2i-1} + \mathbf{y}_{2i}\},$
                        from $L_{2i-1}^{(2)}, L_{2i}^{(2)}, \quad i = 1, 2$
                    $L = \{\mathbf{e} \mid \langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n, \mathbf{e} \in D_4, \mathbf{e} = \mathbf{z}_{2i-1} + \mathbf{z}_{2i}\},$
                        from $L_1^{(3)}, L_2^{(3)}$
                if $|L| > 0$ then
14                  return $\mathbf{e} \in L$
```

with at least $|D_1|^{\frac{1}{7}}$ memory, which is the minimum requirement given by Lemma 2.1.

The main difference to before is that we relaxed the restriction given in Equation (5), such that the last list is not guaranteed to contain a solution anymore. However, we compensate for this by multiple randomized constructions of the final list. In contrast to completely independent executions of the algorithm, which would select all constraints uniformly at random, we only randomize the constraints affecting certain subtrees. However, note that under the standard assumption that the representations distribute independently and uniformly over all constraints, any set of constraints has the same independent probability of leading to a representation of the solution. Now, since we change at least one constraint for every reconstruction of the final list, we can treat the iterations as independent.

In order to ensure that over all iterations we find at least one representation, the final list's size accumulated over all its reconstructions must be at least one, which leads to (compare to Equation (5))

$$\begin{aligned} & q_4 \cdot \mathcal{L}_4 \cdot 2^{t_1+t_2+t_3} && \geq 1 \\ \Leftrightarrow \quad & 4r_1 + 2r_2 + r_3 + t_1 + t_2 + t_3 && \geq 7\ell_1 + 3\ell_2 + \ell_3 . \end{aligned}$$

Note that this constraint is fulfilled for our choice of

$$\begin{aligned} t_1 &= \max(7\ell_1 - r, 0) \\ t_2 &= \max(7\ell_1 + 3\ell_2 - r, 0) - t_1 \\ t_3 &= \max(7\ell_1 + 3\ell_2 + \ell_3 - r, 0) - t_1 - t_2 , \end{aligned}$$

where $r := r_3 + 2r_2 + 4r_1$ and the maximum is needed since we need to build each subtree at least once.

Configuration of our Trade-Off In terms of distributions we adopt the choice of the original BCJ algorithm, specified in Table 1. We then optimize the parameters $\alpha_i, \ell_i, i = 1, 2, 3$ numerically. We optimize such that the time is minimized, while simultaneously ensuring that the saturation constraints are satisfied and a given memory limit of $M = 2^{\lambda n}$ is not exceeded.

The resulting trade-off curve is depicted in Figure 1. We observe that our trade-off outperforms all existing approaches for $M \geq 2^{0.093n}$. Prior to our work, this interval was covered by a diverse landscape of different trade-offs including [7, 16, 19, 22, 27, 30]. For $M < 2^{0.093n}$ a trade-off given in [22] based on a memory-free algorithm by Esser-May [27] becomes superior to our procedure.

Extending the digit set. We also adopted the choice of distributions made by the BBSS algorithm [12] (see Table 1). We find that the refined choice of the D_i gives an overall slight improvement, interpolating smoothly to their $2^{0.283n}$ algorithm. The resulting trade-off curve is depicted in Figure 1 as well, which remains superior to [22, 27] as long as $M \geq 2^{0.091n}$.

Increasing the tree-depth. We also performed a numerical optimization of our trade-off with increased tree-depth of five. However, we were not able to obtain better instantiations, i.e., instantiations with lower time complexity for a given memory.

Linear approximations. Observe that both our trade-offs split in three almost linear segments (compare to Figure 1). To ease the comparison of further results to our trade-offs, we provide a linear approximation $T = -a \cdot M + b$ of these segments in Table 2. This allows to easily compare to the (approximate) running time of our trade-off, without rerunning the optimization of parameters.

$\leq M$	0.18		0.27		0.28	
a	2.54	2.65	1.06	1.13	0.27	0.36
b	0.84	0.85	0.58	0.58	0.37	0.39

Table 2: Slope a and y-intercept b of the linear approximations $T = -a \cdot M + b$ of the three different segments, each denoted by the maximal memory available in the segment. Left columns refer to the trade-off using BCJ-like representations, while right columns use BBSS-syle representations.

Translation of Provable Variant from Previous Works In the original works of Howgrave-Graham-Joux [30] and Becker-Coron-Joux [7] the constraints on each level i are modelled as $c_{\mathbf{v}} \mod \prod_{j=1}^{i} M_i$ for M_i a prime close to 2^{ℓ_i}. This allows to apply a result for the distribution of random modular sums from [35]. In turn the authors are able to analyze the distribution of the list sizes throughout the algorithm as well as the probability that a representation of the solution survives a certain set of constraints. The algorithm in its provable variant is then adapted to repeat the construction of each level for $N \cdot 2^{\varepsilon n}$ random set of constraints, for an arbitrarily small constant $\varepsilon > 0$, and to abort constructions that exceed the expected memory complexity. It is then shown that this variant succeeds with a maximum overhead factor of $N^3 \cdot 2^{3\varepsilon n}$ in time using its expected memory complexity with probability $1 - c^{-N}$ for some constant c.

In spirit this re-randomization technique translates to our new trade-offs. Therefore first note, that we choose $M_i = 2^{\ell_i}$ as modulus just for ease of exposition, instead we could analogous to [7, 30] choose primes close to 2^{ℓ_i}. Now, whenever we have to exchange a set of constraints on any level, i.e., in lines 3,7 and 11 of Algorithm 2, we have to repeat this exchange $N \cdot 2^{\varepsilon n}$ times to make sure the algorithm is successful and not aborted for at least one of those repetitions. Clearly, the running time is at most increased by a factor of $N^3 \cdot 2^{3\varepsilon n}$ while the success probability becomes $(1 - c^{-N})^R$, where $R = 2^{t_1+t_2+t_3}$ is the maximum number of times a constraint is exchanged. Note that this probability for large enough $N = \mathrm{poly}(n)$ is still overwhelming.

To ensure that there are enough "unused" constraints available for the re-randomization approach, we have to restrict the values of t_1, t_2, t_3 to a maximum of $7\ell_1 - \delta$, $3\ell_2 - \delta$ and $\ell_3 - \delta$, respectively, where $\delta := \varepsilon n + \log N$. Technically, this leads to a slightly worse trade-off, but since ε is an arbitrarily small constant this performance gap becomes arbitrarily small.

5 Application to Decoding Binary Linear Codes

A linear code $\mathcal{C} \subset \mathbb{F}_2^n$ is a k-dimensional subspace of $\mathbb{F}_2^n$ and can efficiently be described via a parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, such that $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_2^n \mid \mathbf{Hc} = \mathbf{0}\}$. Decoding an error-prone codeword $\mathbf{y} := \mathbf{c} + \mathbf{e}$ to $\mathbf{c}$ is polynomial-time equivalent to recovering $\mathbf{e}$ from the so-called *syndrome* $\mathbf{s} := \mathbf{Hy} = \mathbf{H}(\mathbf{c}+\mathbf{e}) = \mathbf{He}$. This leads to the following definition of the syndrome decoding problem.

Definition 5.1 (Syndrome Decoding Problem). *Let* $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ *be the parity-check matrix of a code of length* n *and dimension* k*, with constant* code-rate $R := \frac{k}{n}$*. Given a syndrome* $\mathbf{s} \in \mathbb{F}_2^{n-k}$ *and an integer* ω *the* syndrome decoding problem *asks to find a vector* $\mathbf{e} \in \mathbb{F}_2^n$ *of Hamming weight* $\mathrm{wt}(\mathbf{e}) = \omega$ *satisfying* $H\mathbf{e} = \mathbf{s}$.

Note that the problem admits a unique solution as long as $\omega \leq \lfloor \frac{d-1}{2} \rfloor$, where d is the minimum distance of the code, i.e., the minimum Hamming distance between two codewords. We call the setting with unique solution *half distance decoding*, while for $\omega \leq d$ we refer to *full distance decoding.* In those regimes, the time complexity generally increases with ω, such that we only consider the cases where ω is equal to those upper bounds. Further, random linear codes are known to achieve a minimum distance that is equal to the Gilbert-Varshamov bound of $d \approx H^{-1}(1 - \frac{k}{n})n$, i.e., the minimum distance is a function of the rate $R := \frac{k}{n}$ and the code-length n. In our asymptotic analysis we maximize the complexity over all constant rates R to obtain a runtime formula which only depends on n.

The best known algorithms to solve the syndrome decoding problem are Information Set Decoding (ISD) algorithms. In the full and half distance setting these algorithms have exponential time and memory complexity of the form $\tilde{\mathcal{O}}\left(2^{cn}\right)$ for some constant c depending on the algorithm. On the other hand, cryptographic applications usually use a a sublinear weight, i.e., $\omega = o(n)$. In these cases the running time of ISD algorithms is subexponential of the form $\tilde{\mathcal{O}}\left(2^{c\omega}\right)$ for some constant c. Moreover, it was shown [40] that in this case all known ISD algorithms converge to the same running time, i.e., they obtain the same constant c. However, in practical experiments advanced ISD algorithms were shown to provide significant speedups [10, 28].

We therefore first analyse our trade-offs in the full and half distance decoding setting, which allow to easily verify their superiority since they obtain improved constants c. We then study the practical effect of our trade-offs by providing an optimized implementation. Finally, we extrapolate the hardness of cryptographic schemes using our obtained data points.

Information Set Decoding Information Set Decoding algorithms first apply a permutation matrix $\mathbf{P}$ to the columns of the parity-check matrix. This allows to redistribute the weight of the error since the permuted instance $\mathbf{H}' := \mathbf{HP}$ has as valid solution $\mathbf{e}' := \mathbf{P}^{-1}\mathbf{e}$, since $\mathbf{HP}(\mathbf{P}^{-1}\mathbf{e}) = \mathbf{s}$. Then $\mathbf{H}'$ is transformed into semi-systematic form via Gaussian elimination modelled via the multiplication with an invertible matrix $\mathbf{Q}$

$$\mathbf{QH}'(\mathbf{P}^{-1}\mathbf{e}) = \begin{pmatrix} \mathbf{I}_{n-k-\ell} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix} (\mathbf{e}_1, \mathbf{e}_2) = (\mathbf{e}_1 + \mathbf{H}_1\mathbf{e}_2, \mathbf{H}_2\mathbf{e}_2) = \mathbf{Qs} = (\mathbf{s}_1, \mathbf{s}_2), \quad (6)$$

where we write $\mathbf{e}' := \mathbf{P}^{-1}\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{k+\ell}$ with ℓ an optimization parameter of the algorithm. Let us further assume that the permutation distributes the weight on $\mathbf{e}'$ such that $\mathrm{wt}(\mathbf{e}_1) = \omega - p$ and $\mathrm{wt}(\mathbf{e}_2) = p$, for some p that has to be optimized, too.

Now Equation (6) yields a (dimension) reduced syndrome decoding instance in form of the equation $\mathbf{H}_2\mathbf{e}_2 = \mathbf{s}_2$ with weight-p solution $\mathbf{e}_2 \in \mathbb{F}_2^{k+\ell}$. Usually, $\mathbf{e}_2$ is not a unique solution to this reduced instance. The algorithm therefore computes all solutions $\mathbf{x}$ to this smaller instance and checks if the corresponding $\mathbf{e}_1 = \mathbf{s}_1 + \mathbf{H}_1\mathbf{x}$ has weight $\omega - p$. In this case $\mathbf{P}(\mathbf{e}_1, \mathbf{x})$ forms a solution to the original syndrome decoding instance. If no solution is found, the algorithm is repeated for another random permutation.

Complexity. Let us briefly argue about the complexity of such a procedure. The probability of distributing the weight on $\mathbf{e}'$ as desired is

$$q := \frac{\binom{n-k-\ell}{\omega-p}\binom{k+\ell}{p}}{\binom{n}{\omega}}. \tag{7}$$

Hence, we expect that after q^{-1} random permutations one of them distributes the weight as desired. If now the cost to retrieve all weight-p solutions to the reduced instance for any of those permutations is T_{S}, the total complexity becomes

$$T = \tilde{\mathcal{O}}\left(q^{-1} \cdot T_{\mathrm{S}}\right).$$

In a nutshell different ISD algorithms differentiate in how they retrieve the solutions to the reduced instance. Usually they consider the reduced instance as a vectorial subset sum instance, where the solution encodes a size-p subset of the columns of $\mathbf{H}_2$ that sums to $\mathbf{s}_2$. Then they make use of advanced algorithms for subset sum, such as the BCJ algorithm, to retrieve the solutions to that instance. It is not hard to see, that instead of working over $\mathbb{Z}_{2^n}$, the generalized BCJ algorithm outlined in Section 3 and, hence, also our improved trade-off from Section 4, work analogously over $\mathbb{F}_2^n$.

5.1 Improved ISD Trade-Offs

The May-Meurer-Thomae (MMT) ISD algorithm [33] originally uses the BCJ construction in depth-2 to retrieve the solutions to the reduced instance. In the following we give an improved version of the MMT algorithm based on our new subset sum trade-off from Section 4. Our version improves the overall memory complexity and yields a better trade-off curve, i.e., we achieve runtime improvements for every fixed memory.

To make use of our generalized trade-off description (in depth 2) we need to define appropriate sets D_0, D_1 and D_2. Then, to retrieve the running time we calculate the amount of existing representations and optimize the parameter ℓ_1. The pseudocode of our improved MMT algorithm is given in Algorithm 3.

Note that in our ISD application we find that already using the Schroeppel-Shamir technique for level-1 list construction, rather than the 7-Dissection, offers optimal instantiations for all memory parameters $M > 0$. We therefore stick with the Schroeppel-Shamir technique in our description for simplicity.

Algorithm 3: New MMT Trade-Off

```
    Input   : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_2^{n-k}, w \in \mathbb{N}$
    Output : $\mathbf{e} \in \mathbb{F}_2^n, \mathbf{He} = \mathbf{s}$
 1  Choose optimal $\ell, \ell_1, p$
 2  let $r_1 = \log \binom{p}{p/2}$
 3  $\pi_{\ell_1} : \mathbb{F}_2^\ell \to \mathbb{F}_2^{\ell_1}$, $\pi_{\ell_1}(x_1, \ldots, x_\ell) = (x_1, \ldots, x_{\ell_1})$
 4  repeat
 5      choose random permutation matrix $\mathbf{P}$
 6      $\bar{\mathbf{H}} = \begin{pmatrix} \mathbf{I}_{n-k-\ell} & \mathbf{H}_1 \\ 0 & \mathbf{H}_2 \end{pmatrix} = \mathbf{QHP}$, $\mathbf{Qs} = (\mathbf{s}_1, \mathbf{s}_2)$
 7      repeat $2^{\ell_1 - r_1}$ times
 8          Choose random $\mathbf{t} \in \mathbb{F}_2^{\ell_1}$
 9          Compute
              $L_1^{(1)} = \{\mathbf{z}_1 \mid \pi_{\ell_1}(\mathbf{H}_2\mathbf{z}_1) = \mathbf{t}, \mathbf{z}_1 \in D_1\}$          , via Schroeppel-Shamir
              $L_2^{(1)} = \{\mathbf{z}_2 \mid \pi_{\ell_1}(\mathbf{H}_2\mathbf{z}_2 + \mathbf{s}_2) = \mathbf{t}, \mathbf{z}_2 \in D_1\}$   , via Schroeppel-Shamir
10          Compute $L = \{\mathbf{e}_2 \mid \mathbf{H}_2\mathbf{e}_2 = \mathbf{s}_2, \mathbf{e}_2 = \mathbf{z}_1 + \mathbf{z}_2\}$ from $L_1^{(1)}, L_2^{(2)}$
11          for $\mathbf{e}_2 \in L$ do
12              $\mathbf{e}_1 = \mathbf{H}_1\mathbf{e}_2 + \mathbf{s}_1$
13              if $\mathrm{wt}(\mathbf{e}_1) \le \omega - p$ then
14                  return $P(\mathbf{e}_1, \mathbf{e}_2)$
```

Complexity. We let D_2 be the set of vectors from $\mathbb{F}_2^{k+\ell}$ with weight p, as it defines our solution set. The MMT algorithm now chooses D_1 as vectors from $\mathbb{F}_2^{k+\ell}$ with weight $p/2$. Finally D_0 is the set of vectors from $\mathbb{F}_2^{\frac{k+\ell}{2}}$ and weight $p/4$, i.e., a meet-in-the-middle split of D_1, hence $|D_0| = \sqrt{|D_1|}$.The size of D_1 is

$$|D_1| = \binom{k+\ell}{p/2},$$

while the amount of representations of one element from D_2 as sum of two elements from D_1 is

$$2^{r_1} = \binom{p}{p/2} \simeq 2^p.$$

Observe that the binomial coefficient counts the possibilities to distribute half of the ones of the target vector over the first addend, while the other half must then be covered by the second addend. Now, to find one representation of each solution to the reduced instance in the final list we need to ensure (compare to Equation (8))

$$\ell_1 \overset{!}{=} r_1.$$

Our trade-off from Section 4 now allows for $\ell_1 > r_1$ and compensates by repeating the procedure. Note that in depth-2 we have no further saturation constraints, nor can we make use of reconstructing different levels differently many times. The time complexity for finding all solutions to the vectorial subset sum problem then becomes

$$T_{\mathrm{S}} = 2^{\ell_1 - r_1} \cdot \max(\sqrt{|D_1|}, |D_1|/2^{\ell_1}, |D_1|^2/2^{\ell+\ell_1})$$

The memory complexity is equal to the level-0 and level-1 lists, since elements of the final list can be checked on the fly for being a solution. Moreover, by using the Schroeppel-Shamir algorithm for the construction of the level-1 lists we can reduce the memory required for storing the level-0 lists from $|D_0| = \sqrt{|D_1|}$ to $\sqrt{|D_0|} = |D_1|^{1/4}$ (see Section 2), which yields

$$M = \max(|D_1|^{1/4}, |D_1|/2^{\ell_1}).$$

5.2 Asymptotic Behavior of new Trade-Offs‘

For the asymptotic classification of our algorithmic improvement let us first consider the half distance setting, i.e., $\omega := H(1 - \frac{k}{n}) \cdot \frac{n}{2}$. Here our MMT variant improves the memory complexity by almost a square-root down to $2^{0.0135n}$ from $2^{0.0213n}$ of standard MMT, while maintaining the same time complexity of $T = 2^{0.05364n}$. The optimal parameters for our MMT variant in this case are

$$\ell = 0.0278n,\ \ell_1 = 0.0091n \text{ and } p = 0.0064n,$$

where the found worst case rate is $k = 0.45n$ as for standard MMT. We now further optimized the time complexity of our trade-off under a memory limitation of $M \leq 2^{\lambda n}$ for decreasing λ. Figure 4 shows the complete trade-off curves for both MMT variants – the original and our improved version. We observe that our trade-off outperforms the original trade-off for all memory parameters.

In the full distance setting we obtain a similar improvement. Here our improved MMT algorithm improves the memory complexity down to $2^{0.0375n}$ from previously $2^{0.053n}$, while achieving the same time complexity of $2^{0.112n}$. Again we obtain runtime improvements over standard MMT for any fixed memory.

Even though, the MMT algorithm is not the asymptotically fastest ISD algorithm, so far none of its known asymptotic improvements [9,13,14,23,34] did transfer to the implementation level. This makes the MMT algorithm the preferred choice for record computations [3] as well as security estimates [28].

BJMM algorithm. However, we also analyzed the algorithm by Becker-Joux-May-Meurer (BJMM) [9], which in contrast to the MMT algorithm uses slightly different sets D_i. That is, the vectors on each level have a slightly increased weight. Then, in the $\mathbb{F}_2$-addition of those vectors some weight is assumed to cancel to still obtain a vector of weight p. The different possibilities, how the weight can cancel, increase the amount of representations and lead to an increased

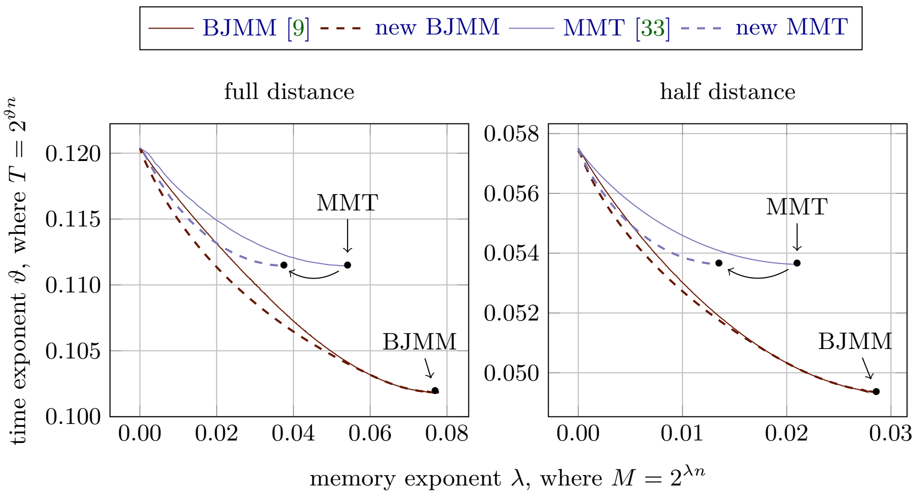

Fig. 4: Comparison between the implicit (solid) and our new trade-off (dashed) for the MMT and BJMM algorithm. Complexity uses known worst case rates of the algorithm in the full distance (left) and half distance setting (right).

optimal tree-depth of three. This increased tree-depth allows us to make use of our subtree reconstruction technique yielding an improved trade-off curve also shown in Figure 4. We observe that the refined choice of the D_i gives the algorithm a possibility to balance the list sizes if memory is not limited. For that reason our strategy yields improvements only for limited memory in the case of the BJMM algorithm.

5.3 Practical Results and Security Estimates

We adapted the MMT / BJMM implementation from [28] to our new trade-off strategy. Interestingly, besides reducing the memory requirements we also obtain practical *running time* improvements, which stem from less, usually costly memory accesses.

We were able to solve several instances provided at `decodingchallenge.org` [3], which were either unsolved or broken using more time and memory. Most notably, we obtained a new record computation in the quasi-cyclic setting, which follows the parameter selection of NIST fourth round candidates BIKE and HQC.

New record computation Precisely, we solved the QC-3138 instance with code parameters $(n, k, \omega) = (3138, 1569, 56)$ with an estimated bit complexity of 66.7 (respectively 60.7 if counted in 64-bit register operations) in only 2.23 CPU years. We estimated the expected time to solve this instance on our cluster, based on the processed permutations per second, to about 9.47 CPU years. The previous best implementation from [28] would need an expected amount of 30.31 CPU years, i.e., our implementation is about 3.2 times faster on this instance.

We also analysed the performance of our implementation on the next instance QC-3366 with parameters $(n, k, \omega) = (3366, 1683, 58)$, which has an estimated bit security of 68.7. We obtain an expected running time of 30.2 CPU years, which corresponds to an improvement by a factor of 5.7.

Furthermore we re-broke the previous QC-2918 record instance with parameters $(n, k, \omega) = (2918, 1459, 54)$ two times in just 224 CPU days, almost precisely hitting its expectation, which is about 6.9 times faster than the previous best implementation.

On McEliece like medium-sized instances we obtain a speedup by a factor of about 2.5. For the current record instance McEliece-1284 using parameters $(n, k, \omega) = (1284, 1028, 24)$ we estimated a running time of 11.06 CPU years, where the initial record computation expected 26.28 CPU years, corresponding to a speedup of about 2.4. Considering the next (unsolved) McEliece record instance with parameters $(n, k, \omega) = (1347, 1078, 25)$, we estimate a running time of about 59.74 CPU years, improving from the previous estimate of 156.6 CPU years by a factor of 2.6.

Security Estimates Next we investigate the impact of our improvement on the security of cryptographic sized instances. Therefore, we first adapted the estimation scripts from [24] to incorporate our trade-off strategy, which allows us to precisely estimate the bit-complexity of given instances. Following previous works [5,24,28] we consider different memory access cost models. A memory access cost tries to model the practically faced memory access timings, by penalizing the algorithm for a high memory usage. Precisely, an algorithm with time complexity T and memory complexity M is assumed to have cost $T \cdot f(M)$, where f determines the penalty. We consider the established models of constant, logarithmic and cube-root access costs, which correspond to $f(M) = 1, f(M) = \log M$ and $f(M) = \sqrt[3]{M}$.

From here we follow two different estimation methodologies. First, we use our estimation script to obtain bit complexity estimates, which we compare directly against similar estimates obtained in [24]. For the second methodology we then extrapolate the time it would take to solve an instance of proposed parameters from our obtained record computations, comparing our results against a similar estimation performed in [28].

Let us start with the bit complexity estimation using our script.

Bit Complexity Estimation The commonly addressed security categories 1, 3 and 5 defined by NIST relate their security to the security of AES-128, -192 and -256. NIST specifies the bit complexity to break those AES instantiations as 143, 207 and 272 respectively.

BIKE / HQC. In Table 3 we state the security margin in bits the corresponding parameter set has over breaking AES with corresponding key-size. Precisely

Quasi-Cyclic		Category 1	Category 3	Category 5
constant:				
BIKE	message	2.90 (0.00)	4.13 (0.00)	4.22 (0.12)
	key	4.31 (0.04)	3.68 (0.08)	6.08 (0.63)
HQC		1.71 (0.00)	5.91 (0.00)	3.08 (0.00)
logarithmic:				
BIKE	message	7.04 (0.46)	8.36 (0.48)	8.62 (0.50)
	key	8.65 (0.42)	8.46 (0.45)	11.29 (0.47)
HQC		5.89 (0.48)	10.18 (0.50)	7.40 (0.52)
cube-root:				
BIKE	message	8.40 (2.57)	9.97 (2.89)	10.41 (3.14)
	key	9.92 (2.34)	9.99 (2.66)	13.01 (2.90)
HQC		7.37 (2.75)	11.91 (3.08)	9.32 (3.30)

Table 3: Bit-difference in security of BIKE/HQC and AES with respective key-length considering different memory access cost.

the table states $T_{\text{Scheme}} - T_{\text{AES}}$, where T_{Scheme} is the bit complexity estimate obtained from our script and T_{AES} the bit complexity of breaking AES, i.e., 143, 207 or 272 respectively. The number in parenthesis states the improvement over the estimation performed in [24], i.e., one obtains their result as the sum of both numbers.

As expected, we obtain essentially the same security margin as [24] if no memory access cost is imposed. However, for logarithmic and, especially, for cube-root memory access costs, our time-memory trade-offs yield reduced security estimates. Furthermore, note that the improvement in the cube-root case is even higher than the improvement of representation-based ISD algorithms like MMT over early algorithms like Stern and Dumer on these instances [24].

In the case of BIKE we distinguish message and key security as both settings allow for slightly different speedups [2].

McEliece. For the round 4 parameter sets of McEliece we performed a similar estimation shown in Table 4.

Since in the McEliece setting ISD algorithms tend to use very high amounts of memory we also consider memory-limited settings. In those we restrict the memory consumption of the algorithm to not exceed 2^{80} or 2^{60} bits respectively. We reduce the security estimates for McEliece by up to 6 bits and obtain the best results in memory-limited settings, where our new time-memory trade-offs can play its strength. Again the number in brackets indicates by how much we reduced the previous estimate from [24].

Note that under cube-root memory access cost none of the optimal algorithmic configurations exceeds 2^{60} bits of memory.

McEliece	Category 1 $n = 3488$	Category 3 $n = 4608$	Category 5a $n = 6688$	Category 5b $n = 6960$	Category 5c $n = 8192$
constant:					
unlimited	-0.98 (0.23)	-25.09 (0.55)	-23.82 (0.17)	-24.51 (0.25)	5.37 (0.27)
$M \leq 80$	0.14 (0.56)	-22.94 (1.74)	-13.42 (1.29)	-13.13 (1.80)	21.40 (1.51)
$M \leq 60$	2.47 (1.49)	-18.84 (0.04)	-8.84 (4.52)	-8.36 (4.10)	26.65 (5.59)
logarithmic:					
unlimited	5.44 (0.22)	-18.39 (0.33)	-16.62 (0.18)	-17.28 (0.25)	12.83 (0.27)
$M \leq 80$	6.14 (0.61)	-16.88 (1.52)	-7.42 (1.23)	-7.13 (1.77)	27.45 (1.45)
$M \leq 60$	8.06 (1.37)	-13.41 (0.13)	-3.34 (4.15)	-2.83 (3.69)	32.19 (5.23)
cube-root:					
	14.25 (1.68)	-7.31 (1.76)	4.13 (2.63)	4.97 (2.35)	41.12 (3.05)

Table 4: Bit-difference in security of McEliece and AES with respective key-length considering different memory access cost.

Extrapolation Methodology. Now, let us provide a security estimation, where we extrapolate the time to solve an instance of suggested parameters from our obtained record computations, as recently proposed in [28]. This methodology scales the time of the largest experiment in the respective setting by the difference in the bit-complexity of our experiment and the suggested parameters.

Methodology Example. Let us give a brief example of that methodology. Take the HQC category 1 parameter set $(n, k, \omega) = (35338, 17669, 132)$, in the constant memory access setting. This instance achieves a bit complexity of 144.7 according to our estimator, while our QC-3138 record has a bit complexity of 66.7 and took us about 2.24 CPU years to compute. We, therefore, extrapolate the time for breaking the HQC 128-bit parameters to $2.24 \cdot 2^{144.7-66.7} \approx 2^{79.16}$ CPU years.

To then set this time into context to the security categories 1, 3 and 5 that relate their security to the security of AES-128, -192 and -256, the time complexity of breaking AES on the used cluster is estimated. Therefore, one benchmarks the number of AES encryptions the cluster is able to perform per second from which the expected time to break AES with respective key size is obtained. While this methodology introduces platform dependencies, it allows for direct comparison between (scaled) practical experiments for both settings.

BIKE / HQC. Table 5 states the security margin (in bits) the corresponding parameter set has over breaking AES. Precisely the table states $\log \frac{T_{\text{Scheme}}}{T_{\text{AES}}}$. Here, T_{Scheme} is the estimated time to break the schemes parameters and T_{AES} the estimated time to break the corresponding AES instantiation on our cluster. The number in parentheses states the improvement over the analysis performed in [28].

Quasi-Cyclic		Category 1	Category 3	Category 5
constant:				
BIKE	message	−0.65 (3.09)	−0.59 (3.09)	0.26 (3.23)
	key	0.73 (3.15)	−1.07 (3.20)	2.13 (3.74)
HQC		−1.84 (3.08)	1.19 (3.09)	−0.86 (3.09)
logarithmic:				
BIKE	message	−0.36 (3.22)	−0.21 (3.25)	0.84 (3.26)
	key	1.24 (3.18)	−0.10 (3.21)	3.50 (3.24)
HQC		−1.51 (3.23)	1.61 (3.26)	−0.38 (3.28)
cube-root:				
BIKE	message	0.37 (4.10)	0.77 (4.43)	1.99 (4.69)
	key	1.89 (3.88)	0.79 (4.21)	4.60 (4.43)
HQC		−0.67 (4.29)	2.71 (4.63)	0.90 (4.85)

Table 5: Bit-difference in security of BIKE/HQC and AES with respective key-length considering different memory access cost obtained via extrapolation methodology.

We now observe already improvements in the constant memory access setting, which reflects our obtained speedup on the mid-sized instance used for the extrapolation. Still we obtain higher gains towards higher memory access costs, due to the reduced memory usage. Overall the margins are slightly lower using the extrapolation methodology compared to the bit complexity estimate, with larger differences towards higher memory access costs. This is because some of the memory access costs are accounted to the mid-sized instance, which is subtracted from the overall estimate in the extrapolation.

McEliece. For the round 4 parameter sets of McEliece we performed a similar extrapolation shown in Table 6. For this extrapolation we used the expected time complexity of 59.74 CPU years for the McEliece-1347 instance.

While reducing the estimate in all settings, the overall picture stays the same under both estimation methods: Essentially all but the category 3 parameter set reach their security claims if cube-root memory access costs are imposed.

Acknowledgements This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID MA 2536/12 and by BMBF under Industrial Blockchain – iBlockchain.

References

1. Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Ishai, Y., Rij-

McEliece	Category 1 $n = 3488$	Category 3 $n = 4608$	Category 5a $n = 6688$	Category 5b $n = 6960$	Category 5c $n = 8192$
constant:					
unlimited	-0.65 (0.74)	-25.93 (1.07)	-23.86 (0.68)	-24.55 (0.75)	5.32 (0.78)
$M \leq 80$	0.47 (1.07)	-23.77 (2.25)	-13.47 (1.80)	-13.18 (2.31)	21.35 (2.02)
$M \leq 60$	2.80 (2.00)	-19.67 (0.55)	$-$ 8.89 (5.03)	$-$ 8.41 (4.61)	26.60 (6.10)
logarithmic:					
unlimited	0.84 (0.93)	-24.16 (1.05)	-21.60 (0.90)	-22.27 (0.98)	7.84 (1.00)
$M \leq 80$	1.53 (1.33)	-22.66 (2.25)	-12.41 (1.95)	-12.12 (2.49)	22.47 (2.17)
$M \leq 60$	3.45 (2.10)	-19.19 (0.86)	$-$ 8.32 (4.86)	$-$ 7.81 (4.41)	27.20 (5.96)
cube-root:					
	8.92 (1.45)	-13.81 (1.54)	$-$ 1.59 (2.41)	$-$ 0.75 (2.13)	35.40 (2.82)

Table 6: Bit-difference in security of Classic McEliece and AES with respective key-length considering different memory access cost obtained via extrapolation methodology.

men, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 717–746. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17656-3_25 2

2. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Guneysu, T., Melchor, C.A., et al.: BIKE: bit flipping key encapsulation (2020) 25
3. Aragon, N., Lavauzelle, J., Lequesne, M.: decodingchallenge.org (2019), http://decodingchallenge.org 5, 22, 23
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, Part I. LNCS, vol. 7965, pp. 45–56. Springer, Heidelberg (Jul 2013). https://doi.org/10.1007/978-3-642-39206-1_5 2
5. Baldi, M., Barenghi, A., Chiaraluce, F., Pelosi, G., Santini, P.: A finite regime analysis of information set decoding algorithms. Algorithms **12**(10), 209 (2019) 24
6. Bardet, M., Bros, M., Cabarcas, D., Gaborit, P., Perlner, R.A., Smith-Tone, D., Tillich, J.P., Verbel, J.A.: Improvements of algebraic attacks for solving the rank decoding and MinRank problems. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part I. LNCS, vol. 12491, pp. 507–536. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64837-4_17 2
7. Becker, A., Coron, J.S., Joux, A.: Improved generic algorithms for hard knapsacks. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 364–385. Springer, Heidelberg (May 2011). https://doi.org/10.1007/978-3-642-20465-4_21 2, 3, 4, 7, 17, 18
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 SODA. pp. 10–24. ACM-SIAM (Jan 2016). https://doi.org/10.1137/1.9781611974331.ch2 2
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.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_31 2, 3, 5, 22, 23

10. Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the McEliece cryptosystem. In: Buchmann, J., Ding, J. (eds.) Post-quantum cryptography, second international workshop, PQCRYPTO 2008. pp. 31–46. Springer, Heidelberg (Oct 2008). https://doi.org/10.1007/978-3-540-88403-3_3 19
11. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. In: 32nd ACM STOC. pp. 435–440. ACM Press (May 2000). https://doi.org/10.1145/335305.335355 2
12. Bonnetain, X., Bricout, R., Schrottenloher, A., Shen, Y.: Improved classical and quantum algorithms for subset-sum. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 633–666. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_22 3, 4, 6, 10, 13, 17
13. Both, L., May, A.: Optimizing bjmm with nearest neighbors: full decoding in 22/21n and mceliece security. In: WCC workshop on coding and cryptography. vol. 214 (2017) 22
14. Both, L., May, A.: Decoding linear codes with high error rate and its impact for LPN security. In: Lange, T., Steinwandt, R. (eds.) Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018. pp. 25–46. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-79063-3_2 22
15. Bricout, R., Chailloux, A., Debris-Alazard, T., Lequesne, M.: Ternary syndrome decoding with large weight. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 437–466. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-38471-5_18 2
16. Delaplace, C., Esser, A., May, A.: Improved low-memory subset sum and LPN algorithms via multiple collisions. In: Albrecht, M. (ed.) 17th IMA International Conference on Cryptography and Coding. LNCS, vol. 11929, pp. 178–199. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-35199-1_9 3, 4, 17
17. Dinur, I.: An algorithmic framework for the generalized birthday problem. Designs, Codes and Cryptography pp. 1–30 (2018) 2, 3, 5
18. Dinur, I.: Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 374–403. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5_14 2
19. 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.) CRYPTO 2012. LNCS, vol. 7417, pp. 719–740. Springer, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_42 2, 3, 4, 5, 7, 13, 17
20. Ducas, L., Stevens, M., van Woerden, W.P.J.: Advanced lattice sieving on GPUs, with tensor cores. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 249–279. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77886-6_9 2
21. Dumer, I.: On minimum distance decoding of linear codes. In: Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory. pp. 50–52 (1991) 3
22. Esser, A.: Memory-efficient algorithms for solving subset sum and related problems with cryptanalytic applications. Ph.D. thesis, Ruhr University Bochum, Germany (2020) 3, 4, 17
23. Esser, A.: Revisiting nearest-neighbor-based information set decoding. Cryptology ePrint Archive (2022) 22

24. Esser, A., Bellini, E.: Syndrome decoding estimator. In: Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography. Lecture Notes in Computer Science, vol. 13177, pp. 112–141. Springer (2022) 24, 25
25. Esser, A., Kübler, R., May, A.: LPN decoded. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 486–514. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63715-0_17 2
26. Esser, A., May, A.: Better sample–random subset sum in $2^{0.255n}$ and its impact on decoding random linear codes. arXiv preprint arXiv:1907.04295, withdrawn (2019) 4, 10
27. Esser, A., May, A.: Low weight discrete logarithm and subset sum in $2^{0.65n}$ with polynomial memory. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 94–122. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45727-3_4 2, 3, 4, 17
28. Esser, A., May, A., Zweydinger, F.: McEliece needs a break - solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 433–457. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_16 2, 5, 19, 22, 23, 24, 26
29. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. Journal of the ACM (JACM) **21**(2), 277–292 (1974) 3, 7
30. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 235–256. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_12 2, 3, 4, 7, 17, 18
31. Karpman, P., Lefevre, C.: Time-memory tradeoffs for large-weight syndrome decoding in ternary codes. In: Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography. Lecture Notes in Computer Science, vol. 13177, pp. 82–111. Springer (2022) 4
32. May, A.: How to meet ternary LWE keys. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 701–731. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84245-1_24 2
33. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124. Springer, Heidelberg (Dec 2011). https://doi.org/10.1007/978-3-642-25385-0_6 2, 3, 5, 20, 23
34. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 203–228. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_9 3, 22
35. Nguyen, P.Q., Shparlinski, I.E., Stern, J.: Distribution of modular sums and the security of the server aided exponentiation. In: Cryptography and Computational Number Theory. pp. 331–342. Springer (2001) 18
36. Nikolic, I., Sasaki, Y.: Refinements of the k-tree algorithm for the generalized birthday problem. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 683–703. Springer, Heidelberg (Nov / Dec 2015). https://doi.org/10.1007/978-3-662-48800-3_28 3
37. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962) 3
38. 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) 3, 7

39. Stern, J.: A method for finding codewords of small weight. In: International Colloquium on Coding Theory and Applications. pp. 106–113. Springer (1988) 3, 4
40. Torres, R.C., Sendrier, N.: Analysis of information set decoding for a sub-linear error weight. In: Takagi, T. (ed.) Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016. pp. 144–161. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-29360-8_10 19
41. Udovenko, A., Vitto, G.: Breaking the $ikep182 challenge. Cryptology ePrint Archive, Report 2021/1421 (2021), https://eprint.iacr.org/2021/1421 2
42. Various: Round 3 official comment: Classic McEliece (2021), available at: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/ldAzu9PeaIM/m/VhLBcydEAAAJ 5
43. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–303. Springer, Heidelberg (Aug 2002). https://doi.org/10.1007/3-540-45708-9_19 3
44. Wang, M., Liu, M.: Improved information set decoding for code-based cryptosystems with constrained memory. In: International Workshop on Frontiers in Algorithmics. pp. 241–258. Springer (2015) 4

A Generalization to arbitrary depth d

Note that in general we have

$$\mathcal{L}_{i+1} = \frac{(q_i \cdot \mathcal{L}_i)^2}{2^{\ell_i}},$$

where ℓ_i is the additional bitwise constraint introduced on level i. The time and memory complexity are then given as before. The saturation constraints extend to

$$q_i \cdot \mathcal{L}_i \leq \frac{|D_i|}{2^{\ell_1 + \ldots + \ell_i}} \text{ for } i = 2, \ldots, d-1,$$

where d is the depth of the tree. Together with the definition of the filtering probability given in Equation (4), we can rewrite the saturation constraints for each level i as

$$\sum_{j=1}^{i} (2^{i-j} - 1)\ell_j \geq \sum_{j=1}^{i} 2^{i-j} \cdot r_j \text{ for } i = 1, \ldots d-2,$$

where there exist 2^{r_j} different representations of any element from D_{j+1} as a sum of two elements from D_j. Finally, the requirement of finding one representation of the solution in the final list is expressed via the condition

$$q_d \cdot \mathcal{L}_d = 1,$$

which similar to the saturation constraints rewrites to

$$\sum_{j=1}^{d-1} (2^{d-j} - 1)\ell_j = \sum_{j=1}^{d-1} 2^{d-j-1} \cdot r_j. \tag{8}$$