Memory-Efficient Attacks on Small LWE Keys*

Andre Esser[1]**, Arindam Mukherjee[2], and Santanu Sarkar[2]***

[1] Technology Innovation Institute, UAE
andre.esser@tii.ae
[2] Department of Mathematics, Indian Institute of Technology Madras, Chennai, India
arindamaths@gmail.com, sarkar.santanu.bir1@gmail.com

Abstract. Combinatorial attacks on small max norm LWE keys suffer enormous memory requirements, which render them inefficient in realistic attack scenarios. Therefore, more memory-efficient substitutes for these algorithms are needed. In this work, we provide new combinatorial algorithms for recovering small max norm LWE secrets outperforming previous approaches whenever the available memory is limited. We provide analyses of our algorithms for secret key distributions of current NTRU, Kyber and Dilithium variants, showing that our new approach outperforms previous memory-efficient algorithms. For instance, considering uniformly random ternary secrets of length n we improve the best known time complexity for *polynomial memory* algorithms from $2^{1.063n}$ down-to $2^{0.926n}$. We obtain even larger gains for LWE secrets in $\{-m,\dots,m\}^n$ with $m = 2, 3$ as found in Kyber and Dilithium. For example, for uniformly random keys in $\{-2,\dots,2\}^n$ as is the case for Dilithium we improve the previously best time under polynomial memory restriction from $2^{1.742n}$ down-to $2^{1.282n}$. Eventually, we provide novel time-memory trade-offs continuously interpolating between our polynomial memory algorithms and the best algorithms in the unlimited memory case (May, Crypto 2021).

Keywords: Learning with Errors · combinatorial attacks · nested collision search · representation technique · polynomial memory · time-memory trade-off

1 Introduction

The Learning with Errors (LWE) problem is one of the most promising candidates for post-quantum cryptographic constructions. Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ and a vector $\mathbf{b} = \mathbf{As} - \mathbf{e} \in \mathbb{Z}_q^n$, where $\mathbf{e}$ is a short error vector, the problem asks to recover the secret vector $\mathbf{s}$. The LWE problem is known to be as hard as some

* This is the extended version of the paper [19] from ASIACRYPT 2023. This version includes new Time-Memory Trade-Offs in Section 7.2 and Appendix D. Additionally, all other appendices provide explanations that were not given in [19].

** supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID MA 2536/12

*** supported by SERB Core Research Grant - CRG/2023/001478

worst case lattice problems, which made it an attractive choice as foundation for several efficient cryptographic systems [7,10,25,33,39,41,42]. The most efficient of these schemes rely on ring variants of LWE, which exploit the algebraic structure of the underlying rings to represent the matrix $\mathbf{A}$ [10,34]. Further, some schemes restrict the error term $\mathbf{e}$, as well as the vector $\mathbf{s}$, to vectors with small max norm [7,17,27,32]. Crystals-Kyber [10], which was recently announced to be standardised by NIST, for example, samples key and error from a centered binomial distribution, which in turn results in small max norm key and error of norm 2 or 3. NTRU-type schemes go even further and choose ternary secrets with coefficients in $\{0, \pm 1\}$, i.e., with max norm 1. Usually, these are efficiency driven decisions, whose security argument is based on the lack of faster algorithms to solve these variants, since lattice reduction is not known to be able to exploit small max norm. However, the best attack on ternary LWE keys is considered to be a combination of combinatorial attacks and lattice reduction, known as *the hybrid attack* introduced by Howgrave-Graham [30]. Internally, this attack balances the complexity of an involved meet-in-the-middle and a lattice reduction step. Therefore, progress on combinatorial attacks has a strong potential to affect parameter selection for those schemes. Putting the focus on the NTRU-family of schemes and its variants we concentrate in this work on LWE with ternary secrets. However, our attacks also translate well to higher max norm variants as we showcase by an application to LWE keys as found in Kyber and Dilithium (see Section 6).

Intuitively, it is clear that small max norm keys with reduced search space of size $\mathcal{D}$ allow for faster combinatorial attacks that rely on enumerating possible keys. However, for a long time, the best combinatorial algorithm was a basic meet-in-the-middle attack by Odlyzko from 1996, mentioned in the original NTRU paper [29], achieving a running time of $\mathcal{D}^{0.5}$. Recently, May [35] showed how to adapt advanced techniques from solving the subset sum problem to the small max norm LWE setting. This results in significant improvements of the running time to approximately $\mathcal{D}^{0.25}$ for ternary LWE keys.

However, the biggest obstacle of all combinatorial approaches, including the results by May and its recent adaptation to the cases of Kyber and Dilithium [26], is their huge memory complexity, which is as high as their time complexity. Even if such large amounts of memory should be ever available, the slowdown emerging from accessing such large-scale memories is likely to render those algorithms inefficient.

In contrast, in this work we first provide new (heuristic) algorithms for solving the LWE problem with small max norm secrets using only *polynomial memory*, after which we extend our results to exploit arbitrary amounts of memory. However, polynomial memory algorithms are already of crucial importance to cryptanalysis for several reasons. On the one hand, they allow for very efficient implementations on inherently memory constrained platforms such as FPGAs or even more commonly used GPUs [6,18,37,38]. Practical record computations, therefore, often start from a low-memory algorithm, with only polynomial memory requirement, which is then supported by the available memory if possible [11,21,

43]. Further, aiming at near- to mid-term quantum cryptanalytic implementations, the focus has to be on low-memory algorithms.

Our polynomial memory algorithms almost achieve the same running time as Odlyzko's meet-in-the-middle, i.e., $\mathcal{D}^{0.5}$, while in contrast only using a negligible amount of memory. Our fastest construction is based on a variety of different techniques, but at its heart lies a nested collision search procedure inspired by the nested rho technique from [16], which is also the foundation of the fastest (heuristic) polynomial space algorithm for subset sum [20]. Our analyses, thereby, rely only on mild heuristics, which are frequently applied and experimentally verified in the context of collision search and the representation technique. Asymptotically our approach outperforms pure lattice enumeration, which has also only polynomial space requirements, but comes at a running time of $2^{cn \log n}$, where c is a constant and n the LWE dimension [3, 24]. In contrast our algorithms' running times are single exponential in the LWE dimension, i.e., of the form $2^{c'n}$ for a constant c'. Further, we significantly improve the constant c' in comparison to previously suggested memoryless algorithms based on conventional collision search techniques, such as [35, 44].

We then present new continuous time-memory trade-offs able to exploit any available amount of memory, interpolating between our polynomial memory endpoints and May's algorithms in the case of unlimited memory. To obtain these results we first extend our polynomial memory algorithms to leverage higher amounts of memory using standard techniques. After that we revisit May's algorithms and show how to adapt recent time-memory trade-off techniques from the subset sum case [23] to tailor the memory usage of May's algorithms to any available amount.

With respect to concrete, currently proposed parameters, pure combinatorial attacks, such as Odlyzko's, May's and ours, are quite far from competing against pure lattice strategies.[3] Hence, our attacks, analogous to those of May [35], do not invalidate security claims of currently suggested parameters as we improve primarily on the memory complexity. However, advances on those attacks, on the one hand, strengthen our understanding of the hardness of those problems by providing clean combinatorial upper bounds; especially they clarify the effect of the sparsity of the secret, heavily exploited by those strategies, showing that overly sparse choices might lead to unwanted drops in security. Furthermore and probably most importantly, combinatorial attacks have a huge potential to improve the Hybrid attack by replacing Odlyzko's meet-in-the-middle with faster routines, such as, May's [35], or more memory-efficient strategies, such as ours. However, replacing Odlyzko's is not possible in a plug-and-play manner as detailed and posed as an open question in [35]. Since then the problem has been actively investigated by multiple recent works [8, 28], and once a clear consensus is reached, we also expect practical implications of our attacks.

[3] Best runtime results from May [35] are slightly less than the square of current lattice complexities.

Our Contribution We first revisit basic collision search techniques for solving the ternary LWE problem introduced by van Vredendaal [44] and recently refined by May [35] to set the baseline for our new algorithmic improvements in the polynomial memory regime. In this context, as a small initial contribution, we provide a single framework from which the algorithms of [44] as well as all variations given in [35] can be obtained as different instantiations.

We then introduce our novel nested collision search algorithm that leads to significant runtime improvements over previous approaches. In terms of the search space size $\mathcal{D}$ our nested algorithm applied to ternary LWE achieves approximately a running time of $\mathcal{D}^{0.55}$, which is just slightly higher than the running time of Odlyzko's meet-in-the-middle but reduces the memory from $\mathcal{D}^{0.5}$ to a negligible amount. In comparison, the polynomial memory technique of van Vredendaal obtains a running time of $\mathcal{D}^{0.75}$, while May obtains roughly $\mathcal{D}^{0.65}$.[4] For keys following distributions as in Kyber, we get even closer to meet-in-the-middle's running time by reaching $\mathcal{D}^{0.513}$ and $\mathcal{D}^{0.508}$ respectively. We illustrate the running time exponent of our algorithm on ternary LWE in comparison to van Vredendaal and May as a function of the Hamming weight w of the solution in Fig. 1. We observe that our technique outperforms both previous methods for all choices of the weight. Furthermore, in contrast to May's method, our technique follows the natural behavior of a reduced time complexity for high weights, i.e., when the search space starts decreasing again.

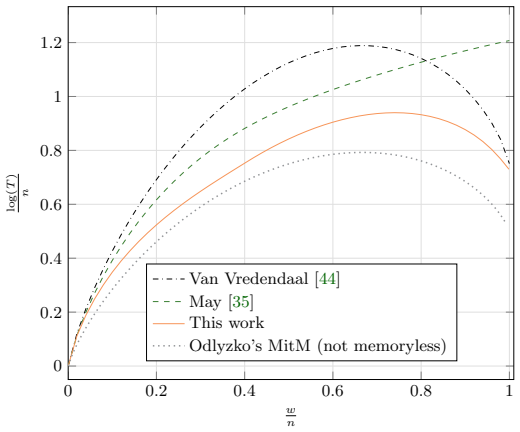


Fig. 1: Runtime exponents c as a function of the relative weight w/n for different polynomial memory algorithms and Odlyzko's MitM, with memory equal to time. The running time is of the form $T = 2^{cn+o(n)}$.

[4] Since May's algorithm performance is worse towards high weights, we considered for this comparison only weights $w/n \leq \frac{2}{3}$.

On the technical side, we employ multiple techniques to make the nested approach functional and efficient. Methods based on conventional collision search rely on Odlyzko's hash function to eliminate $\mathbf{e}$ from the LWE identity. This gives an exact identity which can then be formulated as collision search problem. However, while the solution forms a collision between the defined functions by construction, not necessarily every collision leads to the solution. Therefore, the collision search needs to be re-applied an exponential number of times until a collision is found that gives rise to the solution.

In a nutshell, we replace the iterative application of the collision search by another layer of collision search. While this increases the time to perform a single (two-layer) collision search, it is compensated by eliminating the need for multiple iterations, as a single (two-layer) collision search suffices to identify the solution. Unfortunately, Odlyzko's hash function is not well compatible with our nested approach. First, it is not additive, which is crucial to enable the nesting and its output of only n bits is not sufficient for both collision searches. However, we circumvent this problem by adapting a guessing strategy introduced in [35] in the context of non-polynomial space algorithms. Here, we first guess $r := \frac{n}{\log n}$ coordinates of $\mathbf{e}$, which can be done in subexponential time $\mathcal{O}(3^r)$. We then use the resulting exact identity to identify in the first layer collision search those elements $(\mathbf{x}, \mathbf{y})$ that fulfill the LWE identity $\mathbf{A}(\mathbf{x} + \mathbf{y}) = \mathbf{b} + \mathbf{e}$ on the r known coordinates. In the second layer, we may then again rely on Odlyzko's hash function to extract the solution, similar to the conventional methods. Further, to make the nesting efficient, we incorporate the representation technique from subset sum [31], which allows to increase the number of collisions that give rise to the solution. It has previously been observed that the digit set, i.e., the alphabet to which the coordinates of the vectors $\mathbf{x}, \mathbf{y}$ belong, plays a crucial role for the number of representations [4, 9, 35]. In this context, we also provide the quite technical analysis for an extended digit set of $\{0, \pm 1, \pm 2\}$, i.e., $\mathbf{x}, \mathbf{y} \in \{0, \pm 1, \pm 2\}^n$, to obtain further improvements. Eventually, we use several further tricks to speed up our procedure. Therefore we embed the concept of partial representations introduced in [12, 20] and combine it with an initial instance permutation, similar to the one in [20]. Further, we borrow techniques from decoding random linear codes [40] (Information Set Decoding) to obtain improvements, especially in the case of uniform random ternary secrets. We then extend all our results to the cases of Kyber and Dilithium involving digit sets of $\{-3, \ldots, 3\}$. For a better comparison, we also extend the polynomial memory results from May, which were originally only provided for ternary keys.

Eventually, we extend our algorithms to the exponential memory setting by introducing a Parallel Collision Search (PCS) [43] based time-memory trade-off resulting in competitive instantiations for small amounts of memory. We then further show how to combine May's algorithms with recently introduced trade-off techniques to extend them to any available amount of memory, resulting in the fastest instantiations for high amounts of memory. Overall, we obtain a continuous time-memory trade-off interpolating between both endpoints. We depict this trade-off for the exemplary case of $w/n = 0.5$ in the ternary case in Fig. 2

in comparison to a combinatorial trade-off resulting from a direct application of a time-memory trade-off framework called *Dissection* introduced by Dinur, Dunkelmann, Keller and Shamir [15]. Additionally, as a small contribution, we are able to slightly improve the *running time* of May's algorithms by allowing for a more flexible choices of parameters.

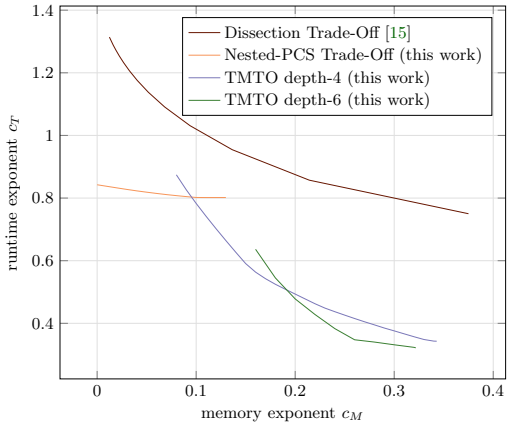


Fig. 2: Runtime exponents c as a function of the available memory for relative weight $w/n = 0.5$ in the case of ternary LWE. The running time is of the form $T = 2^{c_T n + o(n)}$ and the memory consumption is $2^{c_M n + o(n)}$.

Heuristic assumptions. When applied to random LWE instances our algorithms rely only on standard assumptions in the context of collision search and representation based algorithms, which have been extensively verified in multiple prior works [16, 20, 21, 31]. However, we also provide experimental data that verify those assumptions in our precise setting in Appendix C.

An application of our results to structured LWE instances, as found in Kyber, Dilithium or NTRU, further requires the assumption that the introduced structure does not affect the behavior of our algorithms. Note that this assumption is common in the analysis of combinatorial algorithms in the LWE context [26, 35] and was recently made more explicit by Glaser-May [26]. Also, a similar assumption is required in the related code-based setting when applying such algorithms to structured candidates like BIKE or HQC, which has held true in extensive practical experiments [21, 23].

Source Code. All used source code is available at `https://github.com/arindamIITM/Small-LWE-Keys`.

Outline. In Section 2 we give basic notations and definitions including the formalization of the ternary LWE problem and we recall standard techniques

for collision search. Subsequently, in Section 3 we give a framework for methods solving LWE via conventional collision search from which we derive the algorithms of van Vredendaal and May. We give our main result, the nested-collision technique together with several improvements in Section 4. In Section 5 we conclude the ternary analysis with a detailed comparison of our new method and previous approaches, while in Section 6 we provide runtime results of our attacks applied to Kyber and Dilithium keys. Eventually, we present a time-memory trade-off for small but exponential amounts of memory in Section 7.1.

2 Preliminaries

We denote vectors as bold lower case and matrices as bold upper case letters. For a vector $\mathbf{x}$ and an integer ℓ we denote by $\pi_\ell(\mathbf{x}) := (x_1, \ldots, x_\ell)$ the canonical projection to the first ℓ coordinates of $\mathbf{x}$. For a vector $\mathbf{s} \in \mathbb{Z}_q^n$ its Hamming weight or just weight is defined as the number of non-zero coordinates of $\mathbf{s}$.

2.1 Complexity Statements

For complexity statements we use standard Landau notation, where $\tilde{\mathcal{O}}$-notation suppresses polylogarithmic factors. In this context, we frequently use the well known approximation for multinomial coefficients that can be derived from Stirling's formula

$$\binom{n}{k_1 n, \ldots, k_p n} = \tilde{\mathcal{O}}\left(2^{H(k_1,\ldots,k_p)n}\right), \tag{1}$$

where H denotes the Shannon entropy function $H(k_1, \ldots, k_p) = -\sum_1^p k_i \log_2(k_i)$ with $\sum_1^p k_i = 1$. Since k_p is fully determined by the remaining k_i's we define the following notation $\binom{n}{k_1 n, \ldots, k_{p-1} n, \cdot} := \binom{n}{k_1 n, \ldots, k_p n}$.

2.2 LWE and Ternary Vectors

In this work, we focus on LWE instances with max norm one, i.e., ternary secrets and errors. However, in principle our techniques extend to any constant max norm, as we show by application to LWE with secrets in $\{-m, \ldots, m\}^n$ for $m = 2, 3$ in Section 6.

Definition 1 (Ternary LWE problem). *Let $n \in \mathbb{N}$ and $q = poly(n)$. Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$, a vector $\mathbf{b} \in \mathbb{Z}_q^n$ and an integer w the ternary LWE problem asks to find a vector $\mathbf{s} \in \{-1, 0, 1\}^n$ of weight w satisfying the* LWE identity $\mathbf{As} = \mathbf{b} + \mathbf{e} \mod q$, *where $\mathbf{e} \in \{-1, 0, 1\}^n$ is an arbitrary ternary vector.*

Motivated by cryptographic constructions our definition covers only square matrices $\mathbf{A}$, even though our results extend well to the non-square case. Further, we restrict the modulus $q = \text{poly}(n)$ which is proven to be a hard regime and larger choices might allow for faster attacks [2].

In our analysis we assume all entries of the matrix $\mathbf{A}$ are drawn independently and uniformly at random from $\mathbb{Z}_q$. Note that, apart from ring LWE instantiations this is generally the case and we do not exploit the ring structure in our attacks. Moreover, we only consider the case of balanced weight-w solutions, i.e., solutions with the same amount of $w/2$ entries equal to 1 and $w/2$ entries equal to -1. Most NTRU-type instantiations, such as NTRU, GLP, and BLISS, use balanced weight secrets by default. However even if the proportion of ones and minus ones is unknown, our attacks can easily be generalized by iterating our procedures for each possible proportion. For constant max norm secrets this results at most in a polynomial overhead. In this context, we denote the set of ternary vectors of length n and balanced weight w as $\tau^n(w/2)$, that is,

$$\tau^n(w/2) = \{\mathbf{s} \in \{0, \pm 1\}^n : \mathbf{s} \text{ has } w/2 \text{ many 1-entries} \wedge w/2 \text{ many } (-1)\text{-entries}\}.$$

Odlyzko's Hash Function In the context of the LWE problem, Odlyzko made use of a locality sensitive hash function that eliminates the unknown ternary vector $\mathbf{e}$ from the LWE identity. For a vector $x \in \mathbb{Z}_q^n$ the hash function maps each coordinate $x_i \in \{-\lfloor q/2 \rfloor, \ldots, 0, \ldots, \lfloor q/2 \rfloor\}$ to its sign. More precisely let us define $\hbar : \mathbb{Z}_q^n \to \{0,1\}^n$ in the following way. For $\mathbf{x} \in \mathbb{Z}_q^n$ we coordinate-wise assign the binary hash label $\hbar(\mathbf{x})_i$ where,

$$\hbar(\mathbf{x})_i = \begin{cases} 0, & \text{if } x_i < 0 \\ 1, & \text{if } x_i \geq 0 \end{cases}$$

Note that, as long as $\mathbf{e}$ does not cause the signs of both sides of the LWE identity to diverge we have $\hbar(\mathbf{As}) = \hbar(\mathbf{b})$. Such a divergence can only happen if there are coordinates equal to -1 or $\lfloor q/2 \rfloor$ present in $\mathbf{As}$ or $\mathbf{b}$, which are called *edge cases*. Therefore, split the ternary $\mathbf{e} = \mathbf{e}_1 - \mathbf{e}_2$ with $\mathbf{e}_i \in \{0,1\}^n$ and rewrite the LWE identity as $\mathbf{As} + \mathbf{e}_2 = \mathbf{b} + \mathbf{e}_1$. Now the addition of $\mathbf{e}_i$ can only cause a sign flip for the mentioned edge cases of -1 or $\lfloor q/2 \rfloor$ coordinates.

2.3 Collision Search

Let $f : S \to S$ be any random function on S. Then a collision in f defines a tuple $(y_1, y_2) \in S^2$ with $f(y_1) = f(y_2)$. Such a collision can be found using $\mathcal{O}(\sqrt{|S|})$ evaluations of f and polynomial memory. The standard technique is to create a chain of invocations of the function f from a random starting point x. That is iterating $f(x)$, $f^2(x)$, $f^3(x)$, ..., until a repetition occurs, which is found via a cycle detection algorithm. Let $f^k(x)$ be the first repeated value in the chain and let $f^{k+l}(x)$ be its second appearance (compare to Fig. 3). We denote the output of a collision finding algorithm on f with starting point x as $\textsc{Rho}(f, x)$ which gives the colliding inputs. More precisely,

$$\textsc{Rho}(f, x) = (f^{k-1}(x), f^{k+l-1}(x)).$$

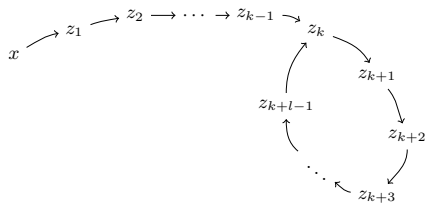


Fig. 3: Application of Rho - function for f with starting point x. $f^i(x)$ is denoted by z_i.

The technique also extends to finding collisions between two different functions, i.e., two random functions $f_1 : S \to S$ and $f_2 : S \to S$. Therefore we define another function $F : S \to S$ as

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

where $g : S \to \{0, 1\}$ is a random function. Now we search for collisions in F using the previously discussed method. A collision (y_1, y_2) in F, i.e., $F(y_1) = F(y_2)$, yields a collision between f_1 and f_2 iff $g(y_1) \neq g(y_2)$, which happens with probability $\frac{1}{2}$. In case of $g(y_1) = g(y_2)$, one might (deterministically) change the starting point and reapply the procedure. Since, in expectation, this results only in a constant factor overhead, we conveniently write $\text{Rho}(f_1, f_2, x)$ to denote the collision (y_1, y_2) between f_1 and f_2 reachable from starting point x still using $\mathcal{O}(\sqrt{|S|})$ evaluations of the function F.

Note that several starting points x might lead to the same collision (y_1, y_2), for instance any point $z_1, \ldots, z_{k-1}$ in Fig. 3 produces the same collision (z_{k-1}, z_{k+l-1}). To obtain (heuristic) independence between different calls to the Rho function we introduce randomizations of the functions called flavors.

Definition 2 (Flavour of a function). *Let $f : S \to S$ be a function and $P_t : S \to S$ be a family of bijective functions indexed by $t \in \mathbb{N}$. Then the t^{th} flavour of f is defined as*

$$f^{[t]}(x) := P_t(f(x)).$$

A collision (y_1, y_2) in $f^{[t]}$ satisfies

$$f^{[t]}(y_1) = f^{[t]}(y_2) \quad \Leftrightarrow \quad P_t(f(y_1)) = P_t(f(y_1)) \quad \Leftrightarrow \quad f(y_1) = f(y_2).$$

Hence, (y_1, y_2) is a collision in f itself. When searching for collisions in randomly flavored functions, i.e., for random choices of t, we (heuristically) assume that different invocations of the Rho-function produce independent and uniformly at random drawn collisions form the set of all collisions. This is a standard assumption in the context of collision search [4, 16, 20] which has been verified experimentally multiple times [16, 20] in different settings.

3 Solving LWE via Collision Search

For didactic reasons and to set the baseline for our improvements, let us start by recalling the memory-less attacks given by van Vredendaal [44] and more recently by May [35] which are based on conventional collision search.

Let us first give a general framework for this kind of attack, which later allows to instantiate the different algorithms. Recall the LWE identity

$$\mathbf{As} = \mathbf{b} + \mathbf{e} \mod q, \tag{2}$$

where $\mathbf{A}, \mathbf{b}$ are known. We split $\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$ in the sum of two addends, where $\mathbf{s}_i \in \mathcal{T}_i$.[5] Further, we define the two functions $f_i \colon \mathcal{T}_i \to \{0,1\}^\ell$, $i = 1, 2$ where

$$f_1 \colon \mathbf{x} \mapsto \pi_\ell\big(\hbar(\mathbf{Ax})\big) \quad \text{and} \quad f_2 \colon \mathbf{x} \mapsto \pi_\ell\big(\hbar(\mathbf{b} - \mathbf{Ax})\big).$$

Hence, the functions output the first ℓ bits of Odlyzko's hash function applied to the respective input. Note that, as long as we restrict to no edge cases regarding the hash function $\hbar$ (see Section 2), any tuple $(\mathbf{s}_1, \mathbf{s}_2)$ that sums to $\mathbf{s}$ forms a collision between the functions f_1 and f_2. The algorithms now search for collisions in f_1, f_2 until they find a collision $(\mathbf{x}, \mathbf{y})$ for which $\mathbf{A}(\mathbf{x} + \mathbf{y}) - \mathbf{b}$ and $\mathbf{x} + \mathbf{y}$ are both ternary, and then outputs $\mathbf{s} = \mathbf{x} + \mathbf{y}$.

Remark 1 (Hashing back to the range). Technically, for a collision search procedure as outlined in Section 2 to work, the used functions need to have same domain and range, as they are iteratively applied to their own output. However, for simplicity of notation, we only ensure that domain and range have the same size in all our algorithms. Prior to applying the functions to their own output, one would apply a bijective mapping from the range to the domain, i.e., here from $\{0,1\}^\ell$ to $\mathcal{T}_i$.

Algorithm 1: Collision-Search

Input: $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n$, positive integer $w \leq n$
Output: $\mathbf{s} \in \tau^n(w/2)$ such that $\mathbf{e} = \mathbf{As} - \mathbf{b} \mod q \in \{-1, 0, 1\}^n$

$\ell := \log |\mathcal{T}_1|$
repeat
  choose random flavour for f_1, f_2
  choose random starting point $\mathbf{v} \in \{0,1\}^\ell$
  $(\mathbf{z}_1, \mathbf{z}_2) \leftarrow \text{Rho}(f_1, f_2, \mathbf{v})$
until $\mathbf{z}_1 + \mathbf{z}_2 \in \tau^n(w/2) \wedge \mathbf{A}(\mathbf{z}_1 + \mathbf{z}_2) - \mathbf{b} \in \{-1, 0, 1\}^n$
return $\mathbf{s} = \mathbf{z}_1 + \mathbf{z}_2$

[5] The precise choice of $\mathcal{T}_i$ depends on the specific instantiation and is described later.

Correctness of Algorithm 1. To ensure that our functions have domain and range of same size we choose $\ell := \log|\mathcal{T}_1|$ and guarantee $|\mathcal{T}_1| = |\mathcal{T}_2|$ by our later choice of $\mathcal{T}_1, \mathcal{T}_2$.

Note that for any $\mathbf{s}_1, \mathbf{s}_2$ that sums to $\mathbf{s}$ we have $f_1(\mathbf{s}_1) = f_2(\mathbf{s}_2)$, as long as there is no edge case among the lower ℓ coordinates of $\mathbf{A}\mathbf{s}_1$ and $\mathbf{b} - \mathbf{A}\mathbf{s}_2$, i.e. a $\mathbb{Z}_q$ coordinate equal to $\lfloor q/2 \rfloor$ or -1. In [44] it was shown, that the probability of no edge case occurring for such a pair is constant. Therefore as long as the function domains include at least a single *representation* of $\mathbf{s}$, i.e., a pair $(\mathbf{s}_1, \mathbf{s}_2) \in \mathcal{T}_1 \times \mathcal{T}_2$ with $\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$, there is a collision that leads to the solution with constant probability. Now, by the standard assumption that the collisions sampled by the algorithm for different function flavors are independent and uniform, the algorithm is able to find this collision and hence, succeeds with constant probability.

Complexity of Algorithm 1. If f_1, f_2 behave like random functions, we expect that there exists a total amount of

$$\frac{|\mathcal{T}_1| \cdot |\mathcal{T}_2|}{|\{0,1\}^\ell|} = \frac{|\mathcal{T}_1|^2}{|\mathcal{T}_1|} = |\mathcal{T}_1|$$

collisions, between them, since $\ell := \log|\mathcal{T}_1|$ and $|\mathcal{T}_1| = |\mathcal{T}_2|$. Further, we know that finding one of these collisions takes time $\tilde{\mathcal{O}}\left(\sqrt{|\mathcal{T}_1|}\right)$. If now there exist R representations of $\mathbf{s}$, i.e., pairs $(\mathbf{s}_1, \mathbf{s}_2) \in \mathcal{T}_1 \times \mathcal{T}_2$ that sum to $\mathbf{s}$, we expect that after finding $\frac{|\mathcal{T}_1|}{R}$ collisions, we found one that is a representation of $\mathbf{s}$. Finding these $\frac{|\mathcal{T}_1|}{R}$ collisions takes expected time

$$T = \tilde{\mathcal{O}}\left(|\mathcal{T}_1|/R \cdot \sqrt{|\mathcal{T}_1|}\right) = \tilde{\mathcal{O}}\left(|\mathcal{T}_1|^{3/2}/R\right).$$

Remark 2 (Random behavior of the functions). All algorithms following this framework are based on the heuristic assumption that the constructed functions behave like random functions with respect to collision search and the total number of existing collisions. This assumption has been verified experimentally various times in different settings [1, 14, 16, 20, 43]. We provide additional experimental evidence for its validity in our precise setting in Appendix C.

The different algorithms from [35, 44] now differ in their choice of function domains $\mathcal{T}_i$.

Van Vredendaal's Instantiation Van Vredendaal [44] chooses a meet in the middle split of $\mathbf{s}$, i.e.,

$$\mathcal{T}_1 := \{(\mathbf{x}, 0^{n/2}) \mid \mathbf{x} \in \tau^{n/2}(w/4)\}$$
$$\mathcal{T}_2 := \{(0^{n/2}, \mathbf{x}) \mid \mathbf{x} \in \tau^{n/2}(w/4)\}.$$

The algorithm assumes that -1 and 1 entries of $\mathbf{s}$ distribute evenly on both sides. Note that if this is not the case one might re-randomize the initial instance by

permuting columns of $\mathbf{A}$, as $\mathbf{AP}$, with solution $\mathbf{P}^{-1}\mathbf{s}$, where $\mathbf{P}$ is a permutation matrix. The expected amount of random permutations until we obtain the desired weight distribution is

$$\frac{\binom{n}{w/2,w/2,\cdot}}{\binom{n/2}{w/4,w/4,\cdot}^2} = \mathrm{poly}(n),$$

which vanishes in our asymptotic notation. For evenly distributed $\mathbf{s}$ and this specific choice of domains $\mathcal{T}_i$, we have clearly only one representation $(\mathbf{s}_1, \mathbf{s}_2) \in \mathcal{T}_1 \times \mathcal{T}_2$ of $\mathbf{s}$, i.e., $R = 1$. Since the domain size is determined as

$$|\mathcal{T}_1| = \mathcal{O}\binom{n/2}{w/4, w/4, \cdot}$$

the time complexity of Algorithm 1 for van Vredendaal's choice of domains becomes

$$T_{\text{v-V}} = \tilde{\mathcal{O}}\left(|\mathcal{T}_1|^{3/2}/R\right) = \tilde{\mathcal{O}}\left(\binom{n/2}{w/4, w/4, \cdot}^{3/2}\right) = \tilde{\mathcal{O}}\left(2^{3H(\omega/2,\omega/2,\cdot)n/4}\right),$$

where $\omega := w/n$.

May's Instantiations May gives three different instantiations for $\mathcal{T}_i$, called Rep-0, Rep-1 and Rep-2. For all choices the weight of the vectors distributes over the full n coordinates. The difference then lies in the precise choice of weight and the digit set. Let us start with the most simple Rep-0 variant.

Rep-0 *Instantiation.* Here the domains are chosen as

$$\mathcal{T}_1 = \mathcal{T}_2 := \tau^n(w/4),$$

which results in a domain size of

$$|\mathcal{T}_i| = \mathcal{O}\binom{n}{w/4, w/4, \cdot}.$$

Note that when representing $\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$ with $\mathbf{s}_i \in \mathcal{T}_i$, we can obtain a 1 (resp. a -1) coordinate only as $1 + 0$ or $0 + 1$ (resp. $-1 + 0$ or $0 - 1$), while a 0 only as $0 + 0$. Therefore the number of representations amounts to

$$R = \binom{w/2}{w/4}^2.$$

as we can freely choose $w/4$ out of $w/2$ of the ones to be represented as $1 + 0$ while the rest is represented as $0 + 1$ (and analogously for the -1's).

The time complexity is then given as

$$T_{\text{Rep-0}} = \tilde{\mathcal{O}}\left(|\mathcal{T}_i|^{3/2}/R\right) = \tilde{\mathcal{O}}\left(2^{\left(3H(\omega/4,\omega/4,\cdot)/2-\omega\right)n}\right),$$

where again $\omega := w/n$.

Rep-1 *Instantiation* The Rep-1 instantiation increases the weight of the vectors to $w/2 + 2d$ for some small d, that has to be optimized, i.e.,

$$\mathcal{T}_1 = \mathcal{T}_2 := \tau^n(w/4 + d).$$

Similar to before we have

$$|\mathcal{T}_i| = \mathcal{O}\binom{n}{w/4 + d, w/4 + d, \cdot}.$$

The benefit of the increased weight lies in an increased number of representations. As now it is possible to represent a zero coordinate in $\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$ not only as $0 + 0$ but also via $-1 + 1$ and $1 + (-1)$. In total, this leads to

$$R = \binom{w/2}{w/4}^2 \binom{n-w}{d, d, \cdot},$$

as we represent d zeros via $-1 + 1$, d as $1 + (-1)$ and $n - w - 2d$ as $0 + 0$. In total the time complexity of this approach then becomes

$$T_{\text{Rep-1}} = \tilde{\mathcal{O}}\left(|\mathcal{T}_i|^{3/2}/R\right) = \tilde{\mathcal{O}}\left(2^{\left(3H(\omega/4+\delta,\omega/4+\delta,\cdot)/2-\omega-(1-\omega)H\left(\delta/(1-\omega),\delta/(1-\omega),\cdot\right)\right)n}\right),$$

where $d = \delta n$.

Rep-2 *Instantiation* In the Rep-2 instantiation May defines the vectors no longer over $\{-1, 0, 1\}^n$ but over $\{-2, -1, 0, 1, 2\}^n$. Again the additional -2 and 2 entries lead to more representations. However, the analysis becomes quite technical. We give an extended analysis of this representation approach for our nested algorithm in Section 4.3 and an analysis of an extension to Rep-3 in the appendix. For a complexity analysis specific to May's instantiation we refer to [35]. In Fig. 4 we illustrate the runtime exponents of the algorithms by May and van Vredendaal.

4 Nested Collision Search for LWE

So far the collision search algorithm solves the LWE identity only on a projection after applying Odlyzko's hash function. To eventually identify the solution among all candidates that satisfy this less restrictive identity, the collision search procedure is repeated an exponential amount of times. In other words, a brute force technique is applied to isolate the solution.

Our nested collision search procedure now replaces the brute force step by a second collision search. While one might hope that a single collision $(\mathbf{x}, \mathbf{y})$ would then suffice to solve the problem, usually $\mathbf{x}, \mathbf{y}$ do not sum to a ternary vector, i.e., $\mathbf{x} + \mathbf{y} \notin \{-1, 0, 1\}^n$. Therefore the algorithm still needs to iterate over multiple collisions. However, as soon as $\mathbf{x} + \mathbf{y} \in \{-1, 0, 1\}^n$, it implies that $\mathbf{s} = \mathbf{x} + \mathbf{y}$ is the solution.

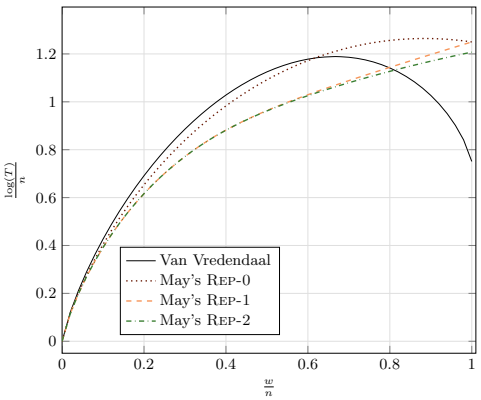


Fig. 4: Comparison between van Vredendaal's instantiation and May's instantiations.

Let us start again with a general framework before discussing our concrete instantiations. For the two-layer approach, we split the solution into four summands $\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2 + \mathbf{s}_3 + \mathbf{s}_4$. This implies

$$\begin{aligned} & \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2 + \mathbf{s}_3 + \mathbf{s}_4) = \mathbf{b} + \mathbf{e} && \bmod q \\ \Leftrightarrow\ & \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2) = \mathbf{b} - \mathbf{A}(\mathbf{s}_3 + \mathbf{s}_4) + \mathbf{e} && \bmod q. \end{aligned}$$

Further, for now we assume that we know the first 2ℓ coordinates of $\mathbf{e}$. Then we obtain

$$\pi_{2\ell}\big(\mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2)\big) = \mathbf{b}' - \pi_{2\ell}\big(\mathbf{A}(\mathbf{s}_3 + \mathbf{s}_4)\big) \mod q, \tag{3}$$

where $\mathbf{b}' := \pi_{2\ell}(\mathbf{b}+\mathbf{e})$ is known. This *layer-2 identity* will later be used to identify $(\mathbf{s}_1, \mathbf{s}_2)$ and $(\mathbf{s}_3, \mathbf{s}_4)$ among a set of candidates. Further let $\mathbf{r} := \pi_\ell\big(\mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2)\big)$ be the lower ℓ coordinates of the left side of this layer-2 identity. Then we obtain our two *layer-1 identities* as

$$\begin{aligned} \pi_\ell(\mathbf{A}\mathbf{s}_1) &= \mathbf{r} - \pi_\ell(\mathbf{A}\mathbf{s}_2) && \bmod q \\ \pi_\ell(\mathbf{A}\mathbf{s}_3) &= \pi_\ell(\mathbf{b}') - \mathbf{r} - \pi_\ell(\mathbf{A}\mathbf{s}_4) && \bmod q. \end{aligned} \tag{4}$$

Now let us define the functions f_1, f_2 and f_3, f_4 used for collision search on layer one, where $f_i \colon \mathcal{T}_i \to \mathbb{Z}_q^\ell$ as

$$f_1, f_3 \colon \mathbf{x} \mapsto \pi_\ell(\mathbf{A}\mathbf{x}), \quad f_2 \colon \mathbf{x} \mapsto \mathbf{r} - \pi_\ell(\mathbf{A}\mathbf{x}) \quad \text{and} \quad f_4 \colon \mathbf{x} \mapsto \pi_\ell(\mathbf{b}') - \mathbf{r} - \pi_\ell(\mathbf{A}\mathbf{x}). \tag{5}$$

Note that the value of $\mathbf{r}$ is not known a priori; hence the algorithm iterates over random choices of $\mathbf{r}$ until it succeeds. By definition any representation $(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4)$ of $\mathbf{s}$ with $\pi_\ell\big(\mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2)\big) = \mathbf{r}$ satisfies the layer-1 (and layer-2) identities and furthermore yields collisions in our functions f_i. Namely $(\mathbf{s}_1, \mathbf{s}_2)$ forms a collision between the functions f_1, f_2, while $(\mathbf{s}_3, \mathbf{s}_4)$ forms a collision in

f_3, f_4. While not every collision is a representation, we can sample candidates for $\mathbf{s}_1, \mathbf{s}_2$ (resp. $\mathbf{s}_3, \mathbf{s}_4$) by finding collisions between f_1, f_2 (resp. f_3, f_4).

Every collision, regardless of being a representation or not, already fulfills one of the layer-1 identities (Eq. (4)) (depending if the collision is between f_1, f_2 or f_3, f_4). Furthermore, note that any tuple $(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4)$ where $(\mathbf{y}_1, \mathbf{y}_2)$ is a collision in f_1, f_2 and $(\mathbf{y}_3, \mathbf{y}_4)$ a collision in f_3, f_4, already fulfills the layer-2 identity (Eq. (3)) on the lower ℓ coordinates. Therefore just consider the summation of both layer-1 identities from Eq. (4).

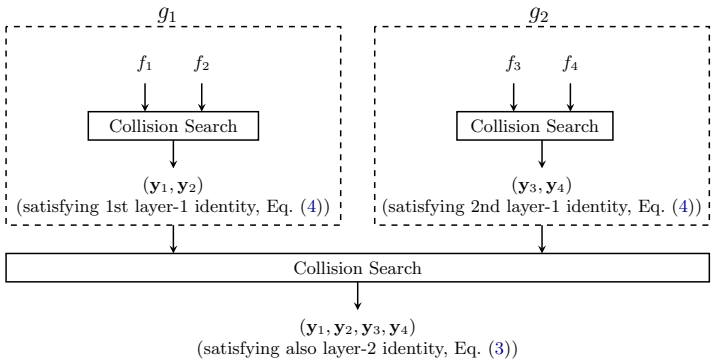


Fig. 5: Schematic illustration of multiple-layer collision search.

We now apply a second collision search to identify those pairs of collisions that jointly satisfy the layer-2 identity on all 2ℓ coordinates. This process is illustrated in Fig. 5.

Let $\vartheta_\ell \colon \mathbb{Z}_q^k \to \mathbb{Z}_q^\ell$, $k \geq 2\ell$ be the projection to the coordinates of the vector indexed by $\ell + 1$ to 2ℓ, i.e., for $\mathbf{x} = (x_1, \ldots, x_k)$ we let $\vartheta_\ell(\mathbf{x}) := (x_{\ell+1}, \ldots, x_{2\ell})$. Now we are ready to define the *second layer functions* $g_i \colon \mathbb{Z}_q^\ell \to \mathbb{Z}_q^\ell$, $i = 1, 2$. These functions take as input a starting point of a collision search procedure between the layer-1 functions f_{2i-1}, f_{2i} and compute the colliding entries $\mathbf{y}_{2i-1}, \mathbf{y}_{2i}$ reachable from that starting point. Finally they output the upper ℓ coordinates of the corresponding value of the layer-2 identity for $(\mathbf{y}_{2i-1}, \mathbf{y}_{2i})$. More formally, we have

$$\begin{aligned} g_1 \colon \mathbf{x} &\mapsto \vartheta_\ell(\mathbf{A}(\mathbf{y}_1 + \mathbf{y}_2)) \quad , \text{ where } (\mathbf{y}_1, \mathbf{y}_2) = \textsc{Rho}(f_1^{[\mathbf{x}]}, f_2^{[\mathbf{x}]}, \mathbf{x}) \quad \text{and} \\ g_2 \colon \mathbf{x} &\mapsto \vartheta_\ell(\mathbf{b}') - \vartheta_\ell(\mathbf{A}(\mathbf{y}_3 + \mathbf{y}_4)), \text{ where } (\mathbf{y}_3, \mathbf{y}_4) = \textsc{Rho}(f_3^{[\mathbf{x}]}, f_4^{[\mathbf{x}]}, \mathbf{x}). \end{aligned} \tag{6}$$

Note that here we flavour the inner functions f_i deterministically via the starting point used for collision search (see Definition 2), similar to [16, 20]. In this way g_1, g_2 stay deterministic, as required for the general collision search procedure, while we obtain (heuristic) independence of returned collisions from the inner functions.

The general algorithm is outlined in Algorithm 2 as pseudocode and visually illustrated in Fig. 6. The smaller *Rho*-structures in the figure represent the layer-1 collision search, while the layer-2 search is formed as a big *Rho* using multiple layer-1 collision searches.

Algorithm 2: NESTED-COLLISION-SEARCH

Input: $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n$, positive integer $w \leq n$
Output: $\mathbf{s} \in \tau^n(w/2)$ such that $\mathbf{e} = \mathbf{As} - \mathbf{b} \mod q \in \{-1, 0, 1\}^n$

Let f_i and g_j be as defined in Eqs. (5) and (6)
$\ell := \frac{\log_q |\tau^n(w/2)|}{2}$
repeat
  Choose random permutation $\mathbf{P}$, $\mathbf{A}' \leftarrow \mathbf{AP}$
  Choose $\mathbf{e}' \in \{-1, 0, 1\}^{2\ell}$ randomly
  $\mathbf{b}' \leftarrow \pi_{2\ell}(\mathbf{b}) + \mathbf{e}'$
  Choose $\mathbf{r}, \mathbf{z} \in \mathbb{Z}_q^\ell$ randomly
  Define functions as in Eqs. (5) and (6) based on $\mathbf{A}', \mathbf{b}'$ and $\mathbf{r}$
  Choose random flavour for g_1, g_2
  $(\mathbf{z_1}, \mathbf{z_2}) \leftarrow \text{RHO}(g_1, g_2, \mathbf{z})$
  Compute $(\mathbf{y_1}, \mathbf{y_2}) = \text{RHO}(f_1, f_2, \mathbf{z_1})$
  Compute $(\mathbf{y_3}, \mathbf{y_4}) = \text{RHO}(f_3, f_4, \mathbf{z_2})$
  Set $\mathbf{s}' = \mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3 + \mathbf{y}_4$
until $\mathbf{s}' \in \tau^n(w/2)$
return $\mathbf{Ps}'$

4.1 Analysis of Nested Collision Search

Correctness First note the permuted instance defined by $\mathbf{A}' = \mathbf{AP}$ has solution $\mathbf{s}' = \mathbf{P}^{-1}\mathbf{s}$. Hence, once this solution is found we have to return $\mathbf{s} = \mathbf{Ps}'$.

We have already shown, that any representation $(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4)$ of the solution $\mathbf{s}$ for the correct choice of $\mathbf{r} = \pi_\ell\big(\mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2)\big)$ and the correct guess for $\mathbf{e}' = \pi_{2\ell}(\mathbf{e})$ satisfies the layer-1 and layer-2 identities (compare to Eq. (3) and Eq. (4)). Further, we know that such a representation forms a collision in g_1, g_2. Therefore by sampling independent and uniformly random collisions between g_1 and g_2 we can find $\mathbf{s}$, given there exist at least one representation (which will be ensured by the choice of $\mathcal{T}_i$ later). Again we obtain heuristic independence of the sampled collisions by the choice of random flavors in each iteration.

It remains to show that after finding a collision $(\mathbf{x}_1, \mathbf{x}_2)$ in g_1, g_2 for which the value $\mathbf{s}' = \mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3 + \mathbf{y}_4 \in \tau^n(w/2)$, i.e., $\mathbf{s}'$ is a ternary vector of weight w, it suffices to conclude that $\mathbf{s}'$ is a solution. Therefore note that the expected number of elements from $\tau^n(w/2)$ that fulfill the layer-2 identity is by the randomness of $\mathbf{A}$

$$\frac{|\tau^n(w/2)|}{q^{2\ell}} = 1,$$

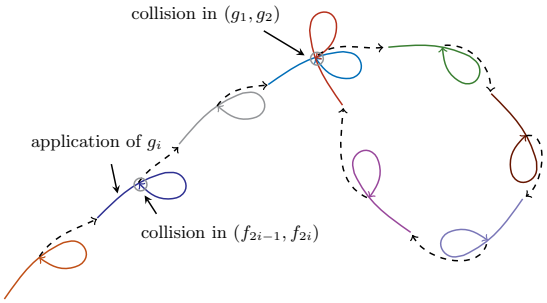


Fig. 6: Illustration of the nested collision search. Different colors identify different function flavors. Dashed arrows indicate mapping from collisions to starting points.

since we choose $\ell = \frac{\log_q |\tau^n(w/2)|}{2}$. Hence, once such an element is found, we conclude that it is $\mathbf{s}$. This proves correctness under the same heuristic used by the algorithms based on conventional collision search (see Remark 2).

Note that the specific choice of ℓ implies that the range of all functions is of size $q^\ell = \sqrt{|\tau^n(w/2)|}$. Hence, to allow for collision search, we have to ensure

$$|\mathcal{T}_i| \stackrel{!}{=} q^\ell = \sqrt{|\tau^n(w/2)|} \tag{7}$$

by our choice of function domains $\mathcal{T}_i$.

Complexity For a representation $(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4)$ of $\mathbf{s}$ with $\mathbf{s}_i \in \mathcal{T}_i$ let

$$\mathbf{s} = \underbrace{\mathbf{s}_1 + \mathbf{s}_2}_{\mathbf{a}_1} + \underbrace{\mathbf{s}_3 + \mathbf{s}_4}_{\mathbf{a}_2}. \tag{8}$$

In our analysis we consider only those representations where $\mathbf{a}_i \in \mathcal{D}_i$ for some set $\mathcal{D}_i$, which we refer to as *mid-level domains*.[6] Let us assume that there exist R_2 different representations $(\mathbf{a}_1, \mathbf{a}_2) \in \mathcal{D}_1 \times \mathcal{D}_2$ of the solution $\mathbf{s}$. Further assume that any such $\mathbf{a}_1$ (analogously any such $\mathbf{a}_2$) has R_1 representations $(\mathbf{s}_1, \mathbf{s}_2) \in \mathcal{T}_1 \times \mathcal{T}_2$ (analogously $(\mathbf{s}_3, \mathbf{s}_4) \in \mathcal{T}_3 \times \mathcal{T}_4$).

Consider one iteration of Algorithm 2. We denote by $E_\mathbf{r}$ the event that there exists a representation $(\mathbf{a}_1, \mathbf{a}_2)$ of $\mathbf{s}$ for the choice of $\mathbf{r}$ made in line 7, i.e., a representation with $\pi_\ell(\mathbf{A}\mathbf{a}_1) = \mathbf{r}$. The event of guessing $\pi_{2\ell}(\mathbf{e})$ correctly we denote by $E_\mathbf{e}$. Eventually, we denote the event that the tuple $(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4)$ obtained in line 11 and 12 is a representation of $\mathbf{s}$ by $E_\mathbf{s}$. Then we expect

$$\Pr[E_\mathbf{e} \cap E_\mathbf{r} \cap E_\mathbf{s}]^{-1} = \left(\Pr[E_\mathbf{e}] \cdot \Pr[E_\mathbf{r} \mid E_\mathbf{e}] \cdot \Pr[E_\mathbf{s} \mid E_\mathbf{e} \cap E_\mathbf{r}]\right)^{-1}$$

iterations of the loop until success.

[6] The concrete choice of $\mathcal{D}_i$, similar to the function domains $\mathcal{T}_i$, depends on the instantiation and is specified later.

The probability of guessing the correct $\mathbf{e}'$ in line 5 of Algorithm 2 is $q_{\mathbf{e}} = 3^{-2\ell} = 3^{-\log_q |\tau^n(w/2)|}$. Since $q = \text{poly}(n)$ and $|\tau^n(w/2)| = 2^{cn}$ for some constant c, it follows that

$$q_3 := \Pr[E_{\mathbf{e}}] = 3^{-2\ell} = 2^{-\Theta(\frac{n}{\log n})}.$$

Further, by the randomness of $\mathbf{A}$, we have

$$q_2 := \Pr[E_{\mathbf{r}} \mid E_{\mathbf{e}}] = \frac{R_2}{q^\ell}.$$

Now given $E_{\mathbf{e}} \cap E_{\mathbf{r}}$ there exists a representation $(\mathbf{a}_1, \mathbf{a}_2)$. As both, $\mathbf{a}_1$ and $\mathbf{a}_2$, have R_1 different representations $(\mathbf{s}_1, \mathbf{s}_2)$ and $(\mathbf{s}_3, \mathbf{s}_4)$, we find a total of $(R_1)^2$ pairs of representations that together lead to $\mathbf{a}_1, \mathbf{a}_2$. Recall that each such pair fulfills the layer-1 and layer-2 identities and, hence, forms a collision between the functions g_1, g_2. Therefore, a random collision in the functions g_1, g_2 leads to $\mathbf{s}$ with probability

$$q_1 := \Pr[E_{\mathbf{s}} \mid E_{\mathbf{e}} \cap E_{\mathbf{r}}] = \frac{(R_1)^2}{q^\ell},$$

as by Remark 2 there exist a total of q^ℓ collisions between g_1 and g_2.

Eventually, the time per iteration of the loop is dominated by the collision search between g_1 and g_2. This collision search requires $\mathcal{O}(q^{\frac{\ell}{2}})$ evaluations of those functions. Now for each evaluation a collision search between f_1, f_2 (resp. f_3, f_4) with time complexity $\tilde{\mathcal{O}}\left(q^{\frac{\ell}{2}}\right)$ is performed. Hence the time per iteration is $\tilde{\mathcal{O}}\left(q^{\frac{\ell}{2}} \cdot q^{\frac{\ell}{2}}\right) = \tilde{\mathcal{O}}\left(q^\ell\right)$.

Overall this leads to time complexity

$$T = (q_1 q_2 q_3)^{-1} \cdot q^\ell = \left(\frac{|\tau^n(w/2)|^{\frac{3}{2}}}{(R_1)^2 \cdot R_2}\right)^{1+o(1)} = \left(\frac{\binom{n}{w/2,w/2,.}^{\frac{3}{2}}}{(R_1)^2 \cdot R_2}\right)^{1+o(1)}. \tag{9}$$

Remark 3. Note that the heuristic specified in Remark 2 must fail if there are significantly more collisions between the constructed functions than there would be between random functions. Precisely, this is the case if $(R_1)^2 > q^\ell$, since there are $(R_1)^2$ collisions caused by representations in the second layer functions, while for random functions we would expect q^ℓ collisions. However, we actively prevent this due to an appropriate choice of function domains ensuring $R_1 < q^{\frac{\ell}{2}}$.

A different analysis approach. Another way to derive the time complexity of Algorithm 2 is via directly computing the probability that the sampled tuple $(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4)$ sums to a ternary vector. We provide this alternative analysis in Appendix A.

Use of Odlyzko's hash function. Our construction does not rely on Odlyzko's hash function but instead guesses 2ℓ coordinates of $\mathbf{e}$ to obtain an exact identity on these coordinates. For the first layer this is necessary to ensure that any pair of collisions between f_1, f_2 and f_3, f_4 jointly satisfy the layer-2 identity on the lower ℓ coordinates. This is because the exact identities in contrast to Odlyzko's hash

function are additive, i.e., adding both identities from Eq. (4) results in a valid identity. Note that, for the second layer, we could apply Odlyzko's hash function rather than relying on the exact identity on the subsequent ℓ coordinates. Then guessing ℓ rather than 2ℓ bits of $\mathbf{e}$ would suffice. However, as this only improves second order terms we decided for ease of exposition to not rely on Odlyzko's hash function at all.

4.2 Concrete Instantiations

Next we give a first concrete instantiation for Algorithm 2, i.e., we specify the choice of function domains $\mathcal{T}_i$ and the mid level domains $\mathcal{D}_i$. We start with a choice of domains representing ternary vectors analogously to the Rep-1 instantiation given in Section 3.

Nested-1 Instantiation Recall that for the nested collision search besides the function domains $\mathcal{T}_i$ we have to specify the sums we aim to obtain on the middle level, i.e., the mid-level domains D_i of the $\mathbf{a}_i$ from Eq. (8). We consider for the D_i ternary vectors of length n with balanced weight $p_2 := w/4 + d_2$, where d_2 is an optimization parameter.

The function domains $\mathcal{T}_i$ are then chosen as all ternary vectors of length n and balanced weight $p_1 := p_2/2 + d_1 = w/8 + d_2/2 + d_1$, where d_1 has again to be optimized. In summary, we have

$$\mathcal{D}_i := \tau^n(p_2) \quad \text{and} \quad \mathcal{T}_i := \tau^n(p_1)$$

This gives function domains of size

$$|\mathcal{T}_i| = \binom{n}{p_1, p_1, \cdot}.$$

Let us now determine the number of representations R_1, R_2. Recall that R_2 is the amount of different $(\mathbf{a}_1, \mathbf{a}_2) \in D_1 \times D_2$ that sums to the solution $\mathbf{s}$. Hence, we have

$$R_2 = \binom{w/2}{w/4}^2 \binom{n-w}{d_2, d_2, \cdot},$$

as $\mathbf{s} \in \tau^n(w/2)$. Furthermore, each element of $\mathbf{a}_1$ respectively $\mathbf{a}_2$ has

$$R_1 = \binom{p_2}{p_2/2}^2 \binom{n-2p_2}{d_1, d_1, \cdot}$$

representations as the sum of elements from $\mathcal{T}_i$.

Now plugging R_1 and R_2 into Eq. (9) gives the running time $T_{\text{Nested-1}}$ of this instantiation.

To obtain the runtime exponent c in $T_{\text{Nested-1}} = 2^{cn}$, we again approximate the involved binomial and multinomial coefficients via Eq. (1). Further we model $d_1 = \delta_1 n$ and $d_2 = \delta_2 n$ for $\delta_i \in [0, 1]$. Eventually we obtain c by minimizing over

the choice of δ_1, δ_2 under the constraint on the function domain's size given in Eq. (7). For this minimization we use a numerical optimizer provided by the *scipy* python library, inspired by the code used for numerical optimization in [9]. The code used to run the numerical optimization for all our algorithms is available at https://github.com/arindamIITM/Small-LWE-Keys.

Remark 4 (Optimization Accuracy). In general these kinds of numerical optimizers do not guarantee to find a global minimum, but instead might return only a local minimum or miss optimal parameters slightly. However, to increase the confidence in the optimality of the returned value, we minimized over thousands of runs of the optimizer on random starting points and multiple different formulations of the problem, until no further improvement could be obtained.

Note that for $w \geq 0.64$ even for $d_1 = d_2 = 0$, which minimizes the function domains we have $|\mathcal{T}_i| > \sqrt{|\tau^n(w/2)|}$. Therefore we do not obtain further instantiations as we can not satisfy Eq. (7). In the following, we make use of the concept of partial representations to allow for an adaptive scaling of the function domain size.

Nested-1$^+$ Instantiation We now split the vectors of the domains into two parts, a disjoint part of length $(1-\gamma)n$ and a joint part of length γn (compare to Fig. 7).

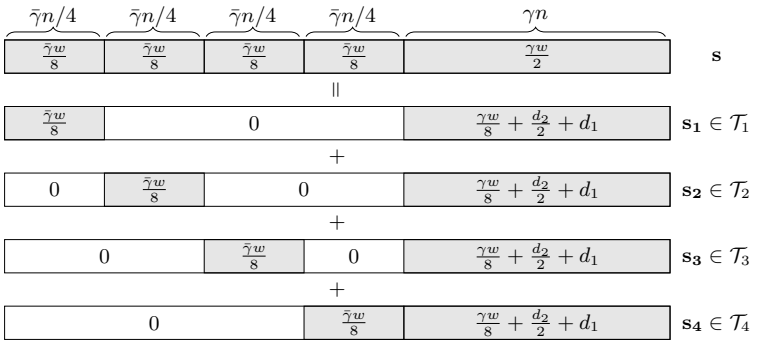


Fig. 7: Weight distribution of function domains using partial representations. Gray areas indicate regions of fixed balanced-ternary weight, where $\bar{\gamma} := 1 - \gamma$

Precisely, for $\gamma \in [0, 1]$ we define the function domains $\mathcal{T}_i$ as

$$\begin{aligned}
\mathcal{T}_1 &= \tau^{\bar{\gamma}n/4}(\bar{\gamma}w/8) \times \mathbf{0} \times \mathbf{0} \times \mathbf{0} \times \tau^{\gamma n}(p_1),\\
\mathcal{T}_2 &= \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\bar{\gamma}w/8) \times \mathbf{0} \times \mathbf{0} \times \tau^{\gamma n}(p_1),\\
\mathcal{T}_3 &= \mathbf{0} \times \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\bar{\gamma}w/8) \times \mathbf{0} \times \tau^{\gamma n}(p_1),\\
\mathcal{T}_4 &= \mathbf{0} \times \mathbf{0} \times \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\bar{\gamma}w/8) \times \tau^{\gamma n}(p_1),
\end{aligned}$$

where $\bar{\gamma} = 1 - \gamma$ and $p_1 := \gamma w/8 + d_2/2 + d_1$. This gives function domain sizes of

$$|\mathcal{T}_i| = \binom{\bar{\gamma}n/4}{\bar{\gamma}w/8, \bar{\gamma}w/8, \cdot}\binom{\gamma n}{p_1, p_1, \cdot}.$$

Analogously, to the previous instantiation we define the domains $\mathcal{D}_i$ on the middle level as

$$\begin{aligned}\mathcal{D}_1 &= \tau^{\bar{\gamma}n/4}(\bar{\gamma}w/8) \times \tau^{\bar{\gamma}n/4}(\bar{\gamma}w/8) \times \quad \mathbf{0} \quad \times \quad \mathbf{0} \quad \times \tau^{\gamma n}(p_2),\\ \mathcal{D}_2 &= \quad \mathbf{0} \quad \times \quad \mathbf{0} \quad \times \tau^{\bar{\gamma}n/4}(\bar{\gamma}w/8) \times \tau^{\bar{\gamma}n/4}(\bar{\gamma}w/8) \times \tau^{\gamma n}(p_2),\end{aligned}$$

where $p_2 = \gamma w/4 + d_2$.

To be able to construct the solution, we assume that on all five parts the weight of the solution is distributed proportionally. This can be achieved by the permutation in line 4 of Algorithm 2. Again, as for the van Vredendaal instantiation from Section 3, this causes only a small polynomial overhead.

Observe that as before we hope that on the jointly enumerated part (now of size γn) the vectors of weight p_1 add up to weight p_2. Further recall, that on the disjoint weight part of length $\bar{\gamma}n = (1-\gamma)n$ we have only a single representation of any element from $(\tau^{\bar{\gamma}n}(\bar{\gamma}w/8))^4$. Hence, the number of representations is similar as before, but takes into account the reduced length of γn, where representations exist. For representations from the middle level we get

$$R_2 = \binom{\gamma w/2}{\gamma w/4}^2 \binom{\gamma(n-w)}{d_2, d_2, \cdot},$$

while every element on the middle level has $R_1 = \binom{p_2}{p_2/2}^2\binom{\gamma n - 2p_2}{d_1, d_1, \cdot}$ many representations.

Similar as before we obtain the running time $T_{\text{Nested-1}^+}$ of this instantiation using Eq. (9). Again, we obtain the runtime exponent c by approximating the multinomial coefficients, letting $d_1 = \delta_1 n, d_2 = \delta_2 n$ and finally minimizing over the choice of δ_1, δ_2 and γ. The obtained runtime exponents of both our instantiations Nested-1 and Nested-1^+ are given in Fig. 8 in comparison to the exponents of van Vredendaal's as well as May's Rep-2 instantiation of the basic collision search. We observe that Nested-1^+ significantly outperforms all other instantiations for almost all choices of the weight w. Only for a weight w close to n, i.e. w/n close to one, van Vredendaal's algorithm offers a slightly better running time. In comparison to May's representation based instantiations our nested approach has the natural property that for large weights, with decreased search space size, the running time also decreases again.

We also observe that Nested-1^+ not only extends Nested-1 to weights $w/n > 0.64$, it also offers runtime improvements in the regime $w/n \geq 0.44$. This value of $w/n = 0.44$ marks the point from where the γ-parameter of the Nested-1^+ instantiation is chosen smaller than one to fulfill the correctness constraint from Eq. (7). The ability to control the domain sizes by γ instead of having to decrease the representation parameters d_1 and d_2 results in the superiority of Nested-1^+ over Nested-1 in this regime.

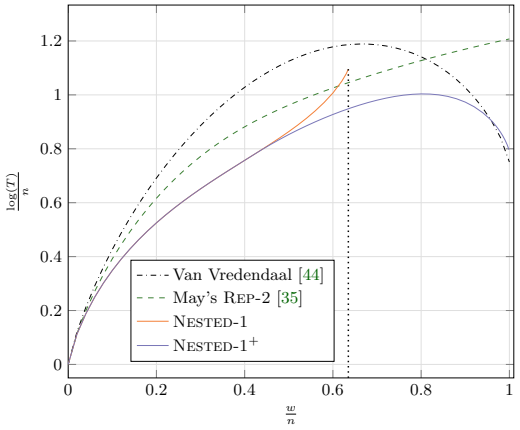


Fig. 8: Runtime exponents of Nested-1 and Nested-1$^+$ instantiations compared to previous work.

4.3 Exploiting the Permutation

Next, we show how to improve the algorithm by aiming at a non-proportional weight distribution induced by the permutation. Then we give two further instantiations for the function domains one based on Rep-1-like representations and one exploiting the Rep-2 concept.

Recall that by our choice of function domains (see Eq. (7)), as soon as we find a collision between the second-layer functions g_i, that leads to an $\mathbf{s}' \in \tau^n(w/2)$ it implies that $\mathbf{s}'$ is a solution. In our previous instantiation Nested-1$^+$, we introduced a disjoint weight part, which automatically leads to elements of the desired form on a $(1-\gamma)$ fraction of the coordinates. In other words a collision between g_1 and g_2 leading to an $\mathbf{s}' \notin \tau^n(w/2)$ is always caused by the coordinates in the jointly enumerated part not adding up as desired.

The idea is now to exploit the permutation to distribute a higher fraction of the weight on the disjoint part in the solution $\mathbf{P}^{-1}\mathbf{s}$ of the permuted instance. Since, in turn the decreased weight on the joint part increases the probability that elements add up to ternary vectors, as desired.

More precisely, instead of obtaining the proportional ternary weight of γw on the γn-part and $(1-\gamma)w/4$ in each of the four disjoint parts we aim at weight $\beta\gamma w$ on the joint part and $(1-\beta\gamma)w/4$ on the disjoint parts for some positive $\beta \in [\frac{w-(1-\gamma)n}{\gamma w}, 1]$. The lower bound on β just ensures that the length of the disjoint parts is larger or equal to the weight, i.e., $(1-\beta\gamma)w/4 \leq (1-\gamma)n/4$. Note that once we assume the solution $\mathbf{s}' = \mathbf{P}^{-1}\mathbf{s}$ to the permuted instance in this form, the search space changes from $\tau^n(w/2)$ to

$$D := \left(\tau^{\bar{\gamma}n/4}\big((1-\beta\gamma)w/8\big)\right)^4 \times \tau^{\gamma n}(\beta\gamma w/2),$$

where $\bar{\gamma} := 1-\gamma$. This means the size of the search space reduces to

$$|D| = \binom{\bar{\gamma}n/4}{(1-\gamma\beta)w/8, (1-\gamma\beta)w/8, \cdot}^4 \binom{\gamma n}{\beta\gamma w/2, \beta\gamma w/2, \cdot}.$$

In turn the expected amount of elements from D that satisfy the second-layer identity Eq. (3) is $\frac{|D|}{q^{2\ell}}$. Hence, to guarantee that there exists only one such element in expectation we have to choose $\ell = \frac{\log_q |D|}{2}$. In other words, the constraint from Eq. (7) now changes to

$$|\mathcal{T}_i| \stackrel{!}{=} \sqrt{|D|}. \tag{10}$$

While the analysis from Section 4.1 in principle still holds, we need to account for the probability of the weight being distributed as desired. Note that this probability can be expressed as

$$q_4 := \Pr[\mathbf{P}^{-1}\mathbf{s} \in D] = \frac{|D|}{|\tau^n(w/2)|}.$$

Hence, in total the algorithm needs to be iterated q_4^{-1} times more often. Together with the changed value of ℓ we obtain (compare to Eq. (9))

$$T = (q_1 q_2 q_3 q_4)^{-1} q^\ell = \left(\frac{|D|^{\frac{1}{2}} \cdot |\tau^n(w/2)|}{(R_1)^2 R_2}\right)^{1+o(1)}. \tag{11}$$

Nested-1* Instantiation Let us first consider an instantiation using again the Rep-1 concept for representations. We now choose according to the changed weight distribution adapted function domains as shown in Fig. 9.

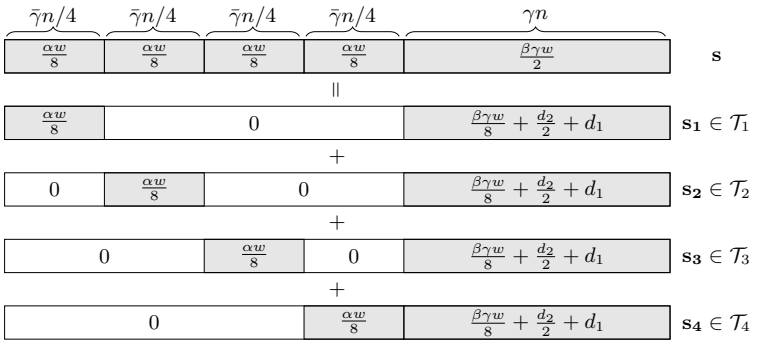


Fig. 9: Weight distribution of function domains for Nested-1* instantiation. Gray regions are of fixed balanced-ternary weight, with $\alpha := 1 - \beta\gamma$

More formally, we let

$$\begin{aligned}
\mathcal{T}_1 &= \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \mathbf{0} \times \mathbf{0} \times \mathbf{0} \times \tau^{\gamma n}(p_1),\\
\mathcal{T}_2 &= \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \mathbf{0} \times \mathbf{0} \times \tau^{\gamma n}(p_1),\\
\mathcal{T}_3 &= \mathbf{0} \times \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \mathbf{0} \times \tau^{\gamma n}(p_1),\\
\mathcal{T}_4 &= \mathbf{0} \times \mathbf{0} \times \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \tau^{\gamma n}(p_1),
\end{aligned}$$

where $\bar{\gamma} := 1-\gamma$, $\alpha := (1-\beta\gamma)$ and $p_1 := \beta\gamma w/8 + d_2/2 + d_1$. This gives function domain sizes of

$$|\mathcal{T}_i| = \binom{\bar{\gamma}n/4}{\alpha w/8, \alpha w/8, \cdot}\binom{\gamma n}{p_1, p_1, \cdot}.$$

Accordingly, we adjust the mid-level domains to

$$\begin{aligned}
\mathcal{D}_1 &= \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \mathbf{0} \times \mathbf{0} \times \tau^{\gamma n}(p_2),\\
\mathcal{D}_2 &= \mathbf{0} \times \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \tau^{\gamma n}(p_2),
\end{aligned}$$

with $p_2 := \beta\gamma w/4 + d_2$ In turn this leads to an amount of

$$R_2 = \binom{\beta\gamma w/2}{\beta\gamma w/4}^2\binom{\gamma(n-\beta w)}{d_2, d_2, \cdot},$$

representations of the solution as sum of elements from $\mathcal{D}_1, \mathcal{D}_2$. Furthermore, every element from $\mathcal{D}_1$ (resp. $\mathcal{D}_2$) as sum of elements from $\mathcal{T}_1, \mathcal{T}_2$ (resp. $\mathcal{T}_3, \mathcal{T}_4$) has

$$R_1 = \binom{p_2}{p_2/2}^2\binom{\gamma n - 2p_2}{d_1, d_1, \cdot}$$

representations.

We now obtain the running time $T^*_{\text{Nested-1}}$ via Eq. (11). As before we approximate the multinomial coefficients via Eq. (1) and perform a numerical optimization to obtain the runtime exponent c in $T^*_{\text{Nested-1}} = 2^{cn}$. Here, we minimize c over the choice of β, γ, δ_1 and δ_2, where $d_1 = \delta_1 n$ and $d_2 = \delta_2 n$, while ensuring the constraint given in Eq. (10).

Nested-2* Instantiation Eventually, we provide an instantiation using Rep-2 like representations, i.e., function and mid level domains whose vectors have coordinates in $\{-2,-1,0,1,2\}$ (see Fig. 10). This increases the number of representations at the cost of quite technical analysis. While in principle it is possible to extend the digit set further, previous results on subset sum [9] and LWE [35] indicate that the runtime quickly converges. For the formal definition of our function domains, let us first extend the definition of $\tau^n(\cdot)$ to $\tau_2^n(a,b) := \{\mathbf{x} \in \{\pm 2, \pm 1, 0\}^n \colon |\mathbf{x}|_1 = |\mathbf{x}|_{-1} = a \wedge |\mathbf{x}|_2 = |\mathbf{x}|_{-2} = b\}$, where $|\mathbf{x}|_i := |\{j \mid x_j = i\}|$.

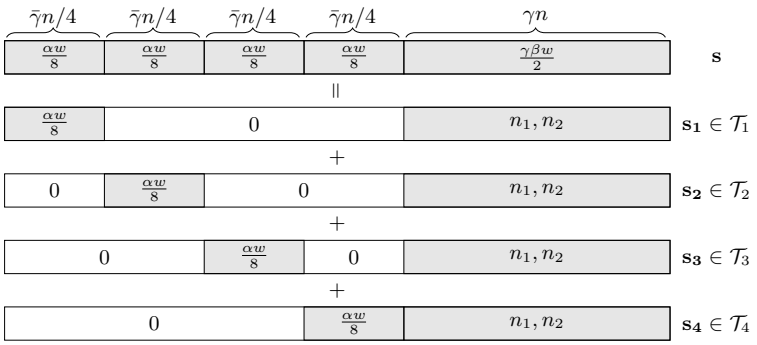


Fig. 10: Weight distribution of function domains for NESTED-2^* instantiation. Gray regions with single numbers indicate parts with fixed balanced-ternary weight, where $\alpha := 1 - \beta\gamma$. Gray parts with two numbers n_1, n_2 contain n_1 1s, n_1 -1s, n_2 2s, n_2 -2s and rest zeros.

The function domains are then defined as

$$\begin{aligned}
\mathcal{T}_1 &= \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \mathbf{0} \times \mathbf{0} \times \mathbf{0} \times \tau_2^{\gamma n}(n_1, n_2),\\
\mathcal{T}_2 &= \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \mathbf{0} \times \mathbf{0} \times \tau_2^{\gamma n}(n_1, n_2),\\
\mathcal{T}_3 &= \mathbf{0} \times \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \mathbf{0} \times \tau_2^{\gamma n}(n_1, n_2),\\
\mathcal{T}_4 &= \mathbf{0} \times \mathbf{0} \times \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \tau_2^{\gamma n}(n_1, n_2),
\end{aligned}$$

where $\bar{\gamma} := 1 - \gamma$ and $\alpha := (1 - \beta\gamma)$, while we derive the precise form of n_1 and n_2 later. This gives function domain sizes of

$$|\mathcal{T}_i| = \binom{\bar{\gamma}n/4}{\alpha w/8, \alpha w/8, \cdot}\binom{\gamma n}{n_1, n_1, n_2, n_2, \cdot}.$$

Accordingly, we adjust the mid-level domains to

$$\begin{aligned}
\mathcal{D}_1 &= \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \mathbf{0} \times \mathbf{0} \times \tau_2^{\gamma n}(n_1^{\mathrm{mid}}, n_2^{\mathrm{mid}}),\\
\mathcal{D}_2 &= \mathbf{0} \times \mathbf{0} \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \tau^{\bar{\gamma}n/4}(\alpha w/8) \times \tau_2^{\gamma n}(n_1^{\mathrm{mid}}, n_2^{\mathrm{mid}}),
\end{aligned}$$

while again we postpone determining $n_1^{\mathrm{mid}}, n_2^{\mathrm{mid}}$ to the analysis of the number of representations.

Let us start by determining the number of representations of the ternary weight-ω solution $\mathbf{s}$ as sum of elements from $\mathcal{D}_1, \mathcal{D}_2$. Recall that we only have representations on the last γn coordinates, where we assume $\mathbf{s}$ to have weight $\hat{w} := \gamma\beta w$. To represent a $-1, 0$ or 1 of the solution we have the following

possibilities

$$\begin{aligned}
0: \quad & \underbrace{0+0}_{m^{\mathrm{mid}}}, \quad \underbrace{1-1}_{z_1^{\mathrm{mid}}}, \quad \underbrace{-1+1}_{z_1^{\mathrm{mid}}}, \quad \underbrace{2-2}_{z_2^{\mathrm{mid}}}, \quad \underbrace{-2+2}_{z_2^{\mathrm{mid}}}, \\
1: \quad & \underbrace{1+0}_{\frac{\hat{w}}{4}-o^{\mathrm{mid}}}, \quad \underbrace{0+1}_{\frac{\hat{w}}{4}-o^{\mathrm{mid}}}, \quad \underbrace{2-1}_{o^{\mathrm{mid}}}, \quad \underbrace{-1+2}_{o^{\mathrm{mid}}}, \\
-1: \quad & \underbrace{-1+0}_{\frac{\hat{w}}{4}-o^{\mathrm{mid}}}, \quad \underbrace{0-1}_{\frac{\hat{w}}{4}-o^{\mathrm{mid}}}, \quad \underbrace{-2+1}_{o^{\mathrm{mid}}}, \quad \underbrace{1-2}_{o^{\mathrm{mid}}},
\end{aligned} \tag{12}$$

where we let $m^{\mathrm{mid}} := \gamma n - \hat{w} - 2z_1^{\mathrm{mid}} - 2z_2^{\mathrm{mid}}$. The number below the corresponding representation denotes how often we expect this representation to appear among all representations of -1, 0 and 1 coordinates. Therefore note that as required the total number of 1 and -1 entries, i.e., the sum over the number of the corresponding row, adds up to $\hat{w}/2$ and the number of 0 entries to $\gamma n - \hat{w}$. After we have specified how often the respective events occur, we can directly derive the number of representations as

$$R_2 = \binom{\gamma n - \hat{w}}{m^{\mathrm{mid}}, z_1^{\mathrm{mid}}, z_1^{\mathrm{mid}}, z_2^{\mathrm{mid}}, z_2^{\mathrm{mid}}} \binom{\hat{w}/2}{\hat{w}/4 - o^{\mathrm{mid}}, \hat{w}/4 - o^{\mathrm{mid}}, o^{\mathrm{mid}}, o^{\mathrm{mid}}}^2,$$

where the first factor counts the possibilities to represent 0s and the second those to represent ± 1s. Now a simple counting argument yields the previously omitted number of coordinates equal to ± 1s and ± 2s in the mid level domains as[7]

$$n_1^{\mathrm{mid}} = z_1^{\mathrm{mid}} + \hat{w}/4 - o^{\mathrm{mid}} + o^{\mathrm{mid}} = \hat{w}/4 + z_1^{\mathrm{mid}} \quad \text{and} \quad n_2^{\mathrm{mid}} = z_2^{\mathrm{mid}} + o^{\mathrm{mid}},$$

where $z_1^{\mathrm{mid}}, z_2^{\mathrm{mid}}$ and o^{mid} are subject to optimization. Note that for $\gamma = \beta = 1$ we obtain as a special case the necessary representation formula for the Rep-2 instantiation of May, which we omitted previously (see Section 3).

Next let us determine the number of representations of any element from the mid-level domains $\mathcal{D}_i$ as sum of elements from the function domains $\mathcal{T}_i$. Therefore let us again specify the number of representations, which is similar to before, but

[7] We have to count the appearances of 1 (resp. 2) entries on the left (or right) of the possible representations given in Eq. (12)

we additionally get multiple possibilities to represent 2 and -2 entries

$$\begin{array}{rlllll}
0: & \underbrace{0+0}_{m}, & \underbrace{1-1}_{z_1}, & \underbrace{-1+1}_{z_1}, & \underbrace{2-2}_{z_2}, & \underbrace{-2+2}_{z_2}, \\
1: & \underbrace{1+0}_{\frac{n_1^{\mathrm{mid}}}{2}-o}, & \underbrace{0+1}_{\frac{n_1^{\mathrm{mid}}}{2}-o}, & \underbrace{2-1}_{o}, & \underbrace{-1+2}_{o}, & \\
-1: & \underbrace{-1+0}_{\frac{n_1^{\mathrm{mid}}}{2}-o}, & \underbrace{0-1}_{\frac{n_1^{\mathrm{mid}}}{2}-o}, & \underbrace{-2+1}_{o}, & \underbrace{1-2}_{o}, & \\
2: & \underbrace{2+0}_{\frac{n_2^{\mathrm{mid}}-t}{2}}, & \underbrace{0+2}_{\frac{n_2^{\mathrm{mid}}-t}{2}}, & \underbrace{1+1}_{t}, & & \\
-2: & \underbrace{-2+0}_{\frac{n_2^{\mathrm{mid}}-t}{2}}, & \underbrace{0-2}_{\frac{n_2^{\mathrm{mid}}-t}{2}}, & \underbrace{-1-1}_{t}, & &
\end{array} \tag{13}$$

where $m := \gamma n - 2(n_1^{\mathrm{mid}} + n_2^{\mathrm{mid}} + z_1 + z_2)$, and again z_1, z_2, o and t denote optimization parameters for the number of zeros, ones and twos represented via the respective combinations. Observe that again the number of total represented 1s (resp. -1s) add to n_1^{mid}, the number of 2s (resp. -2s) to n_2^{mid} and the number of 0s to $\gamma n - 2(n_1^{\mathrm{mid}} + n_2^{\mathrm{mid}})$ as required for mid-level elements. From here we can derive the number of representations as

$$R_1 = \binom{\gamma n - 2(n_1^{\mathrm{mid}} + n_2^{\mathrm{mid}})}{m, z_1, z_1, z_2, z_2} \binom{n_1^{\mathrm{mid}}}{\frac{n_1^{\mathrm{mid}}}{2} - o, \frac{n_1^{\mathrm{mid}}}{2} - o, o, o}^2 \binom{n_2^{\mathrm{mid}}}{\frac{n_2^{\mathrm{mid}}-t}{2}, \frac{n_2^{\mathrm{mid}}-t}{2}, t}^2,$$

where the first term counts the representations of 0, the second those of ± 1 and the last those of ± 2 coordinates. As before, a counting argument yields the necessary number of ± 1 and ± 2 coordinates in the function domains as

$$n_1 = z_1 + \frac{n_1^{\mathrm{mid}}}{2} - o + o + t = z_1 + t + \frac{n_1^{\mathrm{mid}}}{2} \quad \text{and} \quad n_2 = z_2 + o + \frac{n_2^{\mathrm{mid}} - t}{2}.$$

Now that we determined the number of representations R_1 and R_2 we obtain the running time $T_{\text{Nested-2}^*}$ of this instantiation using Eq. (11). In our numerical optimization of the running time we optimize over the choice of $\tilde{z}_1, \tilde{z}_2, \tilde{o}, \tilde{t}, \tilde{z}_1^{\mathrm{mid}}, \tilde{z}_2^{\mathrm{mid}}, \tilde{o}^{\mathrm{mid}}, \gamma$ and β, where for integer optimization parameters χ we let $\chi = \tilde{\chi} n$ with $\chi \in [0, 1]$.

We illustrate the optimized runtime exponents of our Nested-1^* and Nested-2^* instantiations in comparison to our previous Nested-1^+ instantiation in Fig. 11 on the left. We observe improvements especially for high weights. However, we also obtain improvements for smaller weights. In the same figure on the right, we illustrate the exponent difference between Nested-1^* and Nested-1^+ as well as between Nested-1^* and Nested-2^*. For Nested-1^* we observe improvements starting from $w/n \geq 0.44$, which marks the point where we have $\gamma < 1$. Since the improvement of Nested-1^* stems entirely from using

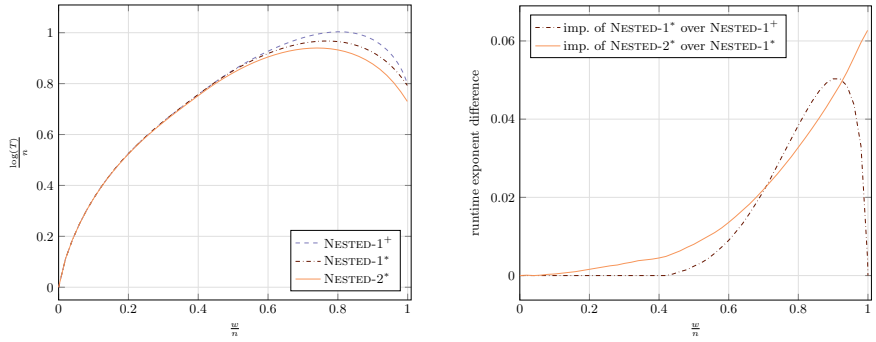


Fig. 11: On the left: Runtime exponents of NESTED-1^+, NESTED-1^* and NESTED-2^*. On the right: Improvement in the runtime exponent $(\log T_A - \log T_B)/n$ of $B =$ NESTED-1^* over $A =$ NESTED-1^+ (dash dotted line) and the improvement of $B =$ NESTED-2^* over $A =$ NESTED-1^* (solid line).

the permutation to shift more weight to the disjoint part of size $(1-\gamma)n$, we expect no improvement as long as $\gamma = 1$. We also observe that for $w/n = 1$ both instantiations NESTED-1^+ and NESTED-1^* converge to the same running time, resulting in a difference of zero. On the other hand, NESTED-2^* obtains further improvements over NESTED-1^* for all choices of the weight w, with higher gains towards larger values of w. The gain in this case stems entirely from adding the ± 2 to the representations and is therefore not bound to parameterizations with $\gamma < 1$.

4.4 An Improvement for Known Error Weight

We conclude this section by outlining a (small) improvement when the error weight is known. Here we outline the technique for NTRU instances, which typically use random ternary errors with expected weight $w/n = 2/3$. The idea is to apply an initial permutation to redistribute the weight on $(\mathbf{e}, \mathbf{s})$, similar to Information Set Decoding (ISD) techniques [40]. Therefore we rewrite the LWE identity $\mathbf{As} = \mathbf{b} + \mathbf{e}$ as

$$(\mathbf{I} \mid \mathbf{A})(-\mathbf{e}, \mathbf{s}) = \mathbf{b},$$

where $\mathbf{I}$ is the $n \times n$ identity matrix. Now applying a permutation to the columns of $(\mathbf{I} \mid \mathbf{A})$ yields

$$(\mathbf{I} \mid \mathbf{A})\mathbf{P}\big(\mathbf{P}^{-1}(-\mathbf{e}, \mathbf{s})\big) = \mathbf{H}(-\mathbf{e}', \mathbf{s}') = \mathbf{b},$$

where $(-\mathbf{e}', \mathbf{s}') := \mathbf{P}^{-1}(-\mathbf{e}, \mathbf{s})$. Further multiplying both sides of the equation with an invertible matrix $\mathbf{Q}$, such that $\mathbf{QH} = (\mathbf{I} \mid \mathbf{A}')$ and defining $\mathbf{b}' := \mathbf{Qb}$ yields

$$\mathbf{A}'\mathbf{s}' = \mathbf{b}' + \mathbf{e}'.$$

Now, assume that the permutation distributes a balanced weight of $w - p$ on $\mathbf{s}'$ and accordingly a balanced weight of $2n/3 + p$ on $\mathbf{e}'$, since $\mathbf{e}$ is usually a uniform ternary

vector. Then we expect Algorithm 2 to perform faster on the reduced weight instance $(\mathbf{A}', \mathbf{b}')$ than on the initial instance $(\mathbf{A}, \mathbf{b})$ as its running time (compare to Eq. (11)) depends only on the weight of $\mathbf{s}$ but not on the weight of $\mathbf{e}$. On the downside we need to reapply the algorithm $P = \frac{\binom{2n}{\frac{n}{3}+\frac{w}{2}, \frac{n}{3}+\frac{w}{2}, .}}{\binom{n}{\frac{w-p}{2}, \frac{w-p}{2}, .}\binom{n}{\frac{n}{3}+\frac{p}{2}, \frac{n}{3}+\frac{p}{2}, .}}$ times on random permutations of the instance to expect the weight to be distributed as desired for one of the instances. The running time is then given as $P \cdot T_{w-p}$, where T_{w-p} is the same as T in Eq. (11) but for $w - p$ instead of w. In the uniform secret case of $w/n = 2/3$ this yields a (slight) improvement from $2^{0.93n}$ down-to $2^{0.926n}$ for our Nested-2^* instantiation. Note that if w is small the secret $\mathbf{s}'$ after the permutation is expected to have weight $w' > w$, which is why we do not obtain improvements in this regime.

Also note that one can always enforce a secret weight of $|\mathbf{s}'| = |\mathbf{e}|$, by choosing the permutation $\mathbf{P}$, such that $\mathbf{P}^{-1}(-\mathbf{e}, \mathbf{s}) = (\mathbf{s}, -\mathbf{e}) =: (-\mathbf{e}', \mathbf{s}')$. For the case of uniform random secrets, this turns out to be beneficial whenever $0.81 \geq w > 0.688n$ (compare to Fig. 12).

Overall this shows that the permutation techniques predominantly used in the ISD context, translate to the LWE setting. However, for a general study, also different error and secret distributions and the effect of the permutation on the corresponding weights have to be considered. In those cases permutations that leverage the knowledge on those distributions as done in [13, 22] might yield improvements over random permutations. We leave the full study of those techniques on arbitrary distributions as a future research direction.

Strong relation to decoding linear codes. Note that the above technique shows that the LWE problem with small max norm error and secret is closely related to the decoding of linear codes in general. Namely, $(\mathbf{I} \mid \mathbf{A}), \mathbf{b}$, can be seen as a syndrome decoding instance, where $\mathbf{H} := (\mathbf{I} \mid \mathbf{A})$ is the parity-check matrix of a linear code over $\mathbb{F}_q$, $\mathbf{b}$ the syndrome and $(-\mathbf{e}, \mathbf{s})$ the low weight solution.

Note that here the solution weight is so low that even if we restrict to the lower 2ℓ rows of $\mathbf{H}$, the solution is still uniquely determined by those 2ℓ parity equations. The resulting parity-check matrix $\mathbf{H}' = (\mathbf{I}_{2\ell} \mid \mathbf{A}')$ then describes a linear code with rate $(n - 2\ell)/n = 1 - o(1)$. Usually the best known algorithms for decoding those codes rely on pure enumeration of the search space. However, the above technique shows that the specific choice of parameters, especially the relation between constant max norm, error weight and q, seems to allow for further optimizations.

5 Complexity of Solving Ternary LWE Without Memory

Eventually, let us give a concluding comparison between the best instantiations of the basic collision search by van Vredendaal (v-V) and May (Rep-2) and our best Nested-2^* instantiation of the nested collision search approach. We illustrate the runtime exponents of all these algorithms on the left of Fig. 12. Observe that our Nested-2^* algorithm yields the best running time for all choices of the

w/n	v-V	Rep-2	Nested-2*
0.300	0.8860	0.7716	**0.6482**
0.375	0.9971	0.8573	**0.7272**
0.441	1.0732	0.9172	**0.7928**
0.500	1.1250	0.9620	**0.8425**
0.621	1.1837	1.0376	**0.9140**
0.668	1.1887	1.0632	**0.9262**

Table 1: Runtime exponents for nested collision search (including improvement from Section 4.4) in comparison to conventional collision search approaches.

weight w. Moreover the improvement in the exponent compared to the minimum of v-V and Rep-2 reaches as high as 0.2 for a weight of $w = 0.81n$. While the most interesting weights are usually smaller than that, note that we also obtain significant improvements for all cryptographically relevant weights. For instance for a weight of $w = 0.667n$, which models the uniform secret case we obtain a significant improvement by a factor larger than $2^{0.13n}$. Table 1 shows the runtime exponent of all three methods for various weights used in schemes belonging to the NTRU-family. The exponent improvement of our Nested-2* for all weights

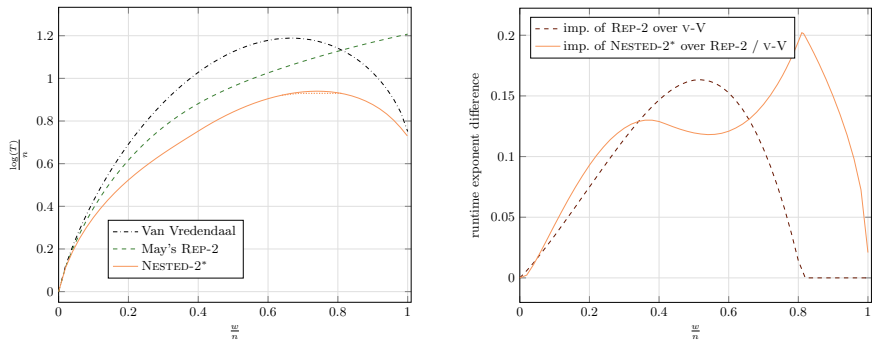


Fig. 12: On the left: Runtime exponents of van Vredendaal's, May's and our nested approach. Improvement for uniform random secrets (Section 4.4) illustrated as orange dotted line. On the right: Improvement in the runtime exponent of May's Rep-2 over van Vredendaal (dashed line) and of Nested-2* over the minimum of van Vredendaal's and May's algorithms (solid line).

w/n compared to the best previous approach is illustrated on the right of Fig. 12. As comparison the graphic shows the runtime improvement of May over van Vredendaal. Note that for $w \geq 0.82$ May does not obtain any improvement over van Vredendaal.

6 Extending Results to Kyber and Dilithium

In the following, we extend our results as well as the results from May and van Vredendaal to the cases of Kyber and Dilithium, which also rely on the hardness of LWE with small max norm keys. We recall that this extension requires the heuristic assumption that the introduced structure does not affect our analysis.

More precisely, Kyber uses keys sampled from a centered binomial distribution $\mathcal{B}(\eta)$ with parameter $\eta \in \{2, 3\}$, resulting in keys $\mathbf{s} \in \{-\eta, \ldots, \eta\}$. Dilithium keys have coordinates uniformly distributed over $\{\pm 2, \pm 1, 0\}$, which we denote by $\mathcal{U}(2)$, implying keys $\mathbf{s} \in \{-2, \ldots, 2\}$.

Key-Dist.	v-V	Rep-3	Nested-3*
$\mathcal{U}(1)$	1.1888	1.0625	**0.9297**
$\mathcal{U}(2)$	1.7415	1.4601	**1.2815**
$\mathcal{U}(3)$	1.9698	1.7323	**1.5049**
$\mathcal{B}(1)$	1.1250	0.9620	**0.8427**
$\mathcal{B}(2)$	1.5230	1.2118	**1.0404**
$\mathcal{B}(3)$	1.7501	1.3585	**1.1838**

Table 2: Runtime exponents for nested collision instantiations and conventional collision search approaches with different key distributions.

We give in Table 2 the runtime exponents on Kyber and Dilithium key distributions of Algorithm 1 using the van-Vredendaal instantiations (v-V) as well as using Rep-3 representations, i.e., we represent the solution $\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$ with $\mathbf{s}_i \in \{\pm 3, \pm 2, \pm 1, 0\}$. Additionally, we state the runtime exponent of our nested collision search, Algorithm 2, using a Nested-3* instantiation, which is the same as Nested-2*, but extending function domains by ± 3. We also provide data for the $\mathcal{U}(1)$,$\mathcal{U}(3)$ and $\mathcal{B}(1)$ distributions to indicate the scaling.

Additionally we provide in Table 3 the running time exponent c in dependence on the search space, i.e., the running time is of the form $T = \mathcal{D}^c$ with $\mathcal{D}$ the size of the search space. We observe that for both distributions the attacks become more efficient for increasing η, indicated by the decreasing value of c. This is related to the representation method, which overcompensates the increase in domain size by the increasing number of representations. Note that this indicates that with respect to combinatorial approaches increasing η will not result in significantly increased security.

Our attacks are especially efficient on the centered binomial distributions used in Kyber, where they reach almost the meet-in-the-middle exponent $c = 0.5$. However, for Dilithium like distributions ($\mathcal{U}(2)$) we also obtain a notable improvement down to a constant of $c = 0.552$. We provide all details on the analysis in Appendix B.

Key-Dist.	v-V	Rep-3	Nested-3*
$\mathcal{U}(1)$	0.75	0.6704	**0.5866**
$\mathcal{U}(2)$	0.75	0.6289	**0.5519**
$\mathcal{U}(3)$	0.75	0.6171	**0.5361**
$\mathcal{B}(1)$	0.75	0.6414	**0.5619**
$\mathcal{B}(2)$	0.75	0.5968	**0.5124**
$\mathcal{B}(3)$	0.75	0.5832	**0.5074**

Table 3: Runtime exponents $c = \log_{\mathcal{D}} T$ for nested collision instantiations and conventional collision search approaches with different key distributions in dependence on the search space size $\mathcal{D}$.

7 Time-Memory Trade-Offs

So far, all presented attacks can be instantiated with a polynomial amount of memory. However, in a realistic attack scenario even low memory devices such as FPGAs or GPUs still have a small amount of memory available. Further, the question remains how to best interpolate these attacks to the exponential memory endpoints given by May's algorithm [35].

In the following sections we construct continuous time-memory trade-offs for arbitrary available memory amounts. We start from our polynomial memory algorithms providing trade-offs for small, but exponential amounts of memory. We then revisit May's algorithm which serves as a starting point for new time-memory trade-offs covering the range of higher available memory amounts.

7.1 Time-Memory Trade-Off using PCS

In the following we show how to apply the time-memory trade-off technique known as Parallel Collision Search (PCS) [43] to our construction to further speed up our algorithms by the use of small but exponential amounts of memory.

Theorem 1 (Parallel Collision Search, [43]). *Let $f_1, f_2 : S \to S$ be two independent random functions. Then Parallel Collision Search finds M collisions between f_1 and f_2 using on expectation $\tilde{\mathcal{O}}\left((M \cdot |S|)^{1/2}\right)$ function evaluations and $\tilde{\mathcal{O}}(M)$ units of memory.*

Recall that, to succeed Algorithm 2 has to find multiple collisions between g_1 and g_2, namely on expectation $C := (q_1 q_2 q_3)^{-1}$ many (compare to Eq. (9)). So far, those collisions are found by iterative applications of the collision search technique. We now use the PCS technique to find M collisions at once by increasing the memory usage of the algorithm to $\tilde{\mathcal{O}}(M)$.

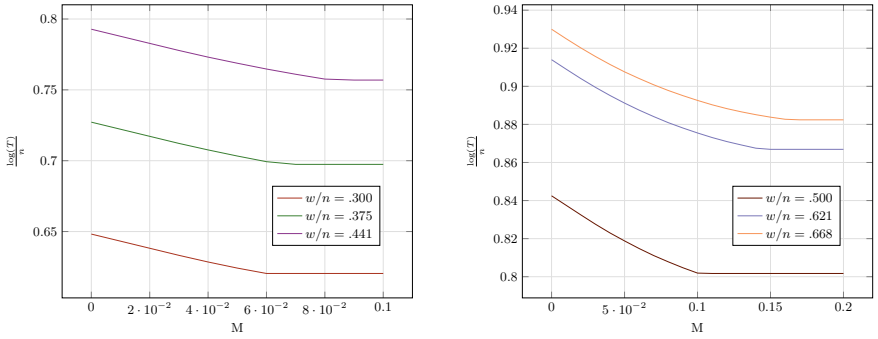
However, such a straightforward application of the PCS technique is not sufficient to achieve meaningful trade-offs for reasonable amounts of memory. This is because the amount of needed collisions C is an upper bound for the

maximum memory that can be spend and usually C is quite small for optimal instantiations. In order to obtain instantiations leveraging more memory, we adapt the time complexity to incorporate the PCS speedup and perform a numerical re-optimization of the running time. This allows for a choice of instantiations with larger C that in turn enables to fully leverage the available memory. However, once C becomes maximal no further speedups by increasing the memory are possible. Table 4 provides a comparison of the running time using polynomial memory and the running time using the maximal amount of memory that can be leveraged.

w/n	time at poly. memory	best time	required memory
0.300	0.6482	0.6204	0.06
0.375	0.7272	0.6974	0.07
0.441	0.7928	0.7569	0.09
0.500	0.8425	0.8017	0.11
0.621	0.9140	0.8669	0.15
0.668	0.9262	0.8824	0.17

Table 4: Runtime and memory exponents for time-memory trade-offs in comparison to polynomial memory algorithm Nested-2^*.

Note that there also exist instantiations for any memory smaller than the maximal memory given in the table. We provide the full trade-off curves in Fig. 13.



(a) w/n=.300,.375,.441. (b) w/n=.500,.621,.668.

Fig. 13: Time-Memory Trade-Off curves using PCS for different weights. Here the memory consumption is $2^{Mn+o(n)}$.

In Fig. 13 observe that, the performance of the time-memory trade-off using parallel collision search is good only for low memory regime. This motivates us to look into trade-offs based on May's technique.

7.2 Time-Memory Trade-Off using Enumeration

The second trade-off we introduce uses May's original algorithm as a starting point. In order to obtain a continuous time-memory trade-off we later adapt a recent technique introduced by Esser and Zweydinger [23] in the context of subset sum to the LWE case. We start by revisiting and slightly generalizing May's original algorithm.

A Generalization of May's Algorithm On a high level, instead of sampling elements via collision search that satisfy the layer-1 and layer-2 identities, May's algorithm enumerates them exhaustively similar to the enumeration improvement from advanced ISD improvements [5, 36]. Since again there exist multiple representations constraints can be imposed on the search space ensuring the construction of only one of those representations, making the algorithm efficient. In contrast to the Nested-Collision-Search algorithm which is optimal in two layers, May's attack improves in the running time when the technique is applied in deeper recursion, i.e., there will be more than two layers. We start by describing May's algorithm for depth-4 similar to the original proposal [35]. We later provide a generalization to arbitrary depth that follows the same principles. In this context we show that the increase to higher depth for our generalization of May's algorithm results in runtime improvements up to depth six, after which the runtime converges. This behavior is similar for May's original algorithm as shown by Glaser and May [26].

Let $\mathbf{As} = \mathbf{b} + \mathbf{e} \mod q$ be a random LWE instance with $\mathbf{s}, \mathbf{e} \in \{-1, 0, 1\}^n$. Similar to the approach in Section 4 the solution is first split as

$$\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2 + \mathbf{s}_3 + \mathbf{s}_4 + \mathbf{s}_5 + \mathbf{s}_6 + \mathbf{s}_7 + \mathbf{s}_8, \quad \text{with } \mathbf{s}_i \in S^{(3)}.$$

Here $S^{(3)}$ describes the level-3 search space which is the analog to the previous function domains $\mathcal{T}_i$ in Section 4. Again we fully describe the search spaces later on in the analysis.

On lower levels of the tree (compare to Fig. 14) candidates for sums of the $\mathbf{s}_i$ are constructed. Therefore, to ease notation, we define the elements constructed on level k as $\mathbf{s}_i^{(k)} = \mathbf{s}_{2i-1}^{(k+1)} + \mathbf{s}_{2i}^{(k+1)}$, where $\mathbf{s}_i^{(3)} := \mathbf{s}_i$, which implies

$$\begin{aligned}
\mathbf{s} &= \mathbf{s}_1 + \mathbf{s}_2 + \mathbf{s}_3 + \mathbf{s}_4 + \mathbf{s}_5 + \mathbf{s}_6 + \mathbf{s}_7 + \mathbf{s}_8 \\
&= \mathbf{s}_1^{(3)} + \mathbf{s}_2^{(3)} + \mathbf{s}_3^{(3)} + \mathbf{s}_4^{(3)} + \mathbf{s}_5^{(3)} + \mathbf{s}_6^{(3)} + \mathbf{s}_7^{(3)} + \mathbf{s}_8^{(3)} \\
&= \mathbf{s}_1^{(2)} + \mathbf{s}_2^{(2)} + \mathbf{s}_3^{(2)} + \mathbf{s}_4^{(2)} \\
&= \mathbf{s}_1^{(1)} + \mathbf{s}_2^{(1)}.
\end{aligned}$$

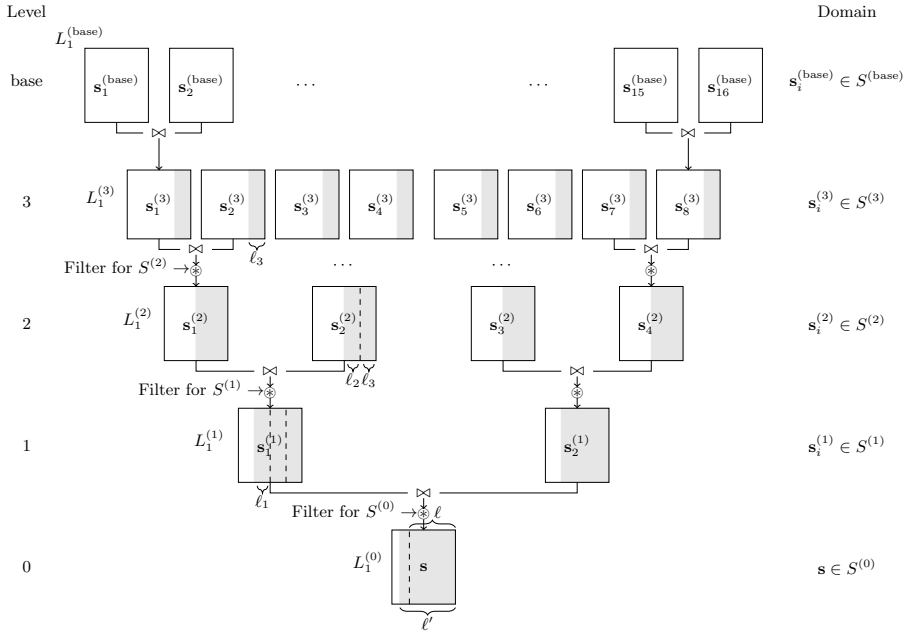


Fig. 14: Combinatorial key search with depth-4 tree

Again we assume that the first ℓ' coordinates of $\mathbf{e}$ are known, as their guessing only leads to a slight sub-exponential overhead. Hence $\mathbf{c}_1^{(0)} := \pi_{\ell'}(\mathbf{b}+\mathbf{e})$ is known, which together with the LWE identity $\mathbf{As} = \mathbf{b} + \mathbf{e}$ leads to

$$\pi_{\ell'}(\mathbf{As}) = \mathbf{c}_1^{(0)} \mod q \tag{14}$$

Analogous to the layer-1 and layer-2 identities from Section 4 we refer to Eq. (14) as *level-0 identity*, which will be satisfied by any element contained in the final level-0 list (compare to Fig. 14). Next, for every level $i = 1, \ldots, d-1$ we identify analogous level-i identities, to be satisfied by elements in the corresponding level-i lists. Therefore, let $\ell := \ell_1 + \ell_2 + \ell_3 \leq \ell'$, where the precise value of these ℓ_i is determined later. On each level $i = 1, \ldots 3$ a total constraint on $\sum_{m=i}^{3} \ell_m =: \ell^{(i)}$ coordinates of $\mathbf{As}_j^{(i)}$ is imposed. Precisely we have $\ell^{(3)} = \ell_3$, $\ell^{(2)} = \ell_2 + \ell_3$ and $\ell^{(1)} = \ell_1 + \ell_2 + \ell_3 = \ell$. The level-0 identity in combination with $\mathbf{s} = \mathbf{s}_1^{(1)} + \mathbf{s}_2^{(1)}$ leads to

$$\pi_{\ell^{(1)}}(\mathbf{As}_1^{(1)}) = \pi_{\ell^{(1)}}(\mathbf{c}_1^{(0)}) - \pi_{\ell^{(1)}}(\mathbf{As}_2^{(1)}) \mod q.$$

We now, define the level-1 identities as

$$\pi_{\ell^{(1)}}(\mathbf{As}_i^{(1)}) = \mathbf{c}_i^{(1)} \mod q, \quad i = 1, 2. \tag{15}$$

Note that the values of $\mathbf{c}_i^{(1)}$ are not known. However, once $\mathbf{c}_1^{(1)}$ is fixed, $\mathbf{c}_2^{(1)} = \pi_\ell(\mathbf{c}_1^{(0)}) - \mathbf{c}_1^{(1)} \mod q$ is fully determined. In the construction, analogous to the

value of $\mathbf{r}$ in the memoryless algorithm (compare to Eq. (3)) $\mathbf{c}_1^{(1)}$ is later chosen at random. We therefore refer to $\mathbf{c}_1^{(1)}$ as *free* level-1 constraint, while we denote $\mathbf{c}_2^{(1)}$ as *fixed*.

Now, analogously to the level-1 identities, we define the *level-2 identities* as

$$\pi_{\ell^{(2)}}(\mathbf{A}\mathbf{s}_i^{(2)}) = \mathbf{c}_i^{(2)} \mod q, \quad i = 1, 2, 3, 4. \tag{16}$$

We let $\mathbf{c}_1^{(2)}$ and $\mathbf{c}_3^{(2)}$ be *free* level-2 constraints, i.e., their value is chosen randomly in $\mathbb{F}_q^{\ell^{(2)}}$. Since from the further recursive splitting of the $\mathbf{s}_j^{(1)} = \mathbf{s}_{2j-1}^{(2)} + \mathbf{s}_{2j}^{(2)}$ we obtain

$$\begin{aligned} \pi_{\ell^{(2)}}(\mathbf{A}(\mathbf{s}_1^{(2)} + \mathbf{s}_2^{(2)})) &= \pi_{\ell^{(2)}}(\mathbf{c}_1^{(1)}) && \mod q \\ \pi_{\ell^{(2)}}(\mathbf{A}(\mathbf{s}_3^{(2)} + \mathbf{s}_4^{(2)})) &= \pi_{\ell^{(2)}}(\mathbf{c}_1^{(0)} - \mathbf{c}_1^{(1)}) && \mod q, \end{aligned}$$

it follows that $\mathbf{c}_2^{(2)}$ and $\mathbf{c}_4^{(2)}$ are *fixed* level-2 constraints, since they are determined by the free constraints as

$$\mathbf{c}_2^{(2)} = \pi_{\ell^{(2)}}(\mathbf{c}_1^{(1)}) - \mathbf{c}_1^{(2)} \mod q \quad \text{and} \quad \mathbf{c}_4^{(2)} = \pi_{\ell^{(2)}}(\mathbf{c}_1^{(0)} - \mathbf{c}_1^{(1)}) - \mathbf{c}_3^{(2)} \mod q.$$

We proceed in the same manner by defining the level-3 identities as

$$\pi_{\ell^{(3)}}(\mathbf{A}\mathbf{s}_i^{(3)}) = \mathbf{c}_i^{(3)} \mod q. \tag{17}$$

Due to the recursive splitting from level 2 to level 3, i.e., $\mathbf{s}_j^{(2)} = \mathbf{s}_{2j-1}^{(3)} + \mathbf{s}_{2j}^{(3)}$, it holds

$$\pi_{\ell^{(3)}}\left(\mathbf{A}(\mathbf{s}_{2i-1}^{(3)} + \mathbf{s}_{2i}^{(3)})\right) = \pi_{\ell^{(3)}}(\mathbf{c}_i^{(2)}) \mod q, \quad \text{for } i = 1, 2, 3, 4$$

This implies that again the constraints $\mathbf{c}_i^{(3)} \in \mathbb{F}_q^{\ell^{(3)}}$ for $i = 1, 3, 5, 7$ are *free*, while those for $i = 2, 4, 6, 8$ are *fixed*.

In order to construct candidates for $\mathbf{s}_i^{(3)}$ satisfying the level-3 constraints (Eq. (17)), a meet in the middle approach is used. Therefore, for $i = 1, \cdots, 8$, $\mathbf{s}_i^{(3)}$ is split as

$$\mathbf{s}_i^{(3)} = (\mathbf{s}_{2i-1}^{(\text{base})}, 0^{n/2}) + (0^{n/2}, \mathbf{s}_{2i}^{(\text{base})}).$$

The algorithm starts by enumerating all possible values of $\mathbf{s}_i^{(\text{base})} \in S^{(\text{base})}$ in the base list $L_i^{(\text{base})}$. Then the two lists $L_{2i-1}^{(\text{base})}$ and $L_{2i}^{(\text{base})}$ are joined ($\bowtie$) by identifying those elements $\mathbf{s}_{2i-1}^{(\text{base})}, \mathbf{s}_{2i}^{(\text{base})}$ which jointly satisfy the level-3 identity, i.e., for which it holds that

$$\pi_{\ell_3}\left(\mathbf{A}(\mathbf{s}_{2i-1}^{(\text{base})}, \mathbf{0})\right) = \mathbf{c}_i^{(3)} - \pi_{\ell_3}\left(\mathbf{A}(\mathbf{0}, \mathbf{s}_{2i}^{(\text{base})})\right) \mod q.$$

Now $L_{2i-1}^{(3)}$ is joined with $L_{2i}^{(3)}$ to create the level-2 list $L_i^{(2)}$ which contains only those elements that satisfy the corresponding level-2 identity (Eq. (16)). During this join elements satisfying the level-2 identities are filtered on the fly for those

falling into the level-2 search space $S^{(2)}$, i.e., any $\mathbf{x} \notin S^{(2)}$ is discarded. Now, the same procedure is repeated to create the level-1 lists and finally the level-0 list, which contains the solution. We outline the algorithm in Algorithm 3 and give a visual illustration of the list creation in Fig. 14.

Note that in contrast to the memoryless version, parameters in May's construction are chosen such that it is guaranteed that for any choice of the *free* constraints $\mathbf{c}_i^{(j)}$ there still exist a representation in expectation. In turn the algorithm does not need to iterate over multiple values of the constraints.

Algorithm 3: Combinatorial Key Search [35]

Input: $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n$, positive integer $w \leq n$
Output: $\mathbf{s} \in \tau^n(w/2)$ such that $\mathbf{e} = \mathbf{As} - \mathbf{b} \mod q \in \{-1, 0, 1\}^n$

$\ell' := \log_q(|\tau^n(w/2)|)$
Choose optimal $\ell_1, \cdots, \ell_{d-1}$ and let $\ell^{(k)} := \sum_{m=k}^{d-1} \ell_m$, $k \geq 1$ and $\ell^{(0)} := \ell'$
for $i = 1, \cdots, 2^d$ **do**
    Enumerate
$$L_i^{(\text{base})} = \{(\mathbf{s}_i^{(\text{base})}, \mathbf{0}^{n/2}) | \mathbf{s}_i^{(\text{base})} \in S^{(\text{base})}\}$$
repeat
    Guess the lower ℓ' coordinates of $\mathbf{e}$ and let $\mathbf{c}_1^{(0)} := \pi_{\ell'}(\mathbf{b} + \mathbf{e})$
    Choose free constraints $\mathbf{c}_{2i-1}^{(j)} \in \mathbb{F}_q^{\ell^{(j)}}$, for $j = 1, \cdots, d-1, \quad i = 1, \cdots, 2^{j-1}$ at random and set fixed constraints accordingly
    for $i = 1, \cdots, 2^{d-1}$ **do**
        Compute
$$L_i^{(d-1)} := \{\mathbf{z} = \mathbf{x}_{2i-1} + \mathbf{x}_{2i} \mid \mathbf{x}_o \in L_o^{(\text{base})} \wedge \pi_{\ell_{d-1}}(\mathbf{Az}) = \mathbf{c}_i^{(d-1)} \mod q\}$$
    for $j = d-1, \cdots, 1$ **do**
        for $i = 1, \cdots, 2^{j-1}$ **do**
            Compute
$$L_i^{(j-1)} = \{\mathbf{z} = \mathbf{s}_{2i-1}^{(j)} + \mathbf{s}_{2i}^{(j)} \mid \mathbf{s}_o^{(j)} \in L_o^{(j)} \wedge \pi_{\ell^{(j-1)}}(\mathbf{Az}) = \mathbf{c}_i^{(j-1)} \mod q\}$$
until $\mathbf{s}' \in L_1^{(0)} \colon \mathbf{As}' - \mathbf{b} \bmod q \in \{-1, 0, 1\}^n$
return $\mathbf{s}'$

Analysis of Algorithm 3. Note that any element $\mathbf{s}'$ in the final list satisfies the level-0 identity

$$\pi_{\ell'}(\mathbf{As}') = \mathbf{c}_1^{(0)} = \pi_{\ell'}(\mathbf{b} + \mathbf{e}).$$

The expected number of ternary elements with $w/2$ entries equal to ± 1 each satisfying this identity is $|\tau^n(w/2)|/q^{\ell'} = 1$. Hence, we conclude that as soon as $|L_1^{(0)}| > 0$, the final list contains the solution $\mathbf{s}$. Further, we choose parameters

such that the expected size of $|L_1^{(0)}|$ is greater than zero, implying that the solution is found, once the correct guess for $\pi_{\ell'}(\mathbf{e})$ has been made.

Next we study the time and memory complexity of the algorithm. Therefore, let us first analyze the list sizes on each level. Lists on any level i are filtered on the fly such that $L_j^{(i)} \subseteq S^{(i)}$. We provide the exact definition of the $S^{(i)}$ at then end of the analysis. However, we choose those search spaces that again there exist $R_{i+1} > 1$ representations of any element from $S^{(i)}$ as sum of two elements from $S^{(i+1)}$.

Let the expected list size of lists on level i (after filtering) be $\mathcal{F}_i$. Since, the initial lists of the tree are constructed via a meet-in-the-middle of the level-$(d-1)$ search space $S^{(d-1)}$, we have $\mathcal{F}_{\text{base}} = \sqrt{|S^{(d-1)}|}$.

Further, let the unfiltered lists at level-i be of size $\mathcal{L}_i$, where $\mathcal{L}_{\text{base}} = \mathcal{F}_{\text{base}}$. Let q_i denote the probability that a random sum of two elements from level $(i+1)$ forms a representation of any level-i element, i.e., q_i denotes the probability that any level-i element survives the filter. Since there are $|S^{(i+1)}|^2$ possible sums of level-$(i+1)$ elements and each level-i element has R_{i+1} representations as sum of two level-$(i+1)$ elements, we have

$$q_i = \frac{|S^{(i)}| \cdot R_{i+1}}{|S^{(i+1)}|^2}.$$

Note that by the definition of $\mathcal{F}_i, \mathcal{L}_i$ and q_i we have $\mathcal{F}_i = q_i \mathcal{L}_i$.

The level-3 lists are constructed by joining two base lists to satisfy a modular constraint $\mathbf{c}_i^{(3)} \in \mathbb{F}_q^{\ell^{(3)}}$, which implies $\mathcal{F}_3 = \frac{\mathcal{F}_{\text{base}}{}^2}{q^{\ell^{(3)}}}$. Since the base lists are a meet-in-the-middle of the search space $S^{(3)}$ any sum of elements from those lists fall into the search space, implying $q_3 = 1$ and, hence, $\mathcal{F}_3 = \mathcal{L}_3$.

Now two level-3 lists are joined to satisfy the level-2 identities, which corresponds to matching a modular constraint $\mathbf{c}_i^{(2)} \in \mathbb{F}_q^{\ell^{(2)}}$. However, note that for any sum of elements from level-3 the level-2 identities are satisfied on the lower $\ell^{(3)}$-coordinates already. Hence,

$$\mathcal{L}_2 = \frac{\mathcal{F}_3^2}{q^{\ell^{(2)} - \ell^{(3)}}} = \frac{\mathcal{F}_3^2}{q^{\ell_2}}$$

and $\mathcal{F}_2 = q_2 \mathcal{L}_2$. For subsequent levels we obtain analogously, $\mathcal{L}_1 = \frac{\mathcal{F}_2^2}{q^{\ell_1}}$, $\mathcal{F}_1 = q_1 \mathcal{L}_1$ and $\mathcal{L}_0 = \frac{\mathcal{F}_1^2}{q^{\ell' - \ell^{(1)}}}$, $\mathcal{F}_0 = q_0 \mathcal{L}_0$.

In order to guarantee that at least one representation of the solution is contained in the final list we enforce the following condition

$$\mathcal{F}_0 \geq 1,$$

in our optimization of the parameters.

Now, the overall time complexity is the time per iteration times the number of iterations. One iteration of the **repeat**-loop in Algorithm 3 is dominated by the

time to construct the unfiltered lists. Note that each such list can be constructed in time linear in the involved list sizes, leading to a time per iteration of

$$T_{\text{it}} = \tilde{\mathcal{O}}\left(\max_i \mathcal{L}_i\right).$$

As shown in the analysis of Algorithm 2 for $\ell' := \log_q |\tau^n(w/2)|$, where $w = \Theta(n)$, the expected amount of iterations until the guess of ℓ' coordinates of the error $\mathbf{e}$ is correct is subexponential in n. Now, recall that parameters are chosen to guarantee that the solution is constructed in the final list once a correct guess is made. Therefore the overall time complexity amounts to

$$T = (T_{\text{it}})^{1+o(1)} = \left(\max_i \mathcal{L}_i\right)^{1+o(1)}.$$

The memory-complexity of the algorithm is given by $M = \tilde{\mathcal{O}}\left(\max_i \mathcal{F}_i\right)$ as only filtered lists are stored.

Note that the above analysis assumes level-i lists to be random selections of elements from the level-i search-space $S^{(i)}$ that satisfy the level-i constraints. This is satisfied as long as level-i lists do not contain duplicates, i.e., the lists do not saturate the search spaces restricted to the elements satisfying the level-i constraints. In order to ensure that no saturation is reached we impose the restrictions

$$\mathcal{F}_3 \leq \frac{|S^{(3)}|}{q^{\ell^{(3)}}}, \quad \mathcal{F}_2 \leq \frac{|S^{(2)}|}{q^{\ell^{(2)}}} \quad \text{and} \quad \mathcal{F}_1 \leq \frac{|S^{(1)}|}{q^{\ell^{(1)}}}.$$

Note that previous works fixed the values of the $\ell^{(i)} = \log_2 R_i$, which we find to be valid but sub-optimal as our runtime results show.

Search spaces and representations. The search spaces $S^{(i)}$ are again defined to contain vectors of max norm smaller or equal to 2 and a certain coordinate distribution. More precisely, we denote the number of $\pm j$ in each element of $S^{(i)}$ by $n_j^{(i)}$. Thus, for $0 \leq i \leq d-1$, the level-i search space is of size

$$|S^{(i)}| = \binom{n}{n_2^{(i)}, n_2^{(i)}, n_1^{(i)}, n_1^{(i)}, .}.$$

Now, if the final solution $\mathbf{s} \in S^{(0)} = \tau^n(w/2)$ is a ternary vector of weight w, then it follows that

$$n_2^{(0)} = 0, \quad n_1^{(0)} = w/2 \quad \text{and} \quad n_0^{(0)} = n - w. \tag{18}$$

The definition of the search spaces then allows to compute the number of representations, similar to the computations done in Section 4. The number of representations of an element in level-0 list as a sum of two level-1 elements R_1 is computed exactly as the value of R_2 in Section 4.3 for $\gamma = 1$. This again involves the optimization parameters $o^{\text{mid}}, z_1^{\text{mid}}, z_2^{\text{mid}}$. The value of those optimization

parameters as before defines the coordinate distribution for the search space on level 1 as

$$n_2^{(1)} = o^{\text{mid}} + z_2^{\text{mid}}, \quad n_1^{(1)} = w/4 + z_1^{\text{mid}} \quad \text{and} \quad n_0^{(1)} = n - 2n_2^{(1)} - 2n_1^{(1)}.$$

Now for $i \geq 1$, the number of representations of an element in level-i list as a sum of two elements in level-$(i+1)$ lists R_{i+1} is computed exactly as the value of R_1 in Section 4.3 for $\gamma = 1$. Therefore for each R_i, $i \geq 1$ we introduce the optimization parameters $t^{(i)}, o^{(i)}, z_2^{(i)}, z_1^{(i)}$ (compare to Eq. (13)). Hence for $i \geq 1$ we have

$$\begin{aligned} n_2^{(i+1)} &= (n_2^{(i)} - t^{(i)})/2 + o^{(i)} + z_2^{(i)} \\ n_1^{(i+1)} &= n_1^{(i)}/2 + z_1^{(i)} + n_2^{(i)} - (n_2^{(i)} - t^{(i)}). \end{aligned}$$

Arbitrary depth. May's original algorithm was specified in depth 4, while Glaser-May then found improvements in the ternary solution case for depth 5 and 6. We therefore provide here the necessary formulas for an arbitrary depth-d version of the algorithm. The level-i constraints, $i = 0, \ldots, d-1$ are defined as

$$\pi_{\ell^{(i)}}(\mathbf{Az}) = \mathbf{c}_j^{(i)} \mod q, \quad \text{for } j = 1, \ldots, 2^i$$

where $\ell^{(i)} = \sum_{m=i}^{d-1} \ell_m$ for $i > 0$ and $\ell^{(0)} := \ell'$. The enforced saturation constraints generalize as

$$\mathcal{F}_i \leq \frac{|S^{(i)}|}{q^{\ell^{(i)}}}, \quad \text{for } i = 1, 2, d-1.$$

List sizes analogously satisfy

$$\mathcal{L}_{j-1} = \frac{\mathcal{F}_j^2}{q^{\ell_{j-1}}} \quad \text{and} \quad \mathcal{F}_j = q_j \mathcal{L}_j, \quad \text{for } j = 1, \cdots, d-1$$

Time and memory complexity again amount to

$$T = \left(\max_i \mathcal{L}_i\right)^{1+o(1)} \quad \text{and} \quad M = \tilde{\mathcal{O}}\left(\max_i \mathcal{F}_i\right)$$

For numerical optimization we analogously to the memoryless algorithm use the scipy python library. In Table 5 we provide a comparison of the time complexity exponent for different weight parameters reported in [26,35] and using our optimization. While we confirm an optimal depth of 6, we obtain slight improvements due to a higher precision in our numerical optimization when using the same fixed choice of $\ell^{(i)} = \log_2 R_{i+1}$ as in [26,35]. Furthermore, allowing for a flexible choice of the $\ell^{(i)}$, that satisfy the saturation constraints we obtain further improvements as shown in Table 5 (Flexible $\ell^{(i)}$)

Additionally we plot the runtime exponents for the depth-4, depth-5 and depth-6 variants with fixed $\ell^{(i)}$ as well as the depth-6 version with flexible $\ell^{(i)}$ choices in Fig. 15.

w/n	Depth-4	Depth-5/6	Depth-4	Depth-6	Flexible $\ell^{(i)}$
	[35]	[26]	this work		
0.300	0.2950	0.2950	0.2924	0.2927	**0.2893**
0.375	0.3180	0.3150	0.3145	0.3117	**0.3070**
0.441	0.3340	0.3260	0.3312	0.3223	**0.3182**
0.500	0.3480	0.3370	0.3447	0.3297	**0.3218**
0.620	0.3710	0.3420	0.3691	0.3389	**0.3308**
0.667	0.3790	0.3450	0.3777	0.3413	**0.3392**

Table 5: Runtime exponents for depth-4 and depth-6 variants of May's algorithms for different weights.

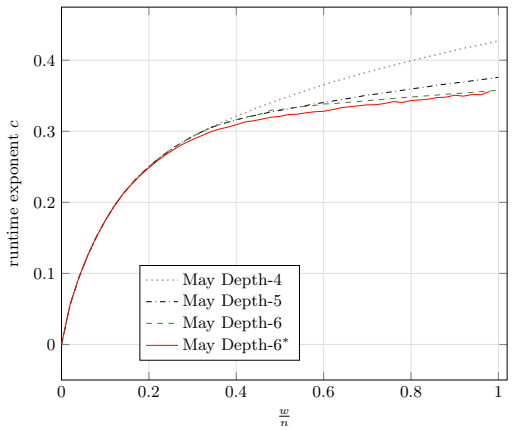


Fig. 15: Runtime exponents c as a function of the relative weight w/n for different depths. The running time is of the form $T = 2^{cn+o(n)}$.

A Continuous Time-memory Trade-off Technically, our generalization of May's algorithm allows for a certain time-memory trade-off, by choosing parameters such that the memory does not exceed a certain, given memory bound. This requires to choose different, namely larger values for the ℓ_i, which in turn leads to smaller list sizes throughout the tree. However, ensuring that the final list contains a solution puts an upper bound on the ℓ_i and in turn does not allow to decrease the memory of May's algorithm significantly.

To circumvent this problem we extend a technique introduced by Esser and Zweydinger [23] in the context of subset sum to the case of LWE. The high-level idea of this trade-off technique can be split in two parts. First, the condition that guarantees at least one representation of the solution to be present in the final list is dropped. This allows for a choice of the ℓ_i that balances the lists in a more memory efficient manner. To compensate for the reduced success probability, we execute the algorithm multiple times to ensure that at least in one of the

executions a representation of the solution is constructed. A core observation in this context is that not for every execution the whole tree has to be rebuild. The executions are randomized by exchanging the constraints on all levels i, $i-1, \ldots, 1$. Note that an exchange of the upper ℓ_i $\mathbb{F}_q$-coordinates of the level-i constraints affects only lists from level i onwards.[8] By starting with small $i = 1$, we can ensure that only levels 1 and 0 of the tree have to be re-build, from the already existing level-2 lists. Only after those lists have been re-build for all possible choices of the upper ℓ_1 coordinates of the free level-1 constraint $c_1^{(1)}$, we have to exchange the upper ℓ_2 coordinates of the free level-2 constraints. This implies then a necessary re-computation of level-2, -1 and -0 lists. Analogously, we proceed until the solution is found, exchanging constraints of higher levels only when all possible choices for lower levels have been examined.

The second key ingredient of the trade-off technique is the use of the general dissection framework by Dinur, Dunkelman, Keller and Shamir [15] for the construction of the level-$(d-1)$ lists. Recall, that in May's algorithm those lists are constructed via a meet-in-the-middle of the level-$(d-1)$ search spaces $S^{(d-1)}$, requiring a memory complexity of at least $\sqrt{|S^{(d-1)}|}$. For small available memories, this meet-in-the-middle construction forms a bottleneck, which does not allow to further reduce the memory. The dissection framework is a generalization of the meet-in-the-middle technique in form of a continuous time memory trade-off, that allows to decrease the memory to any amount smaller than $\sqrt{|S^{(d-1)}|}$ at the cost of an increased time complexity.

Again we give first a description of the trade-off in depth 4. In Appendix D we then give a recursive description for an arbitrary depth d. Let us denote the whole tree by $\mathfrak{T}$ and the subtree excluding level-$(i+1)$ lists and upwards by $\mathfrak{T}_i$. To perform randomized executions, we rebuild $\mathfrak{T}_i$ from the lists of the previous levels for randomized choices of the free level-i constraints q^{t_i} times before exchanging constraints of level $i+1$ (compare to Fig. 16).

Initially the procedure starts by randomly changing the upper ℓ_1-coordinates of the free level-1 constraint $\mathbf{c}_1^{(1)}$. The lists outside of subtree $\mathfrak{T}_1$ are not affected by this change, hence, we just rebuild $\mathfrak{T}_1$ from the existing level-2 lists. Note that there are q^{ℓ_1} choices for the upper ℓ_1-coordinates of the free level-1 constraint, implying that this repetition can be performed at most q^{ℓ_1} times. Hence $q^{t_1} \leq q^{\ell_1}$ or equivalently $t_1 \leq \ell_1$. Now if q^{t_1} randomized executions are not sufficient to find the solution, we start changing the upper ℓ_2-coordinates of the free level-2 constraints $\mathbf{c}_1^{(2)}, \mathbf{c}_3^{(2)}$ and the middle ℓ_2-coordinates of the modular constraint $\mathbf{c}_1^{(1)}$, i.e., the coordinates $(\mathbf{c}_1^{(1)})_{[\ell_1+1,\ell_2]}$. This implies that the subtree $\mathfrak{T}_2$ can be randomized at most q^{t_2} times with $t_2 \leq 3\ell_2$. For each possible choice of those partial constraints, we again re-build $\mathfrak{T}_1$ for all q^{t_1} choices of the upper ℓ_1 coordinates of $\mathbf{c}_1^{(1)}$ (or until the solution is found). Iterating over all constraints that affect only the subtree $\mathfrak{T}_2$ (including $\mathfrak{T}_1$) leads therefore to $q^{3\ell_2+\ell_1}$ randomized iterations. If this amount of iterations is still not sufficient to find the solution, we

[8] Recall, that the lower $\ell^{(i)} - \ell_i$ coordinates are determined by the choice of constraints $\mathbf{c}_o^{(j)}$ in levels $j > i$

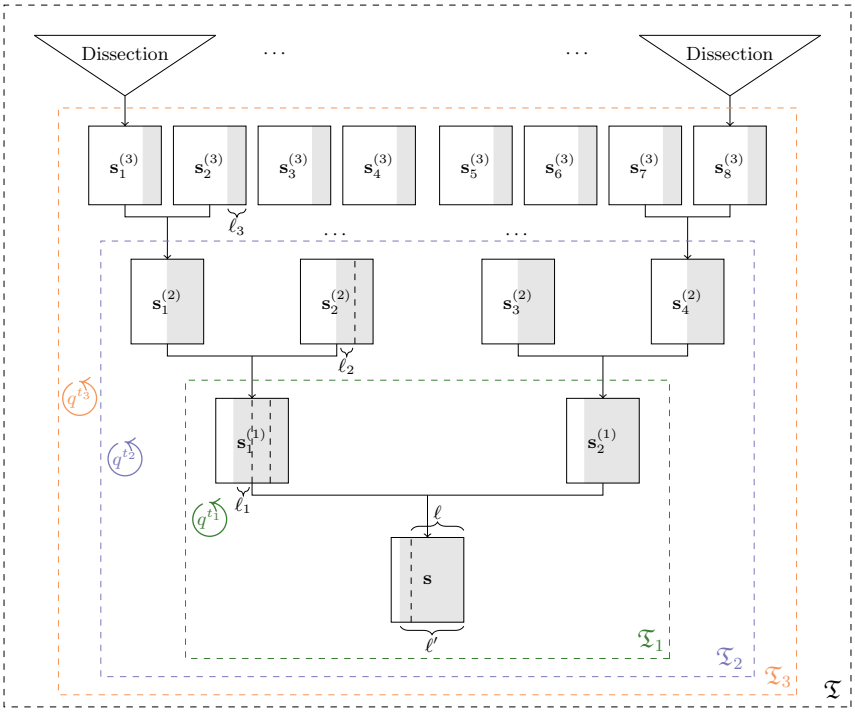


Fig. 16: Combinatorial Key Search with Repetition of Subtrees

start changing the lower ℓ_3-coordinates of all free constraints including those on level three. Similar to before, for each different choice of the lower ℓ_3-coordinates of the constraints we recompute the subtrees $\mathfrak{T}_2$ and $\mathfrak{T}_1$ several times. Since in total there are 7 free constraints in the depth-4 variant, we have that subtree $\mathfrak{T}_3$ offers at most q^{t_3} randomized executions with $t_3 \leq 7\ell_3$. Overall, this yields a total of $q^{7\ell_3+3\ell_2+\ell_1}$ iterations possible randomized executions. The pseudocode of the full procedure is given in Algorithm 4.

The level-3 lists are computed via the dissection framework. For the full details on this framework we refer to [15]. However, for our analysis the following lemma, specifying the time and memory complexity of the technique to compute the level-3 lists is sufficient.

Lemma 1 (Dissection [15]). *Let $c_i = \frac{i^2+3i+4}{2}$, with $i \in \mathbb{N} \bigcup \{0\}$ and $\frac{1}{c_i} \leq \lambda \leq \frac{1}{c_{i-1}}$. Then all the vectors in $S^{(3)}$ that satisfy the corresponding level-3 constraints can be found via the dissection framework in time*

$$T_{\text{Dissection}} = (|S^{(3)}|)^{\frac{(i-c_{i-1}\cdot\lambda+2\lambda)}{(i+1)}}$$

and using memory

$$M_{\text{Dissection}} = (|S^{(3)}|)^{\lambda}.$$

Algorithm 4: Combinatorial Key Search Trade-off for Depth-4

Input: $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n$, positive integer $w \leq n$
Output: $\mathbf{s} \in \tau^n(w/2)$ such that $\mathbf{e} = \mathbf{As} - \mathbf{b} \mod q \in \{-1, 0, 1\}^n$

$\ell' := \log_q |\tau^n(w/2)|$
$L = \emptyset$
Choose optimal ℓ_1, ℓ_2, ℓ_3 and the suitable d_k-dissection depending on the available memory
repeat
  Guess lower ℓ' coordinates of $\mathbf{e}$ randomly
  repeat q^{t_3} **times**
    Choose randomly the free constraints $\mathbf{c}_1^{(3)}, \mathbf{c}_3^{(3)}, \mathbf{c}_5^{(3)}, \mathbf{c}_7^{(3)}, \mathbf{c}_1^{(2)}, \mathbf{c}_3^{(2)}, \mathbf{c}_1^{(1)}$
    for $i = 0, \cdots, 7$ **do**
      Compute $L_{i+1}^{(3)}$ using d_k-dissection of the level-3 search space $S^{(3)}$
    repeat q^{t_2} **times**
      Choose randomly the ℓ_2-coordinates of $\mathbf{c}_1^{(2)}[\ell^{(3)} + 1, \ell^{(2)}]$, $\mathbf{c}_3^{(2)}[\ell^{(3)} + 1, \ell^{(2)}]$, $\mathbf{c}_1^{(1)}[\ell^{(3)} + 1, \ell^{(2)}]$ `// Here x[i, j] denotes the projection onto the` i^{th} `till` j^{th} `coordinate of x and` $\ell^{(i)} := \sum_{m=i}^{3} \ell_m$.
      for $i = 0, \cdots, 3$ **do**
        From $L_{2i+1}^{(3)}$ and $L_{2i+2}^{(3)}$ compute $L_{i+1}^{(2)}$
      repeat q^{t_1} **times**
        Choose randomly the ℓ_1-coordinates of $\mathbf{c}_1^{(1)}[\ell^{(2)} + 1, \ell^{(1)}]$
        for $i = 0, 1$ **do**
          From $L_{2i+1}^{(2)}$ and $L_{2i+2}^{(2)}$ compute $L_{i+1}^{(1)}$
        From $L_1^{(1)}$ and $L_2^{(1)}$ compute $L_1^{(0)}$
        $L \leftarrow L \cup L_1^{(0)}$
until $|L| \geq 1$

Analysis of Algorithm 4. For the memory complexity of the algorithm we now have to additionally consider the memory required by the dissection, namely the memory is given by

$$\tilde{\mathcal{O}}\left(\max_i(\mathcal{F}_i, M_{\text{Dissection}})\right).$$

However, we parameterize the dissection procedure with $\lambda = \log_{|S^{(3)}|} \max_i(\mathcal{F}_i)$, which gives $M_{\text{Dissection}} = \max_i(\mathcal{F}_i)$ (compare to Lemma 1). Hence the memory complexity is given as before as

$$M = \tilde{\mathcal{O}}\left(\max_i(\mathcal{F}_i)\right).$$

Now let T_i be the time required to construct the subtree $\mathfrak{T}_i$ once. Then the total time complexity of the algorithm is given by

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

The time to construct each subtree is linear in the involved list sizes. Note that the subtrees $\mathfrak{T}_2$ (resp. $\mathfrak{T}_1$) can be constructed from the already existing *filtered* level-3 (resp. level-2) lists. Hence we obtain

$$\begin{aligned} T_3 &= \max(T_{\text{Dissection}}, \mathcal{L}_3, \mathcal{L}_2, \mathcal{L}_1, \mathcal{L}_0) \\ T_2 &= \max(\mathcal{F}_3, \mathcal{L}_2, \mathcal{L}_1, \mathcal{L}_0) \\ T_1 &= \max(\mathcal{F}_2, \mathcal{L}_1, \mathcal{L}_0), \end{aligned}$$

where $T_{\text{Dissection}}$ denotes the time complexity to construct the level-3 list via the dissection algorithm given a memory of M, which is specified by Lemma 1.

The correctness mostly follows from the previous algorithm, except from the fact that here we do not guarantee that the last list contains a solution. Instead we construct the final list using multiple randomized executions. In contrast to completely independent runs, we only partially update certain constraints to enhance efficiency. However, a standard assumption in the analysis of representation based algorithms is that representations distribute uniformly and independently over all possible constraints. Employing this assumption, as long as any constraint in the tree is changed, the probability of a success is independent between runs. Now, to ensure that the final list over all randomized executions contains at least one element we obtain the restriction

$$q^{t_3+t_2+t_1} \cdot \mathcal{F}_0 \geq 1.$$

Hence we choose the t_i's in the following way to fulfill the constraint

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

where $r = 4r_3 + 2r_2 + r_1$. Note that this choice implies that constraints on the lowest levels are exchanged before those on higher levels.

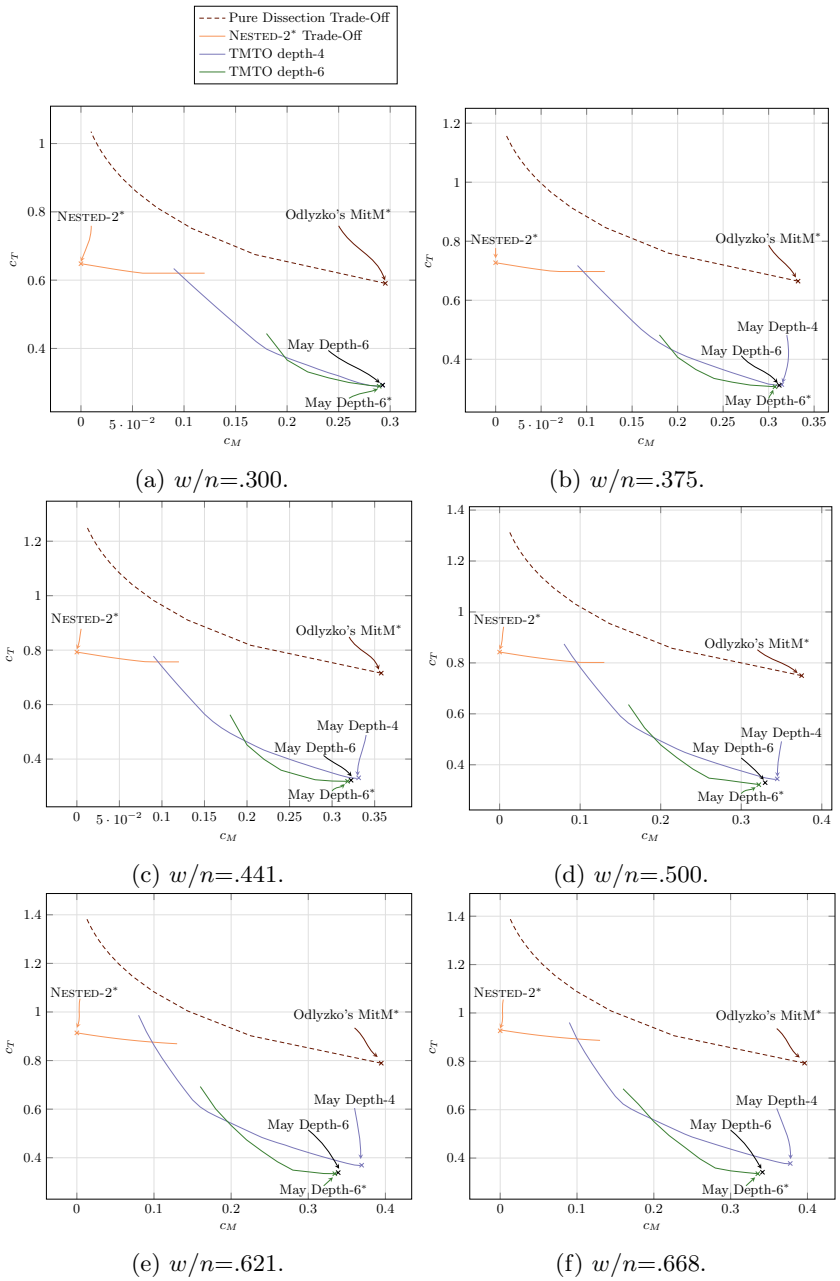
For an arbitrary depth-d, the choice of t_i can be generalized as

$$t_i = \max((\sum_{j=1}^{d-i}(2^{d-j} - 1) \cdot \ell_{d-j}) - r) - (\sum_{1}^{d-i-1} t_{d-j}), \tag{19}$$

where $r = \sum_{i=1}^{d-1} 2^{i-1} \cdot r_i$.

In Fig. 17 we illustrate the runtime exponent of Algorithm 4 as a blue line. Additionally, those plots also depict a generalization of the trade-off to depth 6 for which we give the full details in Appendix D (green line), as well as the PCS-based trade-off using Nested-2* from Section 7.1 (orange line). For comparison we include the time-memory trade-off resulting from a direct application of the Dissection framework (see Lemma 1[9]). Note that the starting point of this direct

[9] In such an application we have $S^{(3)} = \tau^n(w/2)$ with respect to Lemma 1.



(a) w/n=.300. (b) w/n=.375.

(c) w/n=.441. (d) w/n=.500.

(e) w/n=.621. (f) w/n=.668.

Fig. 17: Continuous Time-Memory Trade-Off curves for different weights. Here the running time is of the form $T = 2^{c_T n + o(n)}$ and the memory consumption is $2^{c_M n + o(n)}$.

application is a Schroeppel-Shamir enhanced version of the basic MitM, which obtains the same running time as Odlyzko's MitM but requires only as square-root of the memory (labeled *Odlyzko's MitM** in the figures).

For rather high amounts of memory we find the depth-6 trade-off to obtain the best running time. For unlimited memory this trade-off starts from our variant of May's algorithm in depth 6 that allows for flexible ℓ_i, which we label *May Depth-6** in the figures. For mid-range memory size the depth-4 trade-off offers faster instantiations, while in the low memory regime the PCS-based trade-off relying on NESTED-2* is superior. Overall, the new trade-offs outperform the application of the Dissection framework for any memory significantly.

8 Conclusion

In this study, we have proposed an enhanced memoryless algorithm for small max-norm LWE, utilizing the Nested-Collision-Search technique introduced in [16]. Our research provides a novel time-memory trade-off for small exponential memory, achieved by combining our memoryless algorithm with the parallel collision search technique [43]. Moreover, we have introduced a time-memory trade-off using the enumeration-based algorithm in [35] and the technique introduced in [23]. Collectively, these results yield a continuous trade-off curve (see Fig. 17) spanning from zero to unlimited exponential memory, offering comprehensive insight into the hardness of small max-norm LWE across the full memory spectrum.

This work extends beyond providing a thorough asymptotic analysis of the problem. It also suggests potential integration with lattice reduction techniques, contributing to more effective hybrid attacks. Recent advancements in this area, as indicated in [8, 28, 45], demonstrate that our combinatorial techniques are poised to play a critical role in future attack methodologies.

Acknowledgements. We would like to express our sincere gratitude to Rahul Girme for insightful discussions and his contributions in the initial stages of this work. We also like to thank the anonymous reviewers of this work for their constructive comments and suggestions that led to this improved version.

References

1. Adj, G., Cervantes-Vázquez, D., Chi-Domínguez, J.J., Menezes, A., Rodríguez-Henríquez, F.: On the cost of computing isogenies between supersingular elliptic curves. In: Cid, C., Jacobson Jr:, M.J. (eds.) SAC 2018. LNCS, vol. 11349, pp. 322–343. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-10970-7_15
2. Albrecht, M.R., Bai, S., Ducas, L.: A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 153–178. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_6

3. Albrecht, M.R., Bai, S., Fouque, P.A., Kirchner, P., Stehlé, D., Wen, W.: Faster enumeration-based lattice reduction: Root hermite factor $k^{1/(2k)}$ time $k^{k/8+o(k)}$. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 186–212. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_7
4. 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
5. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_31
6. Bellini, E., Chavez-Saab, J., Chi-Domínguez, J.J., Esser, A., Ionica, S., Rivera-Zamarripa, L., Rodríguez-Henríquez, F., Trimoska, M., Zweydinger, F.: Parallel isogeny path finding with limited memory. In: Progress in Cryptology–INDOCRYPT 2022: 23rd International Conference on Cryptology in India, Kolkata, India, December 11–14, 2022, Proceedings. pp. 294–316. Springer (2023)
7. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: NTRU prime: Reducing attack surface at low cost. In: Adams, C., Camenisch, J. (eds.) SAC 2017. LNCS, vol. 10719, pp. 235–260. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-72565-9_12
8. Bi, L., Lu, X., Luo, J., Wang, K.: Hybrid dual and meet-lwe attack. In: Information Security and Privacy: 27th Australasian Conference, ACISP 2022, Wollongong, NSW, Australia, November 28–30, 2022, Proceedings. pp. 168–188. Springer (2022)
9. 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
10. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber: a cca-secure module-lattice-based kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018)
11. Bos, J.W., Kaihara, M.E., Kleinjung, T., Lenstra, A.K., Montgomery, P.L.: Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction. International Journal of Applied Cryptography **2**(3), 212–228 (2012)
12. 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
13. Carrier, K., Hatey, V., Tillich, J.: Projective space stern decoding and application to sdith. IACR Cryptol. ePrint Arch. p. 1865 (2023), https://eprint.iacr.org/2023/1865
14. 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
15. 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
16. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Memory-efficient algorithms for finding needles in haystacks. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 185–206. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53008-5_7
17. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_3
18. 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
19. Esser, A., Girme, R., Mukherjee, A., Sarkar, S.: Memory-efficient attacks on small LWE keys. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 14441, pp. 72–105. Springer (2023). https://doi.org/10.1007/978-981-99-8730-6_3, https://doi.org/10.1007/978-981-99-8730-6_3
20. 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
21. 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
22. Esser, A., Santini, P.: Not just regular decoding: Asymptotics and improvements of regular syndrome decoding attacks. IACR Cryptol. ePrint Arch. p. 1568 (2023), https://eprint.iacr.org/2023/1568
23. Esser, A., Zweydinger, F.: New time-memory trade-offs for subset sum - improving ISD in theory and practice. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 360–390. Springer (2023). https://doi.org/10.1007/978-3-031-30589-4_13, https://doi.org/10.1007/978-3-031-30589-4_13
24. Gama, N., Nguyen, P.Q., Regev, O.: Lattice enumeration using extreme pruning. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 257–278. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_13
25. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 169–178. ACM Press (May / Jun 2009). https://doi.org/10.1145/1536414.1536440
26. Glaser, T., May, A.: How to enumerate lwe keys as narrow as in kyber/dilithium. In: International Conference on Cryptology and Network Security. pp. 75–100. Springer (2023)
27. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: A signature scheme for embedded systems. In: Prouff, E., Schaumont, P.

(eds.) CHES 2012. LNCS, vol. 7428, pp. 530–547. Springer, Heidelberg (Sep 2012). https://doi.org/10.1007/978-3-642-33027-8_31
28. Hhan, M., Kim, J., Lee, C., Son, Y.: How to meet ternary lwe keys on babai's nearest plane. Cryptology ePrint Archive (2022)
29. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: Third Algorithmic Number Theory Symposium (ANTS). LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (Jun 1998)
30. Howgrave-Graham, N.: A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 150–169. Springer, Heidelberg (Aug 2007). https://doi.org/10.1007/978-3-540-74143-5_9
31. 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
32. Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P.: High-speed key encapsulation from NTRU. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 232–252. Springer, Heidelberg (Sep 2017). https://doi.org/10.1007/978-3-319-66787-4_12
33. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_43
34. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_1
35. 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
36. 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
37. Nguyen, D.H., Nguyen, T.T., Duong, T.N., Pham, P.H.: Cryptanalysis of md5 on gpu cluster. In: Proceedings of International Conference on Information Security and Artificial Intelligence. vol. 2, pp. 910–914 (2010)
38. Niederhagen, R., Ning, K.C., Yang, B.Y.: Implementing joux-vitse's crossbred algorithm for solving $\mathcal{MQ}$ systems over $\mathbb{F}_2$ on GPUs. In: Lange, T., Steinwandt, R. (eds.) Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018. pp. 121–141. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-79063-3_6
39. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 333–342. ACM Press (May / Jun 2009). https://doi.org/10.1145/1536414.1536461
40. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)
41. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060603
42. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS,

vol. 5912, pp. 617–635. Springer, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10366-7_36

43. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. Journal of Cryptology **12**(1), 1–28 (Jan 1999). https://doi.org/10.1007/PL00003816
44. van Vredendaal, C.: Reduced memory meet-in-the-middle attack against the ntru private key. LMS Journal of Computation and Mathematics **19**(A), 43–57 (2016). https://doi.org/10.1112/S1461157016000206
45. Zhu, H., Kamada, S., Kudo, M., Takagi, T.: Improved hybrid attack via error-splitting method for finding quinary short lattice vectors. In: Shikata, J., Kuzuno, H. (eds.) Advances in Information and Computer Security - 18th International Workshop on Security, IWSEC 2023, Yokohama, Japan, August 29-31, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14128, pp. 117–136. Springer (2023). https://doi.org/10.1007/978-3-031-41326-1_7, https://doi.org/10.1007/978-3-031-41326-1_7

Appendix

A Different Analysis of Algorithm 2

Another way to derive the time complexity of Algorithm 2 is via directly computing the probability that the sampled tuple $(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4)$ sums to a ternary vector. That is

$$\begin{aligned} c &:= \Pr\left[\mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3 + \mathbf{y}_4 \in \tau^n(w/2)\right] \\ &\geq \Pr\left[\mathbf{a}_1 \in \mathcal{D}_1 \cap \mathbf{a}_2 \in \mathcal{D}_2 \cap \mathbf{a}_1 + \mathbf{a}_2 \in \tau^n(w/2)\right] \\ &= \underbrace{\Pr\left[\mathbf{a}_1 \in \mathcal{D}_1\right]}_{c_1} \cdot \underbrace{\Pr\left[\mathbf{a}_2 \in \mathcal{D}_2\right]}_{c_2} \cdot \underbrace{\Pr\left[\mathbf{a}_1 + \mathbf{a}_2 \in \tau^n(w/2) \mid \mathbf{a}_i \in \mathcal{D}_i\right]}_{c_3}, \end{aligned}$$

where the last equality follows from the fact that $\mathbf{a}_1$ and $\mathbf{a}_2$ are independent. Now since every element from $\mathcal{D}_1$ has R_1 representations from $\mathcal{T}_1 \times \mathcal{T}_2$ the probability that a random element from $\mathcal{T}_1 \times \mathcal{T}_2$ sums to an $\mathbf{a}_1 \in \mathcal{D}_1$ is

$$c_1 = \frac{R_1 \cdot |\mathcal{D}_1|}{|\mathcal{T}_1 \times \mathcal{T}_2|} = \frac{R_1 \cdot |\mathcal{D}_1|}{|\mathcal{T}_1|^2}.$$

By the same argument we have

$$c_3 = \frac{R_2 \cdot |\tau^n(w/2)|}{|\mathcal{D}_1 \times \mathcal{D}_2|} = \frac{R_2 \cdot |\tau^n(w/2)|}{|\mathcal{D}_1|^2},$$

as there are R_2 representations of every element from $\tau^n(w/2)$ as sum of elements from $\mathcal{D}_1, \mathcal{D}_2$. Further since we choose function domains (and resp. mid level domains) of same size we have $c_2 = c_1$. To be able to find the solution we still need to guess the correct $\mathbf{e}' = \pi_{2\ell}(\mathbf{e})$ and in every iteration we need to perform a collision search between g_1, g_2, which amounts to

$$T = \left((c_1)^2 c_3 \cdot q_3\right)^{-1} \cdot q^\ell = \left(\frac{\binom{n}{w/2,w/2,.}^{\frac{3}{2}}}{(R_1)^2 \cdot R_2} \right)^{1+o(1)},$$

using the fact that $|\mathcal{T}_i| = q^\ell = \sqrt{|\tau^n(w/2)|}$ (compare to Eq. (7)).

B Extension to Kyber and Dilithium

For a vector $\mathbf{s} \in \{-m, \ldots, m\}^n$ let $w_i = |\mathbf{s}|_i$, where $|\mathbf{s}|_i := |\{j \mid s_j = i\}|$, be the amount of its coordinates equal to i. For a solution $\mathbf{s} \in \{-m, \ldots, m\}^n$ the analysis of Algorithm 1 and Algorithm 2 requires knowledge about w_i for $i = -m, \ldots, m$. Kyber and Dilithium sample $\mathbf{s}$ from some distribution D, which does not provide direct information on w_i. However, May and Glaser have recently shown how to

re-randomize keys from probabilistic distributions [26].[10] Their technique allows with subexponential overhead to fix the w_i to their expectation, i.e., $w_i = n \cdot p_i$ where $p_i := \Pr_{X \sim D}[X = i]$.

In our following analysis we make use of this re-randomization approach and therefore assume that the $w_i's$ are known. Furthermore, for all settings we have $w_i = w_{-i}$.

We extend the definition of $\tau_2^n(a, b)$ naturally to $\tau_3^n(a, b, c)$, where c denotes the amount of ± 3 entries in each element $\mathbf{v} \in \tau_3^n(a, b, c)$. To apply Algorithms 1 and 2 we now adapt the final check of the repeat loops to look for a solution $\mathbf{s} \in \tau_3^n(w_1, w_2, w_3)$ rather than $\tau^n(w/2)$.

B.1 Analysis of Algorithm 1 using van Vredendaal instantiation

As shown in Section 3 the running time of Algorithm 1 using the van Vredendaal instantiation is always $T = D^{\frac{3}{4}}$, where D is the search space. The search space in our case with solution $\mathbf{s} \in \{-3, \ldots, 3\}^n$ is

$$D = |\tau_3^n(w_1, w_2, w_3)| = \binom{n}{w_1, w_1, w_2, w_2, w_3, w_3, \cdot}.$$

B.2 Analysis of Algorithm 1 using Rep-3 representations

As before, let the number of $\pm 1, \pm 2, \pm 3$ entries in the final solution be w_1, w_2, w_3 respectively. We define the function domains as $\tau_3^n(n_1, n_2, n_3)$, where we determine n_i later.

Therefore the solution is constructed as a sum $\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$ with $\mathbf{s}_i \in \tau_3^n(n_1, n_2, n_3)$. Analogous to the analysis of our NESTED-2^* instantiation, we define how often we expect each possible representation to appear in the sum

[10] Their technique works whenever the secret and the error of the LWE instance follow the same distribution and exploits a similar strategy as our uniform secret improvement from Section 4.4.

$\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$ as

$$
\begin{aligned}
0: &\quad \underbrace{0+0}_{m},\ \underbrace{1-1}_{z_1},\ \underbrace{-1+1}_{z_1},\ \underbrace{2-2}_{z_2},\ \underbrace{-2+2}_{z_2},\ \underbrace{3-3}_{z_3},\ \underbrace{-3+3}_{z_3},\\
1: &\quad \underbrace{1+0}_{\frac{w_1}{2}-o_1-o_2},\ \underbrace{0+1}_{\frac{w_1}{2}-o_1-o_2},\ \underbrace{2-1}_{o_1},\ \underbrace{-1+2}_{o_1},\ \underbrace{3-2}_{o_2},\ \underbrace{-2+3}_{o_2},\\
-1: &\quad \underbrace{-1+0}_{\frac{w_1}{2}-o_1-o_2},\ \underbrace{0-1}_{\frac{w_1}{2}-o_1-o_2},\ \underbrace{-2+1}_{o_1},\ \underbrace{1-2}_{o_1},\ \underbrace{-3+2}_{o_2},\ \underbrace{2-3}_{o_2},\\
2: &\quad \underbrace{2+0}_{\frac{w_2-t}{2}-t_1},\ \underbrace{0+2}_{\frac{w_2-t}{2}-t_1},\ \underbrace{1+1}_{t},\ \underbrace{3-1}_{t_1},\ \underbrace{-1+3}_{t_1},\\
-2: &\quad \underbrace{-2+0}_{\frac{w_2-t}{2}-t_1},\ \underbrace{0-2}_{\frac{w_2-t}{2}-t_1},\ \underbrace{-1-1}_{t},\ \underbrace{-3+1}_{t_1},\ \underbrace{1-3}_{t_1},\\
3: &\quad \underbrace{3+0}_{\frac{w_3}{2}-r},\ \underbrace{0+3}_{\frac{w_3}{2}-r},\ \underbrace{2+1}_{r},\ \underbrace{1+2}_{r},\\
-3: &\quad \underbrace{-3+0}_{\frac{w_3}{2}-r},\ \underbrace{0-3}_{\frac{w_3}{2}-r},\ \underbrace{-2-1}_{r},\ \underbrace{-1-2}_{r},
\end{aligned}
$$

where $m := n - 2(w_1 + w_2 + w_3 + z_1 + z_2 + z_3)$, and the $z_1, z_2, z_3, o_1, o_2, t, t_1, r$ are optimization parameters. From here we can derive the number of representations as

$$
R = \binom{n-2(w_1+w_2+w_3)}{m, z_1, z_1, z_2, z_2, z_3, z_3} \binom{w_1}{\frac{w_1}{2}-o_1-o_2, \frac{w_1}{2}-o_1-o_2, o_1, o_1, o_2, o_2}^2 \binom{w_2}{\frac{w_2-t}{2}-t_1, \frac{w_2-t}{2}-t_1, t, t_1, t_1}^2 \binom{w_3}{\frac{w_3}{2}-r, \frac{w_3}{2}-r, r, r}^2,
$$

where the first term counts the representations of 0, the second those of ± 1, the third those of ± 2 and the last those of ± 3 coordinates. A counting argument yields the necessary number of $0, \pm 1, \pm 2$ and ± 3 coordinates in the function domains as

$$
\begin{aligned}
n_1 &= z_1 + \frac{w_1}{2} - o_1 - o_2 + o_1 + t + t_1 + r = z_1 + t + t_1 + r - o_2 + \frac{w_1}{2}\\
n_2 &= z_2 + o_1 + o_2 + \frac{w_2 - t}{2} - t_1 + r\\
n_3 &= z_3 + o_2 + t_1 + \frac{w_3}{2} - r.
\end{aligned}
$$

The function domain size constitutes as

$$
|\mathcal{T}| = \binom{n}{n_1, n_1, n_2, n_2, n_3, n_3, .},
$$

while the time complexity is still given as $T = \tilde{\mathcal{O}}\left(\mathcal{T}^{3/2}/R\right)$ (compare to Section 3).

B.3 Analysis of Algorithm 2 using Nested-3* instantiation

We define the function domains analogous to the NESTED-2* instantiation from Section 4.3 (compare to Fig. 10). In contrast to that definition of function domains we use vectors from $\tau_3^{\gamma n}(n_1, n_2, n_3)$ for the joint part, while we adapt the disjoint part according to the respective distribution (detailed later). Again we use a permutation to shift weight into the disjoint part of length $(1-\gamma)n$. Let the permutation distribute β_1-fraction of the expected number of ± 1s, a β_2-fraction of ± 2s, and a β_3-fraction of ± 3s to the γn part and shift the rest into the disjoint $(1-\gamma)n$ part. Then the respective number of $\pm 1, \pm 2$ and ± 3 after the permutation on the joint parts are $\hat{w}_1 = \gamma\beta_1 w_1$, $\hat{w}_2 = \gamma\beta_2 w_2$ and $\hat{w}_3 = \gamma\beta_3 w_3$.

Each of the four disjoint parts of length $(1-\gamma)n/4$ will then have $(w_1 - \hat{w}_1)/4$ many 1s (resp. -1s), $(w_2 - \hat{w}_2)/4$ many 2s (resp. -2s), $(w_3 - \hat{w}_3)/4$ many 3s (resp. -3s). Hence, the total weight in each of the four disjoint parts of length $(1-\gamma)n/4$ is

$$W_{disjoint} = \frac{1}{4}\left(2(w_1 - \hat{w}_1) + 2(w_2 - \hat{w}_2) + 2(w_3 - \hat{w}_3)\right),$$

which implies $W_{disjoint} \leq (1-\gamma)n/4$.

Now we define the mid-level domains as $\tau_3^n(n_1^{\text{mid}}, n_2^{\text{mid}}, n_3^{\text{mid}})$. The number of representations of the final solution as sum of elements from the mid-level domains can then be described in the following way.

$$
\begin{aligned}
0: &\quad \underbrace{0+0}_{m^{\text{mid}}}, \quad \underbrace{1-1}_{z_1^{\text{mid}}}, \quad \underbrace{-1+1}_{z_1^{\text{mid}}}, \quad \underbrace{2-2}_{z_2^{\text{mid}}}, \quad \underbrace{-2+2}_{z_2^{\text{mid}}}, \quad \underbrace{3-3}_{z_3^{\text{mid}}}, \quad \underbrace{-3+3}_{z_3^{\text{mid}}},\\
1: &\quad \underbrace{1+0}_{\frac{\hat{w}_1}{2}-o_1^{\text{mid}}-o_2^{\text{mid}}}, \quad \underbrace{0+1}_{\frac{\hat{w}_1}{2}-o_1^{\text{mid}}-o_2^{\text{mid}}}, \quad \underbrace{2-1}_{o_1^{\text{mid}}}, \quad \underbrace{-1+2}_{o_1^{\text{mid}}}, \quad \underbrace{3-2}_{o_2^{\text{mid}}}, \quad \underbrace{-2+3}_{o_2^{\text{mid}}},\\
-1: &\quad \underbrace{-1+0}_{\frac{\hat{w}_1}{2}-o_1^{\text{mid}}-o_2^{\text{mid}}}, \quad \underbrace{0-1}_{\frac{\hat{w}_1}{2}-o_1^{\text{mid}}-o_2^{\text{mid}}}, \quad \underbrace{-2+1}_{o_1^{\text{mid}}}, \quad \underbrace{1-2}_{o_1^{\text{mid}}}, \quad \underbrace{-3+2}_{o_2^{\text{mid}}}, \quad \underbrace{2-3}_{o_2^{\text{mid}}},\\
2: &\quad \underbrace{2+0}_{\frac{\hat{w}_2-t^{\text{mid}}}{2}-t_1^{\text{mid}}}, \quad \underbrace{0+2}_{\frac{\hat{w}_2-t^{\text{mid}}}{2}-t_1^{\text{mid}}}, \quad \underbrace{1+1}_{t^{\text{mid}}}, \quad \underbrace{3-1}_{t_1^{\text{mid}}}, \quad \underbrace{-1+3}_{t_1^{\text{mid}}},\\
-2: &\quad \underbrace{-2+0}_{\frac{\hat{w}_2-t^{\text{mid}}}{2}-t_1^{\text{mid}}}, \quad \underbrace{0-2}_{\frac{\hat{w}_2-t^{\text{mid}}}{2}-t_1^{\text{mid}}}, \quad \underbrace{-1-1}_{t^{\text{mid}}}, \quad \underbrace{-3+1}_{t_1^{\text{mid}}}, \quad \underbrace{1-3}_{t_1^{\text{mid}}},\\
3: &\quad \underbrace{3+0}_{\frac{\hat{w}_3}{2}-r^{\text{mid}}}, \quad \underbrace{0+3}_{\frac{\hat{w}_3}{2}-r^{\text{mid}}}, \quad \underbrace{2+1}_{r^{\text{mid}}}, \quad \underbrace{1+2}_{r^{\text{mid}}},\\
-3: &\quad \underbrace{-3+0}_{\frac{\hat{w}_3}{2}-r^{\text{mid}}}, \quad \underbrace{0-3}_{\frac{\hat{w}_3}{2}-r^{\text{mid}}}, \quad \underbrace{-2-1}_{r^{\text{mid}}}, \quad \underbrace{-1-2}_{r^{\text{mid}}},
\end{aligned}
$$

where $m^{\text{mid}} := \gamma n - 2(\hat{w}_1 + \hat{w}_2 + \hat{w}_3 + z_1^{\text{mid}} + z_2^{\text{mid}} + z_3^{\text{mid}})$, and the optimization parameters are $z_1^{\text{mid}}, z_2^{\text{mid}}, z_3^{\text{mid}}, o_1^{\text{mid}}, o_2^{\text{mid}}, t^{\text{mid}}, t_1^{\text{mid}}, r^{\text{mid}}$. From here we can

derive the number of representations as

$$\begin{aligned} R_2 =& \binom{\gamma n - 2(\hat{w}_1 + \hat{w}_2 + \hat{w}_3)}{m^{\text{mid}}, z_1^{\text{mid}}, z_1^{\text{mid}}, z_2^{\text{mid}}, z_2^{\text{mid}}, z_3^{\text{mid}}, z_3^{\text{mid}}} \\ &\cdot \binom{\hat{w}_1}{\frac{\hat{w}_1}{2} - o_1^{\text{mid}} - o_2^{\text{mid}}, \frac{\hat{w}_1}{2} - o_1^{\text{mid}} - o_2^{\text{mid}}, o_1^{\text{mid}}, o_1^{\text{mid}}, o_2^{\text{mid}}, o_2^{\text{mid}}}^2 \\ &\cdot \binom{\hat{w}_2}{\frac{\hat{w}_2 - t^{\text{mid}}}{2} - t_1^{\text{mid}}, \frac{\hat{w}_2 - t^{\text{mid}}}{2} - t_1^{\text{mid}}, t^{\text{mid}}, t_1^{\text{mid}}, t_1^{\text{mid}}}^2 \\ &\cdot \binom{\hat{w}_3}{\frac{\hat{w}_3}{2} - r^{\text{mid}}, \frac{\hat{w}_3}{2} - r^{\text{mid}}, r^{\text{mid}}, r^{\text{mid}}}^2, \end{aligned}$$

where the first term counts the representations of 0, the second those of ± 1, the third those of ± 2 and the last those of ± 3 coordinates. A counting argument yields the necessary number of ± 1, ± 2 and ± 3 coordinates in the γn part of the mid level summands as

$$\begin{aligned} n_1^{\text{mid}} &= z_1^{\text{mid}} + \frac{\hat{w}_1}{2} - o_1^{\text{mid}} - o_2^{\text{mid}} + o_1^{\text{mid}} + t^{\text{mid}} + t_1^{\text{mid}} + r^{\text{mid}} \\ &= z_1^{\text{mid}} + t^{\text{mid}} + t_1^{\text{mid}} + r^{\text{mid}} - o_2^{\text{mid}} + \frac{\hat{w}_1}{2}, \\ n_2^{\text{mid}} &= z_2^{\text{mid}} + o_1^{\text{mid}} + o_2^{\text{mid}} + \frac{\hat{w}_2 - t^{\text{mid}}}{2} - t_1^{\text{mid}} + r^{\text{mid}} \text{ and} \\ n_3^{\text{mid}} &= z_3^{\text{mid}} + o_2^{\text{mid}} + t_1^{\text{mid}} + \frac{\hat{w}_3}{2} - r^{\text{mid}}. \end{aligned}$$

The number of representations of elements from the mid-level domains as sums of base-level elements is described as follows.

$$\begin{aligned}
0: &\quad \underbrace{0+0}_{m}, \quad \underbrace{1-1}_{z_1}, \quad \underbrace{-1+1}_{z_1}, \quad \underbrace{2-2}_{z_2}, \quad \underbrace{-2+2}_{z_2}, \quad \underbrace{3-3}_{z_3}, \quad \underbrace{-3+3}_{z_3}, \\
1: &\quad \underbrace{1+0}_{\frac{n_1^{\text{mid}}}{2} - o_1 - o_2}, \quad \underbrace{0+1}_{\frac{n_1^{\text{mid}}}{2} - o_1 - o_2}, \quad \underbrace{2-1}_{o_1}, \quad \underbrace{-1+2}_{o_1}, \quad \underbrace{3-2}_{o_2}, \quad \underbrace{-2+3}_{o_2}, \\
-1: &\quad \underbrace{-1+0}_{\frac{n_1^{\text{mid}}}{2} - o_1 - o_2}, \quad \underbrace{0-1}_{\frac{n_1^{\text{mid}}}{2} - o_1 - o_2}, \quad \underbrace{-2+1}_{o_1}, \quad \underbrace{1-2}_{o_1}, \quad \underbrace{-3+2}_{o_2}, \quad \underbrace{2-3}_{o_2}, \\
2: &\quad \underbrace{2+0}_{\frac{n_2^{\text{mid}} - t}{2} - t_1}, \quad \underbrace{0+2}_{\frac{n_2^{\text{mid}} - t}{2} - t_1}, \quad \underbrace{1+1}_{t}, \quad \underbrace{3-1}_{t_1}, \quad \underbrace{-1+3}_{t_1}, \\
-2: &\quad \underbrace{-2+0}_{\frac{n_2^{\text{mid}} - t}{2} - t_1}, \quad \underbrace{0-2}_{\frac{n_2^{\text{mid}} - t}{2} - t_1}, \quad \underbrace{-1-1}_{t}, \quad \underbrace{-3+1}_{t_1}, \quad \underbrace{1-3}_{t_1}, \\
3: &\quad \underbrace{3+0}_{\frac{n_3^{\text{mid}}}{2} - r}, \quad \underbrace{0+3}_{\frac{n_3^{\text{mid}}}{2} - r}, \quad \underbrace{2+1}_{r}, \quad \underbrace{1+2}_{r}, \\
-3: &\quad \underbrace{-3+0}_{\frac{n_3^{\text{mid}}}{2} - r}, \quad \underbrace{0-3}_{\frac{n_3^{\text{mid}}}{2} - r}, \quad \underbrace{-2-1}_{r}, \quad \underbrace{-1-2}_{r},
\end{aligned}$$

where $m := n - 2(n_1^{\text{mid}} + n_2^{\text{mid}} + n_3^{\text{mid}} + z_1 + z_2 + z_3)$, and the optimization parameters are $z_1, z_2, z_3, o_1, o_2, t, t_1, r$. From here we can derive the number of representations as

$$R_1 = \binom{\gamma n - 2(n_1^{\text{mid}} + n_2^{\text{mid}} + n_3^{\text{mid}})}{m, z_1, z_1, z_2, z_2, z_3, z_3} \binom{n_1^{\text{mid}}}{\frac{n_1^{\text{mid}}}{2} - o_1 - o_2, \frac{n_1^{\text{mid}}}{2} - o_1 - o_2, o_1, o_1, o_2, o_2}^2 \binom{n_2^{\text{mid}}}{\frac{n_2^{\text{mid}}-t}{2} - t_1, \frac{n_2^{\text{mid}}-t}{2} - t_1, t, t_1, t_1}^2 \binom{n_3^{\text{mid}}}{\frac{n_3^{\text{mid}}}{2} - r, \frac{n_3^{\text{mid}}}{2} - r, r, r}^2,$$

where the first term again counts the representations of 0, the second those of ± 1, the third those of ± 2 and the last those of ± 3 coordinates. Counting yields the necessary number of $\pm 1, \pm 2$ and ± 3 on the base level as

$$\begin{aligned} n_1 &= z_1 + \frac{n_1^{\text{mid}}}{2} - o_1 - o_2 + o_1 + t + t_1 + r = z_1 + t + t_1 + r - o_2 + \frac{n_1^{\text{mid}}}{2}, \\ n_2 &= z_2 + o_1 + o_2 + \frac{n_2^{\text{mid}} - t}{2} - t_1 + r \text{ and} \\ n_3 &= z_3 + o_2 + t_1 + \frac{n_3^{\text{mid}}}{2} - r. \end{aligned}$$

The function domain size is given as

$$|\mathcal{T}_i| = \binom{\gamma n}{n_1, n_1, n_2, n_2, n_3, n_3, \cdot} \binom{(1-\gamma)n/4}{\alpha_1 w_1, \alpha_1 w_1, \alpha_2 w_2, \alpha_2 w_2, \alpha_3 w_3, \alpha_3 w_3, \cdot},$$

where $\alpha_i := (1 - \gamma\beta_i)/4$, for $i = 1, 2, 3$, while the search space after permutation is of size

$$|D| = \binom{\gamma n}{\hat{w}_1, \hat{w}_1, \hat{w}_2, \hat{w}_2, \hat{w}_3, \hat{w}_3, \cdot} \binom{(1-\gamma)n/4}{\alpha_1 w_1, \alpha_1 w_1, \alpha_2 w_2, \alpha_2 w_2, \alpha_3 w_3, \alpha_3 w_3, \cdot}^4,$$

Hence, the probability of achieving the desired weight permutation is

$$q_4 = \frac{|D|}{\tau_3^n(w_1, w_2, w_3)}.$$

In our numerical optimization we again ensure that $|\mathcal{T}_i| = \sqrt{|D|}$, implying that any collision that lies in D is a solution to the LWE problem. Eventually, the time complexity is given as (compare to Eq. (11))

$$T = (q_1 q_2 q_3 q_4)^{-1} q^\ell = \left(\frac{|D|^{\frac{1}{2}} \cdot |\tau_3^n(w_1, w_2, w_3)|}{(R_1)^2 R_2} \right)^{1+o(1)}.$$

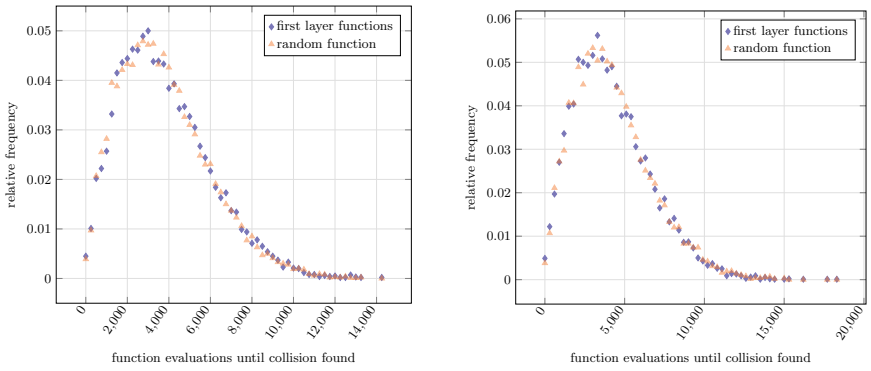
C Experimental Verification of the Randomness Assumption

In this section, we verify the heuristic assumptions on the random behavior of the functions on which our analysis is based by small scale experiments. A

python and a C++ implementation of the following experiments are available at https://github.com/arindamIITM/Small-LWE-Keys.

Note that the core assumption of our analysis is that the constructed functions behave like random functions with respect to collision search. This random behavior is necessary as it implies multiple essential properties. First, the number of function calls to find a collision via standard procedures is roughly $\sqrt{|D|}$, where D is the function domain. Further, there exist a total of about $|D|$ collisions and to find one out of R distinguished collisions on average D/R collisions have to be sampled. Clearly, those properties are not independent, as for example the amount of total collisions clearly impacts the amount of collisions needed to find a distinguished one. We therefore verify those properties jointly in our experiments.

Our experiments focus on verifying the random behavior of the first layer functions. That is the functions f_i from Section 4. Recall that the second layer functions, i.e., the functions g_j, are defined on top of the first layer functions by mapping starting points to collisions. Hence, those functions are independent of the concrete definition of the first layer functions and their randomness translates from the random distribution of collisions in the functions f_i, which we test in the experiments. Further, an experiment that verifies the functionality of the nested collision search, given that the first layer functions satisfy the randomness assumptions, is given in [20].



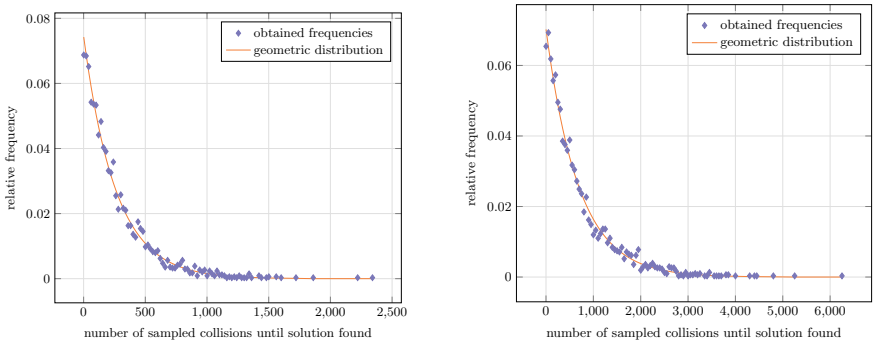
(a) $n = 21, w = 4, w_s = 6$, Sample size 10,000, random function: $(\mathbb{E}, \sigma) = (3891, 2169)$, First layer functions: $(\mathbb{E}, \sigma) = (3939, 2195)$

(b) $n = 32, w = 3, w_s = 4$, Sample size 10,000, random function: $(\mathbb{E}, \sigma) = (4393, 2428)$, First layer functions: $(\mathbb{E}, \sigma) = (4348, 2411)$

Fig. 18: Number of function evaluations needed to find collisions in the first layer functions vs. in random functions.

For our experiments we take as function domains $\tau^n(w)$, i.e., length-n vectors with w entries equal to one and w entries equal to minus one. The secret is chosen from $\tau^n(w_s)$. For our experiments we fix $q = 16$.

Finding a collision. We verify that a collision search on the functions f_i does not take more time than a collision search on random functions. Therefore, we first fix the LWE instance and therefore the precise definition of the f_i. Then, we repeatedly search for collisions in the functions f_i and keep track of the number of function evaluations needed to find the collisions. In a next step, we repeat this experiment but exchange the functions against a random function. Fig. 18 shows both obtained distributions for different parameters. Despite from the obvious visual match in distributions, we find that expectation and standard deviation are also very close.



(a) $n = 21, w = 4, w_s = 6$, sample size 3,375, $\sigma = 258$, $\sigma_{\text{Geo}} = 259$

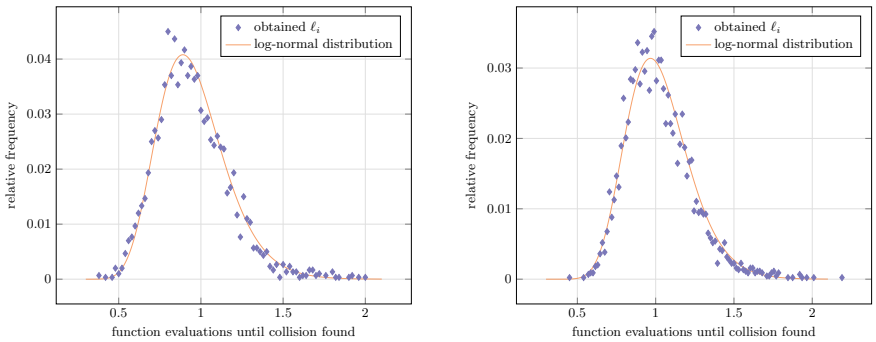
(b) $n = 32, w = 3, w_s = 4$, sample size 3,100, $\sigma = 689$, $\sigma_{\text{Geo}} = 687$

Fig. 19: Distribution of the amount of collision sampled before finding the solution.

Finding a distinguished collision. In a second experiment we measure the amount of collisions we need to sample from (randomly flavoured versions of) the function to find one of the R representations of the solution. In case the functions behave like random functions where R out of all $|D|$ collisions are marked as distinguished, the amount of needed collisions to sample from the distinguished set should be geometrically distributed with parameter $p = \frac{R}{|D|}$. We find that the amount of collisions needed for success is indeed geometrically distributed. In Fig. 19 we show the obtained distribution and for comparison a geometric distribution with parameter p', where $1/p'$ is the empirical average observed in the experiment.

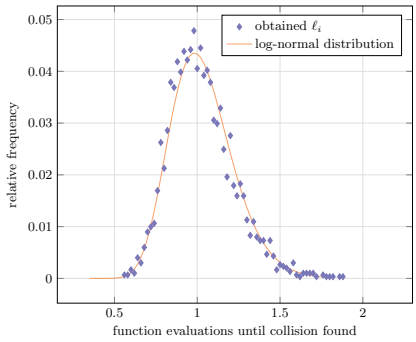
We further repeated this experiment multiple times for different LWE instances, and for each repetition we recorded the experimentally observed parameter p_i of the geometric distribution averaged over 30 successes. Let $\ell_i = \frac{p}{p_i}$ be the quotient of empirical and expected parameter. We would expect ℓ_i to be

close to one. In Fig. 20 we plot the distribution of the ℓ_i observing that they are indeed concentrated around the expected value with low variance. Moreover, the ℓ_i seem to follow a log-normal distribution, which we plot for comparison.



(a) $n = 13, w = 2, w_s = 2$, sample size 3,000, $\mathbb{E}[\ell_i] = 0.96$, $\mathrm{Var}[\ell_i] = 0.048$

(b) $n = 21, w = 3, w_s = 4$, sample size 4,400, $\mathbb{E}[\ell_i] = 1.03$, $\mathrm{Var}[\ell_i] = 0.048$



(c) $n = 32, w = 3, w_s = 4$, sample size 3,000, $\mathbb{E}[\ell_i] = 1.03$, $\mathrm{Var}[\ell_i] = 0.037$

Fig. 20: Distribution of the ℓ_i for different parameter sets.

D Time-Memory Trade-Off for Arbitrary Depth

In this section we give for completeness a description of Algorithm 4 for arbitrary depth. The main algorithm is specified in Algorithm 7 which relies on the recursive algorithms Algorithm 5 and Algorithm 6.

Algorithm 5 randomly chooses the entries for some predetermined coordinates of the free constraints in level-i and below, which is essentially the generalized analog of Lines 7, 11 and 15 in Algorithm 4 for input $i = 3, 2, 1$ respectively. Algorithm 6 is then used to (re-)compute the lists on level-i as many times as

required to find the solution and uses recursive calls to also (re-build) levels j for $j < i$ and finally appends found elements in level 0 to the list L from Algorithm 7. Note that, in each reconstruction Algorithm 5 is called to ensure lists are rebuild for randomized constraints. Finally, Algorithm 7 initiates the procedure by guessing ℓ' coordinates of $\mathbf{e}$ and by enumerating the BaseLevel lists before calling Algorithm 6.

Algorithm 5: SetConstraints(i)

Input: $i \in \mathbb{N}$, specifying level of the tree
Description: Randomly chooses the entries for some predetermined coordinates of the free constraints in level-i and below, that are going to be used in Algorithm 6 and Algorithm 7.

for $k = i, \cdots, 1$ **do**
  for $j = 1, \cdots, 2^{k-1}$ **do**
    Choose randomly the ℓ_i-coordinates of $c_{2j-1}^{(k)}[\ell^{(i+1)} + 1, \ell^{(i)}]$ `// Here` $\ell^{(d)} := 0$ `and` $\ell^{(i)} := \sum_{m=i}^{d-1} \ell_m$.

Algorithm 6: ListCreation(i)

Input: $i \in \mathbb{N}$, specifying level of the tree
Description: Computes the lists on level-i and below and appends the elements in level-0 to the list L which is initiated in Algorithm 7

repeat q^{t_i} **times**
  `// For definition of` t_i `see Eq. (19)`
  SetConstraints(i) `// see Algorithm 5`
  for $j = 0, \cdots, 2^i - 1$ **do**
    From $L_{2j+1}^{(i+1)}$ and $L_{2j+2}^{(i+1)}$ compute and filter on the fly to get $L_{j+1}^{(i)}$, filter such that $L_{j+1}^{(i)} \subseteq S^{(i)}$
  if $i \neq 1$ **then**
    $i \leftarrow i - 1$
    ListCreation(i) `// recursive definition`
  else
    From $L_1^{(1)}$ and $L_2^{(1)}$ compute and filter on the fly to get $L_1^{(0)}$, filter such that $L_1^{(0)} \subseteq S^{(0)}$
    if $|L_1^{(0)}| \geq 1$ **then**
      $L \leftarrow L \cup L_1^{(0)}$

Algorithm 7: Combinatorial Key Search Trade-off for Depth-d

Input: $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n$, positive integer $w \leq n$
Output: $\mathbf{s} \in \tau^n(w/2)$ such that $\mathbf{e} = \mathbf{As} - \mathbf{b} \mod q \in \{-1, 0, 1\}^n$

$\ell' := \log_q |\tau^n(w/2)|$
$L = \emptyset$
Choose optimal $\ell_1, \cdots, \ell_{d-1}$ and the suitable dissection d_k depending on the available memory
for $i = 1, \cdots, d_k \cdot 2^{(d-1)}$ **do**
    Enumerate the list $L_i^{\text{BaseLevel}}$
repeat
    Guess lower ℓ' coordinates of $\mathbf{e}$ randomly
    repeat $q^{t_{d-1}}$ **times**
        Call SetConstraints$(d-1)$ `// see Algorithm 5`
        for $i = 0, \cdots, 2^{(d-1)} - 1$ **do**
            From $L_{d_k \cdot i+1}^{(\text{BaseLevel})}, \cdots, L_{d_k \cdot (i+1)}^{(\text{BaseLevel})}$ compute $L_{i+1}^{(d-1)}$ using d_k-dissection, note that $L_{i+1}^{(d-1)} \subseteq S^{(d-1)}$ holds by construction
        Call ListCreation$(d-2)$ `// see Algorithm 6`
until $|L| \geq 1$