# How to Find Ternary LWE Keys Using Locality Sensitive Hashing

Elena Kirshanova[1,2] ⋆ and Alexander May[1] *

[1] Horst Görtz Institute for IT-Security, Ruhr University Bochum
[2] Immanuel Kant Baltic Federal University, Kaliningrad, Russia
iggy.hoof, elena.kirshanova, alex.may{@rub.de}

**Abstract.** Let $As = b + e \bmod q$ be an LWE-instance with ternary keys $s, e \in \{0, \pm 1\}^n$. Let $s$ be taken from a search space of size $\mathcal{S}$. A standard Meet-in-the-Middle attack recovers $s$ in time $\mathcal{S}^{0.5}$. Using the representation technique, a recent improvement of May shows that this can be lowered to approximately $\mathcal{S}^{0.25}$ by guessing a sub-linear number of $\Theta(\frac{n}{\log n})$ coordinates from $e$. While guessing such an amount of $e$ can asymptotically be neglected, for concrete instantiations of e.g. NTRU, BLISS or GLP the additional cost of guessing leads to complexities around $S^{0.3}$. We introduce a locality sensitive hashing (LSH) technique based on Odlyzko's work that avoids any guessing of $e$'s coordinates. This LSH technique involves a comparably small cost such that we can significantly improve on previous results, pushing complexities towards the asymptotic bound $S^{0.25}$. Concretely, using LSH we lower the MitM complexity estimates for the currently suggested NTRU and NTRU Prime instantiations by a factor in the range $2^{20} - 2^{49}$, and for BLISS and GLP parameters by a factor in the range $2^{18} - 2^{41}$.

**Keywords:** Ternary LWE, Combinatorial attack, Representations, LSH.

## 1 Introduction

The LWE problem is currently without a doubt the richest source for constructing efficient quantum-resistant cryptography. Let $(A, b) \in \mathbb{F}_q^{n \times n} \times \mathbb{F}_q^n$ be an LWE public key with secret key $s \in \mathbb{F}_q^n$ satisfying $As = b + e \bmod q$ for some error $e \in \mathbb{F}_q^n$. The unknown vectors $s, e$ have entries significantly smaller than $q$. For efficiency reasons, many modern LWE variants even use ternary secrets $s, e \in \{0, \pm 1\}^n$. Thus, it is of uttermost interest to understand the complexity of ternary key LWE – also called NTRU-type – schemes.

A standard Meet-in-the-Middle algorithm (MitM) splits $s = s_1 + s_2$ with $s_1 \in \{0, \pm 1\}^{n/2} \times 0^{n/2}$ and $s_2 \in 0^{n/2} \times \{0, \pm 1\}^{n/2}$. Therefore, we obtain the identity

$$As_1 = -As_2 + b + e \bmod q. \tag{1}$$

⋆ Supported by the Ministry of Science and Higher Education of the Russian Federation (agreement no. 075-02-2021-1748) and the "Young Russian Mathematics" grant.

One then computes for all potential $s_1, s_2$ the values $As_1$ and $-As_2 + b$. With high probability only for the correct pair $s_1, s_2$ these values are apart by a ternary error $e \in \{0, \pm 1\}^n$. The correct pair is efficiently identified by a locality sensitive hash function proposed by Odlyzko, mentioned in the original NTRU paper [HPS98].

Recently, the above MitM attack has been improved by May [May21], based on the representation techniques that was developed in [HJ10,BCJ11,BJMM12]. The key idea in [May21] is to search over all $s_1, s_2 \in \{0, \pm 1\}^n$ that satisfy Equation (1) on $k = \Theta(\frac{n}{\log n})$ coordinates exactly, and on the remaining $n - k$ coordinates up to the entries of $e$ (using Odylzko's hashing). This in turn implies that we have to initially guess $k$ coordinates of $e$ to realize the exact matching.

*Our contribution:* We show that a suitable modification of Odylzko's locality sensitive hash function (LSH) allows to avoid any error guessing in [May21]. Since the cost of our LSH function is comparatively small, in turn we significantly improve over the MitM complexities given in [May21], see Table 1.[3]

| | $(n, q, w)$ | $\mathcal{S}$ [bit] | [May21] [bit] | Our [bit] | [DDGR20] Core-SVP |
|---|---|---|---|---|---|
| NTRU IEEE [IEE08] | $(659, 2048, 76)$ | 408 | 146 | **135** | 151 |
| | $(761, 2048, 84)$ | 457 | 166 | **162** | 176 |
| | $(1087, 2048, 126)$ | 680 | 243 | **221** | 260 |
| | $(1499, 2048, 158)$ | 877 | 315 | **283** | 358 |
| NTRU [CDH+20] | $(509, 2048, 254)$ | 754 | 227 | **191** | 124 |
| | $(677, 2048, 254)$ | 891 | 273 | **226** | 167 |
| | $(821, 4096, 510)$ | 1286 | 378 | **358** | 197 |
| | $(701, 8192, 468)$ | 1101 | 327 | **295** | 155 |
| NTRU Prime [BBC+20] | $(653, 4621, 288)$ | 925 | 272 | **228** | 148 |
| | $(761, 4591, 286)$ | 1003 | 301 | **268** | 174 |
| | $(857, 5167, 322)$ | 1131 | 338 | **315** | 196 |
| BLISS I+II [DDLL13] | $(512, 12289, 154)$ | 597 | 187 | **159** | 102 |
| GLP I [GLP12] | $(512, 8383489, 342)$ | 802 | 225 | **184** | 60 |

Table 1: Results of our LSH Meet-in-the-Middle Attack.

In comparison to the results in [May21], for the encryption schemes NTRU and NTRU Prime we gain a run time factor between $2^{20}$ for NTRU-821 and $2^{49}$

[3] The scripts to reproduce the tables are available at https://github.com/ElenaKirshanova/ntru_with_lsh

for NTRU-677. For the signatures schemes we gain a $2^{18}$ factor for BLISS I+II, and a $2^{41}$-factor for GLP I.

In terms of the search space size $\mathcal{S}$ for the secret key, we obtain attacks in the range $\mathcal{S}^{0.23}$ for GLP-I and $S^{0.28}$ for NTRU-821. These exponents in the range $[0.23, 0.28]$ are close to the asymptotic exponents achieved in [May21], and thus indicate the optimality of our LSH approach.

Another direction of improvement is the use of the representation technique not only for the enumeration of $s$ as in [May21], but also for the error vector $e$. This approach yields comparable improvements to our LSH technique: we provide more details in Appendix A. Since LSH and representations of $e$ are somewhat orthogonal techniques to exploit the structure of $e$, we currently do not see a way to combine both approaches.

In comparison to the (highly optimized) lattice attacks in the Core-SVP metric from [DDGR20], our estimates are still a tad bit away. However, we beat current lattice estimates for a selection of the NTRU IEEE 1363-2008 standard [IEE08], see Table 1. For instance, for the ees1499ep1 parameter set we further improve the attack of [May21] by another 32 bits, now beating current lattice estimates by 75 bits.

This demonstrates that our purely combinatorial attack shows its strength in the small weight regime, e.g. for ees1499ep1 with only $w = 158$ non-zero secret key coefficients. We would like to point out that current cold-boot attack scenarios such as [ADP18] live in the (really) small weight regime. We provide cold-boot applications of our attack in Section 6.

On the technical level, we have to construct an LSH approach that realizes an approximate hashing over many levels of a search tree. This is not straightforward, since Odlyzko's original LSH function does not provide linearity. We realize an LSH hashing over search trees via suitable combinations of projections. Given the importance of search tree constructions optimizations with LSH [MO15], we hope that our projection technique will find more applications.

*Notations.* We denote by $\mathbb{Z}_q$ the ring of integers modulo $q \geq 2$. Vectors are denoted by lowercase letters, matrices by uppercase letters. The $n \times n$ identity matrix is denoted by $I_n$. The $\ell_\infty$-norm of vector $x$, denoted by $\|x\|_\infty$, is $\max_i |x_i|$. For a set $S$, we denote by $|S|$ its size.

We shall also make use of multinomial coefficients: for positive integers $n$, $\{n_i\}_{i \leq k}$ such that $n = n_1 + \ldots + n_k$, the multinomial coefficient, denoted by $\binom{n}{n_1, \ldots, n_{k-1}, \cdot}$, is the product $\binom{n}{n_1} \cdot \binom{n-n_1}{n_2} \cdot \ldots \cdot \binom{n - \sum_{i<k} n_i}{n_k}$.

## 2 Generalizing Odlyzko's LSH

In order to generalize Odlyzko's LSH to search trees, we consider the following problem abstraction that we face for every node of our search tree constructions.

**Definition 1 (Close pairs problem in $\ell_\infty$-norm).** *Given two equal-sized lists $L_1, L_2$ of iid. uniform random vectors from $\mathbb{Z}_q^n$, find an $(1 - o(1))$-fraction* *of all pairs* $(x_1, x_2) \in L_1 \times L_2$ *that satisfy* $\|(x_1 - x_2) \bmod q\|_\infty = 1$*. Any such pair is called* 1-close*.*

This is an average-case version of the close pairs problem and we shall make use of the distribution in our analysis. In particular, we assume that elements from the lists $L_1, L_2$ do not cluster, i.e., there is no subset of vectors with small diameter. For the worst-case version of the problem, an algorithm is given by Indyk in [Ind01]. Note also that we are in the special case of the $\ell_\infty$ norm on the torus $\mathbb{Z}_q = \{0, \ldots, q-1\}$, i.e., it holds that $\|0 - (q-1)\|_\infty = 1$. Furthermore, the lists $L_1, L_2$ are assumed to be of $\exp(n)$-size.

The close pairs problem is solved using the so-called *locality-sensitive hash functions* (LSH) [IM98,AI06]. Informally, such a hash function has higher collision probability for elements that are close than for those that are far apart.

For the $\ell_\infty$-norm over $\mathbb{Z}_q$, Odlyzko proposed a construction of a locality-sensitive hash (LSH) function [HPS98]. Odlyzko's LSH splits $\mathbb{Z}_q$ into two halves: $[0, \lfloor q/2 \rfloor - 1]$ and $[\lfloor q/2 \rfloor, q-1]$, and assigns the 0-label to the first half and the 1-label to the second half. It is extended to vectors coordinate-wise thus mapping $\mathbb{Z}_q^n$ to $\{0,1\}^n$. It is likely that close vectors have the same label under this mapping. In order to avoid losing close pairs, Odlyzko suggests to assign both 0- and 1-labels to the "border" values $\lfloor q/2 \rfloor - 1$ and $\lfloor q/2 \rfloor$. We do not perform such a double assignment, but instead we re-randomize the function as we explain below.

The choice to split $\mathbb{Z}_q$ into two halves works particularly well when there is a unique close pair in the sense that the other pairs have a different label under Odlyzko's mapping. In our average case setting non-close pairs differ by label with probability $1 - 2^{-n}$, since the probability that two uniform random elements from $\mathbb{Z}_q$ are in the same half wrt. to $\lfloor q/2 \rfloor$ is $1/2$.

In our applications we will be in the setting where a solution may not be unique and thus we require in Definition 1 to output (almost) all close pairs. Odlyzko's LSH generalises to this setting by

1. dividing the $\mathbb{Z}_q$ torus into more than 2 parts, and
2. re-randomizing the hash function (see also [Ngu21]) so that we can handle border values in a more elegant way than assigning multiple labels[4].

More precisely, consider the following straightforward generalisation of Odlyzko's LSH. For a fixed bound $B \in \{1, \ldots, q\}$ and a uniformly chosen shift-vector $u \in \mathbb{Z}_q^n$ define

$$
\begin{aligned}
h_{u,B} : \mathbb{Z}_q^n &\to \left[0, \ldots, \left\lceil \frac{q}{B} \right\rceil - 1\right]^n \\
(x_1, \ldots, x_n) &\mapsto \left( \left\lfloor \frac{x_1 + u_1}{B} \right\rfloor, \ldots, \left\lfloor \frac{x_n + u_n}{B} \right\rfloor \right).
\end{aligned}
$$

[4] In fact, the 'multiple' labels assignment is what is done in [Ind01] to handle worst-case inputs. We could also use this algorithm but it turns out to be less memory-efficient than what we propose for the average-case setting.

In the original Odlyzko's LSH, $B$ is set to $q/2$. We choose a uniform random function from the family $H_B = \{h_{u,B} \mid u \in \mathbb{Z}_q^n\}$. For a list $L_1 \subset \mathbb{Z}_q^n$, the shift $L_1 + u$ is just a rotation of all the elements on the $\mathbb{Z}_q$ torus. Any function $h_{u,B}$ can be evaluated in $\mathcal{O}(n)$ operations over $\mathbb{Z}_q$.

**Algorithm 1** Our LSH-Odlyzko algorithm for finding 1-close pairs

**Input:** $L_1, L_2$ – list of iid. uniform vectors from $\mathbb{Z}_q^n$, each of size $|L|$.
**Output:** $(1 - o(1))$-fraction of all pairs $(x_1, x_2) \in L_1 \times L_2$ such that $\|(x_1 - x_2) \bmod q\|_\infty = 1$

1: Choose $B \geq \frac{q}{|L|^{1/n}} \in \{1, \ldots, q\}$ suitably. Choose $u \xleftarrow{\$} \mathbb{Z}_q^n$.
2: Apply $h_{u,B}$ to $L_1, L_2$. Sort $L_1, L_2$ according to the hash values.
3: Merge the sorted lists according to their hash labels. Output only those pairs $(x_1, x_2) \in L_1 \times L_2$ that satisfy $\|(x_1 - x_2) \bmod q\|_\infty = 1$
4: Repeat Steps 1–3 $N$ times, where

$$N = \left(\frac{B}{B-1}\right)^n \cdot n \log n \tag{2}$$

Let us now provide our algorithm LSH-Odylzko (Algorithm 1) that solves the close pairs problem from Definition 1. For our NTRU-type applications, we later solve close pairs problems on suitably chosen projections of all $n$ coordinates. Notice that $h_{u,B}$ can easily applied on projections, since it works coordinate-wise.

**Theorem 1 (Adapted from [IM98]).** *Given two lists $L_1, L_2$ of equal size $|L|$ with iid. elements taken from the uniform distribution on $\mathbb{Z}_q^n$,* LSH-Odlyzko *(Algorithm 1) solves the close pairs problem from Definition 1 in space and time complexities*

$$S = \max\left\{|L|, |L|^2 \cdot \left(\frac{3}{q}\right)^n\right\} \cdot \mathrm{poly}(n),$$
$$T_{\mathsf{LSH}}(|L|, n, B) = \max\left\{S, |L|^2 \left(\frac{B^2}{(B-1)q}\right)^n \cdot \mathrm{poly}(n)\right\}.$$

*Proof.* The proof is an adaptation of [IM98, Theorem 5] to the average-case $\ell_\infty$-norm setting.

We start with the analysis of Steps 1–3 of Algorithm 1.

In Step 2, hashing and sorting can be performed within time and memory complexity $\widetilde{\mathcal{O}}(|L|) = |L| \cdot \mathrm{poly}(n)$.

Notice that our choice of $B$ in Step 1 implies $|L| \left(\frac{B}{q}\right)^n \geq 1$, which is the expected number of elements from $L_1$ (or $L_2$) that receive the same hash label. Thus the number of elements in $L_1 \times L_2$ that match by hash label is $|L|^2 \left(\frac{B}{q}\right)^n$, and these pairs can be found in Step 3 in time $|L|^2 \left(\frac{B}{q}\right)^n \cdot \mathrm{poly}(n)$ time. Among the pairs $(x_1, x_2) \in L_1 \times L_2$ we filter out all those that are not 1-close in $\ell_\infty$ norm.

Notice that in total we expect $|L|^2 \cdot \left(\frac{3}{q}\right)^n$ 1-close pairs. However, since we consider only those pairs with matching LSH-label, in each iteration we only obtain a certain fraction of all 1-close pairs. It remains to show that by our choice of $N$ repetitions in Step 4 we eventually find almost all 1-close pairs.

Let $(x_1, x_2) \in L_1 \times L_2$ be a solution to the close pairs problem, and consider the event $E$ that $h_{u,B}(x_1) = h_{u,B}(x_2)$, i.e., $x_1, x_2$ receive the same hash label for a random hash function. Then

$$\Pr[E] = \prod_{i=1}^{n} \left(1 - \Pr_{h_{u,B}}\left[\left\lfloor \frac{x_i + u_i}{B} \right\rfloor \neq \left\lfloor \frac{x'_i + u_i}{B} \right\rfloor\right]\right) = \left(1 - \frac{q/B}{q}\right)^n = (1 - 1/B)^n .$$

Thus, $E$ happens after $N = (\Pr[E])^{-1} n \log n$ repetitions with probability

$$1 - (1 - \Pr[E])^N \leq 1 - e^{-n \log n}.$$

Taking the union bound over all $\exp(n)$-many potentially 1-close pairs $(x_1, x_2) \in L_1 \times L_2$ ensures that we find with high probability an $(1 - o(1))$-fraction of all 1-close pairs. □

Notice that Algorithm 1 requires some optimization of $B$. The larger $B$, the larger is the number of 1-close pairs that we find per iteration, and the smaller the required number $N$ of iterations. In our applications, we found the optimal value $B$ that minimizes $T_{\mathsf{LSH}}(|L|, n)$ in Theorem 1 by an exhaustive search.

*Combining approximate with exact matching.* Algorithm 1 can be easily adapted to exact matching by setting $B = q$, $N = 0$, and the whole process will correspond to simple merge sort. Now, assume we need to combine approximate matching on some $k_1$ coordinates and exact matching on some other $k_2$ coordinates. A hash label is then a concatenation of an approximate label of dimension $k_1$ and an exact label of dimension $k_2$. Then the number of elements in $L_1 \times L_2$ that have the same label is $|L|^2 \left(\frac{B}{q}\right)^{k_1} \left(\frac{1}{q}\right)^{k_2}$. The space and time complexity of this combined LSH+Exact algorithm are up to $\mathrm{poly}(n)$ terms

$$\begin{aligned} S &= \max\left\{ |L|, |L|^2 \cdot \left(\frac{3}{q}\right)^{k_1} \left(\frac{1}{q}\right)^{k_2} \right\}, \\ T_{\mathsf{LSH+Exact}}(|L|, k_1, k_2, B) &= \max\left\{ S, |L|^2 \left(\frac{B}{q}\right)^{k_1} \left(\frac{1}{q}\right)^{k_2} \cdot N \right\}. \end{aligned} \tag{3}$$

## 3 Our LSH-based MitM with Rep-0 Representations

Since our algorithm builds on top of the representation technique-based Meet-LWE algorithm of [May21], let us briefly sketch the idea of representations, how they are used inside Meet-LWE, and how our LSH-technique for 1-close pairs from Section 2 leads to an improved LSH-Meet-LWE algorithm. As a warm-up, for didactical reasons we describe in this section the idea how to use our LSH technique with depth-2 search trees, where our technique is only used once to construct the level-1 lists $L_1^{(1)}$ and $L_2^{(1)}$ (the upper index denotes the level of the lists in Figure 1). In the subsequent sections, we show how to generalize the technique to larger depth.

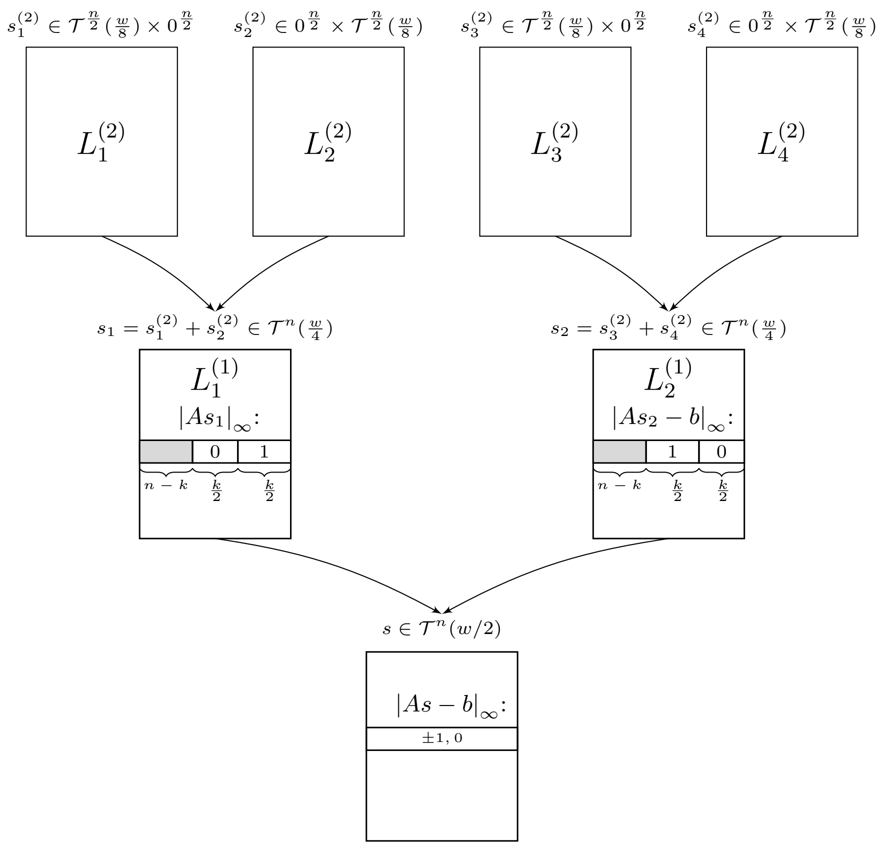


Fig. 1: LSH-Meet-LWE algorithm with Rep-0 representations

*Reprentations and* Meet-LWE. Let $\mathcal{T}^n = \{0, \pm 1\}^n \cap \mathbb{F}_q^n$ denote the set of ternary vectors. Moreover we denote by $\mathcal{T}^n(w/2)$ the set of ternary vectors having weight $w$ with exactly $w/2$ 1-entries and $w/2$ $(-1)$-entries.

Let $As = b + e \bmod q$ be the LWE key equation with $e \in \mathcal{T}^n$ and $s \in T^n(w/2)$. We represent $s = s_1 + s_2$ where $s_1, s_2 \in T^n(w/4)$, i.e. $s_1, s_2$ have exactly $w/4$ 1- and (-1)-entries each. Notice that there are $\mathcal{R} = \binom{w/2}{w/4}^2$ ways to represent $s$

as a sum of two weight $w/2$-vectors $s_1, s_2$. We call each such a tuple $(s_1, s_2)$ a Rep-0-*representation* of $s$.

Choose $k$ maximal such that $q^k < \mathcal{R}$. Assume that on level 1 of the search tree, we first match on $k$ coordinates, and on level 0 we match on the remaining $n-k$ coordinates. Further let $\pi_k : \mathbb{F}_2^n \to \mathbb{F}_2^k$ denote the projection on the first $k$ coordinates.

We rewrite the LWE MitM identity from Equation (1) as

$$\pi_k(As_1+e_1) = \pi_k(b-As_2+e_2) \text{ for some } e_1 \in 0^{k/2}\times\mathcal{T}^{k/2}, e_2 \in \mathcal{T}^{k/2}\times 0^{k/2}. \quad (4)$$

Since $q^k < R$, we expect that for each target value $t \in \mathbb{F}_q^k$ there exists a representation $(s_1, s_2)$ such that $\pi_k(As_1 + e_1) = t = \pi_k(b - As_2 + e_2)$. Meet-LWE guesses $e_1, e_2$ to realize the exact matching to target $t$ on these $k$ coordinates.

*High-Level Idea of* LSH-Meet-LWE. Using our LSH approach, one finds all $s_1$ such that $\pi_k(As_1)$ in Equation (4) matches $t$ on the first $k/2$ coordinates exactly, and on the remaining coordinates up to some ternary vector. By contrast, we construct all $s_2$ such that $\pi_k(b - As_2)$ matches $t$ on the last $k/2$ coordinates exactly, and on the first $k/2$ coordinates up to some ternary vector.

The approximate matching on the remaining $n-k$ coordinates is again done via LSH-Odlyzko. Notice that by construction we eventually construct $s = s_1 + s_2$ such that $As = b$ up to some ternary error vector $e \in \mathcal{T}^n$, as desired.

Let us state our LSH-based algorithm more precisely.

*Description of our* LSH-Meet-LWE *algorithm.*

1. Enumerate the following 4 level-2 lists:

$$\begin{aligned}
L_1^{(2)} &= \{(s_1^{(2)} \in \mathcal{T}^{\frac{n}{2}}\left(\frac{w}{8}\right) \times 0^{\frac{n}{2}})\},\\
L_2^{(2)} &= \{(s_2^{(2)} \in 0^{\frac{n}{2}} \times \mathcal{T}^{\frac{n}{2}}\left(\frac{w}{8}\right))\},\\
L_3^{(2)} &= \{(s_3^{(2)} \in \mathcal{T}^{\frac{n}{2}}\left(\frac{w}{8}\right) \times 0^{\frac{n}{2}})\},\\
L_4^{(2)} &= \{(s_4^{(2)} \in 0^{\frac{n}{2}} \times \mathcal{T}^{\frac{n}{2}}\left(\frac{w}{8}\right))\}.
\end{aligned} \quad (5)$$

2. Let $\mathcal{R} = \binom{w/2}{w/4}^2$. Choose a positive even integer $k < n$ that satisfies

$$k = \left\lfloor \frac{\log_2(\mathcal{R})}{\log_2 q - 0.5\log_2 3} \right\rfloor.$$

   This choice of $k$ allows to expect one solution to survive during the merge of $L_1^{(2)}$ with $L_2^{(2)}$ and $L_3^{(2)}$ with $L_4^{(2)}$ as we find exact matches on $k/2$ coordinates and all 1-close pairs on another $k/2$ coordinates, hence we expect $\mathcal{R} \approx q^{\frac{k}{2}}\left(\frac{q}{3}\right)^{\frac{k}{2}}$.
3. Find all $(As_1^{(2)}, As_2^{(2)})$ that

(a) match (sum to 0) on the coordinates $[k/2+1, k]$, and are
(b) 1-close on the coordinates $[1, k/2]$.
Analogously, find all $(As_3^{(2)}, As_4^{(2)})$ that
(a) match (sum to 0) on the coordinates $[1, k/2]$, and are
(b) 1-close on the coordinates $[k/2+1, k]$.
Use our LSH-Odlyzko (Algorithm 1) with optimal $B$ to find 1-close pairs. This gives us two lists

$$L_1^{(1)} = \{(s_1 \in \mathcal{T}^n\left(\frac{w}{4}\right) \,:\, As_1 \in \mathbb{Z}_q^{n-k} \times 0^{k/2} \times \{\pm 1, 0\}^{k/2})\}$$
$$L_2^{(1)} = \{(s_2 \in \mathcal{T}^n\left(\frac{w}{4}\right) \,:\, As_2 \in \mathbb{Z}_q^{n-k} \times \{\pm 1, 0\}^{k/2} \times 0^{k/2})\}.$$

4. Use LSH-Odlyzko again to find pairs from $L_1^{(1)}, L_2^{(1)}$ that are 1-close on the remaining $n-k$ coordinates.

Let $|L^{(j)}|$ denote the common length of all level-$j$ lists. Notice that on level 1 we obtain expected list length

$$\left|L_1^{(1)}\right| = \left|L_1^{(2)}\right|^2 \cdot \left(\frac{3}{q}\right)^{k/2} \cdot \left(\frac{1}{q}\right)^{k/2}.$$

Using Theorem 1 and ignoring polynomial factors, the running time of LSH-Meet-LWE with Rep-0 representations is (here $N$ is given in Eq (2))

$$\begin{aligned} T_{\text{Rep-0}} &= \max\left\{|L^{(2)}|,\ T_{\mathsf{LSH+Exact}}\left(|L^{(2)}|, \frac{k}{2}, \frac{k}{2}, B\right),\ T_{\mathsf{LSH}}(|L^{(1)}|, n-k, q/2)\right\} \\ &= \max\left\{|L^{(2)}|, \left|L^{(2)}\right|^2 \cdot \left(\frac{B}{q}\cdot\frac{1}{q}\right)^{k/2} \cdot N, \left|L^{(2)}\right|^4 \cdot \left(\frac{3}{q^2}\right)^k \cdot N \cdot 2^{-(n-k)}\right\}. \end{aligned}$$

Table 1 gives concrete values of $T_{\text{Rep-0}}$. For all of them the optimal value of the LSH-Odlyzko parameter is $B = 3$. For concrete parameters, $B$ can be found using a brute-force search.

## 4 Generalizing our LSH-based MitM to Rep-1 Representations

The algorithm from the previous section can be generalised and improved by

1. representing weight-$w$ secrets $s = s_1 + s_2$ with $s_1, s_2$ having weight larger than $w/2$. As opposed to Section 3 this allows to represent 0-coordinates of $s$ not only by $0+0$, but also as $-1+1$ or as $1+(-1)$. These are called Rep-1 representations in [May21]. Notice that Rep-1 in comparison to Rep-0 increases the search space.
2. by constructing a deeper search tree to amortize the increased search space over many levels.

| | $(n,q,w)$ | LSH-Meet-LWE Rep-0 | $\log_2(N), k$ | Meet-LWE [May21] |
|---|---|---|---|---|
| NTRU-Enc | $(509, 2048, 254)$ | 299 | $16, 24$ | 305 |
| | $(677, 2048, 254)$ | 360 | $18, 24$ | 364 |
| | $(821, 4096, 510)$ | 509 | $27, 44$ | 520 |
| | $(701, 8192, 468)$ | 449 | $22, 36$ | 461 |
| NTRU-Prime | $(653, 4621, 288)$ | 370 | $17, 24$ | 370 |
| | $(761, 4591, 286)$ | 407 | $18, 24$ | 408 |
| | $(857, 5167, 322)$ | 473 | $20, 26$ | 459 |
| BLISS I+II | $(512, 12289, 154)$ | 267 | $7, 10$ | 247 |
| GLP I | $(512, 8383489, 342)$ | 326 | $9, 14$ | 325 |

Table 2: Comparison bit complexities for Rep-0 using our LSH-Meet-LWE and Meet-LWE.

Let us describe the depth-3 version of our LSH-Meet-LWE with Rep-1. The reader is advised to follow Figure 2. We implicitly assume that all fractions that appear are integers by appropriate rounding. We count the levels from bottom to top starting with 0, e.g., on level 3 we have 8 lists. The upper index of the elements refers to the level. In Figure 2, we also visualize how we define suitable projections such that our LSH-Odlyzko eventually finds 1-close pairs.

LSH-Meet-LWE *for* Rep-1 *with depth* 3. The eight top-most lists are of the form

$$L_i^{(3)} = \left\{ s_i^{(3)} \in \mathcal{T}^{\frac{n}{2}} \left( \frac{w}{16} + \frac{\varepsilon[1]}{4} + \frac{\varepsilon[2]}{2} \right) \times 0^{\frac{n}{2}} \right\} \quad \text{for odd } i,$$
$$L_i^{(3)} = \left\{ s_i^{(3)} \in 0^{\frac{n}{2}} \times \mathcal{T}^{\frac{n}{2}} \left( \frac{w}{16} + \frac{\varepsilon[1]}{4} + \frac{\varepsilon[2]}{2} \right) \right\} \quad \text{for even } i,$$

where $\varepsilon[i]$ describes the number of additional 1's we add in the representation of the secret $s$ on level $i$. More precisely, on the bottom level, we target the solution $s$ of weight $w$, i.e., $s \in \mathcal{T}^n(w/2)$. We split $s$ into $s = s_1^{(1)} + s_2^{(1)}$, where each $s_1^{(1)}, s_2^{(1)} \in \mathcal{T}^n(w/4 + \varepsilon[1])$ for some $\varepsilon[1] \geq 0$. This gives us, as in the previous section, $\binom{w/2}{w/4}^2$ ways to represent 1's and -1's in $s$, and in addition $\binom{n-w}{\varepsilon[1], \varepsilon[1], \cdot}$ ways to represent 0's in $s$. The total number of representations for $s$ on level 1 is therefore

$$\mathcal{R}^{(1)} = \binom{w/2}{w/4}^2 \cdot \binom{n-w}{\varepsilon[1], \varepsilon[1], \cdot}.$$

Next, we go one level up by splitting $s_1^{(1)}$ (analogously for $s_2^{(1)}$) into two vectors $s_1^{(2)}, s_2^{(2)}$, each from $\mathcal{T}^n(\frac{w}{8} + \frac{\varepsilon[1]}{2} + \varepsilon[2])$. Therefore, the 1's and -1's in $s_1^{(1)}$ can

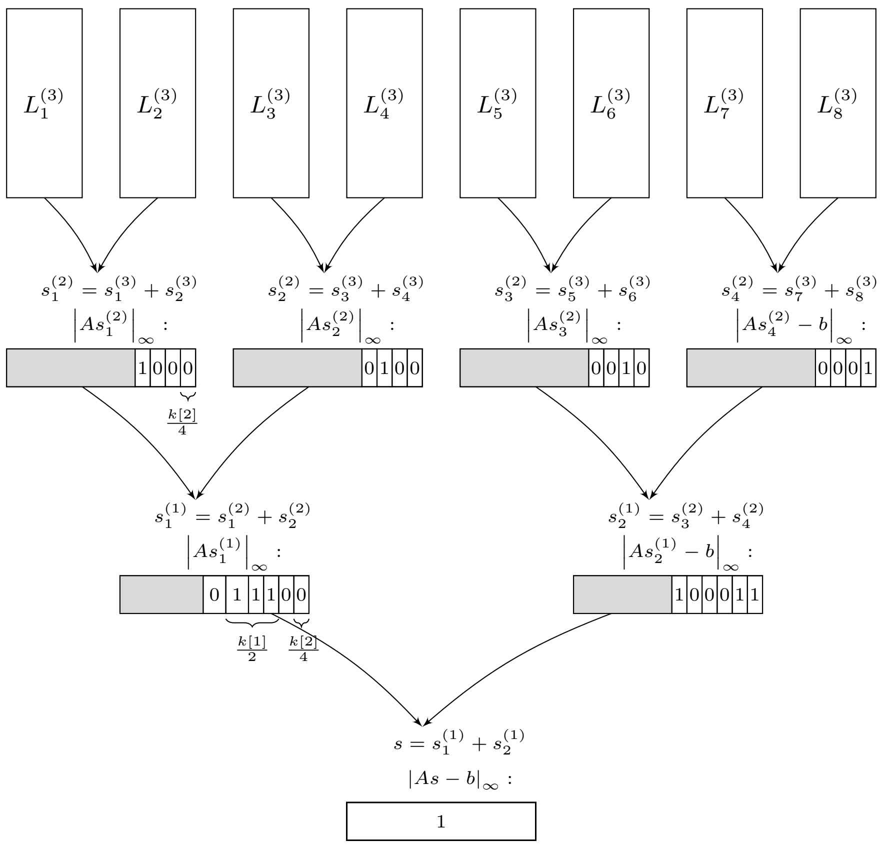


Fig. 2: LSH-Meet-LWE algorithm using Rep-1 with depth 3

be represented in $\binom{w/4+\varepsilon[1]}{w/8+\varepsilon[1]/2}^2$ ways, while for 0's of $s_1^{(1)}$ we have $\binom{n-w/2-2\varepsilon[1]}{\varepsilon[2],\varepsilon[2],\cdot}$ representations. In total, the number of level-2 representations is

$$\mathcal{R}^{(2)} = \binom{w/4+\varepsilon[1]}{w/8+\varepsilon[1]/2}^2 \cdot \binom{n-w/2-2\varepsilon[1]}{\varepsilon[2],\varepsilon[2],\cdot}.$$

If we wanted to construct a tree of depth larger than 3, we would continue with representations for $s_1^{(2)}, s_2^{(2)}$. Instead, our depth-3 algorithm enumerates $s_1^{(2)}, s_2^{(2)}$ a standard Meet-in-Middle way by considering $s_1^{(2)} = s_1^{(3)} + s_2^{(3)}$, where $s_i^{(3)} \in \mathcal{T}^{n/2}(\frac{w}{16} + \frac{\varepsilon[1]}{4} + \frac{\varepsilon[2]}{2})$.

The cost of building the top-level lists is determined by their sizes, i.e.,

$$T[3] = \left|L_i^{(3)}\right|.$$

Having constructed the top-most lists $L_i^{(3)}$, we merge them into the lists $L_i^{(2)}$ leaving only a $1/\mathcal{R}^{(2)}$-fraction of pairs $L_i^{(3)} \times L_{i+1}^{(3)}$. To this end, we consider only those pairs $(s_i^{(3)}, s_{i+1}^{(3)}) \in L_i^{(3)} \times L_{i+1}^{(3)}$ for which

1. $As_i^{(3)} = As_{i+1}^{(3)}$ on certain $\frac{3}{4}k[2]$-coordinates, and
2. $\left|As_i^{(3)} - As_{i+1}^{(3)}\right|_\infty \leq 1$ on certain $\frac{1}{4}k[2]$-coordinates (see Figure 2 for our projections).

Here, $k[2]$ satisfies

$$k[2] = \left\lfloor \frac{\log_2(\mathcal{R}^{(2)})}{\log_2 q - 0.5^2 \log_2 3} \right\rfloor.$$

More generally, we have

$$k[i] = \left\lfloor \frac{\log_2(\mathcal{R}^{(i)})}{\log_2 q - 0.5^i \log_2 3} \right\rfloor.$$

For concrete parameters we must further assure that $k[i]$ is divisible by $2^i$ for realizing our projections.

As before, let $|L^{(j)}|$ denote the common length of all level-$j$ lists. The approximate merging on $\frac{1}{4}k[2]$ coordinates is performed using LSH-Odlyzko with LSH parameter $B[2]$. This is combined with exact merging on $\frac{3}{4}k[2]$ coordinates. This implies that we expect on level 2 list size

$$\left|L_i^{(2)}\right| = \left|L_i^{(3)}\right|^2 \cdot \left(\frac{1}{q}\right)^{\frac{3}{4}k[2]} \cdot \left(\frac{3}{q}\right)^{\frac{1}{4}k[2]}.$$

The complexity of constructing level-2 lists is

$$\begin{aligned} T[2] &= \max\left\{T_{\mathsf{LSH+Exact}}\left(|L^{(3)}|, \frac{1}{4}k[2], \frac{3}{4}k[2], B[2]\right), |L_i^{(2)}|\right\} \\ &= N[2] \cdot (q^{\frac{3}{4}k[2]} \cdot \lceil (q/B[2]) \rceil^{\frac{1}{4}k[2]}) \cdot \left(\left|L_i^{(3)}\right| \cdot \left(\frac{1}{q}\right)^{\frac{3}{4}k[2]} \left(\frac{B[2]}{q}\right)^{\frac{1}{4}k[2]}\right)^2. \end{aligned}$$

Level-1 lists are constructed in a similar way to level-2 lists. Concretely, $L_1^{(1)}, L_2^{(1)}$ are constructed via approximate matching on $\frac{1}{2}k[1]$ coordinates and exact matching on $\frac{1}{2}k[1]$ coordinates. Note that by our construction the elements from $L_1^{(1)}, L_2^{(1)}$ are already 1-close on $k[2]/2$ coordinates. The expected level-1 list size is therefore

$$\left|L_i^{(1)}\right| = \left|L_i^{(2)}\right|^2 \cdot \left(\frac{1}{q}\right)^{\frac{1}{2}k[1]-\frac{1}{2}k[2]} \left(\frac{3}{q}\right)^{\frac{1}{2}k[1]-\frac{1}{2}k[2]}.$$

The complexity of constructing level-1 lists is

$$\begin{aligned} T[1] &= \max\left\{T_{\mathsf{LSH+Exact}}\left(L^{(2)}, \frac{1}{2}\left(k[1]-k[2]\right), B[1]\right), |L^{(1)}|\right\} \\ &= N[1]\left(q^{\frac{k[1]}{2}-\frac{k[2]}{2}} \cdot \left\lceil\frac{q}{B[1]}\right\rceil^{\frac{k[1]}{2}-\frac{k[2]}{2}}\right)\left(\left|L_i^{(2)}\right| \cdot \left(\frac{1}{q}\right)^{\frac{k[1]}{2}-\frac{k[2]}{2}} \left(\frac{B[1]}{q}\right)^{\frac{k[1]}{2}-\frac{k[2]}{2}}\right)^2. \end{aligned}$$

In order to construct the final list and determine the solution $s$, we use LSH-Odylzko once again on the remaining $n - k[1]$ coordinates with parameter $B[0] = q/2$ in time

$$T[0] = |L_i^{(1)}| \cdot 2^{n-k[1]}.$$

Overall, the asymptotic time and memory complexities of LSH-Meet-LWE on Rep-1 with depth 3 are respectively

$$T = \max_{0\leq i\leq 3}\{T[i]\} \text{ and } S = \max_{0\leq i\leq 3}\{L[i]\}.$$

## 5 Results: LSH-Meet-LWE (Rep-1) Compared to Lattices

Let us compare the performance of LSH-Meet-LWE to lattice attacks on NTRU-type cryptosystems. Concrete bit securities of proposed NTRU parameter sets are shown in Table 3.

The estimates for lattice attacks are computed with the help of the "leaky-LWE-Estimator" available at https://github.com/lducas/leaky-LWE-Estimator[5]. We used this estimator in the so-called Probabilistic-simulation regime, which gives slightly more accurate figures than, e.g., predictions from [ACD+18].

The estimator, based on the results from [DDGR20], produces bit securities for the so-called primal lattice attack. This attack runs a BKZ-reduction algorithm on the $2n$-dimensional lattice $\Lambda = \{(x, y) \in \mathbb{Z}^{2n} \ : \ [A|I_n]\begin{bmatrix} x \\ y \end{bmatrix} = 0 \bmod q\}$, where $[A|I_n]$ is the column-wise concatenation of matrices $A$ and $I_n$.

The estimator, given the NTRU parameters, produces a block-size parameter $\beta$, which determines the hardness of the BKZ reduction. In particular, we conservatively assume that the lattice attack will run in time $2^{0.292\beta+16.4}$ [BDGL16]

[5] We used commit 4027151 of the branch NTRU_keygen, https://github.com/lducas/leaky-LWE-Estimator/tree/NTRU_keygen.

| $(n, q, w)$ | Rep-0 [bit] | Rep-1 depth 2 [bit], $\varepsilon$ | Rep-1 depth 3 [bit], $\varepsilon$ | Rep-1 depth 4 [bit], $\varepsilon$ | Lattices [DDGR20] $\beta$, $0.292\beta + 16.4$ |
|---|---|---|---|---|---|
| | | | NTRU IEEE-2008 [IEE08] | | |
| $(401, 2048, 226)$ | 260 | 237, [2] | **179**, [11,2] | 180, [39,18,3] | 273, **96** |
| $(449, 2048, 268)$ | 290 | 271, [2] | 208, [13,4] | **180**, [31,13,3] | 318, **109** |
| $(677, 2048, 314)$ | 414 | 362, [4] | 287, [25,6] | **242**, [31,13,2] | 522, **169** |
| $(1087, 2048, 240)$ | 445 | 375, [10] | **289**, [28,8] | 306, [38,13,3] | 835, **260** |
| $(541, 2048, 98)$ | 213 | 177, [8] | **144**, [13, 3] | 160, [27,8,1] | 372, **126** |
| $(613, 2048, 110)$ | 221 | 192, [6] | **160**, [10,3] | 174, [26,11,1] | 435, **143** |
| $(887, 2048, 162)$ | 342 | 287, [12] | 231, [19,6] | **230**, [26,13,4] | 677, **214** |
| $(1171, 2048, 212)$ | 427 | 365, [12] | 300, [23,6] | **283**, [35,14,3] | 945, **292** |
| $(659, 2048, 76)$ | 191 | 156, [6] | **135**, [13,4] | 167, [27,11,1] | 460, **151** |
| $(761, 2048, 84)$ | 221 | 179, [6] | **162**, [12,1] | 181, [37,17,5] | 545, **176** |
| $(1087, 2048, 126)$ | 311 | 251, [8] | **221**, [16,4] | 230, [38,17,4] | 835, **260** |
| $(1499, 2048, 158)$ | 389 | 324, [12] | 286, [26,7] | **283**, [28,10,0] | 1170, **358** |
| | | | NTRU [CDH+20] | | |
| $(509, 2048, 254)$ | 299 | 282, [6] | 203, [12, 4] | **191**, [27, 16, 3] | 369, **124** |
| $(677, 2048, 254)$ | 360 | 322, [6] | 244, [20, 6] | **226**, [27, 16, 3] | 517, **167** |
| $(821, 4096, 510)$ | 509 | 501, [2] | 374, [18, 5] | **358**, [27, 8, 1] | 619, **197** |
| $(701, 8192, 468)$ | 449 | 441, [0] | 336, [27, 4] | **295**, [23, 14, 2] | 474, **155** |
| | | | NTRU Prime [BBC+20] | | |
| $(653, 4621, 288)$ | 370 | 333, [4] | 265, [22, 3] | **228**, [26, 15, 4] | 449, **148** |
| $(761, 4591, 286)$ | 407 | 359, [6] | 276, [24, 6] | **268**, [24, 6, 5] | 539, **174** |
| $(857, 5167, 322)$ | 473 | 413, [8] | 317, [27, 8] | **315**, [27, 10, 4] | 615, **196** |
| | | | BLISS I+II [DDLL13] | | |
| $(512, 12289, 154)$ | 267 | 216, [6] | 166, [15, 3] | **159**, [23, 11, 1] | 292, **102** |
| | | | GLP I [GLP12] | | |
| $(512, 8383489, 342)$ | 326 | 326, [0] | 262, [10, 0] | **184**, [27, 11, 2] | 148, **60** |

Table 3: Bit complexities for our LSH-Meet-LWE using Rep-0, Rep-1 from Sections 3 and 4 with depths-$\{2-4\}$ search trees. We give the optimized values of $\varepsilon$ in square brackets. The last column provides the complexity of lattice-based attacks relying on the results of [DDGR20].

(the constant 16.4 replaces $o(\beta)$ in the asymptotic SVP complexity $2^{0.292\beta+o(\beta)}$, see [APS15]). The values for $\beta$ as well as the bit complexities of the primal attack are given in the last column of Table 3.

*Parameter Sets.* In Table 3 we consider three different NTRU encryption schemes: the IEEE-2008 NTRU standard from [IEE08] with 12 different parameter sets, 4 parameter sets from the NIST standardisation candidate NTRU [CDH⁺20], and 3 parameter sets from the alternative NIST standardisation candidate NTRU Prime [BBC⁺20]. We also consider two signature schemes based on ternary LWE: BLISS with parameter sets I and II from [DDLL13] and GLP [GLP12]. Except BLISS, all these schemes the weight of $e$ is chosen to be $2 \cdot \lfloor n/3 \rfloor$. Note that the exact value of the error weight is relevant only for the lattice attack, while our LSH-Meet-LWE's complexity algorithm is independent of $e$'s weight, but highly sensitive to the weight of the secret $s$. Both LSH-Meet-LWE and lattice reduction require memory exponential in $n$.

*Conclusions.* From Table 3 we observe that our combinatorial LSH-Meet-LWE attack highly profits from small weight. For example, the third package of NTRU IEEE-2008 parameters (speed optimized according to the specification [IEE08]) has smallest weight relative to $n$. For all four instances of this package, our estimates outperform the lattice estimates.

The decision to choose larger weights in recent standardization proposals such as NTRU [CDH⁺20] and NTRU Prime [BBC⁺20] appears to be a wise decision in light of our new combinatorial attack results. For these instances, we cannot compete with current lattice estimates.

We note that the figures in Table 3 both for lattices and LSH-Meet-LWE are likely to underestimate actual costs. For lattices, the $2^{0.292\beta+16.4}$ Core-SVP model does not include several SVP calls within the BKZ reduction, and also hides the complexity of decoding random spherical codes of length $\mathcal{O}(\sqrt{\beta})$. For LSH-Meet-LWE, we omit polynomial factors for LSH-Odlyzko and sorting.

## 6 Cold boot attack

Our combinatorial Rep-1 attack performs best when the secret is sparse. In some cases, see Table 3, it even outperforms lattice-based attacks. Sparse secrets also naturally appear in the so-called cold boot attack scenario [HSH⁺09]. Belonging to the class of side-channel attacks, in an cold-boot attack one has read-access to RAM where the secret key is stored, but some small fraction of bits in this RAM is flipped (after power shut-down).

Thus an attacker obtains a noisy version $s'$ of the secret key $s$. Concretely, let $s' = s + \Delta$, where $\Delta$ is of small Hamming weight $w_\Delta$. With this noisy secret $s'$, the attacker produces from the original ternary LWE instance $(A, b)$ a new instance $(A, b')$, where

$$b' = b - As' = A \cdot \Delta + e,$$

i.e., we replace the secret $s$ by $\Delta$.

Following [HSH⁺09,ADP18], let us assume a typical average bit flip rate of 0.55%. In order to estimate $w_\Delta$, we notice that a ternary NTRU secret key requires $2n$ bits of storage, since each coefficient occupies 2 bits. Therefore, we

| $(n, q, w, w_\Delta)$ | Rep-1 [bit], $\varepsilon$ | Lattices [ACD+18] $0.292\beta + 16.4$ |
|---|---|---|
| NTRU [CDH+20] | | |
| $(509, 2048, 254, 6)$ | **40**, [0] | 41 |
| $(677, 2048, 254, 8)$ | **42**, [0] | 48 |
| $(821, 4096, 510, 10)$ | 60, [2] | 56 |
| $(701, 8192, 468, 8)$ | **43**, [0] | 47 |
| NTRU Prime [BBC+20] | | |
| $(653, 4621, 288, 8)$ | **42**, [0] | 47 |
| $(761, 4591, 286, 9)$ | 57, [2] | 48 |
| $(857, 5167, 322, 10)$ | 60, [2] | 55 |
| BLISS I+II [DDLL13] | | |
| $(512, 12289, 154, 6)$ | 41, [0] | 38 |
| GLP I [GLP12] | | |
| $(512, 8383489, 342, 6)$ | 40, [0] | 33 |

Table 4: Bit complexities for the cold boot attack on NTRU-type encryption schemes and signatures. Lattice-based attacks are estimated using the results from [ACD+18].

expect $w_\Delta = \lceil 2n \cdot \frac{0.55}{100} \rceil$. For the concrete cryptographic parameters in Table 4 this translates to $w_\Delta$ in a range between 6 and 10.

We note that some implementations may choose to store the secret keys differently than just two bits per coefficient, and this will impact the efficiency of our cold boot attack. For example, [CDH+20] describes a compression mechanism of ternary keys to bit-strings. Thus, flipping one bit of the bit-string may impact many entries in the ternary key. For simplicity of exposition, we ignore such implementation subtleties here.

Let us now apply our Rep-1 attack to this new extremely sparse secret LWE setup. Concrete figures are given in Table 4. Since the secret is very sparse, we do not have to construct deep search trees to outperform lattice attacks. It is suffices to the consider depth-2 Rep-1 (or even sometimes Rep-0) algorithm. To estimate lattice-based attacks for sparse secret we use the estimator from [ACD+18] since it incorporates the so-called 'drop-and-solve' guessing technique for sparse secret, see [ACW19].

This 'drop-and-solve' technique can be applied as well to our algorithm: we guess that a certain $c$ coordinates of $s'$ are 0 and remove these columns from the matrix $A$. The probability of guessing the 0's correctly is $p_0 = \binom{w_\Delta}{n-c} / \binom{w_\Delta}{n}$. The LWE problem becomes easier as the dimension is now $n - c$, but the overall runtime has to take the guessing into account. We find the optimal choice for $c$ by exhaustive search. For our attack, the total saving is around a factor of 2 (i.e., one bit in the security level). For the parameter sets from Table 4 our Rep-1 attack performs similar to or even better than lattice-based attacks.

## References


ACD$^+$18. Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the LWE, NTRU schemes! In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 351–367. Springer, Heidelberg, September 2018. 13, 16

ACW19. Martin R. Albrecht, Benjamin R. Curtis, and Thomas Wunderer. Exploring trade-offs in batch bounded distance decoding. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 467–491. Springer, Heidelberg, August 2019. 16

ADP18. Martin R. Albrecht, Amit Deo, and Kenneth G. Paterson. Cold boot attacks on ring and module LWE keys under the NTT. *IACR TCHES*, 2018(3):173–213, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7273. 3, 15

AI06. Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th FOCS*, pages 459–468. IEEE Computer Society Press, October 2006. 4

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. 14

BBC$^+$20. Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, Chitchanok Chuengsatiansup, Tanja Lange, Adrian Marotzke, Bo-Yuan Peng, Nicola Tuveri, Christine van Vredendaal, and Bo-Yin Yang. NTRU Prime: round 3, 2020. https://ntruprime.cr.yp.to/nist/ntruprime-20201007.pdf. 2, 14, 15, 16, 20

BCJ11. Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 364–385. Springer, Heidelberg, May 2011. 2

BDGL16. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016. 13

BJMM12. Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, Heidelberg, April 2012. 2

CDH$^+$20. Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John M. Schanck, Tsunekazu Saito, Peter Schwabe, William Whyte, Keita Xagawa, Takashi Yamakawa, and Zhenfei Zhang. PQC round-3 candidate: NTRU. technical report, 2020. https://ntru.org/f/ntru-20190330.pdf. 2, 14, 15, 16, 20

DDGR20. Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358. Springer, Heidelberg, August 2020. 2, 3, 13, 14

DDLL13. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 40–56. Springer, Heidelberg, August 2013. 2, 14, 15, 16, 20

GLP12. Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 530–547. Springer, Heidelberg, September 2012. 2, 14, 15, 16, 20

HJ10. Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 235–256. Springer, Heidelberg, May / June 2010. 2

HPS98. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In *Proceedings of the Third International Symposium on Algorithmic Number Theory*, ANTS-III, page 267–288. Springer-Verlag, 1998. 2, 4

HSH+09. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, May 2009. 15

IEE08. IEEE standard specification for public key cryptographic techniques based on hard problems over lattices. *IEEE Std 1363.1-2008*, pages 1–81, 2008. 2, 3, 14, 15, 20

IM98. Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 604–613, 1998. 4, 5

Ind01. Piotr Indyk. On approximate nearest neighbors under $\ell_\infty$-norm. *Journal of Computer and System Sciences*, 63(4):627–638, 2001. 4

May21. Alexander May. How to meet ternary LWE keys. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 701–731, Virtual Event, August 2021. Springer, Heidelberg. 2, 3, 6, 9, 10

MO15. Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 203–228. Springer, Heidelberg, April 2015. 3

Ngu21. Phong Nguyen. Boosting the hybrid attack on ntru: Torus lsh, permuted hnf and boxed sphere, 2021. https://csrc.nist.gov/Presentations/2021/boosting-the-hybrid-attack-on-ntru. 4


## A More representations from the error vector

One of the alternatives to the approximate matching technique from the previous section is *exact* matching, where we explicitly enumerate all possible error vectors on the coordinates we merge on. The simplest algorithm, Rep-0, constructs a depth-2 tree of lists as follows. For some optimal integer $0 \leq k < n$ (we explain how to choose it below), the top-most lists are of the form (cf. Eqs. (5))

$$
\begin{aligned}
L_1^{(2)} &= \{(s_1^{(2)} \in \mathcal{T}^{\frac{n}{2}}\left(\frac{w}{8}\right) \times 0^{\frac{n}{2}})\} \times \{e_1^{(2)} \in \mathcal{T}^k\left(\frac{w_k}{4}\right)\} \\
L_2^{(2)} &= \{(s_2^{(2)} \in 0^{\frac{n}{2}} \times \mathcal{T}^{\frac{n}{8}}\left(\frac{w}{2}\right))\} \times \{e_2^{(2)} \in \mathcal{T}^k\left(\frac{w_k}{4}\right)\} \\
L_3^{(2)} &= \{(s_3^{(2)} \in \mathcal{T}^{\frac{n}{2}}\left(\frac{w}{8}\right) \times 0^{\frac{n}{2}})\} \times \{e_3^{(2)} \in \mathcal{T}^k\left(\frac{w_k}{4}\right)\} \\
L_4^{(2)} &= \{(s_4^{(2)} \in 0^{\frac{n}{2}} \times \mathcal{T}^{\frac{n}{2}}\left(\frac{w}{8}\right))\} \times \{e_4^{(2)} \in \mathcal{T}^k\left(\frac{w_k}{4}\right)\},
\end{aligned}
\tag{6}
$$

where $w_k$ is the expected weight of the error vector on $k$ coordinates. These lists are of size $\left|L_i^{(2)}\right| = \binom{n/2}{w/8, w/8, \cdot} \cdot \binom{w}{w_k/8, w_k/8, \cdot}$. Note that these lists, in addition to $s_i$'s, enumerate partial error vectors $e_i$ such that when we merge $L_1^{(2)}$ with $L_2^{(2)}$, and $L_3^{(2)}$ with $L_4^{(2)}$, we obtain several solutions to the equation

$$A(s_1^{(2)} + s_2^{(2)}) + (e_1 + e_2) = b - A(s_3^{(2)} + s_4^{(2)}) + (e_3 + e_4).$$

In particular, we have on expectation $\mathcal{R}_s = \binom{w/2}{w/4}^2$ representations of $s$ as $s = (s_1^{(2)} + s_2^{(2)}) + (s_3^{(2)} + s_4^{(2)})$, while for $e$ we have $\mathcal{R}_e = \binom{w_k/2}{w_k/4}^2$ representations of the form $e = (e_1^{(2)} + e_2^{(2)}) + (e_3^{(2)} + e_4^{(2)})$. In total, there are $\mathcal{R}_s \cdot \mathcal{R}_e$ representations of the solution $(s, e)$. Therefore, we construct a $1/(\mathcal{R}_s \cdot \mathcal{R}_e)$-fraction of all $(s_1^{(2)}, s_2^{(2)}; e_1^{(2)}, e_2^{(2)})$ by looking only at those that give $As_1^{(2)} + e_1^{(2)} = As_2^{(2)} - e_2^{(2)} \mod q$ on $k$ coordinates for the lists $L_1, L_2$, and only at those that give $As_3^{(2)} + e_3^{(2)} = b - As_4^{(2)} - e_4^{(2)} \mod q$ on $k$ coordinates for $L_3, L_4$. The $k$ is chosen such that $k \approx \log_q(\mathcal{R}_s \cdot \mathcal{R}_e)$, so on expectation one solution quadruple $(s_1^{(2)}, s_2^{(2)}; e_1^{(2)}, e_2^{(2)})$ survives.

After we merge and filter out pairs $(s_1^{(2)}, s_2^{(2)})$ that do not satisfy $s_1^{(2)} + s_2^{(2)} \in \mathcal{T}^n\left(\frac{w}{4}\right)$ (analogously for $e_1^{(2)} + e_2^{(2)}$, $s_3^{(2)} + s_4^{(2)}$, and $e_3^{(2)} + e_4^{(2)}$) we obtain the following two lists

$$
\begin{aligned}
L_1^{(1)} = \{s_1^{(1)} = s_1^{(2)} + s_2^{(2)} \in \mathcal{T}^n\left(\frac{w}{4}\right), e_1^{(1)} = e_1^{(2)} + e_2^{(2)} \in \mathcal{T}^k\left(\frac{w_k}{2}\right) &: \\
As_1^{(1)} + e_1^{(1)} = 0 \bmod q \text{ on } k \text{ coordinates}\} \\
L_2^{(1)} = \{s_2^{(1)} = s_3^{(2)} + s_4^{(2)} \in \mathcal{T}^n\left(\frac{w}{4}\right), e_2^{(1)} = e_3^{(2)} + e_4^{(2)} \in \mathcal{T}^k\left(\frac{w_k}{2}\right) &: \\
b - As_2^{(1)} - e_2^{(1)} = 0 \bmod q \text{ on } k \text{ coordinates}\}
\end{aligned}
$$

These lists are of size $|L_j| = \left|L_i^{(2)}\right|^2 / (\mathcal{R}_s \cdot \mathcal{R}_e)$. It remains to merge the elements from $L_1^{(1)}$ with the elements from $L_2^{(1)}$ on $n-k$ coordinates. We do that

| $(n, q, w)$ | Rep-0 | Rep-1 | $\varepsilon_s, \varepsilon_e$ depth 3 | Rep-1 | $\varepsilon_s, \varepsilon_e$ depth 4 |
|---|---|---|---|---|---|
| NTRU IEEE [IEE08] | | | | | |
| $(401, 2048, 226)$ | 251 | 196 | $[14, 4], [6, 0]$ | **173** | $[26, 10, 4], [0, 0, 0]$ |
| $(449, 2048, 268)$ | 279 | 217 | $[12, 4], [2, 0]$ | 189 | $[32, 10, 4], [0, 0, 0]$ |
| $(677, 2048, 314)$ | 403 | 290 | $[16, 6], [0, 0]$ | 258 | $[32, 16, 6], [6, 0, 0]$ |
| $(1087, 2048, 240)$ | 438 | 330 | $[16, 6], [0, 0]$ | 300 | $[34, 14, 2], [0, 0, 0]$ |
| $(541, 2048, 98)$ | 203 | 158 | $[18, 4], [0, 0]$ | 150 | $[22, 6, 4], [0, 0, 0]$ |
| $(613, 2048, 110)$ | 219 | 165 | $[8, 2], [0, 0]$ | 161 | $[20, 6, 4], [0, 0, 0]$ |
| $(887, 2048, 162)$ | 326 | 241 | $[18, 6], [0, 0]$ | 234 | $[26, 8, 0], [8, 2, 0]$ |
| $(1171, 2048, 212)$ | 428 | 326 | $[16, 4], [0, 0]$ | 297 | $[34, 10, 0], [0, 0, 0]$ |
| $(659, 2048, 76)$ | 184 | 147 | $[8, 0], [0, 0]$ | 147 | $[20, 8, 0], [0, 0, 0]$ |
| $(761, 2048, 84)$ | 204 | 156 | $[6, 0], [0, 0]$ | **156** | $[18, 6, 2], [0, 0, 0]$ |
| $(1087, 2048, 126)$ | 291 | 220 | $[16, 4], [0, 0]$ | **214** | $[24, 8, 2], [0, 0, 0]$ |
| $(1499, 2048, 158)$ | 385 | 286 | $[16, 2], [0, 0]$ | **280** | $[24, 8, 0], [8, 2, 0]$ |
| NTRUEnc [CDH+20] | | | | | |
| $(509, 2048, 254)$ | 300 | 228 | $[12, 4], [0, 0]$ | 198 | $[32, 10, 6], [0, 0, 0]$ |
| $(677, 2048, 254)$ | 360 | 265 | $[16, 6], [8, 2]$ | 235 | $[32, 12, 4], [0, 0, 0]$ |
| $(821, 4096, 510)$ | 521 | 402 | $[16, 6], [0, 0]$ | 365 | $[32, 18, 8], [8, 2, 0]$ |
| $(701, 8192, 468)$ | 464 | 358 | $[10, 8], [0, 0]$ | 296 | $[34, 16, 8], [0, 0, 0]$ |
| NTRU Prime [BBC+20] | | | | | |
| $(653, 4621, 288)$ | 366 | 270 | $[16, 6], [0, 0]$ | 237 | $[32, 14, 6], [0, 0, 0]$ |
| $(761, 4591, 286)$ | 403 | 299 | $[16, 6], [0, 0]$ | 269 | $[32, 16, 10], [4, 0, 0]$ |
| $(857, 5167, 322)$ | 468 | 339 | $[14, 6], [0, 0]$ | 317 | $[34, 14, 2], [0, 0, 0]$ |
| BLISS I+II [DDLL13] | | | | | |
| $(512, 12289, 154)$ | 316 | 244 | $[14, 4], [0, 0]$ | 208 | $[26, 16, 4][0, 0, 0]$ |
| GLP I [GLP12] | | | | | |
| $(512, 8383489, 342)$ | 327 | 250 | $[6, 4], [0, 0]$ | 214 | $[30, 14, 4][0, 0, 0]$ |

Table 5: Bit complexities of the Rep-0, Rep-1 with depth-3 search tree, and Rep-1 with depth-4 search tree algorithms with additional representations coming from enumerating the error vector. In some cases this approach gives slightly better results than the algorithm from Section 4. We mark them in bold.

with Oldyzko's LSH. Overall, the time and space complexities are determined by $\max\left\{\left|L_i^{(2)}\right|, \left|L_j^{(1)}\right|, \left|L_j^{(1)}\right|^2/2^{n-k}\right\}$ for $1 \leq i \leq 4$, $1 \leq j \leq 2$.

The concrete cost of this attack is given in the column Rep-0 in Table 5. From the table, one concludes that this approach performs similarly to the LSH Rep-0 algorithm from the Section 4. Similar to Rep-1 algorithms, the algorithm with representations for error becomes faster when we add representations of 0's and more levels. The details of this extension are given below. We note that with these additional representations we achieve the runtimes that are comparable with those for the LSH algorithms, cf. Table 5.

### A.1 Representations of 0 for the error vector

Let us consider an algorithm that constructs a depth-3 tree of lists. The reader is advised to follow Figure 3 while reading this description. We implicitly assume that all fractions that appear are integers by appropriate rounding. We count the levels from bottom to top starting with 0, e.g., on level 3 we have 8 lists. The algorithm is parametrised by two 2-dimensional arrays $\varepsilon_s$ and $\varepsilon_e$, whose values represent the number of additional 1's and -1's for the secret $s$ added on level 2 ($\varepsilon_s[2]$) and on level 1 ($\varepsilon_s[1]$). These values are subject to optimisations and, for concrete parameters, are given in Table 5.

On each level we target a certain weight of the secret $s^{(0)} := s$. Enumeration for $s$ here is exactly the same as in Section 2. That is, starting from the bottom, where the solution $s$ is of weight $w$, i.e., $s \in \mathcal{T}^n(w/2)$, we split $s$ into $s = s_1^{(1)} + s_2^{(1)}$, thus giving us, as in the previous section, $\binom{w/2}{w/4}^2$ ways to represent 1's and -1's in $s$, and $\binom{n-w}{\varepsilon_s[1], \varepsilon_s[1], \cdot}$ ways to represent 0's in $s$. The total number of representations for $s$ is on level 1 therefore,

$$\mathcal{R}_s^{(1)} = \binom{w/2}{w/4}^2 \cdot \binom{n-w}{\varepsilon_s[1], \varepsilon_s[1], \cdot}.$$

Next, we go one level up by splitting $s_1^{(1)}$ (analogously for $s_2^{(1)}$) into two vectors $s_1^{(2)}, s_2^{(2)}$, each from $\mathcal{T}^n(\frac{w}{8} + \frac{\varepsilon_s[1]}{2} + \varepsilon_s[2])$. Therefore, the 1's and -1's in $s_1^{(1)}$ can be represented in $\binom{w/4+\varepsilon_s[1]}{w/8+\varepsilon_s[1]/2}^2$ ways, while for 0's of $s_1^2$ we have $\binom{n-w/2-2\varepsilon_s[1]}{\varepsilon_s[2], \varepsilon_s[2], \cdot}$ representations. In total on level 2, we have

$$\mathcal{R}_s^{(2)} = \binom{w/4 + \varepsilon_s[1]}{w/8 + \varepsilon_s[1]/2}^2 \cdot \binom{n - w/2 - 2\varepsilon_s[1]}{\varepsilon_s[2], \varepsilon_s[2], \cdot}$$

representations for $s$. Depth-3 algorithm will enumerate them in the meet-in-middle way by considering $s_1^2 = s_1^3 + s_2^3$, where $s_i^3 \in \mathcal{T}^{n/2}(\frac{w}{16} + \frac{\varepsilon_s[1]}{4} + \frac{\varepsilon_s[2]}{2})$.

In order to understand how we enumerate partial errors, let us now go from top to bottom. On each level above the merging, we additionally enumerate the error vectors *for the coordinates we are going to merge on.* For example, on level 3, we enumerate all vectors $e_1, e_2$ from $\mathcal{T}^{k2}(\frac{w_{k2}}{4} + \varepsilon_e[2])$, where $w_{k2}$ is the expected weight of $e$ on some $k2$ coordinates.

$$L_i = \{s_i \in \mathcal{T}^{\frac{n}{2}}\left(\frac{w}{16} + \frac{\varepsilon[1]}{4} + \frac{\varepsilon[2]}{2}\right) \times 0^{\frac{n}{2}}\} \times \{e_i \in \mathcal{T}^{k2}\left(\frac{w_{k2}}{4} + \varepsilon_e[2]\right)\},\ i - \text{odd}$$

$$L_i = \{s_i \in 0^{\frac{n}{2}} \times \mathcal{T}^{\frac{n}{2}}\left(\frac{w}{16} + \frac{\varepsilon[1]}{4} + \frac{\varepsilon[2]}{2}\right)\} \times \{e_i \in \mathcal{T}^{k2}\left(\frac{w_{k2}}{4} + \varepsilon_e[2]\right)\},\ i - \text{even}.$$

We now explain how we choose $k2$. When we merge, say, $L_1^{(3)}$ with $L_2^{(3)}$ into $L_1^{(2)}$, we want to make sure that in $L_1^{(2)}$ there remains on expectation one solution quadruple $(s_1, e_1), (s_2, e_2)$ that satisfies $As_1 + e_1 = As_2 + e_2 \bmod q$. As usual, we do so by considering only those $(s_1, e_1), (s_2, e_2)$ for which $As_1 + e_1 = As_2 + e_2 \bmod q$ on $k2$ coordinates. Note that the number of ways to represent the error vector $e$ on $k2$ coordinates as $e = e_1 + e_2$ is

$$\mathcal{R}_e^{(2)} = \binom{w_{k2}/2}{w_{k2}/4}^2 \cdot \binom{n - w_{k2}}{\varepsilon_e[2], \varepsilon_e[2], \cdot},$$

where the first multiple counts the number of representations of 1's and -1's, while the second computes the number of representations of 0's in $e$ on $k2$ coordinates. Thus the value $k2$ is chosen to satisfy $k2 = \log_q(\mathcal{R}^{(2)}) = \log_q(\mathcal{R}_e^{(2)} \cdot \mathcal{R}_e^{(2)})$. This is an equation in $k2$ that can be found for concrete parameters.

The top-level lists are merged into 4 lists:

$$L_i^{(2)} = \{s_i^{(2)} \in \mathcal{T}^n\left(\frac{w}{8} + \frac{\varepsilon_s[1]}{2}\right), e_i \in \mathcal{T}^{k2}\left(\frac{w_{k2}}{2}\right) : As_i + e_i = 0 \bmod\ q \text{ on } k2 \text{ coordinates}\},$$

for $i \leq 4$. We augment each such list with the set $\{e_i \in \mathcal{T}^{k1}\left(w_{k1}/4 + \varepsilon[1]\right)\}$, where $k1$ is the number of coordinates we are going to merge on. Therefore, the number of representations on level 1, i.e., after we merge on level 2, is $\mathcal{R}^{(1)} = \mathcal{R}_e^{(1)} \cdot \mathcal{R}_s^{(1)}$, where

$$\mathcal{R}_e^{(1)} = \binom{w_{k1}/2}{w_{k1}/4}^2 \cdot \binom{n - w_{k1}}{\varepsilon_e[1], \varepsilon_e[1], \cdot},$$

and $k := k1 + k2 = \log_q(\mathcal{R}^{(1)})$. This gives us an equation in $k1$. We now have two lists $L_1^{(1)}, L_2^{(2)}$, which contain on expectation one pair $s_1^{(1)}, s_2^{(1)}$ that sums to the secret $s$ and one pair $e_1^{(1)}, e_1^{(2)}$ that sums to the error $e$ on $k$ coordinates. We find these elements using Odlyzko's LSH on the remaining $n - k$ coordinates.

The overall runtime is determined by the cost of constructing the most expensive level (we remove the subscripts in the lists, since the lists on the same level are expected to have the same size):

$$T = \max\left\{\left|L^{(3)}\right|, \frac{\left|L^{(3)}\right|^2}{q^{k1}}; \left|L^{(2)}\right|, \frac{\left|L^{(2)}\right|^2}{q^{k2}}; \left|L^{(1)}\right|, \frac{\left|L^{(1)}\right|^2}{2^{n-k1-k2}}\right\}.$$

For concrete parameters the value for $T$ is found by optimising $k_1$ and $k_2$.

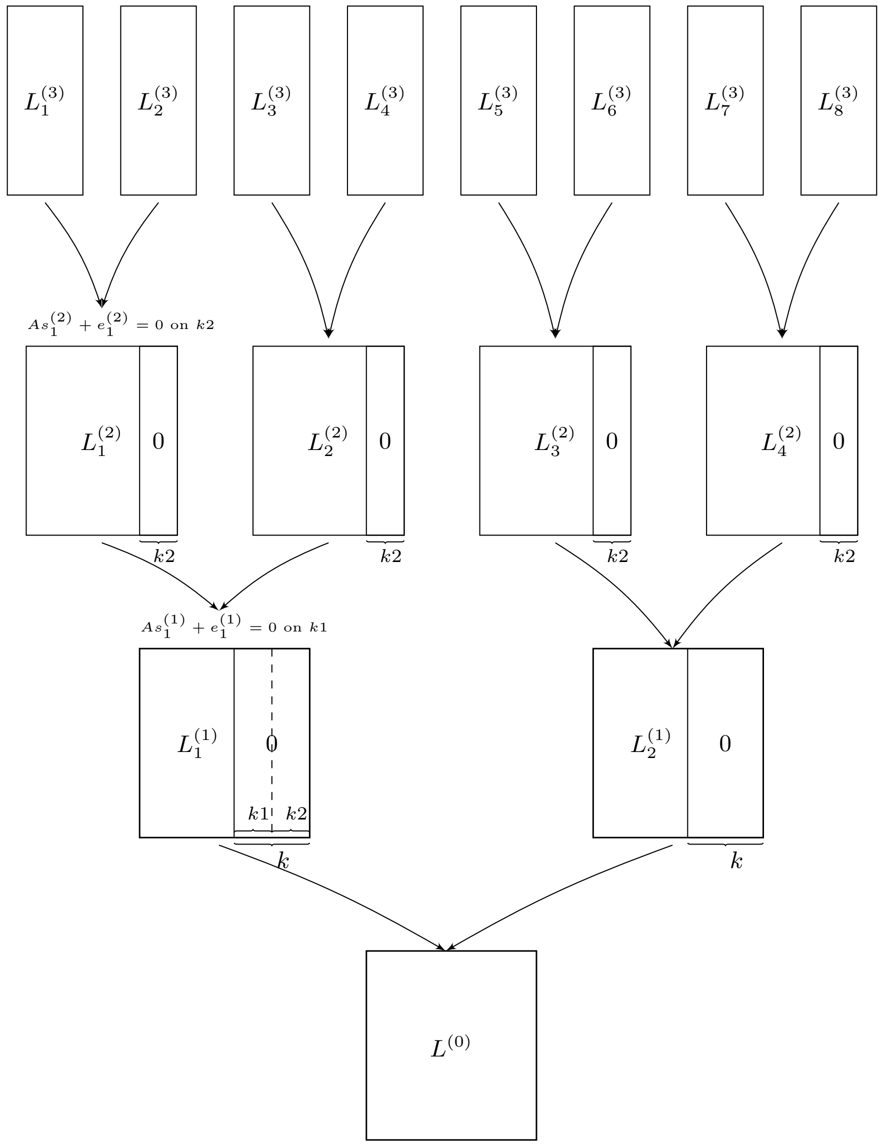


Fig. 3: Rep-1 algorithm of depth-3 with representations for the error vector