Low Memory Attacks on Small Key CSIDH

Jesús-Javier Chi-Domínguez[1], Andre Esser[1], Sabrina Kunzweiler[2,3], and Alexander May[3]

[1] Technology Innovation Institute, UAE
{jesus.dominguez, andre.esser}@tii.ae
[2] Univ. Bordeaux, CNRS, Bordeaux INP, Inria, France
sabrina.kunzweiler@math.u-bordeaux.fr
[3] Ruhr University Bochum, Germany
alex.may@rub.de

Abstract. Despite recent breakthrough results in attacking SIDH, the CSIDH protocol remains a secure post-quantum key exchange protocol with appealing properties. However, for obtaining efficient CSIDH instantiations one has to resort to small secret keys. In this work, we provide novel methods to analyze small key CSIDH, thereby introducing the representation method —that has been successfully applied for attacking small secret keys in code- and lattice-based schemes— also to the isogeny-based world.

We use the recently introduced Restricted Effective Group Actions (REGA) to illustrate the analogy between CSIDH and Diffie-Hellman key exchange. This framework allows us to introduce a REGA-DLOG problem as a level of abstraction to computing isogenies between elliptic curves, analogous to the classic discrete logarithm problem. This in turn allows us to study REGA-DLOG with ternary key spaces such as $\{-1,0,1\}^n, \{0,1,2\}^n$ and $\{-2,0,2\}^n$, which lead to especially efficient, recently proposed CSIDH instantiations. The best classic attack on these key spaces is a Meet-in-the-Middle algorithm that runs in time $3^{0.5n}$, using also $3^{0.5n}$ memory.

We first show that REGA-DLOG with ternary key spaces $\{0,1,2\}^n$ or $\{-2,0,2\}^n$ can be reduced to the ternary key space $\{-1,0,1\}^n$.

We further provide a heuristic time-memory tradeoff for REGA-DLOG with keyspace $\{-1,0,1\}^n$ based on Parallel Collision Search with memory requirement M that under standard heuristics runs in time $3^{0.75n}/M^{0.5}$ for all $M \leq 3^{n/2}$. We then use the representation technique to heuristically improve to $3^{0.675n}/M^{0.5}$ for all $M \leq 3^{0.22n}$, and further provide more efficient time-memory tradeoffs for all $M \leq 3^{n/2}$.

Although we focus in this work on REGA-DLOG with ternary key spaces for showing its efficacy in providing attractive time-memory tradeoffs, we also show how to use our framework to analyze larger key spaces $\{-m,\dots,m\}^n$ with $m = 2,3$.

Keywords: Isogeny · Time-Memory Trade-off · Representation Technique

1 Introduction

In the pre-quantum era the Diffie-Hellman protocol plays a paramount role in securely exchanging secret keys. Diffie-Hellman shows its outstanding performance when instantiated with sufficiently generic elliptic curve groups with prime order q, since for solving the discrete logarithm in these groups on classical computers only generic algorithms with square-root time complexity $\Theta(\sqrt{q})$ are known.

Such a complexity allows for extremely efficient instantiations that provide e.g. 128 bit classical security for 256-bit group order q. Since Shor's algorithm [43] generically breaks discrete logarithms in every commutative group of order q in time polynomial in $\log q$, Diffie-Hellman unfortunately becomes completely insecure in a quantum world.

The current post-quantum substitutes for key exchange primarily stem from lattice problems, like Kyber [11], or from decoding problems, like McEliece [4,35]. However, in both cases we have already classical algorithms that are below square root complexity [6,40]. As a consequence, lattice- and code-based schemes can inherently not achieve the efficiency of the Diffie-Hellman protocol. For exploiting smallness of secret keys, the *representation technique* has been quite successfully applied first in the coding world [7,12,21,32,34], and then subsequently also for lattice-based schemes [22,26,31,45].

Ideally, in a quantum world we would replace Diffie-Hellman by a protocol for which

(a) the best classical algorithm achieves square root complexity, while
(b) the best quantum algorithm does not provide a significant speedup.

Within the last decade isogeny-based cryptography developed as a promising candidate to provide an analogue of Diffie-Hellmann key exchange in the quantum world.

Its hardness is based on the difficulty of computing isogenies between supersingular elliptic curves. If extra information is available, like in the SIDH proposal [28], then recent breakthrough results [14,30,41] show a collapse of the problem's complexity, leading to a devastating attack on the SIDH cryptosystem.

In contrast to that, in the CSIDH cryptosystem [15] no extra information is available to an attacker and the underlying isogeny computation problem remains hard. The construction is made possible by restricting to the set of supersingular elliptic curves defined over a prime field $\mathbb{F}_p$. This set has cardinaltity $N \approx \sqrt{p}$, and the best classical algorithm to recover a secret isogeny is a Meet-in-the-Middle algorithm with square root complexity $\mathcal{O}(\sqrt{N})$. However, the best quantum algorithm, due to Kuperberg [29], is subexponential in $\log N$, with complexity $2^{\mathcal{O}(\sqrt{\log N})}$.

Current CSIDH instantiations. To guard against Kuperberg-style attacks [10,17,39], recent CSIDH instantiations recommend to use 512, 1024 or even 2048-bit field size for $\mathbb{F}_p$. To still retain highly efficient cryptosystems, current proposals [5,15–18,27,36–38] suggest to use secret keys from (sub-) sets of $\{-m,\ldots,m\}^n$ of constant width m, for highly practical schemes like [17] even restricted to ternary key spaces

$$\mathrm{SK}_1 = \{-1,0,1\}^n,\ \mathrm{SK}_2 = \{0,1,2\}^n, \text{ or } \mathrm{SK}_3 = \{-2,0,2\}^n.$$

Ternary keys can be guessed in 3^n steps, and the currently best

(a) classical algorithm for recovering ternary keys is a Meet-in-the-Middle algorithm with square root time and space complexity $3^{n/2}$,
(b) whereas the best quantum algorithm [44] is a mere quantum version of Meet-in-the-Middle, called claw-finding, providing a rather modest speedup to $3^{n/3}$.

Our contributions. We use the Restricted Effective Group Action (REGA) framework, recently introduced in [2]. This abstraction can e.g. be instantiated via the isogeny-based CSIDH group action. Group elements are represented by vectors $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{Z}^n$, efficient implementations require to restrict the vector entries v_i to a small range $\{-m, \dots, m\}$ for some constant m. Highly efficient implementations like [17] choose ternary key spaces for $\mathbf{v}$.

For REGAs we introduce a $\mathsf{REGA\text{-}DLOG}$ problem that denotes the secret key recovery problem in REGA-based cryptography, and resembles the dlog problem for the Diffie-Hellman protocol. As a special case, $\mathsf{REGA\text{-}DLOG}_m$ denotes the secret key recovery problem for secret keys chosen from a small range set $\{-m, \dots, m\}^n$.

We show that the best CSIDH attacks, such as the Pollard-style algorithm going back to Galbraith-Hess-Smart [25] for smallish p and Meet-in-the-Middle (MitM) for small m, generalize to the REGA setting. For ternary key settings we show that $\mathsf{REGA\text{-}DLOG}$ for the key spaces $\mathrm{SK}_1 = \{-1, 0, 1\}^n$ and $\mathrm{SK}_2 = \{0, 1, 2\}^n$ is equivalently hard, and at least as hard as for $\mathrm{SK}_3 = \{-2, 0, 2\}^n$. Therefore, for ternary keys it suffices to concentrate on $\mathsf{REGA\text{-}DLOG}_1$ with keyspace SK_1.

Since $|\mathrm{SK}_1| = 3^n$, our MitM achieves for $\mathsf{REGA\text{-}DLOG}_1$ run time $3^{0.5n}$ with memory consumption also $3^{0.5n}$. We then generalize the best time-memory CSIDH trade-off [1,8,17,19] based on Parallel Collision Search (PCS), due to van Oorschot and Wiener [46] to the $\mathsf{REGA\text{-}DLOG}_1$ setting resulting for memory $M \leq 3^{0.5n}$ in run time

$$T = 3^{0.75n}/M^{0.5}.$$

Notice that for maximal memory $M = 3^{0.5n}$ we again achieve MitM complexity $T = 3^{0.5n}$. However for constant M, also called the *memory-less* setting, we achieve a $T = 3^{0.75n}$ algorithm. See the dotted line (PCS) in Figure 1 for a visualization of the interpolation between the run time exponents 0.75 and 0.5.

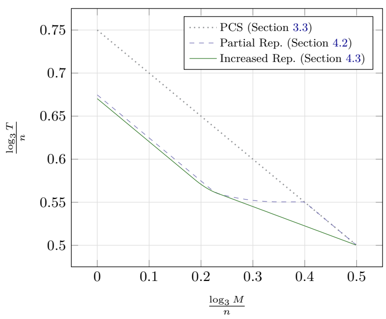


Fig. 1: Complexities for solving $\mathsf{REGA\text{-}DLOG}_1$.

The REGA setting is not only a natural abstraction of isogeny-based group actions, but it also allows us —analogous to codes [7, 24, 32], lattices [26, 31, 45], and low weight discrete logarithms [23, 33]— to naturally exploit the algebraic-combinatorial benefits of using small secret keys from $\mathbb{Z}^n$.

Namely, by additively splitting a secret key $\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2$ with many *representations* $\mathbf{v}_1, \mathbf{v}_2 \in \{-1, 0, 1\}^n$ we significantly improve the standard PCS time-memory trade-off for $M \leq 3^{0.22n}$ to

$$3^{0.675n}/M^{0.5}.$$

Hence for memory $M = 3^{0.22n}$, which is less than the square root of MitM's memory $3^{0.5n}$, we achieve run time $3^{0.565n}$, only slightly inferior to MitM's time $3^{0.5n}$. In the memory-less setting, we obtain a $3^{0.675n}$-algorithm. The tradeoff is visualized as a dashed line (Partial Rep.) in Figure 1.

Using more elaborate representations $\mathbf{v}_1, \mathbf{v}_2 \in \{-2, \ldots, 2\}^n$ of the ternary secret key, we further improve as visualized by the solid green line (Increased Rep.) in Figure 1. Especially, we obtain a memory-less $T = 3^{0.671n}$-algorithm, and a natural interpolation to the exponent point $(0.5, 0.5)$ from MitM. For larger values of $m \in \{2, 3\}$ we observe that the runtime exponent c in $T = (2m+1)^{cn}$, actually improves, we obtain for example memory-less algorithms with time $5^{0.629n}$ $(m = 2)$ and $7^{0.618n}$ $(m = 3)$.

Limitations of our approach. Since all our algorithms are based on collision finding techniques, their expected run times are proven under the standard mild heuristic that the constructed functions behave like random functions with respect to collision search.

Moreover, we assume throughout the paper for sake of simplicity that a random ternary secret key $\mathbf{v} \in \{-1, 0, 1\}^n$ achieves its expected number of $n/3$ entries for each of -1, 0, and 1, respectively, i.e., an equal weight distribution.

However, these limitations are no serious restrictions. First, keys with equal weight distribution have maximal entropy among all ternary keys and thus constitute the worst-case for the standard MitM algorithm (over which we improve). Second, it is not hard to see that keys with equal weight distribution amount to a polynomial fraction of all ternary keys. And last but not least, we show that for almost all randomly chosen ternary keys one can with sub-exponential overhead always enforce an equal weight distribution.

Potential Impact of Our Representation-based Results. Current efficient CSIDH-proposals like [17] define security levels with a memory complexity M significantly smaller than their run time complexity T. For instance [17] suggests 3 parameters sets with

$$(M_1, M_2, M_3) = (2^{80}, 2^{100}, 2^{119}) \text{ and } (T_1, T_2, T_3) = (2^{128}, 2^{128}, 2^{192})$$

for achieving NIST security level $L1, L2, L3$, respectively. The authors of [17] use a PCS-based approach for their analysis. Assuming that PCS has similar polynomial overheads as our representation method (which is certainly a complexity underestimation of the latter), for memories M_1, M_2, M_3 our representation method yields a reduced security level by 4.5, 8 and 13 bit, respectively.

Whether security bit reductions of these orders can be achieved in practice has to be validated by experiments, which are out of the scope of this work.

Organisation of our paper. In Section 2 we recall the definition of Restricted Effective Group Actions (REGA) and present a REGA-based key exchange modelling CSIDH. Further we define REGA-DLOG, the main hardness problem underlying this scheme, and its small key variant $\mathsf{REGA\text{-}DLOG}_m$. We also show that $\mathsf{REGA\text{-}DLOG}_1$ is hardest with ternary keys from $\{-1, 0, 1\}^n$.

In Section 3 we generalize known cryptanalytic results such as a Pollard-style algorithm (Section 3.1), MitM (Section 3.2), and Parallel Collision Search (Section 3.3) to $\mathsf{REGA\text{-}DLOG}_m$.

In Sections 4.1 and 4.2 we introduce representation-based algorithms for $\mathsf{REGA\text{-}DLOG}_1$, and provide a more elaborate version in Section 4.3 . The case of keys with non-equal weight distribution is discussed in Section 4.4. Section 4.5 addresses $\mathsf{REGA\text{-}DLOG}_m$ for larger $m = 2, 3$. Eventually, in Section 4.6 we discuss the possible practical impact of the attack.

2 Preliminaries

The Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) protocol [15] is a promising candidate for quantum-secure cryptography. Similar as its predecessor, the Couveignes-Rostovtsev-Stolbunov (CRS) scheme [20, 42], it is based on a commutative group action $\mathcal{G} \times \mathcal{X} \to \mathcal{X}$. While the underlying mathematics is quite involved, there exists a simple abstraction in the framework of *cryptographic group actions*. This framework was first introduced by Couveignes [20] under the name *hard homogenous spaces*. A more modern treatment is given in [3]. In particular the latter work also introduces *restricted effective group actions* (REGA) which model the properties of the CSIDH-based group action more closely, hence we use that framework in our analysis.

2.1 Restricted Effective Group Actions

This part follows the description of restricted effective group actions in [3] with some small modifications explained in Remark 2.2.

Definition 2.1 (Group Action). *Let $(\mathcal{G}, \circ)$ be a group with identity element $id \in \mathcal{G}$, and $\mathcal{X}$ a set. A map*

$$\star : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$$

is a group action if it satisfies the following properties:

1. *Identity: $id \star x = x$ for all $x \in \mathcal{X}$.*
2. *Compatibility: $(g \circ h) \star x = g \star (h \star x)$ for all $g, h \in \mathcal{G}$ and $x \in \mathcal{X}$.*

Remark 2.1. In practice, one often requires that a group action is regular. This means that for any $x, y \in \mathcal{X}$ there exists precisely one $g \in \mathcal{G}$ satisfying $y = g \star x$. For instance, this is the case for the CSIDH group action which we discuss in Section 2.3.

Definition 2.2 (Effective Group Action). *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a group action satisfying the following properties:*

1. *The group $\mathcal{G}$ is finite, commutative, and there exist efficient (PPT) algorithms for membership and equality testing, (random) sampling, group operation and inversion.*
2. *The set $\mathcal{X}$ is finite and there exist efficient algorithms for membership testing and to compute a unique representation.*
3. *There exists a distinguished element $\tilde{x} \in \mathcal{X}$ with known representation.*
4. *There exists an efficient algorithm to evaluate the group action, i.e., to compute $g \star x$ given g and x.*

Then we call $\tilde{x} \in \mathcal{X}$ the origin and $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ an effective group action (EGA).

In practice, the requirements from the definition of EGA are often too strong. The limitations are reflected in the weaker notion of restricted effective group actions.

Definition 2.3 (Restricted Effective Group Action). *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a group action and let $\boldsymbol{g} = (g_1, ..., g_n)$ be a set of elements in $\mathcal{G}$ and denote $\mathcal{H} = \langle g_1, \ldots, g_n \rangle$ for the subgroup generated by these elements. Assume that the following properties are satisfied:*

1. *The group $\mathcal{G}$ is finite, commutative, and $n = poly(\log(\#\mathcal{H}))$.*
2. *The set $\mathcal{X}$ is finite and there exist efficient algorithms for membership testing and to compute a unique representation.*
3. *There exists a distinguished element $\tilde{x} \in \mathcal{X}$ with known representation.*
4. *There exists an efficient algorithm that given $g_i \in \boldsymbol{g}$ and $x \in \mathcal{X}$, outputs $g_i \star x$ and $g_i^{-1} \star x$.*

Then we call $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x})$ a restricted effective group action (REGA).

Remark 2.2. Note that our definitions for EGA and REGA slightly differ from those in [2]. First, we require that the underlying group $\mathcal{G}$ is commutative. This allows us to formulate a group action based Diffie-Hellman protocol and it is the only relevant case for our analysis. Second, we dropped the condition that the set $(g_1, \ldots, g_n)$ in the definition of REGA is a generating set for $\mathcal{G}$. This is necessary to include CSIDH as a possible instantiation of a REGA. In that setting, it is an open problem to determine a (compact) set of generators for the entire group $\mathcal{G}$. But heuristically a set of generators for a large subgroup $\mathcal{H} \subset \mathcal{G}$ is known. More details are provided in Section 2.3.

Vector representation. Let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x})$ be a REGA with $\mathbf{g} = (g_1, \ldots, g_n)$. Elements in $\mathcal{H}$ can be represented as vectors $\mathbf{v} \in \mathbb{Z}^n$ under the mapping $\phi : \mathbb{Z}^n \to \mathcal{H}$, where

$$\phi : \mathbf{v} = (v_1, \ldots, v_n) \mapsto \prod_{i=1}^{n} {g_i}^{v_i}.$$

Note that this representation depends on the choice of generating set $\mathbf{g}$ for $\mathcal{H}$. And even fixing a set $\mathbf{g}$, the representation is not unique. More precisely, the kernel of the map ϕ is a lattice in $\mathbb{Z}^n$ which is in general not known explicitly.

Via the map ϕ, we define the action of $\mathbb{Z}^n$ on $\mathcal{X}$. Slightly abusing notation, we denote $\mathbf{v} \star x = \phi(\mathbf{v}) \star x$. Given a vector $\mathbf{v} \in \mathbb{Z}^n$, the action $\mathbf{v} \star x$ can be efficiently evaluated for any $x \in \mathcal{X}$ provided that the norm $\|\mathbf{v}\|$ is polynomial in $\log(\#\mathcal{H})$.

We highlight the following properties of the group action that will become important in our analysis. For any $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{Z}^n$ and $x, y \in \mathcal{X}$ it holds that

- $\mathbf{v} \star (\mathbf{u} \star x) = (\mathbf{u} + \mathbf{v}) \star x = \mathbf{u} \star (\mathbf{v} \star x)$,
- $y = (\mathbf{u} + \mathbf{v}) \star x$ implies $\mathbf{v} \star x = -\mathbf{u} \star y$,
- $x = \mathbf{v} \star (-\mathbf{v} \star x)$,
- if $\mathbf{w} \star x = (\mathbf{u} + \mathbf{v}) \star y$, then $(\mathbf{w} - \mathbf{v}) \star x = \mathbf{u} \star y$.

These properties immediately follow from the fact that $\star : \mathbb{Z}^n \times \mathcal{X} \to \mathcal{X}$ is a commutative group action.

Random sampling. In applications, it is often required to sample elements from $\mathcal{H}$. If the structure of $\mathcal{H}$ (in other words $\ker(\phi)$) is not known explicitly, then it is not possible to sample elements uniformly at random. Instead, vectors are sampled from some finite subset $S \subset \mathbb{Z}^n$. For a perfect uniform sampling, the map $\phi_{|S}$ would need to be bijective. In practice, one often uses $S = \{-m, \dots, m\}^n \subset \mathbb{Z}^n$ for some positive integer m. Here, m should be chosen small enough so that for two random vectors $\mathbf{v}, \mathbf{w} \in S$ the probability for $\mathbf{v} - \mathbf{w} \in \ker(\phi)$ is low. Note that this also requires that the generators $g_1, \dots, g_n$ are evenly distributed in the group. On the other hand, if one intends to sample from a large portion of the whole group $\mathcal{H}$, then m must be large enough so that $\phi_{|S}$ is (almost) surjective. However, in some settings it is sufficient to sample elements only from a small part of the group. We already note that this is the case for the key spaces studied in our paper.

2.2 Cryptographic Group Actions and Computational Assumptions

Given an effective group action $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ one can construct a Diffie-Hellman key exchange. The setup chooses a distinguished element $x_0 \in \mathcal{X}$. Then the secret keys of Alice and Bob are group elements $g_a, g_b \in \mathcal{G}$ respectively, and the corresponding public keys are $x_a = g_a \star x_0$ and $x_b = g_b \star x_0$. Now the shared key can be computed as $x_{ab} = g_a \star x_b = g_b \star x_a$. For this protocol to be secure, the following two problems need to be hard.

1. GA-DLOG: Given $(x, y) \in \mathcal{X}^2$, determine $g \in \mathcal{G}$ such that $y = g \star x$.
2. GA-CDH: Given $(x, y, z) \in \mathcal{X}^3$, determine $w \in \mathcal{X}$ such that there exists $g \in \mathcal{G}$ with $y = g \star x$ and $w = g \star z$.

These problems are the natural generalizations of the discrete logarithm problem and the computational Diffie-Hellman problem in the classical prime-order group setting. As in [3], we refer to group actions satisfying these hardness assumptions as *cryptographic group actions.*

In the REGA setting the random sampling of group elements (i.e. secret keys) is not straightforward. A variant of the Diffie-Hellman key exchange adapted to this setting is described in Figure 2. In essence, this is an abstract description of the CSIDH protocol introduced in [15], see also Section 2.3. The security of this key exchange not only relies on the hardness of GA-DLOG and GA-CDH for the group $\mathcal{G}$,[4] but also on the following variant of GA-DLOG which takes into account the choice of the secret keyspace.

Setup: A REGA $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x})$ with $\mathbf{g} = (g_1, \ldots, g_n)$ and a finite set $\mathrm{SK} \subset \mathbb{Z}^n$.
Key generation: Alice generates a private key $\mathbf{a} \in \mathrm{SK}$ and computes the public key $x_a = \mathbf{a} \star \tilde{x}$. Analogously, Bob generates a private key $\mathbf{b} \in \mathrm{SK}$ and computes the public key $x_b = \mathbf{b} \star \tilde{x}$.
Key exchange: Upon receiving Bob's public key, Alice computes $K_a = \mathbf{a} \star x_b$ and similarly Bob computes $K_b = \mathbf{b} \star x_a$. Note that

$$\mathbf{a} \star x_b = \mathbf{a} \star (\mathbf{b} \star \tilde{x}) = (\mathbf{a} + \mathbf{b}) \star \tilde{x} = (\mathbf{b} + \mathbf{a}) \star \tilde{x} = \mathbf{b} \star (\mathbf{a} \star \tilde{x}) = \mathbf{b} \star x_a,$$

hence $K_a = K_b$ is the shared secret.

Fig. 2: A REGA-based Diffie-Hellman protocol.

Definition 2.4 **(REGA-DLOG$_{\mathrm{SK}}$).** *Let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x})$ be a REGA with $\boldsymbol{g} = (g_1, \ldots, g_n)$ and $\mathrm{SK} \subset \mathbb{Z}^n$ a finite subset. Given $(x, y) \in \mathcal{X}^2$, determine $\mathbf{v} \in \mathrm{SK}$ such that $y = \mathbf{v} \star x$ if such a vector $\mathbf{v}$ exists.*

We say that the tuple $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \boldsymbol{g}, x, y)$ is an instance of the REGA-DLOG$_{\mathrm{SK}}$*. In the special case where $\mathrm{SK} = \{-m, \ldots, m\}^n$ for some $m \in \mathbb{N}$, we write REGA-DLOG$_m$ for short.*

Remark 2.3. Breaking the REGA-DLOG$_{\mathrm{SK}}$ assumption corresponds to recovering the secret key of the REGA-based Diffie-Hellman scheme described in Figure 2. We would like to point out that in order to break the scheme, it is sufficient to recover any (compact) vector representation of the secret key. More precisely, if $\mathbf{a} \in \mathrm{SK}$ is Alice's secret key and an attacker finds some $\hat{\mathbf{a}} \in \mathbb{Z}^n$ that satisfies $\phi(\hat{\mathbf{a}}) = \phi(\mathbf{a}) \in \mathcal{H}$, then he can compute the shared key as $K_{\hat{a}} = \hat{\mathbf{a}} \star x_b = \mathbf{a} \star x_b = K_a$. Of course, this further requires that the evaluation $\hat{\mathbf{a}} \star x_b$ is efficiently computable.

In the following we compare the keyspace $\mathrm{SK} = \{-m, \ldots, m\}^n$ to other choices from the literature of same cardinality. In particular the next lemma shows that it suffices to focus on the analysis of REGA-DLOG$_m$ among these choices.

Lemma 2.1. *Let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x})$ be a REGA with $\boldsymbol{g} = (g_1, \ldots, g_n)$. Let $m \in \mathbb{N}$ and consider $\mathrm{SK}_1 = \{-m, \ldots, m\}^n$, $\mathrm{SK}_2 = \{0, \ldots, 2m\}^n$, $\mathrm{SK}_3 = \{-2m, -2(m-1), \ldots, 2m\}^n$.*

1. *Then REGA-DLOG$_{\mathrm{SK}_1}$ and REGA-DLOG$_{SK_2}$ are equivalent.*

Further let $\tilde{\mathcal{H}} = \{g \circ g \mid g \in \mathcal{H}\} \subset \mathcal{H}$, and $\tilde{\boldsymbol{g}} = (\tilde{g_1} = g_1 \circ g_1, \ldots, \tilde{g_n} = g_n \circ g_n)$.

[4] More precisely, it relies on slightly modified versions of the problems, where the adversary additionally knows that there exists a solution with $g \in \mathcal{H} \subset \mathcal{G}$.

2. *An instance* $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \boldsymbol{g}, x, y)$ *of* $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_3}$ *can be transformed to an instance* $\big(\mathcal{G}, \tilde{\mathcal{H}}, \mathcal{X}, \star, \tilde{x}, \tilde{\boldsymbol{g}}, x, y\big)$ *of* $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_1}$.
3. *In particular if* $\#\mathcal{H}$ *is odd, then* $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_3}$ *reduces to* $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_1}$.

Proof. Let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ be an instance of $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_1}$. Define $y' = \mathbf{m} \star y$, where $\mathbf{m} = (m, \ldots, m) \in \mathbb{Z}^n$. Then $\mathbf{w} \in \{0, \ldots, 2m\}^n$ solves $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_2}$ on input $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y')$ if and only if $\mathbf{v} = \mathbf{w} - \mathbf{m}$ solves $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_1}$ on input $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$. In the same way, any instance of $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_2}$ can be transformed to an instance of $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_1}$. This proves the first part of the lemma.

Now consider an instance $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ of $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_3}$. Let $\tilde{\mathcal{G}}$ and $\tilde{\mathbf{g}}$ as defined in the statement of the lemma. Note that $\tilde{\mathcal{H}}$ is a subgroup of $\mathcal{H}$, and $\tilde{\mathbf{g}}$ is a generating set for this group. Moreover if a solution $\mathbf{v} \in \mathrm{SK}_3$ to the $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_3}$ instance exists, then $\phi(\mathbf{v}) \in \tilde{\mathcal{H}}$. As explained in Section 2.1, the vector representation of group elements depends on the choice of generators. For a vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathrm{SK}_3$, we define $\tilde{\mathbf{v}} = (\frac{v_1}{2}, \ldots, \frac{v_n}{2}) \in \mathrm{SK}_1$. Then the vectors $\mathbf{v}$ and $\tilde{\mathbf{v}}$ represent the same element in $\tilde{\mathcal{H}} \subset \mathcal{H}$ with respect to $\mathbf{g}$ and $\tilde{\mathbf{g}}$ respectively. In other words $\prod_{i=1}^n g_i^{v_i} = \prod_{i=1}^n \tilde{g}_i^{\tilde{v}_i} \in \tilde{\mathcal{H}}$. In particular $\mathbf{v}$ solves $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_3}$ on input $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ if and only if $\tilde{\mathbf{v}}$ solves $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_1}$ on input $\big(\mathcal{G}, \tilde{\mathcal{H}}, \mathcal{X}, \star, \tilde{x}, \tilde{\mathbf{g}}, x, y\big)$.

If $\#\mathcal{H}$ is odd, then $\tilde{\mathcal{H}} = \mathcal{H}$ and $\tilde{\mathcal{X}} = \mathcal{X}$. This observation implies the last part of the lemma. □

Remark 2.4. Note that in general, $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_3}$ and $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_1}$ are not equivalent even if $\#\mathcal{H}$ is odd. To see this, consider an instance $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ of $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_1}$. Theoretically, this can be transformed to the instance $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \tilde{\mathbf{g}}, x, y)$ of $\mathsf{REGA\text{-}DLOG}_{\mathrm{SK}_3}$, where $\tilde{\mathbf{g}} = (\sqrt{g_1}, \ldots, \sqrt{g_n})$. Here $\sqrt{g}$ denotes the (unique) element in $\mathcal{G}$ satisfying $\sqrt{g} \circ \sqrt{g} = g$. There are two issues with this transformation:

- It is not clear how to compute the elements $\sqrt{g_i}$ for $i \in \{1, \ldots, n\}$.
- If the group structure is known, one can compute $\sqrt{g_i} = g_i^{(r_i+1)/2}$, where $r_i = \mathrm{ord}(g_i)$. However the integers r_i are only bounded by $\#\mathcal{H}$, hence the evaluation of $\sqrt{g_i} \star x$ for some $x \in \mathcal{X}$ might require exponential time.

2.3 Isogeny-based REGAs

An important instantiation of REGAs is provided by isogeny-based group actions. Here, we explain the *Commutative Supersingular Isogeny Diffie-Hellman* (CSIDH) group action.

Let p be a large prime of the form $p = 4 \cdot \ell_1 \cdots \ell_d - 1$, where the ℓ_i are small distinct odd primes. Fix the elliptic curve $E_0 : y^2 = x^3 + x$ over $\mathbb{F}_p$. The curve E_0 is supersingular and its $\mathbb{F}_p$-rational endomorphism ring is $\mathcal{O} = \mathbb{Z}[\pi]$, where π is the Frobenius endomorphism.[5] Let $\mathcal{E}\ell\ell_p(\mathcal{O})$ be the set of $\mathbb{F}_p$-isomorphism classes

[5] Note that we later use $\mathcal{O}$ also in the context of standard Landau notation for complexity statements, however, its meaning will be clear from the context.

of elliptic curves defined over $\mathbb{F}_p$, with endomorphism ring $\mathcal{O}$. In our setting, an equivalent definition is

$$\mathcal{E}\ell\ell_p(\mathcal{O}) = \{E_A : y^2 = x^3 + Ax^2 + x \mid A \in \mathbb{F}_p \text{ and } E_A \text{ is supersingular}\}.$$

The ideal class group $cl(\mathcal{O})$ acts on the set $\mathcal{E}\ell\ell_p(\mathcal{O})$, i.e., there is a map

$$\begin{aligned} \star : cl(\mathcal{O}) \times \mathcal{E}\ell\ell_p(\mathcal{O}) &\to \mathcal{E}\ell\ell_p(\mathcal{O}) \\ ([\mathfrak{a}], E) &\mapsto [\mathfrak{a}] \star E, \end{aligned}$$

satisfying the properties from Definition 2.1 [15, Theorem 7].

The set

$$\mathbf{g} = ([\mathfrak{l}_1], \ldots, [\mathfrak{l}_n]), \quad \text{where } \mathfrak{l}_i = (\ell_i, \pi - 1) \triangleleft \mathcal{O}, \quad \text{for some } n \leq d$$

generates a large subgroup $\mathcal{H} \subset cl(\mathcal{O})$. The analysis in the original CSIDH paper [15] already implies that under some heuristics $(cl(\mathcal{O}), \mathcal{H}, \mathcal{E}\ell\ell_p(\mathcal{O}), \star, E_0)$ is a REGA. We summarize the most important properties.

1. $\# cl(\mathcal{O}) \approx \#\mathcal{H} \approx \sqrt{p}$.
2. Elements in $\mathcal{E}\ell\ell_p(\mathcal{O})$ can be efficiently represented by their Montgomery coefficient $A \in \mathbb{F}_p$. Given $A \in \mathbb{F}_p$, one can efficiently test whether $E_A \in \mathcal{E}\ell\ell_p(\mathcal{O})$ using [15, Algorithm 1].
3. The distinguished element is $\tilde{x} = E_0$.
4. The expressions $[\mathfrak{l}_i] \star E$ and $[\mathfrak{l}_i]^{-1} \star E$ may be efficiently evaluated for any elliptic curve $E \in \mathcal{E}\ell\ell_p(\mathcal{O})$ and any $i \in \{1, \ldots, n\}$ ([15, §3]).

Elements of the group $\mathcal{H}$ are represented as vectors $\mathbf{v} \in \mathbb{Z}^n$. With this notation, the CSIDH protocol corresponds precisely to the REGA-based protocol from Figure 2. As secret keyspace SK, the original paper [15] suggests $n = d$ and $\mathrm{SK} = \{-m, \ldots, m\}^n$, where m is chosen such that $n \log(2m+1) \approx \log(\sqrt{p})$. Hence, key recovery in CSIDH corresponds to solving REGA-DLOG$_m$. We note that here the choice of $\mathbf{g} = ([\mathfrak{l}_1], \ldots, [\mathfrak{l}_n])$ guarantees that sampling from this keyspace heuristically corresponds to a close to uniform sampling in the group $\mathcal{H}$.

For higher security parameters (e.g., a prime field of at least 2048 bits), follow-up papers [16,17] suggest to sample the vectors from smaller sets. For instance, it is suggested to use $n < d$ and sample vectors from

$$\mathrm{SK}_1 = \{-1, 0, 1\}^n, \quad \mathrm{SK}_2 = \{0, 1, 2\}^n, \quad \text{or} \quad \mathrm{SK}_3 = \{-2, 0, 2\}^n.$$

As a consequence, the public key set $\{\mathbf{v} \star \tilde{x} \colon \mathbf{v} \in \mathrm{SK}_j\}$ is only a subset of $\mathcal{E}\ell\ell_p(\mathcal{O})$ for $j \coloneqq 1, 2, 3$. Further, notice that $\# cl(\mathcal{O})$ and in particular $\mathcal{G}$ are odd, hence Lemma 2.1 implies that the corresponding REGA-DLOG problems are equivalent for the keyspaces SK_1 and SK_2, and as least as hard as for SK_3.

3 Adapting Techniques to the REGA-DLOG$_m$ Setting

Let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ be an instance of the REGA-DLOG$_m$ problem. Using the abstract framework of cryptographic group actions, we present different (classical) algorithms to solve this problem. These algorithms are well-known in the isogeny-based setting and have been used in the cryptanalysis of CSIDH.

In the following, let $N = \#\mathcal{H}$ (a possibly unknown) integer and $N_m = (2m+1)^n$. In the most recent proposals for CSIDH, we are in the situation where the secret keyspace is much smaller than the group, i.e., $N_m \ll N$. In this case the best known attacks are a meet-in-the-middle (Section 3.2), and parallel collision search (Section 3.3) approach. For completeness, we also mention that there exists a memory-less Pollard-style algorithm (Section 3.1) with running time in $\mathcal{O}(\sqrt{N})$ which is preferable if $N_m \approx N$.

3.1 Pollard-style random walks: A Galbraith-Hess-Smart adaptation

There exists a random walk approach to find a solution $\mathbf{v} \in \mathbb{Z}^n$ (of possibly large norm) in time $\mathcal{O}(\sqrt{N})$ using only a polynomial amount of memory.

The random walks will be defined by two deterministic functions

$$f : \mathcal{X} \to \{1, \ldots, n\}, \quad \sigma : \mathcal{X} \to \{-1, +1\}.$$

In the first stage of the algorithm, we set $x_0 = x$ and $\mathbf{v}_0 = \mathbf{0} \in \mathbb{Z}^n$. Then a walk of length $T \approx \sqrt{N}$ is iteratively computed as

$$x_{i+1} = g_{f(x_i)}^{\sigma(x_i)} \star x_i, \quad \mathbf{v}_{i+1} = \mathbf{v}_i + \sigma(x_i)\mathbf{e}_{f(x_i)},$$

where $\mathbf{e}_i$ is the i-th canonical vector. The pair (x_T, v_T) is stored.

In the second stage, we set $y_0 = y$ and $\mathbf{w}_0 = \mathbf{0} \in \mathbb{Z}^n$. Then one computes

$$y_{i+1} = g_{f(y_i)}^{\sigma(y_i)} \star y_i, \quad \mathbf{w}_{i+1} = \mathbf{w}_i + \sigma(y_i)\mathbf{e}_{f(y_i)}$$

until $y_S = x_T$ for some S. Then $(\mathbf{v}_T - \mathbf{w}_S) \star x = y$. Note that most likely $\mathbf{v}_T - \mathbf{w}_S \notin \{-m, \ldots, m\}^n$, so subject to our definitions it is not a solution to REGA-DLOG. For the solution to be useful, one additionally needs a reduction algorithm red which on input $\mathbf{v} \in \mathbb{Z}^n$ computes an element $red(\mathbf{v})$ of small norm so that $red(\mathbf{v}) \star x$ can be evaluated efficiently. In isogeny based group action settings such reduction methods are available. And the corresponding Pollard-style algorithm was first described by Galbraith, Hess, and Smart [25, Section 3]. Note that for the runtime analysis it is necessary that sampling vectors of small norm in $\mathbb{Z}^n$ corresponds to (close to) uniform sampling of group elements in $\mathcal{H}$ as is the case for CSIDH.

3.2 Meet-in-the-Middle (MitM)

The best known attack on REGA-DLOG$_m$ is a meet-in-the-middle-attack with time and memory complexity in $\mathcal{O}(\sqrt{N_m})$. To describe the idea, we introduce the two sets

$$S_{m,0} := \{-m, \ldots, m\}^{\frac{n}{2}} \times \{0\}^{\frac{n}{2}}, \quad S_{m,1} := \{0\}^{\frac{n}{2}} \times \{-m, \ldots, m\}^{\frac{n}{2}}.$$

These are disjoint subsets of $S_m = \{-m, \ldots, m\}^n$ of size $\sqrt{N_m}$ each. Moreover, any element $\mathbf{v} \in S_m$ has a unique representation as $\mathbf{v}_0 + \mathbf{v_1}$ with $\mathbf{v}_0 \in S_{m,0}$ and

$\mathbf{v}_1 \in S_{m,1}$. So given two set elements $x, y \in \mathcal{X}$, the problem of finding $\mathbf{v} \in S_m$ with $y = \mathbf{v} \star x$ reduces to finding vectors $\mathbf{v}_0 \in S_{m,0}$ and $\mathbf{v}_1 \in S_{m,1}$ with

$$\mathbf{v}_0 \star x = (-\mathbf{v}_1) \star y. \tag{1}$$

The time T and memory complexity M of this procedure are linear in the size of the subsets $|S_{m,0}| = |S_{m,1}|$, and therefore gives $T = M = \mathcal{O}(\sqrt{N_m})$. Concretely, for $\mathbf{v}$ being chosen from a ternary alphabet we have $T = M = \mathcal{O}(3^{\frac{n}{2}})$.

In practical applications the memory requirements of the MitM approach usually render it ineffective and require to resort to time-memory trade-offs. The naive trade-off for the MitM algorithm given W units of memory processes the subset $S_{m,0}$ in batches of size W. For each batch it iterates through all candidates $\mathbf{x}_1 \in S_{m,1}$ for $\mathbf{v}_1$ and checks for a match in the current batch. If no match is found it continues with the next batch. Straightforward analysis shows that this reduces the memory to $\tilde{\mathcal{O}}(W)$, while increasing the time complexity to $\tilde{\mathcal{O}}(N_m/W)$.

However, in the limited memory setting the Parallel Collision Search (PCS) technique by van Oorschot and Wiener is known to offer a better trade-off behavior.

3.3 Parallel Collision Search (PCS)

PCS is a technique to accelerate the search for multiple collisions between two functions f_0 and f_1 by the use of memory. A single collision between functions f_0, f_1 with domain D can be found in time $\tilde{\mathcal{O}}\left(\sqrt{|D|}\right)$ using a polynomial amount of memory by standard techniques. The PCS algorithm now allows to find W collisions in time $\tilde{\mathcal{O}}\left(\sqrt{|D| \cdot W}\right)$ using $\tilde{\mathcal{O}}(W)$ memory. This yields a $\tilde{\mathcal{O}}\left(\sqrt{W}\right)$ speedup over naive repetition of the memory-less procedure. We formalize this in the following lemma.

Lemma 3.1 (Parallel Collision Search). *Let $f_i \colon D_i \to D$ with $|D_i| = |D|$, $i = 0, 1$ be two random functions that can be evaluated in time polynomial in $\log D$. Then there is an algorithm that returns W collisions between f_0 and f_1 in time $T = \tilde{\mathcal{O}}\left(\sqrt{|D| \cdot W}\right)$ using $M = \tilde{\mathcal{O}}(W)$ memory.*

Here we do not want to dive into the details on how the technique achieves the acceleration, for those details the reader is referred to [46]. Instead we want to focus on its application to the $\mathsf{REGA\text{-}DLOG}_m$ or more specifically to the CSIDH case. Therefore we first reformulate the search for $\mathbf{v}_0$ and $\mathbf{v}_1$ as a collision search procedure. Let $S_m^{n/2} := \{-m, \ldots, m\}^{\frac{n}{2}}$ and $\mathsf{H} : \{0,1\}^* \to S_m^{n/2}$ be a hash function. Further, define the functions $f_i \colon S_{m,i} \to S_m^{n/2}$, $i = 0, 1$ as

$$f_0 \colon \mathbf{v} \mapsto \mathsf{H}(\mathbf{v} \star x) \quad \text{and} \quad f_1 \colon \mathbf{v} \mapsto \mathsf{H}\big((-\mathbf{v}) \star y\big). \tag{2}$$

Now clearly $\mathbf{v}_0$ and $\mathbf{v}_1$ form a collision between f_0 and f_1 (compare to Equation (1)). However, not every collision $(\mathbf{x}_0, \mathbf{x}_1)$ between f_0 and f_1 leads to $\mathbf{v}$, as the collision might only be a collision in the hash function H, but not necessarily implying that $\mathbf{x}_0 \star x = (-\mathbf{x}_1) \star y$. In order to find the single distinguished (often called *golden*) collision $(\mathbf{v}_0, \mathbf{v}_1)$ that leads to $\mathbf{v}$ we, therefore, have to find all collisions between f_0 and f_1. Now, instead of naively applying the standard memory-less collision search multiple times we make use of PCS to find W collisions at a time using W units of memory. We outline this procedure in pseudocode in Algorithm 1.

Algorithm 1: PCS-Tradeoff to solve $\mathsf{REGA\text{-}DLOG}_m$

Input : Functions $f_i \colon D_i \to D$, $i = 0, 1$ with $|D_i| = |D|$, W units of memory, instance $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ of the $\mathsf{REGA\text{-}DLOG}_m$

Output : solution $\mathbf{v}$ to the $\mathsf{REGA\text{-}DLOG}_m$ instance (x, y) satisfying $y = \mathbf{v} \star x$

repeat
  find W collisions $(\mathbf{w}_i, \mathbf{z}_i)$ between f_0, f_1 using PCS
until $\exists j \colon y = (\mathbf{w}_j + \mathbf{z}_j) \star x$
return $\mathbf{w}_j + \mathbf{z}_j$

Analysis. Let us briefly analyze the correctness of the procedure. For the functions f_0, f_1 defined in Equation (2), we have already shown that the pair of inputs $(\mathbf{v}_0, \mathbf{v}_1)$, with $\mathbf{v} = \mathbf{v}_0 + \mathbf{v}_1$ forms a collision. Therefore the algorithm can succeed in recovering $\mathbf{v} = \mathbf{v}_0 + \mathbf{v}_1$ by finding random collisions between those functions.

Next let us analyze the running time. As already observed, we need to recover all collisions between the functions to guarantee to find the distinguished collision that leads to $\mathbf{v}$. Further, we expect a total amount of $C = |D| = \sqrt{N_m}$ collisions between f_0 and f_1. Therefore after $\mathrm{poly}(n) \cdot \sqrt{N_m}/W$ applications of the PCS technique, each yielding W collisions, we gathered a total of $\mathrm{poly}(n) \cdot \sqrt{N_m}$ collisions. Under a standard assumption that treats those collisions as randomly sampled from the set of all collisions, we found each collision between f_0 and f_1 with high probability using a standard coupon collectors argument. This implies especially that we found the distinguished collision $(\mathbf{v}_0, \mathbf{v}_1)$ and the algorithm terminates. Each of the $\tilde{\mathcal{O}}\left(\sqrt{N_m}/W\right)$ comes at a cost of $\tilde{\mathcal{O}}\left(\sqrt{\sqrt{N_m} \cdot W}\right)$ (compare to Lemma 3.1), yielding a running time of

$$T_{\mathrm{PCS}} = \tilde{\mathcal{O}}\left(\frac{(N_m)^{\frac{3}{4}}}{\sqrt{W}}\right),$$

while the memory complexity is given as $M = \tilde{\mathcal{O}}\left(W\right)$.

4 A New Time-Memory Trade-Off using Representations

In the following we make use of the representation technique to improve the time-memory trade-off behavior of the PCS technique in the REGA setting. Therefore we first re-define the used functions to use larger domains. At first sight, this comes at the downside of increasing the cost for the collision search procedure. However, by carefully choosing the new domains we guarantee that there are several collisions $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, \ldots, N$ that allow to recover the secret $\mathbf{v}$. In turn it is not necessary to compute all existing collisions but only a $1/N$-fraction to find one of these distinguished collisions and recover $\mathbf{v}$, which overall results in a runtime advantage. Motivated by recent proposals to use ternary key spaces and for didactic reasons we first concentrate on the case of $m = 1$. Moreover, in Sections 4.1 to 4.3, we assume that the solution to $\mathsf{REGA\text{-}DLOG}_1$ has the same number of (-1)-, 0-, and 1-entries. Generalizations to the case of arbitrary weight distribution and arbitrary m are given in Section 4.4 and Section 4.5, respectively.

4.1 A First Representation-based Approach

We start with a (slightly) sub-optimal variant of our algorithm for didactic reasons. In the following sections we then subsequently refine this initial algorithm.

Let the set of ternary vectors of length n with exactly αn ± 1 entries each be defined as

$$\mathcal{T}^n(\alpha) := \{\mathbf{x} \in \{-1, 0, 1\}^n \mid \mathbf{x} \text{ contains exactly } \alpha n\ (+1) \text{ and } \alpha n\ (-1) \text{ entries}\}.$$

Now we start by redefining the functions over different domains as

$$f_0, f_1 \colon \mathcal{T}^n(\alpha) \to \mathcal{T}^n(\alpha). \tag{3}$$

Apart from this the functions remain as specified in Equation (2) , where the hash function is now defined on $\mathsf{H}\colon \{0,1\}^* \to \mathcal{T}^n(\alpha)$ and $\alpha \in [\![0, 1]\!]$ is an optimization parameter.

Our algorithm now again searches for collisions between f_0, f_1 via the PCS strategy, until a collision $(\mathbf{x}_0, \mathbf{x}_1)$ with $\mathbf{x}_0 + \mathbf{x}_1 = \mathbf{v}$ is found. A pseudocode description is obtained by using the re-defined functions together with $m = 1$ as input for Algorithm 1.

Analysis. Recall that a collision $(\mathbf{x}_0, \mathbf{x}_1)$ in f is either caused by a collision in the hash function, i.e., $\mathbf{x}_0 \star x \neq (-\mathbf{x}_1) \star y$, but $\mathsf{H}(\mathbf{x}_0 \star x) = \mathsf{H}\big((-\mathbf{x}_1) \star y\big)$ or we have

$$\mathbf{x}_0 \star x = (-\mathbf{x}_1) \star y \quad \Leftrightarrow \quad (\mathbf{x}_0 + \mathbf{x}_1) \star x = y,$$

In the latter case we call the collision *real* and conclude that $\mathbf{x}_0 + \mathbf{x}_1 = \mathbf{v}$, since $\mathbf{v}$ is sufficiently unique. This implies that any *real* collision leads to recovering $\mathbf{v}$.

Next, let us analyze the amount of real collisions $(\mathbf{x}_0, \mathbf{x}_1)$. For this it suffices to analyze the amount of $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{T}^n(\alpha)$ which satisfy $\mathbf{x}_0 + \mathbf{x}_1 = \mathbf{v}$. Those pairs $(\mathbf{x}_0, \mathbf{x}_1)$ are usually called *representations* of $\mathbf{v}$. Note that the amount of these representations of $\mathbf{v} \in \mathcal{T}^n(1/3)$ is

$$R = \binom{n/3}{n/6}^2 \binom{n/3}{\varepsilon, \varepsilon, n/3 - 2\varepsilon},$$

where $\varepsilon = (\alpha - \frac{1}{6})n$. Here the binomial coefficient counts the possibilities how $\frac{n}{6}$ of the 1 (resp. -1) entries of $\mathbf{v}$ can be contributed from $\mathbf{x}_0$, while the remaining $\frac{n}{6}$ 1 (resp. -1) entries have to be present in $\mathbf{x}_1$. The multinomial coefficient then counts the possibilities how the remaining 1s and -1s can cancel out to represent the 0s in $\mathbf{v}$. Since our choice of α will ensure $R \geq 1$ the algorithm can succeed in recovering $\mathbf{v}$ by sampling random collisions between f_0 and f_1

Let us now analyze the time complexity. We expect that after computing $\frac{C}{R}$ random collisions we encounter one that forms a representation of $\mathbf{v}$, where C is the total amount of existing collisions. Again we expect a total number of $C = |\mathcal{T}^n(\alpha)|$ collisions. Further, under the standard assumption that the functions still behave like random functions with respect to collision search, a single collision can be found in time

$$T_1 := \tilde{\mathcal{O}}\left(\sqrt{|\mathcal{T}^n(\alpha)|}\right) = \tilde{\mathcal{O}}\left(\binom{n}{\alpha n, \alpha n, (1-2\alpha)n}^{\frac{1}{2}}\right)$$

and using Lemma 3.1 we can find W collisions in time $T_W = \sqrt{W} \cdot T_1$ using $M = \tilde{\mathcal{O}}(W)$ memory. Computing the required $\frac{C}{R}$ collisions using $M = \tilde{\mathcal{O}}(W)$ memory therefore takes expected time

$$T = \tilde{\mathcal{O}}\left(\frac{C}{R \cdot W} \cdot T_W\right) = \tilde{\mathcal{O}}\left(\frac{|\mathcal{T}^n(\alpha)|^{\frac{3}{2}}}{R \cdot \sqrt{W}}\right),$$

as long as $\frac{C}{R} \geq W$.

To obtain a running time of the form $T = \tilde{\mathcal{O}}\left(3^{c(\alpha)n}\right)$ we approximate the binomial and multinomial coefficients in T using the well known approximation

$$\binom{n}{k} = \tilde{\Theta}\left(2^{nH(k/n)}\right), \tag{4}$$

where $H(x) := -x\log_2(x) - (1-x)\log_2(1-x)$ denotes the binary entropy function. We then perform a numerical optimization using the *python* library *scipy* to find the optimal α for a given amount of memory $W = 3^{\omega n}$, $\omega \in [\![0, 0.5]\!]$. We apply this strategy for all our representation based algorithms and provide our optimization code in the supplementary material. The way we access the numerical optimization framework is inspired by the code of Bonnetain, Bricout, Schrottenloher ans Shen [9].

We illustrate the obtained runtime exponent as a function of the available memory in Figure 3 and give as comparison the standard PCS exponent and the naive MitM trade-off. For an available memory that is only polynomial in n, i.e., $\omega = 0$ we improve the running time from $\tilde{\mathcal{O}}\left(3^{0.75n}\right)$ to $3^{0.675n}$. In turn this leads to an improved trade-off with time complexity $T = 3^{0.675n}/M^{0.5}$ for any available memory $M \leq 3^{0.22n}$. From there on the optimal choice of α does not fulfill the constraint $\frac{C}{R} \geq W$. However, by slightly adapting the choice of α it is possible to enforce $\frac{C}{R} \geq W$ up to $W < 3^{0.265n}$. From there on more memory does not translate into a runtime advantage, as indicated by the horizontal dotted line.

4.2 Interpolation using Partial Representations

In order to interpolate between the standard PCS technique and our representation based method from the previous section we adapt in the following the concept of

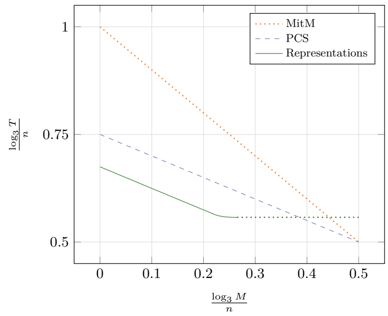


Fig. 3: Complexity of PCS, MitM and the representation-based trade-off

partial representations introduced independently in [13, 23] to our setting. In turn this allows us to achieve runtime improvements for $W \geq 3^{0.265n}$.

So far both methods – standard PCS as well as our representation approach – split the secret $\mathbf{v}$ in the sum of two vectors $\mathbf{x}_0$ and $\mathbf{x}_1$. For the standard PCS technique the vectors $\mathbf{x}_0$ and $\mathbf{x}_1$ have disjoint support, while for the representation method the support overlaps. Partial representations now combine both cases by introducing an additional optimization parameter $\delta \in [\![0, 1]\!]$ that defines how big the fraction of overlapping support of both vectors is. More precisely the secret $\mathbf{v} \in \{-1, 0, 1\}^n$ is split as

$$\mathbf{v} = \underbrace{(\mathbf{y}_0, \mathbf{0}, \mathbf{z}_0)}_{\mathbf{x}_0} + \underbrace{(\mathbf{0}, \mathbf{y}_1, \mathbf{z}_1)}_{\mathbf{x}_1} = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{z}_0 + \mathbf{z}_1)$$

with $\mathbf{y}_0, \mathbf{y}_1 \in \{-1, 0, 1\}^{\frac{(1-\delta)n}{2}}$ and $\mathbf{z}_0, \mathbf{z}_1 \in \mathcal{T}^{\delta n}(\alpha)$, where α is again an optimization parameter. That means the vectors $\mathbf{x}_0$ and $\mathbf{x}_1$ have disjoint support on the first $(1-\delta)n$ coordinates, while on the last δn their support overlaps.

Let us now re-define the functions f_0, f_1 according to partial representations. Therefore we use the following sets as domains

$$\begin{aligned} D_0 &:= \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \{0\}^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\delta n}(\alpha) \quad \text{and} \\ D_1 &:= \{0\}^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha), \end{aligned} \tag{5}$$

and define the common image space as $D := \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha)$. Leading to the functions

$$f_i \colon D_i \to D, \quad i = 0, 1. \tag{6}$$

Again the concrete definition of the functions remains as given in Equation (2), where we now require a hash function $\mathsf{H} \colon \{0,1\}^* \to D$.

In the following we use Algorithm 1 with our adapted functions from Equation (6) and $m = 1$.

Analysis. The correctness follows from the analysis of the previous section and the fact that our choice of parameters will ensure that there is at least one *real* collision, i.e. a representation of the solution or following the notation of the previous section $R \geq 1$.

Let us, hence, start by analyzing the, now changed, amount of representations R. Note, that on the first $(1-\delta)n$ coordinates, where elements from D_0 and D_1 have disjoint support, we have only a single possible decomposition of any element in $\mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3)$. Therefore we assume that the solution $\mathbf{v}$ lies in

$$\mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(1/3),$$

meaning the 1 and -1 entries distribute according to their expectation proportionally onto the three segments of length $\frac{(1-\delta)n}{2}$, $\frac{(1-\delta)n}{2}$ and δn. However, in the following we show that ensuring such a distribution of the coordinates causes at most a polynomial overhead.

Note that the probability over the random choice of $\mathbf{v} \in \mathcal{T}^n(1/3)$ for $\mathbf{v}$ having a proportional coordinate distribution over the three segments is

$$\frac{\binom{\frac{(1-\delta)n}{2}}{\frac{(1-\delta)n}{6},\frac{(1-\delta)n}{6},\frac{(1-\delta)n}{6}}^2 \binom{\delta n}{\delta n/3,\delta n/3,\delta n/3}}{\binom{n}{n/3,n/3,n/3}} = \frac{1}{\mathrm{poly}(n)},$$

which follows from approximating the binomial coefficients via Equation (4). Note that by randomly permuting the order of the generators of $\mathcal{G}$ we can obtain independent uniform distributions of the 1 and -1 entries on $\mathbf{v}$, each having a probability of $\frac{1}{\mathrm{poly}(n)}$ to distribute the coordinates as required. Therefore we expect $\mathrm{poly}(n)$ repetitions of the algorithm with random permutations of the generators to ensure this distribution in at least one of the executions.

On the last δn coordinates of elements from $\mathbf{x}_i \in D_i$, we obtain multiple representations of $\mathbf{v} = \mathbf{x}_0 + \mathbf{x}_1$ as sum of two elements. Similar to the analysis in the previous section the amount of such representations of one element from $\mathcal{T}^{\delta n}(1/3)$ as the sum of two elements from $\mathcal{T}^{\delta n}(\alpha)$ is given as

$$R = \binom{\delta n/3}{\delta n/6}^2 \binom{\delta n/3}{\varepsilon, \varepsilon, \delta n/3 - 2\varepsilon},$$

where $\varepsilon = (\alpha - \frac{1}{6})\delta n$.

The complexity analysis follows along the lines of the analysis in Section 4.1 with the difference that a single collision search now comes at the cost of

$$T_1 = \tilde{\mathcal{O}}\left(\sqrt{|D|}\right) = \tilde{\mathcal{O}}\left(\left(\binom{\frac{(1-\delta)n}{2}}{\frac{(1-\delta)n}{6}, \frac{(1-\delta)n}{6}, \frac{(1-\delta)n}{6}}\binom{\delta n}{\alpha\delta n, \alpha\delta n, (1-2\alpha)\delta n}\right)^{\frac{1}{2}}\right).$$

The final complexity is then analogously given as $T = \tilde{\mathcal{O}}\left(\frac{C}{R \cdot W} \cdot T_W\right) = \tilde{\mathcal{O}}\left(\frac{|D|^{\frac{3}{2}}}{R \cdot \sqrt{W}}\right)$, as long as $\frac{C}{R} \geq W$, where still $T_W = \sqrt{W} \cdot T_1$. The memory complexity is still dominated by the application of the PCS with $M = \tilde{\mathcal{O}}(W)$.

In Figure 4 we illustrate the asymptotic running time exponent obtained by numerical optimization of α, δ as a function of the memory. We observe that partial representations enable a smooth interpolation between the representation method from Section 4.1 and the PCS technique (Section 3.3), while providing improvements over both methods for any $3^{0.25n} \leq M < 3^{0.4n}$.

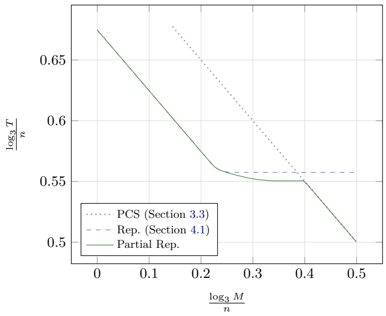


Fig. 4: Complexity of PCS, the representation trade-off, and partial representations.

4.3 Increasing the Amount of Representations

In the following we again slightly adapt the domains of the functions to include elements with coordinates in $\{-2, \dots, 2\}$ rather than $\{-1, 0, 1\}$. While, as before, this increases the size of the domains and, hence, the time for the collision search, it also yields an increased amount of representations, leading to a runtime improvement.

Note that in terms of representations any -1 can additionally be represented as $-2 + 1$ (resp. $1 + (-2)$), accordingly any 1 as $-1 + 2$ (resp. $2 + (-1)$) and any 0 as $-2 + 2$ (resp. $2 + (-2)$). Let the set of vectors with αn ± 1 entries each and β ± 2 entries each be denoted as

$$\mathcal{T}^n(\alpha, \beta) := \{\mathbf{x} \in \{-2, \dots, 2\}^n \mid |\mathbf{x}|_1 = |\mathbf{x}|_{-1} = \alpha n \wedge |\mathbf{x}|_2 = |\mathbf{x}|_{-2} = \beta n\}, \quad (7)$$

where $|\mathbf{x}|_i = |\{j \in \{1, \dots, n\} \mid x_j = i\}|$.

We now adapt the domains of the previous section, i.e., we still use partial representations, but now also including -2 and 2 entries. Therefore let the new domains be

$$\begin{aligned} \tilde{D}_0 &:= \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \{0\}^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\delta n}(\alpha, \beta) \quad \text{and} \\ \tilde{D}_1 &:= \{0\}^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha, \beta), \end{aligned} \quad (8)$$

and re-define the common image space as $\tilde{D} := \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha, \beta)$, where $\delta \in [\![0, 1]\!]$ is subject to optimization and α, β are determined later. The functions are then defined over

$$f_i \colon \tilde{D}_i \to \tilde{D}, \quad i = 0, 1, \quad (9)$$

with their precise mapping still as given in Equation (2), requiring now a hash function $\mathsf{H} \colon \{0, 1\}^* \to \tilde{D}$.

We now analyze the complexity of Algorithm 1 with input $m = 1$ and functions as specified in Equation (9).

Analysis. The analysis again follows along the lines of the analysis of the previous section with the main difference lying in the amount of representations R and the now increased domain size $|\tilde{D}|$. Let us start by examining the amount of representations R. We, again, assume $\mathbf{v}$ to be from $\mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(1/3)$, which we ensure by random permutations of the generators leading to polynomial overhead. In turn there exists exactly one decomposition of $\mathbf{v}$ in the sum of two elements from $\tilde{D}_1$ and $\tilde{D}_2$ with respect to the the first $(1-\delta)n$ coordinates. Multiple representations only exist for the last δn coordinates. We have the following possibilities to represent a $-1, 0$ and 1 entry in $\mathbf{v} = \mathbf{x}_0 + \mathbf{x}_1$

$$\begin{aligned} 0: &\quad \underbrace{0+0}_{z_0}, \quad \underbrace{1-1}_{z_1}, \quad \underbrace{-1+1}_{z_1}, \quad \underbrace{2-2}_{z_2}, \quad \underbrace{-2+2}_{z_2}, \\ 1: &\quad \underbrace{1+0}_{\frac{\delta n}{6}-o}, \quad \underbrace{0+1}_{\frac{\delta n}{6}-o}, \quad \underbrace{2-1}_{o}, \quad \underbrace{-1+2}_{o}, \\ -1: &\quad \underbrace{-1+0}_{\frac{\delta n}{6}-o}, \quad \underbrace{0-1}_{\frac{\delta n}{6}-o}, \quad \underbrace{-2+1}_{o}, \quad \underbrace{1-2}_{o}. \end{aligned} \tag{10}$$

Here the variable below each of the representations specifies how often we want to use the corresponding representation to represent a corresponding coordinate of $\mathbf{v}$. For example we expect z_0 many of the 0 entries in $\mathbf{v}$ to be represented in the sum $\mathbf{v} = \mathbf{x}_0 + \mathbf{x}_1$ as $0+0$, z_1 as $1-1$, z_1 as $-1+1$, z_2 as $2-2$ and z_2 as $-2+2$. It follows that we need to ensure

$$z_0 + 2z_1 + 2z_2 = \frac{\delta n}{3} \quad \Leftrightarrow \quad z_0 = \frac{\delta n}{3} - 2z_1 - 2z_2,$$

as in total we need to represent $\frac{\delta n}{3}$ zeros of $\mathbf{v}$. Note that the total amount of 1 (resp. -1) entries sums to $\frac{\delta n}{6} - o + \frac{\delta n}{6} - o + o + o = \frac{\delta n}{3}$ as required (since there are that many -1 and 1 entries in the last δn coordinates of $\mathbf{v}$). Note that the parameters z_1, z_2 and o are optimization parameters of the algorithm. Given the proportions specified in Equation (10), we can directly derive the amount of representations as

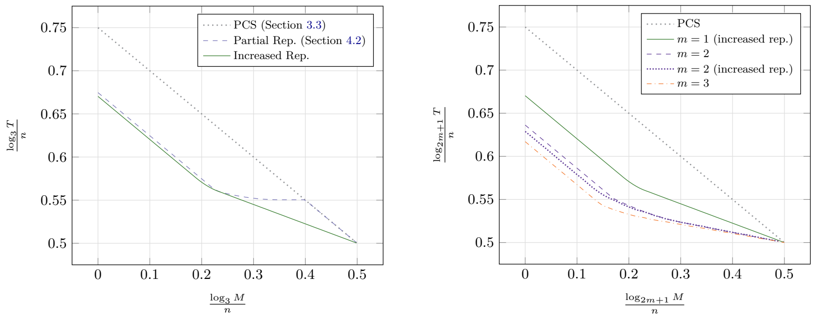
$$R = \binom{\frac{\delta n}{3}}{z_0, z_1, z_1, z_2, z_2} \binom{\frac{\delta n}{3}}{\frac{\delta n}{6} - o, \frac{\delta n}{6} - o, o, o}^2,$$

where the first term counts the possibilities to represent 0s and the second the representations of ± 1 entries A simple counting of the representations from Equation (10) including a ± 1 or ± 2 yields that the initial domains $\tilde{D}_1, \tilde{D}_2$ need to satisfy

$$\alpha = \frac{1}{6} + \frac{z_1}{\delta} \quad \text{and} \quad \beta = \frac{z_2 + o}{\delta}.$$

From here the analysis is identical to the one from Section 4.2, leading to a time complexity of $T = \tilde{\mathcal{O}}\left(\frac{C}{R \cdot W} \cdot T_W\right) = \tilde{\mathcal{O}}\left(\frac{|\tilde{D}|^{\frac{3}{2}}}{R \cdot \sqrt{W}}\right)$, as long as $\frac{|\tilde{D}|}{R} \geq W$, and memory complexity $M = \tilde{\mathcal{O}}(W)$.

In Figure 5a we illustrate the obtained runtime exponent. We observe that the increased amount of representations allows to naturally connect the trade-off to the $(0.5, 0.5)$ endpoint of MitM.



(a) Complexity of different approaches. (b) Complexity for different choices of m.

Fig. 5: On the left: Comparison of different representation based methods.
On the right: Comparison of representation based methods for different m.

4.4 Enforcing an Equal Weight Distribution

Recall that in previous sections we always assumed for simplicity that we attack ternary vectors with equally balanced (up to rounding) number of (-1)-, 0-, and 1-entries. We now show that for almost all ternary vectors we can enforce such an equal weight distribution by increasing the dimension of the $\mathsf{REGA\text{-}DLOG}_1$-problem from n to $n + \mathcal{O}(\sqrt{n})$. Our argument extends to all $\mathsf{REGA\text{-}DLOG}_m$ with constant m.

Notice that our algorithms are of complexity $T = 3^{cn}$ for some constant c, and thus fully exponential in the dimension n. Therefore, our dimension increase only leads to a subexponential overhead $3^{\mathcal{O}(\sqrt{n})}$, i.e., we achieve asymptotic run time

$$3^{c(n+\mathcal{O}(\sqrt{n}))} = T \cdot 3^{\mathcal{O}(\sqrt{n})} = 3^{cn(1+o(1))}.$$

Idea of Balancing. Let $\mathbf{v}$ be a random ternary, and denote by n_i, $i \in \{-1, 0, 1\}$ its numbers of i-entries. Since $n_i \leq n$, we can guess all n_i in polynomial time $\mathcal{O}(n^2)$. We show that with high probability all n_i are bounded by $n/3 \pm \mathcal{O}(\sqrt{n})$. Without loss of generality, let n_{-1} be the maximal value. We then add $n_{-1} - n_0$ coordinates for 0-entries, and $n_{-1} - n_1$ coordinates for 1-entries. These are in total $\ell = \mathcal{O}(\sqrt{n})$ coordinates.

To this end, let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ be an instance of the $\mathsf{REGA\text{-}DLOG}_1$ problem with $\mathbf{g} = (g_1, \dots, g_n)$ and a ternary solution $\mathbf{v}$ satisfying $\mathbf{v} \star x = y$.

Let id be the neutral element in $\mathcal{G}$, and let $\mathbf{g}' = (g_1, \dots, g_n, id, \dots, id)$ be the set of generators enhanced by ℓ times id. Then any $\mathbf{u} = (\mathbf{v}, \mathbf{w})$ with $\mathbf{w} \in \mathbb{Z}^\ell$ is a solution for the dimension-increased instance with $\mathbf{g}'$ iff $\mathbf{v}$ is a solution for the original instance with $\mathbf{g}$. Especially, we obtain a solution for our n-dimensional instance by solving the $n+\ell = n+\mathcal{O}(\sqrt{n})$-dimension instance, and cutting off the last ℓ coordinates.

Chernoff argument. It remains to show that all n_i differ from $n/3$ by at most $\mathcal{O}(\sqrt{n})$. Since $\mathbf{v}$ is a random ternary vector, all n_i are binomially distributed random variables with $\mathbb{E}[n_i] = n/3$. We use the Chernoff bound

$$\Pr\left[|n_i - \mathbb{E}[n_i]| \geq \delta\mathbb{E}[n_i]\right] \leq 2e^{-\mathbb{E}[n_i]\delta^2/3} \text{ for } 0 < \delta < 1.$$

Define $\delta = \frac{3c}{\sqrt{n}}$ for some constant c. Then $\Pr[|n_i - n/3| \geq c\sqrt{n}] \leq 2e^{-c^2}$. Thus, for sufficiently large c, almost all ternary vectors reach their expected value $n/3$ up to an $\mathcal{O}(\sqrt{n})$ error term.

4.5 The Case of Arbitrary m

Intuitively it is clear that a similar approach as in Sections 4.1 to 4.3 can be taken to solve the $\mathsf{REGA\text{-}DLOG}_m$ for an arbitrary integer m. One simply defines the functions over appropriate domains, which allow for multiple representations of $\mathbf{v} \in \{-m, \ldots, m\}^n$ and then applies Algorithm 1 with those functions and the respective choice of m. The main obstacle with this approach lies in the computation of the representations, which already for $m = 1$ became quite technical if applying the technique to its full extend (compare to Section 4.3).

However, for completeness we specify in the following the running time for the case of general m in dependence on the domain size and the amount of representations. The result is an immediate implication of our previous analysis.

Let the functions be specified as $f_i \colon S_i \to S, i = 0, 1$, with the mapping as defined in Equation (2), where $\mathsf{H}\colon \{0,1\}^* \to S$ and $|S_0| = |S_1| = |S|$. Furthermore, let every element $\mathbf{v} \in \{-m, \ldots, m\}^n$ have R representations as the sum of elements from S_0, S_1, i.e., there are R different pairs $(\mathbf{x}_0, \mathbf{x}_1) \in S_0 \times S_1$ with $\mathbf{v} = \mathbf{x}_0 + \mathbf{x}_1$.

Then v can be found via Algorithm 1 with functions f_0, f_1 in time $T = \tilde{\mathcal{O}}\left(\frac{|S|^{\frac{3}{2}}}{R\cdot\sqrt{W}}\right)$, as long as $\frac{|S|}{R} \geq W$, using memory $M = \tilde{\mathcal{O}}\left(W\right)$.

We additionally computed the running time of our technique for $m \in \{2, 3\}$ for an appropriate choice of function domains. We illustrate the corresponding runtime exponents in Figure 5b.

For obtaining the running times we used in the case of $m = 2$ addends $\mathbf{x}_0, \mathbf{x}_1 \in \{-2, \ldots, 2\}$ ($m = 2$) and $\mathbf{x}_0, \mathbf{x}_1 \in \{-3, \ldots, 3\}$ ($m = 2$ (increased rep.)) to represent the solution $\mathbf{v} = \mathbf{x}_0 + \mathbf{x}_1$. In the case of $m = 3$ we used only addends $\mathbf{x}_0, \mathbf{x}_1 \in \{-3, \ldots, 3\}$. For the full technical details of the analysis the reader is referred to Appendix A. It can be observed, that for increasing m the runtime exponent in dependence on search space improves. However, since the improvement is getting smaller for growing m we conjecture that the exponent converges.

4.6 Potential Impact on Bit Security Level

In this section we approximate the maximal bit security reduction for suggested parameter sets for CSIDH by the representation method. In our comparison we assume that the standard PCS based time-memory trade-off (compare to Section 3.3) suffers the same same polynomial overhead as the representation based approach. Since this might underestimate the overhead of the representation based trade-off, the numbers should be seen as a maximal potential gain. Practical experiments will have to determine to which extend this gain can be realized in practice.

In [17] three concrete parameter instantiations for ternary-key CSIDH are given, respectively aiming at satisfying NIST security level $L1, L2$ and $L3$. For matching the security definition of category L_i the authors impose restrictions on the memory and time complexity of $M_i = 2^{w_i}$ and $T_i = w^{t_i}$ with

$$(w_1, w_2, w_3) = (80, 100, 119) \quad \text{and} \quad (t_1, t_2, t_3) = (128, 128, 192).$$

In order to match those security definitions a number of generators n_i equal to $n_1 = 139$ for L_1, $n_2 = 148$ for L_2 and $n_3 = 210$ for L_3 is proposed. The security of those parameter sets is determined via the PCS time-memory trade-off.

In the memory restriction the authors conservatively ignore polynomial factors, i.e., it holds $M_i = 3^{c_i n_i} = 2^{w_i}$, which allows to determine the asymptotic memory exponent as $c_i = \frac{w_i}{n_i \cdot \log_2 3}$. For example for $i = 1$ we obtain $c_i \approx 0.3631$, which yields an asymptotic running time of the PCS approach of $T_{\mathrm{PCS}} = 3^{0.5685n}$. In comparison our technique improves the running time to $T_{\mathrm{Rep}} = 3^{0.5316n}$, corresponding to a gain of

$$\frac{T_{\mathrm{PCS}}}{T_{\mathrm{Rep}}} = 3^{0.5685n} = 3^{0.0369n},$$

which for $n_1 = 139$ yields a reduced security level by $0.0369 \cdot n_1 \cdot \log_2 3 \approx 8.13$ bit.

A similar analysis for the cases of $i = 2$ yields $c_2 \approx 0.4263$ with $T_{\mathrm{PCS}} = 3^{0.5369n}$ and $T_{\mathrm{Rep}} = 3^{0.5174n}$ corresponding to a gain of 4.57 bit. The case of $i = 3$ yields $c_3 \approx 0.3575$ with $T_{\mathrm{PCS}} = 3^{0.5713n}$ and $T_{\mathrm{Rep}} = 3^{0.5330n}$ reducing the security level by 12.75 bit.

Acknowledgements. Sabrina Kunzweiler and Alexander May were funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

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. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 411–439. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_14

3. Alamati, N., Feo, L.D., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12492, pp. 411–439. Springer (2020). https://doi.org/10.1007/978-3-030-64834-3_14, https://doi.org/10.1007/978-3-030-64834-3_14
4. Albrecht, M.R., Bernstein, D.J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Paterson, K.G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C.J., Tomlinson, M., Wang, W.: Classic McEliece: conservative code-based cryptography (2020)
5. Banegas, G., Bernstein, D.J., Campos, F., Chou, T., Lange, T., Meyer, M., Smith, B., Sotáková, J.: CTIDH: faster constant-time CSIDH. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 351–387 (2021). https://doi.org/10.46586/tches.v2021.i4.351-387, https://doi.org/10.46586/tches.v2021.i4.351-387
6. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Krauthgamer, R. (ed.) 27th SODA. pp. 10–24. ACM-SIAM (Jan 2016). https://doi.org/10.1137/1.9781611974331.ch2
7. 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
8. 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)
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. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12106, pp. 493–522. Springer (2020). https://doi.org/10.1007/978-3-030-45724-2_17, https://doi.org/10.1007/978-3-030-45724-2_17
11. 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)
12. Both, L., May, A.: Decoding linear codes with high error rate and its impact for lpn security. In: International Conference on Post-Quantum Cryptography. pp. 25–46. Springer (2018)
13. 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
14. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH (preliminary version). IACR Cryptol. ePrint Arch. p. 975 (2022), https://eprint.iacr.org/2022/975

15. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11274, pp. 395–427. Springer (2018). https://doi.org/10.1007/978-3-030-03332-3_15
16. Cervantes-Vázquez, D., Chenu, M., Chi-Domínguez, J., Feo, L.D., Rodríguez-Henríquez, F., Smith, B.: Stronger and faster side-channel protections for CSIDH. In: Schwabe, P., Thériault, N. (eds.) Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11774, pp. 173–193. Springer (2019). https://doi.org/10.1007/978-3-030-30530-7_9
17. Chávez-Saab, J., Chi-Domínguez, J., Jaques, S., Rodríguez-Henríquez, F.: The SQALE of CSIDH: sublinear vélu quantum-resistant isogeny action with low exponents. J. Cryptogr. Eng. **12**(3), 349–368 (2022). https://doi.org/10.1007/s13389-021-00271-w, https://doi.org/10.1007/s13389-021-00271-w
18. Chi-Domínguez, J., Rodríguez-Henríquez, F.: Optimal strategies for CSIDH. Adv. Math. Commun. **16**(2), 383–411 (2022). https://doi.org/10.3934/amc.2020116, https://doi.org/10.3934/amc.2020116
19. Costello, C., Longa, P., Naehrig, M., Renes, J., Virdia, F.: Improved classical cryptanalysis of the computational supersingular isogeny problem. Cryptology ePrint Archive, Report 2019/298 (2019), https://eprint.iacr.org/2019/298
20. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006), https://eprint.iacr.org/2006/291
21. Esser, A.: Revisiting nearest-neighbor-based information set decoding. Cryptology ePrint Archive, Report 2022/1328 (2022), https://eprint.iacr.org/2022/1328
22. Esser, A., Girme, R., Mukherjee, A., Sarkar, S.: Memory-efficient attacks on small lwe keys. Cryptology ePrint Archive (2023)
23. 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
24. 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
25. Galbraith, S.D., Hess, F., Smart, N.P.: Extending the GHS Weil descent attack. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 29–44. Springer, Heidelberg (Apr / May 2002). https://doi.org/10.1007/3-540-46035-7_3
26. Glaser, T., May, A.: How to enumerate LWE keys as narrow as in kyber/dilithium. Cryptology ePrint Archive, Report 2022/1337 (2022), https://eprint.iacr.org/2022/1337
27. Hutchinson, A., LeGrow, J.T., Koziel, B., Azarderakhsh, R.: Further optimizations of CSIDH: A systematic approach to efficient strategies, permutations, and bound vectors. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 20, Part I. LNCS, vol. 12146, pp. 481–501. Springer, Heidelberg (Oct 2020). https://doi.org/10.1007/978-3-030-57808-4_24
28. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.Y. (ed.) Post-Quantum Cryptography - 4th

International Workshop, PQCrypto 2011. pp. 19–34. Springer, Heidelberg (Nov / Dec 2011). https://doi.org/10.1007/978-3-642-25405-5_2
29. Kuperberg, G.: A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. SIAM Journal on Computing **35**(1), 170–188 (2005)
30. Maino, L., Martindale, C.: An attack on SIDH with arbitrary starting curve. IACR Cryptol. ePrint Arch. p. 1026 (2022), https://eprint.iacr.org/2022/1026
31. 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
32. 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
33. May, A., Ozerov, I.: A generic algorithm for small weight discrete logarithms in composite groups. In: Joux, A., Youssef, A.M. (eds.) SAC 2014. LNCS, vol. 8781, pp. 278–289. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-319-13051-4_17
34. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 203–228. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_9
35. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. The deep space network progress report 42-44, Jet Propulsion Laboratory, California Institute of Technology (Jan/Feb 1978), https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF
36. Meyer, M., Campos, F., Reith, S.: On lions and elligators: An efficient constant-time implementation of CSIDH. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers. Lecture Notes in Computer Science, vol. 11505, pp. 307–325. Springer (2019). https://doi.org/10.1007/978-3-030-25510-7_17
37. Onuki, H., Aikawa, Y., Yamazaki, T., Takagi, T.: (Short paper) A faster constant-time algorithm of CSIDH keeping two points. In: Attrapadung, N., Yagi, T. (eds.) IWSEC 19. LNCS, vol. 11689, pp. 23–33. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26834-3_2
38. Onuki, H., Aikawa, Y., Yamazaki, T., Takagi, T.: A constant-time algorithm of CSIDH keeping two points. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **103-A**(10), 1174–1182 (2020). https://doi.org/10.1587/transfun.2019DMP0008
39. Peikert, C.: He gives c-sieves on the CSIDH. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12106, pp. 463–492. Springer (2020). https://doi.org/10.1007/978-3-030-45724-2_16, https://doi.org/10.1007/978-3-030-45724-2_16
40. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)
41. Robert, D.: Breaking SIDH in polynomial time. IACR Cryptol. ePrint Arch. p. 1038 (2022), https://eprint.iacr.org/2022/1038
42. Rostovtsev, A., Stolbunov, A.: Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145 (2006), https://eprint.iacr.org/2006/145

43. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th FOCS. pp. 124–134. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365700
44. Tani, S.: Claw finding algorithms using quantum walk. Theoretical Computer Science **410**(50), 5285–5297 (2009)
45. Van Hoof, I., Kirshanova, E., May, A.: Quantum key search for ternary LWE. In: Cheon, J.H., Tillich, J.P. (eds.) Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021. pp. 117–132. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-81293-5_7
46. 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

A The Case of Larger m

For larger choices of m we still assume that each coordinate is present $\frac{n}{2m+1}$ times in the solution. For any constant m, this is the case for a polynomial fraction of all keys, and can be ensure with subexponential overhead similar to the procedure explained in Section 4.4. Further, we always use partial representations, i.e., the domains consist, similar to Section 4.2 and Section 4.3 of three parts of length $\frac{(1-\delta)n}{2}$, $\frac{(1-\delta)n}{2}$ and δn. Here we assume that each coordinate is present proportionally to the length of the segment, e.g., that the last segment contains each coordinate exactly $\frac{\delta n}{2m+1}$ times, which again can be ensured at the cost of a polynomial overhead only.

As outlined in Section 4.5, for each choice of m we now specify the used function domains and derive the amount representations of the solution. Let us start with the case of $m = 2$.

The case of $m = 2$. We are looking for a solution $\mathbf{v} \in \{-2, \ldots, 2\}$. For our first instantiation we use the same function definitions as in Section 4.3 given in Equations (8) and (9), where we choose a different α and β, specified later. Let us again specify the possible representations of each entry (similar to Equation (10))

$$\begin{array}{rlllll}
0: & \underbrace{0+0}_{z_0}, & \underbrace{1-1}_{z_1}, & \underbrace{-1+1}_{z_1}, & \underbrace{2-2}_{z_2}, & \underbrace{-2+2}_{z_2}, \\
1: & \underbrace{1+0}_{\frac{\delta n}{10}-o}, & \underbrace{0+1}_{\frac{\delta n}{10}-o}, & \underbrace{2-1}_{o}, & \underbrace{-1+2}_{o}, & \\
-1: & \underbrace{-1+0}_{\frac{\delta n}{10}-o}, & \underbrace{0-1}_{\frac{\delta n}{10}-o}, & \underbrace{-2+1}_{o}, & \underbrace{1-2}_{o}. & \\
2: & \underbrace{2+0}_{\frac{\delta n}{10}-\frac{t}{2}}, & \underbrace{0+2}_{\frac{\delta n}{10}-\frac{t}{2}}, & \underbrace{1+1}_{t}, & & \\
-2: & \underbrace{-2+0}_{\frac{\delta n}{10}-\frac{t}{2}}, & \underbrace{0-2}_{\frac{\delta n}{10}-\frac{t}{2}}, & \underbrace{-1-1}_{t}. & &
\end{array}$$

Recall, that we have only representations on the last segment of length δn. As we expect any coordinate to be present $\delta n/5$ times, we need that the numbers below the representations in every row sum to $\delta n/5$. Therefore we have

$$z_0 + 2z_1 + 2z_2 = \delta n/5 \quad \Leftrightarrow \quad z_0 = \delta n/5 - 2z_1 - 2z_2.$$

Further by counting the respective number of ± 1 and ± 2 entries in those representations we obtain

$$\alpha = \frac{1}{10} + \frac{z_1 + t}{\delta} \quad \text{and} \quad \beta = \frac{1}{10} + \frac{z_2 - t/2 + o}{\delta},$$

while the number of representations is given as

$$R = \binom{\frac{\delta n}{5}}{z_0, z_1, z_1, z_2, z_2} \binom{\frac{\delta n}{5}}{\frac{\delta n}{10} - o, \frac{\delta n}{10} - o, o, o}^2 \binom{\frac{\delta n}{5}}{\frac{\delta n}{10} - \frac{t}{2}, \frac{\delta n}{10} - \frac{t}{2}, t}^2.$$

The values of z_1, z_2, o, t and δ are subject to numerical optimization.

Increased representations for $m = 2$. In the following we represent $\mathbf{v}$ on its last δn coordinates via the sum of two vectors $\mathbf{x}_0, \mathbf{x}_1 \in \{-3, \ldots, 3\}^{\delta n}$. Similar to including -2 and 2 entries in the case of $m = 1$ (Section 4.3), this leads to an increased amount of representations and in turn a runtime improvement.

First we naturally extend the definition $\mathcal{T}^n(\alpha, \beta)$ from Equation (7) to $\mathcal{T}^n(\alpha, \beta, \gamma)$, where in the latter case included vectors contain exactly γn entries equal to ± 3 each. Then we let the new function domains be defined as

$$\begin{aligned} S_0 &:= \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times 0^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\delta n}(\alpha, \beta, \gamma) \quad \text{and} \\ S_1 &:= 0^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha, \beta, \gamma), \end{aligned} \tag{11}$$

Accordingly we let their common image space be $S = \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha, \beta, \gamma)$.

Now we obtain additional representations of any 0, ± 1 and ± 2 entry. Let us again specify all representations and how often they appear in the addition.

$$\begin{array}{rlllllll} 0: & \underbrace{0+0}_{z_0}, & \underbrace{1-1}_{z_1}, & \underbrace{-1+1}_{z_1}, & \underbrace{2-2}_{z_2}, & \underbrace{-2+2}_{z_2}, & \underbrace{-3+3}_{z_3}, & \underbrace{-3+3}_{z_3}, \\ 1: & \underbrace{1+0}_{\frac{\delta n}{10}-o-d_1}, & \underbrace{0+1}_{\frac{\delta n}{10}-o-d_1}, & \underbrace{2-1}_{o}, & \underbrace{-1+2}_{o}, & \underbrace{3-2}_{d_1}, & \underbrace{-2+3}_{d_1}, & \\ -1: & \underbrace{-1+0}_{\frac{\delta n}{10}-o-d_1}, & \underbrace{0-1}_{\frac{\delta n}{10}-o-d_1}, & \underbrace{-2+1}_{o}, & \underbrace{1-2}_{o}, & \underbrace{-3+2}_{d_1}, & \underbrace{2-3}_{d_1}, & \\ 2: & \underbrace{2+0}_{\frac{\delta n}{10}-\frac{t}{2}-d_2}, & \underbrace{0+2}_{\frac{\delta n}{10}-\frac{t}{2}-d_2}, & \underbrace{1+1}_{t}, & \underbrace{3-1}_{d_2}, & \underbrace{-1+3}_{d_2}, & & \\ -2: & \underbrace{-2+0}_{\frac{\delta n}{10}-\frac{t}{2}-d_2}, & \underbrace{0-2}_{\frac{\delta n}{10}-\frac{t}{2}-d_2}, & \underbrace{-1-1}_{t} & \underbrace{-3+1}_{d_2}, & \underbrace{1-3}_{d_2}. & & \end{array} \tag{12}$$

Analogously to before we have

$$z_0 + 2z_1 + 2z_2 + 2z_3 = \delta n/5 \quad \Leftrightarrow \quad z_0 = \delta n/5 - 2z_1 - 2z_2 - 2z_3.$$

Further by counting we obtain

$$\alpha = \frac{1}{10} + \frac{z_1 + t - d_1 + d_2}{\delta}, \quad \beta = \frac{1}{10} + \frac{z_2 - t/2 + o - d_2 + d_1}{\delta} \quad \text{and}$$
$$\gamma = \frac{z_3 + d_1 + d_2}{\gamma}$$

while the number of representations increases to

$$R = \binom{\frac{\delta n}{5}}{z_0, z_1, z_1, z_2, z_2, z_3, z_3} \binom{\frac{\delta n}{5}}{\frac{\delta n}{10} - o - d_1, \frac{\delta n}{10} - o - d_1, o, o, d_1, d_1}^2 \cdot \binom{\frac{\delta n}{5}}{\frac{\delta n}{10} - \frac{t}{2} - d_2, \frac{\delta n}{10} - \frac{t}{2} - d_2, t, d_2, d_2}^2.$$

The values of $z_1, z_2, z_3, o, t, d_1, d_2$ and δ are subject to numerical optimization.

Finally let us consider the case of $m = 3$.

The case of $m = 3$. We now have a solution $\mathbf{v} \in \{-3, \ldots, 3\}$. We represent this solution by using the same function domains as specified in Equation (11), with an adapted choice of α, β and γ.

The possible representations stay therefore as specified in Equation (12), by replacing $\frac{\gamma n}{10}$ by $\frac{\gamma n}{14}$. Since every row has now to add up to $\frac{\gamma n}{7}$ we obtain

$$z_0 + 2z_1 + 2z_2 + 2z_3 = \delta n/7 \quad \Leftrightarrow \quad z_0 = \delta n/7 - 2z_1 - 2z_2 - 2z_3.$$

We now get additionally representations for the ± 3 entries in $\mathbf{v}$:

$$\begin{aligned} 3: &\quad \underbrace{3+0}_{\frac{\delta n}{14} - d_3}, \quad \underbrace{0+3}_{\frac{\delta n}{14} - d_3}, \quad \underbrace{2+1}_{d_3}, \quad \underbrace{1+2}_{d_3}, \\ -3: &\quad \underbrace{-3+0}_{\frac{\delta n}{14} - d_3}, \quad \underbrace{0-3}_{\frac{\delta n}{14} - d_3}, \quad \underbrace{-2-1}_{d_3} \quad \underbrace{-1-2}_{d_3}. \end{aligned}$$

This leads to the adapted choices of

$$\alpha = \frac{1}{14} + \frac{z_1 + t - d_1 + d_2}{\delta}, \quad \beta = \frac{1}{14} + \frac{z_2 - t/2 + o - d_2 + d_1}{\delta} \quad \text{and}$$
$$\gamma = \frac{1}{14} + \frac{z_3 + d_1 + d_2 - d_3}{\gamma}.$$

Eventually the amount of representations is given as

$$R = \binom{\frac{\delta n}{7}}{z_0, z_1, z_1, z_2, z_2, z_3, z_3} \binom{\frac{\delta n}{7}}{\frac{\delta n}{14} - o - d_1, \frac{\delta n}{14} - o - d_1, o, o, d_1, d_1}^2 \cdot \binom{\frac{\delta n}{7}}{\frac{\delta n}{14} - \frac{t}{2} - d_2, \frac{\delta n}{14} - \frac{t}{2} - d_2, t, d_2, d_2}^2 \binom{\frac{\delta n}{7}}{\frac{\delta n}{14} - d_3, \frac{\delta n}{14} - d_3, d3, d3}^2.$$