# dCTIDH: Fast & Deterministic CTIDH

Fabio Campos[1], Andreas Hellenbrand[2], Michael Meyer[3] and Krijn Reijnders[4]

[1] Bonn-Rhein-Sieg University of Applied Sciences, Germany
campos@sopmac.de
[2] RheinMain University of Applied Sciences Wiesbaden, Germany
andreas.hellenbrand@hs-rm.de
[3] University of Regensburg, Germany
michael@random-oracles.org
[4] Radboud University, Nijmegen, The Netherlands
krijn@q1q1.nl

**Abstract.** This paper presents dCTIDH, a CSIDH implementation that combines two recent developments into a novel state-of-the-art deterministic implementation. We combine the approach of deterministic variants of CSIDH with the batching strategy of CTIDH, which shows that the full potential of this key space has not yet been explored. This high-level adjustment in itself leads to a significant speed-up. To achieve an effective deterministic evaluation in constant time, we introduce WOMBats, a new approach to performing isogenies in batches, specifically tailored to the behavior required for deterministic CSIDH using CTIDH batching. Furthermore, we explore the two-dimensional space of optimal primes for dCTIDH, with regard to both the performance of dCTIDH in terms of finite-field operations per prime and the efficiency of finite-field operations, determined by the prime shape, in terms of cycles. This allows us to optimize both for choice of prime and scheme parameters simultaneously. Lastly, we implement and benchmark constant-time, deterministic dCTIDH. Our results show that dCTIDH not only outperforms state-of-the-art deterministic CSIDH, but even *non-deterministic* CTIDH: dCTIDH-2048 is faster than CTIDH-2048 by 17%, and is almost five times faster than dCSIDH-2048.

**Keywords:** post-quantum cryptography · isogeny-based cryptography · CSIDH

## 1 Introduction

At Asiacrypt 2018, Castryck, Lange, Martindale, Panny, and Renes [16] proposed CSIDH, a commutative isogeny-based key exchange protocol. Since its introduction, CSIDH has been one of the most interesting post-quantum cryptoschemes, mainly due to its small public keys (64 bytes in the original NIST Level I parameter set), and its non-interactive properties. In particular, CSIDH was the first practical post-quantum *non-interactive key exchange* (NIKE), which makes it a theoretical post-quantum drop-in replacement of many Diffie-Hellman type systems. Furthermore, its commutative structure allows for building more advanced protocols based on CSIDH, such as digital signatures [10], oblivious transfer [25], or threshold schemes [22].

*Author list in alphabetical order; see https://www.ams.org/profession/leaders/CultureStatement04.pdf. This work has been supported by the German Federal Ministry of Education and Research (BMBF) under the projects SASPIT (ID 16KIS1858), QUDIS (ID 16KIS2089), and 6G-RIC (ID 16KISK033).

Date of this document: 2025-01-23.

Soon after its introduction two important streams of research ignited: the first stream improves the performance and physical security of CSIDH, by making implementations secure against timing attacks or fault injections. Such implementations come in three different flavors: 1) *Constant-time* implementations require that the runtime of the algorithm is independent of any secret data, to prevent *passive* attacks using side-channel analysis. This is nowadays a standard requirement of any cryptographic software used in real-world applications. Due to its probabilistic nature, CSIDH is difficult to implement in constant-time, which led to many approaches [17, 28, 31]. The most efficient implementation in this area is CTIDH [3], a CSIDH variant with a radically different key space, leveraging combinatorial advantages over previous approaches. 2) *Dummy-free* implementations require that the necessity of operations in the algorithm is independent of secret data, to provide a first line defense against *active* attacks, such as fault injections. Most of the above constant-time implementations rely on dummy operations to achieve the constant-time property, by adding operations that do not affect the result depending on values in the secret key. This leads to fault injection attacks [13, 14, 27]. Hence, applications that require protection against such active attacks require dummy-free implementations of CSIDH [12, 17]. 3) *Deterministic* implementations require that the algorithm does not need access to a reliable source of entropy during the computation, and has a very low variance in its runtime. For CSIDH, [17] shows that this requires more isogeny degrees than available in CSIDH-512, and suggests increasing parameters to at least 1500 bits instead of 512 bits in CSIDH-512. Others have explored deterministic CSIDH for larger parameters [12]. Section 2 contains a more detailed overview of the available CSIDH implementations.

The second stream of research focused on cryptanalysis of the hardness of the underlying isogeny-based security assumption, especially with regard to quantum attacks [9, 11, 18, 33]. Although the discussion around the required parameter sets for quantum-secure CSIDH has not been settled, recent works such as [18] suggest that it might be required to go to base fields of at least 2048 bits to achieve quantum security. Although we do not take a position in this debate, we follow the more recent line of work [12, 18] that studies CSIDH implementations for large parameters, starting from 2048-bit base fields. In particular, [12] provides both state-of-the-art implementations for deterministic and dummy-free CSIDH, named dCSIDH, as well as for general probabilistic CSIDH, using CTIDH, in large parameters. Their performance evaluation shows that the additional requirement of a deterministic and dummy-free implementation for dCSIDH compared to probabilistic and dummy-based CTIDH slows down performance by a factor of $\approx 3$ for all security levels starting from 2048-bit fields.

**Our contributions.** In this work, we tackle this large gap and present dCTIDH, a deterministic implementation of CTIDH for large parameter sets. Contrary to the results in [12], we show that deterministic behavior can be achieved without a significant performance hit. On the contrary, dCTIDH *outperforms* state-of-the-art probabilistic CTIDH.

1. We provide a new batching technique based on *Widely Overlapping Meta Batches* (WOMBats) that allows for an efficient instantiation of deterministic CTIDH. It combines the approaches of computing multiple isogenies per batch and overlapping batches. We analyze its properties, describe how to evaluate WOMBats, and how to efficiently implement dCTIDH in constant time.

2. We explore the choice of prime $p$ and parameters for dCTIDH along two different dimensions. First, the number $n$ of small odd divisors $\ell_i$ dividing $p+1$ and the configuration of WOMBats have an immense impact on the performance of dCTIDH. Given $n$, we use a greedy search algorithm to find (locally) optimal WOMBat configurations. Second, the largest power of two $2^f$ dividing $p+1$ has a significant

impact on the efficiency of low-level $\mathbb{F}_p$-arithmetic, more precisely on the cost of modular reductions. This leads to an interesting trade-off: choosing fewer isogeny degrees $n$ leads to worse performance measured in $\mathbb{F}_p$-arithmetic, but allows for primes with a larger power $2^f$ dividing $p+1$, and in turn for faster $\mathbb{F}_p$-arithmetic, and vice versa. We measure these effects in detail, which allows us to choose optimal primes and parameters for dCTIDH, and do so for 2048-bit base fields.

3. We implement our dCTIDH instantiation in C, based on the dCSIDH code package [12], focusing on 2048-bit prime parameters. This allows for a direct comparison to dCSIDH-2048, and shows that our deterministic CTIDH approach is 4.99 times faster than dCSIDH-2048. Somewhat surprisingly, our deterministic dCTIDH-2048 implementation outperforms non-deterministic CTIDH-2048 [12] by 17%, setting a new speed record for large-parameter CSIDH/CTIDH (see Table 1).

Table 1: Performance results of a group action evaluation in total number of $\mathbb{F}_p$-operations (measured in $\mathbb{F}_p$-multiplications) and in median megacycle count (Mcycles) of 25,000 experiments, performed on an Intel Core i7-6700 (Skylake) CPU.

| *variant* | *source* | $\mathbb{F}_p$-mult. | Mcycles |
|---|---|---|---|
| CTIDH-2048 | [12] | 358,307 | 1,695.38 |
| dCSIDH-2048 | [12] | 1,542,704 | 7,039.53 |
| dCTIDH-2048-205 | This work. | **314,370** | 1,430.31 |
| dCTIDH-2048-194 | This work. | 317,359 | **1,409.47** |

**Availability of software.** Our optimized dCTIDH implementation and parameter search scripts are available at

https://github.com/PaZeZeVaAt/dCTIDH/.

*Remark* 1. As CTIDH relies crucially on dummy-operations, dCTIDH is not dummy-free, in contrast to dCSIDH. However, we argue that the benefit of a dummy-free implementation is limited: Although dummy-freeness prevents some fault injection attacks from the literature, it does not prevent disorientation fault attacks [4] or side-channel attacks based on curve detection [15]. Therefore, a truly physical-attack resistant implementation of CSIDH requires a much wider scope of protection than merely being dummy-free. Nevertheless, we outline approaches to achieve a dummy-free implementation of dCTIDH in Appendix A, whose performance loss seems much less severe than when using dCSIDH instead.

**Related work.** CSIDH is not the only approach to post-quantum NIKEs. By generalizing the class group action construction, SCALLOP [21], SCALLOP-HD [19], and PEARL-SCALLOP [1] provide another post-quantum cryptographic group action, with the advantage that the group structure is known. Unfortunately, for equivalent security sizes, these generalized group actions are a few orders of magnitude slower, and therefore only seem beneficial where it is crucial to know the group structure.

Two other approaches are Clapoti(s) [32] and $\otimes$-MIKE [36]. The former provides a new method to compute the class group action $E \to E/\mathfrak{a}$ using higher-dimensional isogenies, whereas the latter generalizes the class group action to the module action to provide a post-quantum NIKE. So far, neither have efficient implementations yet. This work therefore also serves as a baseline for future implementations of Clapoti(s) and $\otimes$-MIKE: We provide a state-of-the-art deterministic version of a CSIDH-based NIKE using realistic parameters, serving as a reference point of efficiency for future higher-dimensional NIKEs.

Beyond isogeny-based cryptography, the lattice-based Swoosh [24] achieves a post-quantum NIKE, although it seems that the keys are prohibitively large.

**Outline.** Section 2 introduces the necessary background on elliptic curves and isogenies, and gives an overview of existing CSIDH implementations. Section 3 introduces the WOMBat batching technique and describes how to instantiate the deterministic dCTIDH variant. Section 4 describes how to choose parameters for dCTIDH, considering both optimizing for minimal number of $\mathbb{F}_p$-multiplications, and fast $\mathbb{F}_p$-arithmetic. Section 5 presents our optimized dCTIDH implementation and compares its performance to the prior state of the art. Appendix A gives an outline on methods towards dummy-free dCTIDH.

# 2 Preliminaries

In this section, we briefly introduce the mathematical background of CSIDH, and recall relevant results from the vast literature on efficient constant-time implementations. We refer to Silverman [37] for a comprehensive introduction to elliptic curves and isogenies.

**Isogenies.** An isogeny $\varphi : E \to E'$ is a non-constant morphism between elliptic curves $E$ and $E'$, sending $\infty_E$ to $\infty_{E'}$. An isogeny is uniquely determined by its kernel, which is a finite subgroup of $E$, and Vélu's formulas [39] allow us to compute an isogeny given a description of its kernel. The degree of an isogeny is the size of its kernel. Isogenies from $E$ to itself are called endomorphisms, and the set of all endomorphisms forms a ring under composition and addition, called the endomorphism ring of $E$.

## 2.1 CSIDH

**Background.** Let $p$ be a prime such that $p+1 = 2^f \cdot g \cdot \prod_{i=1}^{n} \ell_i$ with small primes $\ell_i$, $2^f \geq 4$, and a small odd cofactor $g \geq 1$ chosen such that $p$ is prime. In the following, we will work with the set $\mathcal{E}$ of supersingular elliptic curves over $\mathbb{F}_p$, whose endomorphism ring over $\mathbb{F}_p$, denoted $\mathcal{O}$, is isomorphic to $\mathbb{Z}[\sqrt{-p}]$. All curves in $\mathcal{E}$ have $p+1$ $\mathbb{F}_p$-rational points, and we have $E(\mathbb{F}_p) \cong \mathbb{Z}_{2^f} \times \mathbb{Z}_g \times \prod_{i=1}^{n} \mathbb{Z}_{\ell_i}$.

The main operation in CSIDH can be represented by walks in isogeny graphs $\mathcal{G}_{\mathcal{E},\ell_i}$ for primes $\ell_i$, whose vertices are all curves in $\mathcal{E}$, and edges are $\mathbb{F}_p$-rational $\ell_i$-isogenies. For a fixed $\ell_i$, each $E \in \mathcal{E}$ has exactly two neighbors in this graph, i.e., two curves $E'$ and $E''$ that are connected to $E$ by an $\ell_i$-isogeny.

Mathematically speaking, these graphs arise from the class group action of $\mathcal{Cl}(\mathcal{O})$ on the set $\mathcal{E}$. In particular, we are interested in the action of certain ideals $\mathfrak{l}_i$ and $\mathfrak{l}_i^{-1}$, whose action corresponds to the two paths mentioned above. To evaluate the action of $\mathfrak{l}_i$ on $E \in \mathcal{E}$, we compute the $\ell_i$-isogeny $\varphi$ of kernel $E[\ell_i] \cap E[\pi - 1]$, where $\pi$ is the Frobenius endomorphism. This kernel consists of the group of $\mathbb{F}_p$-rational points of order $\ell_i$, together with the point at infinity $\infty$ on $E$. Note that this $\ell_i$-torsion subgroup of $E(\mathbb{F}_p)$ is generated by any $\mathbb{F}_p$-rational point $P$ of order $\ell_i$. The codomain of this isogeny is $E' = \mathfrak{l}_i * E$.

The evaluation of the action of $\mathfrak{l}_i^{-1}$ on $E$ corresponds to computing the $\ell_i$-isogeny $\varphi'$ of kernel $E[\ell_i] \cap E[\pi + 1]$. Its codomain is $E'' = \mathfrak{l}_i^{-1} * E$. The kernel subgroup consists of $\infty$ and the points of order $\ell_i$ on $E$ with $\mathbb{F}_p$-rational $x$-coordinates, but whose $y$-coordinates are defined over $\mathbb{F}_{p^2}$. That is, these points lie on the *twist* of $E$, which we denote by $E^t$. Nevertheless, in CSIDH we usually work with $x$-only arithmetic, and the resulting isogenies are $\mathbb{F}_p$-rational. Hence, all computations take place over $\mathbb{F}_p$.

This class group action is commutative: Given two ideals $\mathfrak{l}_i, \mathfrak{l}_j$ and a curve $E \in \mathcal{E}$, we have $\mathfrak{l}_i * (\mathfrak{l}_j * E) = (\mathfrak{l}_i \cdot \mathfrak{l}_j) * E = (\mathfrak{l}_j \cdot \mathfrak{l}_i) * E = \mathfrak{l}_j * (\mathfrak{l}_i * E)$. Furthermore, we have $\mathfrak{l}_i * (\mathfrak{l}_i^{-1} * E) = \mathfrak{l}_i^{-1} * (\mathfrak{l}_i * E) = E$.

**CSIDH key exchange.** The class group action applied in CSIDH combines the action of the available $\mathfrak{l}^{\pm 1}$. We fix bounds $m_i \geq 1$ and define ideals as $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$ with $e_i \in [-m_i, m_i]$. Hence, such a vector $(e_1, \ldots, e_n)$ is the secret key, and can be applied to

any supersingular curve $E \in \mathcal{E}$ through $\mathfrak{a} * E = (\prod_{i=1}^{n} \mathfrak{l}_i^{e_i}) * E$. This can be seen as walking $|e_i|$ steps through the isogeny graph $\mathcal{G}_{\mathcal{E},\ell_i}$ for each $i \leq n$, either with a *positive* or *negative orientation*, depending on the sign of $e_i$.

For the CSIDH key exchange, we pick a starting curve $E_0 \in \mathcal{E}$. Alice samples a secret key vector $(e_1, \ldots, e_n)$ that corresponds to the ideal $\mathfrak{a} = \prod \mathfrak{l}_i^{e_i}$ and computes the public key curve $E_a = \mathfrak{a} * E_0$. Similarly, Bob samples a secret $\mathfrak{b}$ and computes the public key $E_b = \mathfrak{b} * E_0$. Due to the commutativity of the class group action, Alice can compute $E_{ba} = \mathfrak{a} * E_b$, while Bob computes $E_{ab} = \mathfrak{b} * E_a$, and $E_{ba} \cong E_{ab}$ is the shared secret.

**Security of CSIDH.** The security of CSIDH relies on the isogeny finding problem: Given curves $E, E' \in \mathcal{E}$, find an $\mathbb{F}_p$-rational isogeny $\varphi : E \to E'$, or equivalently an ideal $\mathfrak{a}$ such that $E' = \mathfrak{a} * E$. The hardness of this problem relies on two parameters: the size of the key space, determined by the bounds $m_i$, and the size of the class group, which equals the cardinality of $\mathcal{E}$. Heuristics imply that the latter is roughly $\sqrt{p}$.

The classical security of CSIDH mainly relies on the size of the key space, under the assumption that $\#\mathcal{E}$ is at least of this size. CSIDH-512, the original proposal for NIST Level I security, uses a 512-bit prime $p$ with 74 $\ell_i$, and $m_i = 5$ for all $i$, reaching a key space of size $(2 \cdot 5 + 1)^{74} \approx 2^{256}$. On the other hand, the quantum security of CSIDH is debated (see, e.g., [9, 33]), and mostly relies on the size of $p$. Several works analyze the required sizes of $p$, where the most conservative choice is to use 4096–bit primes for NIST Level I [18]. Note that these quantum attacks do not rely on the size of the key space. This means that we can specify key spaces smaller than $\sqrt{p}$ to balance classical and quantum security. For large primes $p$, it then becomes natural to restrict to $e_i \in \{-1, 0, 1\}$ when enough distinct small prime divisors $\ell_i$ of $p + 1$ are available for a large enough key space.

In this work, we do not take a position in the ongoing debate about CSIDH's quantum security. However, we follow [12, 18] and focus on deterministic variants. We shall see that these variants require relatively large primes starting from 2048 bits, although some of our techniques may apply to smaller parameter sets such as CSIDH-512 too.

## 2.2 Computing (Chains of) Isogenies

The isogeny $\varphi_{\mathfrak{a}} : E \to E/\mathfrak{a}$ can be computed as a chain of $\ell_i$-isogenies by the decomposition $\mathfrak{a} = \prod \mathfrak{l}_i^{e_i}$. There are several methods to optimally compute such a chain of isogenies. We give a brief overview of methods "via the kernel," as is standard for CSIDH. Other methods, most notably Clapoti(s) [32], are out of scope for this work.

**Vélu's formulas.** The classical work by Vélu [39] shows how to compute an isogeny $\varphi : E \to E/\langle P \rangle$ given a description of $P \in E$. For our purposes, $P \in E(\mathbb{F}_p)$ of order $\ell_i$, and so these formulas are efficient, allowing us to compute an $\ell$-isogeny and evaluate points in $\tilde{\mathcal{O}}(\ell)$. A significant asymptotic improvement to $\tilde{\mathcal{O}}(\sqrt{\ell})$ is given by [7] and denoted √élu. This approach outperforms the classical Vélu formulas roughly for $\ell > 89$.

**Amortizing the cost.** To optimize the cost of evaluating the $\ell_i$-isogenies, we can use a single point $P$ of order $p + 1$ from which we derive a chain of isogenies, analogous to how a single point of order $2^k$ can be used to compute a $2^k$-isogeny efficiently in $k$ isogenies of degree 2 [23]. That is, by multiplying $P \in E$ with the right cofactor, we obtain a point $P' \in E$ of order $\ell_i$, from which we derive $\varphi : E \to E/\langle P' \rangle$ and $P \leftarrow \varphi(P)$ using Vélu's formulas. Repeating this, we can evaluate $E \to \left(\prod_{i \in S} \mathfrak{l}_i\right) * E$ for a subset $S \subset [1..n]$.

This allows us to compute the *positive* part $\mathfrak{a}^+ = \prod \mathfrak{l}_i^{e_i}$ for those $e_i > 0$ using $\max |m_i|$ points of order $p + 1$. The *negative* part can be computed similarly, but instead using points $P \in E^t(\mathbb{F}_p)$ of order $p + 1$. Specifically, by choosing $e_i \in \{-1, 0, +1\}$ we are ensured

Table 2: CSIDH variants targeting NIST Level I ($\lambda = 128$), where the second half assumes the larger sizes required for quantum security. The last three columns describe whether the variant is constant-time (**CT**), dummy-free (**DF**), or deterministic (**Det.**). All measurements are given in GCycles and obtained on the same hardware, an Intel Core i7-6700 (Skylake) CPU.

| Variant | Size (in bits) | Group Action (in GCycles) | **CT** | **DF** | **Det.** |
|---|---|---|---|---|---|
| CSIDH [16] | 512 | 0.114 | ✗ | ✗ | ✗ |
| MCR [28] | 512 | 0.242 | ✓ | ✗ | ✗ |
| OAYT [31] | 512 | 0.188 | ✓ | ✗ | ✗ |
| CTIDH [3] | 512 | 0.124 | ✓ | ✗ | ✗ |
| Dummy-free [17] | 512 | 0.348 | ✓ | ✓ | ✗ |
| Derandomized [17] | $\approx 1500$ | – | ✓ | ✗ | ✓ |
| SQALE [18] | 2048 | 6.209 | ✓ | ✓ | ✗ |
| CTIDH [12] | 2048 | 1.695 | ✓ | ✗ | ✗ |
| dCSIDH [12] | 2048 | 7.039 | ✓ | ✓ | ✓ |

that we can evaluate $E \to \mathfrak{a} * E$ using only two starting points of order $p+1$, from now on denoted $P_+ \in E(\mathbb{F}_p)$ and $P_- \in E^t(\mathbb{F}_p)$.

**Strategies.** Similar to the strategies in SIDH [23], we can improve the performance of a chain of isogenies by pushing appropriate intermediate points $P''$ through $\varphi : E \to E/\langle P' \rangle$. As long as the cost of such an evaluation is cheaper than the cost of the multiplication required to retrieve $\varphi(P'')$ from $\varphi(P)$, this decreases the cost of the chain. [20] explores how to generalize the approach in SIDH for $2^k$-isogenies to the CSIDH case, where we have an isogeny chain of degree $\prod \ell_i$ and we must push both positive and negative points.

**Finding kernel generators.** In the above, we assumed that points of order $p+1$ are readily available on any curve $E \in \mathcal{E}$. In practice, finding such points is relatively inefficient [34]. Instead, CSIDH implementations usually rely on a probabilistic approach: We sample random points $P_+ \in E(\mathbb{F}_p)$ resp. $P_- \in E^t(\mathbb{F}_p)$ and proceed as if they have order $p+1$. However, for each $\ell_i \mid p+1$, we only have a probability of $1 - 1/\ell_i$ for the order of $P_+$ to be divisible by $\ell_i$, and analogously for $P_-$. If we fail to find a suitable kernel generator, we simply skip the corresponding $\ell_i$-isogeny. The algorithm then repeats as many rounds of point samplings until all isogeny computations succeeded.

## 2.3 Constant-time Implementations of CSIDH

The secret key $(e_1, \ldots, e_n)$ essentially describes only how many isogenies of a certain degree need to be performed and in which direction. It is therefore essential that an implementation does not leak information on either. As the cost of an isogeny heavily depends on the degree of the isogenies, this is a non-trivial task to perform in constant-time. We describe the most well-known methods to achieve such *constant-time* implementations, separated into three categories. We give a full overview of these different variants and implementations of CSIDH in Table 2.

*Remark* 2. Note that the term *constant time* does not refer to a constant running time, but to the fact that the running time is independent of secret data. Indeed, constant-time implementations of CSIDH usually rely on the probabilistic approach to sample points described above, leading to a variable running time. Nevertheless, if these implementations

avoid data flow from secret input to branch conditions, array indices or other secret-dependent behavior, they are considered constant-time.

### 2.3.1 Dummy-based Implementations

Using dummy operations, we can achieve constant-time behavior by essentially adding superfluous operations that mask secret information. We list three approaches for dummy-based constant-time CSIDH, sorted both chronologically and performance-wise.

**MCR.** Meyer, Campos, and Reith [28] propose to use strictly positive exponents $e_i \in \{0, \dots, m_i\}$, and to perform $m_i - e_i$ dummy $\ell_i$-isogenies to ensure that any evaluation performs $m_i$ isogenies of degree $\ell_i$ regardless of the secret key $(e_1, \dots, e_n)$, and that there is nothing to leak about the orientation: it is always positive. This achieves constant-time behavior, at the cost of roughly doubled bounds $m_i$ compared to "traditional" CSIDH, and a much larger total number of isogenies, $\sum m_i$ compared to $\sum |e_i|$.

**OAYT.** Onuki, Aikawa, Yamazaki, and Takagi [31] improve on the MCR-approach using two points, both a negative and a positive point, and hide the orientation of an isogeny in constant time by picking the corresponding positive or negative kernel generator. Thus, we can use the same bounds $m_i$ as "traditional" CSIDH, leading to improved performance compared to the MCR-approach.

**CTIDH.** CTIDH [3] takes a rather different approach. Using *Matryoshka isogenies* [9], one can disguise an $\ell_i$-isogeny as an $\ell'$-isogeny for $\ell' \geq \ell_i$ using dummy operations. Thus, we can take a batch $\mathcal{B}_i = \{\ell_{i,1}, \dots, \ell_{i,N_i}\}$ of degrees and disguise each $\ell_{i,j}$-isogeny as an $\ell_{i,N_i}$-isogeny. By performing these disguised isogenies, we only leak $\sum_j |e_{i,j}|$ instead of $|e_{i,j}|$. By rearranging the key space and choosing bounds $M_i$ *per batch* $\mathcal{B}_i$, we gain a significant combinatorial advantage: instead of choosing per degree a number of isogenies $e_i$, we randomly choose (up to) $M_i$ degrees (with repetition) per batch. As the bound $M_i$ per batch is public, this achieves constant-time behavior. We explain the algorithmic details in Section 2.4. To date, CTIDH is the most efficient approach for constant-time CSIDH.

### 2.3.2 Dummy-free Implementations

Although the mentioned dummy-based implementations reach constant-time behavior, they impose vulnerabilities by *physical attacks*, e.g., fault injections, as demonstrated by [13]. It depends on the specific use case if this is problematic or not. In order to thwart this vulnerability, dummy-free implementations provide additional security against such physical attacks, beyond constant-time behavior. We list two approaches.

**Using cancellations.** [17] introduces the idea of using isogenies that cancel each other out. Assuming an even bound $m_i$, and by choosing only even secret exponents $e_i \in [-m_i, \dots, m_i]$, we can always compute the action of $\mathfrak{l}_i^{e_i}$ using first $e_i$ isogenies in the correct direction, followed by $m_i - e_i$ isogenies in alternating directions. As $m_i - e_i$ is even, the alternating isogenies effectively cancel each other out. We thus compute $m_i$ isogenies of degree $\ell_i$ to compute the action of $\mathfrak{l}_i^{e_i}$ without performing dummy isogenies. However, this increases $m_i$ significantly, and there is potentially a physical attack which uses *two* fault injections to skip pairs of alternating isogenies.

**Unitary secret exponents.** SQALE [18] explores large-parameter CSIDH using primes that allow more than $2 \cdot \lambda$ small odd primes $\ell_i$ and, therefore, propose to sample secret exponents $e_i$ only from $\{-1, +1\}$. This ensures that a secret key $\mathfrak{a}$ can always be evaluated by performing precisely one isogeny per degree $\ell_i$. By hiding the orientation of this single isogeny, we ensure constant-time behavior without using any dummy isogenies.

### 2.3.3 Deterministic Implementations

Beyond constant-time behavior and security against physical attacks, real-world implementations often require *deterministic* behavior, i.e., void of randomness. Deterministic behavior can be required for a number of reasons, for example, the cost of proper entropy can be prohibitive, or the risk of random behavior can be deemed too high. Additionally, the probabilistic point sampling approach prohibits the definition of an upper bound for the worst-case runtime of the CSIDH action. dCSIDH [12] is a deterministic and dummy-free variant of CSIDH, using the approach from SQALE [18] and providing full-torsion points $P_+$ and $P_-$ in the public key, among other improvements.[1] Thus, there is no need to probabilistically sample points in the key-agreement phase of CSIDH, since all isogenies can be computed from the provided $P_+$ and $P_-$. We note that this variant requires to validate that $P_+$ and $P_-$ have full order, to prevent attacks through malicious points. Moreover, it suffices to represent these point in the public key via a seed of a few bytes, instead of transmitting the full points. [17] outlines this approach for a derandomized version of CSIDH, yet using dummy isogenies and without giving details or an implementation.

## 2.4 Algorithmic Details of CTIDH

In this section, we detail some algorithmic building blocks of CTIDH [3] that will become relevant in the remainder of this work. As described above, CTIDH distributes the available factors $\ell_i$ into batches $\mathcal{B}_i = \{\ell_{i,1}, \ldots, \ell_{i,N_i}\}$ of size $N_i$ in consecutive order. It fixes a bound $M_i$ per batch $\mathcal{B}_i$, and samples secret keys such that $\sum_j |e_{i,j}| \leq M_i$. The CTIDH algorithm computes exactly $M_i$ isogenies for the batch $\mathcal{B}_i$ by filling up with $M_i - \sum_j |e_{i,j}|$ dummy isogenies. Hence, for a given vector of batch sizes $B = (B_1, \ldots, B_n)$ and bounds $M = (M_1, \ldots, M_n)$, we obtain a key space of size

$$\#\mathcal{K}_{N,M} = \prod_{i=1}^{n} \Phi(N_i, M_i), \text{ where } \Phi(x, y) = \sum_{k=1}^{\min\{x,y\}} \binom{x}{k} 2^k \binom{y}{k}.$$

Evaluating the CTIDH action in constant time then requires to hide the exact degree of each computed isogeny, i.e., making each isogeny for a batch $\mathcal{B}_i$ look equal through the timing channel. CTIDH achieves this through the following techniques.

**Matryoshka isogenies.** The cost for computing Vélu or $\sqrt{}$élu isogenies directly depends on the degree $\ell_i$, with a complexity of $\widetilde{O}(\ell_i)$ resp. $\widetilde{O}(\sqrt{\ell_i})$. For achieving the same number of operations for isogenies of all degrees $\ell_{i,j}$ within a batch $\mathcal{B}_i$, CTIDH uses the Matryoshka structure of isogenies, outlined for the Vélu approach in [9]. Given a point of order $\ell_i$, these formulas essentially evaluate the kernel polynomial $h_S(x) = \prod_{s \in S}(x - x([s]P))$ at certain inputs $x$ to compute the codomain and evaluate points, where $S = \{1, \ldots, (\ell_i - 1)/2\}$, and $x(P)$ denotes the $x$-coordinate of $P$. The Vélu formulas thus cycle through the points $[s]P$ and generate and evaluate $h_S(X)$ on the fly, leading to a total cost of $\widetilde{O}(\ell_i)$. It is then easy to compute an $\ell_i$-isogeny at the cost of an $\ell_j$ with $\ell_j > \ell_i$, by adding dummy steps to

[1] Note that [12] also explores CTIDH in large-parameters to give a performance comparison to dCSIDH, but leaves the analysis of *deterministic* CTIDH variants as future work. Our work closes this gap by exploring deterministic CTIDH.

this loop after reaching the bound $(\ell_i - 1)/2$. Hence, we can compute an isogeny of any degree within a batch $\mathcal{B}_i$ at the cost of the maximal degree $\ell_{i,N_i}$.

This approach can be extended to √élu isogenies. In these formulas, $h_S(x)$ is evaluated via a baby-step giant-step approach. We equivalently set $S = \{1, 3, \ldots, \ell_i - 2\}$, and split it into a "box" $U \times V$ and a "leftover set" $W$ such that $S \leftrightarrow (U \times V) \cup W$. Then $h_S(x)$ can be computed by multiplying the resultant of $h_U(x)$ and a polynomial derived from $h_V(x)$ with $h_W(x)$. All sets $U, V, W$ can be chosen of size $O(\sqrt{\ell_i})$, and the resulting computations only require $\widetilde{O}(\sqrt{\ell_i})$ operations. Imposing a Matryoshka structure on √élu is then easy: for a batch $\mathcal{B}_i$, we choose $U, V$ optimal for the smallest degree $\ell_{i,1}$, and $W$ according to the largest degree $\ell_{i,N_i}$. Then we use an analogous approach to the Vélu case, and introduce Matryoshka dummy operations in the computation of the linear part $h_W(x)$. √élu Matryoshka isogenies thus become slightly less efficient due to the non-optimal choices of $U, V, W$, and depend on both the largest and smallest degree per batch. We write $\texttt{Matryoshka}_{[\ell_i,\ell_j]}$ for such an isogeny using Vélu or √élu up to $\ell_i$, depending on the size of $\ell_i$, and then Vélu for the remaining, possibly dummy, steps up to $\ell_j$. Thus, $\texttt{Matryoshka}_{[\ell_i,\ell_j]}$ can be used to perform any isogeny of degree between $\ell_i$ and $\ell_j$ at the cost of $\ell_j$. For more details, we refer to [3, 7].

**Point multiplications.** CTIDH samples points $P_+$ and $P_-$, and multiplies them by appropriate cofactors to find a kernel generator of order $\ell_{i,j}$ before computing the respective $\ell_{i,j}$-isogeny. For efficiency reasons, these multiplications use Differential Addition Chains (DACs) [5], which compute a precomputed fixed sequence of a doubling and several differential additions.[2] However, the length of these DACs, and therefore the computational effort for a multiplication by $\ell_i$ directly depends on $\ell_i$. Since CTIDH requires to keep the isogeny degree $\ell_{i,j}$ secret, it requires constant-time multiplications by all factors within a batch $\mathcal{B}_i$. CTIDH guarantees this by precomputing an optimal DAC for each $\ell_{i,j} \in \mathcal{B}_i$, but potentially padding this DAC with dummy steps, such that multiplication by any cofactor from $\mathcal{B}_i$ requires the same number of operations.

**Point rejections.** As outlined above, sampling points and multiplying by an appropriate cofactor to obtain a point of order $\ell_i$ has a failure probability of $1/\ell_i$. Thus, for a batch $\mathcal{B}_i = \{\ell_{i,1}, \ldots, \ell_{i,N_i}\}$ this point rejection probability varies per degree. To mitigate this, CTIDH artificially fixes the failure probability to the maximum $1/\ell_{i,1}$ by introducing an additional coin flip of suitable success probability per degree $\ell_{i,j}$. In particular, after successfully finding a point of order $\ell_{i,j}$, the coin flip decides if we artificially reject this point anyway, in order not to leak information on which point orders have been generated through the rejection rate. Note that in comparison to traditional CSIDH implementations, this step requires an additional usage of high-quality randomness.

Using these building blocks, CTIDH proceeds in multiple rounds of sampling points $P_+, P_-$, attempting to compute only one isogeny per batch to prevent secret-dependent behavior. In total, CTIDH thus requires at least $\max\{M_i\}$ rounds, plus potential extra rounds in the case of point rejections. The performance of CTIDH relies on a good choice of batches $\mathcal{B}_i$ and bounds $M_i$. The best way to find such parameters is through a two-layer greedy approach. For detailed algorithms, we refer to [3].

# 3 Deterministic CTIDH Using WOMBat Batching

We have seen that CTIDH is probabilistic, using multiple rounds of sampling points and computing at most one isogeny per batch. This approach is incompatible with the idea

[2] See [6] for the state of the art on this topic.

of a deterministic action using only one pair of full-order points for limited numbers of factors $\ell_i$ and reasonably large batch sizes. In order to reach deterministic behavior in CTIDH, we analyze the approaches of multiple isogenies per batch and overlapping batches in Section 3.1, whose interplay leads to a new batching technique we call WOMBats. In Section 3.2, we explain how to evaluate WOMBats in constant time, and how to combine several WOMBats to achieve a deterministic implementation, and in Section 3.3, we show that our WOMBat approach is compatible with the optimal strategies framework, leading to optimal performance for given batching parameters. Since techniques like Matryoshka isogenies strongly rely on dummy operations, we will focus on a dummy-based approach in this work. However, we describe how to design dummy-free variants of these tools in Appendix A.

## 3.1 WOMBats: Widely Overlapping Meta-Batches

Earlier versions of CSIDH [12, 17, 18] analyze the efficiency of a key space with $e_i \in \{-1, +1\}$ so that the full group action evaluation $E \to E/\mathfrak{a}$ can be performed with only two full-order points $P_+ \in E(\mathbb{F}_p)$ and $P_- \in E^t(\mathbb{F}_p)$, given at the start. We first show that such an approach can easily be adapted to CTIDH's batching technique, massively decreasing the number of isogenies required, before generalizing this approach.

**Multiple isogenies per batch.** For each round of point sampling, the original CTIDH instantiation [3] computes exactly one isogeny per batch to avoid secret-dependent behavior, e.g., when multiple isogenies of the same degree have to be computed, or when re-trying to compute isogenies after point rejections. This problem completely vanishes when using unitary secret exponents: for a batch $\mathcal{B}_i = \{\ell_{i,1}, \ldots, \ell_{i,N_i}\}$ we can compute any number $M_i \leq N_i$ of isogenies of distinct degrees through $M_i$ applications of $\texttt{Matryoshka}_{[\ell_{i,1},\ell_{i,N_i}]}$. The key space covered by this batch is then given by

$$2^{M_i} \cdot \binom{N_i}{M_i}, \quad \text{resp.} \quad \sum_{j=0}^{M_i} 2^j \cdot \binom{N_i}{j}$$

when allowing dummy isogenies. Especially for large batches, this leads to a combinatorial effect that allows for reducing the required number of isogenies significantly. We illustrate this in the following example.

**Example 1.** Consider dCSIDH-2048 from [12] with a key space of size $2^{221}$, using 221 prime factors $\ell_i$, and unitary secret keys restricted to $e_i \in \{-1, 1\}$, therefore computing exactly one isogeny per degree $\ell_i$ in the group action evaluation. If we instead view $\{\ell_1, \ldots, \ell_{221}\}$ as a single batch $\mathcal{B}_1$, it suffices to pick $M_1 = 52$ to reach a key space of size $2^{221}$, allowing dummy isogenies. Hence, we only need to compute 52 isogenies, using $\texttt{Matryoshka}_{[\ell_1,\ell_{221}]}$, instead of 221 isogenies in dCSIDH-2048. However, due to the large batch size, the cost of $\texttt{Matryoshka}_{[\ell_1,\ell_{221}]}$ is rather high: in this example, we would have to compute 52 isogenies at the cost of an $\ell_{221}$-isogeny with *no* use of $\sqrt{\text{élu}}$, whereas many of the $\ell_i$-isogenies in dCSIDH are significantly cheaper and *can* use $\sqrt{\text{élu}}$.

**Overlapping batches.** A different approach to achieving a deterministic variant of CTIDH is to use overlapping batches. Assume we have a batch $\mathcal{B}_1 = \{\ell_1, \ldots, \ell_{N_1}\}$. Instead of starting $\mathcal{B}_2$ at $\ell_{N_1+1}$, we use an overlap of $\omega_{1,2}$ factors. That is, we set $\mathcal{B}_2 = \{\ell_{N_1-\omega_{1,2}+1}, \ldots, \ell_{N_1+N_2-\omega_{1,2}}\}$, such that $\omega_{1,2}$ factors are included in both batches, where $\omega_{1,2} \leq \min\{N_1, N_2\}$. In order to always allow for unitary secret keys, we further require the batch bounds to satisfy $M_1 + M_2 \leq N_1 + N_2 - \omega_{1,2}$, which ensures that we do not compute multiple isogenies per degree. Compared to distinct batches, this significantly

improves the combinatorial effect, and therefore reduces the number of isogenies we need to compute in order to achieve a given key space.

To further increase this effect, it appears promising to not only define two overlapping batches, but generalize this to multiple overlapping batches. However, this comes with major restrictions in the sizes of overlaps $\omega_{i,j}$ and bounds $M_i$ for all involved batches, making an analysis in full generality extremely tedious. In the following, we show that a particular instantiation of this approach is especially promising for the design of a deterministic CTIDH variant.

**Combining both approaches.** We combine both approaches—multiple isogenies per batch and overlapping batches—to achieve an efficient approach for deterministic CTIDH. For ease of notation, consider the batch $\mathcal{B} = \{\ell_1, \ldots, \ell_N\}$ and $M < N$. As described above, we can compute the required $M$ isogenies of distinct degrees via $M$ calls to $\texttt{Matryoshka}_{[\ell_1,\ell_N]}$. To lower the cost for computing these isogenies, we view $\mathcal{B}$ as $M$ overlapping batches $\mathcal{B}_1 = \{\ell_1, \ldots, \ell_{N-M+1}\}$, $\mathcal{B}_2 = \{\ell_2, \ldots, \ell_{N-M+2}\}$, …, $\mathcal{B}_M = \{\ell_M, \ldots, \ell_N\}$, where each batch has size $N-M+1$ and overlaps in $N-M$ degrees $\ell_i$ with the previous batch, and the next batch. For each batch, we then compute exactly one isogeny, running $\texttt{Matryoshka}_{[\ell_1,\ell_{N-M+1}]}$ for $\mathcal{B}_1$, $\texttt{Matryoshka}_{[\ell_2,\ell_{N-M+2}]}$ for $\mathcal{B}_2$, and so on, until $\texttt{Matryoshka}_{[\ell_M,\ell_N]}$ for $\mathcal{B}_M$. It is easy to see that this covers all possible distributions of $M$ distinct-degree isogenies that could be prescribed by the secret key: the smallest degree must be included in $\mathcal{B}_1$, the second-smallest degree must be included in $\mathcal{B}_2$, and so on. We illustrate this approach in Example 2. The benefit of this approach is a significant improvement in performance, as we will see in Example 3.

The above approach uses $M$ overlapping batches to achieve a single 'meta'-batch. We define a WOMBat to generalize this approach, which allows us to achieve an efficient deterministic CTIDH variant.

**Definition 1.** A Widely Overlapping Meta Batch (WOMBat) is a batch $\mathcal{W} = \{\ell_{i,1}, \ldots, \ell_{i,N}\}$ of size $N$ and bound $M$, where each $\ell_{i,j} \in \mathcal{W}$ has exponent $e_{i,j} \in \{-1, 0, 1\}$. Thus, all isogenies can be performed by a single full-order point, at a cost of $M$ Matryoshka isogenies through $\texttt{Matryoshka}_{[\ell_{i,1},\ell_{i,N-M+1}]}$ up to $\texttt{Matryoshka}_{[\ell_{i,M},\ell_{i,N}]}$.

**Example 2.** As a concrete example, consider the WOMBat $\mathcal{W}$ of size $N = 10$ with prime degrees $\{3, 5, 7, 11, 13, 17, 19, 23, 29, 31\}$ and consider the bound $M = 5$. The smallest of the five isogeny degrees specified by a secret key must lie between 3 and 17, so we compute the corresponding isogeny via $\texttt{Matryoshka}_{[3,17]}$, followed by $\texttt{Matryoshka}_{[5,19]}$ for the second-smallest degree, and so on. This is illustrated in Figure 1.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{W}$ | = ( | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | ) |
| $\mathcal{W}^{(1)}$ | = ( | 3 | **5** | 7 | 11 | 13 | 17 | ) | | | | $\texttt{Matryoshka}_{[3,17]}$ |
| $\mathcal{W}^{(2)}$ | = | ( | 5 | 7 | **11** | 13 | 17 | 19 | ) | | | $\texttt{Matryoshka}_{[5,19]}$ |
| $\mathcal{W}^{(3)}$ | = | | ( | 7 | 11 | **13** | 17 | 19 | 23 | ) | | $\texttt{Matryoshka}_{[7,23]}$ |
| $\mathcal{W}^{(4)}$ | = | | | ( | 11 | 13 | 17 | 19 | **23** | 29 | ) | $\texttt{Matryoshka}_{[11,29]}$ |
| $\mathcal{W}^{(5)}$ | = | | | | ( | 13 | 17 | 19 | 23 | **29** | 31 | ) $\texttt{Matryoshka}_{[13,31]}$ |

Figure 1: Illustration of a WOMBat $\mathcal{W}$ of size $N = 10$ and bound $M = 5$, performing the secret key $(0, 1, 0, 1, 1, 0, 0, 1, 1, 0)$. The computed isogeny per sub-batch $\mathcal{W}^{(i)}$ of $\mathcal{W}$ is depicted by a blue square.

**Example 3.** Recall from Example 1 that our simple deterministic CTIDH approach required to compute 52 isogenies via $\texttt{Matryoshka}_{[\ell_1,\ell_{221}]}$. Instead, we can define a WOMBat $\mathcal{W}$ of size $N = 221$ and bound $M = 52$, whose isogenies can be computed using $\texttt{Matryoshka}_{[\ell_1,\ell_{170}]}$, $\texttt{Matryoshka}_{[\ell_2,\ell_{171}]}$, ..., $\texttt{Matryoshka}_{[\ell_{52},\ell_{221}]}$, significantly improving efficiency. However, each batch is still very wide, resulting in limited usage of $\sqrt{\text{élu}}$.

Similar to how CTIDH is more effective when using $n$ batches instead of a single large batch, we may naturally expect an improvement in performance using $n$ disjoint WOMBats $\mathcal{W}_1, \ldots, \mathcal{W}_n$ instead of the single WOMBat discussed in Example 3. We name this variant dCTIDH, using and evaluating $N_{\mathcal{W}}$ disjoint WOMBats $\mathcal{W}_i$ of size $N_i$ and bound $M_i$. As discussed above, our approach is deterministic. Furthermore, Section 3.2 will explain how to evaluate WOMBats in constant time, making dCTIDH constant-time and deterministic.

**Key space.** The key space of a WOMBat $\mathcal{W}$ of size $N$ and bound $M$ is easily calculated as $\Psi(N, M) := \binom{N}{M} \cdot 2^M$. Similarly, for $N_{\mathcal{W}}$ disjoint WOMBats $\mathcal{W}_1, \ldots, \mathcal{W}_{N_{\mathcal{W}}}$, each of size $N_i$ and bound $M_i$, the resulting key space is

$$\prod_{i=1}^{N_{\mathcal{W}}} \Psi(\mathcal{W}_i) = \prod_{i=1}^{N_{\mathcal{W}}} \binom{N_i}{M_i} \cdot 2^{M_i}.$$

We can further allow dummy isogenies, by choosing between 0 and $M_i$ real isogenies per WOMBat, padded by dummy isogenies to ensure we always perform precisely $M_i$ isogenies in total per WOMBat. The key space then increases to

$$\prod_{i=1}^{N_{\mathcal{W}}} \Psi_{\text{dummy}}(\mathcal{W}_i) = \prod_{i=1}^{N_{\mathcal{W}}} \sum_{j=0}^{M_i} \binom{N_i}{j} \cdot 2^j.$$

## 3.2 Evaluating a WOMBat

This section analyzes how to evaluate the action of a secret key derived using the WOMBat approach in constant time. Let $\mathfrak{a}$ be such a secret key, with $(e_1, \ldots, e_n)$ drawn from the key space with $N_{\mathcal{W}}$ WOMBats $\mathcal{W}_1, \ldots, \mathcal{W}_{N_{\mathcal{W}}}$. Let $\mathfrak{a}_i$ denote the part of the secret key associated to WOMBat $\mathcal{W}_i = (\ell_{i,1}, \ldots, \ell_{i,N_i})$, represented as $e_{\mathcal{W}_i} = (e_{i,1}, \ldots, e_{i,N_i})$.

Assuming we are given a pair of points $(P_{i,+}, P_{i,-})$ of order of the required isogeny, i.e., their order is the product of all $\ell_{i,j}$ for which $e_{i,j} \neq 0$, Algorithm 1 details a deterministic evaluation of $\mathcal{W}_i$. Each step in Algorithm 1 can be implemented in constant time, using similar techniques as in CTIDH [3], as we describe below.

**Scalar multiplications.** Algorithm 1 involves several scalar multiplications, whose scalars directly depend on the secret key. However, implementing this in constant time is equivalent to the case in CTIDH [3]: we compute the maximal DAC length of the factors $\ell_{i,j} \in \mathcal{W}_i$, and use dummy steps if necessary, such that all multiplications $[\ell_{i,j}]P$ for $\ell_{i,j} \in \mathcal{W}_i$ run in the same number of steps. Furthermore, the number of necessary multiplications only depends on the public parameter $M_i$. Picking the correct DAC and factors $\ell_{i,j}$ requires constant-time look-ups and swaps. Note that we process the degrees $\ell_{i,j}$ in descending order, which saves computational effort in Line 9 of Algorithm 1, since multiplications by large factors are repeated less often.

**Isogenies.** The Matryoshka isogenies are equivalent to the ones used in CTIDH [3], hence run in constant time. In our case, the upper and lower bounds vary throughout Algorithm 1, yet this only depends on the public parameters $M_i$ and $N_i$. Keeping the actual

**Algorithm 1** WOMBat evaluation EvaluateWombat

**Input:** WOMBat $\mathcal{W}_i$ of size $N_i$ and bound $M_i$, Montgomery coefficient $A$ of the domain curve, partial secret key $e_{\mathcal{W}_i} = (e_{i,1}, \ldots, e_{i,N_i})$, points $P_{i,+}, P_{i,-}$ of order of the required isogeny, a list of points `pts` to be pushed through isogenies.

**Output:** Montgomery coefficient $A$ of the codomain curve, a list `pts` of image points.

1: **for** $r = 0$ **to** $M_i - 1$ **do**
2: Pick the largest $k$ with $e_{i,k} \neq 0$.
3: **if** $e_{i,k} = 1$ **then**
4: $K \leftarrow P_{i,+}$
5: **else**
6: $K \leftarrow P_{i,-}$
7: **end if**
8: **for all** $j \in \{1, \ldots, k\}$ with $e_{i,j} \neq 0$ **do**
9: $K \leftarrow [\ell_{i,j}]K$
10: **end for**
11: $(A, (P_{i,+}, P_{i,-}, \texttt{pts})) \leftarrow \texttt{Matryoshka}_{[\ell_{M_i - r}, \ell_{N_i - r}]}(\ell_{i,k}, A, K, (P_{i,+}, P_{i,-}, \texttt{pts}))$
12: $(P_{i,+}, P_{i,-}) \leftarrow ([\ell_{i,k}]P_{i,+}, [\ell_{i,k}]P_{i,-})$
13: $e_{i,k} \leftarrow 0$
14: **end for**
15: **return** $(A, \texttt{pts})$

isogeny degree $\ell_{i,k}$ secret works equivalently as in CTIDH. In Algorithm 1 the function $\texttt{Matryoshka}_{[\ell_i, \ell_j]}$ takes as input the isogeny degree $\ell_k$ that must satisfy $\ell_i \leq \ell_k \leq \ell_j$, the Montgomery curve coefficient $A$ of the domain curve, a kernel generator $K$ of order $\ell_k$, and a list of points to be evaluated. It outputs the codomain Montgomery coefficient, and the list of evaluated points. In a dummy-based implementation, we simply discard the result of Line 11 in the dummy case.

**Point swaps.** Algorithm 1 contains an if-branch that depends on the secret key. However, it is easy to implement this via a simple constant-time point swap depending on the sign of $e_{i,k}$. Equivalently, the implementation of CTIDH [3] relies on this technique.

**Deterministic CTIDH using WOMBats.** Given this action per $\mathfrak{a}_i$, we easily obtain a full evaluation $E \to E/\mathfrak{a}$ via Algorithm 2. Given $P_+$ and $P_-$ of full order $\prod \ell_i$, we first multiply out all unused $\ell_i$, i.e., those $\ell_i$ with $e_i = 0$, in constant time as described above. To achieve this, note that the number of scalar multiplications per WOMBat $\mathcal{W}_i$ is $N_i - M_i$, which is constant and publicly known. We obtain $P_+, P_-$ both of order $\prod_{e_i \neq 0} \ell_i$. Then, per WOMBat $\mathcal{W}_i = (\ell_{i,1}, \ldots, \ell_{i,N_i})$, we clear all orders $\ell_{i',j}$ with $i' \neq i$ that have not been cleared before to get the two points $(P_{i,+}, P_{i,-})$ both of order $\prod_{e_{i,j} \neq 0} \ell_{i,j}$ used in Algorithm 1. While evaluating $\mathcal{W}_i$, we push the original points $P_+, P_-$ through to use these for the next WOMBat. All multiplications run in constant time when implemented as described above.

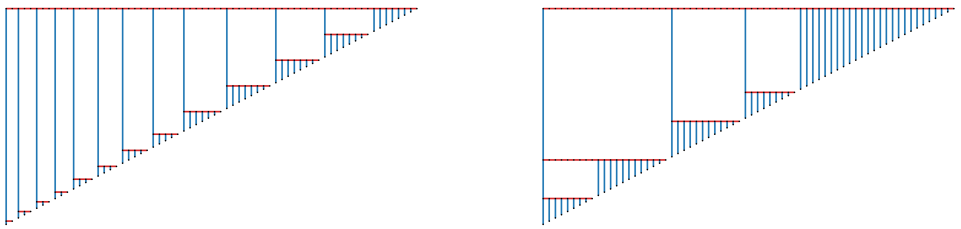
**Finding full-order points.** Algorithm 2 assumes two points $P_+, P_-$ of full order, that is, of order $\prod \ell_i$, which are computationally expensive to find.[3] For the public key generation phase, we can simply precompute such points on the starting curve $E_0$ and use them as system parameters. In the key agreement phase however, this is not possible. To avoid the costly points sampling in this phase, we can shift this task to the public-key generation phase: after computing a public key curve $E_{\mathsf{pk}}$, full-order points are sampled and included

[3] Indeed, this is why most CSIDH implementations use probabilistic point sampling.

**Algorithm 2** Deterministic CTIDH action using WOMBats

**Input:** Montgomery coefficient $A$ of the domain curve, secret key $(e_1 \dots, e_n)$, $N_{\mathcal{W}}$ WOMBats $\mathcal{W}_i$ of size $N_i$ and bound $M_i$, and full-order $P_+ \in E(\mathbb{F}_p)$ and $P_- \in E^t(\mathbb{F}_p)$.
**Output:** Montgomery coefficient $A$ of the codomain curve.

1: **for all** $i \in \{1, \dots, n\}$ with $e_i = 0$ **do**
2: $\quad P_+ \leftarrow [\ell_i]P_+$, $P_- \leftarrow [\ell_i]P_-$
3: **end for**
4: **for** $i = N_{\mathcal{W}}$ **to** 1 **do**
5: $\quad (P_{i,+}, P_{i,-}) \leftarrow (P_+, P_-)$
6: $\quad$ **for** $j = 1$ **to** $i - 1$ **do**
7: $\quad\quad$ **for all** $\ell_{j,k} \in \mathcal{W}_j$ with $e_{j,k} \neq 0$ **do**
8: $\quad\quad\quad P_{i,+} \leftarrow [\ell_{j,k}]P_{i,+}$, $P_{i,-} \leftarrow [\ell_{j,k}]P_{i,-}$
9: $\quad\quad$ **end for**
10: $\quad\quad (A, P_+, P_-) \leftarrow$ EvaluateWombat$(i, A, e_{\mathcal{W}_i}, P_{i,+}, P_{i,-}, P_+, P_-)$
11: $\quad$ **end for**
12: **end for**
13: **return** $A$



(a) Multiplicative evaluation per WOMBat. (b) Optimal strategy combining WOMBats.

Figure 2: Strategies to evaluate dCTIDH, corresponding to the dCTIDH-2048-205 parameter set (see Section 5), using a 2048-bit prime $p$, and 13 WOMBats of differing sizes.

in the public key. To avoid a large increase of public key sizes, we follow the approach of dCSIDH [12]: instead of directly sampling points (resp. their $x$-coordinates), we try small seeds $u \in \mathbb{F}_p$ and deterministically compute the two points from this seed using the `Elligator` sampling approach [8, 9]. In the key agreement phase, we then only have to expand this seed $u$ via `Elligator` to get the points $P_+$ and $P_-$. As in dCSIDH, $u$ only increases the public key size by a few bytes. The points derived from $u$ have order $p + 1$, so a user needs to clear the unused cofactor $2^f \cdot g$ as well as the unused factors $\ell_i$ for which $e_i = 0$ (Lines 1-3 in Algorithm 2) before starting the action.

## 3.3 Using Optimal Strategies

The previous section shows how to evaluate $E \to E/\mathfrak{a}$ by separately evaluating WOMBats, following the approach of CTIDH. Following Algorithm 1, each WOMBat is evaluated through a multiplicative strategy. Instead, we could use the optimal strategy framework [20] to define other strategies per WOMBat, replacing some multiplications by point evaluations. However, our experiments show that for the relatively small numbers of isogenies per WOMBat that we will be using, the multiplicative approach is optimal in most cases.

Figure 2a depicts the multiplicative approach, using the code from Chi-Domínguez and Rodríguez-Henríquez [20]. In a nutshell, each segment of a vertical line is a scalar multiplication by a factor $\ell_i$ and each segment of a horizontal line is a point evaluation

through an $\ell_i$-isogeny. The algorithm starts in the top left corner, and we need to reach the lower diagonal of the triangle at each vertical level to generate a point of suitable order, which generates the next isogeny, i.e., the next horizontal step. At the end of the algorithm, we reach the upper right corner after computing all corresponding isogenies. We refer to [20] for more details. Figure 2a visualizes our approach, with each subtriangle corresponding to the evaluation of a single WOMBat using Algorithm 1, resulting in an overall multiplication-based evaluation of the 13 distinct WOMBats.

**Optimal strategies in dCTIDH.** Although the approach of evaluating each WOMBat separately might make sense intuitively, there is no reason why this approach should be optimal. Instead, contrary to the CTIDH case, we may also see the full evaluation $E \to E/\mathfrak{a}$ as a single chain of isogenies that can be evaluated using optimal strategies [20], which do not necessarily have to be split 'per WOMBat'. That is, given the points $P_+$, $P_-$, we can ignore the batching structure by simply using the same multiplicative cost for each factor $\ell_{i,j}$ within a WOMBat $\mathcal{W}_i$. We observe that the series of calls to $\texttt{Matryoshka}_{[\ell_{i,j},\ell_{i,k}]}$ is constant without requiring to prescribe the precise moment they are called, and thus, we can compute an optimal strategy without restricting to the batching structure. Figure 2b shows the resulting optimal strategy for the same parameters as in Figure 2a. Indeed, the optimal approach does not evaluate each WOMBat separately, but even places degrees of the same WOMBat in separate subtriangles. In our example, the optimal strategy outperforms the multiplicative approach by 7.5%.

Note that the algorithm still runs in constant time, since the number and size of multiplications and calls to `Matryoshka` are constant, following the explanation in Section 3.2. We refer to [20] for more details and algorithms for evaluating optimal strategies.

*Remark* 3. Multiplicative approaches usually benefit from evaluating the corresponding isogenies in descending order of their degrees [29], while optimal strategies benefit from swapping this to an ascending ordering [20]. Our analysis takes this into account.

## 3.4 Key Validation

To ensure no information is leaked on the secret key $\mathfrak{a}$ during evaluation, public keys must be validated. This requires two checks: first, that the curve $E_{\mathsf{pk}}$ is supersingular, and second, that the seed $u$ generates two points $P_+, P_-$ whose order is divisible by all $\ell_i$. Otherwise, a maliciously crafted public key might contain a seed generating a point $P_+$ whose order is not divisible by, say, $\ell_1 = 3$. A successful derivation of the shared secret then leaks that no positive $\ell_1$-isogeny was performed, hence leaking information on $\mathfrak{a}$. This is in contrast with dCSIDH, which computes an $\ell_i$-isogeny for every $\ell_i$ regardless of the secret key, and hence the order of $P_+$ and $P_-$ is verified throughout the computation $E \to E/\mathfrak{a}$ and does not leak information.

Whenever $\prod \ell_i > 4\sqrt{p}$, ensuring a point has order at least $\prod \ell_i$ also verifies supersingularity. Thus, for dCTIDH, we only need to verify the second claim: that $P_+$ and $P_-$ have orders divisible by all $\ell_i$. We use Algorithm 4 from [34]: we multiply $P_+$ and $P_-$ by the cofactor $(p+1)/\prod \ell_i$ to get points $P'_+$ and $P'_-$, and then compute the reduced Tate pairing $\zeta = t_L(P'_+, P'_-)$ of degree $L = \prod \ell_i$, using cubical arithmetic [35]. We then verify that the result $\zeta \in \mu_L$ has order precisely $L$ using a product-tree approach, using the fact that we can apply fast arithmetic using Lucas sequences for such elements [38].

In terms of cost, this requires two Montgomery ladders of $\log p - \log L$ bits, the pairing computation of $\log L$ bits, and the product-tree verification of length $L \log L$ bits. Montgomery ladders have a well-known cost of $6\mathbf{M} + 4\mathbf{S} + 12\mathbf{A}$ per bit, whereas the pairing computation requires both $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ arithmetic, and therefore takes $19\mathbf{M} + 4\mathbf{S} + 41\mathbf{A}$ per bit. The product-tree verification requires $1\mathbf{M} + 1\mathbf{S}$ per bit. In total, this gives us a

close estimate of the cost of key validation, which comes down to $60,000$ $\mathbb{F}_p$-operations for dCTIDH using a prime of 2048 bits.

# 4 Optimal dCTIDH Parameters

While Section 3 presents optimized approaches for evaluating the group action in dCTIDH, it requires as input a WOMBat batch configuration, i.e., the number $N_{\mathcal{W}}$ of wombats $\mathcal{W}_i$ and their sizes $N_i$ and bounds $M_i$. Optimizing this configuration is a difficult task, and the main focus of this section.

While for a given prime $p$, finding WOMBat batching parameters is related to the approach provided in CTIDH [3], we further need to take into account the choice of $p$. Recall that our primes are of shape

$$p + 1 = 2^f \cdot g \cdot \prod_{i=1}^{n} \ell_i.$$

For dCSIDH, choosing $p$ is somewhat trivial, as the number $n$ of distinct prime factors $\ell_i$ is prescribed by the required size $2^n$ of the key space. This is then padded with $f$ as large as possible to reach the target size of $p$, and a cofactor $g$ to ensure $p$ is prime. In contrast, the combinatorial advantage in dCTIDH allows to reach the same key space size with smaller $n$, which, in turn, allows for increasing $f$, the power of 2 dividing $p+1$, leading to more efficient $\mathbb{F}_p$-arithmetic. In this section, we analyze in detail how to balance the two involved dimensions in dCTIDH: the number of WOMBats $N_{\mathcal{W}}$ with the associated optimal batching parameter, which determines the number of $\mathbb{F}_p$-operations, and the size of $n$, impacting the efficiency of the $\mathbb{F}_p$-arithmetic. For a concrete treatment, we will limit to 2048-bit primes $p$ in this section, and note that adapting our search approach to larger parameters is straightforward.

## 4.1 Optimizing WOMBat Batches

Given a prime $p$, we want to optimize the WOMBat batching parameters in terms of a cost function. We adapt the greedy search from CTIDH [3] to our setting.

**Cost function.** Analogously to CTIDH, we measure the cost of a group action evaluation in $\mathbb{F}_p$-operations, given all parameters $p$, $N_{\mathcal{W}}$, $N_i$, and $M_i$. While in CTIDH this cost function must account for the probabilistic nature of the algorithm, and therefore the variability of the runtime, this is greatly simplified in dCTIDH. Due to the deterministic approach and assuming full-order points as input, we can predict the exact number of operations. For this we use the optimal strategy framework [20] adapted to our context, to find a (locally) optimal strategy for evaluating the class group action, and output the number of $\mathbb{F}_p$-operations. Using a fixed ratio between multiplications, squarings, additions, and inversions, we translate this to an equivalent of $\mathbb{F}_p$-multiplications, i.e., $\mathbf{S} = \mathbf{M}$ and ignoring $\mathbf{A}$. Note that we need to add a small overhead to the obtained optimal strategy cost to account for the scalar multiplications required to clear the cofactors $\prod_{e_i=0} \ell_i$ before entering the strategy.

**Greedy search.** Given a prime $p$, we adapt the multi-layer greedy search from CTIDH [3] to find optimal batching parameters. To achieve this, we fix the number of WOMBats $N_{\mathcal{W}}$, and let the upper level of the greedy search algorithm optimize for the best choice of WOMBat sizes $N_i$ such that $\sum N_i = n$. To determine the optimal set of $N_i$, each step in this upper layer calls the lower level of the greedy search, which, given the sizes $N_i$, searches for the optimal choice of bounds $M_i$ such that we reach a large enough

key space. In particular, we start with equally distributed $N_i$ and $M_i = \lfloor N_i/2 \rfloor$. The lower layer optimizes by calling the cost function from above to get precise costs for the group action evaluation for this specific set of $N_i$ and $M_i$. The steps the greedy algorithm takes to optimize in the upper and lower layer are analogous to the CTIDH greedy implementation [3]. Running this multi-layer greedy search for different numbers of WOMBats $N_{\mathcal{W}}$ then leads to an optimal parameter configuration for the chosen prime $p$.

Note that this algorithm finds a *local* minimum, but ideally our goal is to be as close as possible to the *global* minimum. To increase the probability of reaching this minimum, we add a randomization step when selecting the bounds $M_i$ per batch and set $M_i = \lfloor N_i/2 \rfloor - \delta$ where $\delta \in \{0, \ldots, 5\}$ is a random integer. Such a random starting choice might obtain a different local optimal choice of $M_i$. Due to the increased search space when running the greedy search multiple times, we increase the probability for reaching the global minimum of $\mathbb{F}_p$-multiplications for the given $p$. However, the results are no longer reproducible between runs.

*Remark* 4. Our approach produces two invalid edge cases: 1) when there is no possible configuration to obtain a large enough key space for the specified number of batches $N_{\mathcal{W}}$, and 2) when the resulting parameter set contains *empty* batches with a bound of $M_i = 0$. In the second case, there must be a better parameter set with smaller $n$ and without empty batches. Therefore, we disregard such configurations.

## 4.2 Impact of 2-valuation of $p + 1$ on Performance

While Section 4.1 discusses optimizations at a higher level, this section focuses on the underlying field arithmetic. Our arithmetic is strongly based on the Karatsuba approach from [12], which uses the word version of the Montgomery reduction from [2] (see Algorithm 3) for efficient reductions modulo $p$. The main idea of this approach is to reduce the number of limbs to be multiplied depending on the cofactor $2^f = 2^{kw+\varepsilon}$, where $w$ is the word size in bits and $\varepsilon < w$. Note that the complexity of this algorithm is dominated by the $\nu$ multiplications of $\alpha 2^{(kw)-w}$ by a single word value $r_0$ (see Line 4 of Algorithm 3). The combinatorial advantage of dCTIDH enables the value of $k$ to be increased, which leads to a speedup of the reduction. As indicated in Line 4 of Algorithm 3, increasing $k$ leads to a multiplication in which the least $k - 1$ words do not have to be taken into account. Particularly, increasing $k$ leads to a speedup by saving at least $k \cdot \nu$ MULX instructions and $k \cdot \nu \cdot 2$ ADOX/ADCX instructions compared to the standard Montgomery reduction (see Algorithm 3 in [2]). We present and discuss the resulting performance figures for our implementation in Section 4.3. For more details on this Montgomery reduction approach, we refer to [2].

**Algorithm 3** Word version of the Montgomery reduction if $p = 2^{kw}\alpha - 1$

**Input:** $0 \leq a < p\beta^{\nu}$, where $\beta$ is the radix with $\beta = 2^w$, and $\nu =$ number of limbs
**Output:** $r = a\beta^{-\nu} \mod p$ and $0 \leq r < p$

1: $r \leftarrow a$
2: **for** $i = 0$ **to** $\nu - 1$ **do**
3: $\quad r_0 \leftarrow r \mod \beta$
4: $\quad r \leftarrow (r - r_0)/\beta + r_0 \times \alpha 2^{(kw)-w}$
5: **end for**
6: $r' \leftarrow r + (\beta^{\nu} - p)$
7: **if** $r' \geq \beta^{\nu}$ **then**
8: $\quad r \leftarrow r' - \beta$
9: **end if**
10: **return** $r$

Table 3: Benchmarking results for performing an $\mathbb{F}_p$-multiplication for $k \in \{1, \ldots, 12\}$. Numbers are median clock cycles of $10^8$ executions on a Skylake CPU.

| $k$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4120 | 4109 | 4085 | 4045 | 4029 | 3991 | 3967 | 3925 | 3896 | 3880 | 3847 | 3825 |

## 4.3 Selecting Parameter Sets

This section combines the search for optimal batches from Section 4.1 with the performance model from Section 4.2 to find optimal parameter sets for dCTIDH in terms of cycles, for a prime $p$ of 2048 bits.

As our primes are of the form $p+1 = 2^f \cdot g \cdot \prod_n \ell_i$, we first look for the optimal number $n$ of small odd primes $\ell_i$ that gives the lowest cost for a group action evaluation in terms of $\mathbb{F}_p$-multiplications. The lower bound is $n = 151$, given by the minimal number of $\ell_i$ required to reach a key space of size $2^{221}$, and the upper bound is $n = 226$. This gives the first dimension of the search space for optimal parameter sets, which directly also determines the largest power $2^f$ dividing $p+1$ that is available: the larger $n$ is, the smaller $2^f$ must be. The second dimension is the number of batches $\mathcal{W}_i$ into which we divide the $n$ small odd primes $\ell_i$, which we range from 1 to 18. This gives us a two-dimensional search space of $75 \cdot 18$ possible configurations. Exploring this search space took about 3 weeks on an AMD EPYC 7643 running on 192 threads.

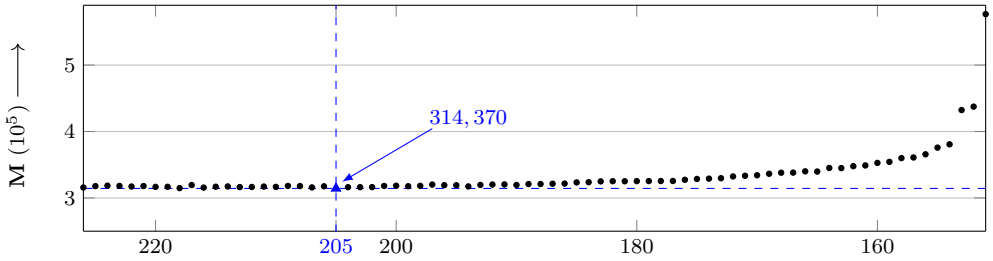
**In terms of $\mathbb{F}_p$-multiplications.** Given the possible configurations, we run the greedy algorithm described in Section 4.1 to find optimal strategies in terms of $\mathbb{F}_p$-multiplications. Figure 3 summarizes these results, showing the cost of the best strategy we could find for $n = 226$ to $n = 151$ targeting a key space of size $2^{221}$.

The plot highlights several key observations regarding the performance of dCTIDH. Notably, the action reaches a minimum at $n = 205$ with $314,370$ multiplications. In the range from $n = 226$ to $n = 193$, the results show minimal divergence, with only 5816 multiplications separating these points. This variation corresponds to less than $\pm 1\%$ of the average, likely due to the compensating effects of the reduced overhead from unused $\ell_i$ and the increasing number of isogenies, as the number of $\ell_i$ decreases.

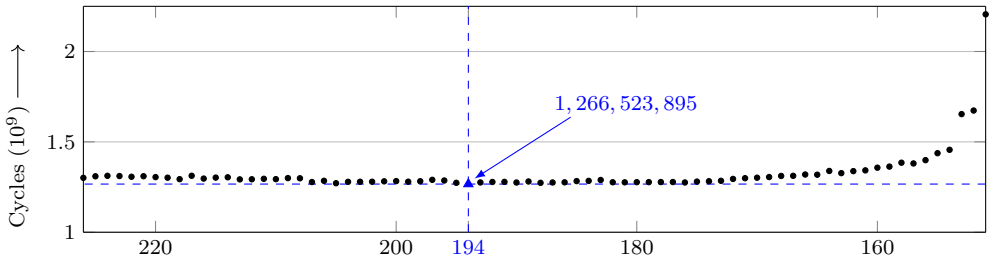
Starting at $n = 186$, the plot shows a noticeable upward slope, due to the reduced number of available $\ell_i$, resulting in fewer and larger batch sizes. Consequently, we see an increase in overhead due to the wider Matryoshka isogeny bounds. The extrema at $n = 151$ represent the upper bound of the observed trend, where only a single WOMBat can be utilized. This emphasizes the growing inefficiency as the number of $\ell_i$ decreases.

**In terms of cycles.** In terms of CPU cycles, as described in Section 4.2, the size of $k$ significantly impacts the cost of an $\mathbb{F}_p$-multiplication. The size of $k$ is directly related to the number $n$ of $\ell_i$, as a larger $n$ allows for fewer bits left for $k$, assuming a fixed bit size for the resulting prime $p$. We thus multiply the number of multiplications obtained by greedy per configuration by the cycles per multiplication given in Table 3 for $k \in \{1, \ldots, 12\}$. Due to the optimized arithmetic, we find that this shifts the optimal parameter set from the one obtained by simply optimizing only the $\mathbb{F}_p$-multiplication count.

Figure 4b shows this effect. The new optimal configuration shifts to the right and is now at $n = 194$ with 1.266 Gcycles. Further, the plateau between $n = 226$ and $n = 193$ from Figure 4a transforms into a slight downwards slope as multiplication costs reduce. Up to $n = 177$, this scaling compensates for the increasing number of multiplications, maintaining an overall low cycle count. However, beyond this point, the larger multiplication counts associated with fewer $\ell_i$ overwhelm the benefits provided by the larger $2^k$-cofactor. This




(a) Total number of $\mathbb{F}_p$-operations per number of $\ell_i$.




(b) Total number of clock cycles per number of $\ell_i$.

Figure 3: Cost of a group action evaluation using the optimal strategy for $n$ from 226 to 151 in $\mathbb{F}_p$-multiplications and clock cycles. Minima indicated by blue triangles.

tipping point is evident in the plot, where the CPU cycle curve begins to rise steeply, illustrating the diminishing returns as the number of $\ell_i$ decreases further.

Ultimately, the behavior of the curve emphasizes the need to balance the number of $\ell_i$ with the 2-valuation, as indicated by the new optimal configuration at $n = 194$.

# 5 Implementation Results

Following the findings of Section 4, this section gives the performance results for our selected parameters.[4]

**Implementation.** As outlined above, we employ optimal strategies for our WOMBat evaluation, making the dCSIDH [12] code the clear choice as a starting point for our implementation. We extend the code base using the constant-time strategies discussed in Section 3.2 and Section 3.3, including Matryoshka isogenies, fixed length multiplications, and constant-time lookups. In the case of dummy isogenies, we utilize constant-time swaps to discard the isogeny result. We furthermore add the finite-field speedup from Section 4.2 to our asm-based Karatsuba implementation.

**Constant-time verification.** In order to be constant-time, we designed each step of our algorithm without secret-dependent branches and memory accesses. To verify this property for our implementation, we carried out automatic tests based on the `ctgrind` [26] library and the `valgrind` [30] analysis tool. All tests were completed successfully, indicating that branches and memory addresses are independent of any secret inputs.

[4]All benchmarks in this section are performed on an Intel Core i7-6700 (Skylake) CPU running Debian 12 with Hyper-threading and Turbo Boost disabled and compiled with gcc-12.2.0.

**Results.** Based on the results discussed in Section 4, we implement three configurations. dCTIDH-2048-226 serves as the *baseline*, given it uses the same prime and arithmetic as CTIDH-2048 and dCSIDH-2028. We choose the other configurations, dCTIDH-2048-205 and dCTIDH-2048-194, based on their optimal parameters for multiplications and CPU cycles. Table 4 summarizes the selection.

Table 4: Parameter overview for dCTIDH-2048 for a key space of $2^{221}$ bits. All primes are of the form $p+1 = 2^f \cdot g \cdot \prod_{i=1}^{n} \ell_i$. dCTIDH-2048-226 uses the prime and arithmetic from CTIDH-2048 [12], the other two primes use the first $n = 205$, resp. $n = 194$, odd primes $\ell_i$.

| *variant* | *criteria* | isogenies | WOMBats | 2-valuation ($2^f$) | cofactor ($g$) |
|---|---|---|---|---|---|
| dCTIDH-2048-226 | baseline | 67 | 15 | $2^{64}$ | 1 |
| dCTIDH-2048-205 | best mults | 68 | 13 | $2^{64\cdot4}$ | $2^{19} \cdot 13 \cdot 17$ |
| dCTIDH-2048-194 | best cycles | 69 | 12 | $2^{64\cdot6}$ | $2^3 \cdot 7 \cdot 41$ |

Table 5: Performance results of a group action evaluation in multiplications (**M**), squarings (**S**), and additions (**A**), and median megacycle count (Mcycles) of 25,000 experiments, performed on an Intel Core i7-6700 (Skylake) CPU. The column '$\mathbb{F}_p$-mult.' is calculated using the model $\mathbf{S} = \mathbf{M}$, ignoring additions. Bold numbers denote the best performance.

| *variant* | **M** | **S** | **A** | $\mathbb{F}_p$-mult. | Mcycles |
|---|---|---|---|---|---|
| CTIDH-2048 [12] | 279,586 | 78,721 | – | 358,307 | 1,695.38 |
| dCSIDH-2048 [12] | 1,315,203 | 227,501 | – | 1,542,704 | 7,039.53 |
| dCTIDH-2048-226 | 265,192 | 50,572 | 474,336 | 315,764 | 1,497.72 |
| dCTIDH-2048-205 | 263,545 | 50,825 | 465,224 | **314,370** | 1,430.31 |
| dCTIDH-2048-194 | 266,101 | 51,258 | 469,258 | 317,359 | **1,409.47** |

Table 5 summarizes the performance of different dCTIDH-2048 configurations. These results reveal significant performance improvements through optimized configurations and key space selection. The effect of selecting a parameter with a larger $2^f$-cofactor becomes evident when comparing the performance of dCTIDH-2048-226, dCTIDH-2048-205, and dCTIDH-2048-194. dCTIDH-2048-205 achieves a 4.5% reduction in cycle count compared to dCTIDH-2048-226 (1,430.31 Mcycles versus 1,497.72 Mcycles). dCTIDH-2048-194 improves further, reducing the cycle count by 5.9% compared to dCTIDH-2048-226 (1,409.47 Mcycles vs. 1,497.72 Mcycles), resp. 1.46% compared to dCTIDH-2048-205, even though both other configurations require notable lower $\mathbb{F}_p$-operations.

**Conclusion.** All dCTIDH-2048 configurations significantly outperform dCSIDH-2048, by a speed-up of up to 4.9 times in $\mathbb{F}_p$-operations and 5.0 times in Mcycles. We gain improvements of up to 12.2% in $\mathbb{F}_p$-operations and 16.8% in cycles in comparison to non-deterministic CTIDH-2048. Overall, dCTIDH emerges as the most efficient CSIDH implementation to date, achieving notable speedups compared to both CTIDH and dCSIDH.

# A Towards Dummy-Free CTIDH

We briefly sketch two techniques that may be used to remove dummy operations from (deterministic) CTIDH. We leave a full treatment of these techniques as future work.

**Dummy-free Matryoshka Isogenies.** The Matryoshka isogeny, as introduced in [9] and used in CTIDH [3], uses dummy operations to hide the degree $\ell$ of the isogeny, by either multiplying the kernel polynomial $h_S(x)$ by a real factor $x - x([i]P)$ when $i \leq \frac{\ell-1}{2}$, or performing a dummy multiplication when $i > \frac{\ell-1}{2}$ (see Section 2.4). Using that $x([i]P) = x([\ell - i]P)$, we can ensure that all multiples $x([i]P)$ have to be correctly

computed, which ensures that no operation in this computation is 'dummy' and leaving no room for fault-injections. Namely, instead of performing a dummy multiplication whenever $i > \frac{\ell-1}{2}$, we instead always multiply $h_S(x)$ by $(x - x([i]P)/2) - \alpha \cdot (x[i]P)/2$, where $\alpha = -1$ whenever the value of $x([i]P)$ has appeared already for any $j < i$, and 1 otherwise, computed in constant time. Thus, every multiplication is real and takes the same cost: the multiplication is by $x([i]P)$ when $i \leq \frac{\ell-1}{2}$, and by $x$ otherwise. This gives us a dummy-free variant of the Matryoshka isogeny. However, for technical reasons, the bounds of each batch need to be carefully reconsidered, as this dummy-free Matryoshka becomes more expensive for larger batches. Initial experiments suggest only a moderate increase in cost.

**Dummy-free DACs.** Recall from Section 2.4 that CTIDH and dCTIDH precompute optimal DACs for each $\ell_{i,j} \in \mathcal{W}_i$ and potentially add dummy differential additions to reach the same length for each DAC. To avoid dummy operations, we can instead fix the maximal optimal DAC length per WOMBat, and try to precompute (suboptimal) DACs of precisely this length for each $\ell_{i,j}$ in this batch.[5] Our experiments show that this ensures that there are only minor restrictions to use such DACs on the batching, such as the fact that the factor 3 cannot be in any batch, and our approach leads to a very minor expected overhead for scalar multiplications in dummy-free dCTIDH. We dub this approach DACsHUND (Differential Addition Chains Having Unnecessities Needed for Dummy-freeness). Rough calculations show that DACsHUNDs are 26% faster per bit than Montgomery ladders for constant-time cofactor clearance.

## References

[1] Bill Allombert, Jean-François Biasse, Jonathan Komada Eriksen, Péter Kutas, Chris Leonardi, Aurel Page, Renate Scheidler, and Márton Tot Bagi. *PEARL-SCALLOP: Parameter Extension Applicable in Real-Life SCALLOP*. Cryptology ePrint Archive, Paper 2024/1744. 2024. URL: https://eprint.iacr.org/2024/1744 (cit. on p. 3).

[2] Jean-Claude Bajard and Sylvain Duquesne. "Montgomery-friendly primes and applications to cryptography". In: *J. Cryptogr. Eng.* 11.4 (2021), pp. 399–415. DOI: 10.1007/S13389-021-00260-Z. URL: https://doi.org/10.1007/s13389-021-00260-z (cit. on p. 17).

[3] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. "CTIDH: faster constant-time CSIDH". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.4 (2021), pp. 351–387. DOI: 10.46586/TCHES.V2021.I4.351-387. URL: https://doi.org/10.46586/tches.v2021.i4.351-387 (cit. on pp. 2, 6–10, 12, 13, 16, 17, 20).

[4] Gustavo Banegas, Juliane Krämer, Tanja Lange, Michael Meyer, Lorenz Panny, Krijn Reijnders, Jana Sotáková, and Monika Trimoska. "Disorientation Faults in CSIDH". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. Lecture Notes in Computer Science. Springer, 2023, pp. 310–342. DOI: 10.1007/978-3-031-30589-4_11. URL: https://doi.org/10.1007/978-3-031-30589-4_11 (cit. on p. 3).

[5] In minor cases, this is not possible: for example, the maximal DAC length to compute a scalar multiplication by 5 is 4 and cannot be stretched. In this case, we can resort to computing scalar multiplications by $c \cdot \ell_{i,j}$, for small $c$ coprime to all other $\ell_k \neq \ell_{i,j}$, and make sure we do not compute any other multiple by a factor $c' \cdot \ell_{i,j}$ during the chain.

[5] Daniel J Bernstein. "Differential addition chains". In: (2006). URL: http://cr.yp.to/ecdh/diffchain-20060219.pdf (cit. on p. 9).

[6] Daniel J. Bernstein, Jolijn Cottaar, and Tanja Lange. *Searching for differential addition chains*. Cryptology ePrint Archive, Paper 2024/1044. 2024. URL: https://eprint.iacr.org/2024/1044 (cit. on p. 9).

[7] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. "Faster computation of isogenies of large prime degree". In: *ANTS XIV – Proceedings of the Fourteenth Algorithmic Number Theory Symposium*. https://msp.org/obs/2020/4-1/obs-v4-n1-p04-p.pdf. MSP, 2020 (cit. on pp. 5, 9).

[8] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. "Elligator: elliptic-curve points indistinguishable from uniform random strings". In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM, 2013, pp. 967–980. DOI: 10.1145/2508859.2516734. URL: https://doi.org/10.1145/2508859.2516734 (cit. on p. 14).

[9] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. "Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies". In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. Lecture Notes in Computer Science. Springer, 2019, pp. 409–441. DOI: 10.1007/978-3-030-17656-3_15. URL: https://doi.org/10.1007/978-3-030-17656-3_15 (cit. on pp. 2, 5, 7, 8, 14, 20).

[10] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. "CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations". In: *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11921. Lecture Notes in Computer Science. Springer, 2019, pp. 227–247. DOI: 10.1007/978-3-030-34578-5_9. URL: https://doi.org/10.1007/978-3-030-34578-5_9 (cit. on p. 1).

[11] Xavier Bonnetain and André Schrottenloher. "Quantum Security Analysis of CSIDH". In: *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*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020, pp. 493–522. DOI: 10.1007/978-3-030-45724-2_17. URL: https://doi.org/10.1007/978-3-030-45724-2_17 (cit. on p. 2).

[12] Fabio Campos, Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Michael Meyer, Krijn Reijnders, Francisco Rodríguez-Henríquez, Peter Schwabe, and Thom Wiggers. "Optimizations and Practicality of High-Security CSIDH". In: *IACR Communications in Cryptology* 1.1 (2024). ISSN: 3006-5496. DOI: 10.62056/anjbksdja (cit. on pp. 2, 3, 5, 6, 8, 10, 14, 17, 19, 20).

[13] Fabio Campos, Matthias J. Kannwischer, Michael Meyer, Hiroshi Onuki, and Marc Stöttinger. "Trouble at the CSIDH: Protecting CSIDH with Dummy-Operations Against Fault Injection Attacks". In: *2020 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE, 2020, pp. 57–65. DOI: 10.1109/FDTC51366.2020.00015 (cit. on pp. 2, 7).

[14] Fabio Campos, Juliane Krämer, and Marcel Müller. "Safe-Error Attacks on SIKE and CSIDH". In: *Security, Privacy, and Applied Cryptography Engineering - 11th International Conference, SPACE 2021, Kolkata, India, December 10-13, 2021, Proceedings*. Ed. by Lejla Batina, Stjepan Picek, and Mainack Mondal. Vol. 13162. Lecture Notes in Computer Science. Springer, 2021, pp. 104–125. DOI: 10.1007/978-3-030-95085-9_6. URL: https://doi.org/10.1007/978-3-030-95085-9_6 (cit. on p. 2).

[15] Fabio Campos, Michael Meyer, Krijn Reijnders, and Marc Stöttinger. "Patient Zero & Patient Six: Zero-Value and Correlation Attacks on CSIDH and SIKE". In: *Selected Areas in Cryptography - 29th International Conference, SAC 2022, Windsor, ON, Canada, August 24-26, 2022, Revised Selected Papers*. Ed. by Benjamin Smith and Huapeng Wu. Vol. 13742. Lecture Notes in Computer Science. Springer, 2022, pp. 234–262. DOI: 10.1007/978-3-031-58411-4_11. URL: https://doi.org/10.1007/978-3-031-58411-4_11 (cit. on p. 3).

[16] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *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*. Ed. by Thomas Peyrin and Steven D. Galbraith. Vol. 11274. Lecture Notes in Computer Science. Springer, 2018, pp. 395–427. DOI: 10.1007/978-3-030-03332-3_15. URL: https://doi.org/10.1007/978-3-030-03332-3_15 (cit. on pp. 1, 6).

[17] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. "Stronger and Faster Side-Channel Protections for CSIDH". In: *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*. Ed. by Peter Schwabe and Nicolas Thériault. Vol. 11774. Lecture Notes in Computer Science. Springer, 2019, pp. 173–193. DOI: 10.1007/978-3-030-30530-7_9. URL: https://doi.org/10.1007/978-3-030-30530-7_9 (cit. on pp. 2, 6–8, 10).

[18] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. "The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents". In: *Journal of Cryptographic Engineering* (2021). DOI: 10.1007/s13389-021-00271-w (cit. on pp. 2, 5, 6, 8, 10).

[19] Mingjie Chen, Antonin Leroux, and Lorenz Panny. "SCALLOP-HD: Group Action from 2-Dimensional Isogenies". In: *Public-Key Cryptography - PKC 2024 - 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15-17, 2024, Proceedings, Part III*. Ed. by Qiang Tang and Vanessa Teague. Vol. 14603. Lecture Notes in Computer Science. Springer, 2024, pp. 190–216. DOI: 10.1007/978-3-031-57725-3_7. URL: https://doi.org/10.1007/978-3-031-57725-3_7 (cit. on p. 3).

[20] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. "Optimal strategies for CSIDH". In: *Advances in Mathematics of Communications* (2020). DOI: 10.3934/amc.2020116 (cit. on pp. 6, 14–16).

[21] Luca De Feo, Tako Boris Fouotsa, Péter Kutas, Antonin Leroux, Simon-Philipp Merz, Lorenz Panny, and Benjamin Wesolowski. "SCALLOP: Scaling the CSI-FiSh". In: *Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part I*. Ed. by Alexandra Boldyreva and Vladimir Kolesnikov. Vol. 13940. Lecture Notes in Computer Science. Springer, 2023, pp. 345–375. DOI:

10.1007/978-3-031-31368-4_13. URL: https://doi.org/10.1007/978-3-031-31368-4_13 (cit. on p. 3).

[22] Luca De Feo and Michael Meyer. "Threshold Schemes from Isogeny Assumptions". In: *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II*. Ed. by Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas. Vol. 12111. Lecture Notes in Computer Science. Springer, 2020, pp. 187–212. DOI: 10.1007/978-3-030-45388-6_7. URL: https://doi.org/10.1007/978-3-030-45388-6_7 (cit. on p. 1).

[23] Luca De Feo, David Jao, and Jérôme Plût. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies". In: *J. Math. Cryptol.* 8.3 (2014), pp. 209–247. DOI: 10.1515/JMC-2012-0015. URL: https://doi.org/10.1515/jmc-2012-0015 (cit. on pp. 5, 6).

[24] Phillip Gajland, Bor de Kock, Miguel Quaresma, Giulio Malavolta, and Peter Schwabe. "SWOOSH: Efficient Lattice-Based Non-Interactive Key Exchange". In: *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. Ed. by Davide Balzarotti and Wenyuan Xu. USENIX Association, 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/gajland (cit. on p. 3).

[25] Yi-Fu Lai, Steven D. Galbraith, and Cyprien Delpech de Saint Guilhem. "Compact, Efficient and UC-Secure Isogeny-Based Oblivious Transfer". In: *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Springer, 2021, pp. 213–241. DOI: 10.1007/978-3-030-77870-5_8. URL: https://doi.org/10.1007/978-3-030-77870-5_8 (cit. on p. 1).

[26] Adam Langley. *ctgrind*. (accessed 2024-11-21). URL: https://github.com/agl/ctgrind (cit. on p. 19).

[27] Jason T. LeGrow and Aaron Hutchinson. "(Short Paper) Analysis of a Strong Fault Attack on Static/Ephemeral CSIDH". In: *Advances in Information and Computer Security - 16th International Workshop on Security, IWSEC 2021, Virtual Event, September 8-10, 2021, Proceedings*. Ed. by Toru Nakanishi and Ryo Nojima. Vol. 12835. Lecture Notes in Computer Science. Springer, 2021, pp. 216–226. DOI: 10.1007/978-3-030-85987-9_12. URL: https://doi.org/10.1007/978-3-030-85987-9_12 (cit. on p. 2).

[28] Michael Meyer, Fabio Campos, and Steffen Reith. "On Lions and Elligators: An Efficient Constant-Time Implementation of CSIDH". In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers*. Ed. by Jintai Ding and Rainer Steinwandt. Vol. 11505. Lecture Notes in Computer Science. Springer, 2019, pp. 307–325. DOI: 10.1007/978-3-030-25510-7_17. URL: https://doi.org/10.1007/978-3-030-25510-7_17 (cit. on pp. 2, 6, 7).

[29] Michael Meyer and Steffen Reith. "A Faster Way to the CSIDH". In: *Progress in Cryptology - INDOCRYPT 2018 - 19th International Conference on Cryptology in India, New Delhi, India, December 9-12, 2018, Proceedings*. Ed. by Debrup Chakraborty and Tetsu Iwata. Vol. 11356. Lecture Notes in Computer Science. Springer, 2018, pp. 137–152. DOI: 10.1007/978-3-030-05378-9_8. URL: https://doi.org/10.1007/978-3-030-05378-9_8 (cit. on p. 15).

[30] Nicholas Nethercote and Julian Seward. "Valgrind: a framework for heavyweight dynamic binary instrumentation". In: *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10-13, 2007*. Ed. by Jeanne Ferrante and Kathryn S. McKinley. ACM, 2007, pp. 89–100. DOI: 10.1145/1250734.1250746. URL: https://doi.org/10.1145/1250734.1250746 (cit. on p. 19).

[31] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. "(Short Paper) A Faster Constant-Time Algorithm of CSIDH Keeping Two Points". In: *Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings*. Ed. by Nuttapong Attrapadung and Takeshi Yagi. Vol. 11689. Lecture Notes in Computer Science. Springer, 2019, pp. 23–33. DOI: 10.1007/978-3-030-26834-3_2. URL: https://doi.org/10.1007/978-3-030-26834-3_2 (cit. on pp. 2, 6, 7).

[32] Aurel Page and Damien Robert. "Introducing Clapoti(s): Evaluating the isogeny class group action in polynomial time". In: *IACR Cryptol. ePrint Arch.* (2023), p. 1766. URL: https://eprint.iacr.org/2023/1766 (cit. on pp. 3, 5).

[33] Chris Peikert. "He Gives C-Sieves on the CSIDH". In: *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*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020, pp. 463–492. DOI: 10.1007/978-3-030-45724-2_16. URL: https://doi.org/10.1007/978-3-030-45724-2_16 (cit. on pp. 2, 5).

[34] Krijn Reijnders. "Effective Pairings in Isogeny-Based Cryptography". In: *Progress in Cryptology - LATINCRYPT 2023 - 8th International Conference on Cryptology and Information Security in Latin America, LATINCRYPT 2023, Quito, Ecuador, October 3-6, 2023, Proceedings*. Ed. by Abdelrahaman Aly and Mehdi Tibouchi. Vol. 14168. Lecture Notes in Computer Science. Springer, 2023, pp. 109–128. DOI: 10.1007/978-3-031-44469-2_6. URL: https://doi.org/10.1007/978-3-031-44469-2_6 (cit. on pp. 6, 15).

[35] Damien Robert. *Fast pairings via biextensions and cubical arithmetic*. Cryptology ePrint Archive, Paper 2024/517. 2024. URL: https://eprint.iacr.org/2024/517 (cit. on p. 15).

[36] Damien Robert. *The module action for isogeny based cryptography*. Cryptology ePrint Archive, Paper 2024/1556. 2024. URL: https://eprint.iacr.org/2024/1556 (cit. on p. 3).

[37] Joseph H Silverman. *The arithmetic of elliptic curves*. Vol. 106. Springer, 2009 (cit. on p. 4).

[38] Martijn Stam. "Speeding up subgroup cryptosystems". Doctoral Dissertation. Technische Universiteit Eindhoven, 2003 (cit. on p. 15).

[39] Jacques Vélu. "Isogénies entre courbes elliptiques". In: *Comptes Rendus de l'Académie des Sciences de Paris, Séries A* 273 (1971), pp. 238–241 (cit. on pp. 4, 5).