Hardened CTIDH: Dummy-Free and Deterministic CTIDH

Gustavo Banegas[1], Andreas Hellenbrand[2], and Matheus Saldanha[3]

[1] Inria and Laboratoire d'Informatique de l'Ecole polytechnique, Institut Polytechnique de Paris, Palaiseau, France
`gustavo@cryptme.in`

[2] RheinMain University of Applied Sciences Wiesbaden, Germany
`andreas.hellenbrand@hs-rm.de`

[3] Universidade Federal de Santa Catarina, Florianópolis, Brazil
`matheus.saldanha@posgrad.ufsc.br`

Abstract. Isogeny-based cryptography has emerged as a promising post-quantum alternative, with CSIDH and its constant-time variants CTIDH and dCTIDH offering efficient group-action protocols. However, CTIDH and dCTIDH rely on dummy operations in differential addition chains (DACs) and Matryoshka, which can be exploitable by fault-injection attacks. In this work, we present the first *dummy-free* implementation of dCTIDH. Our approach combines two recent ideas: `DACsHUND`, which enforces equal-length DACs within each batch without padding, and a reformulated Matryoshka structure that removes dummy multiplications and validates all intermediate points. Our analysis shows that small primes such as $3, 5$, and 7 severely restrict feasible `DACsHUND` configurations, motivating new parameter sets that exclude them. We implement dummy-free dCTIDH-2048-194 and dCTIDH-2048-205, achieving group action costs of roughly 357,000–362,000 $\mathbb{F}_p$-multiplications, with median evaluation times of 1.59–1.60 (Gcyc). These results do not surpass dCTIDH, but they outperform CTIDH by roughly 5% while eliminating dummy operations entirely. Compared to dCSIDH, our construction is more than 4× faster. To the best of our knowledge, this is the first *efficient* implementation of a CSIDH-like protocol that is simultaneously deterministic, constant-time, and fully dummy-free.

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

* Author list in alphabetical order; see https://ams.org/profession/leaders/CultureStatement04.pdf. This work has been supported by the Federal Ministry of Research, Technology and Space (BMFTR) under the project QUDIS (ID 16KIS2089), and by the HYPERFORM consortium, funded by France through Bpifrance, and by the France 2030 program under grant agreement ANR-22-PETQ-0008 PQ-TLS.

Date of this document: 2025-09-11.

1 Introduction

In recent years, isogeny-based cryptography has attracted significant attention from both mathematicians and cryptographers, due to its features such as non-interactive key exchange and compact key sizes. Following the cryptanalysis of SIKE [7, 12, 16], research on isogeny-based key exchange has shifted toward CSIDH [8], which currently remains *unbroken*. The attacks that compromised SIKE–based on the supersingular isogeny framework–do not apply to CSIDH or its variants, thereby preserving their relevance as viable post-quantum key exchange candidates.

Despite its resilience, CSIDH is relatively slow compared to other post-quantum schemes. Furthermore, achieving secure implementations requires countermeasures against side-channel attacks, which further increase computational overhead. To address these limitations, variants such as CTIDH [1] and dCTIDH [5] have been proposed. These schemes improve performance by introducing more structured sets of isogeny paths and leveraging fixed parameter sets that simplify implementations. Both employ *isogeny batching* techniques, which simultaneously enhance security and performance. CTIDH achieves faster key exchange by introducing a new key space based on batches of isogenies, together with a constant-time algorithm for the CSIDH group action that synergizes with the new structure. Building on CTIDH, dCTIDH adopts a more deterministic approach, refining the batching technique through the introduction of *Widely Overlapping Meta-Batches* (WOMBats).

Beyond performance, dummy operations introduce an attack surface for *active* side-channels (fault injection): by targeting these redundant steps, an adversary can induce faults that desynchronize the control flow and leak secrets. This risk is not merely theoretical: Campos, Kannwischer, Meyer, Onuki, and Stöttinger [6] demonstrated fault attacks against dummy-padded isogeny computations, and more recently [11] exploited dummies in implementations of CSIDH which are constant-time. While batching in CTIDH raises the bar, their results indicate that **practical fault attacks remain feasible**, which motivates pursuing dummy-free techniques.

Contributions. In this work, we investigate in depth the use of `DACsHUND` and dummy-free Matryoshka isogenies, and their combined role in enabling a fully dummy-free dCTIDH implementation. Our primary goal is to produce an optimized variant of CTIDH/dCTIDH that eliminates dummy operations while maintaining strong security properties.

1. We analyze the `DACsHUND` method for dummy-free DAC computations. For each prime, we enumerate all possible DAC configurations and adapt the dCTIDH greedy parameter search to enforce equal-length DACsHUNDs within each batch, thus avoiding dummy operations. We evaluate the resulting configurations under different dCTIDH settings and quantify the performance impact.

2. We implement the dummy-free Matryoshka isogeny approach and integrate its cost into the greedy search process, enabling parameter optimization that accounts for its specific constraints.
3. We propose new dCTIDH parameter sets that leverage these dummy-free techniques for improved performance. We focus on configurations that exclude the primes $3, 5, 7$, as they are less compatible with DACsHUND.

Availability of software. Our implementation and greedy search scripts are available at

https://github.com/AndHell/hardenedCTIDH.

Related Work. Several works have sought to make CSIDH constant-time or deterministic. For instance, in [13], the authors address challenges such as point sampling and introduce the SIMBA technique. However, their approach still relies on dummy operations to compute isogenies. In parallel, other research has explored dummy-free constant-time methods, including two-point ladders and strategy-based scheduling of small-prime isogenies [14]. While these methods help mitigate timing leakage, they do not fully resolve batch-level DAC harmonization or eliminate the dummy padding inherent in Matryoshka. In CTIDH [1], the authors apply batching of isogenies using atomic blocks and Matryoshka to achieve a faster constant-time implementation of CSIDH. However, this approach is neither deterministic nor dummy-free.

Campos, Hellenbrand, Meyer, and Reijnders introduced dCTIDH [5], a deterministic variant of CTIDH. Their central innovation is the use of *WOMBats*, which combine overlapping batches with multiple isogenies per batch to enable efficient deterministic evaluation. Their implementation is highly optimized, both in terms of the number of finite-field operations per prime and the efficiency of those operations. Nevertheless, as the authors emphasize, dCTIDH still relies on dummy operations in both Matryoshka isogenies and DAC padding. As a result, it is not dummy-free, leaving open the challenge of combining determinism, constant-time execution, and full dummy-freeness in a single construction. To address this, dCTIDH proposed two potential directions: *DACsHUND* and *dummy-free Matryoshka isogenies*. In this work, we investigate these approaches in detail, with the goal of achieving the first fully dummy-free variant of dCTIDH.

Recent work has explored radical 3-isogenies as a replacement for small-degree isogenies in CSIDH-like protocols, reporting up to a $4\times$ speedup for dCTIDH [9]. However, initiating a 3-isogeny chain still requires repeated sampling, which introduces probabilistic behavior. As a result, radical 3-isogeny chains cannot be made dummy-free and remain too costly in practice compared to 2-isogeny walks.

Addressing a related challenge in isogeny computation, Bernstein, Cottaar, and Lange [2] revisit the problem of constructing differential addition chains, introducing new algorithms that minimize both chain length and computational overhead. Their work focuses on faster methods for finding minimum-length continued-fraction differential addition chains, significantly improving over previous search strategies. In our setting, we also rely on efficient DACs and employ a greedy search procedure to identify chains that satisfy the structural constraints imposed by DACsHUND. While their algorithms target global optimality in chain length, our approach prioritizes compatibility with batching and constant-time requirements, aiming at practical dummy-free instantiations within the dCTIDH framework.

2 Background

2.1 Elliptic Curves and Isogenies

Given a finite field $\mathbb{F}_p$, an *elliptic curve* E over $\mathbb{F}_p$ is defined by the equation

$$y^2 = x^3 + ax + b,$$

where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \neq 0$ to ensure the curve is nonsingular. Elliptic curves can also be expressed in alternative forms. For instance, *Montgomery curves* constitute a special class of elliptic curves defined over $\mathbb{F}_p$ by

$$By^2 = x^3 + Ax^2 + x,$$

where $A, B \in \mathbb{F}_p$ and $B(A^2 - 4) \neq 0$ guarantees nonsingularity. The group law on $E(\mathbb{F}_p)$ uses the point at infinity $\mathcal{O}$ as the identity element. For a detailed treatment of elliptic curve theory and arithmetic, see [17].

Projective Coordinates. In practice, elliptic-curve arithmetic is often performed in *projective coordinates* to avoid costly field inversions. An affine point (x, y) is represented as $(X : Y : Z)$, corresponding to $(X/Z, Y/Z)$ when $Z \neq 0$, while the point at infinity $\mathcal{O}$ is given by $(0 : 1 : 0)$. This representation replaces inversions with a few additional multiplications, making additions, doublings, and isogeny evaluations both more efficient and easier to implement in constant time.

Isogenies. An *isogeny* between elliptic curves, $\phi : E \to E'$, is a non-constant algebraic map that preserves the group law. Every isogeny is uniquely determined by its kernel, which is a finite subgroup of E. When working with Montgomery curves, these maps can be efficiently evaluated using only the x-coordinates of points, yielding major computational advantages for large-degree isogeny walks as required in CSIDH [8] and related protocols.

The use of x-only arithmetic not only simplifies the application of Vélu's formulas—the classical tool for computing an isogeny from its kernel—but also enables constant-time implementations via techniques such as the Montgomery ladder. For small odd prime degrees ℓ, the kernel is usually generated by an $\mathbb{F}_p$-rational point of order ℓ, which allows efficient construction of the corresponding quotient curve.

The two main algorithms for evaluating such isogenies are *Vélu's formulas* [18] and $\sqrt{}$*élu* method [3]. Both reduce the problem to computing a polynomial of the form

$$h_S(X) = \prod_{s \in S} \big(X - x([s]P)\big), \tag{1}$$

where P is a point of order ℓ, and S is an index set determined by ℓ. The main computational tasks are to determine the new Montgomery coefficient A' of the image curve E' and to evaluate the images of selected points under ϕ.

In the classical Vélu approach, the index set is $S = \{1, 2, \ldots, (\ell - 1)/2\}$; one computes $x([s]P)$ for all $s \in S$ and forms the product $h_S(X)$. This yields essentially linear cost in ℓ: about 4ℓ $\mathbb{F}_p$-multiplications to update A' and 2ℓ per evaluated image point, i.e., overall $\widetilde{O}(\ell)$; Vélu is conceptually simple and practically optimal for small prime degrees.

For larger ℓ, the $\sqrt{}$élu[4] algorithm applies a baby-step/giant-step decomposition on the odd-index set $S = \{1, 3, 5, \ldots, \ell - 2\}$ via $S \leftrightarrow (U \times V) \cup W$, obtaining h_S as h_W times a resultant involving h_U and a polynomial derived from h_V. This reorganizes the arithmetic to $\widetilde{O}(\sqrt{\ell})$. Vélu is a special case of $\sqrt{}$élu with $U = V = \varnothing$.

Remark 1. We note that both CTIDH and dCTIDH rely on *Matryoshka* "isogenies," a technique to enforce uniform evaluation costs across batches of primes. Because this construction is central to CTIDH, dCTIDH, and also to our work, we provide a more detailed discussion in §2.6.

2.2 CSIDH

Introduced by [8] in 2018, CSIDH is a non-interactive key exchange protocol based on the action of the ideal class group of an imaginary quadratic order on a set of supersingular elliptic curves defined over a prime field $\mathbb{F}_p$. This class group action is realized through chains of isogenies between elliptic curves, each of small odd prime degree ℓ_i dividing $p + 1$.

The protocol operates on a restricted set $\mathcal{E}$ of supersingular elliptic curves $E/\mathbb{F}_p$ whose endomorphism ring is isomorphic to $\mathbb{Z}[\sqrt{-p}]$, with all curves having exactly $p + 1$ points. For a prime p of the form

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

where $f \geq 2$, g a small cofactor, and the ℓ_i are small, distinct odd primes, the group structure of $\mathcal{E}(\mathbb{F}_p)$ admits a torsion decomposition enabling efficient computation of ℓ_i-degree isogenies.

The underlying group action is defined as follows: a secret key is a vector $(e_1, \ldots, e_n)$ with $e_i \in [-m_i, m_i]$, representing the ideal class

$$\mathfrak{a} = \prod_{i=1}^{n} \ell_i^{e_i}.$$

[4] Pronounced "square-root Vélu".

Its action on a fixed base curve E_0 is computed as a walk in the ℓ_i-isogeny graph, with each step corresponding to an ℓ_i-isogeny in the forward or backward direction according to the sign of e_i. The resulting curve $E' = \mathfrak{a} * E_0$ is the public key.

CSIDH is *commutative*: for secret keys $\mathfrak{a}$ and $\mathfrak{b}$,

$$\mathfrak{a} * (\mathfrak{b} * E_0) = \mathfrak{b} * (\mathfrak{a} * E_0),$$

allowing both parties to derive the same shared secret curve without interaction.

Security relies on the *isogeny path-finding problem*: given two supersingular curves E and E' over $\mathbb{F}_p$ with the same $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$, find an explicit $\mathbb{F}_p$-rational isogeny $\phi : E \to E'$ of smooth degree. This problem is believed to be hard for both classical and quantum algorithms when instantiated with sufficiently large p and appropriate parameters. Quantum security analysis remains active, with recent work suggesting that primes of at least 2048 bits may be required for conservative approaches.

2.3 Constant-Time Isogeny Diffie–Hellman (CTIDH)

The CTIDH [1] variant removes timing side channels by ensuring that all isogeny walks execute in constant time. Instead of conditionally applying an ℓ_i-isogeny based on the exponent e_i, CTIDH introduces a batching strategy with a redefined key space. A batch is defined as $\mathcal{B}_i = \{\ell_{i,1}, \ldots, \ell_{i,N_i}\}$, where all primes in $\mathcal{B}_i$ are handled collectively. An $\ell_{i,j}$-isogeny is then computed as an ℓ_{i,N_i}-isogeny through *Matryoshka isogenies*, which conceal the degree of each isogeny by padding smaller ones with dummy computations so that every evaluation matches the cost of an ℓ_{i,N_i}-isogeny for each batch B_i.

To further mitigate leakage, CTIDH assigns a bound m_i to each batch, prescribing a fixed number of isogeny evaluations. When the required number of evaluations is smaller than m_i, dummy isogenies are inserted so that every batch always performs exactly m_i evaluations. This masks timing variations across batches. However, it does not protect against fault attacks, since the dummy operations themselves remain a potential target.

The dominant cost of an isogeny evaluation lies in computing its kernel polynomial, which involves scalar multiplications by different prime factors and would otherwise lead to timing variations. To mitigate this, CTIDH employs *differential addition chains* (DACs). By padding shorter chains with dummy operations, all scalar multiplications are forced to cost the same, analogous to the Matryoshka approach used for isogeny evaluations.

Finally, performance improvements arise from assigning bounds m_i to entire batches rather than to individual primes. This yields a larger combinatorial key space:

$$\#K_{N,M} = \prod_{i=1}^{B} \Phi(N_i, m_i), \quad \Phi(x,y) = \sum_{k=0}^{\min\{x,y\}} \binom{x}{k}\binom{y}{k} 2^k,$$

where $\Phi(x,y)$ counts integer vectors in $\mathbb{Z}^x$ with ℓ_1-norm at most y.

2.4 dCSIDH: Deterministic and Dummy-Free CSIDH

The dCSIDH, or `secsidh`, variant [4] was introduced as a high-security implementation of CSIDH that simultaneously achieves *determinism* and *dummy-freeness*. Unlike CTIDH, which relies on dummy operations to enforce constant-time behavior, dCSIDH eliminates both randomness and dummy padding by restricting the key space to exponents $e_i \in \{-1, 1\}$. This restriction ensures that every isogeny degree is used exactly once in a fixed direction, providing determinism in both point sampling and isogeny evaluation.

From a security perspective, determinism provides a stronger defense against fault attacks. However, from a performance standpoint, this comes at a significant cost: eliminating dummies removes batching flexibility, and determinism requires larger parameter sizes (typically starting at 2048-bit primes) to maintain security. As a result, benchmarks show that dCSIDH runs approximately 3 to 5 times slower than probabilistic CTIDH at equivalent parameter sizes.

2.5 dCTIDH: Deterministic CTIDH

While CSIDH offers an elegant algebraic structure and promising post-quantum security, its reference design is vulnerable to practical implementation issues most notably timing and fault attacks. To address these challenges, dCTIDH [5] was introduced as a refinement of CTIDH, enhancing the original protocol with deterministic evaluation, stronger side-channel resistance, and improved performance.

The dCTIDH scheme is a deterministic variant of CTIDH that resolves the reliance on probabilistic point sampling and non-deterministic isogeny evaluation. Its key innovation is the introduction of *Widely Overlapping Meta-Batches* (WOMBats), which combine two complementary batching ideas: *multiple isogenies per batch* and *overlapping batches*.

In the original CTIDH, exactly one isogeny is computed per batch in order to avoid secret-dependent behavior. This constraint limits efficiency, since even if the secret key requires several isogenies from the same batch, only one can be evaluated. In contrast, if we restrict secret exponents to unitary values $e_i \in \{-1, 1\}$, then multiple isogenies of distinct degrees can be computed safely within a single batch. For a batch $\mathcal{B}_i = \{\ell_{i,1}, \ldots, \ell_{i,N_i}\}$, we can choose any number $M_i \leq N_i$ of distinct degrees, evaluating M_i isogenies via M_i calls to Matryoshka$[\ell_{i,1}, \ell_{i,N_i}]$.

This significantly reduces the number of total isogenies needed, as the key space grows combinatorially:

$$\Psi(N_i, M_i) = \binom{N_i}{M_i} \cdot 2^{M_i} \quad \text{or} \quad \Psi_{\text{dummy}}(N_i, M_i) = \sum_{j=0}^{M_i} \binom{N_i}{j} \cdot 2^j$$

if dummy isogenies are allowed (i.e. $e_i \in \{-1, 0, 1\}$).

Another approach to enlarge the key space and improve efficiency is to use batches that overlap in some of their prime factors. Suppose the first batch is

$\mathcal{B}_1 = \{\ell_1, \dots, \ell_{N_1}\}$. Instead of defining $\mathcal{B}_2 = \{\ell_{N_1+1}, \dots, \ell_{N_1+N_2}\}$, we let the batches share $\omega_{1,2}$ primes:

$$\mathcal{B}_2 = \{\ell_{N_1-\omega_{1,2}+1}, \dots, \ell_{N_1+N_2-\omega_{1,2}}\}.$$

This overlapping structure amplifies the combinatorial growth of the key space without requiring a proportional increase in the number of isogeny evaluations. To ensure determinism, the bounds M_1, M_2 must satisfy $M_1 + M_2 \leq N_1 + N_2 - \omega_{1,2}$, preventing multiple isogenies from being applied to the same degree.

dCTIDH combines the two techniques above into WOMBats. A WOMBat $\mathcal{W} = \{\ell_{i,1}, \dots, \ell_{i,N}\}$ with bound M is evaluated as M overlapping batches

$$\mathcal{B}_1 = \{\ell_1, \dots, \ell_{N-M+1}\}, \quad \mathcal{B}_2 = \{\ell_2, \dots, \ell_{N-M+2}\}, \quad \dots \quad \mathcal{B}_M = \{\ell_M, \dots, \ell_N\}.$$

Each $\mathcal{B}_j$ overlaps in $N - M$ primes with its neighbors, and exactly one isogeny is computed from each, realized as a Matryoshka isogeny Matryoshka$[\ell_j, \ell_{N-M+j}]$. In this way, the WOMBat structure deterministically covers all possible distributions of M distinct isogeny degrees, while guaranteeing constant computational cost. The resulting key space of $N_{\mathcal{W}}$ disjoint WOMBats is

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

To mitigate timing leakage, dCTIDH employs *DACs*, which pad shorter chains with dummy steps within each WOMBat to achieve constant-time scalar multiplication as previous mentioned. Consequently, even though dCTIDH eliminates *randomness* during evaluation, *the DAC and Matryoshka computations still incorporate dummy steps to maintain constant-time execution.*

2.6 Techniques in CSIDH-like Schemes

Efficient implementations of CSIDH and its variants rely on specialized techniques that simultaneously ensure constant-time execution and improve the performance of scalar multiplications and isogeny evaluations. Among the most important are *Differential Addition Chains (DACs)*, which realize scalar multiplications in constant time using only x-coordinates, and *Matryoshka isogenies*, which enable constant-time evaluation of isogenies while reducing computational cost through the exploitation of nested structures within isogeny chains.

Differential Addition Chains (DACs). Differential Addition Chains (DACs) are algorithmic frameworks for scalar multiplication on elliptic curves, particularly in the Montgomery model, where only x-coordinates are used. By avoiding full group operations and secret-dependent branching, DACs enable constant-time and side-channel-resistant implementations—an essential feature in isogeny-based cryptography where points are ephemeral and curves evolve along isogeny walks.

Definition 1 (Differential Addition Chain). *A* differential addition chain *for an integer n is a sequence $1 = c_0, c_1, \ldots, c_r = n$ such that for each $i \in \{1, \ldots, r\}$ there exist indices $j, k < i$ with*

$$c_i = c_j + c_k, \textit{and } c_j - c_k \in \{0, c_0, c_1, \ldots, c_{i-1}\}.$$

In other words, each new sum in the chain must correspond to a difference already present in the chain (or zero).

Example 1. A differential addition chain for 29 is 1, 2, 3, 5, 8, 13, 21, 29, since, for instance, $13 = 8 + 5$ with difference $8 - 5 = 3 \in \{1, 2, 3, 5, 8\}$.

In this work we focus on the subclass of *continued-fraction DACs*, which admit a compact bitstring encoding. For simplicity, we use the terms *DAC* and *continued-fraction DAC* interchangeably throughout.

Definition 2 (Continued-fraction DAC). *Let $(a_2, b_2, c_2), \ldots, (a_r, b_r, c_r)$ be a sequence of triples with $n \geq 3$, $(a_2, b_2, c_2) = (1, 2, 3)$, $c_r = n$, and for each $i \geq 3$:*

$$(a_i, b_i, c_i) = \begin{cases} (b_{i-1}, c_{i-1}, c_{i-1} + b_{i-1}), & \textit{if } f_i = 0, \\ (a_{i-1}, c_{i-1}, c_{i-1} + a_{i-1}), & \textit{if } f_i = 1, \end{cases}$$

with $c_i = a_i + b_i$. Then the continued-fraction DAC is the sequence $1, 2, c_2, \ldots, c_r = n$.

Example 2. A continued-fraction DAC for 13 admits the compressed bitstring $f = 11110$.

In the Montgomery model, scalar multiplication $[k]P$ can be realized by iterating only differential operations:

$$\texttt{DIFF\_ADD}(P, Q, P - Q) \quad \text{and} \quad \texttt{xDBL}(P),$$

while tracking the differential $P - Q$. This makes the procedure fully deterministic and constant-time. Algorithm 1 illustrates how scalar multiplication by n can be carried out using a compressed DAC bitstring, relying solely on the two fundamental operations `xDBL` and `DIFF_ADD`.

Note that two DACs corresponding to different integers incur the same computational cost whenever their compressed representations have the same length, regardless of the integers themselves.

Remark 2. Compared to the classic Montgomery ladder (which is also constant-time), continued-fraction DACs compress structured additions/doublings for fixed small ℓ and integrate more naturally with batch scheduling, which is why CTIDH/dCTIDH prefer DACs for kernel generation.

In CSIDH-like protocols, DACs are used to compute kernel generators for isogenies. Secret keys are exponent vectors $(e_1, \ldots, e_n)$ indicating how many times an isogeny of a specific degree is applied. Each scalar multiplication $[\ell_i]P_i$

Algorithm 1: `DAC` — Scalar multiplication via a compressed DAC

```
Input: Point $P$, compressed DAC $f_3, \dots, f_r$ for $n$
Output: $[n]P$
 1: $X_0 \leftarrow P$
 2: $X_1 \leftarrow$ xDBL($P$)
 3: $X_2 \leftarrow$ DIFF_ADD($P, P, X_1$)
 4: for $i = 3$ to $r$ do
 5:     if $f_i = 0$ then
 6:         $(X_0, X_1, X_2) \leftarrow (X_1, X_2,$ DIFF_ADD$(X_0, X_1, X_2))$
 7:     else
 8:         $(X_0, X_1, X_2) \leftarrow (X_0, X_2,$ DIFF_ADD$(X_1, X_0, X_2))$
 9:     end if
10: end for
11: return $X_2$
```

(for small primes ℓ_i) is performed using a fixed DAC, ensuring constant-time execution.

In CTIDH and dCTIDH, DACs are precomputed according to the allowed exponent bounds, and scalar multiplications are often *batched* to reuse intermediate results. However, the length of these DACs—and therefore the computational cost of a multiplication by ℓ_i—depends directly on ℓ_i. To keep the isogeny degree ℓ_i secret, CTIDH enforces constant-time multiplications for all factors within a batch B. This is done by precomputing an optimal DAC for each $\ell_i \in B$ and padding it with dummy steps if necessary, so that multiplication by any cofactor from B requires the same number of operations as the largest ℓ_i in the batch.

However, since dummy padding is normally applied to maintain constant-time execution, it can leave room for active attacks, such as fault injection. The dCTIDH scheme addresses this issue with *DACsHUND*, a technique for dummy-free DAC evaluation, which we explore later in this work.

Matryoshka isogenies. As previously mentioned, the computational cost of evaluating isogenies via Vélu's formulas or $\sqrt{\text{é}}$lu grows respectively as $\widetilde{O}(\ell)$ and $\widetilde{O}(\sqrt{\ell})$ in the isogeny degree ℓ. Since in CSIDH-like protocols one must evaluate isogenies of different prime degrees, these costs naturally vary across primes, potentially leaking information and complicating optimization. Additionally, when primes are grouped in batches, such as in CTIDH and dCTIDH, isogeny evaluations must also cost the same within each batch. To address this, CTIDH introduces the notion of *Matryoshka isogenies*, a technique that enforces uniform evaluation cost across a batch of primes.

The core idea is to impose a "nested" evaluation structure on the kernel polynomial

$$h_S(X) = \prod_{s \in S} \big(X - x([s]P)\big), \tag{2}$$

where $S = \{1, 2, \dots, (\ell - 1)/2\}$ in the Vélu case, or $S = \{1, 3, 5, \dots, \ell - 2\}$ in the $\sqrt{\text{é}}$lu case. For Vélu's method, this amounts to cycling through the multiples

$[s]P$, generating and evaluating $h_S(X)$ on the fly. Once the loop reaches $(\ell-1)/2$, one can continue appending *dummy iterations*, thereby aligning the total number of operations to that required by the largest prime $\ell_{\max}$ in the batch. In this way, any ℓ-isogeny in the batch can be evaluated at the uniform cost $\widetilde{O}(\ell_{\max})$.

The same concept extends to $\sqrt{\text{é}}$lu evaluations. In this case, the index set $S \longleftrightarrow (U \times V) \cup W$ is split into a box $U \times V$ and a leftover set W. Then, $h_S(X)$ can be computed by multiplying $h_W(X)$ with the resultant of $h_U(X)$ and a polynomial derived from V, with all sets U, V, and W having size $\widetilde{O}(\sqrt{\ell})$.

To apply a Matryoshka structure, U and V are chosen according to the smallest degree in the batch, while W is padded according to the largest. This ensures a uniform evaluation cost across primes in the batch, albeit with some efficiency loss since the parameters U, V, W are no longer optimally tuned for each ℓ.

We denote by Matryoshka$[\ell_i, \ell_j]$ a computation that performs any isogeny of degree $\ell \in [\ell_i, \ell_j]$ at the cost of ℓ_j, whether using Vélu or $\sqrt{\text{é}}$lu as appropriate. This nested framework makes it possible to batch isogeny evaluations without leaking degree information, while still achieving sublinear performance when $\sqrt{\text{é}}$lu is applicable. For further details, see [1, 3].

3 DACsHUND

In dCTIDH, key generation requires computing a sequence of scalar multiplications $[\ell_i]P$ for a fixed set of primes $\ell_1, \ldots, \ell_n$. These multiplications are carried out on Montgomery curves using x-only arithmetic (`xADD`, `xDBL`) and differential addition chains (DACs). Implementations must be side-channel resistant, deterministic, and ideally batched to maximize performance.

In constant-time settings, the minimal DAC for each prime generally has a different length. To equalize the execution flow, previous approaches required padding shorter DACs with dummy operations so that all scalar multiplications within a batch completed in the same number of steps. While effective, this introduces redundancy and increases susceptibility to certain advanced fault-injection attacks. To overcome this limitation, we introduce ***DACsHUND*** (Differential Addition Chain Having Unnecessities Needed for Dummy-freeness), originally proposed in the future work of the dCTIDH paper, which enables dummy-free DAC execution.

Definition 3 (`DACsHUND`). *Let $\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ be a family of n batches, where each batch $\mathcal{B}_i$ consists of N_i primes: $\mathcal{B}_i = \{\ell_{1,i}, \ldots, \ell_{N_i,i}\}$ with $\ell_{1,i} \leq \cdots \leq \ell_{N_i,i}$. Each prime $\ell_{j,i}$ has an associated set $\mathcal{D}_{j,i}$ of admissible DAC lengths. The configuration $\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ is a valid* `DACsHUND` *if, for every batch $\mathcal{B}_i$, the intersection $\bigcap_{j=1}^{N_i} \mathcal{D}_{j,i}$ is non-empty.*

Intuitively, the idea is to partition the primes into batches such that all DACs in a batch share at least one common length. This eliminates the need for dummy padding while preserving constant-time execution. Algorithm 2 formalizes the batch validation procedure. This general framework not only supports dCTIDH, but can also be applied to related protocols such as CTIDH.

Example 3. Consider a batch $\mathcal{B}_1 = \{11, 13, 17, 19\}$. The corresponding DAC sets are:

$$\begin{aligned}\mathcal{D}_{1,1} &= \{3, 4, 8\},\\ \mathcal{D}_{2,1} &= \{3, 4, 5, 10\},\\ \mathcal{D}_{3,1} &= \{4, 5, 7, 14\},\\ \mathcal{D}_{4,1} &= \{4, 5, 6, 8, 16\}.\end{aligned}$$

Since their intersection is $\{4\}$, this is a valid *DACsHUND* configuration. However, if prime 5 is added, its DAC set $\{1, 2\}$ leads to an empty intersection, invalidating the batch.

Algorithm 2: `IsValidDACsHUND` — Validation of DACsHUND Compatibility

```
Input: Batch sizes $N = (N_1, \ldots, N_B)$, number of batches $B$, prime list $\mathcal{P}$
Output: True if valid; False otherwise
1: Partition $\mathcal{P}$ into batches $\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(B)}$ of sizes $N_1, \ldots, N_B$
2: for $i = 1$ to $B$ do
3:     $I \leftarrow \bigcap_{p \in \mathcal{P}^{(i)}}$ DACsHUND[$p$]
4:     if $I = \emptyset$ then
5:         return False
6:     end if
7: end for
8: return True
```

DACsHUND Map. The first step in building a DACsHUND configuration is to enumerate all admissible DACs for each prime in the range of interest. Instead of storing only the shortest DAC, we record every possible DAC length and its corresponding representation. This yields a map DACsHUND associating each prime p with its set of DAC lengths. For example, DACsHUND$[13] = \{3, 4, 5, 10\}$.

We adopt a straightforward *brute-force* strategy: enumerating all possible compressed DAC representations up to a prescribed length (e.g., 16), and testing each candidate to verify whether it corresponds to a valid prime. Although this approach does not exploit optimized DAC search methods [2], the search space remains sufficiently small that an exhaustive traversal can be completed in a small time frame.

3.1 Searching Batch Configurations

With DACsHUND in place, the next step is to search for valid batch configurations. The dCTIDH batch search builds on the greedy strategy of CTIDH and is defined by three parameters: the number of batches B, the batch size vector $N = (N_1, \ldots, N_B)$ specifying the number of primes per batch, and the bound vector $M = (M_1, \ldots, M_B)$ that ensures the resulting configuration spans a sufficiently large key space.

Initialization. The standard dCTIDH greedy initialization assigns equal size to all batches ($N_i = n/B$ with $\sum N_i = n$), but this often produces invalid `DACsHUND` configurations with empty intersections. To address this, we construct an initial configuration iteratively: starting with $N = (1, \ldots, 1)$, we cycle through the batches, incrementing one N_i at a time, and accept the update only if the resulting configuration is `DACsHUND`-valid. This continues until all primes are allocated. The procedure is shown in Algorithm 3.

Algorithm 3: `FindInitialBatchSizes` — Search for Valid Initial Configurations

Input: Number of batches B, prime list $\mathcal{P}$
Output: Batch size tuple N if valid; `None` otherwise

```
1:  Initialize $N \leftarrow (1, \ldots, 1) \in \mathbb{Z}^B$
2:  while $\sum_{i=1}^{B} N_i < |\mathcal{P}|$ do
3:      $\Delta \leftarrow$ False
4:      for $i = 1$ to $B$ do
5:          Let $N' \leftarrow N$ with $N'_i \leftarrow N_i + 1$
6:          if IsValidDACsHUND$(N', B, \mathcal{P})$ then
7:              $N \leftarrow N'$, $\Delta \leftarrow$ True
8:          end if
9:      end for
10:     if $\Delta =$ False then
11:         return None
12:     end if
13: end while
14: return $N$
```

Greedy search. The greedy algorithm modifies a configuration by decreasing the size N_i of one batch B_i and increasing the size of another $B_j \neq B_i$. This is repeated while exploring feasible bounds M_i for each batch. To integrate `DACsHUND`, we introduce a validation step at each modification to ensure that the new batch configuration preserves non-empty DAC intersections. If multiple DAC lengths are available, the smallest one is selected to minimize scalar multiplication cost. The cost function is thus adapted to consider the shortest valid DAC from the intersection of each batch.

Remark 3. Small primes such as 3, 5, and 7 have very restricted DAC sizes (e.g., $\mathcal{D}_3 = \{0\}$). Their inclusion can yield inefficient configurations under `DACsHUND` constraints. For this reason, we also explore configurations excluding these primes and substituting them with larger ones to assess the performance trade-offs.

4 Dummy-Free Matryoshka

As outlined in §2.6, both CTIDH and dCTIDH employ the Matryoshka structure to conceal the true degree of an isogeny within a batch. In this setting, an isogeny of degree ℓ_k contained in a batch (ℓ_l, ℓ_r) is evaluated at the uniform cost of an ℓ_r-isogeny. The classical construction proceeds as follows. One first computes the sequence of points

$$P, [2]P, \ldots, \left[\tfrac{\ell_r-1}{2}\right]P,$$

and from these builds the kernel polynomial. The polynomial is factored into two parts: the *real factors*,

$$\prod_{i=0}^{(\ell_k-1)/2} \big(x - x([i]P)\big),$$

which correspond to the actual ℓ_k-isogeny, and the *dummy factors*,

$$\prod_{i=(\ell_k-1)/2+1}^{(\ell_r-1)/2} \big(x - x([i]P)\big),$$

which pad the cost up to ℓ_r and thereby hide the true degree ℓ_k.

This dummy-based approach introduces two distinct entry points for fault-injection attacks. First, the dummy multiplications in the kernel polynomial may be distinguishable from real ones, enabling targeted faults. Second, the unused multiples

$$\left[\tfrac{\ell_k-1}{2}+1\right]P, \ldots, \left[\tfrac{\ell_r-1}{2}\right]P,$$

although computed, are never required by the true kernel and thus create additional leakage channels.

To address these vulnerabilities, [5] introduced a modified Matryoshka structure. Their refinement eliminates dummy multiplications by reformulating the kernel product so that redundant terms cancel out algebraically, rather than being introduced explicitly. Furthermore, the unused multiples are validated against their expected relations, preventing an adversary from exploiting them as a source of leakage. This restructuring preserves the constant-time nature of Matryoshka while *significantly reducing its exposure to fault attacks.*

4.1 Matryoshka 2.0

The idea described in [5, Appendix A], eliminates dummy operations entirely while retaining the same cost profile. The key observation is that for any point P, we have $x([i]P) = x([\ell - i]P)$. This symmetry allows the algorithm to verify that every multiple's x-coordinate must be computed correctly, since each will appear twice.

Algorithm 4 shows the full computation of the kernel polynomial h using the dummy-free Matryoshka approach, as described in [5]. Instead of inserting dummy factors, Matryoshka 2.0 replaces them with real multiplications of a modified form:

$$x = \tfrac{1}{2}x([i]P) - \alpha \cdot \tfrac{1}{2}x([i]P),$$

where α is chosen in constant time to be -1 if the value $x([i]P)$ has already appeared for some $j < i$, and 1 otherwise. Thus, lines 12 to 14 carry the same information as checking whether $i > \frac{\ell_k - 1}{2}$ to determine if a dummy operation needs to be computed in the original version.

This achieves two crucial properties: uniformity of computation, since every iteration performs a real multiplication of the same cost, leaving no distinction between *real* and *dummy* steps; and the absence of unused data, since all multiples $x([i]P)$ are incorporated into the product, eliminating the risk of computing unnecessary points.

Algorithm 4: Matryoshka 2.0 (based on [5])

Input: A degree ℓ_k, a batch $[\ell_l, \dots \ell_r]$ and a point P such that $\ell_k \cdot P = \mathcal{O}$
Output: The kernel polynomial $h(x)$ for $\phi : E \to E/\langle P \rangle$

```
 1: $b_k \leftarrow \frac{\ell_k - 1}{2}, b_l \leftarrow \frac{\ell_l - 1}{2}, b_r \leftarrow \frac{\ell_r - 1}{2}$
 2: $t \leftarrow b_r - b_l$
 3: Compute ($x$-coordinates of) $\{P, [2]P, \dots, [b_r]P\}$.
 4: $h(x) \leftarrow 1$

 5: for $i \in [1, \dots, b_l]$ do                          ▷ compute the linear part up to $b_l$
 6:     $m \leftarrow x([i]P)$
 7:     $h(x) \leftarrow h(x) \cdot (x - m)$
 8: end for

 9: for $i \in [b_l + 1, \dots, b_r]$ do
10:     $m \leftarrow \frac{1}{2}x([i]P)$
11:     $\alpha \leftarrow 1$
12:     for $j \in [(b_l + 1 - t), \dots, (i - 1)\}$ do      ▷ checks if $x[iP]$ has appeared already
13:         $\alpha \leftarrow \alpha \cdot$ cCompare($x([i]P)$, $x([j]P)$)          ▷ returns -1 if so
14:     end for
15:     $h_1(x) \leftarrow h(x) \cdot (x - m)$
16:     $h_2(x) \leftarrow h(x) \cdot \alpha \cdot m$
17:     $h(x) \leftarrow h_1(x) - h_2(x)$
18: end for

19: return $h(x) \leftarrow x^{b_k - b_r} \cdot h(x)$                        ▷ fix the degree
```

The original Matryoshka implementation in CTIDH (and dCTIDH) uses projective space to represent x-only points as $(X : Z)$, thereby avoiding costly inversions. As in Vélu's formulas, the kernel polynomial must be evaluated at $\frac{h(1)}{h(-1)}$ to compute the codomain coefficient A'. In the projective setting, the evaluations at 1 and -1 are directly integrated into the implementation.

To adapt Algorithm 4 to projective space, we replace the affine expression $m = \frac{1}{2}x([i]P)$ with its projective equivalent. Writing $[i]P = (X_i : Z_i)$, we obtain

$$\frac{m_x}{m_z} = \frac{X_i}{2 \cdot Z_i}.$$

Accordingly, we updated in lines 15–17 with the following

$$\frac{h_x}{h_z} = \frac{h_x \cdot \big((\alpha \cdot m_x) + m_x - m_z\big)}{h_z \cdot \big((\alpha \cdot m_x) + m_x + m_z\big)}.$$

The `cCompare` routine must also be modified to compare projective points, increasing its cost to $2\mathbf{M}$. Finally, the degree correction step in line 19 simplifies to a constant-time sign flip of h_z.

Igonoring additions, the computation of one $\texttt{Matryoshka}_{[\ell_l,\ell_r]}$-isogeny is thereby increased by $\sum_{i=1}^{t}(t-1+i)\cdot 2\mathbf{M}$, with $t = ((\ell_r - 1)/2) - ((\ell_l - 1)/2)$, compared to the dummy based version.

4.2 Matryoshka 1.414 ($\sqrt{}$élu)

For the Matryoshka[5] variant using $\sqrt{}$élu, *Algorithm 4 cannot be applied directly*, since not all multiples $[i]K$ required for comparison are available due to the index system that splits the computation into $U \times V \cup W$. However, we can exploit the structure of Matryoshka-$\sqrt{}$élu: the $U \times V$ component covers the kernel polynomial only up to ℓ_l, so all dummy factors necessarily appear in W. Moreover, W consists solely of even multiples of P. This enables us to validate each x-coordinate of the multiples $[2]P, [4]P, \ldots, [\frac{\ell_r - 1}{2}]P$ by checking whether they match the double of their corresponding halves, that is, by verifying $xDBL(x([i]P)) = x([2i]P)$.

Depending on the batch size and the velusqrt parameters, in some cases, not all odd halves are generated within $U \times V$. Therefore, the odd points must be computed explicitly in the range

$$max\big(bs, (\frac{(\ell_r - 1)}{2} - 2 \cdot bs \cdot gs)/2\big),$$

where (bs, gs) denote the baby-step/giant-step parameters of $\sqrt{}$élufor the ℓ_l-isogeny. This ensures that every even multiple in W pairs with its half, allowing for consistent validation without dummy points. Algorithm 5 summarizes the resulting *dummy-free Matryoshka algorithm* adapted to $\sqrt{}$élu.

As a result of the `xDBL` trick, the overhead of projective Matryoshka 1.414 is just $2 \cdot \mathbf{M} + xDAC$ for iteration, together with the $((br - 2 * bs * gs)/2) - bs$ additionial `xADD` to compute the missing point halves.

[5] We called Matryoshka 1.414 since $\sqrt{2} \approx 1.414$.

Algorithm 5: Matryoshka 1.414

Input: A degree ℓ_k, a batch $[\ell_l, \dots \ell_r]$, a point P such that $\ell_k \cdot P = \mathcal{O}$ and $\sqrt{\text{é}}$lu parameters (bs, gs) for ℓ_l

Output: The kernel polynomial $h(x)$ for $\phi : E \to E/\langle P\rangle$

```
1: $b_k \leftarrow \frac{\ell_k - 1}{2}, b_l \leftarrow \frac{\ell_l - 1}{2}, b_r \leftarrow \frac{\ell_r - 1}{2}$
2: $t \leftarrow b_r - b_l$
3: Compute multiples according to $\sqrt{\text{é}}$lu
4: Compute odd multiples $[bs+2]P, \dots, [(br - 2*bs*gs)/2]P$ if $bs < (br - 2*bs*gs)/2$
5: $h(x) \leftarrow 1$

6: Compute $\sqrt{\text{é}}$lu using $(bs, gs)$

7: **for** $i \in [0, \dots, b_r - 2*bs*gs]$ **do**
8:     $m \leftarrow \frac{1}{2}x([2*i+2]P)$
9:     $\alpha \leftarrow 1$ if $i \leq b_k - 2*bs*gs$ else $-1$
10:    $\alpha \leftarrow \alpha\cdot$ -cCompare(xDBL($x([i+1]P)$), $x([2*i+2]P)$)  ▷ -1 if points are equal, else 1.
11:    $h_1(x) \leftarrow h(x) \cdot (x - m)$
12:    $h_2(x) \leftarrow h(x) \cdot \alpha \cdot m$
13:    $h(x) \leftarrow h_1(x) - h_2(x)$  ▷ $h$ is multiplied by $x$ when $\alpha = -1$
14: **end for**

15: **return** $h(x) \leftarrow x^{b_k - b_r} \cdot h(x)$
```

5 Implementation

We base our implementation on the dCTIDH code from https://github.com/PaZeZeVaAt/dCTIDH, which in turn builds on the secsidh[6] implementation [4]. This code incorporates the optimal strategies introduced in [10] to accelerate kernel point computations by balancing the trade-off between pushing points through isogenies and computing kernels via DACs. In addition, it provides assembly-optimized $\mathbb{F}_p$ arithmetic for the different parameter sets.

We extend this implementation by integrating the new DACsHUND parameters for DAC computation and by adapting Algorithm 4 and Algorithm 5 to projective space.

5.1 Batch Configurations

To determine optimal parameter sets for dCTIDH, we build on the configurations reported in the original dCTIDH work. In particular, we focus on the parameter sets dCTIDH-194 and dCTIDH-205, which serve as natural starting points and enable direct comparison with their non–dummy-free dCTIDH counterparts.

[6] Publicly available at https://github.com/kemtls-secsidh/secsidh.

Further analysis shows that the small primes 3, 5, and 7 in the set $\{\ell_i\}$ severely restrict possible batch structures under DACsHUND constraints. To address this, we run our greedy search while excluding either 3, or $3, 5, 7$ from the set $\{\ell_i\}$.

Table 1 presents the results for the dCTIDH-194 and dCTIDH-205 parameter sets. We evaluate configurations with between 12 and 20 batches for each parameter set. A complete run over all batch configurations requires approximately 16 hours using 32 threads on a server equipped with dual AMD EPYC 7643 processors (2.3 GHz, 192 threads in total).

Table 1: Best greedy results for the dCTIDH-194 and dCTIDH-205 parameter sets.

variant	ℓ skipped	batches	isogenies	cost
dCTIDH-205	–	15	70	$327,942$
dCTIDH-194	–	17	75	$334,458$
dCTIDH-205	3	17	73	$327,390$
dCTIDH-194	3	14	73	$332,920$
dCTIDH-205	$3, 5, 7$	13	70	$334,846$
dCTIDH-194	$3, 5, 7$	13	72	$341,526$

While the performance differences remain within $\approx 5\%$, our results indicate that the best configuration comes from skipping only the prime 3. Therefore, we implement dummy-free dCTIDH for the parameter sets dCTIDH-205 and dCTIDH-194 by excluding the 3-isogeny.

Remark 4. The greedy search only optimizes the plain cost of isogeny evaluation using optimal strategies. Therefore, it does not account for additional, albeit constant, costs in the group action, such as cofactor removal, and a final inversion to return an affine codomain, are not accounted for, explaining the differences. to the benchmarks measured in Table 2.

5.2 Performance

All benchmarks were performed on an Intel Core i7-6700 (Skylake) processor, running Debian 12 with Hyper-Threading and Turbo Boost disabled, and compiled using gcc-12.2.0.

Table 2 compares the results against dCSIDH as only other constant-time, dummy-free and deterministic scheme, CTIDH (from the secsidh implementation), as well as the relevant dCTIDH parameter sets.

Table 2 compares the cost of the group action across different CSIDH implementations. As expected, dCSIDH is by far the most expensive: its fully deterministic and dummy-free design results in more than 1.5 million field multiplications

Table 2: Results of a group action evaluation in multiplications (**M**), squarings (**S**), and additions (**a**), and median cycle count (Gcyc) of 10,000 experiments, performed on a Skylake CPU.

variant	**M**	**S**	**a**	$\mathbb{F}_p$-mult.	Gcyc
CTIDH-2048	$287,207 \pm 21\%$	$83,759 \pm 9\%$	–	$370,966 \pm 17\%$	$1.652 \pm 17\%$
dCSIDH-2048 [4]	$1,315,203$	$227,501$	–	$1,542,704$	7.039
dCTIDH-2048-205 [5]	$263,545$	$50,825$	$465,224$	$314,370$	1.418
dCTIDH-2048-194 [5]	$266,101$	$51,258$	$469,258$	$317,359$	1.410
This work (205)	$303,058$	$54,074$	$560,276$	$357,132$	1.600
This work (194)	$307,004$	$55,215$	$553,193$	$362,219$	1.595

and a median cost of 7.0 Gigacycles, making it impractical in comparison with other approaches.

Both parameter sets of dCTIDH (194 and 205) are more efficient, requiring about 314–317k $\mathbb{F}_p$ multiplications and completing a group action in roughly 1.410–1.418 Gigacycles. This confirms that batching and WOMBats provide a strong efficiency, albeit at the cost of dummy operations.

Our dummy-free implementation adds a small overhead compared to dC-TIDH: 358–362k $\mathbb{F}_p$-multiplications and 1.595–1.600 Gigacycles. This represents a slowdown of only 12–14%, while completely eliminating dummy multiplications in both DACs and Matryoshka isogenies (when we compare with dCTIDH). At the same time, we still outperform the original CTIDH by about 4%, demonstrating the advantages of the WOMBat keyspace, even under the additional DACsHUND constraints.

Remark 5. Similar to dCTIDH, this work focuses solely on optimizing the group action, which is just one part of a full key exchange. During key generation, One also needs to compute a torsion point of order $\prod \ell_i$, and in the key derivation step, the order of this point must be validated (which also ensures supersingularity). However, excluding the degree 3 speeds up the point search and validation by up to 20% compared to the dCTIDH. Recent work by Pope, Reijnders, Robert, Sferlazza, and Smith [15] used a pairing-based approach for validation, suggesting a possible fourfold speedup. We leave the integration of pairing-based validation and point search into the dCTIDH-framework as future work.

6 Conclusion

We have presented the first **dummy-free implementation of dCTIDH**, combining DACsHUND with *dummy-free Matryoshka isogenies*. Our approach eliminates all dummy operations in both differential addition chains and isogeny evaluations, providing the first dCTIDH implementation that is deterministic, constant-time, and fully dummy-free. We showed how to adapt the greedy parameter search to incorporate these constraints, and identified viable parameter sets for dCTIDH-194 and dCTIDH-205, noting that very small primes such as $3, 5, 7$ are incompatible with DACsHUND.

In our implementation, we report results in Table 2 using the new batching strategy and the Matryoshka 1.414 variant. We show that even without dummy isogenies, our performance remains close to that of dCTIDH. Moreover, we demonstrate an improvement of roughly 4% over CTIDH for both our implementations of dCTIDH-2048-194 and dCTIDH-2048-205.

Acknowledgements

The authors would like to thank Fabio Campos, Krijn Reijnders, and Michael Meyer for their discussions and insights on Matryoshka isogenies.

References

1. Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: faster constant-time CSIDH. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):351–387, 2021.
2. Daniel J. Bernstein, Jolijn Cottaar, and Tanja Lange. Searching for differential addition chains. *IACR Cryptol. ePrint Arch.*, page 1044, 2024.
3. Daniel J Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *Open Book Series*, 4(1):39–55, 2020.
4. 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. *IACR Commun. Cryptol.*, 1(1):5, 2024.
5. Fabio Campos, Andreas Hellenbrand, Michael Meyer, and Krijn Reijnders. dCTIDH: Fast & deterministic CTIDH. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(3):516–541, 2025.
6. 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 *17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020*, pages 57–65. IEEE, 2020.
7. Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *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*, volume 14008 of *Lecture Notes in Computer Science*, pages 423–447. Springer, 2023.
8. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In Thomas Peyrin and Steven D. Galbraith, editors, *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*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.

9. Jesús-Javier Chi-Domínguez, Eduardo Ochoa-Jiménez, and Ricardo Neftali Pontaza Rodas. Let us walk on the 3-isogeny graph: efficient, fast, and simple. *IACR Cryptol. ePrint Arch.*, page 691, 2025.
10. Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. Optimal strategies for CSIDH. *Adv. Math. Commun.*, 16(2):383–411, 2022.
11. Tinghung Chiu, Jason LeGrow, and Wenjie Xiong. Practical fault injection attacks on constant time CSIDH and mitigation techniques. In Chip-Hong Chang, Ulrich Rührmair, Jakub Szefer, Lejla Batina, and Francesco Regazzoni, editors, *Proceedings of the 2024 Workshop on Attacks and Solutions in Hardware Security, ASHES 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 11–22. ACM, 2024.
12. Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *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*, volume 14008 of *Lecture Notes in Computer Science*, pages 448–471. Springer, 2023.
13. Michael Meyer, Fabio Campos, and Steffen Reith. On Lions and Elligators: An efficient constant-time implementation of CSIDH. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 307–325, Cham, 2019. Springer International Publishing.
14. Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. A constant-time algorithm of CSIDH keeping two points. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E103-A(10):1174–1182, October 2020.
15. Giacomo Pope, Krijn Reijnders, Damien Robert, Alessandro Sferlazza, and Benjamin Smith. Simpler and faster pairings from the montgomery ladder. *IACR Commun. Cryptol.*, 2(2):29, 2025.
16. Damien Robert. Breaking SIDH in polynomial time. In Carmit Hazay and Martijn Stam, editors, *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*, volume 14008 of *Lecture Notes in Computer Science*, pages 472–503. Springer, 2023.
17. Joseph H Silverman. *The arithmetic of elliptic curves*. Graduate texts in mathematics. Springer, New York, NY, 2 edition, December 2009.
18. Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes-Rendus de l'Académie des Sciences*, 273:238–241, 1971.