On the Practicality of Post-Quantum TLS Using Large-Parameter CSIDH

Fabio Campos[1,2], Jorge Chavez-Saab[3,4], Jesús-Javier Chi-Domínguez[4], Michael Meyer[5], Krijn Reijnders[2], Francisco Rodríguez-Henríquez[3,4], Peter Schwabe[2,6], and Thom Wiggers[7]

[1] RheinMain University of Applied Sciences, Wiesbaden, Germany
campos@sopmac.de
[2] Radboud University, Nijmegen, The Netherlands
krijn@cs.ru.nl
[3] Departamento de Computación, CINVESTAV-IPN, Mexico
[4] Cryptography Research Center, Technology Innovation Institute, Abu Dhabi, United Arab Emirates
jorge.saab@tii.ae, jesus.dominguez@tii.ae, francisco.rodriguez@tii.ae
[5] University of Regensburg, Germany
michael@random-oracles.org
[6] Max Planck Institute for Security and Privacy, Bochum, Germany
peter@cryptojedi.org
[7] PQShield, Nijmegen, The Netherlands
thom@thomwiggers.nl

Abstract. The isogeny-based scheme CSIDH is considered to be the only efficient post-quantum non-interactive key exchange (NIKE) and poses small bandwidth requirements, thus appearing to be an attractive alternative for classical Diffie–Hellman schemes. A crucial CSIDH design point, still under debate, is its quantum security when using prime fields of 512 to 1024 bits. Most work has focused on prime fields of that size and the practicality of CSIDH with large parameters, 2000 to 9000 bits, has so far not been thoroughly assessed, even though analysis of quantum security suggests these parameter sizes. We fill this gap by providing two CSIDH instantiations: A deterministic and dummy-free instantiation based on SQALE, aiming at high security against physical attacks, and a speed-optimized constant-time instantiation that adapts CTIDH to larger parameter sizes. We provide implementations of both variants, including efficient field arithmetic for fields of such size, and high-level optimizations. Our deterministic and dummy-free version, dCSIDH, is almost twice as fast as SQALE, and, dropping determinism, CTIDH at these parameters is thrice as fast as dCSIDH. We investigate their use in real-world scenarios through benchmarks of TLS

Author list in alphabetical order; see https://www.ams.org/profession/leaders/CultureStatement04.pdf. This work has been supported by Deutsche Forschungsgemeinschaft (DFG, German research Foundation) as part of the Excellence Strategy of the German Federal and State Governments – EXC 2092 CASA - 390781972; and by the European Commission through the ERC Starting Grant 805031 (EPOQUE).
Date: May 30, 2023

using our software. Although our instantiations of CSIDH have smaller communication requirements than post-quantum KEM and signature schemes, both implementations still result in too-large handshake latency (tens of seconds), which hinder further consideration of using CSIDH in practice for conservative parameter set instantiations.

Keywords: post-quantum cryptography, isogenies, CSIDH, TLS

1 Introduction

The commutative isogeny-based key exchange protocol (CSIDH) was proposed by Castryck, Lange, Martindale, Panny, and Renes [22] at Asiacrypt 2018. Although it was proposed too late to be included as a candidate in the NIST post-quantum standardization effort [51], it has since received significant attention from the post-quantum-crypto research community. From a crypto-engineering point of view, this attention can be explained by two unique features of CSIDH: Firstly, with the originally proposed parameters, CSIDH has remarkably small bandwidth requirements. Specifically, CSIDH-512, the parameter set targeting security equivalent to AES-128, needs to transmit only 64 bytes each way, more than a factor of 3 less than `SIKEp434_compressed`, the most bandwidth-efficient competitor submitted to the NIST project. Secondly—and more importantly—CSIDH is so far the only realistic option for post-quantum *non-interactive key exchange*. This means that it could in principle be used as a post-quantum drop-in replacement for Diffie–Hellman key exchange.

Unfortunately, quite soon after CSIDH was proposed, several security analyses called into question the claimed concrete security against quantum attacks achieved by the proposed parameters [12, 24, 53]. The gist of these analyses seems troublesome; for example, [53] states that *"the cost of CSIDH-512 key recovery is only about* 2^{16} *quantum evaluations using* 2^{40} *bits of quantumly accessible classical memory (plus relatively small other resources)"*. On the other hand, [12] claims a cost of 2^{19} quantum evaluations for attacking the same instance, but proposes a quantum circuit requiring only $2^{52.6}$ T-gates per evaluation, which means the security is still insufficient. Upon exploring the quantum cost of attacking larger instances but ignoring the cost per CSIDH quantum evaluation, [24] found that instances ranging from 2048 to 4096 bits may be necessary to achieve the security level originally claimed by CSIDH-512.

Interestingly, although some of these concerns were raised as early as May 2018 (i.e., at a time when [22] was available only as preprint), most research on efficient implementations [5, 23, 48, 52], side-channel attacks [17], and fault attacks against CSIDH [7, 16, 44] continued to work with the original parameters. This can probably partly be explained by the fact that the software implementation referenced in [22][8] implements only the smaller two of the three original parameter sets, i.e., CSIDH-512 and CSIDH-1024. However, another reason is that the concerns about the quantum security of CSIDH were (and to some extent still

[8] Available from `https://yx7.cc/code/csidh/csidh-latest.tar.xz`

are) subject of debate. Most notably, Bernstein, Lange, Martindale, and Panny [9] point out that one issue with quantum attacks against CSIDH is the rather steep cost of implementing CSIDH on a quantum computer in the first place. Their analysis of quantum attacks against CSIDH-512 concludes that *"the cost of each query pushes the total attack cost above* 2^{80}*"*.

In this paper we do not take any position in this ongoing debate, but rather set out to answer the question of what it means for CSIDH performance and applicability if we choose more conservative parameters to resist quantum attacks in the cost models used by [12, 24, 53].

Contributions of this paper. On a high level, the contributions of this paper are two-fold: First, we present optimized software for CSIDH at high(er) security levels; second, we benchmark CSIDH as a Diffie–Hellman replacement in cryptographic protocols. The software we present here starts from the analysis and parameter sizes proposed in [24] to reach NIST security levels 1 (equivalent AES-128) and 3 (equivalent AES-192) under different assumptions about the efficiency of quantum attacks. However, our results go much further with regard to optimizing parameters and implementation techniques than [24]:

1. Efficient CSIDH instantiations: We present parameter choices following two different approaches of implementing CSIDH with large parameters:
   - The first approach aims at easy (verification of) side-channel protections, and is based on SQALE [24]. In this approach we eliminate randomness requirements and the use of dummy operations in the CSIDH computation by restricting private keys to be drawn from $\{-1, 1\}^n$. This was proposed by Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith [23], who already mentioned that this approach becomes more natural for larger CSIDH parameters. We refer to this deterministic version of CSIDH as dCSIDH.
   - The second approach optimizes purely for performance and uses the CTIDH batching techniques introduced in [5]. We refer to this variant of CSIDH as CTIDH. In particular, we extend the implementation from [5] to larger parameter sets.
2. Optimizations of dCSIDH and CTIDH: We present a faster key validation approach for large parameters, and discuss how adding a small number of bits to public keys improves the shared-key generation in dCSIDH.
3. Fast finite field arithmetic: All our implementations use curves over prime fields $\mathbb{F}_p$, with $p = f \cdot \prod_{i=1}^{n} \ell_i - 1$, where ℓ_i are distinct odd primes and f is a power of 2. The sizes of p range from 2048 to 9216 bits. We carefully optimize arithmetic in these fields for 64-bit Intel processors, specifically the Skylake microarchitecture. We present implementations of three different options for the underlying field arithmetic, namely,
   (a) an approach based on the GNU Multiple Precision Library (GMP)
   (b) MULX-based multiplier using the schoolbook approach (OpScan)
   (c) MULX-based multiplier using the Karatsuba approach (Karatsuba)

4. Performance evaluation: We combine the contributions in an optimized C/assembly library outperforming the software presented in [24] by a factor up to 2.53× for dCSIDH. Our CSIDH library follows the "constant-time" paradigm for cryptographic implementations and thus protects against timing attacks by avoiding secret-dependent branch conditions and memory indices.
5. Performance in practice: In order to evaluate the performance of our CSIDH software in the context of network protocols, we extend the Rustls library [11] to support OPTLS [38], which uses a NIKE for early authentication. We integrate our CSIDH software as a replacement for Diffie–Hellman and compare the performance of the resulting post-quantum OPTLS to post-quantum KEMTLS as proposed in [58].

Related work. The impact that the proposal of CSIDH has produced in the community can be assessed by the many papers that have been produced around this protocol. Since Castryck, Lange, Martindale, Panny, and Renes [22] left open the problem of implementing CSIDH in constant-time, several papers were published proposing different strategies for achieving this property.

The first constant-time implementation of CSIDH was reported by Bernstein, Lange, Martindale, and Panny [9]. Their analysis focused on assessing the quantum security level provided by CSIDH. For this purpose, they strove for producing not only a constant-time CSIDH instantiation but also a randomness-free implementation of it. Meyer, Campos, and Reith [48] (see also [49]) presented a more efficient constant-time instantiation of CSIDH for practical purposes. They introduced several algorithmic tricks, including the SIMBA technique, and sampling secret keys from varying intervals. This was further improved by Onuki, Aikawa, Yamazaki, and Takagi [52], who proposed to keep track of two points to evaluate the action of an ideal: one in $E(\mathbb{F}_p)$, and one in $E(\mathbb{F}_{p^2})$ with its x-coordinate in $\mathbb{F}_p$. Moreover, Moriya, Onuki, and Takagi [50], and Cervantes-Vázquez *et al.* [23], performed more efficient CSIDH isogeny computations using the twisted Edwards model of elliptic curves. The authors of [23] proposed a more computationally demanding dummy-free variant of CSIDH, which in exchange, is arguably better suited to resist physical attacks from stronger adversaries, such as fault attacks.

In a second wave of studies around CSIDH, several crucial building blocks were improved. [26, 34] presented a framework that permits to adapt the optimal strategies of SIDH/SIKE into the context of CSIDH. The computation of large degree isogenies using an improved version of Vélu's formulas known as $\sqrt{\text{élu}}$ [8], was exploited in [1, 5]. Variants of CSIDH were reported in [19, 20, 25].

A breakthrough in the performance of constant-time CSIDH was achieved by Banegas, Bernstein, Campos, Chou, Lange, Meyer, Smith, and Sotáková [5], resulting in an almost twofold speedup. They introduce a variant, named CTIDH, using a new key space and accompanying constant-time algorithms that exploit the idea of batching isogeny degrees. However, the performance evaluation of [5] is restricted to primes of 512 and 1024 bits. The authors of [24] presented SQALE, the first CSIDH implementation at higher security levels going all the way from primes of size 2000 bits up to 9000 bits.

CSIDH is not the only attempt at building a post-quantum NIKE. SIDH [28, 35] was first introduced as a NIKE, but was shown to be insecure in the static-static scenario [31]. At the cost of many parallel executions of SIDH, this was addressed in [3], before SIDH was completely broken in [18, 46, 56]. The only post-quantum NIKE that is not based on isogenies is based on (R/M)LWE and, according to Lyubashevsky, goes back to "folkore" [45]. Such a NIKE was first analyzed in [37]. A more concrete instantiation of this approach is the recently proposed Swoosh [30]. We discuss differences between CSIDH and Swoosh in a bit more detail in Section 7.

Availability of software. We place our CSIDH software into the public domain (CC0). All software described in this paper and all measurement data from the TLS experiments are available at `https://github.com/kemtls-secsidh/code`.

Organization of this paper. Section 2 presents the necessary background on isogeny-based cryptography and introduces CSIDH and its CTIDH instantiation. Section 3 explains how we instantiate dCSIDH and CTIDH and choose parameters for our optimized implementations. Section 4 introduces algorithmic optimizations that apply to our instantiations of dCSIDH and CTIDH. Section 5 details our optimization techniques for finite field arithmetic, in particular the efficient Karatsuba-based field arithmetic, and presents benchmarking results for the group action evaluation for dCSIDH and CTIDH. Section 6 describes our integration of dCSIDH and CTIDH into OPTLS and presents handshake performance results. Finally, Section 7 concludes the paper and sketches directions for future work.

2 Preliminaries

2.1 NIKEs vs. KEMs

We briefly recall the definitions of non-interactive key exchange (NIKE) and key-encapsulation mechanism (KEM) as follows:

Definition 1. *A* non-interactive key exchange (NIKE) *is a collection of two algorithms,* KeyGen *and* $\mathsf{SharedKey}$*, where*

- KeyGen *is a probabilistic algorithm that on input* 1^k*, where* k *is a security parameter, outputs a keypair* $(\mathsf{sk}, \mathsf{pk})$*; and*
- $\mathsf{SharedKey}$ *is a deterministic algorithm that on input a public key* pk *and a secret key* sk *outputs a shared key* K*.*

A NIKE is correct if for any $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{KeyGen}(1^k)$ *and* $(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{KeyGen}(1^k)$ *it holds that* $\mathsf{SharedKey}(\mathsf{pk}_1, \mathsf{sk}_2) = \mathsf{SharedKey}(\mathsf{pk}_2, \mathsf{sk}_1)$*.*

Definition 2. *A* key-encapsulation mechanism (KEM) *is a collection of three algorithms,* KeyGen*,* Encaps*, and* Decaps*, where*

- KeyGen *is a probabilistic algorithm that on input* 1^k*, where* k *is a security parameter, outputs a keypair* $(\mathsf{sk}, \mathsf{pk})$*; and*

– Encaps *is a probabilistic algorithm that on input a public key* pk *outputs a ciphertext* ct *and a shared key* K.
– Decaps *is a deterministic algorithm that on input a ciphertext* ct *and a secret key* sk *outputs a shared key* K.

A KEM is correct if for any $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^k)$ *and* $(\mathsf{ct}, K) \leftarrow \mathsf{Encaps}(\mathsf{pk})$ *it holds that* $\mathsf{Decaps}(\mathsf{ct}, \mathsf{sk}) = K$.

Both NIKEs and KEMs can be used for key exchange, but the non-interactive nature of a NIKE makes it more flexible than a KEM. In the context of their use in protocols, there are three different scenarios:

1. Some scenarios naturally use a KEM. Those scenarios can alternatively also use a NIKE, but they do not benefit in any way from the non-interactive nature of a NIKE. An example for this scenario is the ephemeral key exchange in TLS 1.3, which currently uses (EC)DH, but will easily migrate to post-quantum KEMs [13, 14, 41, 42, 61].
2. Some protocols, most notably the X3DH protocol in Signal [47] have to use a NIKE and cannot replace this NIKE by a KEM. The reason is that this protocol cannot assume communication partners to be online at the same time and critically relies on the non-interactive nature of a NIKE.
3. Some protocols are somewhat in between: they can be designed from KEMs only, but this comes at the cost of more communication rounds. This has been discussed in some detail in the design of post-quantum Noise [2] and also in the context of the NIKE-based OPTLS [38] vs. the KEM-based KEMTLS [58]. We will revisit the comparison of these two protocols in a post-quantum context in more detail in Section 6.

2.2 The CSIDH NIKE

Background. Let $\mathbb{F}_p$ be a finite field of prime order p, such that $p + 1 = f \cdot g \cdot \prod_{i=1}^{n} \ell_i$, where each ℓ_i is a small odd prime, $f \geq 4$ is a power of two, and g is a cofactor guaranteeing that p is prime. Now consider the set of supersingular elliptic curves over $\mathbb{F}_p$, i.e., the elliptic curves with $p + 1$ $\mathbb{F}_p$-rational points. We will represent these curves in the Montgomery model, i.e., through an equation of the form

$$E_A \colon y^2 = x^3 + Ax^2 + x, \quad A \in \mathbb{F}_p. \tag{1}$$

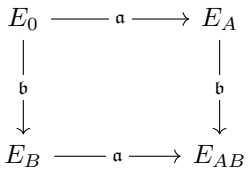
This is possible since the group order $(p + 1)$ is a multiple of 4. In the context of CSIDH we are interested in *isogeny graphs of degree* N, denoted $\mathcal{G}_N(\mathbb{F}_p)$. The vertices of such graphs are precisely the supersingular curves over $\mathbb{F}_p$; the edges are $\mathbb{F}_p$-rational isogenies of degree N. CSIDH relies on the following property: for each small odd prime ℓ_i dividing $p + 1$, a supersingular curve E_A has only two (supersingular) neighbors in the isogeny graph $\mathcal{G}_{\ell_i}(\mathbb{F}_p)$ (i.e., isogenies over $\mathbb{F}_p$ of degree ℓ_i). We can uniquely describe these isogenies by their kernels: The unique cyclic subgroup of order ℓ_i of $E_A(\mathbb{F}_p)$ defines the isogeny from E_A to one of these

neighbors $E_{A'}$. This cyclic subgroup can be described by any of its generators, which in this case means that finding a point in $E_A(\mathbb{F}_p)$ of order ℓ_i is enough to describe an isogeny of degree ℓ_i. As $E_{A'}$ is again supersingular, $E_{A'}(\mathbb{F}_p)$ has order $p+1$ as well and hence a unique cyclic subgroup of order ℓ_i, which gives an isogeny to the unique neighbor that is not E_A. The general action of moving in this direction in this graph $\mathcal{G}_{\ell_i}(\mathbb{F}_p)$ using the unique subgroup of order ℓ_i is denoted by $\mathfrak{l}_i$, and the curve $E_{A'}$ that is reached from E_A by this action is denoted $\mathfrak{l}_i * E_A$. In short, $\mathfrak{l}_i$ represents one step in the isogeny graph $\mathcal{G}_{\ell_i}(\mathbb{F}_p)$, and each small odd prime ℓ_i dividing $p+1$ gives us such an $\mathfrak{l}_i$. Steps in $\mathcal{G}_{\ell_i}(\mathbb{F}_p)$, represented by $\mathfrak{l}_i$, are *commutative*, so that applying $\mathfrak{l}_i$ to $\mathfrak{l}_j * E_A$ is the same as applying $\mathfrak{l}_j$ to $\mathfrak{l}_i * E_A$ for different degrees ℓ_i and ℓ_j. We can also compute steps in the other direction, which is denoted by $\mathfrak{l}_i^{-1} * E_A$. The subgroup of points of order ℓ_i with x-coordinate in $\mathbb{F}_p$ and y-coordinate in $\mathbb{F}_{p^2} \setminus \mathbb{F}_p$ uniquely defines the corresponding isogeny kernels. Applying both $\mathfrak{l}_i$ and $\mathfrak{l}_i^{-1}$ effectively cancels out, i.e., we have $\mathfrak{l}_i * (\mathfrak{l}_i^{-1} * E) = \mathfrak{l}_i^{-1} * (\mathfrak{l}_i * E) = E$.

The CSIDH scheme. The CSIDH scheme [22] unrolls naturally from the action described above: The secret key is a vector of n integers $(e_1, \ldots, e_n)$ defining the product $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$. In the original proposal the integers e_i are chosen from $\{-m, \ldots, m\}$ for some $m \in \mathbb{N}$, which results in a key space of size $(2m+1)^n$. The public key is the supersingular curve E_A which corresponds to the secret key $\mathfrak{a}$ applied to a publicly known starting curve E_0:

$$E_A = \mathfrak{a} * E_0 = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_0. \tag{2}$$

This public key E_A can be encoded by the single value $A \in \mathbb{F}_p$ (see Equation (1)). Shared-key computation is the same as public-key computation, except that instead of the public parameter E_0 it uses a public key E_A as input curve. That is, Alice and Bob compute their shared secret by calculating $E_{AB} = \mathfrak{a} * E_B = (\mathfrak{a} \cdot \mathfrak{b}) * E_0$ and $E_{BA} = \mathfrak{b} * E_A = (\mathfrak{b} \cdot \mathfrak{a}) * E_0$, respectively, with $E_{AB} = E_{BA}$ thanks to the commutativity. This is summarized by the following diagram:



Computing the group action $\mathfrak{a} * E$. Straightforward high-level pseudocode for the computation of the group action $\mathfrak{a} * E$ is given in Algorithm 1. The dominating cost is the construction and evaluation of the ℓ_i-isogenies corresponding to the action of the $\mathfrak{l}_i$ (Lines 5 and 7), which in turn decompose into a sequence of operations in $\mathbb{F}_p$. However, the high-level view also illustrates an additional complication for *secure* implementations of CSIDH, namely that the number of iterations of the inner loop (Line 3) and the direction of the isogenies corresponding to the action of $\mathfrak{l}_i$ (Line 4) depend on the secrets e_i and naive implementations thus leak secret information through timing.

Algorithm 1 High-level view of the CSIDH group action computation.

Input: $I \in \mathbb{F}_p$ defining a curve E_I
Input: secret key $(e_1, \ldots, e_n)$
Output: $R \in \mathbb{F}_p$ defining a curve $E_R = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_I$

$E_R \leftarrow E_I$
for i from 1 to n **do**
  for j from 1 to $|e_i|$ **do**
    if $e_i > 0$ **then**
      $E_R \leftarrow \mathfrak{l}_i * E_R$
    else
      $E_R \leftarrow \mathfrak{l}_i^{-1} * E_R$
    end if
  end for
end for
return R

For constant-time behavior, we need to be careful not to leak this information on e_i. Current implementations of CSIDH hide e_i by computing m isogenies per degree ℓ_i, while effectively performing $|e_i|$ isogenies, e.g., by using dummy computations or computations that effectively cancel each other such as $\mathfrak{l}_i * \mathfrak{l}_i^{-1} * E$.

For the sake of simplicity, Algorithm 1 omits the description of several underlying building blocks. For example, the computation of an isogeny of degree ℓ_i requires as input a point of order ℓ_i. Points of a prescribed order can be obtained probabilistically by sampling random points on the current curve. Any randomly sampled point T can generate exactly one isogeny of those degrees ℓ_i that divide the order of T, by pushing T through such isogenies to get a similar point T on the codomain curve. The order in which we perform such ℓ_i-isogenies giving a point T that can perform multiple of them influences the performance. Hence, different *strategies*, i.e. orderings of ℓ_i-isogenies, point evaluations, and point multiplications, can affect performance. Several efficient strategies are described in, e.g., [22, 26]. We describe our choices for the CSIDH group action computation in more detail in Section 3 and Section 4.

Computing a single isogeny $E_R \leftarrow \mathfrak{l}_i * E_R$. A single isogeny $E_R \leftarrow \mathfrak{l}_i * E_R$ can be computed in multiple ways: Traditionally, the formulas introduced by Vélu [60] are used, at a cost of approximately 6ℓ field multiplications for an isogeny of degree ℓ. In 2020, [8] presented new formulas for constructing and evaluating isogenies of degree ℓ, at a combined cost of just $\tilde{O}(\sqrt{\ell})$ field multiplications, denoted as √élu. With respect to CSIDH, [1] reports that the √élu formulas of [8] improve the traditional formulas for isogenies of degree $\ell \geq 89$, and concludes that constant-time CSIDH implementations using 511- and 1023-bit primes are moderately improved by the √élu formulas. [21] introduced a third method to compute isogenies, specifically those of degree less than 13, by using radical computations. We will not make use of this method, as analysis by [25] shows that this is unfavorable when the base field $\mathbb{F}_p$ is larger than 1024 bits.

2.3 CTIDH

Banegas, Bernstein, Campos, Chou, Lange, Meyer, Smith, and Sotáková [5] proposed a new approach for constant-time CSIDH, named CTIDH. The main novelties are a different way of specifying the key spaces, and some algorithmic adaptions in order to obtain a constant-time algorithm.

CTIDH key spaces. For defining CTIDH keyspaces, we organize the primes ℓ_i in N *batches*, such that each batch consists of consecutive primes. In particular, we choose a vector of batch sizes $N = (N_1, \ldots, N_B)$ with all $N_i > 0$, such that $\sum_{i=1}^{B} N_i = n$. Then we distribute the n prime degrees $\ell_1, \ldots, \ell_n$ among those B batches. That is, we define the first batch as $(\ell_{1,1}, \ldots, \ell_{1,N_1}) := (\ell_1, \ldots, \ell_{N_1})$, the second batch as $(\ell_{2,1}, \ldots, \ell_{2,N_2}) := (\ell_{N_1+1}, \ldots, \ell_{N_1+N_2})$, etc. Accordingly, we relabel the private key elements e_k as $e_{i,j}$.

Instead of directly sampling the key elements $e_{i,j}$ from some interval $[-m, m]$ as in CSIDH, CTIDH only limits the 1-norm of each key batch. That is, for the i-th batch $(\ell_{i,1}, \ldots, \ell_{i,N_i})$, we fix a bound m_i and sample corresponding key elements $e_{i,j}$ such that $\sum_{j=1}^{N_i} |e_{i,j}| \leq m_i$. This means that for each isogeny we compute for the i-th batch, its degree could be any of $\ell_{i,1}, \ldots, \ell_{i,N_i}$. This adds a combinatorial advantage, in the sense that the same number of isogenies as in CSIDH leads to a much larger key space size in CTIDH. In other words, CTIDH requires a smaller number of isogenies for reaching the same key space size. For example, the fastest previous constant-time implementation of CSIDH-512 with key space size 2^{256} required the computation of 438 isogenies, while the CTIDH parameters of [5] only requires 208 isogenies for the same key space size. For details, we refer to [5]. We note that as defined above, CSIDH is a special case of CTIDH using n batches of size 1.

CTIDH algorithm. The main problem for constant-time implementations with this adapted key space lies in the fact that we must hide the degree of each isogeny from side channels. Given that the computational effort for an isogeny directly depends on its degree, a straightforward implementation of CTIDH would leak the degree of each isogeny. On the other hand, an attacker must not be able to observe to which degree out of $\{\ell_{i,1}, \ldots, \ell_{i,N_i}\}$ each isogeny for the i-th batch corresponds. [5] achieves this by using an observation from [9]. The usual isogeny formulas [8, 60], have a *Matryoshka-doll structure*. That is, if $\ell_i < \ell_j$, then an ℓ_j-isogeny performs exactly the computations that an ℓ_i-isogeny would require, plus some extra operations. Therefore, we can easily compute an ℓ_i-isogeny at the cost of an ℓ_j-isogeny, by performing dummy operations for the extra steps. In CTIDH, we use this idea to compute each isogeny for the i-th batch $(\ell_{i,1}, \ldots, \ell_{i,N_i})$ at the cost of the most expensive degree, i.e., an ℓ_{i,N_i}-isogeny. In this way, the isogeny degrees do not leak via timing channels.

There are several other operations that require adjustments in CTIDH in order to obtain a constant-time implementation. For instance, this includes scalar multiplications that produce points of suitable order, or point rejections, which must occur independently of the required isogeny degree. For details on how these issues are resolved, we refer to [5].

Even though these algorithmic adjustments induce some computational overhead, CTIDH is almost twice as fast as its CSIDH counterpart for the CSIDH-512 and CSIDH-1024 parameter sets from [22] (see [5]).

2.4 Quantum security

While classical security imposes a restriction on the minimum key space size, quantum security usually poses more restrictive requirements. However, it is argued in [24] that for reasonable key spaces (that is, spaces large enough to achieve classical security), the quantum security of CSIDH relies only on the size of the prime p, regardless of the size of the actual key space being used. This is due to the fact that the most efficient quantum attack, Kuperberg's algorithm [40], requires working over a set with a group structure. Since the entire group representing all possible isogenies is of size roughly $\sqrt{p}$,[9] this attack needs to search a space much larger than the keyspace itself, which only depends on n and the exponent bound m. For example, in the case of CSIDH-512, the element $\mathfrak{l}_3$ alone generates the entire group of size roughly 2^{257} [10]. It is expected that a handful of $\mathfrak{l}_i$ generate the entire group also for larger instances. In a nutshell, classical security is determined by the size of the key space, whereas quantum security is determined by the size of p, as long as the key space is not chosen particularly badly, e.g., as a small subgroup of the full class group.

3 Proposed instantiations of CSIDH

In this section, we describe how to instantiate and choose parameters for large-parameter CSIDH. We describe two different approaches to selecting parameters: dCSIDH targets a deterministic and dummy-operation-free implementation[10], whereas CTIDH optimizes for the batching strategies proposed in [5]. This reflects the two extreme choices one can make to either prioritize security against physical attacks or speed. We note that there are several choices in the middle ground, trading off physical security for speed. For comparability, both approaches share the choice of underlying finite fields $\mathbb{F}_p$, which we detail in Section 3.1.

3.1 The choice of p

In this work, we take the conservative parameter suggestions from [24] at face value. In particular, we consider primes of 2048 and 4096 bits to target NIST security level 1, 5120 and 6144 bits to target NIST security level 2, and 8192 and 9216 bits to target NIST security level 3. Each pair of bitsizes represents a choice between more "aggressive" assumptions (with attacker circuit depth bounded by 2^{60}) or more "conservative" assumptions (attacker circuit depth bounded by

[9] The $\mathfrak{l}_i$ represent elements of the *class group* $\mathcal{C}\ell(\mathbb{Z}[\sqrt{p}])$, which has size roughly $\sqrt{p}$.

[10] Our implementation does not take the recent physical attacks [7, 17] into account, whose impact in the high-parameter range is unclear. Heuristically, countermeasures against both attacks should not impact performance by much.

2^{80}). As stressed in [24], this choice of parameters does not take into account the cost of calls to the CSIDH evaluation oracle on a quantum computer and is likely to underestimate security. However, as discussed in Section 1, we merely aim at giving performance results for conservative parameters.

All our implementations use primes of the form $p = f \cdot \prod_{i=1}^{n} \ell_i - 1$, where ℓ_i are distinct odd primes, f is a large power of 2 and n denotes the number of such ℓ_i dividing $p+1$. For these sizes of p, it becomes natural to pick secret key exponents $e_i \in \{-1, +1\}$, as n can be chosen large enough to reach the desired keyspace size [23, 24]. In particular, to achieve a keyspace of b bits in CSIDH we need to have at least $n = b$ of these ℓ_i in this case.

For conservative instances, we base the keyspace sizes on the classical meet-in-the-middle (MITM) attack considered in [22], requiring $b = 2\lambda$ for security parameter λ. That is, $b = 256,\ 256,\ 384$ for p4096, p6144, p9216, respectively. On the other hand, for aggressive instances we based the keyspace size on the limited-memory van Oorschot-Wiener golden collision search [59] with the assumptions from [24], which leads to $b = 221,\ 234,\ 332$ for p2048, p5120, p8192, respectively.

Finally, we restrict to cofactors f for which the power of 2 is a multiple of 64, since the arithmetic optimizations discussed in Section 5 require this shape. Hence, to find optimal primes for our implementation, we let $\ell_1, \ldots, \ell_b$ be the b smallest odd primes and then compute the cofactor f as the largest power of 2^{64} that fits in the leftover bitlength. This still leaves us with a bitlength slightly smaller than the target, and hence the leftover bits can be used to search for additional factors ℓ_i (making $n > b$) that make $f \cdot \prod_{i=1}^{n} \ell_i - 1$ a prime number. These extra factors go unused for dCSIDH, where they are viewed as part of the cofactor, but are exploited by the batching strategies of CTIDH to increase performance. We set a minimum requirement of 5 additional ℓ_i factors (that is, $n \geq b + 5$), decreasing f by a single factor of 2^{64} when not enough bits were left over. The results of this search are shown in Table 1.

Table 1: Parameters for reconstructing each prime $p = f \cdot \prod_{i=1}^{n} \ell_i - 1$. In each case the ℓ_i are assumed to be the first n odd primes, excluding some primes and including larger primes ℓ_i to ensure that p is prime. These are given in the Excluded and Included columns.

Prime bits	f	n	**Excluded**	**Included**	**Key Space**	**NIST level**
p2048	2^{64}	226	$\{1361\}$	–	2^{221}	1 (aggressive)
p4096	2^{1728}	262	$\{347\}$	$\{1699\}$	2^{256}	1 (conservative)
p5120	2^{2944}	244	$\{227\}$	$\{1601\}$	2^{234}	2 (aggressive)
p6144	2^{3776}	262	$\{283\}$	$\{1693, 1697, 1741\}$	2^{256}	2 (conservative)
p8192	2^{4992}	338	$\{401\}$	$\{2287, 2377\}$	2^{332}	3 (aggressive)
p9216	2^{5440}	389	$\{179\}$	$\{2689, 2719\}$	2^{384}	3 (conservative)

3.2 Parameters for dummy-free, deterministic dCSIDH

The restriction of exponents to $\{-1, +1\}$ makes it easier to make dCSIDH deterministic and dummy free [23, 24], as we always perform only one isogeny of each degree, with the only variable being the "direction" of each isogeny. Since isogenies in either direction require exactly the same operations, it is easy to obtain a constant-time implementation without using dummy operations.

Randomness appears in the traditional CSIDH implementation: it arises from the fact that performing isogenies of degree ℓ_i requires a point of order ℓ_i as input, and such a point is obtained by sampling random points on the current curve. Any random point can either be used for "positive" steps $\mathfrak{l}_i^{+1}$ or "negative" steps $\mathfrak{l}_i^{-1}$. Hence, a point of order ℓ_i can be used only once and only for a specific orientation. Doing more than one isogeny of each degree requires us, therefore, to sample new points midway. However, by restricting e_i to $\{-1, +1\}$, we have to compute only one isogeny per degree ℓ_i. This allows us to avoid random sampling by providing a pair of points T_+, T_- beforehand whose orders are divisble by all ℓ_i, where T_+ can be used for the positive steps $\mathfrak{l}_i$ with $e_i = 1$, and T_- for the negative steps $\mathfrak{l}_i^{-1}$, with $e_i = -1$. We refer to such points as *full-torsion* points, as they allow us to perform an isogeny of every degree ℓ_i by multiplying them by the right scalar. That is, to perform an ℓ_i-isogeny in the "plus" direction, we can use the point $[\frac{p+1}{\ell_i}]\, T_+$ of order ℓ_i.

Note that the probability for the order of a random point to contain the factor ℓ_i is given by $\frac{\ell_i - 1}{\ell_i}$. Thus, sampling for a pair of full-torsion points can be expensive when small factors ℓ_i are used, as they dominate the probability $\prod \frac{\ell_i - 1}{\ell_i}$ of sampling a full-torsion point. Since the primes we use always have additional ℓ_i factors that are unused in dCSIDH (see Section 3.1), we make point sampling more efficient by always discarding the smallest primes rather than the largest ones, increasing the odds to sample a full-torsion point. For example, the prime p4096 has 262 ℓ_i factors but only needs a keyspace of 2^{256}, hence we can discard 6 primes. By discarding the 6 *smallest* ones, the probability to sample a full-torsion point goes up from $\prod_{i=1}^{256} \frac{\ell_i - 1}{\ell_i} \approx 0.151$ to $\prod_{i=7}^{262} \frac{\ell_i - 1}{\ell_i} \approx 0.418$, making it more than 2.7 times as easy to sample full-torsion points T_+ and T_-. Such a shift in primes causes a trade-off in the rest of the protocol, as higher-degree isogenies are more expensive. However, due to the improvements in [8], the extra cost of using $\ell_{257}, \ldots, \ell_{262}$ instead of $\ell_1, \ldots, \ell_6$ is relatively small in comparison to the total cost of a group action computation. Thus, discarding the smallest ℓ_i is preferable as it significantly decreases the cost of sampling full-torsion points, and only increases the cost of computing $\mathfrak{a} * E$ by a marginal amount.

The points T_+, T_- on the starting curve E_0 can be precomputed and considered public parameters, but for the public-key curves they must be computed in real time. We include the computation of these points in the key generation, and include them in the public key, which makes the shared-secret derivation completely constant-time and deterministic. The key generation is then the only part that does not run in strictly constant wall-clock time (yet is implemented following the constant-time paradigm), but is still made deterministic by sampling points in a pre-defined order. As we describe in Section 4, these points can be

represented in a very compact form, which increases public-key sizes by only a few bits. We further emphasize that in order to avoid active attacks, the shared-key computation must validate these transmitted points to be full-torsion points.

Following the SQALE implementation [24], we use the optimal strategy approach from [26] to efficiently evaluate the class group action.

3.3 Parameters for CTIDH

As mentioned above, the instantiations of dCSIDH that we use are designed as dummy-free and deterministic algorithms, in order to avoid potential issues with randomness and dummy operations. However, these choices induce significant computational overhead. Therefore, we additionally give performance results for CTIDH [5], the fastest available constant-time implementation of CSIDH (allowing randomness and dummy operations), at the same security levels so that we can compare performance. Note that [5] only reports performance results for 512-bit and 1024-bit primes.

For the parameter sizes considered in this work, we thus use the same primes as in the dCSIDH case (see Table 1). This allows for a simple comparison of the two approaches, since both implementations use the same finite field arithmetic (see Section 5). On the other hand, it is unclear which parameters are optimal for CTIDH with the given prime sizes. A larger number of small prime factors ℓ_i in the factorization of $p+1$ can be beneficial, since the combinatorial advantage of CTIDH batching increases with the number of available prime degrees. On the other hand, this would mean that we have to include larger ℓ_i, and therefore compute more expensive large degree isogenies. Furthermore, the choice of CTIDH parameters, i.e., batches and norm bounds, becomes more challenging at larger prime sizes. We thus leave the exploration of optimal CTIDH parameters for large primes as future work.

For the given primes, we use the greedy algorithm from [5] for determining these additional parameters, adapted to the case of the cofactor $f > 4$. On input of the primes ℓ_i and a fixed number of batches, the algorithm searches for a locally optimal way of batching the primes, and according norm bounds, such that the expected number of field multiplications per group action evaluation is minimized. However, for the parameter sizes in this work, the greedy search becomes increasingly inefficient. We could thus only run searches for a small set of potential batch numbers, especially for the larger parameters. We obtained these potential inputs by extrapolating from the data of smaller parameter sizes from [5] and slightly beyond. For concrete parameter choices, we refer to our software. Note that the choice of a different number of batches could improve the results, but an exhaustive search using the greedy algorithm seems out of reach.

Apart from the parameters and batching setup, our CTIDH implementation uses the algorithms and strategies from [5]. We remark that CTIDH could in theory also be implemented in a dummy-free or deterministic way. [5] presents an algorithm that avoids dummy isogenies, but points out that the Matryoshka isogenies require dummy operations by design. Thus, the current techniques do not allow for a dummy-free implementation of CTIDH. Further, the design of

a deterministic variant of CTIDH requires some adaptions, such as computing multiple isogenies per batch in a single round. We leave the design and analysis of such an implementation for future work.

4 Optimizing dCSIDH and CTIDH

Given the parameter choices from Section 3, we describe the high-level optimizations we apply for dCSIDH and CTIDH. Note that apart from the improved public key validation, we use the standard CTIDH implementation from [5] extended to the parameter sizes from Section 3.3. For dCSIDH, we present several improvements in Section 4.2.

4.1 Supersingularity verification

For the prime choices from Section 3.1, we need to adapt the supersingularity verification from [22]. In particular, given primes with cofactor $\log f > \frac{1}{2}\log p$, both algorithms discussed in [22, Alg. 1 and Alg. 3] to test supersingularity of a public key E_A do not work.

Note that these supersingular tests, verify whether $\#E_A(\mathbb{F}_p) = p+1$, by showing that there is a point P with large enough order $N \mid p+1$. Both algorithms start by sampling a random point P, followed by a multiplication by the cofactor $P \leftarrow [f]P$, and then by checking whether the resulting point has ℓ_i-torsion. This is done by calculating if $[\prod_{j\neq i} \ell_j]P \neq \mathcal{O}$ and $[\prod \ell_j]P = \mathcal{O}$. If the random point P has ℓ_i-torsion for enough ℓ_i such that their product $\prod \ell_i \geq 4\sqrt{p}$, then in the Hasse interval $p+1-2\sqrt{p} \leq \#E_A(\mathbb{F}_p) \leq p+1+2\sqrt{p}$, $p+1$ is the only possible multiple of its order $\mathrm{ord}(P)$. This implies that $\#E_A(\mathbb{F}_p) = p+1$. Unfortunately, this approach cannot be applied to our setting, because for primes where $\log f > \frac{1}{2}\log p$, even a point with ℓ_i-torsion for all i does not reach the threshold $4\sqrt{p}$, as $\log(\prod \ell_i) = \log p - \log f \leq \frac{1}{2}\log p$. We conclude that due to the large cofactors included in the primes targeted in this work, [22, Alg. 1 and Alg. 3] cannot perform a sound supersingularity test within our setting.

Luckily, in the primes as above, where $f = 2^k$, we can improve this algorithm to verify supersingularity: Instead of verifying that the order of a random point P has enough ℓ_i-torsion, we verify P has 2^k-torsion. When $\log f = k > \frac{1}{2}\log p$, verifying that P has 2^k-torsion implies that E_A must be supersingular by the same logic as above. Furthermore, for Montgomery curves E_A, we can sample P directly from $E_A(\mathbb{F}_p) \setminus [2]E_A$ by picking a point with rational non-square x-coordinate [27]. This ensures we always sample P with maximum 2^k-torsion. Using x-only arithmetic, we only have to keep track of x_P. We name this approach to verify supersingularity VeriFast, as described in Algorithm 2.

VeriFast can be performed deterministically or probabilistically: Given a point with rational non-square x-coordinate, the algorithm always returns $v_2 = k$ in case of supersingularity. Otherwise, any random point is likely to have v_2 close to k, and hence still verifies supersingularity if the cofactor is a few bits larger than $4\sqrt{p}$. For the probabilistic approach, we pick $x_P = 2 \in \mathbb{F}_p$, hence $P = (2, -)$,

Algorithm 2 VeriFast: Supersingularity verification for primes with cofactor $2^k > 4\sqrt{p}$.

Input: $A \in \mathbb{F}_p$ defining a curve E_A
Output: `true` or `false`, verifying the supersingularity of E_A

$x_P \leftarrow 2$, $v_2 \leftarrow 1$
$x_P \leftarrow [\frac{p+1}{f}]x_P$
while $x_P \neq \mathcal{O}$ and $v_2 < k$ **do**
    $x_P \leftarrow \text{xDBL}(x_P)$, $v_2 \leftarrow v_2 + 1$
end while
if $v_2 > 1 + \log p/2$ and $x_P = \mathcal{O}$ **then**
    return `true`
end if
return `false`

for all supersingularity checks. This has the advantage that multiplication by 2 can be performed as a simple addition, and hence, $x_P = 2$ optimizes the arithmetic in the computation of $x_P \leftarrow [\frac{p+1}{f}]x_P$. Furthermore, the bound $4\sqrt{p}$ can be improved to $2\sqrt{p}$ as this still implies $p + 1$ is the only multiple in the Hasse interval. VeriFast is faster than any of the analyzed algorithms in [6], with a cost of $\mathcal{O}(\log p)$. More specifically, it requires a scalar multiplication by a scalar of $\log p - k$ bits and (at most) k point doublings, where $f = 2^k$ is the cofactor. In comparison to Doliskani's test [6, 29], also of complexity $\mathcal{O}(\log p)$, we have the advantage that we can stay over $\mathbb{F}_p$. The condition that $f > 1 + \log p/2$ holds for our primes p5120 and beyond. More importantly, even with the probabilistic approach, for these primes the probability to sample a point that does *not* have large enough 2^z-torsion is lower than 2^{-256}. For the primes where $f \leq 1 + \log p/2$, we can still use the 2^f-torsion, as in VeriFast, but we are required to also verify some ℓ_i-torsion to cross the bound $2\sqrt{p}$. A comparison of performance between VeriFast and previous methods is given in Table 2, showing VeriFast is 28 to 38 times as fast for large primes.

Table 2: Benchmarking results for supersingularity verification using VeriFast for primes with cofactor $\log f > \frac{1}{2}\log p$. Results of [24] added for comparison. Numbers are median clock cycles (in gigacycles) of 1024 runs on a Skylake CPU.

	p5120	**p6144**	**p8192**	**p9216**
VeriFast	0.53	0.81	1.88	2.54
SQALE [24]	14.90	27.65	67.79	96.99

4.2 Optimized dCSIDH public keys

As described in Section 3.2, dCSIDH is dummy-free and deterministic by using secret key exponents $e_i \in \{-1, 1\}$, and public keys of the form (A, T_+, T_-). Recall, T_+ and T_- are full-torsion points that can be used to perform positive steps $\mathfrak{l}_i^{+1}$ and negative steps $\mathfrak{l}_i^{-1}$ respectively. For sampling suitable points T_+ and T_- for public keys during key generation, we use the Elligator map $(A, u) \mapsto (T_+, T_-)$ from [23], with Montgomery parameter $A \in \mathbb{F}_p$ and an Elligator seed $u \in \mathbb{F}_p$. The output of Elligator is exactly such a pair of points T'_+ and T'_-, although they might not be *full-torsion*, that is, their respective orders might not be divisible by *all* ℓ_i. Let P be either T_+ or T_-. To efficiently determine if P is a full-torsion point, we follow the usual product-tree approach that was also applied for public key validation in [22]. This requires us to compute $\left[\frac{p+1}{\ell_i}\right] P$ for each ℓ_i, and checking that these points are not equal to the point at infinity. In order to obtain a deterministic algorithm, we try Elligator seeds from a pre-defined sequence $(u_1, u_2, \ldots)$ until we find full-torsion points T_+ and T_-. To determine which of the points $T_\pm$ is T_+ resp. T_-, Elligator requires a Legendre symbol computation. In the case of our proposed dCSIDH configuration with public inputs A and u, we can use a fast non-constant-time algorithm for the Legendre symbol computation as the one presented in Hamburg [33].

Thus, a dCSIDH public key consists of an affine Montgomery coefficient $A \in \mathbb{F}_p$, and an Elligator seed $u \in \mathbb{F}_p$ such that elligator(A, u) returns two full-torsion points T_+ and T_- on E_A. We choose the fixed potential values for u small to get a public key (A, u) of only $\log_2(p) + \varepsilon$ bits for small $\varepsilon > 0$.

Finally, a user has to verify such a public key (A, u). For A, we verify E_A is supersingular as described in Section 4.1. For u, we verify that it generates two full-torsion points T_+ and T_-, by ensuring at the computation of each step $\mathfrak{l}_i^{\pm 1} * E$ that the correct multiple of *both* T_+ and T_- are not the point at infinity (i.e., both have order ℓ_i) regardless of which point we use to compute the step.

Remark 1. An alternative to finding and including an Elligator seed $u \in \mathbb{F}_p$ in the public key is to find and include small x-coordinates x_+ and x_- that define full-torsion points $T_+ = (x_+, -)$ and $T_- = (x_-, -)$. Information-theoretically, u and the pair (x_+, x_-) share similar probabilities (to generate full-torsion points) and hence their bitlengths should be comparatively small. One advantage of x_+ and x_- is that they can be found individually, which should speed up their search. We choose, however, the more succinct approach using u and Elligator.

5 Implementation

In this section, we describe the optimization steps at the level of field arithmetic to speed up both variants of CSIDH we consider. First and foremost, to enable a fair comparison, we implement a common code base for dCSIDH and CTIDH. Besides sharing the same field arithmetic, both instantiations of CSIDH share all the underlying functions required for computing the group action. However, some

required parameters and the strategy within the group action strongly differ between dCSIDH and CTIDH. In the case of dCSIDH, the group action strategy and all the required parameters are based on the implementation provided by [24]. In the case of CTIDH, we generate the batching and other parameters using the methods provided by [5].

5.1 Low-level approaches for the field arithmetic layer

For the underlying field arithmetic, we implement three different approaches. They all share the representation of integers in radix 2^{64} and use Montgomery arithmetic for efficient reductions modulo p.

1. To establish a performance baseline, our first method uses the low-level functions for cryptography (`mpn_sec_`) of the GNU Multiple Precision Arithmetic Library (GMP). Modular multiplication uses a combination of `mpn_sec_mul` and `mpn_add_n` to implement Montgomery multiplication, i.e., interleaving multiplication with reduction. We refer to this first approach as **GMP**.
2. The second approach extends the optimized arithmetic from [22], using the MULX instruction, going from 512-bit and 1024-bit integers to the larger sizes we consider in this paper. Here, we also interleave multiplication with reduction; we generate code for all field sizes from a Python script. We refer to this second approach as **OpScan**.
3. Our third strategy uses Karatsuba multiplication [36] together with the MULX optimizations used in our second approach. We describe this strategy, and in particular an optimized reduction for primes of 5120 bits and above, in more detail in Section 5.2. We refer to this third approach as **Karatsuba**.

We follow the earlier optimization efforts for CSIDH from [5, 22, 24] and focus on optimizing our code primarily on Intel's Skylake microarchitecture. More specifically, we perform all benchmarks on one core of an Intel Core E3-1260L (Skylake) CPU with hyperthreading and TurboBoost disabled. An overview of (modular) multiplication performance of the three approaches for the different field sizes is given in Table 3. In the following, we will focus on describing the fastest of the three strategies mentioned above, i.e., **Karatsuba**, in more detail.

5.2 Optimized field arithmetic using MULX and Karatsuba

We present scripts to generate optimized code using the **Karatsuba** approach, based on the **OpScan** approach. More precisely, compared to the **OpScan** approach, we achieve speedups for multiplication, squaring, and reduction.

Multiplication. The implementation of Karatsuba follows careful considerations to optimize performance. To improve efficiency, we select a breakout level into a MULX-based schoolbook multiplication with a maximum of 9×9 limbs. By choosing this threshold, the implementation aims to strike a balance between utilizing the benefits of Karatsuba's divide-and-conquer strategy and minimizing

Table 3: Benchmarking results for multiplication and reduction. Numbers are median clock cycles of 100000 runs on a Skylake CPU. Note that for the **OpScan** and the **GMP** approach, we can only provide clock cycles for multiplication including reduction, due to the interleaved Montgomery reduction.

Prime	**GMP**	**OpScan**	**Karatsuba**		
	mult + redc	mult + redc	mult	redc	mult + redc
p2048	8662	4538	1442	2648	4090
p4096	34 030	20 318	4981	9777	14 758
p5120	51 671	33 676	8601	6528	15 129
p6144	74 338	53 746	10 210	9517	19 727
p8192	131 858	92 793	17 073	17 295	34 268
p9216	168 375	118 302	20 248	19 709	39 957

the overhead of stack operations. This leads to the following number of layers of Karatsuba: $2, 3, 4, 4, 4$, and 4 for the cases p2048, p4096, p5120, p6144, p8192, and p9216, respectively. To further enhance the speed of the implementation, the assembly code avoids function calls. By generating the assembly code dynamically, the implementation can adapt to different prime sizes and adjust the multiplication algorithm accordingly.

Squaring. For squaring, we take advantage of the fact that some partial products ($a_i a_j$ such that $i \neq j$) only need to be calculated once, and then accumulated/used twice. On the lowest level of Karatsuba, where the schoolbook multiplication takes place, we implement a squaring function with the corresponding savings based on *lazy doubling* method [43] by adapting the assembly code of the squaring function of the GMP library. For a given n, the implemented method achieves the lower bound of $\frac{n^2-n}{2} + n$ required multiplications. Furthermore, we save additions on the higher levels of Karatsuba by reusing calculated values. However, as shown in Table 4, due to the chosen breakout into schoolbook multiplication and the number of available registers, the effort for dealing with the carry chains only leads to a maximum speedup of 17%. Adding a layer of Karatsuba to reduce the number of limbs for the schoolbook multiplication leads to a speedup at this level. Overall, however, extra layers negate speed-ups gained from reducing limbs.

Montgomery reduction. For the cases $p \in \{$p5120, p6144, p8192, p9216$\}$, the reduction is calculated according to the *intermediate Montgomery reduction* [4]. For this, we use Montgomery-friendly primes of the form $p = f \cdot \prod_{i=1}^{n} \ell_i - 1$ with the cofactor $f = 2^{e_2}$ where $e_2 \geq \log_2(p)/2$. Table 1 shows the respective values for f and accordingly e_2 for all chosen prime numbers.

As shown in Algorithm 3, the basic idea of this reduction is to perform two Montgomery-reduction steps modulo 2^{e_2} instead of n steps modulo 2^w as in the standard Montgomery reduction. Based on this reduction approach, we can

Table 4: Benchmarking results for multiplication and squaring for the **Karatsuba** approach. Numbers are median clock cycles of 100000 runs on a Skylake CPU.

Prime	**multiplication**	**squaring**
p2048	1442	1230
p4096	4981	4431
p5120	8601	7990
p6144	10 210	9120
p8192	17 073	15 050
p9216	20 248	19 197

further apply the available Karatsuba-based multiplication when calculating $q_0 \times \alpha$ and $q_1 \times \alpha$ (see Line 2 and 4 in Algorithm 3), leading to further speedups.

Algorithm 3 Intermediate Montgomery reduction for $p = 2^{e_2}\alpha - 1$ with $e_2 \geq \log_2(p)/2$

Input: $0 \leq a < 2^e p$
Output: $r = a2^{-2e_2} \mod p$ and $0 \leq r < p$
1: $q_0 \leftarrow a \mod 2^{e_2}$
2: $r_0 \leftarrow (a - q_0)/2^{e_2} + q_0 \times \alpha$ ▷ 1st reduction
3: $q_1 \leftarrow r_0 \mod 2^{e_2}$
4: $r \leftarrow (r_0 - q_1)/2^{e_2} + q_1 \times \alpha$ ▷ 2nd reduction
5: $r' \leftarrow r - p + 2^e$
6: **if** $r' \geq 2^e$ **then**
7: $\quad r \leftarrow r' \mod 2^e$
8: **end if**
9: **return** r

For the cases $p \in \{\text{p2048, p4096}\}$, the respective primes cannot fulfill the described requirements. Hence, we implement the *word version of the Montgomery reduction* from [4] for these cases. The complexity of Algorithm 4 is dominated by multiplications by α in Line 4. Compared to the standard Montgomery reduction, this approach reduces the number of limbs to be multiplied depending on the value of e_2. We show the results for the corresponding reduction in Table 3.

5.3 Performance results

We demonstrate the performance increase due to the high-level improvements from Section 4 and the low-level improvements from Section 5.2 for dCSIDH and CTIDH in Table 5. We compare our results to [24], the only other available implementation of CSIDH for similar parameters listing performance numbers.

Algorithm 4 Word version of the Montgomery reduction if $p = 2^{e_2}\alpha - 1$

Input: $0 \leq a < p\beta^n$
Output: $r = a\beta^{-n} \mod p$ and $0 \leq r < p$
1: $r \leftarrow a$
2: **for** $i = 0$ **to** $n - 1$ **do**
3: $\quad r_0 \leftarrow r \mod \beta$
4: $\quad r \leftarrow (r - r_0)/\beta + r_0 \times \alpha 2^{e_2 - w}$
5: **end for**
6: $r' \leftarrow r + (\beta^n - p)$
7: **if** $r' \geq \beta^n$ **then**
8: $\quad r \leftarrow r' - \beta$
9: **end if**
10: **return** r

For parameter sizes above p5120, our implementation of dCSIDH is between 55% and 60% faster than SQALE (dummy-free), and CTIDH consistently achieves a speed-up of almost 75% compared to SQALE (OAYT).

Table 5: Benchmarking results for performing a group action for dCSIDH and CTIDH, excluding key validation. Results for the dummy-free and OAYT version of [24] added for comparison. Numbers are median clock cycles (in gigacycles) of 1024 executions on a Skylake CPU.

	p2048	**p4096**	**p5120**	**p6144**	**p8192**	**p9216**
dCSIDH	7.48	34.64	31.80	47.47	127.57	219.09
SQALE (dummy-free)	–	39.35	73.57	117.57	322.57	475.64
CTIDH	2.21	11.11	11.26	17.13	43.65	68.78
SQALE (OAYT)	–	23.21	44.56	74.88	199.15	292.41

6 Non-Interactive Key Exchange in Protocols

Diffie–Hellman (DH) key exchange is probably the most well-known example of a NIKE protocol, even if it is often used as a "simple" interactive key exchange. One such example is TLS, where ephemeral DH key exchange is authenticated via a signature. This key exchange can be replaced with a KEM, as shown in [13]. Experiments by Google and Cloudflare [14, 41, 42] used the same approach.

However, in two scenarios the inherently interactive character of a KEM creates issues for protocol designers. When used with long-term keys (and a suitable PKI), a NIKE allows a user Alice to send an authenticated ciphertext to an *offline* user Bob. Signal's X3DH handshake [47] is a notable example using

this feature of NIKEs. Indeed, [15] shows that a naive replacement of the DH operations by KEMs does not work.

In the early stages of the development of TLS 1.3, Krawczyk and Wee proposed OPTLS [38], a variant that uses DH key exchange not only for ephemeral key exchange, but also for authentication. Many elements of this proposal, made it into the eventual RFC8446 [54]. Though the standard reverted to handshake signatures, the idea lives on in an Internet Draft [55].

As Kuhnen pointed out, OPTLS does use the non-interactive property of DH [39]. As part of the ephemeral key exchange, the client sends their ephemeral DH public key. For authentication, the server takes this ephemeral key share and combines it with their long-term DH key. The obtained shared secret is used to compute a MAC which is used in place of the signature in the `CertificateVerify` message. This computation proves the server's possession of the long-term secret key corresponding to the public key in the certificate. The client can compute the same shared secret by combining its ephemeral secret DH key with the certified public key, and thus verify the MAC.

6.1 Post-Quantum TLS without signatures

In a naive instantiation of an OPTLS-like protocol with KEMs, we require an additional round-trip. To compute the authentication message, the server needs to first receive the ciphertext that was encapsulated against the long-term public key held in its certificate—which the client can not send before having received it from the server. The KEMTLS proposal by Schwabe, Stebila, and Wiggers avoids this issue partially by letting the client already transmit data immediately after computing and sending the ciphertext to the server [58]. This relies on the fact that any keys derived from the shared secret encapsulated to the server's long term key are *implicitly authenticated*. KEMTLS has the advantage of not having to compute any typically expensive and/or large post-quantum signatures during the handshake protocol. Only the variant that assumes the client already has the server's public key, for example through caching, can achieve a protocol flow that is similar to OPTLS and TLS 1.3 [57]. In that flow, the server can send authenticated data immediately on their first response to the client.

However, as CSIDH does provide post-quantum NIKE we can use it to instantiate post-quantum OPTLS and avoid any online post-quantum signatures. Because OPTLS immediately confirms the server's authenticity, its handshake has the same number of transmissions of messages as in TLS 1.3 and there is no need to rely on implicit authentication.

6.2 Benchmarking set-up

Integration into Rustls. To investigate the performance of OPTLS with CSIDH, we integrate our optimized implementations into the implementation and the measurement framework of the authors of KEMTLS. As a side effect of this work, we also provide a Rust wrapper around our C implementations.

We add OPTLS to the same modified version of Rustls [11] used to implement KEMTLS. This allows us to straightforwardly compare to KEMTLS and TLS 1.3 handshakes instantiated with post-quantum primitives.

Ephemeral key generation. Because CSIDH key generation takes a lot of time, we implement caching of ephemeral keys. This reduces the forward secrecy; but it emulates a best-case scenario for CSIDH-based OPTLS in which the keys are generated "offline", outside the handshake context. We exclude all first TLS handshakes from clients and servers from our measurements, to exclude this key generation time: in the *pregen* OPTLS instances, all subsequent handshakes use the same public key material. In the *ephemeral* OPTLS instances, we generate ephemeral keys in each handshake.

Measurement setup. We run all TLS handshake experiments on a server with two Intel Xeon Gold 6230 CPUs, each featuring 20 physical cores. This gives us 80 hyperthreaded cores in total. For these experiments, we do *not* disable hyperthreading or frequency scaling, as these features would also be enabled in production scenarios. We run 80 servers and clients in parallel, as each pair of client and server roughly interleave their execution. We collect 8000 measurements per experiment. Every 11 handshakes, we restart the client and server, so that we measure many ephemeral keys even in the scenarios that use ephemeral-key caching. We exclude the first handshake from the measurements to allow for cache warm-up and ephemeral-key generation in the caching scenario.

As in the KEMTLS papers [57, 58], we measure the performance of the different TLS handshakes when run over two network environments: a low-latency 30.9 ms RTT, 1000 Mbps and a high-latency 195.5 ms, 10 Mbps network connection. The latency of the former represents a continental, high-bandwidth connection, while the latter represents a transatlantic connection.

6.3 Benchmarking results

In Table 6, we compare OPTLS with dCSIDH and CTIDH with the performance of instantiations of KEMTLS and TLS 1.3.

Comparing the sizes of the handshakes, OPTLS requires fewer bytes on the wire, as it only needs to transmit two ephemeral public keys and one static public key (and the CA signature). KEMTLS requires an additional ciphertext, and TLS an additional signature.

In OPTLS, like in TLS 1.3, the client receives the server's handshake completion message SFIN first and then sends the CFIN message (and its request) immediately after. In KEMTLS, SFIN and full server authentication is received a full round-trip after CFIN is received. However, it is clear that the runtime requirements of dCSIDH are almost insurmountable, even for the smallest parameters (p2048). Even CTIDH, which is much more efficient, is orders of magnitude slower than the KEMTLS and OPTLS instances. If the more conservative p4096 prime is required for NIST level 1 security, even CTIDH handshakes do not complete in under 30 seconds. Due to better reduction, the p5120 prime performs roughly on par with p4096, while providing NIST level 2 security in the aggressive analysis.

Table 6: Public key cryptography transmission sizes in bytes and time in seconds until client receives and sends `Finished` messages for OPTLS, TLS 1.3 and KEMTLS.

				Handshake latencies (RTT, link speed)			
		Transmission		**30.9 ms, 1000 Mbps**		**195.5 ms, 10 Mbps**	
		KEX	**Auth**	**SFIN recv**	**CFIN sent**	**SFIN recv**	**CFIN sent**
OPTLS (pregen)	dCSIDH p2048	1024	1178	24.476	24.476	24.185	24.185
	CTIDH p2048	1024	1178	7.350	7.350	7.189	7.189
	CTIDH p4096	2048	1690	36.189	36.189	36.032	36.032
	CTIDH p5120	2560	1946	35.624	35.624	35.417	35.417
OPTLS (ephemeral)	dCSIDH p2048	1024	1178	43.705	43.705	43.295	43.295
	CTIDH p2048	1024	1178	10.059	10.059	9.816	9.816
	CTIDH p4096	2048	1690	49.868	49.868	49.660	49.660
	CTIDH p5120	2560	1946	49.487	49.487	49.023	49.023
TLS	Kyber512–Falcon512	1568	2229	0.064	0.064	0.428	0.428
	Kyber512–Dilithium2	1568	4398	0.063	0.063	0.520	0.520
	Kyber768–Falcon1024	2272	3739	0.065	0.065	0.498	0.498
KEMTLS	Kyber512	1568	2234	0.094	0.063	0.593	0.396
	Kyber768	2272	2938	0.094	0.063	0.598	0.400

All instantiations use Falcon-512 for the certificate authority; the CA public key is not transmitted. Bytes necessary for authentication includes 666 bytes for the Falcon-512 CA signature on the server's certificate.

As discussed for (KEM)TLS in [32], for constrained environments, such as 46 kbps IoT networks, in certain scenarios the transmission size can become the dominant factor instead of computation time. However, with the results shown here, we expect the environments in which CSIDH-based OPTLS instances are competitive to be very niche.

Interestingly, the CSIDH experiments run on the high-latency, low-bandwidth networks show slightly lower latencies than those on the high-bandwidth, low-latency network. We suspect that this is due to an interaction with the TCP congestion control algorithm's transmission windows.

7 Conclusion and future work

In this paper, we presented low-level and high-level optimizations for CSIDH at larger parameter sets, focusing on deterministic and dummy-free behavior in dCSIDH, and on speed in CTIDH. These optimizations achieve impressive results on their own; dCSIDH is almost twice as fast as the state-of-the-art, and CTIDH, dropping determinism, is again three times as fast as dCSIDH. Further optimizations of the field arithmetic, i.e. by utilizing the vector processing capabilities of modern processors, might lead to additional speed-ups.

Nevertheless, when integrated into OPTLS, both implementations still have too-large handshake latency in comparison to TLS or KEMTLS using lattice-based KEMs. We conclude that the reduced number of roundtrips, through the non-interactive nature of CSIDH, does not make up for the performance hit.

However, for truly non-interactive settings that cannot replace NIKEs by KEMs, the performance of CSIDH is sufficient even at high-security levels. This includes, for example, using CSIDH in X3DH [47] for post-quantum Signal, as it would incur a delay of seconds only when sending the first message to another user who is assumed to be offline.

Unless significant performance improvements occur for CSIDH in large parameter sets, or the quantum-security debate shifts in favor of 512- to 1024-bits parameter sets, we conclude that CSIDH is unlikely to be practical in real-world applications, outside of those that specifically require NIKEs.

It will be interesting to investigate how CSIDH and SWOOSH—the only two current proposals for a post-quantum NIKE—compare in a protocol context. There is no full implementation of SWOOSH, yet; the cycle counts reported in [30] are for the passively-secure core component only. Based on the available figures it seems likely that SWOOSH outperforms CSIDH with the large parameters we consider in this paper *computationally*, but that key sizes are much smaller for CSIDH.

References

[1] Gora Adj, Jesús-Javier Chi-Domínguez, and Francisco Rodríguez-Henríquez. "Karatsuba-based square-root Vélu's formulas applied to two isogeny-based protocols". In: *Journal of Cryptographic Engineering* (2022). DOI: 10.1007/s13389-022-00293-y. URL: https://doi.org/10.1007/s13389-022-00293-y.

[2] Yawning Angel, Benjamin Dowling, Andreas Hülsing, Peter Schwabe, and Florian Weber. "Post Quantum Noise". In: *ACM CCS 2022*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. ACM Press, Nov. 2022, pp. 97–109. DOI: 10.1145/3548606.3560577.

[3] Reza Azarderakhsh, David Jao, and Christopher Leonardi. "Post-Quantum Static-Static Key Agreement Using Multiple Protocol Instances". In: *SAC 2017*. Ed. by Carlisle Adams and Jan Camenisch. Vol. 10719. LNCS. Springer, Heidelberg, Aug. 2017, pp. 45–63. DOI: 10.1007/978-3-319-72565-9_3.

[4] Jean-Claude Bajard and Sylvain Duquesne. "Montgomery-friendly primes and applications to cryptography". In: *Journal of Cryptographic Engineering* 11.4 (Nov. 2021), pp. 399–415. DOI: 10.1007/s13389-021-00260-z.

[5] 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 TCHES* 2021.4 (2021). https://tches.iacr.org/index.php/TCHES/article/view/9069, pp. 351–387. ISSN: 2569-2925. DOI: 10.46586/tches.v2021.i4.351-387.

[6] Gustavo Banegas, Valerie Gilchrist, and Benjamin Smith. *Efficient supersingularity testing over $\mathbb{F}_p$ and CSIDH key validation*. Cryptology ePrint Archive, Report 2022/880. https://eprint.iacr.org/2022/880. 2022.

[7] Gustavo Banegas, Juliane Krämer, Tanja Lange, Michael Meyer, Lorenz Panny, Krijn Reijnders, Jana Sotáková, and Monika Trimoska. *Disorientation faults in CSIDH*. Cryptology ePrint Archive, Report 2022/1202. https://eprint.iacr.org/2022/1202. 2022.

[8] 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.

[9] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. "Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies". In: *EUROCRYPT 2019, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 409–441. DOI: 10.1007/978-3-030-17656-3_15.

[10] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. "CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations". In: *ASIACRYPT 2019, Part I*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11921. LNCS. Springer, Heidelberg, Dec. 2019, pp. 227–247. DOI: 10.1007/978-3-030-34578-5_9.

[11] Joseph Birr-Pixton. *A modern TLS library in Rust*. https://github.com/rustls/rustls (accessed 2021-12-20).

[12] Xavier Bonnetain and André Schrottenloher. "Quantum Security Analysis of CSIDH". In: *EUROCRYPT 2020, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. LNCS. Springer, Heidelberg, May 2020, pp. 493–522. DOI: 10.1007/978-3-030-45724-2_17.

[13] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. "Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem". In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 553–570. DOI: 10.1109/SP.2015.40.

[14] Matt Braithwaite. *Experimenting with Post-Quantum Cryptography*. Google Online Security Blog. https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html (accessed 2021-12-20). 2016.

[15] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. "Towards Post-Quantum Security for Signal's X3DH Handshake". In: *SAC 2020*. Ed. by Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn. Vol. 12804. LNCS. Springer, Heidelberg, Oct. 2020, pp. 404–430. DOI: 10.1007/978-3-030-81652-0_16.

[16] 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.

[17] Fabio Campos, Michael Meyer, Krijn Reijnders, and Marc Stöttinger. *Patient Zero and Patient Six: Zero-Value and Correlation Attacks on*

CSIDH and SIKE. IACR Cryptology ePrint Archive, Report 2022/904. To appear in SAC 2022. 2022. URL: https://eprint.iacr.org/2022/904.

[18] Wouter Castryck and Thomas Decru. "An Efficient Key Recovery Attack on SIDH". In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Heidelberg, Apr. 2023, pp. 423–447. DOI: 10.1007/978-3-031-30589-4_15.

[19] Wouter Castryck and Thomas Decru. "CSIDH on the Surface". In: *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*. Ed. by Jintai Ding and Jean-Pierre Tillich. Springer, Heidelberg, 2020, pp. 111–129. DOI: 10.1007/978-3-030-44223-1_7.

[20] Wouter Castryck, Thomas Decru, Marc Houben, and Frederik Vercauteren. "Horizontal Racewalking Using Radical Isogenies". In: *ASIACRYPT 2022, Part II*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13792. LNCS. Springer, Heidelberg, Dec. 2022, pp. 67–96. DOI: 10.1007/978-3-031-22966-4_3.

[21] Wouter Castryck, Thomas Decru, and Frederik Vercauteren. "Radical Isogenies". In: *ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, Heidelberg, Dec. 2020, pp. 493–519. DOI: 10.1007/978-3-030-64834-3_17.

[22] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *ASIACRYPT 2018, Part III*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11274. LNCS. Springer, Heidelberg, Dec. 2018, pp. 395–427. DOI: 10.1007/978-3-030-03332-3_15.

[23] 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: *LATINCRYPT 2019*. Ed. by Peter Schwabe and Nicolas Thériault. Vol. 11774. LNCS. Springer, Heidelberg, Oct. 2019, pp. 173–193. DOI: 10.1007/978-3-030-30530-7_9.

[24] 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* 12.3 (Sept. 2022), pp. 349–368. DOI: 10.1007/s13389-021-00271-w.

[25] Jesús-Javier Chi-Domínguez and Krijn Reijnders. "Fully Projective Radical Isogenies in Constant-Time". In: *CT-RSA 2022*. Ed. by Steven D. Galbraith. Vol. 13161. LNCS. Springer, Heidelberg, Mar. 2022, pp. 73–95. DOI: 10.1007/978-3-030-95312-6_4.

[26] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. "Optimal strategies for CSIDH". In: *Adv. Math. Commun.* 16.2 (2022), pp. 383–411. DOI: 10.3934/amc.2020116. URL: https://doi.org/10.3934/amc.2020116.

[27] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. "Efficient Compression of SIDH Public Keys". In:

EUROCRYPT 2017, Part I. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. LNCS. Springer, Heidelberg, 2017, pp. 679–706. DOI: 10.1007/978-3-319-56620-7_24.

[28] Craig Costello, Patrick Longa, and Michael Naehrig. "Efficient Algorithms for Supersingular Isogeny Diffie-Hellman". In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 572–601. DOI: 10.1007/978-3-662-53018-4_21.

[29] Javad Doliskani. "On division polynomial PIT and supersingularity". In: *Applicable Algebra in Engineering, Communication and Computing* 29.5 (2018), pp. 393–407.

[30] Phillip Gajland, Bor de Kock, Miguel Quaresma, Giulio Malavolta, and Peter Schwabe. *Swoosh: Practical Lattice-Based Non-Interactive Key Exchange*. Cryptology ePrint Archive, Report 2023/271. https://eprint.iacr.org/2023/271. 2023.

[31] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. "On the Security of Supersingular Isogeny Cryptosystems". In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 63–91. DOI: 10.1007/978-3-662-53887-6_3.

[32] Ruben Gonzalez and Thom Wiggers. "KEMTLS vs. Post-quantum TLS: Performance on Embedded Systems". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Lejla Batina, Stjepan Picek, and Mainack Mondal. Cham: Springer Nature Switzerland, 2022, pp. 99–117. ISBN: 978-3-031-22829-2. DOI: 10.1007/978-3-031-22829-2. URL: https://thomwiggers.nl/publication/kemtls-embedded/.

[33] Mike Hamburg. *Computing the Jacobi symbol using Bernstein-Yang*. Cryptology ePrint Archive, Report 2021/1271. https://eprint.iacr.org/2021/1271. 2021.

[34] Aaron Hutchinson, Jason T. LeGrow, Brian Koziel, and Reza Azarderakhsh. "Further Optimizations of CSIDH: A Systematic Approach to Efficient Strategies, Permutations, and Bound Vectors". In: *ACNS 20, Part I*. Ed. by Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi. Vol. 12146. LNCS. Springer, Heidelberg, Oct. 2020, pp. 481–501. DOI: 10.1007/978-3-030-57808-4_24.

[35] David Jao and Luca De Feo. "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies". In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*. Ed. by Bo-Yin Yang. Springer, Heidelberg, 2011, pp. 19–34. DOI: 10.1007/978-3-642-25405-5_2.

[36] Anatolii Karatsuba and Yuri Ofman. "Multiplication of multidigit numbers on automata". In: *Soviet Physics Doklady* 7 (1963). Translated from Doklady Akademii Nauk SSSR, Vol. 145, No. 2, pp. 293–294, July 1962., pp. 595–596.

[37] Bor de Kock. *A non-interactive key exchange based on ring-learning with errors*. Master's thesis, Eindhoven University of Technology. 2018.

[38] Hugo Krawczyk and Hoeteck Wee. "The OPTLS Protocol and TLS 1.3". In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 81–96. DOI: 10.1109/EuroSP.2016.18.

[39] Wouter Kuhnen. "OPTLS revisited". https://www.ru.nl/publish/pages/769526/thesis-final.pdf. MA thesis. Radboud University, 2018.

[40] Greg Kuperberg. "Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem". In: *8th Conference on the Theory of Quantum Computation, Communication and Cryptography*. Ed. by Simone Severini and Fernando G. S. L. Brandão. Vol. 22. LIPIcs 22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013, pp. 20–34. DOI: 10.4230/LIPIcs.TQC.2013.20.

[41] Kris Kwiatkowski and Luke Valenta. *The TLS Post-Quantum Experiment*. Cloudflare blog. https://blog.cloudflare.com/the-tls-post-quantum-experiment/ (accessed 2022-01-06). 2019.

[42] Adam Langley. *CECPQ2*. ImperialViolet blog. https://www.imperialviolet.org/2018/12/12/cecpq2.html (accessed 2021-12-20). 2018.

[43] Younho Lee, Il-Hee Kim, and Yongsu Park. "Improved multi-precision squaring for low-end RISC microcontrollers". In: *J. Syst. Softw.* 86.1 (2013), pp. 60–71. DOI: 10.1016/j.jss.2012.06.074. URL: https://doi.org/10.1016/j.jss.2012.06.074.

[44] Jason LeGrow and Aaron Hutchinson. *An Analysis of Fault Attacks on CSIDH*. Cryptology ePrint Archive, Report 2020/1006. https://eprint.iacr.org/2020/1006. 2020.

[45] Vadim Lyubashevsky. *Converting NewHope/LWE key exchange to a Diffe-Hellman-like algorithm*. Crypto Stack Exchange. [Online:] https://crypto.stackexchange.com/questions/48146/converting-newhope-lwe-key-exchange-to-a-diffe-hellman-like-algorithm. 2017. URL: https://crypto.stackexchange.com/questions/48146/converting-newhope-lwe-key-exchange-to-a-diffe-hellman-like-algorithm (visited on 06/10/2017).

[46] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. "A Direct Key Recovery Attack on SIDH". In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Heidelberg, Apr. 2023, pp. 448–471. DOI: 10.1007/978-3-031-30589-4_16.

[47] Moxie Marlinspike and Trevor Perrin. *The X3DH Key Agreement Protocol*. Signal Specifications. https://signal.org/docs/specifications/x3dh/(accessed2022-01-04). 2016.

[48] 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*. Ed. by Jintai Ding and Rainer Steinwandt. Springer, Heidelberg, 2019, pp. 307–325. DOI: 10.1007/978-3-030-25510-7_17.

[49] Michael Meyer and Steffen Reith. "A Faster Way to the CSIDH". In: *INDOCRYPT 2018*. Ed. by Debrup Chakraborty and Tetsu Iwata. Vol. 11356. LNCS. Springer, Heidelberg, Dec. 2018, pp. 137–152. DOI: 10.1007/978-3-030-05378-9_8.

[50] Tomoki Moriya, Hiroshi Onuki, and Tsuyoshi Takagi. "How to Construct CSIDH on Edwards Curves". In: *CT-RSA 2020*. Ed. by Stanislaw Jarecki. Vol. 12006. LNCS. Springer, Heidelberg, Feb. 2020, pp. 512–537. DOI: 10.1007/978-3-030-40186-3_22.

[51] NIST. *Post-Quantum Cryptography Standardization*. 2017 (last modified Dec 2, 2021). URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization(accessed2022-01-04).

[52] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. "(Short Paper) A Faster Constant-Time Algorithm of CSIDH Keeping Two Points". In: *IWSEC 19*. Ed. by Nuttapong Attrapadung and Takeshi Yagi. Vol. 11689. LNCS. Springer, Heidelberg, Aug. 2019, pp. 23–33. DOI: 10.1007/978-3-030-26834-3_2.

[53] Chris Peikert. "He Gives C-Sieves on the CSIDH". In: *EUROCRYPT 2020, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. LNCS. Springer, Heidelberg, May 2020, pp. 463–492. DOI: 10.1007/978-3-030-45724-2_16.

[54] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. IETF RFC 8446. https://rfc-editor.org/rfc/rfc8446.txt. 2018.

[55] Eric Rescorla, Nick Sullivan, and Christopher A. Wood. *Semi-Static Diffie-Hellman Key Establishment for TLS 1.3*. Internet-Draft. https://tools.ietf.org/html/draft-rescorla-tls-semistatic-dh-02. Internet Engineering Task Force, 2020.

[56] Damien Robert. "Breaking SIDH in Polynomial Time". In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Heidelberg, Apr. 2023, pp. 472–503. DOI: 10.1007/978-3-031-30589-4_17.

[57] Peter Schwabe, Douglas Stebila, and Thom Wiggers. "More Efficient Post-quantum KEMTLS with Pre-distributed Public Keys". In: *ESORICS 2021, Part I*. Ed. by Elisa Bertino, Haya Shulman, and Michael Waidner. Vol. 12972. LNCS. Springer, Heidelberg, Oct. 2021, pp. 3–22. DOI: 10.1007/978-3-030-88418-5_1.

[58] Peter Schwabe, Douglas Stebila, and Thom Wiggers. "Post-Quantum TLS Without Handshake Signatures". In: *ACM CCS 2020*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM Press, Nov. 2020, pp. 1461–1480. DOI: 10.1145/3372297.3423350.

[59] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications". In: *Journal of Cryptology* 12.1 (Jan. 1999), pp. 1–28. DOI: 10.1007/PL00003816.

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

[61] Bas Westerbaan and Cefan Daniel Rubin. *Defending against future threats: Cloudflare goes post-quantum*. Cloudflare blog. https://blog.cloudflare.com/post-quantum-for-all/. 2022.