Arithmetic Considerations for Isogeny Based Cryptography

Joppe W. Bos and Simon Friedberger

Abstract—In this paper we investigate various arithmetic techniques which can be used to potentially enhance the performance in the supersingular isogeny Diffie-Hellman (SIDH) key-exchange protocol which is one of the more recent contenders in the post-quantum public-key arena. Firstly, we give a systematic overview of techniques to compute efficient arithmetic modulo $2^x p^y \pm 1$. Our overview shows that in the SIDH setting, where arithmetic over a quadratic extension field is required, the approaches based on Montgomery reduction for such primes of a special shape are to be preferred. Moreover, the outcome of our investigation reveals that there exist moduli which allow even faster implementations.

Secondly, we investigate if it is beneficial to use other curve models to speed-up the elliptic curve scalar multiplication. The use of twisted Edwards curves allows one to search for efficient addition-subtraction chains for fixed scalars while this is not possible with the differential addition law when using Montgomery curves. Our preliminary results show that despite the fact that we found such efficient chains, using twisted Edwards curves does not result in faster scalar multiplication arithmetic in the setting of SIDH.

1 Introduction

Recent significant advances in quantum computing have accelerated the research into *post-quantum* cryptography schemes [21], [35], [49]. Such schemes can be used as drop-in replacements for classical public-key cryptography primitives. This demand is driven by interest from standardization bodies, such as the call for proposals for new public-key cryptography standards [47] by the National Institute of Standards and Technology (NIST) [15] and the European Union's prestigious PQCrypto research effort [2].

One such recent approach is called Supersingular Isogeny Diffie-Hellman (SIDH), which was introduced in 2011 and is based on the hardness of constructing a smooth-degree isogeny, between two supersingular elliptic curves defined over a finite field [31]. The Supersingular Isogeny Key Encapsulation (SIKE) protocol [3] submission to NIST is based on the SIDH approach with optimizations from recent work such as [19], [24]. The full details of this protocol are outside the scope of this paper. However, the arithmetic in supersingular isogeny cryptography is performed in quadratic extension fields of a prime field $\mathbf{F}_q$ with $q = 2^x p^y \pm 1$; where the extension field is formed as $\mathbf{F}_{q^2} = F_q(i)$ with $i^2 = -1$. The computationally expensive operations consist of computing a number of elliptic curve scalar multiplications with ℓ and evaluations of ℓ-isogenies for $\ell \in \{2, p\}$ which in turn translate to a number of arithmetic operation in $\mathbf{F}_{q^2}$.

In the proposed SIKE protocol $p = 3$ and the elliptic curve arithmetic is performed using Montgomery curves [46]. These choices are motivated by performance arguments. In this paper we investigate alternative approaches. On the one hand we study, in Section 3, if different choices for p in the modulus $q = 2^x p^y \pm 1$ can result in faster modular reduction. We find that alternative design choices can indeed lead to practical performance gain for the modular arithmetic. However, this ignores the impact on the isogeny computations: details about such larger odd degree isogeny computation can be found in [18].

On the other hand we investigate in Section 4 if the elliptic curve scalar multiplications can be done more efficiently by using curve arithmetic on different curve models. Montgomery curves are extremely efficient but require the differential addition law to gain this performance. We study if switching to twisted Edwards curves [23], [7], [6], [29] is beneficial: this setting allows one to more generic use addition/subtraction chains which can lower the number of arithmetic operations when computing small powers of p at-a-time. We note that addition chains in the setting of SIDH have been studied in [39] before. However, the focus and goal of [39] is to find addition chains to aid in the computation of modular inversion and modular square root computation.

This paper is an extended version of a previous work which appeared as [11]. The main result presented in [11] is captured in Section 3. This has been extended with the addition-subtraction chain investigation as presented in Section 4. The code which implements the modular arithmetic presented in Section 3 can be found at https://github.com/sidh-arith.

2 Preliminaries

2.1 Modular Multiplication

One well-known approach to enhance the practical performance of modular multiplication by a constant factor is based on precomputing a particular value when the used modulus m is fixed. We recall two such approaches in this section.

In the remainder of the paper we use the following notation. By $\mathbf{Z}_m$ we denote the finite ring $\mathbf{Z}/m\mathbf{Z}$: the ring of integers modulo m which we might write as $\mathbf{F}_m$ when m

- *J. W. Bos is with NXP Semiconductors, Leuven, Belgium.*
- *S. Friedberger is with NXP Semiconductors, Leuven, Belgium and KU Leuven - iMinds - COSIC, Leuven, Belgium.*

is prime. The bit-length of m is denoted by $N = \lceil \log_2(m) \rceil$. We target computer architectures which use a *word* size w which can represent unsigned integers less than $r = 2^w$ (typical values are $w = 32$ or $w = 64$): this means that most unsigned arithmetic instructions work with inputs bounded by 0 and r and the modulus m can be represented using $n = \lceil N/w \rceil$ computer words. We represent integers (or residues in $\mathbf{Z}_m$) in a radix-R representation: given a positive integer R, a positive integer $a < R^\ell$ for some positive integer ℓ can be written as $a = \sum_{i=0}^{\ell-1} a_i \cdot R^i$ where $0 \leq a_i < R$ for $0 \leq i < \ell$. In order to assess the performance of various modular multiplication or reduction approaches we count the number of required multiplication instructions to implement this in software. This instruction is a map $\text{mul} : \mathbf{Z}_r \times \mathbf{Z}_r \to \mathbf{Z}_{r^2}$ where $\text{mul}(x, y) = x \cdot y$. We are aware that just considering the number of multiplication instructions is a rather one-dimensional view which ignores the required additions, loads / stores and cache behavior but we argue that this metric is the most important characteristic when implementing modular arithmetic for the medium sized residues which are used in the current SIDH schemes. We verify this assumption by comparing to implementation results in Section 3.5.

2.1.1 Montgomery reduction

The idea behind Montgomery reduction [45] is to change the representation of the integers used and change the modular multiplication accordingly. By doing this one can replace the cost of a division by roughly the cost of a multiplication which is faster in practice by a constant factor. Given a modulus m co-prime to r, the idea is to select the Montgomery radix such that $r^{n-1} < m < r^n$.

Given an integer c (such that $0 \leq c < m^2$) Montgomery reduction computes

$$\frac{c + (\mu \cdot c \bmod r^n) \cdot m}{r^n} \equiv c \cdot r^{-n} \pmod{m},$$

where $\mu = -m^{-1} \bmod r^n$ is the precomputed value which depends on the modulus used. After changing the representation of $a, b \in \mathbf{Z}_m$ to $\tilde{a} = a \cdot r^n \bmod m$ and $\tilde{b} = b \cdot r^n \bmod m$, Montgomery reduction of $\tilde{a} \cdot \tilde{b} \equiv a \cdot b \cdot r^{2n} \pmod{m}$ becomes $a \cdot b \cdot r^{2n} \cdot r^{-n} \equiv a \cdot b \cdot r^n \pmod{m}$ which is the Montgomery representation of $a \cdot b \bmod m$. Hence, at the start and end of the computation a transformation is needed to and from this representation. Therefore, Montgomery multiplication is best used when a long series of modular arithmetic is needed; a setting which is common in public-key cryptography.

It can be shown that when $0 \leq c < m^2$ then $0 \leq \frac{c + (\mu \cdot c \bmod r^n) \cdot m}{r^n} < 2m$ and at most a single conditional subtraction is needed to reduce the result to $[0, 1, \ldots, m-1]$. This conditional subtraction can be omitted when the Montgomery radix is selected such that $4m < r^n$ and a redundant representation is used for the input and output values of the algorithm. More specifically, whenever $a, b \in \mathbf{Z}_{2m}$ (the redundant representation) where $0 \leq a, b < 2m$, then the output $a \cdot b \cdot r^{-n}$ is also upper-bounded by $2m$ and can be reused as input to the Montgomery multiplication again without the need for a conditional subtraction [52], [55].

As presented the multiplication and the modular reduction steps are separated. This has the advantage that asymptotically fast approaches for the multiplication can be used. The downside is that the intermediate results in the reduction parts of the algorithm are stored in up to $2n + 1$ computer words. The radix-r interleaved Montgomery multiplication algorithm [22] combines the multiplication and reduction step digit wise. This means the precomputed Montgomery constant needs to be adjusted to $\mu = -m^{-1} \bmod r$ and the algorithm initializes c to zero and then updates it according to

$$c \leftarrow \frac{c + a_i \cdot b + (\mu \cdot (c + a_i \cdot b) \bmod r) \cdot m}{r} \tag{1}$$

for $i = 0$ to $n-1$. The intermediate results are now bounded by r^{n+1} and occupy at most $n + 1$ computer words. It is not hard to see that the cost of computing the reduction part of the interleaved Montgomery multiplication requires $n^2 + n$ multiplication instructions since the divisions and multiplications by r in the interleaved algorithm (or r^n in the non-interleaved algorithm) can be computed using shift operations when r is a power of two.

2.1.2 Barrett Reduction

After the publication of Montgomery reduction Barrett proposed a different way of computing modular reductions using precomputed data which only depends on the modulus used [5]. The idea behind this method is inspired by a technique of emulating floating point data types with fixed precision integers. Let $m > 0$ be the fixed modulus used such that $r^{n-1} < m < r^n$ where r is the word-size of the target architecture (just as in Section 2.1.1). Let $0 \leq c < m^2$ be the input which we want to reduce. The idea is based on the observation that $c' = c \bmod m$ can be computed as

$$c' = c - \left\lfloor \frac{c}{m} \right\rfloor \cdot m. \tag{2}$$

Hence, this approach computes not only the remainder c' but also the quotient

$$q = \lfloor \frac{c}{m} \rfloor$$

of the division of c by m and does not require any transformation of the inputs.

In order to compute this efficiently the idea is to use a precomputed value $\mu = \lfloor \frac{r^{2n}}{m} \rfloor < r^{n+1}$ to *approximate* q by

$$q_1 = \left\lfloor \frac{c \cdot \mu}{r^{2n}} \right\rfloor = \left\lfloor \frac{c}{r^{2n}} \cdot \left\lfloor \frac{r^{2n}}{m} \right\rfloor \right\rfloor.$$

This is a close approximation since one can show that $q - 1 \leq q_1 \leq q$ and the computation uses cheap divisions by r which are shifts. The multiplication $c \cdot \mu$ can be computed in a naive fashion with $2n(n+1)$ multiplication instructions. The computation of $q_1 \cdot m$ (to compute the remainder c' in Eq. 2) can be carried out with $(n + 1)n$ multiplication instructions for a total of $3n(n + 1)$ multiplications.

Since $m > r^{n-1}$ the $n - 1$ lower computer words of c contribute at most 1 to $q = \lfloor \frac{c}{m} \rfloor$. When defining $\hat{c} = \lfloor c / r^{n-1} \rfloor < r^{n+1}$ one can further approximate q by

$$q_2 = \left\lfloor \frac{\lfloor c / r^{n-1} \rfloor \cdot r^{n-1} \cdot \mu}{r^{2n}} \right\rfloor = \left\lfloor \frac{\hat{c} \cdot r^{n-1} \cdot \mu}{r^{2n}} \right\rfloor = \left\lfloor \frac{\hat{c} \cdot \mu}{r^{n+1}} \right\rfloor.$$

This approximation is still close since $q - 2 \leq q_2 \leq q$.

A straight-forward optimization is to observe that $\hat{c} \cdot \mu$ is divided by r^{n+1} and a computation of the full-product is therefore not needed. It suffices to compute the $n+3$ most significant words of the product ignoring the lower $n-1$ computer words. Similarly, the product $q_2 \cdot m$ in Eq. 2 only requires the $n+1$ least significant words of the product. Hence, these two products can be computed using

$$\underbrace{(n+1)^2 - \sum_{i=1}^{n-1} i}_{\text{for } \hat{c}\cdot\mu} + \underbrace{\sum_{i=1}^{n} i + n}_{\text{for } q_2\cdot m} = (n+1)^2 + 2n \qquad (3)$$

multiplication instructions. This is larger compared to the n^2+n multiplication instructions needed for the Montgomery multiplication but no change of representation is required.

2.1.3 Reducing arbitrary length input.

Barrett reduction is typically analyzed for an input c which is bounded above by r^{2n} and a modulus $m < r^n$. We now consider the more general scenario where c is bounded by r^ℓ and the quotient and remainder are computed for a divisor $m < r^n$. We derive the number of multiplications required for Barrett reduction.

Computing the k least significant words using schoolbook multiplication of a times b where $0 \le a < r^{\ell_a}$ and $0 \le b < r^{\ell_b}$ can be done using $\mathcal{L}(\ell_a, \ell_b, k)$ multiplication instructions where

$$\mathcal{L}(\ell_a, \ell_b, k) = \sum_{i=0}^{\min\{k-1, \ell_a+\ell_b\}} \min\{i+1, \ell_a+\ell_b-(i+1), \ell_a, \ell_b\}.$$

On the other hand, to compute the k most significant words of a multiplication result with an error of at most 1 we need to compute $k+2$ words of the result. For a product of length n this means *not* computing the least significant $n-k-2$ words. Hence, computing the most significant $k+2$ words of the product of a and b costs $\mathcal{H}(\ell_a, \ell_b, k)$ multiplication instructions where

$$\mathcal{H}(\ell_a, \ell_b, k) = k^2 - \mathcal{L}(\ell_a, \ell_b, \ell_a + \ell_b - k - 2).$$

Combining these two we get the total cost CostBarrett(ℓ, n) expressed in multiplication instructions for computing the quotient and remainder when dividing c ($0 \le c < r^\ell$) by m ($0 \le m < r^n$) using Barrett reduction as CostBarrett$(\ell, n) = \mathcal{H}(\ell-n+1, \ell-n+1, \ell-n+1) + \mathcal{L}(\ell-n+1, n, n+1)$. Note that CostBarrett$(2n, n) = (n+1)^2 - \sum_{i=0}(i+1) + n + \sum_{i=0}^{n-1}(i+1)$ which equals Eq. (3) as expected.

2.1.4 Folding.

An optimization to Barrett reduction which needs additional precomputation but reduces the number of multiplications is called *folding* [28]. Given an N-bit modulus m and a D-bit integer c where $D > N$ a partial reduction step is used first and next regular Barrett is used to reduce this number further. First, a cut-off point x such that $N < x < D$ is selected and a precomputed constant $m' = 2^x \bmod m$ is used to compute $c' \equiv c \bmod m$ as

$$c' = (c \bmod 2^x) + \left\lfloor \frac{c}{2^x} \right\rfloor \cdot m'.$$

Now $c' < 2^x + 2^{D-x+N}$ at the cost of multiplying a $D-x$ bit with an N bit integer. This is a more general description compared to the one in [28] where $D = 2N$ and $x = 3N/2$ is used such that c is reduced from $2N$ bits to at most $1.5N+1$ bits.

2.2 Efficient Elliptic Curve Arithmetic

For a field $\mathbf{K}$ of characteristic larger than three, any $a, b \in \mathbf{K}$ with $4a^3 + 27b^2 \neq 0$ define an elliptic curve $E_{a,b}$ over $\mathbf{K}$ (see for more details e.g. [53]). The group of points is defined as the set of pairs $(x, y) \in \mathbf{K} \times \mathbf{K}$ that satisfy the short Weierstrass equation

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

combined with the zero point. The group law is written additively.

In practice, different defining equations and coordinate systems can be used to speed-up the curve arithmetic. One such example are Montgomery curves [46]. Montgomery showed that it is possible to simplify the computations by dropping the y-coordinate. This allows very fast doubling operations and *differential* additions where P+Q is computed from P, Q, and $P-Q$ for $P, Q \in E_{a,b}(\mathbf{K})$. This type of arithmetic is compatible with SIDH and is the preferred option used in practice [3]. The cost for the group law expressed in terms of operations in $\mathbf{K}$ is shown in Table 1. However, the cost for doubling and tripling is one multiplication by a curve constant higher compared to the typical usage in elliptic curve cryptography. This is due to the use of projective curve coefficients to avoid modular inversion as outlined in [19].

Currently the asymptotically fastest elliptic curves for random scalars with bit-length going to infinity are due to Edwards in 2007 [23]. Here, performance is understood as the cost of a group operation expressed in multiplications and squarings in $\mathbf{K}$. These Edwards curves have been generalized in [7], [6] showing their practical use in cryptology. Moreover, it can be shown that every such twisted Edwards curve is birationally equivalent to a Montgomery curve over $\mathbf{K}$ [6]. The fastest known approach to perform elliptic curve point addition and doubling uses extended twisted Edwards coordinates [29]. See Table 1 for an overview of the cost to compute the group law expressed in terms of multiplication and squarings in $\mathbf{K}$. The cost can be reduced even further when the curve coefficient a used in the definition of the twisted Edwards curve is set to -1 as shown in [29]. Unfortunately, it does not seem evident how one can force the output of an isogeny computation to produce such $a = -1$ twisted Edwards curves instead of one with seemingly random a and d curve coefficients except for computing this at the cost of a modular inversion. Therefore, we assume the usage of the general curve shape instead in the remainder of the paper.

It should be noted that the faster addition formula from [29] are due to the usage of the extended coordinate system which uses an additional coordinate T. However, this T coordinate is *not* used in the doubling formula. Hence, as explained in [29], when the typical windowed algorithm is used for scalar multiplication one can reduce the cost of the addition formula by one multiplication (by

TABLE 1
Overview of the cost of elliptic curve (differential) addition, double, and triple operations expressed as the number of multiplications ($\mathbf{M}$), squarings ($\mathbf{S}$) or multiplications by a curve constant ($\mathbf{D}$) in the field $\mathbf{K}$ for twisted Edwards curves and Montgomery curves in the setting of curve arithmetic for SIDH. The costs in brackets are when the output coordinate T for extended twisted Edwards is *not* computed.

twisted Edwards curves

Coordinates	add	double	triple [16]
Projective [6]	$10\mathbf{M}+1\mathbf{S}+2\mathbf{D}$	$3\mathbf{M}+4\mathbf{S}+1\mathbf{D}$	$9\mathbf{M}+3\mathbf{S}+1\mathbf{D}$
Extended [29]	$9\mathbf{M}+1\mathbf{D}$	$4\mathbf{M}+4\mathbf{S}+1\mathbf{D}$	$11\mathbf{M}+3\mathbf{S}+1\mathbf{D}$
	$(8\mathbf{M}+1\mathbf{D})$	$(3\mathbf{M}+4\mathbf{S}+1\mathbf{D})$	$(9\mathbf{M}+3\mathbf{S}+1\mathbf{D})$

Montgomery curves

	diff. add	double	triple
XZ [46]	$4\mathbf{M}+2\mathbf{S}$	$2\mathbf{M}+2\mathbf{S}+2\mathbf{D}$	$5\mathbf{M}+5\mathbf{S}+2\mathbf{D}$

not computing the T coordinate), use the faster projective doubling formula for all but the last doubling and use the extended doubling formula for the final doubling. In total this is equivalent to using both, the faster point addition from the extended coordinate system and the faster point doubling from the projective coordinate system. We discuss the application and impact of this technique to the SIDH setting in Section 4.2.2.

2.3 Addition Chains

Addition chains [51] are a well known method for speeding-up modular exponentiation (or, equivalently, scalar multiplication when the group law is additive). An addition chain is simply a way to construct a specific number by starting at 1 and repeatedly summing up some of the previous terms.

More formally, an *addition chain for* n is a sequence of integers

$$1 = a_0, a_1, a_2, \ldots, a_r = n$$

such that

$$a_i = a_j + a_k, \tag{4}$$

for $0 \leq i \leq r$ and $0 \leq j, k < i$ (cf. [37]).

Given an addition chain for $n \in \mathbf{Z}$ one can calculate the elliptic curve scalar multiplication $nP \in E_{a,b}(\mathbf{K})$ by computing

$$P = a_0P, a_1P, a_2P, a_3P, \ldots, a_rP = nP.$$

Since every term a_iP is computed using a point addition or a point doubling, finding a shorter addition chain for n provides a speed-up for the scalar multiplication. (The commonly used windowing [12] based elliptic curve scalar multiplication algorithm can also be seen as an addition chain.) In the setting of elliptic curve scalar multiplication one can use the so-called *addition-subtraction chains* where Eq. (4) is modified to

$$a_i = a_j + a_k, \text{ or } a_i = a_j - a_k$$

because elliptic curve point subtraction (or addition of a negated point) can be computed efficiently [48].

Moreover, to simplify, it is common to restrict the definition of addition chains with obviously unnecessary steps [25]. However, because we are dealing with addition-subtraction chains we cannot require that the $a_0, \ldots, a_r$ be ordered ascendingly and we cannot require that all $a_i < n$. Hence, we keep the following conditions for our addition-subtraction chains

- no duplicates: $a_i \neq a_j$ for $0 \leq i, j, \leq r$ where $i \neq j$,
- all intermediate values must be used: for all a_j $(0 \leq j < r)$ there is a a_k and a_i such that $a_i = a_j + a_k$ with $j < i \leq r$ and $0 \leq k < i$.

In fact, when presenting our cost function for chains it will become clear that not explicitly stating j, k for $a_i = a_j + a_k$ leads to ambiguous results. We therefore use the *formal chains* introduced by Clift [17] for our cost calculation. A formal addition chain additionally specifies exactly which terms have been used to form a specific sum. Clift does so using two mappings γ and δ with $\gamma(i) = j$ and $\delta(i) = k$ for our above example. In practice, we can simply store indices into the previous steps to solve this problem.

To accelerate our search for good addition chains we restrict it to *star chains*, also known as *Brauer chains* [12]. These are chains where each step has to use the result of the previous step. The formula that must hold for all elements thus becomes $a_i = a_{i-1} \pm a_k$. Star chains are known to be suboptimal. However, empirically star chains give close to optimal results in length and because the last value is always used a certain amount of storage optimization is built-in.

3 Fast arithmetic modulo $2^x p^y \pm 1$

The SIDH key-exchange approach uses isogeny classes of supersingular elliptic curves with smooth orders so that isogenies of exponentially large but smooth degree can be computed efficiently as a composition of low degree isogenies. To instantiate this approach let p and q be two small prime numbers and let f be an integer cofactor, then the idea is to find a prime $m = f \cdot q^x \cdot p^y \pm 1$. It is then possible to construct a supersingular elliptic curve E defined over $\mathbf{F}_{m^2}$ of order $(f \cdot q^x \cdot p^y)^2$ [13] to be used in SIDH. For efficiency reasons it makes sense to fix q to 2, which will become clear in this section. In practice, most instantiations use $q = 2$ and $p = 3$. Moreover, we assume that the cofactor $f = 1$ to simplify the explanation: our methods can be immediately generalized for other values of f.

In this section we survey different approaches to optimizing arithmetic modulo $m = 2^x p^y \pm 1$ where p is an odd small prime. The common idea is to use the special shape of the modulus to reduce the number of multiplication instructions needed in an implementation when computing arithmetic modulo m. Typically, there are two approaches to realizing this modular multiplication: the first approach computes the multiplication and reduction in two separate steps while the second approach combines these two steps by interleaving them. We refer to these methods as non-interleaved or separated and interleaved modular multiplication, respectively.

Both of these approaches have advantages and disadvantages. For instance, intermediate results are typically longer and therefore require more memory or registers in the non-interleaved approach. In some applications one approach is clearly to be preferred over the other. One such setting is when computing arithmetic in $\mathbf{F}_{m^2} = \mathbf{F}(i)$ for $i^2 = -1$ (as used in SIDH). Let $a, b \in \mathbf{F}_{m^2}$ and write $a = a_0 + a_1 \cdot i$ and $b = b_0 + b_1 \cdot i$, then $c = a \cdot b = c_0 + c_1 \cdot i$

where $c_0 = a_0b_0 - a_1b_1$ and $c_1 = a_0b_1 + a_1b_0$. This can be computed using four interleaved modular multiplications or four multiplications and two modular reductions. When using Karatsuba multiplication [33] this can be reduced to three multiplications and two modular reductions by computing c_1 as $(a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1$. In the interleaved setting this requires three modular multiplications while in the non-interleaved setting computing three multiplications and two modular reductions suffice. Hence, when computing modular arithmetic in $\mathbf{F}_{m^2}$, which is the setting in SIDH, the non-interleaved modular multiplication is to be preferred.

In this section we describe techniques to speed-up both, modular reduction and interleaved modular multiplication plus reduction when using primes of the form $2^x p^y \pm 1$.

3.1 Using Barrett reduction

The first implementation of SIDH [4] uses Barrett reduction (see Section 2.1.2) to compute modular reductions and uses primes of the form $2^x 3^y - 1$ to define the finite field. The special shape of the modulus is not exploited in this implementation.

As explained in Section 2.1.2 Barrett reduction requires two multiplications, one with the precomputed constant μ and one with the modulus m. It seems non-trivial to accelerate the multiplication with

$$\mu = \left\lfloor \frac{r^{2n}}{m} \right\rfloor = \left\lfloor \frac{r^{2n}}{2^x p^y \pm 1} \right\rfloor$$

since this typically does not have a special shape. The multiplication with $m = 2^x p^y \pm 1$, however, can be computed more efficiently since the product $a \cdot m = a \cdot 2^x \cdot p^y \pm a$ and this can be computed using shift operations (for the 2^x part) and a shorter multiplication by p^y followed by an addition or subtraction depending on the sign of the ± 1.

Assuming $2^x \approx p^y$ (which is the case in the SIDH setting) then the computation of $q_2 \cdot m$ where only the least significant $n + 1$ computer words are required can be done using

$$\text{CostBarrett}(\frac{3}{2}n, \frac{1}{2}n) = \frac{5}{8}n^2 + \frac{13}{4}n + 1$$

multiplication instructions.

3.2 Using Montgomery reduction

It is well-known, and has been rediscovered multiple times, that performance of Montgomery multiplication *can* benefit from a modulus of a specific form (cf. e.g., [43], [1], [36], [27], [9], [10]). When $m = \pm 1 \bmod r^n$ then $\mu = -m^{-1} = \mp 1 \bmod r^n$ and the multiplications by μ become negligible. Such moduli are sometimes referred to as *Montgomery-friendly primes*. In the SIDH setting $m = 2^x p^y \pm 1 = \pm 1 \bmod 2^x$, hence one can reduce the number of multiplications required when multiplying with μ. This is used by the authors of [19], [42], [30] in their high-performance SIDH implementation. Their non-interleaved approach for Montgomery multiplication uses the so-called product scanning technique [38], which was introduced in [22], which eliminates all multiplications by μ. We describe this approach and some variants below in detail. We note that in the hardware implementations described in [41], [40] a high-radix variant of the Montgomery multiplication suitable for hardware architectures [50], [8] is used.

3.2.1 Interleaved Montgomery multiplication.

As described in Section 2.1.1 the interleaved Montgomery multiplication approach interleaves the computation of the product and the modular reduction. We now describe an optimization when computing multiplications modulo $m = 2^x p^y - 1$. Every step of the algorithm computes the product of a single digit from the input $\tilde{a}$ with all digits from the other input $\tilde{b}$ and reduces the result after accumulation by one digit. We write our integers in a radix-R representation. For a fixed word size w, pick $B \in \mathbf{Z}_{>0}$ such that $Bw \leq x$ and let $R = 2^{Bw}$. This removes the multiplication with the precomputed constant μ since $\mu = -m^{-1} \bmod R = -(2^x p^y - 1)^{-1} \bmod 2^{Bw} = 1$. Hence, after initializing c to zero Eq. (1) simplifies to

$$\begin{aligned} c &\leftarrow \frac{c + a_i b + (\mu(c + a_i b) \bmod R)m}{R} \\ &= \frac{\sum_{i=0}^{n} d_i R^i + d_0(2^x p^y - 1)}{2^{Bw}} \\ &= \frac{\sum_{i=1}^{n} d_i R^i + d_0 2^x p^y}{2^{Bw}} \\ &= \sum_{i=1}^{n} d_i R^{i-1} + d_0 2^{x-Bw} p^y \end{aligned} \tag{5}$$

where $d = c + a_i \cdot b = \sum_{i=0}^{n} d_i R^i$ and this computation is repeated $\lceil \frac{N}{Bw} \rceil$ times.

The computation cost expressed in the number of multiplication instructions of Eq. (5) is

$$\left\lceil \frac{N}{Bw} \right\rceil \left(B \left\lceil \frac{N}{w} \right\rceil + B \left(\left\lceil \frac{N}{w} \right\rceil - B \right) \right). \tag{6}$$

However, since in practice $N \approx 2x$, the N-bit modulus $2^x p^y \pm 1$ has a special shape which ensures that the multiplication by $2^x p^y$ in Eq. (5) can be computed more efficiently (for some values of B). We illustrate this in the following example.

Example 1. Consider the prime $m = 2^{372}3^{239} - 1$ as used in the SIDH key-exchange protocol in [19]. In this setting $p = 3$, $x = 372$, $y = 239$, $N = 751$, and $\mu = 1$. Assuming that we target a 64-bit platform, with $w = 64$ and $n = 12$, there are five different values for B such that $64B \leq 372$. The following table shows the cost for the modular multiplication when evaluating Eq. (6)

B	Eq. (6) (#mul instructions)
$B = 1$	276
$B = 2$	264
$B = 3$	252
$B = 4$	240
$B = 5$	285

However, the multiplication by $2^{372-64B}3^{239}$ can be done more efficiently since $2^{372-64B}3^{239} \equiv 0 \pmod{2^{(5-B)\cdot 64}}$. This results in a total of 144 multiplication instructions to compute the various $a_i \cdot b$ and 84 multiplication instructions to compute the multiplication with $2^{372-64B}3^{239}$ for all $B \in \{1, 2, 3, 4\}$. When using $B = 5$ three iterations are required

resulting in 180 and 105 multiplication instructions for both parts, respectively. Therefore, in order to lower the overall number of arithmetic instruction required, B should exactly divide $n = \lceil \frac{N}{w} \rceil = 12$ (the number of 64-bit digits of m).

The authors of [19] used $B = 1$ in their implementation but this analysis shows that using a larger radix $R = 2^{Bw}$ results in the same number of multiplication instructions required.

To summarize, using $B = 1$ (or any larger B such that $B \equiv 0 \bmod n$, where $n = \lceil \frac{N}{w} \rceil$) and assuming 2^x fits in $\frac{n}{2}$ computer words and that n is even, the total cost of the modular reduction can be reduced from n^2+n (for a generic Montgomery multiplication approach) to $\frac{1}{2}n^2$ (when using the special prime shape) multiplication instructions. This is a performance increase by more than a factor two.

3.2.2 Non-interleaved Montgomery multiplication

The non-interleaved Montgomery approach, where the multiplication $c = a \cdot b$ is performed first and the modular reduction is computed in a separate next step can be done in exactly the same way as the interleaved approach. The idea is to use the product c immediately (instead of initializing this to zero in the first iteration when computing Eq. (5)) and therefore not adding the $a_i b$ values. This means that the values of d are not bounded by r^{n+1} anymore but by r^{2n+1} instead which requires computing more additions while the number of multiplications remains unchanged compared to the interleaved approach at $\frac{n^2}{2}$ (when taking advantage of the special prime shape).

3.3 Using an unconventional radix

Another idea is to use a function of the prime shape as the radix of the representation. This is exactly what a recent approach [34] suggested in the setting of designing a hardware implementation. We summarize the approach here and introduce another approach inspired by this technique. We assume that $m = 2 \cdot 2^x \cdot 3^y - 1$ where x and y are even. Use a radix $R = 2^{\frac{x}{2}}3^{\frac{y}{2}}$ to represent an integer $a < m$ as $a = a_2 \cdot R^2 + a_1 \cdot R + a_0$ where $0 \leq a_0, a_1 < R$ and $0 \leq a_2 \leq 1$. The approach outlined in [34] converts integers once at the start and once at the end of the algorithm to and from a radix-$2^{\frac{x}{2}}3^{\frac{y}{2}}$ representation, similar to when using the Montgomery multiplication. The proposed method is an interleaved modular multiplication method where throughout the computation the in- and output remain in this representation. The idea is to use the fact that $2^x \cdot 3^y \equiv 2^{-1} \pmod m$. Given two integers a and b their modular product $c = a \cdot b \bmod m$ can be computed using

$$
\begin{aligned}
&c_2 \cdot R^2 + c_1 \cdot R + c_0 = \\
&((a_2b_0 + a_1b_1 + a_0b_2) \bmod 2) \cdot R^2 + \\
&\left(\left\lfloor \frac{a_2b_1 + a_1b_2}{2} \right\rfloor + (a_1b_0 + a_0b_1) \right) \cdot R + \\
&\Big((2^{-2} \bmod m)a_2b_2 + a_0b_0 + \\
&((a_2b_1 + a_1b_2) \bmod 2) \cdot \frac{R}{2} + \left\lfloor \frac{a_2b_0 + a_1b_1 + a_0b_2}{2} \right\rfloor \Big).
\end{aligned}
$$

Assuming $r^{\frac{n}{2}-1} < 2^{\frac{x}{2}}3^{\frac{y}{2}} < r^{\frac{n}{2}}$ then each multiplication $a_i \cdot b_j$ where $i \neq 2 \neq j$ can be computed using $\frac{n^2}{4}$ multiplication instructions and four such multiplications need to be computed in total. Whenever one of the operands is either a_2 or b_2 the product can be computed without multiplications by simply selecting (or masking) the correct result. Note, however, that c_1 and c_0 need to be reduced further since they are larger than R. This is done using Barrett reduction which takes advantage of the fact that the divisions by $2^{\frac{x}{2}}$ can be done more efficiently. Assuming that c_1 and c_0 each fit in n computer words and, for simplicity, that $n \equiv 0 \bmod 4$, computing a Barrett reduction of c_0 or c_1 by $R = 2^{\frac{x}{2}}3^{\frac{y}{2}}$ requires CostBarrett$(n, \frac{n}{2}) = \frac{n^2}{4} + 2n + 1$ multiplication instructions (see Section 2.1.2). However, since the computation of the quotient and the remainder when dividing by $2^{\frac{x}{2}}$ requires no multiplications this computation can be done with CostBarrett$(\frac{3}{4}n, \frac{1}{4}n) = \frac{1}{32}n(5n+52)+1$ multiplication instructions (assuming that $r^{\frac{n}{4}-1} \leq 2^{\frac{x}{2}}, 3^{\frac{y}{2}} < r^{\frac{n}{4}}$). This is a reduction of $\frac{3}{32}n(n+4)$ multiplication instructions. Assuming the inputs are already converted in this radix-$2^{\frac{x}{2}}3^{\frac{y}{2}}$ representation (which is just a one time cost) the total cost for a single interleaved modular multiplication becomes $n^2 + 2(\frac{1}{32}n(5n + 52) + 1) = \frac{21}{16}n^2 + \frac{13}{4}n + 2$ multiplication instructions. This can be further optimized when using Karatsuba multiplication [33] to compute $a_1b_0 + a_0b_1$ as $(a_0 + a_1)(b_0 + b_1) - a_1b_1 - a_0b_0$. This lowers the number of multiplications to only *three* and improves the approach to $\frac{17}{16}n^2 + \frac{13}{4}n + 2$ multiplication instructions.

In the following we present a different method inspired by this approach for moduli of the form $m = 2^x p^y - 1$. For simplicity, we assume that both x and y are even and use a radix $R = 2^{\frac{x}{2}}p^{\frac{y}{2}}$. Represent integers as usual as $a = a_1 \cdot R + a_0$ where $0 \leq a_0, a_1 < R$. Since $R^2 = 1 \bmod m$. We have

$$
\begin{aligned}
c &\equiv a_1b_1 \cdot R^2 + \\
&\quad \Big((a_0 + a_1)(b_0 + b_1) - a_1b_1 - a_0b_0 \Big) \cdot R + a_0b_0 \\
&\equiv a_1b_1 + (\sigma_1 \cdot R + \sigma_0) \cdot R + a_0b_0 \\
&\equiv \sigma_0 \cdot R + (a_0b_0 + a_1b_1 + \sigma_1) \pmod m
\end{aligned}
$$

again using Karatsuba multiplication where $\sigma_1 \cdot R + \sigma_0 = a_0b_1 + a_1b_0$ is computed with a Barrett reduction using the special shape of R. However, this approach *does* require *both* inputs to be converted to their radix-R representation at the cost of two special Barrett reductions. Assuming $r^{\frac{n}{2}-1} < 2^{\frac{x}{2}}p^{\frac{y}{2}} < r^{\frac{n}{2}}$ then this approach requires to compute three times $\frac{n^2}{4}$ multiplication instructions and three calls to the Barrett reduction for a total of $3\frac{n^2}{4} + 3 \cdot$CostBarrett$(\frac{3n}{4}, \frac{n}{4}) = \frac{39}{32}n^2 + \frac{39}{8}n + 3$ multiplication instructions and significantly fewer additions compared to the approach from [34]. The main difference with the approach presented in [34] is that this approach requires the inputs to be converted to the correct radix system every time when computing a modular multiplication while we do not need to convert the output. When comparing the two multiplication counts it becomes clear that this approach is inherently slower compared to the one presented in [34]. However, in certain situations this approach might be preferred. For instance, when computing a modular squaring the input only needs to be converted once while such an optimization is not possible with the other approach.

TABLE 2
Estimates of the number of multiplication instructions required when using different modular multiplication (interleaved) or modular reduction (non-interleaved) approaches for a modulus stored in n computer words. It is assumed that the size of the input(s) to the modular multiplication and modular reduction have n and $2n$ computer words, respectively.

approach	method	moduli family	# muls
Montgomery [45]	interleaved	generic	$2n^2+n$
	interleaved	$2^x p^y - 1$	$\frac{3}{2}n^2$
use radix directly [34]	interleaved	$2 \cdot 2^x \cdot 3^y - 1$	$\frac{17}{16}n^2 + \frac{13}{4}n + 2$
use radix directly (new)	interleaved	$2^x p^y - 1$	$\frac{39}{32}n^2 + \frac{39}{8}n + 3$
Barrett [5]	non-interleaved	generic	n^2+4n+1
	non-interleaved	$2^x p^y \pm 1$	$\frac{5}{8}n^2 + \frac{13}{4}n + 1$
Montgomery [45]	non-interleaved	generic	n^2+n
	non-interleaved	$2^x p^y - 1$	$\frac{n^2}{2}$
use radix directly (new - v1)	non-interleaved	$2^x p^y - 1$	$\frac{5}{8}n^2 + \frac{13}{4}n + 1$
use radix directly (new - v2)	non-interleaved	$2^x p^y - 1$	$\frac{1}{2}n^2 + 2n + 1$
use radix directly (new - v3)	non-interleaved	$2^x p^y - 1$	$\frac{1}{2}n^2 + \frac{5}{4}n + 1$

3.3.1 A non-interleaved approach.

Both interleaved approaches as presented do not compete with the non-interleaved approaches when arithmetic in quadratic extension fields is required, as for SIDH. In this section we explore the possibility of using the special prime shape directly for this non-interleaved use-case.

Let the radix R be defined and bounded as $r^{n-1} \leq R = 2^x p^y > m < r^n$ and assume throughout this section that $r^{\frac{n}{2}-1} \leq 2^x < r^{\frac{n}{2}}$, the multiplication counts can be trivially adjusted when these bounds are different. Then after a multiplication step $c = a \cdot b$ ($0 \leq c < m^2$) write this integer in the radix-$2^x p^y$ representation $c = c_1 \cdot R + c_0$ and compute $c' \equiv c \bmod m$ as

$$\begin{aligned} c' &\equiv c_1 \cdot R + c_0 \\ &\equiv c_1(m+1) + c_0 \\ &\equiv c_1 + c_0 \pmod{m} \end{aligned} \tag{7}$$

where $0 \leq c' < 2R$. Hence, the main computational complexity is when computing the Barrett reduction to write c in the radix-$2^x p^y$ representation. The naive way of computing this (denoted version 0) simply does this directly using Barrett reduction at the cost of CostBarrett$(2n, n) = n^2+4n+1$ multiplication instructions. By simplifying and doing the division by 2^x separately we obtain a method which needs CostBarrett$(\frac{3}{2}n, \frac{1}{2}n) = \frac{5}{8}n^2 + \frac{13}{4}n + 1$ multiplication instructions (since $r^{\frac{n}{2}-1} \leq 2^x < r^{\frac{n}{2}}$) as outlined in the Barrett discussion. We denote this approach version 1.

However, we can do even better by using the folding approach (see Section 2.1.4). By choosing $x = n$ we can reduce the input c from $\frac{3}{2}n$ bits to n bits at the cost of $\frac{1}{4}n^2$ multiplication instructions. Afterwards it suffices to compute CostBarrett$(n, \frac{1}{2}n) = \frac{n^2}{4} + 2n + 1$ multiplication instructions: the total number of multiplication instructions to compute the modular reduction becomes $\frac{1}{2}n^2 + 2n + 1$ which is significantly better compared to version 1. We denote this version 2.

When applying another folding step we can use $x = 0.75n$ such that we reduce the input from n bits to $n - x + 0.5n = 0.75n$ bits at the cost of another $\frac{1}{8}n^2$ multiplication instructions. The total cost to compute the modular reduction is slightly reduced compared to version 2 to $\frac{1}{4}n^2 + \frac{1}{8}n^2 + \text{CostBarrett}(0.75n, 0.5n) = \frac{1}{2}n^2 + \frac{5}{4}n + 1$. We denote this version 3. Computing additional folding steps does not lower the number of required multiplication instructions.

Table 2 summarizes our findings from this section. Note that in the interleaved setting the cost for both the multiplication and the modular reduction are included while for the non-interleaved algorithms only the modular reduction cost is stated. The user can choose any asymptotically fast multiplication method in this latter setting. From Table 2 it becomes clear that the approach from [34] is to be preferred in the interleaved setting while the Montgomery approach is best in the non-interleaved setting. As mentioned before, the SIDH setting favors the non-interleaved approach due to the computation of the arithmetic in a quadratic extension field. Moreover, assuming the multiplication part is done with one level of Karatsuba, at the cost of $\frac{3}{4}n^2$ multiplication instructions, the non-interleaved approach has a total cost of $\frac{5}{4}n^2$ and is faster compared to the interleaved approach for all positive $n < 18$. Hence, independent of the application, using the non-interleaved Montgomery algorithm is the best approach for moduli up to 1100 bits.

3.4 Alternative implementation-friendly moduli

The basic requirement for SIDH-friendly moduli of the form $2^x p^y \pm 1$ when targeting the 128-bit post-quantum security level is $x \approx \log_2(p^y) \approx 384$. For security considerations the difference between the bit-sizes of 2^x and p^y can not be too large.

We fix the word size of the target platform to 64 bits for the following discussion. This can be adjusted for different architectures if needed. Hence, we expect a modulus of (around) $n = \frac{2 \cdot 384}{64} = 12$ computer words. Table 2 summarizes the effort expressed in the number of multiplication instructions needed for modular multiplication when using the interleaved approach or for the modular reduction when using the non-interleaved approach. The approaches are as outlined in Section 3 and the estimates are given as a function of n: the number of computer words required to represent the modulus. We assume that the inputs to the modular multiplication or reduction are n-words or $2n$-words long, respectively.

TABLE 3
SIDH-friendly prime moduli which target 128-bit post-quantum security.

prime shape $2^x p^y \pm 1$	bit sizes $(x, \lceil \log_2(p^y) \rceil, \lceil \log_2(2^x p^y \pm 1) \rceil)$
$2^{385}3^{227} - 1$	$(385, 360, 745)$
$2^{394}5^{154} + 1$	$(394, 358, 752)$
$2^{394}5^{155} - 1$	$(394, 360, 754)$
$2^{396}7^{131} + 1$	$(396, 368, 764)$
$2^{393}17^{91} + 1$	$(393, 372, 765)$
$2^{391}19^{88} - 1$	$(391, 374, 765)$

Below we discuss the properties of two moduli proposed in SIDH implementations and constraints to search for other SIDH-friendly moduli which enhances the practical performance of the modular reduction even further. Lower security levels like the 80-bit post-quantum security targeted by Koziel et al. [40] for their FPGA implementation are out of scope in this work.

3.4.1 The modulus $m_1 = 2(2^{386}3^{242}) - 1$.

This modulus was proposed in the first implementation of SIDH [4] and used in the implementations presented in [4], [34]. The disadvantage of m_1 is that $\lceil \log_2(m_1) \rceil = 771 > 12 \cdot 64$ which implies that $n = 13$ computer words are required to represent the residues modulo m_1. Hence, the arithmetic is implemented using one additional computer word for the same target of 128-bit post-quantum security. This increases the total number of instructions required when implementing the modular arithmetic. This is true for the modular addition and subtraction but also for the Montgomery multiplication since it needs, for instance, to compute $n = 13$ rounds when using a Montgomery radix of 2^{64} instead of the 12 rounds for slightly smaller moduli. Moreover, when using the special Montgomery reduction algorithm a multiplication with the value $2^3 3^{242} > 2^{6 \cdot 64}$ is required which fits in $\lfloor n/2 \rfloor + 1$ computer words. Hence, the number of required multiplication instructions is $n \cdot \lceil \frac{n}{2} \rceil = 91$ (see Table 2) for an odd n.

3.4.2 The modulus $m_2 = 2^{372}3^{239} - 1$.

This modulus is proposed in [19] and used in the implementations [19], [42]. The modulus m_2 was picked to resolve the main disadvantage when using m_1: $\lceil \log_2(4m_2) \rceil = 753 < 12 \cdot 64$ implies that the number of computer words required to represent residues modulo m_2 is $n = 12$ (which significantly lowers the number of multiplication instructions required compared to when implementing arithmetic modulo m_1). However, when dividing out the powers of two (due to the Montgomery radix, see Eq. (5)) we need to multiply by $2^{52}3^{239}$ which is larger than $2^{6 \cdot 64}$ and is therefore stored in seven computer words. This implies that the multiplication with this constant is more expensive than necessary and explains why the estimate for the modular reduction from Table 2 of $n^2/2 = 72$ multiplication instructions is too optimistic. When using m_2 the correct number of multiplication instructions required to implement the modular reduction is $n \cdot \left(\frac{n}{2} + 1\right) = 84$ as reported in [19]

3.4.3 Alternative moduli.

We searched for alternative implementation- and SIDH-friendly prime moduli using constraints which enhance the practical performance when using the fastest special modular reduction techniques from Section 3. Besides the size requirement to target the 128-bit post-quantum security ($n = 12$) we also set additional performance related constraints. We outline these requirements below when looking for moduli of the form $2^x \cdot p^y \pm 1$.

1) p is small in order to construct curves in SIDH, hence all the odd primes below 20, $p \in \{3, 5, 7, 11, 13, 17, 19\}$
2) require 2^x to be at least six 64-bit computer words: $384 \leq x < 450$ and $2^{300} < p^y < 2^{450}$,
3) the size of modulus is $n = 12$ computer words, the bit-length is not too small when targeting the 128-bit post-quantum security: $2^{740} < 2^x p^y \pm 1 < 2^{768}$,
4) the difference between the size of the two prime powers is not too large (balance security): $|2^x - p^y| < 2^{40}$,
5) $2^x \cdot p^y + 1$ or $2^x \cdot p^y - 1$ is prime.

Table 3 summarizes the results of our search when taking these constraints into account. The entry which maximizes $\min(x, \lceil \log_2(p^y) \rceil)$ is for the prime $m_3 = 2^{391}19^{88} - 1$ where the size of 19^{88} is 374 bits. Moreover, $\lceil \log_2(4m_3) \rceil = 767 < 12 \cdot 64$ which means one can use $n = 12$ using the subtraction-less version of the Montgomery multiplication algorithm. The input operand used for the multiplication in the Montgomery reduction is $2^7 19^{88} < 2^{6 \cdot 64}$ which lowers the overall number of multiplication instructions required to the estimate of $n^2/2 = 72$.

3.5 Benchmarking

Table 2 gives an overview of the cost of multiplication modulo $2^x p^y \pm 1$ in terms of the *word length* n of a specific modulus on a target computer platform when working with the constraints as outlined in Section 3. In practice, however, one carefully selects one particular modulus for implementation purposes. The choice of this modulus is first of all driven by the selected security parameter, which determines n, and secondly by the practical performance. This latter requirement selects the parameters p, x, and y and (up to a certain degree) can have some trade-offs with the security parameter. In this section we compare the practical performance of some of the most promising techniques from Section 3 when using the prime moduli from the proposed cryptographic implementations of SIDH presented in [19] (since this is the fastest cryptographic implementation of SIDH). Such a comparison between the fastest techniques to achieve the various modular reduction algorithms allows us to confirm if the analysis based on the number of required multiplication instructions is sound. Moreover, this immediately gives an indication of the real practical performance enhancements the various techniques or different primes give in practice.

Our benchmark platform is an Intel Xeon CPU E5-2650 v2 (running at 2.60GHz). We have created a benchmarking framework where we measure the number of cycles using the time stamp counter using the `rdtsc` instruction. More

TABLE 4
Benchmark summary and implementation details. The mean $\bar{x}$ and standard deviation σ are stated expressed in the number of cycles together with the number of assembly instructions used in the implementation.

	#cycle ($\bar{x} \pm \sigma$))	#mul	#add	#mov	#other
$2^{372}3^{239} - 1$ ($B = 1$) [19]	254.9 ± 9.5	84	332	157	41
$2^{372}3^{239} - 1$ ($B = 2$) this paper	275.3 ± 11.2	84	358	202	59
$2^{372}3^{239} - 1$ (shifted) this paper	240.2 ± 10.9	72	299	223	85
$2^{391}19^{88} - 1$ this paper	224.5 ± 8.8	72	292	145	38

specifically, we measure the time to compute 10^5 dependent modular reductions and store the mean for one operation. This process is repeated 10^4 times and from this data set the mean $\bar{x}$ and standard deviation σ are computed. After removing outliers (more than 2.5 standard deviations away from the mean) we report these findings as $\bar{x} \pm \sigma$ in Table 4. This table also summarizes the number of various required assembly instructions used for the modular reduction implementation.

Our base line comparison is the modular reduction implementation from the cryptographic software library presented in [19] which can be found online [20]. This implementation *includes* the conditional subtraction (computed in constant-time) although this is not strictly necessary. In order to make a fair comparison we include this conditional subtraction in all the other presented modular reduction algorithms as well. As indicated in Table 4 and discussed in Section 3.4 the implementation from [19] requires 84 multiplication instructions and uses the optimized non-interleaved Montgomery multiplication approach (which corresponds to the $B = 1$ setting in our generalized description from Section 3.2).

We experimented with a Montgomery multiplication version where $B = 2$ (see Section 3.2). This corresponds to using a radix-2^{128} representation and although this should not change the number of multiplications required (since $n = 12$ is even) this could potentially lower the number of other arithmetic instructions. However, due to the limited number of registers available we had a hard time implementing the radix-2^{128} arithmetic in such a way that all intermediate results are kept in register values. This means we had to move values in- and out of memory which in turn led to an increase of instructions and cycle count. This can be observed in Table 4 which shows that the implementation of this approach is slightly slower to the one used in [19]. When implemented on a platform with sufficient registers this approach should be at least as efficient as the approach which uses $B = 1$.

An immediate optimization based on the idea from Example 1 is to compute the Montgomery reduction for this particular modulus as

$$\begin{aligned} c \leftarrow \frac{c + (\mu c \bmod R)m}{R} &= \frac{c + (c \bmod 2^{64})(2^{372}3^{239} - 1)}{2^{64}} \\ &= c_0(2^{308}3^{239}) + \sum_{i=1}^{23-j} c_i 2^{64(i-1)} \\ &= 2^{256}2^{52}c_0 3^{239} + \sum_{i=1}^{23-j} c_i 2^{64(i-1)}. \end{aligned}$$

This process is repeated 12 times (for $j = 0$ to 11) and the input c is overwritten as the output for the next iteration. The advantage from this approach is that multiplying with $3^{239} < 2^{6 \cdot 64}$ reduces the number of multiplication instructions (as explained in Section 3.4). The price to pay is the additional shift of 52 bits (the multiplication with 2^{52}). This approach lowers the number of required multiplication instruction by a factor $6/7$ and this results in a performance increase of over five percent (see Table 4).

Finally, we implemented arithmetic modulo $2^{391}19^{88} - 1$: the prime that is the result of our search for SIDH-friendly primes. The number of multiplications required is exactly the same as for the "shifting" approach but it avoids the computation of the shift operation.

$$\begin{aligned} c \leftarrow \frac{c + (\mu c \bmod R)m}{R} &= \frac{c + (c \bmod 2^{64})(2^{391}19^{88} - 1)}{2^{64}} \\ &= c_0(2^{327}19^{88}) + \sum_{i=1}^{23-j} c_i 2^{64(i-1)} \\ &= 2^{320}c_0(2^{7}19^{88}) + \sum_{i=1}^{23-j} c_i 2^{64(i-1)}. \end{aligned}$$

Since, where the multiplication by 2^{52} is computed using shifts, the multiplication by 2^{320} is a straight-forward relabeling of the indices of the 64-bit digits. This simplifies the code and Table 4 summarizes the reduction of the total number of instructions required for the implementation. Moreover, this immediately results in an almost 12% speed-up in the modular reduction routine.

4 Curve Arithmetic using Addition-Subtraction Chains

The main computations in SIDH are ellitpic curve scalar multiplications of powers of 2 and p and evaluations of isogenies using Vélu's formula [54]. In this section we investigate if curve models other than Montgomery curves can be used to speed-up the elliptic curve scalar multiplication.

We are motivated by the fact that asymptotically, when the bit-length of the scalar goes to infinity, for random scalars the usage of twisted Edwards curves is faster compared to Montgomery curves. This is the case since one can use large window sizes when computing the scalar multiplication with twisted Edwards curves: something which is not possible with the differential addition law of Montgomery curves (see Section 2.2 and the counts in Table 1). We are aware of the y-only efficient differential formulas for twisted Edwards curves (see [26], [14], [32]) and their potential application to SIDH [44] but since these are comparatively slower than the differential addition law of Montgomery curves and our addition-subtraction chains do not work

TABLE 5
The cost of different scalar multiplications expressed in multiplications ($\mathbf{M}$) and squarings ($\mathbf{S}$) in $\mathbf{F}_{q^2}$ as well as the number of multiplications per bit of the scalar.

operation	cost in $\mathbf{F}_{q^2}$	$\mathbf{M}$/bit
Montgomery triple (3)	$7\mathbf{M} + 3\mathbf{S}$	6.78
Montgomery quintuple (5)	$11\mathbf{M} + 7\mathbf{S}$	7.00
Montgomery septuple (7)	$15\mathbf{M} + 9\mathbf{S}$	7.75
twisted Edwards double (2)	$4\mathbf{M} + 4\mathbf{S}$	7.01
twisted Edwards triple (3)	$10\mathbf{M} + 3\mathbf{S}$	7.73
twisted Edwards quantuple (5)	$18\mathbf{M} + 8\mathbf{S}$	10.34
twisted Edwards septuple (7)	$25\mathbf{M} + 7\mathbf{S}$	10.42

with a differential group law this is out of the scope for our investigation here. More specifically, for a fixed scalar p^z, for some integer z such that $1 \leq z \leq y$, we search for the *optimal* addition-subtraction chain (in terms of arithmetic in $\mathbf{F}_{q^2}$ see Section 4.2) which can be computed using the group law on twisted Edwards curves and check if this outperforms the differential approach on using Montgomery curves. The impact on the computation of the isogenies when using this different curve model is left as future work but as outlined in [18] this computation is practical.

4.1 Outperforming Montgomery Curves?

Before starting to describe our approach to search for efficient addition-subtraction chains let us take a step back and investigate if twisted Edwards curves can outperform Montgomery curves in the setting of SIDH. The extremely efficient double and triple formulas for Montgomery curves, which are used to compute elliptic curve scalar multiplication of 2^x and 3^y in practical implementations, seem hard to outperform with another curve model. In order to quantify this we ran some experiments with the optimized implementation of the SIKE protocol [3]: on various x86_64 architectures the ratio between the time to compute a modular squaring and a modular multiplication in $\mathbf{F}_{q^2}$ is 0.75 (for other ratios the discussion below can be adjusted accordingly). Table 5 summarizes the cost to compute certain scalar multiples for Montgomery curves and twisted Edwards curves when using the counts given in Table 1. The cost is expressed in multiplications (in $\mathbf{F}_{q^2}$) per bit of the scalar.

From Table 5 it becomes clear that there is indeed no hope for twisted Edwards curves to outperform Montgomery curves when $p \in \{3, 5\}$. Even if one would only use twisted Edwards point doublings to compute the required bit-length the cost would be higher than repeatedly applying the triple or quintuple cost for Montgomery curves.

However, motivated by our findings in Section 3, where we enhance the performance of the arithmetic in $\mathbf{F}_{q^2}$ by selecting a modulus with $p > 5$, and the results from [18], which show how to compute isogenies with such modified moduli, we think that a systematic search for optimal addition-subtraction chains is an interesting alternative route to look for optimization potential for SIDH. This approach is outlined below.

4.2 Finding Addition-Subtraction Chains

To optimize the elliptic curve scalar multiplications with a prime $\ell > 2$ the idea is to search for fast addition-subtraction chains for powers of ℓ. The expectation is that for large scalars (e.g. larger powers of ℓ) better chains can be found; better is measured as lowering the number of multiplications in $\mathbf{F}_{q^2}$ per bit of the scalar to compute the various elliptic curve group operations in the addition-subtraction chain.

Algorithm 1 Recursive approach "brute(chain,index)" to find an addition-subtraction chain with restrictions on the length and the number of additions / subtractions used.

Input:
- **Global**.
- target $\in \mathbf{Z}$, target number,
- $c \in \mathbf{Z}$, threshold which determines the maximum number of steps,
- $\text{max}_{\text{steps}} \in \mathbf{Z}$, maximum number of allowed steps in the chain
- $\text{max}_{\text{nadd}} \in \mathbf{Z}$, maximum number of allowed adds/subs in the chains,
- **Local**.
- $i \in \mathbf{Z}$, current position in the chain,
- chain, chain of length i.

Output: Store all chains found.

1: **if** $\text{chain}_{i-1} = \text{target}$ **then**
2: $\quad \text{max}_{\text{steps}} \leftarrow (i-1) + c$
3: $\quad$ Output or store chain: $\{\text{chain}_0, \ldots, \text{chain}_{i-1}\}$
4: $\quad$ **return**
5: **end if**
6: **if** $i \geq \text{max}_{\text{steps}}$ or $\text{nadd} > \text{max}_{\text{nadd}}$ **then**
7: $\quad$ **return**
8: **end if**
9: $\text{chain}_i \leftarrow 2 \cdot \text{chain}_{i-1}$
10: brute(chain, $i+1$)
11: **if** $\text{nadd} \leq \text{max}_{\text{nadd}}$ **then**
12: $\quad \text{nadd} \leftarrow \text{nadd} + 1$
13: $\quad$ **for** $j \leftarrow 0$ until $j < i - 1$ **do**
14: $\quad\quad \text{chain}_i \leftarrow \text{chain}_{i-1} + \text{chain}_j$
15: $\quad\quad$ brute(chain, $i+1$)
16: $\quad\quad$ **if** $\text{chain}_{i-1} > \text{chain}_j$ **then**
17: $\quad\quad\quad \text{chain}_i \leftarrow \text{chain}_{i-1} - \text{chain}_j$
18: $\quad\quad\quad$ brute(chain, $i+1$)
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: $\quad \text{nadd} \leftarrow \text{nadd} - 1$
22: **end if**

4.2.1 Generating Chains

The search for such addition-subtraction chains is done with a recursive algorithm which iterates over all possible chains. This approach is outlined in Algorithm 1. However, in order to speed-up this search one can set global parameters which control the maximum number of steps in the chain as well as the maximum number of additions/subtractions. The reasoning here is that most likely shorter chains, i.e. lower number of elliptic curve group operations, also result in a lower number of arithmetic operations in $\mathbf{F}_{q^2}$. The same is true for the threshold on the number of additions and subtractions in the chain: overall it is expected that the majority of the steps are double operations and only a couple of other operations are required. In order to be more flexible this threshold can be increased with the global c

TABLE 6
Overview of the best addition-subtraction chains found for usage with twisted Edwards curves (top part of the table) and the differential addition chains for usage with Montgomery curves. The cost is expressed in group operation as explained in the text and in multiplications in $\mathbf{F}_{q^2}$ where we assume that the ratio between the cost of a modular squaring and a modular multiplication is 0.75.

scalar	chain	eAp	eAe	pDp	pDe	pTp	pTe	$\#s$	$\#\mathbf{M}$	$\#\mathbf{M}$/bits
3^x	3^x	0	0	0	0	x	0	1	$12.25 \cdot x$	7.73
5^1	2^2+1	1	0	1	1	0	0	2	24	10.34
5^2	$2^2 \cdot 3 \cdot 2 + 1$	1	0	2	1	1	0	2	43.25	9.31
5^3	$(2^2+1) \cdot 2^2 \cdot 3 \cdot 2 + 5$	1	1	3	2	1	0	2	68.25	9.80
5^4	$(2 \cdot 3 \cdot 2 + 1) \cdot 2^3 \cdot 3 \cdot 2 + 1$	2	0	4	2	2	0	2	86.5	9.31
5^5	$((2^2+1) \cdot 2 \cdot 3 \cdot 2 + 5) \cdot 2^3 \cdot 3 \cdot 2 + 5$	2	1	5	1	2	0	2	111.5	9.60
5^6	$((2^5-1) \cdot 2^6 + 31) \cdot 2^3 + 1$	2	1	11	3	0	0	3	129	9.26
7^1	$3 \cdot 2 + 1$	1	0	0	1	1	0	2	29.25	10.42
7^2	$2^3 \cdot 3 \cdot 2 + 1$	1	0	3	1	1	0	2	50.25	8.95
7^3	$((2^3 \cdot 3 \cdot 2 + 1) \cdot 3 \cdot 2 + 49)$	1	1	3	2	2	0	2	80.5	9.56
7^4	$(2^2 \cdot 3 \cdot 2 + 1) \cdot 2^4 \cdot 3 \cdot 2 + 1$	2	0	6	2	2	0	2	100.5	8.95
11^1	$2 \cdot 3 \cdot 2 + 1$	1	0	1	1	1	0	2	36.25	10.48
11^2	$(2^2+1) \cdot 2^2 \cdot 3 \cdot 2 + 1$	2	0	3	2	1	0	2	67.25	9.72
11^3	$(((2^6-1)+64) \cdot 2 + 63) \cdot 2^2 + 63$	3	1	6	3	0	0	2	103	9.92
13^1	$2 \cdot 3 \cdot 2 + 1$	1	0	1	1	1	0	2	36.25	9.80
13^2	$(3 \cdot 2 + 1) \cdot 2^2 \cdot 3 \cdot 2 + 1$	2	0	2	2	2	0	2	72.5	9.80
13^3	$((2^4 \cdot 3 \cdot 2 - 1) + 96) \cdot 2 \cdot 3 \cdot 2 + 95$	2	1	5	2	2	0	2	103.5	9.32
17^1	$2^4 + 1$	1	0	3	1	0	0	2	38	9.30
17^2	$2^4 \cdot 3^2 \cdot 2 + 1$	1	0	4	1	2	0	2	69.5	8.50
17^3	$(2^4+1) \cdot 2^4 \cdot 3^2 \cdot 2 + 17$	1	1	7	2	2	0	2	108.5	8.85
19^1	$3^2 \cdot 2 + 1$	1	0	0	1	2	0	2	41.5	9.78
19^2	$(2^2+1) \cdot 2^2 \cdot 3^2 \cdot 2 + 1$	2	0	3	2	2	0	2	79.5	9.36
19^3	$(2^7-1) \cdot 3^3 \cdot 2 + 1$	2	0	6	2	3	0	2	112.75	8.85
scalar	chain	Differential addition chains				A	T	$\#s$	$\#\mathbf{M}$	$\mathbf{M}$/bits
3	$(1 \cdot 2 + 1)$					0	1	2	10.75	6.78
5	$(1 \cdot 2 + 1) + 2$					1	1	3	16.25	7.00
7	$(1 \cdot 2 + 1) + 1 + 3$					2	1	3	21.75	7.75
11	$(1 \cdot 2 + 1) + 1 + 3 + 4$					3	1	3	27.25	7.88
13	$(1 \cdot 2 + 1) + 2 + 3 + 5$					3	1	3	27.25	7.36
17	$(1 \cdot 2 + 1) + 1 + 3 + 3 + 7$					4	1	3	32.75	8.01
19	$(1 \cdot 2 + 1) + 2 + 2 + 5 + 7$					4	1	3	32.75	7.71

parameters as outlined in Algorithm 1. We have performed a search for various powers[1] with the following starting conditions as outlined in this table.

scalar	$\max_{\text{steps}}$	$\max_{\text{nadd}}$
3^x	$2x+3$	x
5^x	$3x+4$	x
7^x	$4x+5$	$2x$
11^x	$5x+6$	$2x$
13^x	$5x+6$	$2x$
17^x	$5x+6$	$2x$
19^x	$6x+7$	$2x$

For all computations we used $c = 1$ to increase the threshold for the maximum number of additions / subtractions by one.

4.2.2 Arithmetic Cost of Chains

After these chains have been found we have to find the "optimal" ones. This is done by expressing the cost for the chain in arithmetic operations in $\mathbf{F}_{q^2}$ according to Table 1 and selecting the one which minimizes this cost. This is not completely trivial: since efficient triple formulas are known we have to identify, in a post-processing step, triple steps in the chains produced by Algorithm 1. This has to be done carefully since these triple formulas compute $3 \cdot c$ as $2 \cdot c + c$ but the intermediate result $2 \cdot c$ is not directly stored: hence, if this is used in an addition later in the chain these more efficient formulas can not be applied.

To accurately compute the cost of an addition-subtraction chain it is assumed twisted Edwards curves are used together with extended twisted Edwards coordinates as outlined in Section 2.2. This means that one should be careful when to compute or omit the calculation of extended T coordinate since this has a direct impact on the arithmetic cost in $\mathbf{F}_{q^2}$. Let us write an elliptic curve operation O as aOb where $O \in \{\text{A}, \text{D}, \text{T}\}$ (which correspond to point addition (A), point doubling (D), and point triple (T)) and the $a, b \in \{\text{p}, \text{e}\}$ denote of the in- or output to the elliptic curve operation is in the projective coordinate ((p), no T-coordinate) or extended coordinate system ((e), with T-coordinate). This gives the following *six* options in practice: eAp, eAe, pDp, pDe, pTp, and pTe. The input to the point addition is always in extended twisted Edwards form since the T-coordinate is required, however the computation of the output T-coordinate can be omitted if needed. The point double and point triple formulas do not need the T-coordinate as input.

The post-processing scripts go over the found chains and, besides locating and merging the triple operations, determine when the T-coordinate is required and when not. It should be noted that it is not immediately clear if the result of an operation O might require the computation of

1. This computation is still running for different exponents, the results will be added to the final version of the paper.

the T-coordinate. This extended coordinate might not be immediately needed for the next step in the chain; however, when this value is later used in an point addition then this elliptic curve point should be stored together with the extended coordinate T. Hence, the situation in SIDH is different from the one in elliptic curve cryptography where in a windowing algorithm one can assume the precomputed points always are in extended form (see [29]). The usage of extended coordinates comes at a slightly higher price in SIDH. A summary of our findings is presented in Table 6. When displaying the chains the second operand to the point addition has occurred before in the computation.

4.3 Discussion of Results

As can be seen from Table 6 the cost expressed in multiplication per bit of the scalar used in the elliptic curve scalar multiplication does go down when multiplying with larger scalars (i.e. addition chains for primes raised to larger exponents). In order to make an easy comparison the lower part of Table 6 shows the optimal differential chains for usage with Montgomery curves. It should be noted that the notation for the chains is slightly different compared to the upper part of the table: this is to emphasize that when adding a term the difference has occurred in the chain as well. Moreover, the notation $(1 \cdot 2 + 1)$ means the chain starts with the computation of a triple but in contrast to the twisted Edwards formula used the value $2 \cdot 1$ *and* $3 \cdot 1$ are computed and stored which means the value 2 can be used later.

Hence, we were not able to find any addition-subtraction chains which when used in combination with the extended twisted Edwards coordinates result in a speed-up for the elliptic curve scalar multiplication in SIDH. When one simply applies the differential chains for the primes and applies them repeatedly to compute the scalar multiplication for higher powers using the efficient Montgomery arithmetic the result always outperforms more advanced addition-subtraction chains as far as we could verify. For larger numbers, addition chains will become more useful for twisted Edwards curves because of the differential restriction on Montgomery curves. However, given that chains for such large numbers are difficult to find and the generally more expensive operations it is unclear if twisted Edwards curves can become faster in this setting.

5 Conclusions and Future Work

We have studied various arithmetic properties which are useful for enhancing the performance in a recent post-quantum key encapsulation candidate based on the hardness of constructing an isogeny between two isogenous supersingular elliptic curves defined over a finite field. We have provided an overview of different techniques to compute arithmetic modulo $2^x p^y \pm 1$. Although we have surveyed this in more generality it turns out that non-interleaved Montgomery reduction which is optimized for such primes is the most efficient approach in practice. Additionally, we have identified other moduli suitable for SIDH which allow even faster implementations.

Furthermore, we have analyzed the relative costs of Montgomery curves, the current state-of-the-art curve type for SIDH, and the twisted Edwards family of curves which allows precomputing more efficient addition chains. We found multiple efficient addition-subtraction chains for the scalar powers required in the key encapsulation mechanism computation. However, based on these results we have to conclude that these more efficient chains cannot compensate for the more expensive group law in twisted Edwards curves. We are still looking for more efficient addition chains and incorporating them into the computation of the isogeny tree presents an interesting challenge. The algorithm for calculating the optimal tree given by Jao and De Feo [31] cannot be easily generalized to arbitrary steps since the problem becomes much harder.

Acknowledgements

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme Marie Skłodowska-Curie ITN ECRYPT-NET (Project Reference 643161) and Horizon 2020 project PQCRYPTO (Project Reference 645622). We would like to thank Michael Naehrig and Craig Costello for insightful discussions and useful comments on an early version of this paper.

References

[1] T. Acar and D. Shumow. Modular reduction without precomputation for special moduli. Technical report, Microsoft Research, 2010.

[2] D. Augot, L. Batina, D. J. Bernstein, J. W. Bos, J. Buchmann, W. Castryck, O. Dunkelman, T. Güneysu, S. Gueron, A. Hülsing, T. Lange, M. S. E. Mohamed, C. Rechberger, P. Schwabe, N. Sendrier, F. Vercauteren, and B.-Y. Yang. Initial recommendations of long-term secure post-quantum systems, 2015. http://pqcrypto.eu.org/docs/initial-recommendations.pdf.

[3] R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, V. Soukharev, and D. Urbanik. Supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Standardization project, 2017.

[4] R. Azarderakhsh, D. Fishbein, and D. Jao. Efficient implementations of a quantum-resistant key-exchange protocol on embedded systems. Technical report, http://cacr.uwaterloo.ca/techreports/2014/cacr2014-20.pdf, 2014.

[5] P. Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In A. M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 311–323. Springer, Heidelberg, Aug. 1987.

[6] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. In S. Vaudenay, editor, *AFRICACRYPT 08*, volume 5023 of *LNCS*, pages 389–405. Springer, Heidelberg, June 2008.

[7] D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In K. Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 29–50. Springer, Heidelberg, Dec. 2007.

[8] T. Blum and C. Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactions on Computers*, 50(7):759–764, 2001.

[9] J. W. Bos, C. Costello, H. Hisil, and K. Lauter. Fast cryptography in genus 2. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 194–210. Springer, Heidelberg, May 2013.

[10] J. W. Bos, C. Costello, H. Hisil, and K. Lauter. High-performance scalar multiplication using 8-dimensional GLV/GLS decomposition. In G. Bertoni and J.-S. Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 331–348. Springer, Heidelberg, Aug. 2013.

[11] J. W. Bos and S. Friedberger. Fast arithmetic modulo 2ˆ xpˆ y±1. In *Symposium on Computer Arithmetic – ARITH*, pages 148–155. IEEE, 2017.

[12] A. Brauer. On addition chains. *Bulletin of the American Mathematical Society*, 45:736–739, 1939.

[13] R. Bröker. Constructing supersingular elliptic curves. *J. Comb. Number Theory*, 1(3):269–273, 2009.

[14] W. Castryck, S. Galbraith, and R. R. Farashahi. Efficient arithmetic on elliptic curves using a mixed edwards-montgomery representation. Cryptology ePrint Archive, Report 2008/218, 2008. http://eprint.iacr.org/2008/218.

[15] L. Chen, S. Jordan, Y. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone. Report on post-quantum cryptography. NISTIR 8105, National Institute of Standards and Technology, 2016. http://csrc.nist.gov/publications/drafts/nistir-8105/nistir_8105_draft.pdf.

[16] C. Chuengsatiansup. *Optimizing curve-based cryptography*. PhD thesis, Technische Universiteit Eindhoven, 2017.

[17] N. Clift. Calculating optimal addition chains. *Computing*, 91(3):265–284, Mar 2011.

[18] C. Costello and H. Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In T. Takagi and T. Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 303–329. Springer, Heidelberg, Dec. 2017.

[19] C. Costello, P. Longa, and M. Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 572–601. Springer, Heidelberg, Aug. 2016.

[20] C. Costello, P. Longa, and M. Naehrig. SIDH library version 1.0. https://www.microsoft.com/en-us/download/details.aspx?id=52438, 2016.

[21] M. H. Devoret and R. J. Schoelkopf. Superconducting circuits for quantum information: an outlook. *Science*, 339(6124):1169–1174, 2013.

[22] S. R. Dussé and B. S. Kaliski Jr. A cryptographic library for the Motorola DSP56000. In I. Damgård, editor, *EUROCRYPT'90*, volume 473 of *LNCS*, pages 230–244. Springer, Heidelberg, May 1991.

[23] H. M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44:393–422, July 2007.

[24] A. Faz-Hernández, J. López, E. Ochoa-Jiménez, and F. Rodríguez-Henríquez. A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. *IEEE Transactions on Computers*, 2017.

[25] A. Flammenkamp. Integers with a small number of minimal addition chains. *Discrete mathematics*, 205(1-3):221–227, 1999.

[26] P. Gaudry and D. Lubicz. The arithmetic of characteristic 2 Kummer surfaces and of elliptic kummer lines. *Finite Fields and Their Applications*, 15(2):246–260, 2009.

[27] M. Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. http://eprint.iacr.org/2012/309.

[28] W. Hasenplaugh, G. Gaubatz, and V. Gopal. Fast modular reduction. In *IEEE Symposium on Computer Arithmetic – ARITH*, pages 225–229. IEEE, 2007.

[29] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson. Twisted Edwards curves revisited. In J. Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 326–343. Springer, Heidelberg, Dec. 2008.

[30] A. Jalali, R. Azarderakhsh, M. M. Kermani, and D. Jao. Supersingular isogeny Diffie-Hellman key exchange on 64-bit arm. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[31] D. Jao and L. D. Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In B. Yang, editor, *Post-Quantum Cryptography*, volume 7071 of *LNCS*, pages 19–34. Springer, 2011.

[32] B. Justus and D. Loebenberger. Differential addition in generalized Edwards coordinates. In I. Echizen, N. Kunihiro, and R. Sasaki, editors, *Advances in Information and Computer Security*, pages 316–325. Springer Berlin Heidelberg, 2010.

[33] A. A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. Number 145 in Proceedings of the USSR Academy of Science, pages 293–294, 1962.

[34] A. Karmakar, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Efficient finite field multiplication for isogeny based post quantum cryptography (to appear). In *Workshop on the Arithmetic of Finite Fields – WAIFI 2016*, LNCS. Springer, 2016.

[35] J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, I.-C. Hoi, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, A. N. Cleland, and J. M. Martinis. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature*, 519:66–69, 2015.

[36] M. Knezevic, F. Vercauteren, and I. Verbauwhede. Speeding up bipartite modular multiplication. In M. A. Hasan and T. Helleseth, editors, *Arithmetic of Finite Fields – WAIFI*, volume 6087 of *LNCS*, pages 166–179. Springer, 2010.

[37] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[38] C. K. Koç, T. Acar, and B. S. Kaliski Jr. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.

[39] B. Koziel, R. Azarderakhsh, D. Jao, and M. Mozaffari-Kermani. On fast calculation of addition chains for isogeny-based cryptography. In *International Conference on Information Security and Cryptology*, pages 323–342. Springer, 2016.

[40] B. Koziel, R. Azarderakhsh, M. M. Kermani, and D. Jao. Post-quantum cryptography on FPGA based on isogenies on elliptic curves. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(1):86–99, 2017.

[41] B. Koziel, R. Azarderakhsh, and M. Mozaffari-Kermani. Fast hardware architectures for supersingular isogeny Diffie-Hellman key exchange on FPGA. In *International Conference in Cryptology in India*, pages 191–206. Springer, 2016.

[42] B. Koziel, A. Jalali, R. Azarderakhsh, D. Jao, and M. Mozaffari-Kermani. NEON-SIDH: Efficient implementation of supersingular isogeny Diffie-Hellman key exchange protocol on ARM. In S. Foresti and G. Persiano, editors, *Cryptology and Network Security*, pages 88–103. Springer International Publishing, 2016.

[43] A. K. Lenstra. Generating RSA moduli with a predetermined portion. In K. Ohta and D. Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 1–10. Springer, Heidelberg, Oct. 1998.

[44] M. Meyer, S. Reith, and F. Campos. On hybrid SIDH schemes using Edwards and Montgomery curve arithmetic. Cryptology ePrint Archive, Report 2017/1213, 2017. https://eprint.iacr.org/2017/1213.

[45] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[46] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[47] D. Moody. Post-quantum cryptography: NIST's plans for the future. Presentation at PKC 2016, http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/pqcrypto-2016-presentation.pdf, 2016.

[48] F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Informatique Thèorique et Applications/Theoretical Informatics and Applications*, 24:531–544, 1990.

[49] M. Mosca. Cybersecurity in an era with quantum computers: Will we be ready? Cryptology ePrint Archive, Report 2015/1075, 2015. http://eprint.iacr.org/2015/1075.

[50] H. Orup. Simplifying quotient determination in high-radix modular multiplication. In *Symposium on Computer Arithmetic – ARITH*, pages 193–199. IEEE, 1995.

[51] A. Scholz. Aufgabe 253. *Jahresbericht der deutschen Mathematiker-Vereingung*, 47:41–42, 1937.

[52] M. Shand and J. Vuillemin. Fast implementations of RSA cryptography. In E. E. Swartzlander Jr., M. J. Irwin, and G. A. Jullien, editors, *11th Symposium on Computer Arithmetic*, pages 252–259. IEEE Computer Society, 1993.

[53] J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Gradute Texts in Mathematics*. Springer-Verlag, 1986.

[54] J. Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sc. Paris.*, 273:238–241, 1971.

[55] C. D. Walter. Montgomery exponentiation needs no final subtractions. *Electronics Letters*, 35:1831–1832, 1999.