Fast Arithmetic Modulo $2^x p^y \pm 1$

Joppe W. Bos[1] and Simon Friedberger[1,2]

[1] NXP Semiconductors
[2] KU Leuven - iMinds - COSIC

Abstract. We give a systematic overview of techniques to compute efficient arithmetic modulo $2^x p^y \pm 1$. This is useful for computations 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. One of the main computational bottlenecks in this key-exchange protocol is computing modular arithmetic in a finite field defined by a prime of this special shape.
Recent implementations already use this special prime shape to speed up the cryptographic implementations but it remains unclear if the choices made are optimal or if one can do better. Our overview shows that in the SIDH setting, where arithmetic over a quadratic extension field is required, the approaches based on Montgomery multiplication are to be preferred. Furthermore, the outcome of our search reveals that there exist moduli which result in even faster implementations.

1 Introduction

Over the last years there have been significant advancements in quantum computing [19,30]. This has significantly accelerated the research into *post-quantum* cryptographic schemes [36] which can be used as drop-in replacements for the current classical public-key cryptographic primitives. This demand is driven by the interest from standardization bodies such as the National Institute of Standards and Technology [15] (which even put forward a call for proposals for new public-key cryptographic standards [35]) and the European Union's prestigious PQCrypto research effort [2].

One relatively recent approach to realize post-quantum public-key cryptography is based on the hardness of constructing an isogeny between two isogenous supersingular elliptic curves defined over a finite field where the degree of the isogeny is smooth and known and was introduced in 2011 [27]. This problem was used to define a supersingular isogeny Diffie-Hellman (SIDH) key-exchange protocol. Recently, the authors of [17] have improved the performance of SIDH significantly and demonstrated the potential of this post-quantum key-exchange approach by providing a fast constant-time software implementation. The initial results look promising especially with respect to the key sizes. The public key is 564 bytes for 128-bit post-quantum security. However, more research is required in terms of cryptanalysis to gain more confidence. One step in this direction has recently been taken by Galbraith et al. [21] who describe a general active attack against the static-key variant of the SIDH key-exchange protocol. However, the approach from [17] is not subject to this type of attack since it describes an ephemeral key exchange. Another direction where more research is required is to study the various algorithms and techniques to translate the SIDH approach to an efficient cryptographic implementation.

This paper is concerned with the latter research direction, more specifically investigating the main computational bottleneck: computing arithmetic modulo $m = 2^x \cdot p^y \pm 1$ where p is a small prime such that $2^x \approx p^y$ (and $m \approx 2^{768}$ to guarantee 128-bit post-quantum security). The cryptographic research area which studies efficient modular arithmetic and its

implementations is vast and ranges from designing methods which can run more efficiently using the single instruction, multiple data programming paradigm (e.g. [12,23,13,37]) to using the single instruction multiple threads paradigm (e.g. [6,25]) to modular arithmetic methods designed for a specific family of moduli (e.g. [8,22,16] but see also the references given in Section 3).

This work is about optimizing various modular multiplication and reduction techniques for a modulus of the form $2^x \cdot p^y \pm 1$. We study the application of Barrett reduction [7] as used in the first implementation of SIDH [3], Montgomery reduction [34] as applied by the more recent implementations presented in [17,4,5] and approaches which use a radix-system which is directly derived from the prime shape similar to the approach presented in [29]. This paper gives a systematic overview of the different optimizations one can apply when using the different modular reduction algorithms and in which setting they make sense (e.g. computing arithmetic in the finite field $\mathbf{F}_m$ or in an extension field thereof) with respect to an efficient implementation.

Although the Montgomery modular multiplication [34] approach from [17] demonstrates that the number of *multiplication instructions* per modular reduction can be reduced significantly when exploiting the special prime shape, we show how this can be improved even further for the specific prime used in their implementation (cf. Section 5). In the non-interleaved setting, which is the one considered for SIDH implementations, we introduce a competitive reduction method based on Barrett reduction [7] which uses a folding technique [26] which is slightly slower compared to the special Montgomery multiplication. Finally, we have conducted a search for more suitable SIDH-friendly prime moduli of the form $2^x p^y \pm 1$ which result in better performance. In Section 5 we show our implementation results and compare this to the current state-of-the-art. We plan to make all our source code freely and publicly available soon.

2 Preliminaries

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 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 $r = 2^w$ (e.g. 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 $\mathrm{mul} : \mathbf{Z}_r \times \mathbf{Z}_r \to \mathbf{Z}_{r^2}$ where $\mathrm{mul}(x, y) = x \cdot y$. 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 sizes residues which are used in the

current SIDH schemes. We verify this assumption by comparing to implementation results in Section 5.

2.1 Montgomery reduction

The idea behind Montgomery reduction [34] 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 [38,39].

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 [20] 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.

2.2 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 [7]. 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). 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

$$q_1 = \left\lfloor \frac{c \cdot \mu}{r^{2n}} \right\rfloor = \left\lfloor \frac{c}{r^{2n}} \cdot \left\lfloor \frac{r^{2n}}{m} \right\rfloor \right\rfloor. \tag{3}$$

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

$$q_2 = \left\lfloor \frac{\lfloor c/r^{n-1} \rfloor \cdot r^{n-1} \cdot \mu}{r^{2n}} \right\rfloor = \left\lfloor \frac{\hat{c} \cdot r^{n-1} \cdot \mu}{r^{2n}} \right\rfloor = \left\lfloor \frac{\hat{c} \cdot \mu}{r^{n+1}} \right\rfloor. \tag{4}$$

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} in Eq. 4 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

$$\underbrace{(n+1)^2 - \sum_{i=1}^{n-1} i}_{\text{for } \hat{c} \cdot \mu} + \underbrace{\sum_{i=1}^{n} i + n}_{\text{for } q_2 \cdot m} = (n+1)^2 + 2n \tag{5}$$

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.

Reducing arbitrary length input. Barrett reduction is typically analyzed for an input c which is bounded above by r^{2n} and a modulus $m < r^n$. Let us 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 $0 \leq a < r^{\ell_a}$ and $0 \leq b < r^{\ell_b}$ 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 $(0 \leq c < r^\ell)$ by m $(0 \leq m < r^n)$ using Barrett reduction as

$$\text{CostBarrett}(\ell, n) = \mathcal{H}(\ell - n + 1, \ell - n + 1, \ell - n + 1) + \mathcal{L}(\ell - n + 1, n, n + 1). \quad (6)$$

Note that CostBarrett$(2n, n) = (n+1)^2 - \sum_{i=0}(i+1) + n + \sum_{i=0}^{n-1}(i+1)$ which equals Eq. (5) as expected.

Folding. An optimization to Barrett reduction which needs additional precomputation but reduces the number of multiplications is called *folding* [26]. 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 [26] where $D = 2N$ and $x = 3N/2$ is used such that c is reduced from $2N$ bits to at most $1.5N + 1$ bits.

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 rational 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 $(f \cdot q^x \cdot p^y)^2$ [14] which is used in SIDH. For efficiency reasons (as will become clear in this section) it makes sense to fix q to 2. In practice, most instantiations use $q = 2$ and $p = 3$. In order to be generic we leave the parameter p but fix q to 2. Moreover, we assume the cofactor $f = 1$ to simplify the explanation: our methods can be immediately generalized for non-trivial values of the cofactor f.

In this section we survey different approaches to optimize arithmetic modulo $m = 2^x p^y \pm 1$ where p is an odd small prime. The 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 realize 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 their individual dis-/advantages. 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 $i^2 = -1$. 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 $c_0 = a_0 b_0 - a_1 b_1$ and $c_1 = a_0 b_1 + a_1 b_0$. This can be computed using four interleaved modular multiplications or four multiplications and two modular reductions. When using Karatsuba multiplication [28] this can be reduced to three multiplications and two modular reductions by computing c_1 as $(a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$. In the interleaved setting this can be computed with three modular multiplications while in the non-interleaved setting computing three multiplications and two modular reductions suffices. 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 except when the interleaved approach significantly outperforms the separated technique.

In this section we describe techniques to speed-up modular reduction and interleaved modular multiplication plus reduction (which is of independent interest) when using the shape of $2^x p^y \pm 1$.

3.1 Using Barrett reduction

The first implementation of SIDH [3] uses Barrett reduction (see Section 2.2) 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 2.2 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 $\mu = \lfloor \frac{r^{2n}}{m} \rfloor = \lfloor \frac{r^{2n}}{2^x p^y \pm 1} \rfloor$ 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 (re)discovered multiple times, that Montgomery multiplication *can* benefit (in terms of lowering the number of multiplications) from a modulus of a specific form (cf. e.g., [33,1,31,24,10,11]). 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 exactly the approach used by the authors of [17,4] in their high-performance SIDH implementation. They use the non-interleaved approach for Montgomery multiplication using the so-called product scanning technique [32], which was introduced in [20], where all the multiplications by μ are eliminated. We describe this approach and some variants below in detail. We note that in the hardware implementation described in [5] a high-radix variant of the Montgomery multiplication suitable for hardware architectures [9] is used.

Interleaved Montgomery multiplication. As described in Section 2.1 the interleaved Montgomery multiplication approach interleaves the computation of the product and the modular reduction. Let us describe an optimization when computing multiplication 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 $R = 2^{Bw}$. Now, the multiplication with the precomputed constant μ vanishes 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

$$\begin{aligned} c \leftarrow \frac{c + a_i b + (\mu(c + a_i b) \bmod R)m}{R} &= \frac{\sum_{i=0}^{n} d_i R^i + d_0(2^x p^y - 1)}{2^{Bw}} \\ &= \frac{\sum_{i=1}^{n} d_i R^i + d_0 2^x p^y}{2^{Bw}} \\ &= \sum_{i=1}^{n} d_i R^{i-1} + d_0 2^{x-Bw} p^y \end{aligned} \tag{7}$$

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

The computation cost expressed in the number of multiplication instructions of Eq. (7) is

$$\left\lceil \frac{N}{Bw} \right\rceil \left(B \left\lceil \frac{N}{w} \right\rceil + B \left(\left\lceil \frac{N}{w} \right\rceil - B \right) \right). \tag{8}$$

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. (7) 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 [17]. This is the setting where $p = 3$, $x = 372$, $y = 239$, $N = 751$, and $\mu = 1$. Assume we target a 64-bit platform (which means $w = 64$ and $n = 12$), then 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. (8)

B	Eq. (8) (#mul instructions)
$B = 1$	276
$B = 2$	264
$B = 3$	252
$B = 4$	240
$B = 5$	285

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 $B \in \{1, 2, 3, 4\}$. However, 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 $n = \left\lceil \frac{N}{w} \right\rceil = 12$ (the number of 64-bit digits of m).

The authors of [17] 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 then the total cost of the reduction can be reduced using this technique from $n^2 + n$ to $\frac{1}{2}n^2$ multiplication instructions (a performance increase by more than a factor two) assuming n is even.

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. (7)) 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: this means computing more additions while the number of multiplications remains unchanged compared to the modular reduction as outlined for the interleaved Montgomery approach ($\frac{n^2}{2}$ multiplication instructions).

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 [29] suggested in the setting of designing a hardware implementation. Let us summarize the approach here and provide a detailed analysis where we assume that the radix r is the same as the word-size of the target architecture and $r^{n-1} \leq m < r^n$. 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 [29] 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

$$\begin{aligned} c_2 \cdot R^2 + c_1 \cdot R + c_0 = {} & ((a_2 b_0 + a_1 b_1 + a_0 b_2) \bmod 2) \cdot R^2 + \\ & \left(\left\lfloor \frac{a_2 b_1 + a_1 b_2}{2} \right\rfloor + (a_1 b_0 + a_0 b_1) \right) \cdot R + \\ & \Big((2^{-2} \bmod m) a_2 b_2 + a_0 b_0 + \\ & ((a_2 b_1 + a_1 b_2) \bmod 2) \cdot \frac{R}{2} + \left\lfloor \frac{a_2 b_0 + a_1 b_1 + a_0 b_2}{2} \right\rfloor \Big). \end{aligned}$$

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 with two calls to a Barrett reduction which takes advantage of the fact that the divisions by $2^{\frac{x}{2}}$ can be done more efficiently. Assume c_1 and c_0 each fit in n computer words (for simplicity assume $n \equiv 0 \bmod 4$) then computing a Barrett reduction of c_0 (or c_1) by $R = 2^{\frac{x}{2}} 3^{\frac{y}{2}}$ requires $\text{CostBarrett}(n, \frac{n}{2}) = \frac{n^2}{4} + 2n + 1$ multiplication instructions (see Section 2.2). However, since

the computation of the quotient and the remainder when dividing by $2^{\frac{x}{2}}$ comes essentially for free (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) then 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 [28] 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.

Let us present a different method inspired by this approach for moduli of the form $m = 2^x p^y - 1$. For simplicity, assume that both x and y are even (although this can be made to work for odd values as well) when using 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

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

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 $3\frac{n^2}{4} + 3 \cdot \text{CostBarrett}(\frac{3n}{4}, \frac{n}{4}) = \frac{39}{32}n^2 + \frac{39}{8}n + 3$ multiplication instructions and significantly fewer additions compared to the approach from [29]. The main difference with the approach presented in [29] 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 [29]. 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.

A non-interleaved approach. Both interleaved approaches as presented do not compete with the non-interleaved approaches when arithmetic in quadratic extension field is required as used in SIDH. Let us explore the possibility to use 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

$$\begin{aligned} c' &\equiv c_1 \cdot R + c_0 \\ &\equiv c_1(m+1) + c_0 \\ &\equiv c_1 + c_0 \pmod m \end{aligned} \tag{9}$$

Table 1. 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 [34]	interleaved	generic	$2n^2 + n$
	interleaved	$2^x p^y - 1$	$\frac{3}{2}n^2$
use radix directly [29]	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 [7]	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 [34]	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	$2^x p^y - 1$	$\frac{1}{2}n^2 + \frac{5}{4}n + 1$

where $0 \leq c' < 2R$. Hence, the main computational complexity is when computing 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 $\text{CostBarrett}(2n, n) = n^2 + 4n + 1$ multiplication instructions. By simplifying and doing the division by 2^x separately we obtain a method which needs $\text{CostBarrett}(\frac{3}{2}n, \frac{1}{2}n) = \frac{5}{8}n^2 + \frac{13}{4}n + 1$ 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 2.2). 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 $\text{CostBarrett}(n, \frac{1}{2}n) = \frac{n^2}{4} + 2n + 1$ 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 1 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 1 it becomes clear that the approach from [29] 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 prefers the non-interleaved setting 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) then 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.

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.

Let us fix the word size of the target platform to 64 bits in the following discussion. This can be adjusted for smaller architectures when needed. Hence, we expect a modulus of (around) $n = \frac{2 \cdot 384}{64} = 12$ computer words. Table 1 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.

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. [5] for their FPGA implementation are out of scope in this work.

The modulus $m_1 = 2(2^{386}3^{242}) - 1$. This modulus was proposed in the first implementation paper of SIDH [3] and used in the implementations presented in [3,29]. 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 1) for an odd n.

The modulus $m_2 = 2^{372}3^{239} - 1$. This modulus is proposed in [17] and used in the implementations [17,4]. 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. (7)) 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 1 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 [17]

Alternative moduli. We searched for alternative implementation- and SIDH-friendly prime moduli using constraints which enhance the practical performance when using the fastest

Table 2. SIDH-friendly prime moduli which target 128-bit post-quantum security.

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

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 2 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 $n^2/2 = 72$.

5 Benchmarking

Table 1 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 [17] (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

Table 3. 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	#mul	#add	#mov	#other
$2^{372}3^{239} - 1$ ($B = 1$) [17]	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

stamp counter using the `rdtsc` instruction. More 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 3. 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 [17] which can be found online [18]. 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 3 and discussed in Section 4 the implementation from [17] 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 3 which shows that the implementation of this approach is slightly slower to the one used in [17]. When implemented on a platform where there are sufficient registers then 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

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

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 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 3).

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.

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

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 3 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.

6 Conclusions and Future Work

We have provided an overview of different techniques to compute arithmetic modulo $2^x p^y \pm 1$. Primes of this shape are of interest to instantiate a recent post-quantum key-exchange candidate based on the hardness of constructing an isogeny between two isogenous supersingular elliptic curves defined over a finite field. Although we have surveyed this in more generality it turns out that non-interleaved Montgomery reduction which is optimized for such primes is the best approach. We have identified faster ways of implementing the modular reduction for a recently proposed prime. Moreover, as the outcome of our search we have found better SIDH-friendly moduli which lead to faster implementations.

In this work we have primarily focused on arithmetic modulo $2^x p^y \pm 1$. It would be interesting to study if other prime shapes can be used in SIDH schemes and if such primes have additional benefits. Moreover, the applicability to implement the modular reduction algorithm in parallel (e.g. using SIMD instructions) is not considered and we leave this as future work. Another aspect is to study the arithmetic at the elliptic curve and the isogeny level: both aspects were left out of scope of this work.

Acknowledgments

This work is supported in part by the Commission of the European Communities through the Horizon 2020 program under project number 645622 (PQCRYPTO) and 643161 (ECRYPT-NET). We would like to thank Michael Naehrig for his comments on an earlier version of this paper.

References

1. T. Acar and D. Shumow. Modular reduction without pre-computation 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, 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.
4. R. Azarderakhsh, B. Koziel, A. Jalali, M. M. Kermani, and D. Jao. NEON-SIDH: Efficient implementation of supersingular isogeny Diffie-Hellman key-exchange protocol on ARM. Cryptology ePrint Archive, Report 2016/669, 2016. `http://eprint.iacr.org/2016/669`.
5. R. Azarderakhsh, B. Koziel, S. H. F. Langroudi, and M. M. Kermani. FPGA-SIDH: High-performance implementation of supersingular isogeny Diffie-Hellman key-exchange protocol on FPGA. Cryptology ePrint Archive, Report 2016/672, 2016. `http://eprint.iacr.org/2016/672`.
6. J. Bajard, L. Didier, and P. Kornerup. An RNS montgomery modular multiplication algorithm. *IEEE Trans. Computers*, 47(7):766–776, 1998.
7. 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.
8. D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, Apr. 2006.
9. T. Blum and C. Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactions on Computers*, 50(7):759–764, 2001.
10. 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.
11. 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.
12. J. W. Bos, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery. Efficient SIMD arithmetic modulo a Mersenne number. In E. Antelo, D. Hough, and P. Ienne, editors, *IEEE Symposium on Computer Arithmetic – ARITH-20*, pages 213–221. IEEE Computer Society, 2011.
13. J. W. Bos, P. L. Montgomery, D. Shumow, and G. M. Zaverucha. Montgomery multiplication using vector instructions. In T. Lange, K. Lauter, and P. Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 471–489. Springer, Heidelberg, Aug. 2014.
14. R. Bröker. Constructing supersingular elliptic curves. *J. Comb. Number Theory*, 1(3):269–273, 2009.
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. Costello and P. Longa. Four$\mathbb{Q}$: Four-dimensional decompositions on a $\mathbb{Q}$-curve over the Mersenne prime. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 214–235. Springer, Heidelberg, Nov. / Dec. 2015.
17. 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.
18. C. Costello, P. Longa, and M. Naehrig. SIDH library version 1.0. `https://www.microsoft.com/en-us/download/details.aspx?id=52438`, 2016.
19. M. H. Devoret and R. J. Schoelkopf. Superconducting circuits for quantum information: an outlook. *Science*, 339(6124):1169–1174, 2013.
20. 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.
21. S. D. Galbraith, C. Petit, B. Shani, and Y. B. Ti. On the security of supersingular isogeny cryptosystems. Asiacrypt 2016 (to appear), 2016.
22. R. Granger and M. Scott. Faster ECC over $\mathbb{F}_{2^{521}-1}$. In J. Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 539–553. Springer, Heidelberg, Mar. / Apr. 2015.
23. S. Gueron and V. Krasnov. Software implementation of modular exponentiation, using advanced vector instructions architectures. In F. Özbudak and F. Rodríguez-Henríquez, editors, *Workshop on Arithmetic of Finite Fields – WAIFI*, volume 7369 of *LNCS*, pages 119–135. Springer, 2012.

24. M. Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. http://eprint.iacr.org/2012/309.
25. O. Harrison and J. Waldron. Efficient acceleration of asymmetric cryptography on graphics hardware. In B. Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 350–367. Springer, Heidelberg, June 2009.
26. W. Hasenplaugh, G. Gaubatz, and V. Gopal. Fast modular reduction. In *IEEE Symposium on Computer Arithmetic – ARITH*, pages 225–229. IEEE, 2007.
27. 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.
28. 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.
29. 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.
30. 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.
31. 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.
32. C. K. Koç, T. Acar, and B. S. Kaliski Jr. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.
33. 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.
34. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.
35. 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.
36. 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.
37. H. Seo, Z. Liu, J. Großschädl, J. Choi, and H. Kim. Montgomery modular multiplication on ARM-NEON revisited. In J. Lee and J. Kim, editors, *ICISC 14*, volume 8949 of *LNCS*, pages 328–342. Springer, Heidelberg, Dec. 2015.
38. 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.
39. C. D. Walter. Montgomery exponentiation needs no final subtractions. *Electronics Letters*, 35:1831–1832, 1999.