Efficient Finite Field Multiplication for Isogeny Based Post Quantum Cryptography

Angshuman Karmakar[1], Sujoy Sinha Roy[1], Frederik Vercauteren[1,2], and Ingrid Verbauwhede[1]

[1]KU Leuven ESAT/COSIC and iMinds
Kasteelpark Arenberg 10 bus 2452, B-3001 Leuven-Heverlee, Belgium
[2]Open Security Research, Fangda 704, Kejinan-12th, Nanshan, 518000 Shenzhen, China
Angshuman.Karmakar@esat.kuleuven.be, sujoy.sinharoy@esat.kuleuven.be, frederik.vercauteren@gmail.com, Ingrid.Verbauwhede@esat.kuleuven.be

Abstract. Isogeny based post-quantum cryptography is one of the most recent addition to the family of quantum resistant cryptosystems. In this paper we propose an efficient modular multiplication algorithm for primes of the form $p = 2 \cdot 2^a 3^b - 1$ with b even, typically used in such cryptosystem. Our modular multiplication algorithm exploits the special structure present in such primes. We compare the efficiency of our technique with Barrett reduction and Montgomery multiplication. Our C implementation shows that our algorithm is approximately 3 times faster than the normal Barrett reduction.

Keywords: Modular multiplication, Isogeny, Post-quantum cryptography

1 Introduction

The rapid development in the field of quantum computing has increased the possibility of practical quantum computer arriving within a few decades [19]. Using a powerful quantum computer, Shor's [2] algorithm can factor integers and can compute discrete logarithm in polynomial time. This has rendered cryptosystems such as RSA and those using elliptic curve cryptography highly vulnerable.

Due to these developments, research in post-quantum cryptography has seen a flurry of activity that resulted in many novel post-quantum cryptosystems. Though the cryptosystems based on learning with errors or LWE has gained the most interest, there exist other cryptosystems such as the McEliece cryptosystem [20], cryptosystems based on isogeny between elliptic curves [1, 15], the multivariate cryptosystem [21] etc. Many cryptographic schemes based on these primitives have been proposed which are analogous to their classical counterparts and hopefully will replace them in the near future.

A cryptosystem based on the computation of isogenies between elliptic curves was first proposed by Anton Stolbunov [15]. The security of this cryptosystem was based on the hardness of computing isogenies between ordinary elliptic

curves. The best known classical algorithm to solve this problem has exponential [13] complexity. But the work of Childs et al. [8] has shown that this problem has sub-exponential complexity on a quantum computer. Also their system was slow for practical purposes.

The isogeny based post quantum cryptosystem proposed by De Feo et al. [1] uses supersingular elliptic curves instead of ordinary elliptic curves. The authors in [1] have argued that the problem of computing isogenies between supersingular elliptic curves is quantum secure. They have also shown that their cryptosystem is many times faster than the previous system and offers post-quantum security for practical parameter sizes.

2 Motivation

The isogeny based post-quantum cryptosystem proposed by De Feo et al. [1] is based on the difficulty of computing isogenies between supersingular elliptic curves. Computing isogenies and applying them to the points of elliptic curves ultimately boils down to arithmetic operations in a finite field over which the supersingular curve is defined. In isogeny based cryptography the prime p is of the form $p = f \cdot 2^a 3^b - 1$ where f is a small number. Such a special structure of the prime is essential for the scheme. Like many other cryptosystems, isogeny based cryptosystem rely heavily on modular multiplication.
Montgomery multiplication [3] and Barrett reduction [7] are two ingenious methods to replace computationally costly divisions used in modular reduction with additional multiplications, additions, bit shifts etc. These methods tackle the costly modular multiplication quite efficiently and they can be applied for any general prime. So they are unable to exploit any special structure of the prime for even faster reduction.
Mersenne primes [5] and Pseudo-Mersenne primes [6] offer very fast reduction due to their special structure. Also the NIST-curves [22] which are used in elliptic curve cryptography frequently use fields over generalized Mersenne primes [4] for the advantage of extremely fast modular reduction. Even though the primes we discuss cannot be categorized as a Mersenne prime, generalized Mersenne prime or Pseudo-Mersenne prime, the possibility of exploiting the special structure of the prime for an efficient modular multiplication calculation is highly intriguing. The parameters a and b for the prime $p = 2 \cdot 2^a \cdot 3^b - 1$ in the isogeny based post-quantum protocol are chosen in such a way that $\log_2(2^a) \approx \log_2(3^b)$. For example, the 771-bit prime $p = 2 \cdot 2^{386} 3^{242} - 1$ is used in [1] for 128-bit security.

Our contribution. In this work we describe a fast modular multiplication algorithm for the primes used in isogeny based post-quantum cryptosystems. Our algorithm is inspired by the Barrett reduction [7] and leverages special structures of the primes used in such cryptosystems. While there are several techniques for performing efficient arithmetic in fields whose characteristic is a Mersenne prime or a Pseudo-Mersenne prime [4], we are not aware of any techniques that could accelerate modular arithmetic in finite fields of characteristic $p = f \cdot 2^a 3^b - 1$

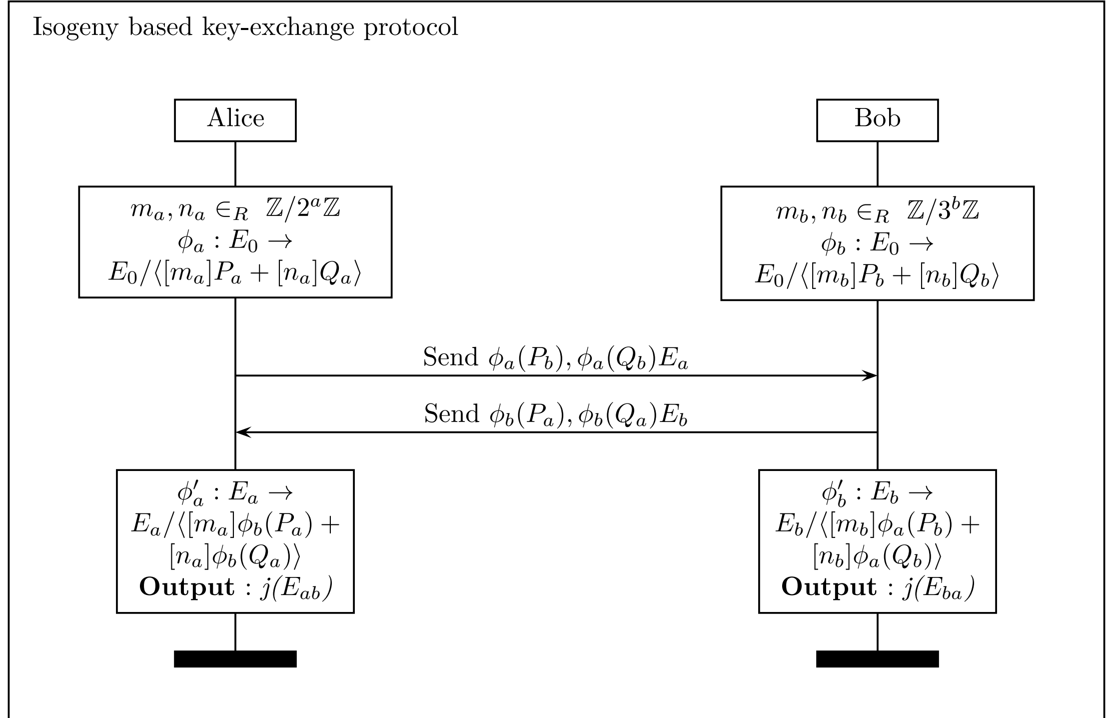
where f is a small number. In this paper we propose an efficient algorithm to perform fast modular arithmetic with primes of the form $p = 2 \cdot 2^a 3^b - 1$ with b even. Besides the new algorithm, we list a number of such primes for different security levels. These primes are listed in Appendix C.

3 Mathematical Background

In this section we will briefly describe the isogeny based key exchange protocol and then focus on efficient modular multiplication techniques. For a detailed description of isogeny based key exchange interested readers may follow [1].

3.1 Isogenies of Elliptic curves

An isogeny $\phi : E_1 \to E_2$ is a basepoint preserving, i.e. $\phi(\mathcal{O}) \to \mathcal{O}$, morphism between two elliptic curves E_1 and E_2 defined over $\mathbb{F}_q$ (Sec. III.4 in [14]). Two elliptic curves are said to be *isogenous* if there exists an isogeny between them. This is an equivalence relation and symmetry is given by the existence of a *dual isogeny*. As mentioned in [1], an isogeny class is an equivalence class under the above equivalence relation. Inside the same isogeny class the curves are either all supersingular or all ordinary curves. The post-quantum key exchange scheme by De Feo et al. in [1] uses supersingular curves.
In this key-exchange scheme the public parameters are a supersingular curve E_0 defined over a field $\mathbb{F}_{p^2}$ with $p = f \cdot 2^a 3^b \pm 1$, and bases $\{P_a, Q_a\}$ and $\{P_b, Q_b\}$ which generate the *torsion groups* $E_0[2^a]$ and $E_0[3^b]$ respectively. Alice chooses $m_a, n_a \in_R \mathbb{Z}/2^a\mathbb{Z}$ and computes the isogeny $\phi_a : E_0 \to E_a,\ E_a = E_0/\langle [m_a]P_a + [n_a]Q_a \rangle$. Alice also computes $\phi_a(P_b)$ and $\phi_a(Q_b)$ under this isogeny and sends $E_a,\ \phi_a(P_b)$, and $\phi_a(Q_b)$ to Bob. Similarly Bob chooses $m_b, n_b \in_R \mathbb{Z}/3^b\mathbb{Z}$ computes the isogeny $\phi_b : E_0 \to E_b,\ E_b = E_0/\langle [m_b]P_b + [n_b]Q_b \rangle$ and sends $E_b,\ \phi_a(P_b)$, and $\phi_a(Q_b)$ to Alice. After this Alice calculates the isogeny $\phi'_a : E_a \to E_{ab},\ E_{ab} = E_a/\langle [m_a]\phi_b(P_a) + [n_a]\phi_b(Q_a) \rangle$ and similarly Bob calculates $\phi'_b : E_b \to E_{ba}$. Bob and Alice then use their common *j-invariant* $j(E_{ab})=j(E_{ba})$ as their shared key.
The difficulty of the key-exchange scheme is based on the hardness of computing isogenies between supersingular elliptic curves. The authors in [1] have argued that the complexity of the best known algorithm [18] for solving this problem is $\sqrt[4]{p}$ using classical computers and $\sqrt[6]{p}$ using a quantum computer, where p is the characteristic of the field over which the curves are defined (more details in Section. 5 and 6 of [1]). The authors have described post quantum protocols for zero knowledge proof, key-exchange and public key cryptosystem in their paper [1]. Hash functions [17] and digital signature schemes [16] based on the isogenies have also been proposed.



3.2 Efficient modular arithmetic

In this section we describe two famous algorithms for efficient modular reductions: the Barrett reduction, and the Montgomery reduction.

Barrett Reduction: Euclid's division lemma tells us that for any two positive integers a and b there exist q and r such that $a = q \cdot b + r,\ r \in [0, b-1]$. Here of course, $a = r \pmod{b}$, but finding such q and r requires division of a by b. There exist fast methods for division by small constants [10], but in general for practical cryptographic settings, division is a computationally costly operation. For constant divisors, Barrett's reduction is a clever trick. it estimates $1/b$ to substitute division by a few multiplications and bit shifts. The $1/b$ in Barrett reduction is approximated as,

$$1/b = \frac{(2^k)/b}{b \cdot 2^k/b} = \frac{(2^k)/b}{2^k} \approx \frac{x}{2^k}$$

Usually the value of x is taken as $x = \lfloor 2^k/b \rfloor$ where the parameter k depends on a. The error e of the approximation of $1/b$ is $e = 1/b - x/2^k$. Hence, the error in approximating the quotient q is ae. As $q \in \mathbb{Z}^+$, for a correct result we require that the error in approximating q is smaller than 1. This condition is satisfied

when $k = \log_2(a)$. The Barrett reduction algorithm is shown in Algorithm 1.

Input: Two numbers a and b, parameter k, $x = \left\lfloor \frac{2^k}{b} \right\rfloor$
Output: $a \pmod b$
1 $q \leftarrow (a \times x) >> k$;
2 $r \leftarrow a - q \times b$;
3 **if** $r \geq b$ **then**
4 | $r \leftarrow r - b$
5 **end**
6 **return** r

Algorithm 1: Barrett's Reduction Algorithm

Montgomery Multiplication : Montgomery multiplication [3] is another technique used to remove the necessity of performing modular reduction after each multiplication of the field elements. To use Montgomery's technique we need a number r co-prime to the modulus p or equivalently $r \cdot r' + p \cdot p' = 1$. The values r' and p' can be calculated by the extended Euclidean algorithm [12]. Montgomery multiplication first converts the operands a and b to the Montgomery domain as $a_M = a \cdot r \pmod p$, $b_M = b \cdot r \pmod p$, the multiplication algorithm described in Algorithm 2 ensures that the product also stays in the Montgomery domain as $a_M \times b_M = c_M \pmod p = a \cdot b \cdot r \pmod p$. Also the result of addition and subtraction between operands in the Montgomery domain stays in the Montgomery domain. As the conversion to and from the Montgomery domain is a costly procedure, this technique is useful where we need many multiplications, additions or subtractions in close succession.

Input: Two numbers $a_M = a \cdot r \pmod p$ and $b_M = b \cdot r \pmod p$
Output: $c_M = a \cdot b \cdot r \pmod p$
1 $t \leftarrow a_M \cdot b_M$;
2 $c_M \leftarrow \big(t + (t \cdot p' \pmod r)) \cdot p)/r$;
3 **if** $c_M \geq p$ **then**
4 | $c_M \leftarrow c_M - p$
5 **end**
6 **return** c_M

Algorithm 2: Montogomery Multiplication

As mentioned before, the above two methods do not utilize the special structure of the primes for faster modular multiplication. In the next section we are going to describe our modular multiplication algorithm which exploits the special structure of the prime for efficient modular multiplication.

4 New Modular Multiplication Algorithm

In our method, the representation of field elements plays an important role in the efficiency of the method. We represent a field element, let's say $A \in \mathbb{F}_p$,

where $p = 2 \cdot 2^a 3^b - 1$, as

$$A = a_1 \cdot 2^a 3^b + a_2 \cdot 2^{a/2} 3^{b/2} + a_3, \quad a_1 \in [0,1],\ a_2, a_3 \in [0, 2^{a/2} 3^{b/2}) \tag{1}$$

In the above representation we have assumed that a is even. However it is not mandatory. If a is odd we can write $p = 4 \cdot 2^{a-1} 3^b - 1$. This change in the value of the cofactor (from 2 to 4) does not affect the performance of the algorithm. During the course of our modular multiplication algorithm the only significance of the value of the cofactor is to determine the value of the coefficient a_1, where we need to divide some numbers by the cofactor. As division by 4 is almost as easy as division by 2 in binary representation, the change of value of the cofactor from 2 to 4 has little impact on the performance. In case a is odd the range of a_1 will change to $[0, 3]$. Here we want to note that we could have written $p = 2^{a+1} 3^b - 1$ instead of $p = 4 \cdot 2^{a-1} 3^b - 1$ with the cofactor equal to one, both of these representations of p have no major impact on the performance and can be switched between one another trivially by simple mathematical manipulations. Using the same argument as above we need b to be even else it will impact the performance significantly, as division by 6 or 12 is not as easy as division by 2 or 4.
We note that this conversion from normal integer representation to this special representation and vice versa is a costly procedure. But we explain at the end of this section that this conversion and reconversion are one-time procedures that we need to perform at the beginning and the end of the key-exchange algorithm.

4.1 Multiplication Algorithm

Let's suppose we have two numbers $A, B \in \mathbb{F}_p$ as represented in Equation (1). After multiplying them we get the result C as per the equation shown below:

$$\begin{aligned} C = {} & a_1 b_1 \cdot 2^{2a} 3^{2b} + (a_1 b_2 + a_2 b_1) 2^{3a/2} 3^{3b/2} + (a_1 b_3 + a_2 b_2 + a_3 b_1) 2^a 3^b \\ & + (a_2 b_3 + a_3 b_2) 2^{a/2} 3^{b/2} + a_3 b_3. \end{aligned} \tag{2}$$

Since the prime p is of the form $2 \cdot 2^a 3^b - 1$, we can replace $2^a 3^b$ in Equation (2) by $2^{-1} (mod\ p)$. Hence $a_1 b_1 \cdot 2^{2a} 3^{2b}$ gets replaced by 0 or $2^{-2} \pmod p$ as $a_1, b_1 \in \{0, 1\}$ and $a_1 b_1 \in \{0, 1\}$. Note that for a fixed prime we can precompute the value of $2^{-2} \pmod p$ and use that for the above replacement in Equation (2).

We can replace $(a_1 b_3 + a_2 b_2 + a_3 b_1) 2^a 3^b$ as follows. If $(a_1 b_3 + a_2 b_2 + a_3 b_1)$ is even, we can write $(a_1 b_3 + a_2 b_2 + a_3 b_1) 2^a 3^b = (a_1 b_3 + a_2 b_2 + a_3 b_1)/2 \pmod p$. Otherwise we can write $(a_1 b_3 + a_2 b_2 + a_3 b_1) 2^a 3^b = ((a_1 b_3 + a_2 b_2 + a_3 b_1 - 1)/2) \pmod p + (a_1 b_3 + a_2 b_2 + a_3 b_1) \pmod 2 \cdot 2^a 3^b$. Considering both the even and odd cases we can write the following equation:

$$\begin{aligned} & (a_1 b_3 + a_2 b_2 + a_3 b_1) 2^a 3^b \\ \implies & \Big(\lfloor (a_1 b_3 + a_2 b_2 + a_3 b_1)/2 \rfloor \Big) + \Big((a_1 b_3 + a_2 b_2 + a_3 b_1) \mod 2 \Big) 2^a 3^b. \end{aligned}$$

Similarly,

$$
\begin{aligned}
&\Big(a_1b_2 + a_2b_1\Big) \cdot 2^{3a/2}3^{3b/2} \\
\implies &\Big(\lfloor (a_1b_2 + a_2b_1)/2 \rfloor\Big) \cdot 2^{a/2}3^{b/2} + \Big((a_1b_2 + a_2b_1) \mod 2\Big) \cdot 2^{a/2-1}3^{b/2}.
\end{aligned}
$$

Rewriting Equation (2) by replacing the coefficients we get the following equation:

$$
\begin{aligned}
A \times B \ = \ &\Big(\underbrace{2^{-2} \pmod p}_{\text{replacing } 2^{2a}3^{2b}} a_1b_1 \ + \ a_3b_3 + ((a_1b_2 + a_2b_1) \pmod 2)2^{a/2-1}3^{b/2} + \\
&\underbrace{\lfloor (a_1b_3 \ + \ a_2b_2 \ + \ a_3b_1)/2) \rfloor}_{\text{replacing } (a_1b_3 + a_2b_2 + a_3b_1)2^a3^b} \Big) + \Big(\underbrace{\lfloor (a_1b_2 + a_2b_1)/2 \rfloor}_{\substack{\text{replacing} \\ (a_1b_2 + a_2b_1)2^{3a/2}3^{3b/2}}} + (a_2b_3 + a_3b_2) \Big) 2^{a/2}3^{b/2} \\
&+ \Big((a_1b_3 \ + \ a_2b_2 \ + \ a_3b_1) \pmod 2 \Big) 2^a3^b.
\end{aligned}
$$

The algorithm is described in Algorithm 4. To compute the above expression we have to perform four smaller multiplications: $a_2b_2,\ a_2b_3,\ a_3b_2\ \ a_3b_3$, as the other terms which are multiplied with $a_1, b_1 \in \{0, 1\}$.

Now we have the product as $A \times B = C = C_1 \cdot 2^a3^b \ + \ C_2 \cdot 2^{a/2}3^{b/2} \ + \ C_3$, but in this expression the coefficients C_2 and C_3 lie in the range $[0, 2^a3^b)$, which is not consistent with our representation where C_2 and C_3 should lie in the range $[0, 2^{a/2}3^{b/2})$. Hence we need to split them further so that they fit according to our representation scheme. This splitting involves divisions of the coefficients C_i for $i = 2$ and 3 by $2^{a/2}3^{b/2}$. In the next section we are going to explain how we can do this division efficiently.

4.2 Efficient Division

Our purpose is to divide a number $C_i \in [0, 2^a3^b)$ by $2^{a/2}3^{b/2}$ and calculate the quotient q and remainder r in an efficient way. We note that division by two is a simple right shift operation. Hence we perform the division by $2^{a/2}3^{b/2}$ using the steps shown below.

1. Extract the $a/2$ least significant bits of C_i and store them in a variable r_1.
2. Right shift C_i by $a/2$ bits to obtain C_i'.
3. Divide C_i' by $3^{b/2}$ to get the quotient q and the remainder r_2.

Hence we have $C_i = q \cdot 2^{a/2}3^{b/2} + (r_2 \cdot 2^{a/2} + r_1) = q \cdot 2^{a/2}3^{b/2} + r$.

The division operation by $3^{b/2}$ in Step 3 is not as easy as the division by $2^{a/2}$. However since b is a fixed integer, the division can be performed using multiplications similar to the Barrett reduction technique [7] as described in Algorithm1 in Section 3.

Obtaining the quotients and remainders after dividing C_2 and C_3 by $2^{a/2}3^{b/2}$, it is trivial to write C in the desired representation of a finite field element.

Input: 2 numbers $Q \in [0, 2^a 3^b)$ and $P = 2^{a/2} 3^{b/2}$ and $\log_2 Q \approx 2 \cdot \log_2 P$. $P' = P/2^{a/2}$ precomputed $x = 2^k/P'$, k is as described in Section 3.2

Output: q and r such that $Q = q \cdot P + r$

1 $t \leftarrow \lfloor Q/2^{a/2} \rfloor, s = Q \pmod{2^{a/2}}$;
2 $q \leftarrow t \times x >> k$;
3 $r \leftarrow t - P' \times q$;
4 $r \leftarrow r \times 2^{a/2} + s$;
5 **if** $r > P$ **then**
6 | $r \leftarrow r - P$;
7 | $q \leftarrow q + 1$
8 **end**
9 **return** q, r

Algorithm 3: Our Division Algorithm

In the next part of this section we will compare the cost of our modular reduction technique with the original Barrett reduction technique. Note that the parameters a and b in the prime $p = 2 \cdot 2^a \cdot 3^b - 1$ are chosen in such a way that $\log_2(2^a) \approx \log_2(3^b)$. For convenience let us take $\log_2(2^a) \approx \log_2(3^b) \approx N$. So the prime is of size $2N$ bits.

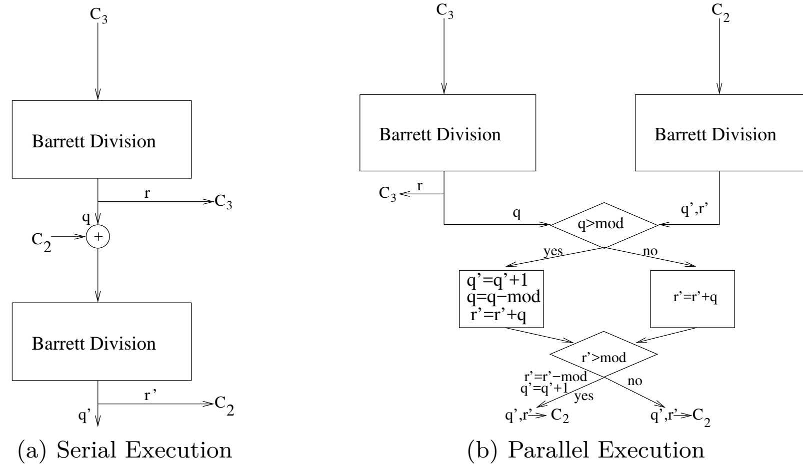
Comparison with Barrett Reduction : In the Barrett reduction technique in Algorithm 1 the result of an integer multiplication that is of size $\leq 4N$ bits is reduced by a prime of size $2N$ bits. For correct computation k is of size $4N$ bits. In this scenario we have to perform one $4N \times 2N$ bit multiplication to compute the quotient (line 1 in Algorithm 1) and one $2N \times 2N$ bit multiplication to compute the remainder (line 2 in Algorithm 1). Thus using a quadratic complexity multiplier, the Barrett reduction technique has a cost of $12N^2$. In our modular reduction technique we perform divisions of two numbers C_2 and C_3 of size $\leq 2N$ by an N bit number $2^{a/2}3^{b/2}$. Since division by a power of two is almost free, the cost of each division reduces to the cost of dividing a number of size $\leq 3N/2$ bits by a $N/2$ bit number. To perform the divisions correctly we need to fix the value of k to $3N/2$. Hence each of the two division operations perform a $3N/2 \times N$ bit multiplication and an $N \times N/2$ bit multiplication (lines 2 and 3 in Algorithm 3). Thus using a quadratic complexity multiplier, our reduction technique has a cost of $4N^2$.

Comparison with Montogomery multiplication : In this section we provide a comparison of the computational cost of Montgomery multiplication with our technique. As defined in Section 4.2 our prime is of size $2 \cdot N$ bits. For executing a single round of Montgomery multiplication we need two $2N \times 2N$ bit multiplications. And a relatively easier multiplication of $t \cdot p' \pmod{r}$ where only the last $2 \cdot N$ bits of the result are required. In our case we need only four $N \times N$ bit multiplications for the first part of our algorithm and two $3N/2 \times N$ bit and $N \times N/2$ bit multiplications for the final reduction.

```
Input: A, B ∈ F_p , A = a_1 · 2^a 3^b + a_2 · 2^(a/2) 3^(b/2) + a_3 and
       B = b_1 · 2^a 3^b + b_2 · 2^(a/2) 3^(b/2) + b_3; 2^-2 (mod p) precalculated
Output: C = A × B (mod p) ,C = C_1 · 2^a 3^b + C_2 · 2^(a/2) 3^(b/2) + C_3
1  C_1 = 0; C_2 = 0; C_3 = 0;
2  Multiply a_2 × b_2, a_2 × b_3, a_3 × b_2, a_3 × b_3;                // ∈ [0, 2^a 3^b)
3  Multiply
     a_1 × b_1, a_1 × b_2, a_1 × b_3, b_1 × a_2 b_1 × a_3;           // ∈ [0, 2^(a/2) 3^(b/2))
4  C_3 ← a_1 b_1 · 2^-2 (mod p) + a_3 b_3;
5  C_2 ← a_2 b_3 + a_3 b_2;
6  t ← (a_1 b_2 + a_2 b_1) ;                  // replacing (a_1 b_2 + a_2 b_1) 2^(3a/2) 3^(3b/2)
7  if isEven(t) then
8  |  C_2 ← C_2 + t/2
9  else
10 |  t ← t − 1;
11 |  C_2 ← C_2 + t/2;
12 |  C_3 ← C_3 + 2^(a/2−1) 3^(b/2)
13 end
14 t ← (a_1 b_3 + a_2 b_2 + a_3 b_1) ;        // replacing (a_1 b_3 + a_2 b_2 + a_3 b_1) 2^a 3^b
15 if isEven(t) then
16 |  C_3 ← C_3 + t/2;
17 |  C_1 ← 0
18 else
19 |  t ← t − 1;
20 |  C_3 ← C_3 + t/2;
21 |  C_1 ← 1
22 end
   /* End of first part C = C_1 2^a 3^b + C_2 2^(a/2) 3^(b/2) + C_3, reduce
      C_2, C_3 = O(2^a 3^b) further by Barrett division                      */
23 q, r ← BarrettDivision(C_3);
24 C_3 ← r;
25 C_2 ← C_2 + q;
26 q, r ← BarrettDivision(C_2);
27 C_2 ← r;
28 C_1 ← C_1 + q;
29 if isEven(C_1) then
30 |  C_3 ← C_3 + C_1/2;
31 |  C_1 ← 0
32 else
33 |  C_3 ← C_3 + (C_1 − 1)/2;
34 |  C_1 ← 1
35 end
36 if C_3 overflows i.e C_3 > 2^(a/2) 3^(b/2), then C_3 ← C_3 − 2^(a/2) 3^(b/2), C_2 ← C_2 + 1 if
     C_2 also overflows C_1 ← C_1 + 1 and repeat steps 29 to 35, this situation
     occurs rarely and also then we have to perform this step at most once;
37 return (C_1 · 2^a 3^b + C_2 · 2^(a/2) 3^(b/2) + C_3)
```

Algorithm 4: Multiplication Algorithm

Here we want to mention that the two *Barrett Divisions* performed in the reduction stage (23 and 26) of Algorithm 4 can be run in parallel, effectively reducing the computing time by half.



(a) Serial Execution (b) Parallel Execution

Fig. 1. Serial and Parallel execution for the reduction part of Algorithm 4

5 Software implementation

For a comparison of the effective speedup of our algorithm we implemented our algorithm using C in a 32 bit multi-precision format for a security level of 128 bits. We also implemented a normal Barrett reduction using the same multi-precision format. The cost of multiplication when multiplying two input

Operation	running time (μ s)
Barrett Reduction	50.547
Normal multiplication	67.097
Our Reduction	19.565
Our Multiplication	38.490

Table 1. Comparison of Our algorithm with normal Barrett reduction algorithm

numbers in both of the algorithms is expected to remain the same. Therefore we used normal schoolbook multiplication. Upon running multiple instances of both

the algorithms on a computer with CentOS on a core i5 CPU and averaging the running times we obtain the results as given in Table 1. As we can see from the table, we achieve an approximate 62% speed-up in reduction and 43% speed-up for modular multiplication (multiplication + reduction) with our method against the normal Barrett reduction. This result is consistent with our prediction in Section 4.2.

6 Hardware implementation

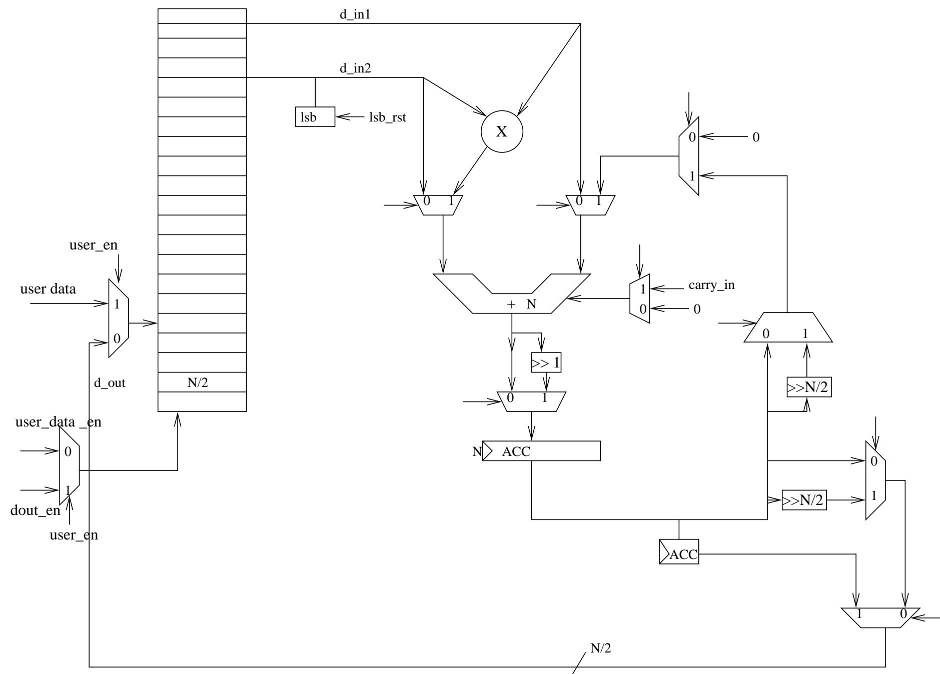


Fig. 2. Hardware Architecture

To check the performance of the new modular multiplication scheme, we have designed a hardware architecture that performs modular multiplications following Algorithm. 4. The arithmetic unit of the architecture consists of a combinational multiplier of input size N/2 and an addition/subtraction circuit of input size N. The operands are stored as arrays of N/2 bit words in a register file that contains 52 registers in total and of which 16 registers were used to store the pre-computed values as required by the algorithm 4 . Since the proposed algorithm performs arithmetic operations on two operands, we kept two output ports and one input port in the register file. During a multiplication, the multiplier

performs multiplications of words and the adder helps to accumulate the result in the accumulator register ACC. For performing multi-precision additions and subtractions, only the lower half (i.e. the N/2 bits) of the addition/subtraction circuit is used. The control signals are generated by a hierarchy of finite state machines for multi-precision addition, subtraction, shifting and multiplication. On the top of the hierarchy, a finite state machine executes the operations required for the modular multiplication operation.

We have compiled the hardware architecture using the Xilinx ISE 14.4 tool targetting the Virtex 6 FPGA (xc6vcx240t-2ff784). For this evaluation, we chose the field generated by the prime $2 \cdot 2^{386} 3^{242} - 1$ (hence N is 385 bits). After place and route operation, the architecture consumes $11,924$ registers and $12,790$ look-up-tables, accounting to 3% and 8% of the resources available in the FPGA. The operating frequency of the architecture is 31 MHz. One modular multiplication (integer multiplication + modular reduction) takes 236 cycles and hence 7.61 μs.

7 Conclusion

We presented a fast modular multiplication algorithm that exploits the special structure of primes of the form $p = 2 \cdot 2^a 3^b - 1$, used in isogeny based post-quantum cryptography. To our knowledge there is no other algorithm that exploits the structure of such primes for fast reduction. We have shown that our algorithm is more efficient than Montgomery multiplication and Barrett reduction. We believe that with our algorithm will significantly decrease the time required to calculate isogenies between supersingular elliptic curves, which will strengthen the potential of isogeny based post-quantum cryptography as a practical post-quantum cryptosystem.

Acknowlegments

A. Karmakar and S. Sinha Roy were supported by Erasmus Mundus PhD Scholarship. This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work was supported in part by iMinds, the Flemish Government, FWO G.0550.12N, G.00130.13N and FWO G.0876.14N, by the Hercules Foundation AKUL/11/19, and by the European Commission through the Horizon 2020 research and innovation programme under contract No H2020-ICT-2014-644371 WITDOM, and H2020-ICT-2014-644209 HEAT, and H2020-ICT-2014-645622 PQCRYPTO.
We would also like to thank Carl Bootland for his help in proof checking the manuscript.

References

1. Luca De Feo, David Jao & Jérôme Plût, *"Towards quantum resistant cryptosystems from supersingular elliptic curve isogenies"*, https://eprint.iacr.org/2011/506.pdf.

2. Shor, Peter W. (1997), *"Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer"*, SIAM J. Comput. 26 (5): 1484–1509, arXiv:quant-ph/9508027v2
3. Peter Montgomery, *"Modular Multiplication Without Trial Division,"*, http://www.ams.org/journals/mcom/1985-44-170/S0025-5718-1985-0777282-X/home.html vol. 44, pp. 519–521, 1985
4. Jerome A. Solinas, *"Generalized Mersenne Prime"*,Encyclopedia of cryptography and security, pp509-510, 2011
5. Jerome A. Solinas, *"Mersenne Prime"*, Encyclopedia of cryptography and security, pp 774-775, 2011.
6. Jerome A. Solinas, *"Pseudo-Mersenne Prime"*, Encyclopedia of cryptography and security, pp 992-992, 2011.
7. Barrett, P. (2006). *"Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor"*. Advances in Cryptology — CRYPTO' 86. Lecture Notes in Computer Science 263. pp. 311–323.
8. Andrew Childs, David Jao, and Vladimir Soukharev. *Constructing elliptic curve isogenies in quantum subexponential time*, 2010. http://arxiv.org/abs/1012.4019/.
9. Ireland, Kenneth; Rosen, Michael (1990), *A Classical Introduction to Modern Number Theory (2nd ed.)*, New York: Springer, Proposition 1.1.1, ISBN 0-387-97329-X
10. Florent De Dinechin, *Laurent-Stèphane Didier. Table-based division by small integer constants.* Applied Reconfigurable Computing, Mar 2012, Hong Kong, Hong Kong SAR China. pp.53-63, 2012.
11. D. Jao and L. De Feo. *Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies*. In B. Yang, editor, PQCrypto 2011, volume 7071 of LNCS, pages 19–34. Springer, 2011.
12. Knuth, Donald. *The Art of Computer Programming.* Addison-Wesley. Volume 2, Chapter 4.
13. Steven Galbraith, Anton Stolbunov, *Improved algorithm for the isogeny problem for ordinary elliptic curves*, Applicable Algebra in Engineering, Communication and Computing, June 2013, Volume 24, Issue 2, pp 107-131
14. Joseph H. Silverman, *The arithmetic of elliptic curves* Graduate Texts in Mathematics Volume 106 2009. Springer-Verlag, New York.
15. Anton Stolbunov. *Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves*. Adv. Math. Commun., 4(2):215–235, 2010.
16. David Jao and Vladimir Soukharev, *Isogeny-based quantum-resistant undeniable signatures*, Post-Quantum Cryptography Volume 8772 of the series Lecture Notes in Computer Science pp 160-179
17. Denis X. Charles, Kristin E. Lauter , Eyal Z. Goren *Cryptographic hash functions from expander graphs*, Journal of Cryptology, January 2009, Volume 22, Issue 1, pp 93-113
18. Seiichiro Tani. *Claw Finding Algorithms Using Quantum Walk.* http://arxiv.org/abs/0708.2584, March 2008.
19. *Microsoft predicts practical quantum computers within 10 years* http://www.ibtimes.co.uk/microsoft-predicts-practical-quantum-computers-within-10-years-1524268
20. McEliece, Robert J. (1978). *"A Public-Key Cryptosystem Based On Algebraic Coding Theory"*. DSN Progress Report 44: 114–116
21. Jintai Ding, Dieter Schmidt *"Rainbow, a New Multivariable Polynomial Signature Scheme".* Third International Conference, ACNS 2005, New York, NY, USA, June 7–10, 2005. Proceedings. Lecture Notes in Computer Science 3531: 64–175.

22. *Recommended elliptic curves for federal government use* http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf
23. *The GNU Multiple Precision Arithmetic Library* .https://gmplib.org/
24. *The PARI/GP computer algebra system* http://pari.math.u-bordeaux.fr/

A An Example

In this section we provide a small example of the method described in the paper. Let $a = 22$,and $b = 16$ so that the prime is $p = 2 \cdot 2^a \cdot 3^b - 1 = 361102068154367$, $n = 2^a \cdot 3^b = 180551034077184$, $\sqrt{n} = 13436928$
$A = 128965951662196 \ = 0 * n \ + \ 9597874 * \sqrt{n} \ + \ 971124$, and
$B = 230338429880123 \ = 1 * n \ + \ 3705266 * \sqrt{n} \ + \ 334009$
After executing the first stage of the multiplication algorithm, we reached $A \times B = C = C_1 n + C_2\sqrt{n} + C_3$ with $C_1 = 0$, $C_2 = 68262390904455$, $C_3 = 50417786320088$. We have to reduce C_2 and C_3 further by dividing them using $\sqrt{n}$. Using our Barrett division algorithm we found $C_3 = 3752181 * \sqrt{n} + 380120$, we set the remainder 380120 to C_3 and add the quotient with C_2. We again divide C_2 with $\sqrt{n}$
$C_2 = 68262390904455 + 3752181 = 68262394656636$
$C_2 = 5080208 * \sqrt{n} + 5535612$, we set the remainder to C_2 and add the quotient with C_1 to get $C_1 = 5080208$.
As $C_1 \pmod 2 = 0$,we add $C_1/2 = 2540104$ to C_3 to get $C_3 = 380120 + 2540104 = 2920224$.
Here C_3 is smaller than $\sqrt{n}$ and there is no overflow. So we stop our algorithm here. Finally, we get the result as $C = 0 * n + 5535612 * \sqrt{n} + 2920224 = 74381622800160$, which is indeed $A \times B \pmod p$.

B Application in Isogeny based post-quantum key exchange protocol

The isogeny based post-quantum protocol, described in Section 3 works by computing and applying isogenies over supersingular elliptic curve groups. These operations are fundamentally field arithmetic operations over the field $\mathbb{F}_{p^2}$, where the curve is defined.
Here we want to mention that modular addition and subtraction is also easy in our representation. Let's say we want to add two numbers $A, B \in \mathbb{F}_p$ to get the sum $C = (a_1 + b_1) \cdot n + (a_2 + b_2) \cdot \sqrt{n} + (a_3 + b_3) = C_1 \cdot n + C_2 \cdot \sqrt{n} + c_3$ for convenience we have assumed $n = 2^a 3^b$. Here again, similar to multiplication algorithm, C_1, C_2 and C_3 may not be consistent with our representation as given in Equation (1). To make C consistent with our representation we follow steps 23 to 36 of Algorithm 4. But here we do not have to use the division Algorithm 3, only a subtraction by $2^{a/2}3^{b/2}$ will suffice. For subtraction we first negate a number $B \in \mathbb{F}_p$ as $-B = p - b = (1 - b_1) \cdot n + (\sqrt{n} - 1 - b_1) \cdot \sqrt{n} + (\sqrt{n} - 1 - b_3)$ followed by an addition.
To apply our method to the isogeny based key exchange algorithm as mentioned

in section 3.1, we changed the representation of the public parameters in the beginning of the algorithm and executed the algorithm. In the last step we changed the representation back to the original form and matched both Alice and Bob's *j-invariant*.
To further test the correctness of the algorithm we ran an instance of the unmodified algorithm with same parameter set and numbers m and n. We verified that both executions produce identical results.

C List of Primes

In this section we list values for a and b for security level of around 256 bit and 512 bit. We found these values by a simple brute-force search using a C implementation. As mentioned before the prime is $p = 2 \cdot 2^a 3^b + k$, with the value of $\log_2(3^b)$ close to a. The primality has been tested using *GMP* [23] and *PARI/GP* [24]. Also we should mention that this list is not exhaustive.

#	a	b	k	#	a	b	k	#	a	b	k
1	738	514	+1	10	760	490	-1	19	814	538	-1
2	741	510	+1	11	764	484	-1	20	819	552	+1
3	747	468	+1	12	768	518	+1	21	826	528	+1
4	748	468	-1	13	772	478	-1	22	826	538	-1
5	750	482	-1	14	774	476	-1	23	829	458	+1
6	750	490	+1	15	778	484	+1	24	830	512	+1
7	752	542	-1	16	784	496	+1	25	832	470	+1
8	756	468	-1	17	792	480	+1	26	834	488	-1
9	758	514	-1	18	798	526	+1				

Table 2. Table for primes with around 256 bit PQ security

#	a	b	k	#	a	b	k
1	1538	946	+1	5	1556	958	+1
2	1541	982	+1	6	1569	966	+1
3	1550	1018	-1	7	1570	942	-1
4	1551	964	+1	8	1598	1034	+1

Table 3. Table for primes with around 512 bit PQ security