Efficient Word Size Modular Arithmetic

Thomas Plantard

Abstract—Modular multiplication is used in a wide range of applications. Most of the existing modular multiplication algorithms in the literature often focus on large size moduli. However, those large moduli oriented modular multiplication solutions are also used to implement modular arithmetic for applications requiring modular arithmetic on moduli of size inferior to a word size i.e. 32/64bits. As it happens, a large majority of applications are using word size modular arithmetic.
In this work, we propose a new modular multiplication designed to be computed on one word size only. For word size moduli, in a large majority of instances, our solution outperforms other existing solutions including generalist solutions like Montgomery's and Barrett's modular multiplication as well as classes of moduli like Mersenne, Pseudo-Mersenne, Montgomery-Friendly and Generalized Mersenne.

Index Terms—Modular Arithmetic, Modular Multiplication, Modular Exponentiation, Polynomial Evaluation, Number Theoretical Transform, Residue Number System, Mixed Radix System.

1 Introduction

Modular arithmetic has a wide range of applications. Amongst its many applications, cryptographic protocols require efficient modular exponentiation (for Diffie-Hellman key exchange protocol [1], RSA [2], DSA [3] ...) and computation on finite field for Elliptic Curves Cryptography [4], [5] (ECC) or Isogeny based Cryptography [6], [7]. Consequently, a huge number of solution have been proposed to operate efficiently on moduli used in cryptography i.e. moduli of hundreds or even thousands of bits.

However, there is a large number of applications using modular arithmetic on smaller moduli i.e. ~ 10 to 32 bits moduli, these moduli operations fit in a word size. For example, one application is Residue Number System [8] which purposefully replace large operations with multiple smaller modular ones. An other widely used application of modular arithmetic is polynomial evaluation. Modular polynomial evaluation is used for polynomial factorization [9], to compute polynomial greatest common divisor [10], as well as for secret sharing protocols [11]. Recently, lattice based cryptography [12] has become one of the leading solution for post-quantum cryptography [13]; lattice based cryptography is mainly based on modular arithmetic on small moduli, 10 to 13 bits (see for example [12]) often using Number Theoretical Transform to encrypt/decrypt efficiently [14], [15], [16], [17]

Existing solutions for large moduli do not perform as well as expected when restrained to a single word.

Main result

In this work, we present a new modular reduction designed to fully take advantage of a word size modular arithmetic specificity. We show how it outperform existing solutions.

- *Thomas Plantard is with IC2, Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, Faculty of Engineering and Information Sciences, University of Wollongong, Wollongong NSW 2522, Australia. E-mail: thomaspl@uow.edu.au*

Organization of the paper

In Section 2, we present different existing solutions to perform modular operation and more specifically modular multiplication which is the key - and most costly - operation of modular arithmetic.

In Section 3, we present our new modular multiplication.

In Section 4, we present some comparisons as well as some performance evidences for different applications, namely modular exponentiation, polynomial evaluation, Number Theoretical Transform and Residue Number System to Mixed Radix System conversion.

Section 5 will conclude this work.

2 Existing Modular Multiplications

In this section, we present the most used modular multiplication methods. We are placing ourselves in the context where the multiplication is correct only on $2n$ bits i.e. $2n$ will classically correspond to a word size of 32 or 64. In such context, the size of the modulo P will be bounded to guarantee correctness of those classical algorithms.

Those methods can often be described in 3 consecutive steps:

- Firstly, an integer multiplication is performed,
- Secondly, a modular reduction is performed,
- Finally, a correction is performed, and it is composed by one or two tests (**IF**) followed if necessary by few additions/substations.

To obtain efficient modular arithmetic, the final corrective step can be omitted if some redundancy is allowed. One will only requires then, that for a constant S, the modular multiplication algorithm receiving in input two value A, B both inferior to S, guarantee an output C inferior to S as well. This will guarantee a stable representation during modular operation. However, in such context one needs to bound even further the potentially usable moduli.

To have clearer and more readable algorithms, we will note $[X]_k = X \bmod 2^k$, the result of a simple mask to extract the k least significant bits of X. For the same reasons,

we will note $[X]^k = \lfloor \frac{X}{2^k} \rfloor$, the result of a simple shift of k bits on X.

2.1 Montgomery Multiplication

In 1985 [18], P. Montgomery proposed one of the most used modular multiplication algorithm.

Algorithm 1: Montgomery Modular Multiplication [18]

Input : A, B, P, R, n with $0 \leq A, B < P < 2^n$ and $R = (-P^{-1}) \bmod 2^n$
Output: C with $0 \leq C < P$ and $C = AB2^{-n} \bmod P$
begin
  $C \leftarrow AB$
  $C \leftarrow [C + [CR]_n P]^n$
  if $C \geq P$ **then**
    return $C - P$
  end
  return C
end

Montgomery algorithm can perform modular multiplication correctly,

- in general, if P is odd and $P < 2^n$,
- on $2n$ bits, if $P^2 + 2^n P < 2^{2n}$ allows $C + [CR]_n P$ to be computed correctly before being divided by 2^n i.e. $P < \frac{2^n}{\phi}$ with $\phi = \frac{1+\sqrt{5}}{2}$, the golden ratio,
- when omitted final correction, if $P < 2^{n-2}$ as proposed by C. D. Walter [19]. One can check that to guarantee than $C < S$ with $A, B < S$ then one needs $\frac{S^2 + 2^n P}{2^n} < S$ and therefore $P < \frac{S2^n - S^2}{2^n}$. Consequently the largest P, one can operate on, can be computed by deriving $\frac{S2^n - S^2}{2^n}$ on S and obtaining $S = 2^{n-1}$ as a maxima. Finally, one can obtain for $P < \frac{S2^n - S^2}{2^n} = \frac{2^{2n-1} - 2^{2n-2}}{2^n} = 2^{n-1} - 2^{n-2} = 2^{n-2}$.

Remark 1 (Montgomery Representation). An important remark is that Montgomery modular multiplication (**MONT**) is not returning the correct value $X \bmod P$ but rather $X2^{-n} \bmod P$. This is generally solved by using a specific representation. Indeed, if $A \bmod P$ (respectively $B \bmod P$) is represented by $\widetilde{A} = A2^n \bmod P$ (respectively $\widetilde{B} = B2^n \bmod P$) then $\widetilde{AB} = AB2^n \bmod P$ representing $AB \bmod P$ can be computed directly using Montgomery's Algorithm: $\mathbf{MONT}(\widetilde{A}, \widetilde{B}) = (A2^n)(B2^n)2^{-n} \bmod P = AB2^n \bmod P$. Montgomery's representation is also stable by addition and consequently will be stable during a full modular computation.
To enter or exit Montgomery's representation one can simply uses Montgomery Algorithm (Algorithm 1) as well:

- to enter, using a precomputed value, $2^{2n} \bmod P$, one can obtain $\mathbf{MONT}(A \bmod P, 2^{2n} \bmod P) = A(2^{2n})2^{-n} \bmod P = A2^n \bmod P = \widetilde{A}$,
- to exit, $\mathbf{MONT}(\widetilde{A}, 1) = A2^n 2^{-n} \bmod P = A \bmod P$.

2.2 Barrett Multiplication

In 1986 [20], P. Barrett proposed an algorithm slightly more expensive than Montgomery's, however which did not require a dedicated representation.

Algorithm 2: Barrett Modular Multiplication [20]

Input : A, B, P, R, n with $0 \leq A, B < P$ and $R = \lfloor \frac{2^{2n}}{P} \rfloor$
Output: C with $0 \leq C < P$ and $C = AB \bmod P$
begin
  $C \leftarrow AB$
  $C \leftarrow C - [[C]^{n-1} R]^{n+1} P$
  if $C \geq 2P$ **then**
    return $C - 2P$
  end
  if $C \geq P$ **then**
    return $C - P$
  end
  return C;
end

Barrett's algorithm can perform modular multiplication correctly

- in general, if $P < 2^n$,
- on $2n$ bits, only if $P < 2^{n-1}$ allows $[[C]^{n-1} R] < \frac{P^2}{2^{n+1}} \frac{2^{2n}}{P} < P2^{n+1}$ to be computed correctly before being divided by 2^{n+1}.

In 1991, J.-J. Quisquater [21] proposed some improvement of Barrett's algorithm specially efficient for hardware implementation. See [22] a survey.

2.3 Mersenne Moduli

Some applications allow users to pick the modulo P, generally under some restrictions. Consequently, different options for picking moduli with efficient modular arithmetic have been proposed [8]. Besides the most natural option for a CPU, $P = 2^n$, one of the oldest number proposed to be used for modular arithmetic and named after Marin Mersenne is the *Mersenne's number*, $P = 2^n - 1$.

Algorithm 3: Multiplication modulo a Mersenne [8]

Input : A, B, P, n with $0 \leq A, B < P = 2^n - 1$
Output: C with $0 \leq C < P$ and $C = AB \bmod P$
begin
  $C \leftarrow AB$
  $C \leftarrow [C]_n + [C]^n$
  if $C \geq P$ **then**
    return $C - P$
  end
  return C
end

Algorithm 3 is certainly one of the most efficient modular multiplication algorithm. The final correction can be omitted if one replace it by a repetition of the second step and consequently avoiding comparison. We will call *Forced* such type of reduction modulo a Mersenne and we will show in Section 4.2 that in some case it allows faster reduction.

2.4 Pseudo Mersenne

Even if users can pick which moduli they are using, some applications require to have multiple options as they required more than one modulo (for example Residue Number System implementation) or a special subclass (prime number, for example for finite field). Consequently, denser class of moduli have been proposed. In 1992 [23], R. Crandall proposed to use the so-called *pseudo-Mersenne number* i.e. $P = 2^n - K$ with $K < 2^{\frac{n}{2}}$.

Algorithm 4: Multiplication modulo a Pseudo Mersenne [23]

Input : A, B, P, K, n with $0 \leq A, B < P = 2^n - K$ and $K^2 < 2^n$
Output: C with $0 \leq C < P$ and $C = AB \bmod P$
begin
  $C \leftarrow AB$
  $C \leftarrow [C]_n + [C]^n K$
  $C \leftarrow [C]_n + [C]^n K$
  if $C \geq P$ **then**
    return $C - P$
  end
  return C
end

Pseudo-Mersenne are widely used in cryptography [24]. It is also widely used for Residue Number System [25].

Generalizations have been proposed by [26] and by [27] to allow modular operations on a even larger family. As those families are slower than pseudo-Mersenne and as we will show in Section 4.2 that Pseudo-Mersenne moduli offer a modular arithmetic not competitive for word-size arithmetic. We will not detailed those families in this work, we will only detailed how to operate on Pseudo-Mersenne.

Remark 2 (Redundancy). Redundancy can be allowed to omit final correction as for Montgomery algorithm. If $P = 2^n - K$ and $K^2 \leq 2^{n-3}$ then if $A, B < 2P$, one can remark that result before correction will be inferior to

$$\frac{4P^2K^2}{2^{2n}} + 2^n.$$

Then we obtain

$$\frac{4P^2K^2}{2^{2n}} + 2^n < 4K^2 + 2^n.$$

So if $K^2 \leq 2^{n-3}$ then one obtains

$$4K^2 + 2^n \leq 2^{n-1} + 2^n < 2P.$$

Such restriction could be refined, however, we will see that those restrictions are still allowing a family of moduli enough large for the different applications studied in Section 4.

However, as we can operate only on $2n$ bits, then in a word size context we obtain $P = 2^{n-1} - K$ and $K^2 \leq 2^{n-4}$ to guarantee correctness of the initial product AB. With $AB < 4P^2 \leq 42^{2(n-1)} \leq 2^{2n}$, we guarantee correct computation.

2.5 Generalized Mersenne

To avoid expensive multiplication, J. Solinas [28] proposed to replace the variable part of the pseudo-Mersenne, $K = 2^n - P$, with a number with low hamming weight. Consequently, multiplication by K in Algorithm 4 can be replaced by shifts and additions. Those numbers are often called *Generalized Mersenne*. In this work, we will focus on the most efficient ones, the ones representable by $P = 2^n - 2^k - 1$. Even if options with more moduli exist, we will focus on *trinoms*, $P = 2^n - 2^k - 1$, as we will show in Section 4.2 that Generalized Mersenne moduli based on trinoms are already not the most competitive option for word-size arithmetic. A larger class of Generalized Mersenne could become only even slower. Algorithm 5 presents how to perform a modular multiplication modulo a Generalized Mersenne.

Algorithm 5: Multiplication modulo a Generalized Mersenne [28]

Input : A, B, P, n, k with $0 \leq A, B < P = 2^n - 2^k - 1$ and $k < \frac{n}{2}$
Output: C with $0 \leq C < P$ and $C = AB \bmod P$
begin
  $C \leftarrow AB$
  $C \leftarrow [C]_n + [C]^n + [C]^n 2^k$
  $C \leftarrow [C]_n + [C]^n + [C]^n 2^k$
  if $C \geq P$ **then**
    return $C - P$
  end
  return C
end

Generalized Mersenne have been standardised by the U.S. National Institute of Standards and Technology to be used for efficient elliptic curve cryptography [29]. For Residue Number System, it allows some powerful relations when the full basis is only composed of Generalized Mersenne [30].

To omit final correction, one can use the same reasoning applied for Pseudo-Mersenne (See Remark 2).

2.6 NFLlib Modular Multiplication

If one needs access to a larger family than Pseudo-Mersenne, in NFLlib [31], authors have proposed a modification of a general work on euclidean division by a constant by Moller and Granlund [32].

Algorithm 6: NFLlib Modular Multiplication [31]

Input : A, B, P, n, k with $0 \leq A, B < P < 2^{n-e}$ and $R = \lfloor \frac{2^{2n}}{P} \rfloor \bmod 2^n$
Output: C with $0 \leq C < P$ and $C = AB \bmod P$
begin
  $C \leftarrow AB$
  $Q \leftarrow [R[C]^n + 2^e C]_{2n}$
  $C \leftarrow [C - [Q]^n P]_n$
  if $C \geq P$ **then**
    return $C - P$
  end
  return C
end

To operate correctly Algorithm 6 requires that

$$\frac{1+2^{-3e}}{2^e+1}2^n < P < \frac{2^n}{2^e}$$

with $1 \leq e < n$. From which one obtains that

$$2^{n-e} - 2^{n-2e} + 2^{n-3e} < P < 2^{n-e}.$$

It is important to note that if the size of the moduli on which the modular multiplication operates is somewhat smaller, $n-1$ instead of n for Pseudo-Mersenne, the number of moduli is significantly larger: $2^{n-2e} - 2^{n-3e}$ instead of $2^{\frac{n}{2}}$. For moduli of size $30, 31$ this difference is significant.

2.7 Montgomery-Friendly

Pseudo-Mersenne can be seen as a type of modulo particularly well suited for Barrett's algorithm. Also, another class of moduli used initially by [33] and [34] and suited for Montgomery's modular reduction is the *Montgomery-Friendly* moduli. These are constructed as follow $P = K2^e - 1$ with $2^{n-e-1} < K \leq 2^{n-e}$ with $e \geq \frac{n}{2}$.

Algorithm 7: Multiplication modulo a Montgomery-Friendly [33]

Input : A, B, P, K, n with $0 \leq A, B < P = K2^e - 1$ with $2^{n-e-1} < K \leq 2^{n-e}$ and $e \geq \frac{n}{2}$
Output: C with $0 \leq C < P$ and $C = AB2^{-2e} \bmod P$
begin
  $C \leftarrow AB$
  $C \leftarrow [C]_e K + [C]^e$
  $C \leftarrow [C]_e K + [C]^e$
  if $C \geq P$ **then**
    return $C - P$
  end
  return C
end

As for Montgomery's multiplication, when using Montgomery-Friendly, one needs to use a dedicated representation which is the same as the one explained in Remark 1, simply using 2^{2e} instead of 2^n. Algorithm 7 has strong connections with Montgomery's initial work, Algorithm 1. However, it follows more closely its word based version [18]. In actual fact, it can be directly extracted from it: the quotient is simply extracted using a mask instead of a multiplication followed by a mask in the general method. This is due to the specific form of Montgomery-Friendly moduli.

Montgomery-Friendly moduli have been used for example for fast Number Theoretical Transform in [16], [35], [36].

For a survey on Montgomery-Friendly moduli, see [37].

Remark 3 (Ommiting final Correction). Redundancy can be allowed as well for Montgomery-Friendly moduli by omitting final correction. However, some further restrictions on e will be necessary. Indeed, if $P = K2^e - 1$ with $2^{n-e-1} < K \leq 2^{n-e}$ and $e \geq \frac{n+2}{2}$, one can notice that before final for $A, B < 2P$, one obtains a value for C which is inferior to $\frac{(2P-1)^2 + (2^e-1)K}{2^e} + (2^e-1)K$ which leads to

$$\begin{aligned} C &\leq \frac{\frac{(2P-1)^2}{2^e} + (2^e-1)K}{2^e} + (2^e-1)K \\ &\leq \frac{\frac{(2P-1)^2}{2^e} - K}{2^e} + K + (2^e-1)K \\ &\leq \frac{\frac{(2P-1)^2}{2^e} - K}{2^e} + 2^e K \end{aligned}$$

By using the property, that $K2^e = P + 1$, we obtain

$$\begin{aligned} C &\leq \frac{\frac{(2P-1)^2}{2^e} - K}{2^e} + P + 1 \\ &\leq \frac{\frac{(2P-1)^2 - K2^e}{2^e}}{2^e} + P + 1 \\ &\leq \frac{(2P-1)^2 - K2^e}{2^{2e}} + P + 1 \end{aligned}$$

Once again, if we use the property $K2^e = P + 1$, one obtains

$$\begin{aligned} C &\leq \frac{(2P-1)^2 - P - 1}{2^{2e}} + P + 1 \\ &\leq \frac{4P^2 - 4P + 1 - P - 1}{2^{2e}} + P + 1 \\ &\leq \frac{4P^2 - 5P}{2^{2e}} + P + 1 \\ &< \frac{4P^2 - 4P}{2^{2e}} + P + 1 \\ &< \frac{(P-1)4P}{2^{2e}} + P + 1 \\ &< \frac{(P-1)42^n}{2^{2e}} + P + 1 \\ &< (P-1)2^{n+2-2e} + P + 1 \end{aligned}$$

To have such computation guaranteed to be inferior to $2P$, one simply needs $2^{n+2-2e} \leq 1$. Consequently, one can fix $e \geq \frac{n+2}{2}$.
However, once again as operating only on $2n$ bits in a word size context, one obtains $P = K2^e - 1$ with $2^{n-e-2} < K \leq 2^{n-e-1}$ and $e \geq \frac{n+1}{2}$ guarantee correctness of the initial product AB.

3 New Modular Arithmetic

All existing methods listed in Section 2 are efficient algorithms and each offers an advantage depending of the context. However, those algorithms are generally highly efficient for multi-precision operation i.e. when $n >> 64, 32$. Nevertheless, there is a huge number of cases when modular arithmetic is performed on a single word size. In such context, bounding operands are relevant only to avoid overflow.

In this section, we proposed a new modular multiplication purposefully efficient when operating on a unique word. This algorithm will take advantage of the fact that multiplication on a $2n$ bits word are equivalent to a modular multiplication modulo 2^{2n} i.e. any multiplication AB are in this context $[AB]_{2n}$.

Algorithm 8: New Modular Multiplication

Input : A, B, P, R, n with $0 \leq A, B \leq P$ and $R = P^{-1} \bmod 2^{2n}$
Output: C with $0 \leq C < P$ and $C = AB(-2^{-2n}) \bmod P$
begin
  $C \leftarrow [([[ABR]_{2n}]^n + 1)P]^n$
  if $C = P$ **then**
    return 0
  end
  return C
end

Theorem 1 (Correctness). Let P an odd modulo with $P < \frac{2^n}{\phi}$ and $\phi = \frac{1+\sqrt{5}}{2}$, then Algorithm 8 is correct.

Proof 1 (of Algorithm 8).
Algorithm 8 first and main step is to compute $[([[ABR]_{2n}]^n + 1)P]^n$ i.e.

$$\left\lfloor \frac{\left(\left\lfloor \frac{(ABR \bmod 2^{2n})}{2^n} \right\rfloor + 1\right) P}{2^n} \right\rfloor.$$

i) Firstly, we check that $C < P$. It is known that $[[ABR]_{2n}]^n$ fits on $n$bits and therefore $\leq 2^n - 1$. Therefore, after Algorithm 8 first step, we obtain

$$\begin{aligned} C &= [([[ABR]_{2n}]^n + 1)P]^n \\ &\leq \frac{((2^n - 1) + 1)P}{2^n} \\ &\leq \frac{2^n P}{2^n} \\ &\leq P \end{aligned}$$

Consequently, after reduction, either $C = P$ or $C < P$. The final correction guarantee $C < P$.

ii) Secondly, we check that $C \equiv AB(-2^{-n}) \bmod P$. As P is odd, we know that there exist a $Q = ABP^{-1} \bmod 2^{2n}$. Consequently, we have

$$\begin{aligned} QP - AB &\equiv (ABP^{-1})P - AB \bmod 2^{2n} \\ &\equiv AB - AB \bmod 2^{2n} \\ &\equiv 0 \bmod 2^{2n} \end{aligned}$$

We obtain that $QP - AB$ is divisible by 2^{2n} and therefore that

$$AB(-2^{-2n}) \bmod P = \frac{QP - AB}{2^{2n}}.$$

We will note $Q_1 = \lfloor \frac{Q}{2^n} \rfloor$ and $Q_0 = Q - Q_1 2^n$ with $0 \leq Q_0 < 2^n$.
Now, we analyse $P2^n - Q_0P + AB$, knowing that $0 \leq A, B \leq P$, then we obtain $P2^n - Q_0P + AB \leq P2^n + P^2$. As by Theorem statement, we have $P < \frac{2^n}{\phi}$ with $\phi = \frac{1+\sqrt{5}}{2}$. Consequently, we obtain

$$\begin{aligned} P2^n - Q_0P + AB &< 2^n \frac{2^n}{\phi} + \left(\frac{2^n}{\phi}\right)^2 \\ &< \frac{2^{2n}}{\phi} + \frac{2^{2n}}{\phi^2} \\ &< 2^{2n} \frac{\phi + 1}{\phi^2}. \end{aligned}$$

Therefore, to guarantee than $P2^n - Q_0P + AB < 2^{2n}$, one need to check that $\frac{\phi+1}{\phi^2} = 1$. This is correct as

$$\begin{aligned} \phi^2 &= (\frac{1+\sqrt{5}}{2})^2 \\ &= \frac{1 + 2\sqrt{5} + 5}{4} \\ &= \frac{6 + 2\sqrt{5}}{4} \\ &= \frac{3 + \sqrt{5}}{2} \\ &= \frac{1 + \sqrt{5}}{2} + 1 \\ &= \phi + 1. \end{aligned}$$

Furthermore, as $Q_0 < 2^n$ and $A, B \geq 0$, we obtain as well that $P2^n - Q_0P + AB > 0$, we, consequently, obtain that

$$0 < \frac{P2^n - Q_0P + AB}{2^{2n}} < 1.$$

Therefore and because $QP - AB$ is divisible by 2^{2n},

$$\begin{aligned} \frac{QP - AB}{2^{2n}} &= \left\lfloor \frac{QP - AB}{2^{2n}} + \frac{P2^n - Q_0P + AB}{2^{2n}} \right\rfloor \\ &= \left\lfloor \frac{QP - AB + P2^n - Q_0P + AB}{2^{2n}} \right\rfloor \\ &= \left\lfloor \frac{(Q - Q_0 + 2^n)P}{2^{2n}} \right\rfloor \\ &= \left\lfloor \frac{(Q_1 2^n + 2^n)P}{2^{2n}} \right\rfloor \\ &= \left\lfloor \frac{(Q_1 + 1)P}{2^n} \right\rfloor \end{aligned}$$

As

$$Q_1 = \left\lfloor \frac{Q}{2^n} \right\rfloor = \left\lfloor \frac{(ABP^{-1} \bmod 2^{2n})}{2^n} \right\rfloor,$$

we obtain that

$$\begin{aligned} AB(-2^{-2n}) \bmod P &\equiv \frac{QP - AB}{2^{2n}} \\ &\equiv \left\lfloor \frac{(Q_1 + 1)P}{2^n} \right\rfloor \\ &\equiv \left\lfloor \frac{\left\lfloor \frac{(ABP^{-1} \bmod 2^{2n})}{2^n} \right\rfloor + 1}{2^n} \right\rfloor \end{aligned}$$

To finish this proof, we simply note that the last step of Algorithm 8 does not change the value of C modulo P. □

As for other methods, final correction can be omitted to allow redundancy. However, as redundancy aims to be minimal ($P + 1$ instead of P values are authorised), constraints for correctness are the same as the ones in Theorem 1[1].

Remark 4 (Dedicated Representation). One can also note that Algorithm 8 does not return $AB \bmod P$, but rather

1. For simplicity, Algorithm 8 already assumes than A and B could be equal to P.

$AB(-2^{-2n}) \bmod P$. As for Montgomery's multiplication (Algorithm 1), one will need to use a specific representation. However, it can be managed exactly as for Algorithm 1 and as explained in Remark 1.

Remark 5 (Multiplication by a Constant). An important remark is that Algorithm 8 first step is to perform a double multiplication, ABR. Often, one of the two values of a multiplication will be used multiple times. For example, for a modular exponentiation, $A^e \bmod P$, multiple multiplications by A will be performed. Therefore, using Algorithm 8, after multiplying AR, the first time, one can keep AR instead of A gaining one multiplication on any further modular multiplication involving A. This is a powerful advantage which can be used multiple times depending on which modular computation is being performed. Furthermore, any constant precomputed for an algorithm will fall under the same rule.

4 Comparison

To perform a precise comparison, the size and number of moduli on which each algorithm can perform will be given and we will count the different operations used by each modular multiplication method, namely:

- $\log_2(P)$, the size of possible moduli,
- $\log_2(\#P)$, the number of possible moduli,
- REP, the need of a specific representation indicated by T, true, or F, false,
- MUL, the number of word size multiplication,
- ADD, the number of word size addition/subtraction,
- IF, the number of time an instruction if is used which includes the cost of a comparison,
- SFT, the number of shift i.e. $[x]_k$. However, we should note that all shifts are noted as costly. However, on a 64bits processor, 32bits shifts are generally cheaper than other ones. As our new method uses only n shifts on $2n$ digit size no matter the modulo size used, it will allow a comparable gain when measured against other methods. This will be shown later in the timing section (Section 4.2),
- MSK, the number of mask used i.e. $[x]^k$. We note that our new proposal uses a theoretical mask. In reality, there aren't any as they are included in the natural modularity (modulo 2^{2n}) of word size multiplication/addition,
- TOT, the total number of operations required to perform a modular multiplication. We note that such number is not always representative and only timing test will help to fairly compare all those methods.
- CNT, the number of constant used by each algorithm to perform a modular multiplication,
- REG, the number of variable used by each algorithm. We note than it is not always trivial to evaluate this number. However, it is important to notice that our new method does not require to keep different parts of the product result, which is not the case in all other methods. This could be a key parameter for implementation on constrained devices.

To give a full comparison we also included information regarding moduli size and density on which each algorithm can perform and if algorithms are requiring dedicated representation.

Table 1 presents each algorithm advantages and drawback. Our new method is indicated on the same row for with/without correction as only the number of IFs is changing.

4.1 Example of Applications

In this section, we present four classical applications for modular arithmetic.

1) Algorithm 9 describes how to perform a modular exponentiation,
2) Algorithm 10 describes how to evaluate a modular polynomial evaluation,
3) Algorithm 11 describes how to perform a number theoretical transform,
4) Algorithm 12 describes how to convert a number from its residue number representation to its mixed radix representation.

4.1.1 Modular Exponentiation

Modular exponentiation is a widely used computation, especially for cryptography. Even if modular exponentiation are mainly used with moduli of larger size i.e $\log_2 P$ going from 200 to 4000 bits in general, it is still a good candidate to compare efficiency of different modular arithmetic. Algorithm 9 describes how to perform a modular exponentiation using a right-to-left method.

Algorithm 9: Right-to-Left Modular Exponentiation

Input : A, e, P with $0 \leq A, e < P$
Output: B with $0 \leq B < P$ and $B = A^e \bmod P$
begin
  $B \leftarrow 1$
  $C \leftarrow A$
  $k \leftarrow e$
  while $k > 1$ **do**
    if $k \bmod 2 = 1$ **then**
      $B \leftarrow BC \bmod P$
    end
    $C \leftarrow CC \bmod P$;
    $k \leftarrow \frac{k}{2}$;
  end
  $B \leftarrow BC \bmod P$;
  return B
end

Algorithm 9 is one of most standard ways to compute a modular exponentiation. However, we should note that there exists a lot of different methods to compute efficiently modular exponentiation. For surveys, see [38].

4.1.2 Polynomial Evaluation

Modular polynomial evaluation has multiple applications [9], [10], [11] and it corresponds to one of the most fundamental operation for polynomials. Algorithm 10 describes how to evaluate a modular polynomial using Horner's method [8].

Algorithm 10 gives a different set of needs for modular arithmetic compare to the modular exponentiation.

TABLE 1
Comparison of different methods for modular multiplication

Algorithm	$\log_2(P)$	$\log_2(\#P)$	REP	MUL	ADD	IF	SFT	MSK	TOT	CNT	REG	TOT
New method (8) w/wo corr.	n-0.694	n-1.694	T	3	1	1/0	2	0	7/6	2	1	3
New (8) for a constant w/wo corr.	n-0.694	n-1.694	T	2	1	1/0	2	0	6/5	1	1	2
Montgomery (1)	n-0.694	n-1.694	T	3	2	1	1	1	8	2	2	4
Montgomery (1) wo corr.	n-2	n-3	T	3	1	0	1	1	6	2	2	4
Barrett (2)	n-1	n-1	F	3	2	2	2	0	9	2	2	4
NFLlib (6)	n-1	n-3	F	3	3	1	3	1	11	2	2	4
Pseudo-Mersenne (4)	n	$\frac{n}{2}$	F	3	3	1	2	2	11	2	2	4
Pseudo-Mersenne (4) wo corr.	n-1	$\frac{n-4}{2}$	F	3	2	0	2	2	9	2	2	4
Montgomery-Friendly (7)	n	$\frac{n}{2}$	T	3	3	1	2	2	11	2	2	4
Montgomery-Friendly (7) wo corr.	n-1	$\frac{n-3}{2}$	T	3	2	0	2	2	9	2	2	4
Generalized Mersenne (5)	n	$\frac{\log_2(n)-2}{2}$	F	1	5	1	4	2	11	1	2	3
Generalized Mersenne (5) wo corr	n-1	$\frac{\log_2(n)-6}{2}$	F	1	4	0	4	2	9	1	2	3
Mersenne (3)	n	0	F	1	2	1	1	1	6	0	2	2
Forced Mersenne (3)	n	0	F	1	2	0	2	2	7	0	2	2

Algorithm 10: Polynomial Evaluation

Input : A, P and $F(X)$ with $0 \leq A < P$
Output: B with $0 \leq B < P$ and $B = F(A) \bmod P$
begin
  $B \leftarrow F_{\deg F}$
  for $i \leftarrow \deg F - 1$ **to** 0 **by** -1 **do**
    $B \leftarrow (BA) \bmod P$
    $B \leftarrow (B + F_i) \bmod P$
  end
  return B
end

4.1.3 Number Theoretical Transform

Number Theoretical Transform (NTT) is an algorithm to evaluate efficiently a polynomial $A(X)$ for all roots of unity in $\mathbb{F}_P$ i.e.

$$A(X) \rightarrow < A(\zeta_1) \bmod P, \ldots, A(\zeta_n) \bmod P >$$

with $\Phi(\zeta_i) \bmod P = 0$, $\Phi(X) = X^k + 1$ and $k = 2^t$.

NTT has became the key operation for a wide number of lattice based cryptosystems, see [35] for a classic implementation. We note that we are focusing on NTT operating on $\Phi(X) = X^{2^t} + 1$ as it is the most used in cryptography. Other versions exist for $X^{2^t} - 1$.

Lattice based cryptography has started to use more and more lattice based on rings or modules. In such lattices, operations are performed on polynomial ring modulo a small number, 10 to 20 bits. The polynomial ring is generally quotiented by a cyclotomic polynomial, $X^{2^t} + 1$. Consequently, the main cost of encrypting/decrypting or signing/verifying signature is to perform NTT to make efficient polynomial multiplications quotiented by a cyclotomic polynomial. U.S. National Institute of Standards and Technology standardization process for post-quantum cryptography [13] is currently keeping, after 3 rounds of selections, 3 lattice based cryptosystems based on NTT [16], [17], [39] on the 4 finalist post-quantum cryptosystems and 2 lattice based signatures based on NTT [14], [40] on the 3 finalist for post-quantum signatures. Algorithm 11 describes how to perform a number theoretical transform.

Algorithm 11: Number Theoretical Transform

Input : $A(X)$ with $\deg A < k$ and Z a k root of unity, $Z^k + 1 \equiv 0 \bmod P$.
Output: $B_i = A(Z_i) \bmod P$ with $0 \leq B_i < P$
begin
  for $i \leftarrow 1$ **to** k **do**
    $B_i \leftarrow A_i$
  end
  $c \leftarrow 1$
  $l \leftarrow \frac{k}{2}$
  while $l > 0$ **do**
    for $s \leftarrow 0$ **to** $k - 1$ **by** $2l$ **do**
      for $j \leftarrow s$ **to** $s + l - 1$ **do**
        $tmp \leftarrow B_{j+l} Z^{rev(c)} \bmod P$
        $B_{j+l} \leftarrow (B_j - tmp) \bmod P$
        $B_j \leftarrow (B_j + tmp) \bmod P$
      end
      $c \leftarrow c + 1$
    end
    $l \leftarrow \frac{l}{2}$
  end
  return B
end

Algorithm 11 uses some precomputed value $Z^{rev(c)} \bmod P$ where $rev(c)$ is the k bit-for-bit reverse of c.

Algorithm 11 offers another perspective (in term of needs for modular arithmetic) compare to Algorithm 9 or Algorithm 10. Especially, it uses modular multiplication by some precomputed constants.

4.1.4 RNS to MRS Conversion

Residue Number System (RNS) is an important number system used in a lot of applications from signal processing [41] to cryptography [42].

A RNS basis of size k is a set of k moduli M_i, pairwise co-prime i.e.

$$\forall i \neq j, GCD(M_i, M_j) = 1.$$

A number

$$0 \leq X < \prod M_i$$

is represented by its residual modulo M_i i.e.

$$\overline{X} =< X \bmod M_1, \ldots, X \bmod M_k > .$$

Residue number systems offer a natural paralelization which makes it highly efficient for some operations. However, as it is not a positional number system, it does not offer a trivial way to compare two numbers ($\overline{X} \overset{?}{<} \overline{Y}$). One of the possibility to compare two numbers in their RNS representation is to first convert them from RNS to Mixed Radix System.

Mixed Radix System (MRS) is a positional number system where a number $0 \leq X < \prod M_i$ is represented by $\widetilde{X} =< X_1, \ldots, X_k >$ such that $0 \leq X_i < M_i$ and

$$X = \sum_{i=1}^{n} X_i \prod_{j=1}^{i-1} M_j$$

allowing to compare two numbers. It is also used to convert a number represented in one RNS basis to another one [43], even if some other methods are generally more efficient [25], [44].

Algorithm 12 describes how to convert a number from its residue number representation to its mixed radix representation.

Algorithm 12: RNS to MRS

Input : $\overline{X}$ representing X in a RNS Basis $[M_i]$
Output: $\widetilde{X}$ representing X in MRS with basis $[M_i]$
begin
  for $i \leftarrow 0$ **to** $n-1$ **do**
    for $j \leftarrow i+1$ **to** $n-1$ **do**
      $X_j \leftarrow (X_j - X_i) \bmod M_j$
      $X_j \leftarrow X_j(M_i^{-1}) \bmod M_j$
    end
  end
  return $\widetilde{X}$
end

Algorithm 12 uses some precomputed value $(M_i^{-1}) \bmod M_j$. The interesting aspect of Algorithm 12 is that in comparison to the other 3 applications previously presented, it requires to perform modular multiplication on different moduli. We note that other algorithms are more relevant in for RNS, however they often perform simple vector matrix multiplication followed by a modular reduction. Therefore, the quality of modular arithmetic is less important. Hence, those algorithms - such as conversion between two different RNS basis - are less relevant for comparing modular arithmetic methods.

4.2 Timings

In this section, we present result of tests performed on a Linux 5.8.0-26-generic with Intel Core i7-8565U CPU (1.80GHz) and compiled with g++ 10.2.0 with -Ofast -flto -fwhole-program -march=native -std=c++17 options. Tests have been performed 10^7 times on Algorithm 9, Algorithm 10, Algorithm 11 and Algorithm 12. Polynomial evaluations were taken with a degree less than 64, Number Theoretical Transform on cyclotomic polynomial $x^{16}+1$ and RNS basis were taken with 32 moduli.

Macro have been used to guarantee equivalence and same moduli have been used when possible i.e. at least for all 3 generalist algorithms i.e. Montgomery's (Algorithm 1), Barrett's (Algorithm 2) and our new method (Algorithm 8).

4.2.1 Comparison with Montgomery's Method

In this section, we compare our new method with Montgomery's method.

Remark 6 (Avoiding Using Dedicated Representation). Both algorithms require a specific representation (Remark 1 and Remark 4). However, this representation is only necessary for Modular Exponentiation (Algorithm 9). For the three others applications, as variable are only multiplied by constants, only constants will be used in specific representations. Effectively, for Montgomery modular multiplication for example, if a variable a is multiplied by $\widetilde{c}$ where $\widetilde{c}$ represents c in Montgomery representation i.e. $\widetilde{c} = c2^n \bmod P$, then

$$\begin{aligned} MONT(a, \widetilde{c}) &= a\widetilde{c}2^{-n} \bmod P \\ &= ac2^n 2^{-n} \bmod P \\ &= ac \bmod P. \end{aligned}$$

Consequently, for Polynomial Evaluation, Number Theoretical Transform and Residue Number System to Mixed Radix System Conversion, as variables are only multiplied by constants, one does not need to put variables in Montgomery representation and the same goes for our new method. Only constant values will be prepare in Montgomery representation. By avoiding to represent variables in Montgomery representations, one gains two modular multiplications for each variable i.e. one in input, one in output (Remark 1).[2]
Furthermore, in our new method constants will be multiply by R ($C' = CR$) to transform ACR by AC' in the first step of Algorithm 8 and gain a word size multiplication as explain in Remark 5.

Table 2 shows average number of cycles for our new and Montgomery's algorithms.

Additionally, it shows that our new method clearly outperformed Montgomery's for word size modular arithmetic: for each applications, Montgomery's method is slower by 10% to 50%. It is only when Montgomery's method is used without correction and consequently only for smaller

2. Furthermore, some works have been done to use modular exponentiation with Montgomery algorithm without Montgomery's representation as well, using Barrett's algorithm to operate correction [45]. However, this work did not allow faster computation as it was focusing on side-channel resistance.

TABLE 2
Comparison between our and Montgomery's modular arithmetic

	$\log_2 P$	New (8)	Montgomery (1)	
			w corr.	wo corr.
EXP	30	128.9	178.3 (+38%)	126.5 (-1.9%)
	31	134.5	189.8 (+41%)	n/a
	32	137.2	169.6 (+24%)	n/a
EVL	30	330.2	450.4 (+36%)	357.4 (+8.2%)
	31	330.1	450.1 (+36%)	n/a
	32	329.5	450.0 (+36%)	n/a
NTT	30	189.8	204.8 (+7.9%)	202.2 (+6.6%)
	31	188.3	210.9 (+12%)	n/a
	32	189.0	210.9 (+12%)	n/a
RNS	30	944.9	1366 (+44%)	1187 (+26%)
	31	946.9	1390 (+47%)	n/a
	32	954.4	1419 (+48%)	n/a

TABLE 3
Comparison between our and Montgomery's modular arithmetic for NTT

	$\log_2 P$	New (8)	Montgomery (1)	
			w corr.	wo corr.
NTT with Correction	30	189.8	204.8 (+7.9%)	202.2 (+6.6%)
	31	188.3	210.9 (+12%)	n/a
	32	189.0	210.9 (+12%)	n/a
Without Correction	30	161.5	173.0 (+7.1%)	n/a
	31	159.9	173.7 (+8.6%)	n/a

moduli, that Montgomery's method offers a reasonably close efficiency for some applications. Moreover, Montgomery's method could offer some instances (for Modular Exponentiation) were it could eventually be faster than our new method. However, for most applications, our new method offers clear gain to Montgomery's which is up to 25% slower for RNS to MRS, and up to 48% slower when not allowed to use redundancy.

Furthermore, it is important to note that if redundancy is used by Montgomery's reduction, then redundancy can not be use to optimize the applications themselves. As an example, our new method can be used to implement NTT with a modification where the two modular additions/subtractions of the variable tmp are not reduced i.e. performing the addition/substation without correcting it afterwards. This is a gain not available if redundancy is already used by Montgomery's method without correction.

Table 3 shows average number of cycles for our new algorithm and Montgomery's one for NTT and NTT without correction.

Table 3 clearly shows that redundancy is better used by optimizing NTT itself. The advantage between new method and Montgomery's one is stable for NTT and sits around 8% gain in our new method.

4.2.2 Comparison with Barrett's Method

Table 4 shows average number of cycles for our new algorithm and Barrett's one.

Barrett's is offering slower timing than our method by at least 27% and up to 85% depending of applications. As for Montgomery's without correction, Barrett cannot offer solutions for 32bits moduli.

TABLE 4
Comparison between our and Barrett's modular arithmetic

	$\log_2 P$	New (8)	Barrett (2)
EXP	30	128.9	168.0 (+30%)
	31	134.5	171.0 (+27%)
	32	137.2	n/a
EVL	30	330.2	473.9 (+44%)
	31	330.1	461.6 (+40%)
	32	329.5	n/a
NTT	30	189.8	229.4 (+21%)
	31	188.3	228.6 (+22%)
	32	189.0	n/a
RNS	30	944.9	1723 (+82%)
	31	946.9	1753 (+85%)
	32	954.4	n/a

TABLE 5
Comparison between New method and Mersenne based arithmetic

	$\log_2 P$	New	Mersenne	
			Classic	Forced
EXP	30	128.9	146.3 (+14%)	134.2 (+4.1%)
	31	134.5	151.3 (+12%)	138.6 (+3.0%)
	32	137.2	154.3 (+12%)	144.2 (+5.1%)
EVL	30	330.2	339.8 (+2.9%)	361.5 (+9.5%)
	31	330.1	337.0 (+2.1%)	361.4 (+9.5%)
	32	329.5	335.9 (+1.9%)	296.7 (-10%)

4.2.3 Comparison with Mersenne Moduli

Mersenne based arithmetic is highly limited and cannot be used for applications requiring multiple moduli of the same size (RNS to MRS) or when requiring a prime number (NTT). As explained in Section 2.3, two options are available when implemented Mersenne based arithmetic:

- a classic version, with an IF for correction after reduction,
- a forced version, imposing a second reduction whatever is the result of the first one to avoid the IF.

Table 5 shows average number of cycles for our new method and Mersenne based arithmetic.

We observe that forced approach is outperforming classical approach when $n = 32$. Indeed, in such situation, the shift become cheaper than in other sizes. This is also one of the reasons why our method outperforms Mersenne based arithmetic in most of cases: our approach uses two shifts of 32bits even when operating on smaller size moduli.

Nevertheless, in the case of polynomial evaluation on a 32bits Mersenne number, Mersenne forced approach outperforms ours. This is the one and only case.

4.2.4 Comparison with Pseudo-Mersenne

Pseudo-Mersenne numbers offer enough moduli for them to be used in each application we are proposing. However,

TABLE 6
Comparison between New method and Pseudo-Mersenne based arithmetic

	$\log_2 P$	New (8)	Pseudo-Mersenne (4)	
			w corr.	wo corr.
EXP	30	128.9	180.9 (+40%)	152.4 (+18%)
	31	134.5	189.2 (+40%)	157.2 (+17%)
	32	137.2	187.0 (+36%)	n/a
EVL	30	330.2	433.3 (+31%)	418.8 (+27%)
	31	330.1	441.8 (+34%)	405.1 (+23%)
	32	329.5	429.2 (+30%)	n/a
NTT	30	189.8	224.9 (+18%)	214.4 (+13%)
	31	188.3	224.0 (+19%)	214.5 (+14%)
	32	189.0	231.0 (+22%)	n/a
RNS	30	944.9	1440 (+52%)	1288 (+36%)
	31	946.9	1441 (+52%)	1310 (+38%)
	32	954.4	1429 (+50%)	n/a

TABLE 7
Comparison between our and NFLlib's modular arithmetic

	$\log_2 P$	New (8)	NFLlib (6)
EXP	30	128.9	188.6 (+46%)
	31	134.5	176.3 (+31%)
	32	137.2	n/a
EVL	30	330.2	429.5 (+30%)
	31	330.1	436.3 (+32%)
	32	329.5	n/a
NTT	30	189.8	216.4 (+14%)
	31	188.3	215.1 (+14%)
	32	189.0	n/a
RNS	30	944.9	1414 (+50%)
	31	946.9	1445 (+53%)
	32	954.4	n/a

table 6 clearly shows that even without correction, pseudo-Mersenne based arithmetic is slower than our by at lease 17% and up to 38% depending of applications, and up to 52% when redundancy is not available.

4.2.5 Comparison with NFLlib's Method

Table 7 shows average number of cycles for our new algorithm and NFLlib's one.

NFLlib's is offering slower timing than our method by at least 13% and up to 53% depending of applications. As previously stated, NFLlib's solution can not offer solutions for 32bits moduli.

4.2.6 Comparison with Montgomery-Friendly

Montgomery-Friendly arithmetic offers enough moduli for most of our applications. However, its specific structure does not allow it to have prime moduli with 2^k roots of unity (for NTT computation). A solution is possible by using an extended class of Montgomery-friendly numbers i.e. $P = K2^e + 1$ with $2^{n-e-1} < K \leq 2^{n-e}$. Such numbers are close from Montgomery-Friendly, see [37] for more details. As this work, is investigating already 3 generalist algorithms and 4 special classes of moduli, an additional class has not be pursued. Furthermore, this extended class should provide a highly close efficiency than Montgomery-Friendly.

TABLE 8
Comparison between New method and Montgomery's Friendly based arithmetic

	$\log_2 P$	New (8)	Montgomery-Friendly (7)	
			w corr.	wo corr.
EXP	30	128.9	189.5 (+47%)	146.5 (+14%)
	31	134.5	206.0 (+53%)	151.4 (+12%)
	32	137.2	194.9 (+42%)	n/a
EVL	30	330.2	477.7 (+44%)	397.3 (+20%)
	31	330.1	478.2 (+45%)	396.9 (+20%)
	32	329.5	481.7 (+46%)	n/a
RNS	30	944.9	1485 (+57%)	1227 (+30%)
	31	946.9	1464 (+54%)	1215 (+28%)
	32	954.4	1492 (+56%)	n/a

TABLE 9
Comparison between New method and Generalized Mersenne based arithmetic

	$\log_2 P$	New (8)	Generalized Mersenne (5)	
			w corr.	wo corr.
EXP	30	128.9	180.3 (+40%)	177.2 (+38%)
	31	134.5	186.0 (+38%)	183.7 (+36%)
	32	137.2	197.1 (+44%)	n/a
EVL	30	330.2	378.9 (+15%)	349.0 (+5.7%)
	31	330.1	375.3 (+14%)	351.9 (+6.6%)
	32	329.5	370.0 (+12%)	n/a

As for Pseudo-Mersenne, table 8 clearly shows that even without correction, Montgomery-Friendly based arithmetic is slower by at lease 12% and up to 30% depending of applications, and up to 57% when not allowed redundancy.

We note that Montgomery-Friendly numbers seem a bit slower than pseudo-Mersenne. However, when redundancy is allowed Montgomery-Friendly numbers outperformed Pseudo-Mersenne.

4.2.7 Comparison with Generalized Mersenne

Generalized Mersenne numbers offer a wider range of moduli that Mersenne number. However, as for Mersenne numbers, they can be applied for only 2 of our 4 applications. This is due to the smaller number of moduli available for one size.

Table 9 shows average number of cycles for our new algorithm and Generalized Mersenne based arithmetic.

Generalized Mersenne numbers offer some advantages compare to Pseudo-Mersenne numbers for polynomial evaluation. However, it is still slower on all applications, even using redundancy, compare to our new approach.

4.2.8 Comparison Summary

Table 10 summarizes timing analysis by comparing our new method to the best available existing solutions: we kept the algorithm which returns the smallest average number of cycles. *wo c.* indicate when algorithms are using without final correction.

TABLE 10
Comparison between New method and Best Existing Solution on an Intel Core i7

	$\log_2 P$	New (8)	Best Existing Solution # cycles	Best Existing Solution Algorithm
EXP	30	128.9	126.5 (-1.9%)	Montgomery (1) wo c.
	31	134.5	138.6 (+3.0%)	Forced Mersenne (3)
	32	137.2	144.2 (+5.1%)	Forced Mersenne (3)
EVL	30	330.2	339.8 (+2.9%)	Mersenne (3)
	31	330.1	337.0 (+2.1%)	Mersenne (3)
	32	329.5	296.7 (-10%)	Forced Mersenne (3)
NTT	30	189.8	202.2 (+6.6%)	Montgomery (1) wo c.
	31	188.3	210.9 (+12%)	Montgomery (1)
	32	189.0	210.9 (+12%)	Montgomery (1)
RNS	30	944.9	1187 (+26%)	Montgomery (1) wo c.
	31	946.9	1215 (+28%)	M.-Friendly (7) wo c.
	32	954.4	1419 (+48%)	Montgomery (1)

TABLE 11
Comparison between New method and Best Existing Solution on an Intel Xeon Gold

	$\log_2 P$	New (8)	Best Existing Solution # cycles	Best Existing Solution Algorithm
EXP	30	101.8	101.1 (-0.71%)	Montgomery (1) wo c.
	31	105.2	117.0 (+11%)	M.-Friendly (7) wo c.
	32	109.9	132.5 (+20%)	Barrett (2)
EVL	30	246.1	231.9 (-5.8%)	Forced Mersenne (3)
	31	246.3	231.0 (-6.2%)	Forced Mersenne (3)
	32	246.2	227.3 (-7.7%)	Forced Mersenne (3)
NTT	30	171.0	180.0 (+5.3%)	Montgomery (1) wo c.
	31	170.4	181.8 (+6.7%)	NFLlib (6)
	32	172.6	184.5 (+6.9%)	Montgomery (1)
RNS	30	764.6	907.2 (+19%)	Montgomery (1) wo c.
	31	765.1	934.8 (+22%)	M.-Friendly (7) wo c.
	32	774.9	1096 (+42%)	Montgomery (1)

To confirm competitivity of our new method, we performed the exact same tests on a second machine, a Linux 5.4.0-40-generic with Intel Xeon Gold 6128 CPU (3.40GHz). Table 11 presents those results confirming the same behaviour even if some small change can be observed. If some applications are close or even slower, a large number of applications are faster when implemented with our new method with even some instances were every other solutions can go slower by 48% on the Intel Core i7 and by 42% on the Intel Xeon Gold.

5 Conclusion

In this work, we proposed a new modular multiplication designed to be computed on a unique word. It offers some strong advantages compare to other existing solutions:

- a computational cost low in word size instructions,
- possibility to be computed on one word size register,
- a dedicated - with further efficiency - modular multiplication by a constant.

We showed that our new method outperformed all existing solutions in the majority of cases.

However, further investigation should be done to evaluate if such approach can accelerate application using not a family of moduli but rather one specific moduli. Lattice based cryptosystems such as New Hope [15] or Kyber [16] use one unique prime moduli, $2^{13} + 2^{12} + 1$ for New Hope, $2^{13} - 2^9 + 1$ for Kyber. With such specific moduli, [15] and [16] have used a combination of Lazy Barrett's reduction and Montgomery's ones, to minimize the number of operations required to perform a NTT. In those works, usage of redundancy for NTT have been optimized for those moduli. Furthermore, AVX instructions have to accelerate further computation.

There is the potential to further investigate how our new method perform on NTT on such moduli when maximizing redundancy and when using AVX instructions.

Nevertheless, we think that our new method offers a highly competitive option for word size modular arithmetic.

References

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, nov 1976.

[2] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb 1978.

[3] National Institute for Standards and Technology, "Digital Signature Standard (DSS)," Jun 2009.

[4] V. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology, proceeding's of CRYPTO'85*, ser. LNCS, vol. 218. Springer-Verlag, 1986, pp. 417–426.

[5] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, January 1987.

[6] J.-M. Couveignes, "Hard homogeneous spaces," Cryptology ePrint Archive, Report 2006/291, 2006, https://eprint.iacr.org/2006/291.

[7] D. Jao and L. D. Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," in *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, ser. Lecture Notes in Computer Science, B. Yang, Ed., vol. 7071. Springer, 2011, pp. 19–34.

[8] D. E. Knuth, *The Art of Computer Programming, Vol. 2. Seminumerical Algorithms*, 3rd ed. Addison-Wesley, 1997.

[9] M. B. Monagan and B. Tuncer, "The complexity of sparse hensel lifting and sparse polynomial factorization," *J. Symb. Comput.*, vol. 99, pp. 189–230, 2020.

[10] J. Hu and M. B. Monagan, "A fast parallel sparse polynomial GCD algorithm," in *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016*, S. A. Abramov, E. V. Zima, and X. Gao, Eds. ACM, 2016, pp. 271–278.

[11] A. Shamir, "How to share a secret," *CACM: Communications of the ACM*, vol. 22, pp. 612–613, 1979.

[12] D. Micciancio and O. Regev, "Lattice-based cryptography," in *Post-quantum Cryprography*, D. J. Bernstein, J. Buchmann, and E. Dahmen, Eds. Springer, 2008.

[13] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, "Status report on the second round of the NIST post-quantum cryptography standardization process," National Institute of Standards and Technology, Tech. Rep. NIST IR 8309, Jul. 2020.

[14] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 1, pp. 238–268, 2018. [Online]. Available: https://doi.org/10.13154/tches.v2018.i1.238-268

[15] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange – a new hope," 2016, pp. 327–343.

[16] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "Crystals - kyber: A cca-secure module-lattice-based kem," 2018, pp. 353–367.

[17] J.-P. D'Anvers, A. Karmakar, S. Sinha Roy, and F. Vercauteren, "Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10831 LNCS, pp. 282–305, 2018.

[18] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, Apr 1985.

[19] C. D. Walter, "Precise bounds for Montgomery modular multiplication and some potentially insecure RSA moduli," in *CT-RSA*, 2002, pp. 30–39.

[20] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology – CRYPTO '86*, ser. LNCS, A. M. Odlyzko, Ed., vol. 263. Springer-Verlag, 1986, pp. 311–326.

[21] J.-J. Quisquater, "Encoding system according to the so-called rsa method, by means of a microcontroller and arrangement implementing this system," U.S. Patent number # 5, 166–978, 1991.

[22] M. Joye, "On quisquater's multiplication algorithm," in *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, ser. Lecture Notes in Computer Science, D. Naccache, Ed., vol. 6805. Springer, 2012, pp. 3–7.

[23] R. Crandall, "Method and apparatus for public key exchange in a cryptographic system," U.S. Patent number 5159632, 1992.

[24] Certicom Research, "SECG SEC2: Recommended Elliptic Curve Cryptography Domain Parameters," Sep 2000.

[25] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower architecture for fast parallel montgomery multiplication," in *Proc. Int'l Conf. Theory and Application of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT 2000)*, ser. Lecture Notes in Computer Science, Springer, Ed., no. 1807, 2000.

[26] J.-C. Bajard, L. Imbert, and T. Plantard, "Improving Euclidean division and modular reduction for some classes of divisors," in *37th IEEE Asilomar Conference on Signals, Systems, and Computers*, November 2003.

[27] J. Chung and A. Hasan, "More generalized Mersenne numbers," in *Selected Areas in Cryptography – SAC 2003*, ser. LNCS, M. Matsui and R. Zuccherato, Eds., vol. 3006. Ottawa, Canada: Springer-Verlag, Aug 2003, (to appear).

[28] J. Solinas, "Generalized Mersenne numbers," Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada, Research Report CORR-99-39, 1999.

[29] National Institute for Standards and Technology, *FIPS PUB 186-2: Digital Signature Standard (DSS)*, Gaithersburg, MD, USA, Jan. 2000.

[30] J.-C. Bajard, M. Kaihara, and T.Plantard, "Selected RNS bases for modular multiplication," in *ARITH'19: Proceedings of the 19th IEEE Symposium on Computer Arithmetic*. IEEE Computer Society, June 2009.

[31] C. A. Melchor, J. Barrier, S. Guelton, A. Guinet, M. Killijian, and T. Lepoint, "Nfllib: Ntt-based fast lattice library," in *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, ser. Lecture Notes in Computer Science, K. Sako, Ed., vol. 9610. Springer, 2016, pp. 341–356.

[32] N. Moller and T. Granlund, "Improved division by invariant integers," *IEEE Trans. Computers*, vol. 60, no. 2, pp. 165–175, 2011.

[33] M. Hamburg, "Fast and compact elliptic-curve cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 309, 2012.

[34] J. W. Bos, C. Costello, H. Hisil, and K. E. Lauter, "Fast cryptography in genus 2," in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, ser. Lecture Notes in Computer Science, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, 2013, pp. 194–210.

[35] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, ser. Lecture Notes in Computer Science, S. Foresti and G. Persiano, Eds., vol. 10052, 2016, pp. 124–139. [Online]. Available: https://doi.org/10.1007/978-3-319-48965-0_8

[36] G. Seiler, "Faster avx2 optimized ntt multiplication for ring-lwe lattice cryptography," Cryptology ePrint Archive, Report 2018/039, 2018.

[37] J. Bajard and S. Duquesne, "Montgomery-friendly primes and applications to cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 665, 2020. [Online]. Available: https://eprint.iacr.org/2020/665

[38] A. Menezes, P. C. V. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[39] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory (ANTS 1998), Lecture Notes in Computer Science 1423, Springer-Verlag*, 1998, pp. 267–288.

[40] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirch-ner, V. Lyubashevsky, T. Pornin, T. Ricos-set, G. Seiler, W. Whyte, , and Z. Zhang, "Falcon: Fast-fourier lattice-basedcompact signatures over NTRU," Submission to the NIST's post-quantum cryptography standardization process, 2017.

[41] G. A. Jullien and W. C. Miller, "Application of the residue number system to computer processing of digital signals," in *IEEE Symposium on Computer Arithmetic ARITH 4*, 1978, pp. 220–225.

[42] J.-C. Bajard and L. Imbert, "A full RNS implementation of RSA," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 769–774, 2004.

[43] J.-C. Bajard and T. Plantard, "RNS bases and conversions," in *SPIE'04: Advanced Signal Processing Algorithms, Architectures and Implementations XIV*, August 2004.

[44] J.-C. Bajard, L.-S. Didier, and P. Kornerup, "An RNS Montgomery modular multiplication algorithm," *IEEE Transactions on Computers*, vol. 47, pp. 766–776, 1998.

[45] A. Lesavourey, C. Nègre, and T. Plantard, "Efficient randomized regular modular exponentiation using combined montgomery and barrett multiplications," in *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4: SECRYPT, Lisbon, Portugal, July 26-28, 2016*, C. Callegari, M. van Sinderen, P. G. Sarigiannidis, P. Samarati, E. Cabello, P. Lorenz, and M. S. Obaidat, Eds. SciTePress, 2016, pp. 368–375.

Thomas Plantard is a senior research fellow at the Institute of Cybersecurity and Cryptology at the University of Wollongong. He has worked for more than 10 years at the University of Wollongong on various topics related to computer arithmetic, algorithmic number theory, computer algebra and cryptology, in particular modular arithmetic, number system and lattice based cryptography.