HAL
archives-ouvertes.fr

RNS arithmetic in Fpk and application to fast pairing computation

Sylvain Duquesne

▶ To cite this version:

Sylvain Duquesne. RNS arithmetic in Fpk and application to fast pairing computation. Journal of Mathematical Cryptology, De Gruyter, 2011, 5 (1), pp.51-88. 10.1515/jmc.2011.006 . hal-00687220

HAL Id: hal-00687220
https://hal.archives-ouvertes.fr/hal-00687220

Submitted on 11 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives| 4.0 International License

J. Math. Cryptol. **5** (2011), 51–88
DOI 10.1515/JMC.2011.006 © de Gruyter 2011

RNS arithmetic in $\mathbb{F}_{p^k}$ and application to fast pairing computation

Sylvain Duquesne

Communicated by Alfred Menezes

Abstract. In this work, we are interested in arithmetic on large prime field and their extensions of small degree. We explain why it is very interesting to use RNS arithmetic for the base field $\mathbb{F}_p$ when computations in $\mathbb{F}_{p^k}$ have to be done, essentially thanks to lazy reduction. This is for example the case for pairing computations on ordinary curves (as MNT or BN curves). We show that using RNS can considerably decrease the number of basic operations required for a pairing computation in many popular situations.

Keywords. Extension filed a arithmetic, Residue Number System, lazy reduction, elliptic curves, pairing.

2010 Mathematics Subject Classification. 11T55, 11T71, 14G50, 94A60.

1 Introduction

In recent years, pairing based write cryptography became more and more popular. Thus efficient software and hardware implementation are necessary. For some time past, ordinary curves superseded supersingular curves [14, 50] on large prime fields. Such pairings involve many arithmetic in extensions of small degrees of the base field. Hence one approach angle for efficient pairing computation is the optimization of extension field arithmetic. The Residue Number System (RNS) has already been introduced in elliptic curve cryptography in [5, 6]. This is a system to represent numbers which uses the Chinese Remainder Theorem. It is recalled in Section 2. Contrary to standard arithmetic in $\mathbb{F}_p$, the RNS introduces a large gap of complexity between the multiplication and the reduction step of a modular multiplication. In extension fields, we show in Section 3 how lazy reduction allows to considerably decrease number of reduction. Hence the RNS is particularly well suited for extension field arithmetic and consequently for pairing computations. In Section 4, after some background on pairings and their computation, we study in detail the complexities of RNS-based implementations of current pairings (Tate,

This work was supported by the French ANR projects no. 07-BLAN-0248 “ALGOL” and 09-BLAN-0020-01 “CHIC”.

Ate, R-Ate) and compare it to standard complexities. For this, we made some assumptions (for example on the cost of RNS multiplications or on the negligibility of additions compared to multiplications) so that, in practice, our results will fluctuate depending on the implementation platform. We will concentrate on MNT and BN curves, which are the most popular today for embedding degrees 6 and 12 since their cardinality can be prime or almost prime contrary to other constructions in the literature. In all cases, we obtain significant theoretical savings so that the RNS is very promising for future pairing implementations, especially in hardware.

2 Efficient arithmetic on prime fields

2.1 Modular multiplication

Elliptic curve arithmetic over $\mathbb{F}_p$ mainly involves modular multiplications modulo p. Such a modular multiplication can be decomposed into one classic multiplication followed by one modular reduction. Because of the small size of numbers used in elliptic curve cryptography (192 to 512 bits, i.e., 6 to 16 32-bit words), the multiplication is performed by a common method. Let us consider a and b, as two n-word integers given in radix representation (i.e., $x = \sum_{i=0}^{n} x_i \mathbf{B}^i$ with $0 \leq x_i < \mathbf{B}$). Then ab can be computed by a succession of word multiplications and additions (which will be considered in the following as basic word operations). We can summarize this by the equation

$$ab = b_0 a + \mathbf{B}(b_1 a + \mathbf{B}(b_2 a + \cdots + \mathbf{B} b_n a)\ldots).$$

The complexity is then n^2 word multiplications. We note that for the current ECC key sizes, Karatsuba or Toom–Cook approaches are not competitive as discussed in the study made by the GMP group [1].

The reduction of an integer modulo p consists of finding the remainder of the Euclidean division of this integer by p. This operation is costly. It can be substantially sped up by using the Montgomery reduction or by using a special modulo.

Montgomery general reduction algorithm

In [48], Montgomery proposed to substitute the reduction modulo p by a division by a power of the radix $\mathbf{B}$ (which is a simple shift). The result is not exactly $a \mod p$ but $a\mathbf{B}^{-n} \mod p$. This problem can be overcome by using Montgomery representation where $a' = a \times \mathbf{B}^n \mod p$.

Algorithm 2.1. Montgomery$_p(c)$
Require: $c(= ab) < p\mathbf{B}^n$, $\mathbf{B}^{n-1} \leq p < \mathbf{B}^n$,
a precomputed value $(-p^{-1} \bmod \mathbf{B}^n)$.

Ensure: r such that $r \equiv c\mathbf{B}^{-n} \pmod{p}$ and $r < 2p$.
$q \leftarrow -c \times p^{-1} \bmod \mathbf{B}^n$
$r \leftarrow (c + qp)/\mathbf{B}^n$

The complexity of this reduction is $n^2 + n$ word operations [19]. For $a < \mathbf{B}^n$, its Montgomery representation is obtained via Aslgorithm 2.1 with $c = a \times (\mathbf{B}^n \bmod p)$. In the same way, if a' is the Montgomery representation of a, then we recover a using Algorithm 2.1 with $c = a'$. Of course, such conversion is done only once at the beginning and once at the end of the complete cryptographic computing so that all the computations can be done in Montgomery representations. Hence we ignore the cost of the conversion from Montgomery to classic representation (and reciprocally) in the following. We note, as $r < 2p$, that a comparison and a final subtraction could occur, but the output of Algorithm 2.1 can be used as input by adding a condition on p, specifically $4p < \mathbf{B}^n$.

Reduction using special modulo

Usually, when using ECC, one can choose the underlying field without restrictions. In this case, the cost of a modular reduction can be reduced to several shifts and additions. This is why the generalized Mersenne number class was introduced [22, 57]. This is used in most of the standards. However, this approach has several drawbacks:

- It requires a dedicated architecture to such a particular p which cannot be used for other prime fields. Consequently, it is not practical in either software or hardware implementation and many customers prefer flexible products.
- In pairing based cryptosystems based on ordinary curves, the underlying fields cannot be chosen because curves are built via complex multiplication methods. Moreover, it has been shown in [52] that the use of such special moduli can introduce weakness in pairing based cryptosystems.

For these reasons we do not consider this approach in this paper.

2.2 The Residue Number Systems (RNS)

The Residue Number Systems are a corollary of the Chinese Remainder Theorem (CRT). They are based on the fact that a number a can be represented by its residues $(a_1, a_2, \ldots, a_n)$ modulo a set of coprime numbers $(m_1, m_2, \ldots, m_n)$, called RNS basis, thus $a_i = a \bmod m_i$. We generally assume that $0 \leq a < M = \prod_{i=1}^{n} m_i$. The elements a_i are called *RNS-digits*. The strongest advantage of such a system is that it distributes large integer operations on the small residue values.

The operations are performed independently on the residues. In particular, there is no carry propagation. These systems were introduced and developed in [32,58,59]. A good introduction can be found in [41].

For constructing an arithmetic over $\mathbb{F}_p$, we assume that $M = \prod_{i=1}^{n} m_i$ is such that $p < M$. In this system, two numbers a and b can be represented by their remainders modulo the m_i, $i = 1, \ldots, n$:

$$a = (a_1, \ldots, a_n) \quad \text{and} \quad b = (b_1, \ldots, b_n).$$

A multiplication modulo M is reduced to n independent RNS-digits products. An RNS-digits product is equivalent to a classical digit product followed by a modular reduction modulo m_i, which represents few additions (see [8,9]),

$$r = (a_1 \times b_1 \bmod m_1, \ldots, a_n \times b_n \bmod m_n) \equiv a \times b \pmod{M}. \tag{2.1}$$

In this paper, we consider an RNS basis $(m_1, \ldots, m_n)$ with elements such that $m_i = 2^h - c_i$, where c_i is small and sparse, $c_i < 2^{h/2}$. The reduction modulo m_i is, in this case, obtained with few shift and additions as in Section 2.1. As explained in [5, 8, 9, 20] this property ensures that an RNS-digits product can be considered to be equivalent to 1.1 word-product (word $= h$-bits). Of course, this assumption is depending on the platform in practice.

RNS Montgomery reduction

We now focus on the multiplication modulo p using the Montgomery algorithm presented in [3] and [4]. This is a direct transposition of the classical Montgomery method. The main difference is due to the representation system. When the Montgomery method is applied in a classical radix $\mathbf{B}$ number system, the value $\mathbf{B}^n$ occurs in the reduction, division and Montgomery factor. In RNS, this value is replaced by M. However, an auxiliary RNS basis is needed to handle the inverse of M. Hence some operations as the initial product must be performed on the two bases, which cost $2n$ words-products.

For two numbers a and b given in RNS, this algorithm evaluates $r = abM^{-1} \bmod p$ in RNS. As in the classical Montgomery method given in Section 2.1, this problem can be overcome by using Montgomery representation where $a' = a \times M \bmod p$, which is stable for the Montgomery product and addition. Of course, the conversion is done only once at the beginning by performing Montgomery product with a and $(M^2 \bmod p)$ as operands, and once at the end of the complete cryptographic computing with 1 as second operand. Hence this transformation will be neglected in the following. Moreover, as the RNS is not redundant, this representation is well suited for cryptography without any conversion [7].

Algorithm 2.2 presents the RNS Montgomery reduction (c can be considered as the result of an RNS product on the two bases), where all the operations considered are in RNS. We clarify on which basis they are done.

Algorithm 2.2. MontgR_RNS(c, p)

Require: Two RNS bases $\mathcal{B} = (m_1, \ldots, m_n)$, and $\mathcal{B}' = (m_{n+1}, \ldots, m_{2n})$, such that $M = \prod_{i=1}^{n} m_i < M' = \prod_{i=1}^{n} m_{n+i}$ and $\gcd(M, M') = 1$,

a prime number p such that $4p < M$ and $\gcd(p, M) = 1$; p is represented in basis $\mathcal{B}'$ and $-p^{-1}$ is precomputed in basis $\mathcal{B}$,

a positive integer c represented in RNS in both bases, with $c < Mp$.

Ensure: A positive integer $r \equiv cM^{-1} \pmod{p}$ represented in RNS in both bases, with $r < 2p$.

1: $q \leftarrow (c) \times (-p^{-1})$ in $\mathcal{B}$
2: $[q \text{ in } \mathcal{B}] \longrightarrow [q \text{ in } \mathcal{B}']$ *First base extension*
3: $r \leftarrow (c + q \times p) \times M^{-1}$ in $\mathcal{B}'$
4: $[r \text{ in } \mathcal{B}\,] \longleftarrow [r \text{ in } \mathcal{B}']$ *Second base extension*

Note that instructions 1 and 3 of Algorithm 2.2 above are RNS additions or multiplications which are performed independently for each element of the basis, so they are very efficient (linear). Instructions 2 and 4 represent RNS base extensions which are quadratic and then costly. To reduce this cost, we can use two different full RNS extensions as shown in [3, 4].

Finally, it is shown in [5] that the overall complexity of Algorithm 2.2 is $\frac{7}{5}n^2 + \frac{8}{5}n$ RNS-digits products.

If we operate with an architecture of n basic word-arithmetic cells, RNS arithmetic can be easily performed in a parallel manner due to the independence of the RNS-digits operations. A parallel evaluation of the multiplication (in both bases) requires only two steps whereas Algorithm 2.2 can be done in $\frac{12}{5}n + \frac{3}{5}$ steps [5].

Advantages of the RNS

Even though the number of operations needed for the reduction is somewhat higher than in a classical representation ($n^2 + n$ words products for the classical Montgomery reduction), RNS has some important advantages.

- Assuming that for ECC size the multiplication needs n^2 word-products, the RNS approach is asymptotically quite interesting for a modular multiplication which represents $2n^2 + n$ word-products in classical systems and $(\frac{7}{5}n^2 + \frac{18}{5}n) \times 1.1$ in RNS.

- As shown in [34], RNS is easy to implement, particularly in hardware, and it provides a reduced cost for multiplication and addition and a competitive modular reduction. Furthermore, due to the independence of the modular operations, computations can be performed in a random way and the architecture can be parallelized.
- An RNS based architecture is flexible: with a given structure of n modular digit operators, it is possible to handle any values of p such that $4p < M$. Hence the same architecture can be used for different levels of security and several base fields for each of these levels.
- There is a large gap between the cost of the reduction and the cost of the multiplication ($\frac{7}{5}n^2$ vs. $2n$) which is much smaller in classical systems ($n^2 + n$ vs. n^2). We can take great advantage of this gap by accumulating multiplications before reduction. This method is called *lazy reduction*.

2.3 Lazy reduction

Lazy reduction is often used in optimized implementations [44, 54, 61] and, independently of this work, in the context of pairing implementations [2]. It consists in delaying the reduction step after computing several products which must be summed. For example, assume that we want to compute $ab + cd$ with a, b, c and d in $\mathbb{F}_p$, where p is an n-word prime number. A classical implementation involves 2 modular multiplications and then requires $4n^2 + 2n$ word-products. In a lazy reduction implementation, we first compute the 2 multiplications and add them before a unique reduction step. Thus it requires only $3n^2 + n$ word-products. Of course, this implies that the reduction algorithm can take larger integers as input (less than $2p^2$ instead of less than p^2 in the above example). It means that $\mathbf{B}^n$ or M must be larger than p. This is not really cumbersome since there are several means to do this:

- If, as often in cryptography, the size of p is an exact multiple of the word size of the architecture, we need an additional word to handle $\mathbf{B}^n$ or M. Of course, this method becomes costly if n is small as for example on 64-bit architecture.
- Use a prime p whose size is a little bit smaller than an exact multiple of the word size of the architecture. Then $\mathbf{B}^n$ or M can be chosen larger than p. The (psychological) drawback is that we obtain security levels which are not standard. For example, a 254-bit prime p is used in [2, 15] ensuring 127 bits of security.
- If larger words are used, like 36 bits words on FPGA, there are also sufficiently extra bits to handle $\mathbf{B}^n$ or M for cryptographic applications [34].

This method is particularly interesting if an RNS arithmetic is used because of the large gap of complexity between the multiplication and the reduction step. As an example: while the classical computation of $ab + cd$ requires $\frac{14}{5}n^2 + \frac{36}{5}n$ RNS-digits products, the use of lazy reduction requires only $\frac{7}{5}n^2 + \frac{28}{5}n$ RNS-digits products. Hence lazy reduction is particularly well adapted to RNS arithmetic. This has already been used in [5,6] for elliptic curve cryptography. The goal of this paper is to use it for efficient arithmetic on extension fields and consequently for pairing based cryptography.

3 Fast arithmetic in $\mathbb{F}_{p^k}$ combining lazy reduction and RNS

3.1 Polynomial reduction

Efficient arithmetic in finite extensions of prime fields are usually done with sparse polynomials with small coefficients so that the cost of the reduction modulo this polynomial is given by some additions. More precisely, we give this cost when the extension is defined by a trinomial or a binomial, which is almost always the case in practice.

Proposition 3.1. *Let p be a prime number and $X^k - \delta X^d - \varepsilon$ be in $\mathbb{F}_p[X]$ such that $d \leq k/2$ and δ, ε small (by 'small' we mean that the multiplication by such a number is cheap in $\mathbb{F}_p$). The reduction modulo $X^k - \delta X^d - \varepsilon$ of a degree $2k-1$ polynomial requires only few additions in $\mathbb{F}_p[X]$.*

Proof. Write a degree $2k-1$ polynomial P as $X^{2k-d}P_1 + X^k P_2 + P_3$, where P_1, P_2 and P_3 are polynomials of degree respectively at most $d-1$, $k-d-1$ and $k-1$. We have

$$\begin{aligned}
P &= (\delta X^d + \varepsilon)X^{k-d}P_1 + (\delta X^d + \varepsilon)P_2 + P_3 && \bmod X^k - \delta X^d - \varepsilon \\
&= \delta X^k P_1 + \varepsilon X^{k-d} P_1 + \delta X^d P_2 + \varepsilon P_2 + P_3 && \bmod X^k - \delta X^d - \varepsilon \\
&= \delta(\delta X^d + \varepsilon) P_1 + \varepsilon X^{k-d} P_1 + \delta X^d P_2 + \varepsilon P_2 + P_3 && \bmod X^k - \delta X^d - \varepsilon \\
&= \delta^2 X^d P_1 + \delta\varepsilon P_1 + \varepsilon X^{k-d} P_1 + \delta X^d P_2 + \varepsilon P_2 + P_3 && \bmod X^k - \delta X^d - \varepsilon \\
&= \delta X^d(\delta P_1 + P_2) + \varepsilon(\delta P_1 + P_2 + X^{k-d} P_1) + P_3 && \bmod X^k - \delta X^d - \varepsilon.
\end{aligned}$$

It is easy to verify that all polynomials involved in this sum have degree less than or equal to $k-1$, so that this last expression is the reduced form of P. This sum requires only

- 4 additions in $\mathbb{F}_p[X]$,
- 2 multiplications by monomials in $\mathbb{F}_p[X]$, which are nothing but shifts of the coefficients and then are usually almost free,

- 3 multiplications by δ or ε, which can be counted as few additions in $\mathbb{F}_p[X]$ since δ and ε are assumed to be small.

Note that most of these cheap operations are even non-existent if the extension is defined by a binomial. □

This means that, if the irreducible polynomial defining $\mathbb{F}_{p^k}$ is well chosen, the cost of the reduction step in $\mathbb{F}_{p^k}$ arithmetic is negligible compared to a multiplication in $\mathbb{F}_p[X]$. In all the cases we are interested in this paper, and more generally in cryptography, such cheap reduction always holds. Hence we will focus in this paper on multiplication in $\mathbb{F}_p[X]$.

3.2 Multiplication in $\mathbb{F}_{p^k}$

These multiplications can be done using schoolbook method or using alternative well-known methods like Karatsuba or Toom–Cook. In this paper, we are interested in small values of k so that methods based on FFT are not interesting. Several people have already studied in detail which method must be used for each value of k and each construction of the extension. For example, [25] is very complete for extensions used in pairing-based cryptography. We will not recall these results here but use them for our comparisons. Anyway, whatever the method used, it requires k^λ multiplications in $\mathbb{F}_p$, with $1 < \lambda \leq 2$. The use of lazy reduction in this case is immediate. We just have to delay the reduction steps at the end of the computation. Then, only k reductions in $\mathbb{F}_p$ are required.

Example with $k = 2$

Assume $p \equiv 3$ modulo 4 so that -1 is not a square in $\mathbb{F}_p$. Then $\mathbb{F}_{p^2}$ can be defined by $\mathbb{F}_p[X]/(X^2+1)$. We want to compute the product of $P = a_0 + a_1 X$ and $Q = b_0 + b_1 X$. Using schoolbook multiplication, we have

$$PQ = a_0 b_0 - a_1 b_1 + (a_0 b_1 + a_1 b_0) X.$$

This is the typical case where lazy reduction is interesting since $ab + cd$ patterns occur. Finally, such a multiplication in $\mathbb{F}_{p^2}$ involves 4 multiplications in $\mathbb{F}_p$ but only 2 modular reductions. Note that, as elements in $\mathbb{F}_{p^2}$ have 2 independent components, it is not possible to have less than 2 reductions in $\mathbb{F}_p$ in the general case. Thus, using Karatsuba multiplication allows us to perform only 3 multiplications in $\mathbb{F}_p$ but always 2 reductions thanks to the formula

$$PQ = a_0 b_0 - a_1 b_1 + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1) X.$$

The interest of using the RNS for the arithmetic in $\mathbb{F}_{p^k}$ then becomes evident. Indeed, the expensive step of the RNS, namely the reduction step, is used linearly when $\mathbb{F}_{p^k}$ arithmetic is performed whereas the cheaper step, namely the multiplication step, is used quadratically or sub-quadratically in k. More precisely, we have the following property:

Proposition 3.2. *Let p be a prime number which can be represented by n words in radix representation and n RNS-digits in RNS representation. Let $\mathbb{F}_{p^k}$ be a finite extension of $\mathbb{F}_p$ defined by a sparse polynomial with small coefficients. We assume that the multiplication in $\mathbb{F}_p[X]$ requires k^λ multiplications in $\mathbb{F}_p$, with $1 < \lambda \leq 2$, and that we use lazy reduction in $\mathbb{F}_p$. A multiplication in $\mathbb{F}_{p^k}$ then requires*

- $(k^\lambda + k)n^2 + k\,n$ *word multiplications in radix representation,*
- $1.1\times\left(\frac{7k}{5}n^2 + \frac{10k^\lambda+8k}{5}n\right)$ *word multiplications if RNS is used.*

Proof. A complete multiplication in $\mathbb{F}_{p^k}$ requires k^λ multiplications in $\mathbb{F}_p$ thanks to Karatsuba-like methods and k reductions in $\mathbb{F}_p$ thanks to the use of the lazy reduction method. We have seen in Section 2.1 that a multiplication in $\mathbb{F}_p$ requires n^2 word multiplications and that a reduction requires $n^2 + n$ of them. This trivially gives the first assertion. The second one is obtained thanks to the cost of RNS multiplication ($2n$ RNS-digits products) and reduction ($\frac{7}{5}n^2 + \frac{8}{5}n$ RNS-digits products) given in Section 2.2, which must be multiplied by 1.1 to have an equivalent in word multiplications. □

Most of the gain is due to the accumulation of many products before reducing and not only 2 as in [5,6]. Of course, both the classical and the RNS reduction algorithms must be adapted. Indeed, input data can have a large size compared to p because of this accumulation process. More precisely, input data have maximal size $k'p^2$, where k' has the same size than k (it is not equal to k only because of the polynomial reduction step). Then it is sufficient to choose the radix such that $\mathbf{B}^n > k'p$ (or the RNS basis such that $M > k'p$). Moreover, if we want to use the output of the reduction algorithm (which is in $[0, 2p[$) as an input without a final comparison and subtraction, each product becomes less than $4p$ so that we have to choose $\mathbf{B}^n > 4k'p$ (or $M > 4k'p$). This is not restrictive in practice as long as k is not too large as explained in Section 2.3.

For values of k and n greater than or equal to 6, the gain is spectacular. For instance, if $n = k = 6$ and $\lambda = 1.5$ (which is a mean between Karatsuba and Toom–Cook complexities), a multiplication in $\mathbb{F}_{p^6}$ requires 781 word multiplications in radix representation while it requires only 590 in RNS. Of course, this is just a rough estimation to give an idea of the expected gain. Each particular situation must be studied in detail. In this paper, we did it for the extension degrees 6

and 12 for two reasons:

- they are of particular interest in pairing-based cryptography as we will see at the end of this paper,
- the classical arithmetic on such extensions is well studied ([25, 26]) which facilitates comparisons.

Some other extension degrees, like 2, 8 or 10, also have an interest in pairing-based cryptography but do not involve new materials relatively to 6 and 12. They can be done by the interested reader or can be demanded from the author. Note that, by increasing k, RNS can provide more benefits compared to radix representation because there is no $k^\lambda n^2$ term in the Proposition 3.2.

3.3 Lazy arithmetic in $\mathbb{F}_{p^6}$

In this section, we recall the different ways to perform efficient arithmetic in $\mathbb{F}_{p^6}$ and combine them with lazy reduction and of course RNS arithmetic. In fact, we are especially interested in multiplication in $\mathbb{F}_{p^6}$. There are three different ways to build $\mathbb{F}_{p^6}$, namely as a quadratic extension of a cubic one, as a cubic extension of a quadratic one or directly as a sextic extension (i.e., with an irreducible polynomial of degree 6). For each of these constructions Devegilli et al. [25] studied in detail all the possible arithmetic (schoolbook, Karatsuba, Toom–Cook and Chung–Hasan for squaring). Their conclusions are:

- The Toom–Cook method requires asymptotically less multiplications or squarings in $\mathbb{F}_p$ but is inefficient in practice for cryptographic sizes because it requires many additions in $\mathbb{F}_p$.
- The most efficient implementation for multiplication and squaring is obtained when $\mathbb{F}_{p^6}$ is built as a quadratic extension of a cubic one. This is due to the fact that there are very efficient formulas for quadratic and cubic extensions, but not for higher degrees.

We then assume in the following that $\mathbb{F}_{p^6}$ is built as a quadratic extension of a cubic one. Of course, those extensions are built as in Proposition 3.2 so that the reduction step in $\mathbb{F}_{p^6}$ is negligible compared to the multiplication step. In this case, according to [25], the most efficient way to perform a multiplication is to use the Karatsuba method for both the multiplication in the quadratic extension and in the cubic extension. The cost of a multiplication in $\mathbb{F}_{p^6}$ is then 18 multiplications in $\mathbb{F}_p$. Concerning squaring, there are several methods more or less equivalent in [25]. The choice is depending on the cost of the squaring in $\mathbb{F}_p$ compared to a multiplication. In this paper, we assume that these costs are the same so that the best squaring in

$\mathbb{F}_{p^6}$ requires 12 multiplications in $\mathbb{F}_p$ using Karatsuba squaring for the cubic extension and the complex method for the quadratic one.

Performing lazy reduction is immediate assuming that it is possible to reduce sums of several products. To ensure this, we have to relax the condition on $\mathbf{B}^n > 4p$ (for Montgomery reduction) or $M > 4p$ (for RNS arithmetic). However, this is not restrictive in practice as explained in Section 2.3.

Finally, the cost of a multiplication in $\mathbb{F}_{p^6}$ is 18 multiplication in $\mathbb{F}_p$ and 6 modular reductions whereas a squaring requires only 12 multiplications but also 6 reductions.

3.4 Example of degree 6 extension in 192 bits

In this section, we give an explicit example of degree 6 extension of a 192-bit prime field. This example comes from [50] and is linked to an MNT curve suitable for pairing based cryptography, which is the subject of the next section. Let $\mathbb{F}_p$ be defined by the prime number

$$p = 4691249309589066676602717919800805068538803592363589996389.$$

In this case, $\mathbb{F}_{p^6}$ can be defined by a quadratic extension of a cubic one thanks to the polynomials $X^3 - 2$ and $Y^2 - \boldsymbol{\alpha}$, where $\boldsymbol{\alpha}$ is a cubic root of 2,

$$\mathbb{F}_{p^3} = \mathbb{F}_p[X]/(X^3 - 2) = \mathbb{F}_p[\boldsymbol{\alpha}] \quad \text{and}$$
$$\mathbb{F}_{p^6} = \mathbb{F}_{p^3}[Y]/(Y^2 - \boldsymbol{\alpha}) = \mathbb{F}_{p^3}[\boldsymbol{\beta}].$$

As we want to use lazy reduction, the arithmetic of this extension must be completely unrolled. Hence let

$$A = a_0 + a_1\boldsymbol{\alpha} + a_2\boldsymbol{\alpha}^2 + \left(a_3 + a_4\boldsymbol{\alpha} + a_5\boldsymbol{\alpha}^2\right)\boldsymbol{\beta} \quad \text{and}$$
$$B = b_0 + b_1\boldsymbol{\alpha} + b_2\boldsymbol{\alpha}^2 + \left(b_3 + b_4\boldsymbol{\alpha} + b_5\boldsymbol{\alpha}^2\right)\boldsymbol{\beta}$$

be two elements of $\mathbb{F}_{p^6}$. Using Karatsuba on the quadratic extension leads to

$$\begin{aligned} AB = {} & \left(a_0 + a_1\boldsymbol{\alpha} + a_2\boldsymbol{\alpha}^2\right)\left(b_0 + b_1\boldsymbol{\alpha} + b_2\boldsymbol{\alpha}^2\right) \\ & + \boldsymbol{\alpha}\left(a_3 + a_4\boldsymbol{\alpha} + a_5\boldsymbol{\alpha}^2\right)\left(b_3 + b_4\boldsymbol{\alpha} + b_5\boldsymbol{\alpha}^2\right) \\ & + \Big[\left(a_0 + a_3 + (a_1 + a_4)\boldsymbol{\alpha} + (a_2 + a_5)\boldsymbol{\alpha}^2\right) \\ & \qquad \times \left(b_0 + b_3 + (b_1 + b_4)\boldsymbol{\alpha} + (b_2 + b_5)\boldsymbol{\alpha}^2\right) \\ & \quad - \left(a_0 + a_1\boldsymbol{\alpha} + a_2\boldsymbol{\alpha}^2\right)\left(b_0 + b_1\boldsymbol{\alpha} + b_2\boldsymbol{\alpha}^2\right) \\ & \quad - \left(a_3 + a_4\boldsymbol{\alpha} + a_5\boldsymbol{\alpha}^2\right)\left(b_3 + b_4\boldsymbol{\alpha} + b_5\boldsymbol{\alpha}^2\right)\Big]\boldsymbol{\beta}. \end{aligned}$$

Using Karatsuba again to compute each of these 3 products leads to

$$\begin{aligned}
AB = {} & a_0b_0 + 2\big(a_4b_4 + (a_1+a_2)(b_1+b_2) - a_1b_1 \\
& \quad + (a_3+a_5)(b_3+b_5) - a_3b_3 - a_5b_5\big) \\
& + \big[a_3b_3 + (a_0+a_1)(b_0+b_1) - a_0b_0 - a_1b_1 \\
& \quad + 2(a_2b_2 + (a_4+a_5)(b_4+b_5) - a_4b_4 - a_5b_5)\big]\boldsymbol{\alpha} \\
& + \big[a_1b_1 + 2a_5b_5 + (a_0+a_2)(b_0+b_2) - a_0b_0 - a_2b_2 \\
& \quad + (a_3+a_4)(b_3+b_4) - a_3b_3 - a_4b_4\big]\boldsymbol{\alpha}^2 \\
& + \big[(a_0+a_3)(b_0+b_3) - a_0b_0 - a_3b_3 \\
& \quad + 2\big((a_1+a_2+a_4+a_5)(b_1+b_2+b_4+b_5) \\
& \qquad - (a_1+a_4)(b_1+b_4) - (a_2+a_5)(b_2+b_5) \\
& \qquad - (a_1+a_2)(b_1+b_2) + a_1b_1 + a_2b_2 \\
& \qquad - (a_4+a_5)(b_4+b_5) + a_4b_4 + a_5b_5\big)\big]\boldsymbol{\beta} \\
& + \big[(a_0+a_1+a_3+a_4)(b_0+b_1+b_3+b_4) - (a_0+a_3)(b_0+b_3) \\
& \quad - (a_1+a_4)(b_1+b_4) - (a_0+a_1)(b_0+b_1) + a_0b_0 + a_1b_1 \\
& \quad - (a_3+a_4)(b_3+b_4) + a_3b_3 + a_4b_4 \\
& \quad + 2\big((a_2+a_5)(b_2+b_5) - a_2b_2 - a_5b_5\big)\big]\boldsymbol{\alpha\beta} \\
& + \big[(a_1+a_4)(b_1+b_4) - a_1b_1 - a_4b_4 \\
& \quad + (a_0+a_2+a_3+a_5)(b_0+b_2+b_3+b_5) \\
& \quad - (a_0+a_3)(b_0+b_3) - (a_2+a_5)(b_2+b_5) \\
& \quad - (a_0+a_2)(b_0+b_2) + a_0b_0 + a_2b_2 \\
& \quad - (a_3+a_5)(b_3+b_5) + a_3b_3 + a_5b_5\big]\boldsymbol{\alpha}^2\boldsymbol{\beta}.
\end{aligned}$$

It is easy to verify that this formula requires 18 multiplications in $\mathbb{F}_p$. Of course, it also requires many additions, but this is due to the Karatsuba method, not to lazy reduction. As explained in Section 3.3, it requires only 6 reductions thanks to the accumulation of all the operations in each component. However, this accumulation implies that the input of the reduction step can be very large. More precisely, thanks to the existence of the schoolbook method for computing AB, we can easily prove that if the components of A and B (i.e., the a_i and the b_i) are between 0 and $2p$ (which is the case when Algorithm 2.1 or 2.2 is used for reduction) then each component of AB is between 0 and $44p^2$. This means that $\mathbf{B}^n$ in Montgomery representation and M in RNS representation must be greater than $44p$ to perform lazy reduction in this degree 6 field.

3.5 Lazy arithmetic in $\mathbb{F}_{p^{12}}$

The same work as in Section 3.3 can be done in the case of $\mathbb{F}_{p^{12}}$. This has already been done in the recent literature because of the success of Barreto–Naehrig curves. For example, Devegili, Scott and Dahab explain in [26] that $\mathbb{F}_{p^{12}}$ must be built as a tower of extensions: quadratic on top of a cubic on top of a quadratic. This is nothing but Section 3.3 applied to a quadratic extension of $\mathbb{F}_p$. Then, if the Karatsuba method is used for the multiplication in this quadratic extension, a multiplication in $\mathbb{F}_{p^{12}}$ requires 54 multiplications in $\mathbb{F}_p$ and 12 modular reductions and a squaring requires 36 multiplications in $\mathbb{F}_p$ and 12 modular reductions.

3.6 Other useful operations in $\mathbb{F}_{p^k}$

Three other operations in $\mathbb{F}_{p^k}$ are necessary for pairing computations, the inversion, the Frobenius action (i.e., powering to the p) and the squaring in cyclotomic subgroups. Of course, additions in $\mathbb{F}_{p^k}$ are also often used. It is not clear that they are always negligible especially for small values of n. It depends on the implementation context. However, they are usually cheap compared to multiplications and we chose not to take them into account in this work.

Inversion

Performing an inversion in $\mathbb{F}_{p^k}$ must be done very carefully because it is an expensive operation. The general idea is that the inverse of an element is the product of its conjugates divided by its norm. This allows to replace an inversion in $\mathbb{F}_{p^k}$ by an inversion in $\mathbb{F}_p$ and some multiplications. For example, in $\mathbb{F}_{p^2} = \mathbb{F}_p[X]/X^2 - \varepsilon$ we have

$$\frac{1}{a_0 + a_1\sqrt{\varepsilon}} = \frac{a_0 - a_1\sqrt{\varepsilon}}{a_0^2 - \varepsilon a_1^2}$$

and an inversion in $\mathbb{F}_{p^2}$ requires 1 inversion, 2 squarings, 2 multiplications and 3 reductions in $\mathbb{F}_p$. In the same way an inversion in $\mathbb{F}_{p^3}$ (defined by a cubic root) requires 1 inversion, 9 multiplications, 3 squarings and 7 reductions in $\mathbb{F}_p$ ([54]). We can easily deduce that an inversion in $\mathbb{F}_{p^6}$, built as a quadratic extension of a cubic one, requires 1 inversion, 36 multiplications and 16 reductions in $\mathbb{F}_p$. In the same way, if $\mathbb{F}_{p^{12}}$ is built as in Section 3.5, it is easy to show that an inversion requires 1 inversion, 97 multiplications ([35]) and 35 reductions in $\mathbb{F}_p$.

Frobenius action

Contrary to the inversion, the Frobenius action in $\mathbb{F}_{p^k}$ is cheap. Indeed, it is easy to prove that if $\{\xi_i\}_{i=0..k-1}$ (with $\xi_0 = 1$) is a basis for $\mathbb{F}_{p^k}$ as a $\mathbb{F}_p$ vector space,

we have

$$\left(\sum_{i=0}^{k-1} a_i \xi_i\right)^p = a_0 + \sum_{i=1}^{k-1} a_i \xi_i^p.$$

Thus, if the ξ_i^p are precomputed, this Frobenius operation requires only $k(k-1)$ multiplications in $\mathbb{F}_p$ and $k-1$ reductions. In fact, we can do even better with a good choice of the basis. For example, if the extension is defined by a root $\boldsymbol{\gamma}$ of the polynomial $X^k - \varepsilon$, then $\xi_i = \boldsymbol{\gamma}^i$ and for $1 \leq i \leq k-1$,

$$\xi_i^p = \boldsymbol{\gamma}^{ip} = c_i \boldsymbol{\gamma}^{r_i} = c_i \xi_{r_i}$$

with $c_i \in \mathbb{F}_p$ and $0 \leq r_i < k$. In this case, computing the Frobenius action requires only $k-1$ multiplications and $k-1$ reductions in $\mathbb{F}_p$. This is the case for the example given in Section 3.4, where additionally $p \equiv 1 \bmod 6$ so that $r_i = i$. Of course, the same holds also for raising up an element to any power of p.

Squaring in cyclotomic subgroups

Finally, we have already seen that squaring in a degree 6 extension of $\mathbb{F}_q$ usually requires 12 multiplications and 6 reductions in the base field $\mathbb{F}_q$. However, as noticed in [33], if the element to be squared lies in the cyclotomic subgroup

$$G_{\Phi_6}(\mathbb{F}_q) = \{A \in \mathbb{F}_{q^6} \mid A^{\Phi_6(q)} = 1\},$$

where Φ_6 denotes the 6-th cyclotomic polynomial, this can be done faster. More precisely, assuming the extension is defined by a binomial, such squarings require only 3 squarings in $\mathbb{F}_{q^2}$ (and always 6 reductions). For example, squaring an element of $G_{\Phi_6}(\mathbb{F}_p) \subset \mathbb{F}_{p^6}$ requires only 6 multiplications and squaring an element of $G_{\Phi_6}(\mathbb{F}_{p^2}) \subset \mathbb{F}_{p^{12}}$ requires only 18 multiplications.

4 Pairing on elliptic curves and their computation

4.1 Pairings in cryptography

Bilinear pairings on elliptic curves have been introduced in cryptography in the middle of the 90's for cryptanalysis. Indeed, they allow to transfer the discrete logarithm on an elliptic curve to a discrete logarithm in the multiplicative group of a finite field, where subexponential algorithms are available [29, 45]. In 2000, Joux introduced the first constructive use of pairings with a tripartite key exchange protocol [38]. Since then, it has been shown that pairings can be used to construct new protocols like identity based cryptography [17] or short signature [18]. As a consequence, pairings became very popular in asymmetric cryptography and computing

them as fast as possible is very important. Let us first briefly recall the state of the art in this field and then explain how an RNS arithmetic can be helpful.

4.2 The Tate pairing

The most popular pairing used in cryptography is the Tate pairing. We present it here in a simplified and reduced form because it is the one usually used in cryptographic applications. More details and generalities can be found in [16, 23]. In this paper we assume that E is an elliptic curve defined over $\mathbb{F}_p$ by an equation

$$y^2 = x^3 + a_4 x + a_6. \tag{4.1}$$

Let ℓ be a prime divisor of $\#E(\mathbb{F}_p) = p + 1 - t$, where t denotes the trace of the Frobenius map on the curve. The embedding degree k of E with respect to ℓ is the smallest integer such that ℓ divides $p^k - 1$. This means that the full ℓ-torsion of the curve is defined over the field $\mathbb{F}_{p^k}$. For any integer m and ℓ-torsion point P, if P_m is the point mP on E, $f_{(m,P)}$ is the function defined on the curve whose divisor is $\mathrm{div}(f_{(m,P)}) = m\,P - P_m - (m-1)\mathcal{O}$. The Tate pairing can then be defined by

$$\begin{aligned} e_T : E(\mathbb{F}_p)[\ell] \times E(\mathbb{F}_{p^k}) &\to \mathbb{F}_{p^k}^* / \left(\mathbb{F}_{p^k}^*\right)^{\ell}, \\ (P, Q) &\mapsto f_{(\ell,P)}(Q)^{\frac{p^k-1}{\ell}}. \end{aligned}$$

The first step to compute the Tate pairing is the computation of $f_{(\ell,P)}(Q)$. It is done thanks to an adaptation of classical scalar multiplication algorithm due to Miller [46], which is given here in a more general case to cover other pairings.

Algorithm 4.1. Miller(m, P, Q)

Require: An integer m with binary representation $(m_{s-1}, \ldots, m_0)_2$, two points P and Q in $E(\mathbb{F}_{p^k})$.

Ensure: $f_{(m,P)}(Q) \in \mathbb{F}_{p^k}$.

$T \leftarrow P$
$f \leftarrow 1$
for i from $s-2$ downto 0 **do**
  $f \leftarrow f^2 . \frac{g_{(T,T)}(Q)}{v_{2T}(Q)}$
  $T \leftarrow 2T$
  if $m_i = 1$ **then**
    $f \leftarrow f . \frac{g_{(T,P)}(Q)}{v_{T+P}(Q)}$
    $T \leftarrow T + P$
  end if
end for
return f

In this algorithm, $g_{(A,B)}$ is the equation of the line passing through the points A and B (or tangent to E in A if $A = B$) and v_A is the equation of the vertical line passing by A, so that $\frac{g_{(A,B)}}{v_{A+B}}$ is the function on E involved in the addition of A and B.

The second step is to raise f to the power $\frac{p^k-1}{\ell}$. There are several ways to speed up the pairing computation:

- simplifying and optimizing the operations inside the Miller loop ([10, 12, 13, 24, 53]),
- constructing pairing-friendly elliptic curves ([11, 14, 21, 27, 31, 39, 47, 51, 55]) and a good survey is [28],
- more recently, reducing the length of the Miller loop ([36, 37, 43, 60]) thanks to the introduction of new pairings,
- simplifying the final exponentiation ([30, 40, 56]).

Note that it is not interesting to use better exponentiation techniques as sliding windows for pairing computations. This is because even if $3P$, for instance, can be precomputed, the writing up of f requires an additional (expensive) multiplication in $\mathbb{F}_{p^k}$ by the function involved in the computation of $3P$.

4.3 Ordinary curves with prescribed embedding degrees

The embedding degree k is usually very large so that computing in $\mathbb{F}_{p^k}$ is not reasonable. This is reassuring regarding the destructive use of pairings but annoying if one wants to use pairing based cryptosystems. Curves with small embedding degrees can be obtained in two different ways. The first one is to use supersingular curves. However, this work focuses on large characteristic base fields and, in this case, the embedding degree is less than or equal to 2, which is too small for security reasons without working on artificially large base fields. The second one is to use ordinary curves with prescribed embedding degrees constructed via the complex multiplication method as surveyed in [28]. We will focus here on the most popular ones, namely the MNT curves having an embedding degree equal to 6 ([47]) and the BN curves having embedding degree equal to 12 ([14]).

MNT curves

In [47], the authors explain how to use the complex multiplication method to construct ordinary elliptic curves with embedding degrees 3, 4 and 6. Their goal was to characterize elliptic curves with small embedding degrees to protect against destructive use of pairings, but it has also been used as a mean to construct ordinary curves with embedding degree 6.

Theorem 4.2. *Let p be a large prime and E be an ordinary elliptic curve defined over $\mathbb{F}_p$ such that $\#E(\mathbb{F}_p) = p + 1 - t$ is prime. Then E has embedding degree 6 if and only if there exists $l \in \mathbb{Z}$ such that $p = 4l^2 + 1$ and $t = 1 \pm 2l$.*

The strategy to generate ordinary elliptic curve of prime order with embedding degree 6 is the following:

- Select a small discriminant D which is 3 mod 8 but not 5 mod 10 and such that -8 is a quadratic residue modulo $3D$.
- Compute solutions $(X = 6l \pm 1, Y)$ of the generalized Pell equation $X^2 - 3DY^2 = -8$ until the values of p and $\#E(\mathbb{F}_p)$ corresponding to this value of l are prime numbers of the desired size.
- Repeat with another D if not found.

The main drawback of this method is that the consecutive solutions of generalized Pell equations grow exponentially so that only very few curves are found. However, it is possible to relax the constraints in order to obtain more curves as done in [31, 55]. As an example, the following 192-bit curve has been found with this method ([50]).

Proposition 4.3. *Let p be the prime number given in Section* 3.4. *The curve defined over $\mathbb{F}_p$ by the equation*

$$y^2 = x^3 - 3x + 311201765051646778586510196202962102273165873896518652743 3$$

has embedding degree 6 *and cardinality* 2ℓ*, where ℓ is the prime number*

$$\ell = 23456246547945333383013589599423455729182157373985290948 37.$$

Barreto–Naehrig curves

Barreto and Naehrig devised in [14] a method to generate pairing friendly elliptic curves over a prime field, with prime order and embedding degree 12. The equation of the curve is

$$y^2 = x^3 + a_6, \quad a_6 \neq 0,$$

and the trace of the Frobenius t, the cardinality of the curve r and the base field $\mathbb{F}_p$ are parameterized as

- $t = 6l^2 + 1$,
- $r = 36l^4 - 36l^3 + 18l^2 - 6l + 1$,
- $p = 36l^4 - 36l^3 + 24l^2 - 6l + 1$.

For example, $a_6 = 3$ and $l = -6000000000001F2D$ (hex) are such that both r and p are prime numbers ([26]). Other examples with low Hamming weight can be found in [51].

4.4 Fast Tate pairing computation

In this section we explain how to efficiently compute the Tate pairing.

Formulas for Miller loop

Of course, the easiest way to speed up the pairing computation is to choose ℓ as sparse as possible. As this can be done in many cases, we give only formulas for the "doubling" step of the Miller loop, which are therefore representative of the whole Miller loop. Moreover, the formulas for the addition are included in the final Algorithms 5.1 and 5.2.

We assume in this section that Jacobian coordinates have been chosen because it is usually the case in pairing implementations (cf. [15,35,49]). Of course, other choices are possible depending on the implementation context. For example, projective coordinates require less multiplications (but many more additions) if a_6 is small ([2,24]). However, this has almost no consequences for the results obtained in this paper because its main topic is low-level arithmetic. Let E be an elliptic curve defined as in (4.1).

Let $P = (X_P, Y_P, Z_P)$ and $T = (X_T, Y_T, Z_T)$ be two points in $E(\mathbb{F}_{p^k})$ given in Jacobian coordinates and $Q = (x_Q, y_Q) \in E(\mathbb{F}_{p^k})$ given in affine coordinates. Formulas for the "doubling" step of the Miller loop are given by

$$\begin{aligned}
A &= 3X_T^2 + a_4 Z_T^4,\\
C &= 4X_T Y_T^2,\\
X_{2T} &= A^2 - 2C,\\
Y_{2T} &= A(C - X_{2T}) - 8Y_T^4,\\
Z_{2T} &= 2Y_T Z_T,\\
g_{T,T}(Q) &= \frac{2Y_T Z_T^3 y_Q - A(Z_T^2 x_Q - X_T) - 2Y_T^2}{2Y_T Z_T^3},\\
v_{T,T}(Q) &= \frac{x_Q Z_{2T}^2 - X_{2T}}{Z_{2T}^2}.
\end{aligned}$$

If the Tate pairing is used, then the point P (and consequently the point T) is in $E(\mathbb{F}_p)$ so that most of the terms in these formulas are in $\mathbb{F}_p$. The only elements

lying in the extension field $\mathbb{F}_{p^k}$ are x_Q and y_Q and we will now see that, thanks to the use of twists, we can almost assume that they are lying in a proper subfield of $\mathbb{F}_{p^k}$. This will have important consequences on the efficiency of the Miller loop.

Use of twists

Definition 4.4. Two elliptic curves E and $\tilde{E}$ defined over $\mathbb{F}_q$ are said to be *twisted* if there exists an isomorphism Ψ_d between E and $\tilde{E}$ defined over an extension $\mathbb{F}_q^d$ of $\mathbb{F}_q$. The *degree* of this twist is the degree of the smallest extension on which Ψ_d is defined.

The possible twist degrees are 2, 3, 4 and 6, depending on the embedding degree k. We will focus in this paper on even embedding degrees k and twists of degree 2 and 6. We have the following properties:

- Let ν be a non-quadratic residue in $\mathbb{F}_{p^{k/2}}$. The curves E and $\tilde{E}$ given by the equations

$$E : y^2 = x^3 + a_4x + a_6, \quad \tilde{E} : \nu y^2 = x^3 + a_4x + a_6$$

  are twisted by the twist of order 2 (i.e., defined over $\mathbb{F}_{p^k}$)

$$\Psi_2 : \tilde{E} \to E,$$
$$(x, y) \mapsto (x, y\nu^{\frac{1}{2}}).$$

- An elliptic curve E defined as in (4.1) has a degree 6 twist if and only if $a_4 = 0$. If ν is an element in $\mathbb{F}_{p^{k/6}}$ which is not a sixth power, then E is the twisted of the curve $\tilde{E}$ defined by

$$y^2 = x^3 + \frac{b}{\nu}$$

  by the twist of order 6

$$\Psi_6 : \tilde{E} \to E,$$
$$(x, y) \mapsto (x\nu^{\frac{1}{3}}, y\nu^{\frac{1}{2}}).$$

Let Ψ be such a twist. As E and $\tilde{E}$ are isomorphic over $\mathbb{F}_{p^k}$, we can define a variant of the Tate pairing without loss of generality as

$$e_T(P, Q) = f_{(\ell,P)}(\Psi(Q))^{\frac{p^k-1}{\ell}}.$$

This is nothing but the Tate pairing defined on $E(\mathbb{F}_p)[\ell]\times\tilde{E}(\mathbb{F}_{p^k})$. For the twists given above, this means that the coordinates of Q can be written as $(x_Q, y_Q\nu^{\frac{1}{2}})$

or $(x_Q \nu^{\frac{1}{3}}, y_Q \nu^{\frac{1}{2}})$, where x_Q and y_Q are defined over $\mathbb{F}_{p^{k/d}}$. There are three important consequences on the Miller loop.

- The computation of f involves only $\mathbb{F}_{p^{k/d}}$ arithmetic (but the result is still in $\mathbb{F}_{p^k}$).
- The vertical lines, and more generally all the factors of f lying in a proper subfield of $\mathbb{F}_{p^k}$ (as $\mathbb{F}_p$ or $\mathbb{F}_{p^{k/d}}$), are wiped out by the final exponentiation. Hence, in the doubling step for example, only the expression $2Y_T Z_T^3 y_Q - A(Z_T^2 x_Q - X_T) - 2Y_T^2$ has to be computed before the writing up of f. This is the famous denominator elimination introduced in [10].
- In the case of twists of order 6 for Tate pairings, this expression has the particular form
$$g_0 + g_2 \nu^{\frac{1}{2}} + g_3 \nu^{\frac{1}{3}},$$
where $g_0 \in \mathbb{F}_p$ and $g_1, g_2 \in \mathbb{F}_{p^2}$, which contains only 5 coefficients instead of 12. Hence the multiplication by such an element during the writing up of f is cheaper than a complete multiplication in $\mathbb{F}_{p^k}$. More precisely, the multiplication of an arbitrary element of $\mathbb{F}_{p^k}$ by g_0 requires 12 multiplications in $\mathbb{F}_p$ and the one by $g_2 \nu^{\frac{1}{2}} + g_3 \nu^{\frac{1}{3}}$ requires 27 multiplications in $\mathbb{F}_p$ using Karatsuba. Finally, this operation requires 39 multiplications in $\mathbb{F}_p$ instead of 54 for a complete Karatsuba multiplication in $\mathbb{F}_{p^k}$. Even if this expression has a different form for other pairings, it is still sparse and we can also perform this multiplication with only 39 multiplications in $\mathbb{F}_p$.

Final exponentiation

For the Tate pairing (but also for other pairings), the exponent of the final exponentiation is $\frac{p^k-1}{\ell}$, which has the size of p^{k-1} and the base field is $\mathbb{F}_{p^k}$. Hence, at first sight, the final exponentiation seems to be very expensive. Hopefully, Koblitz and Menezes show in [42] that this cost can be reduced thanks to the factorization

$$\frac{p^k - 1}{\ell} = \left(p^{k/2} - 1\right)\left(\frac{p^{k/2} + 1}{\Phi_k(p)}\right)\left(\frac{\Phi_k(p)}{\ell}\right),$$

where Φ_k is the k-th cyclotomic polynomial. The fact that $\Phi_k(p)$ divides $p^{k/2}+1$ (as polynomials in p) is a direct consequence of the definition of k as the smallest integer such that ℓ divides $p^k - 1$. Then, the final exponentiation can be split into three parts. The first two parts are very fast since they are obtained via cheap Frobenius computations.

The third part, usually called the hard part, is given by the exponent $\frac{\Phi_k(p)}{\ell}$. In the general case, the best way is to develop this exponent in base p so that, thanks to multi-exponentiation, its cost is the same as if an exponent of the size of p were

used. However, both in the case of MNT curves and BN curves, we can do better thanks to the parametrization of p and ℓ. For example, in the case of MNT curves, we have

$$\frac{p^6-1}{\ell} = (p^3-1)\left(\frac{p^3+1}{\Phi_6(p)}\right)\left(\frac{\Phi_6(p)}{\ell}\right)$$

with

$$\begin{aligned}\Phi_6(p) &= p^2 - p + 1,\\ p &= 4l^2 + 1,\\ \ell &= 4l^2 - 2l + 1,\end{aligned}$$

and an elementary calculation gives

$$\frac{p^6-1}{\ell} = (p^3-1)(p+1)(p+2l).$$

Then, the final exponentiation is obtained thanks to several easy Frobenius applications and an exponentiation with $2l$ as an exponent (whose size is the half of the size of p). Note that it also involves an inversion in $\mathbb{F}_{p^6}$.

Of course, it is better to choose l as sparse as possible, but MNT curves are rare so that it is not feasible in practice. Then, efficient exponentiation methods, as sliding window, must be used.

On the contrary, the parameter can be chosen sparse for BN curves and it is not so easy to reduce the size of the exponent. More precisely, the hard part of the final exponentiation for BN curves consist in raising f to the

$$p^3 + (6l^2+1)p^2 + (36l^3 - 18l^2 + 12l - 1)p + 36l^3 - 30l^2 + 18l - 2.$$

This can be computed using a multi-addition chain requiring 3 exponentiations to the l, 13 multiplications, 4 squarings, and 7 Frobenius actions in $\mathbb{F}_{p^{12}}$ ([56]). As $p \approx 36l^4$, the cost is around three-quarters of the cost of an exponentiation with an exponent having the same size as p.

Finally, the result of the first two steps of the final exponentiation has order $\Phi_k(p)$. Then, as noticed in [33], it is in $G_{\Phi_k}(\mathbb{F}_q)$ so squaring such an element (which is the most used operation in the hard part of the exponentiation) is less expensive than a classical squaring in $\mathbb{F}_{p^k}$ as explained in Section 3.6. More recently, Karabina proposed to compress the elements of $G_{\Phi_6}(\mathbb{F}_{p^2})$ before squaring ([40]). This method provides faster squaring steps in the final exponentiation, but it involves extra inversions so that its efficiency is depending on the platform. It is successfully used in software in [2] but is probably much less interesting in hardware implementations. Moreover, it will have negligible consequences on our result because it can be used both in RNS and in radix representation. So we did not use it in this work.

4.5 Other pairings

Recently, some variants of the Tate pairing appear in the literature. The main goal is to reduce the length of the Miller loop. The price to be paid is that the points P and Q are swapped. This means that the elliptic curve arithmetic will hold in $\mathbb{F}_{p^k}$ (or $\mathbb{F}_{p^{k/d}}$ for twisted versions) instead of $\mathbb{F}_p$ so that even if the number of steps in the Miller loop will decrease, the cost of each step will increase. We give here the Ate and R-Ate pairings.

The Ate pairing

This pairing was first introduced in [37]. It is defined by

$$e_A : E(\mathbb{F}_{p^k}) \cap \mathrm{Ker}(\pi - p) \times E(\mathbb{F}_p)[\ell] \to \mathbb{F}_{p^k}^* / \left(\mathbb{F}_{p^k}^*\right)^\ell,$$
$$(Q, P) \mapsto f_{(t-1,Q)}(P)^{\frac{p^k-1}{\ell}},$$

where π is the Frobenius map on the curve. This construction works because, since $Q \in \mathrm{Ker}(\pi - p)$ and $\ell \mid \#E(\mathbb{F}_p) = p + 1 - t$, we have $\pi(Q) = (t-1)Q$. Finally, because $t - 1 \approx \sqrt{\ell}$, the length of the Miller loop is divided by 2 compared to the Tate pairing. This pairing is optimal for MNT curves in the sense of [60].

The R-Ate pairing

This is a generalization of the Ate pairing introduced in [43]. We give here its expression only in the case of BN curves. If l is the parameter used to construct the BN curve and $b = 6l + 2$, the R-Ate pairing is defined by

$$e_R : E(\mathbb{F}_{p^k}) \cap \mathrm{Ker}(\pi - p) \times E(\mathbb{F}_p)[\ell] \to \mathbb{F}_{p^k}^* / \left(\mathbb{F}_{p^k}^*\right)^\ell$$

with

$$(Q, P) \mapsto \left(f_{(b,Q)}(P) \cdot \left(f_{(b,Q)}(P) \cdot g_{(bQ,Q)}(P)\right)^p \cdot g_{(\pi((b+1)Q),bQ)}(P)\right)^{\frac{p^k-1}{\ell}},$$

where $g_{(A,B)}$ is the equation of the line passing through A and B. In this case, $l \approx \sqrt[4]{\ell}$, so the length of the Miller loop is divided by 4 compared to the Tate pairing. This pairing is optimal for BN curves in the sense of [60]. Note that the so-called optimal Ate pairing used in recent implementations ([2,15]) is almost the same as the R-Ate pairing: it only requires one multiplication in $\mathbb{F}_{p^{12}}$ less. Hence the results we obtained in this work for the R-Ate pairing are also valid for the optimal Ate pairing.

5 RNS arithmetic for fast pairing computation

When the embedding degree is large, the RNS does not yield to any noteworthy improvement on the computation of T or $g(Q)$ but, thanks to its linear complexity regarding the extension degree, we obtain very important improvements on the writing up of f and the final exponentiation. As these are the most expensive steps of the Tate pairing computation, we can say that the RNS arithmetic is particularly well adapted to fast pairing computation. But let us see more in detail the expected gains for most popular pairing friendly curves (in large characteristic), say MNT curves and BN curves.

5.1 Using RNS for MNT curves

We assume in this section that E is an elliptic curve defined over $\mathbb{F}_p$ by an equation (4.1) and obtained as described in Section 4.3 so that $p = 4l^2 + 1$ for some integer l. We also assume that $\mathbb{F}_{p^6}$ is built as a quadratic extension of $\mathbb{F}_{p^3}$,

$$\mathbb{F}_{p^6} = \mathbb{F}_{p^3}[Y]/(Y^2 - \nu) = \mathbb{F}_{p^3}[\boldsymbol{\beta}].$$

Tate Pairing

As MNT curves have a twist of order 2, the input elements of the Tate pairing can be written in the form $P = (x_P, y_P) \in E(\mathbb{F}_p)[\ell]$ and $Q = (x_Q, y_Q\boldsymbol{\beta})$ with x_Q and $y_Q \in \mathbb{F}_{p^3}$. Applying the improvements explained in Section 4.4, the Tate pairing is then given by Algorithm 5.1.

Algorithm 5.1. Tate(P, Q)

Require: $p = 4l^2 + 1$ prime,
E a MNT elliptic curve defined over $\mathbb{F}_p$,
$\ell \mid \#E(\mathbb{F}_p)$ prime with binary representation $(\ell_{s-1}, \ldots, \ell_0)_2$,
$P = (x_P, y_P) \in E(\mathbb{F}_p)[\ell]$,
$Q = (x_Q, y_Q\boldsymbol{\beta})$ with x_Q and $y_Q \in \mathbb{F}_{p^3}$.

Ensure: $e_T(P, Q) \in \mathbb{F}_{p^6}$.

1: $T = (X_T, Y_T, Z_T) \leftarrow (x_P, y_P, 1)$
2: $f \leftarrow 1$
3: **for** i from $s - 2$ downto 0 **do**
4: $\quad A = 3X_T^2 + a_4 Z_T^4, \quad C = 4X_T Y_T^2$
5: $\quad X_{2T} = A^2 - 2C$
6: $\quad Y_{2T} = A(C - X_{2T}) - 8Y_T^4$
7: $\quad Z_{2T} = 2Y_T Z_T$

8: $g = Z_{2T} Z_T^2 y_Q \boldsymbol{\beta} - A Z_T^2 x_Q + A X_T - 2 Y_T^2$
9: $f \leftarrow f^2.g$
10: $T \leftarrow [X_{2T}, Y_{2T}, Z_{2T}]$
11: **if** $\ell_i = 1$ **then**
12: $E = x_P Z_T^2 - X_T, \quad F = y_P Z_T^3 - Y_T$
13: $X_{T+P} = F^2 - 2X_T E^2 - E^3$
14: $Y_{T+P} = F(X_T E^2 - X_{T+P}) - Y_T E^3$
15: $Z_{T+P} = Z_T E$
16: $g = Z_{T+P} y_Q \boldsymbol{\beta} - Z_{T+P} y_P - F(x_Q - x_P)$
17: $f \leftarrow f.g$
18: $T \leftarrow [X_{T+P}, Y_{T+P}, Z_{T+P}]$
19: **end if**
20: **end for**
21: $f \leftarrow f^{p^3-1}$
22: $f \leftarrow f^{p+1}$
23: $f \leftarrow f^p . f^{2l}$
24: **return** f

Let us now precisely analyze the complexity of each line of this algorithm. Note that we choose to not distinguish multiplication and squaring in $\mathbb{F}_p$ for simplicity because this has no notable consequence on our study.

- The lines 4 to 7 are the standard doubling of the point $T \in E(\mathbb{F}_p)$ in Jacobian coordinates and require 10 multiplications in $\mathbb{F}_p$ (if a_4 has no special form). Lazy reduction can be used when computing A and Y_{2T} so that 8 modular reductions are necessary.
- In the same way, the lines 12 to 15 are for the mixed addition of T and P and require 11 multiplications and 10 reductions in $\mathbb{F}_p$.
- As x_Q and y_Q are in $\mathbb{F}_{p^3}$, line 8 requires 9 multiplications in $\mathbb{F}_p$. Note that this is not a good idea to write $A(Z_T^2 x_Q + X_T)$ in line 8 because it requires 2 multiplications of an element of $\mathbb{F}_{p^3}$ by an element of $\mathbb{F}_p$ instead of one. However, we can use the lazy reduction technique on the constant term of this expression. Finally, 8 modular reductions are necessary.
- The situation is the same for line 16, which requires 7 multiplications and 6 modular reductions in $\mathbb{F}_p$.
- Line 9 involves both a multiplication and a squaring in $\mathbb{F}_{p^6}$. The cost of such operations is detailed in Section 3.3. We deduce that the total complexity for this line is 30 multiplications and 12 modular reductions in $\mathbb{F}_p$.
- The situation for line 17 is similar and leads to 18 multiplications and 6 reductions.

- Line 21 is computed as $\frac{f^{p^3}}{f}$. As $f \in \mathbb{F}_{p^6}$, the exponentiation by p^3 is nothing but the conjugation on $\mathbb{F}_{p^3}[\boldsymbol{\beta}]$ so it is for free. Finally, this step requires a multiplication and an inversion in $\mathbb{F}_{p^6}$. As recalled in Section 3.6, such an inversion can be done with only 1 inversion, 36 multiplications and 16 reductions in $\mathbb{F}_p$. Finally, this first step of the final exponentiation requires 54 multiplications, 22 modular reductions and one inversion in $\mathbb{F}_p$.
- Line 22 involves one multiplication in $\mathbb{F}_{p^6}$ and one application of the Frobenius map. We have seen in Section 3.6 that, if the polynomial defining $\mathbb{F}_{p^6}$ is well chosen (which is always the case in practice) the Frobenius map requires only 5 modular multiplications in $\mathbb{F}_p$. Hence this second step of the final exponentiation requires 23 multiplications and 11 reductions in $\mathbb{F}_p$.
- The hard part of the final exponentiation involves one Frobenius (i.e., 5 modular multiplications), one multiplication in $\mathbb{F}_{p^6}$ (18 multiplications and 6 reductions) and one exponentiation by $2l$. We have already seen that l cannot be chosen sparse for MNT curves, so that advanced exponentiation methods must be used. In line 21 and 22, f has been raised to the power $(p^6-1)/\Phi_6(p)$, so that it is in $G_{\Phi_6}(\mathbb{F}_p)$ and can be squared with only 6 multiplications and 6 reductions in $\mathbb{F}_p$ as explained in Section 3.6. Then, for each step of this exponentiation, the cost is 6 multiplications and 6 reductions and 18 additional multiplications and 6 reductions if a multiplication is required by the exponentiation algorithm.

It is now necessary to fix the security level to have an idea of the overall complexity of the Tate pairing. We choose a 96-bit security level which is quite reasonable for embedding degree 6. Hence ℓ has bit-length 192 which means that the lines 4 to 9 are done 191 times and the lines 12 to 17 around 96 times. Concerning the hard part of the final exponentiation, as $2l$ is 96 bits long, it is reasonable to use the sliding window method with a window size of 3 for computing f^{2l}. It requires 96 squarings in $\mathbb{F}_{p^6}$ and, on average, 24 multiplications in $\mathbb{F}_{p^6}$ (plus 3 for precomputations).

Then, as summarized in Table 1, the full Tate pairing computation requires 13977 multiplications but only 8242 reductions. For this level of security, 6 (32-bit) words are necessary so a radix implementation requires

$$13977\times 6^2 + 8242\times(6^2+6) = 849336$$

word multiplications whereas an RNS implementation requires

$$1.1\left(13977\times 2\times 6 + 8242\times\left(\frac{7}{5}6^2+\frac{8}{5}6\right)\right) = 728468$$

word multiplications. This represents a gain of 14.2%.

	multiplications	reductions
lines 4 to 7	10	8
line 8	9	8
line 9	30	12
doubling step	49	28
lines 12 to 15	11	10
line 16	7	6
line 17	18	6
addition step	36	22
Miller loop	12815	7460
line 21	54	22
line 22	23	11
line 23	1085	749
final exponentiation	1162	782
Tate pairing	13977	8242

Table 1. Number of $\mathbb{F}_p$ operations for the Tate pairing on MNT curves for 96 bits of security.

Ate pairing

The algorithm is very similar to Algorithm 5.1, but the arguments P and Q are swapped. This means that operations of the lines 4 to 7 and 12 to 15 are done in $\mathbb{F}_{p^3}$ so multiplications are 6 times more expensive and reductions only 3 times. It is easy to prove that, if the coordinates of T are $(X_T, Y_T\beta, Z_T)$, the lines 8 and 16 must be replaced by

$$8' \quad g = Z_{2T}Z_T^2 y_P \beta + A(X_T - Z_T^2 x_P) - 2vY_T^2$$
$$16' \quad g = -Z_{T+Q} y_Q \beta + Z_{T+Q} y_P - F(x_P - x_Q)$$

where Z_{2T}, Z_T^2, $A = 3X_T^2 - a_4 Z_T^4$, Y_T^2, Z_{T+P} and $F = y_Q Z_T^3 - Y_T$ were computed in $\mathbb{F}_{p^3}$ during the previous steps. The first requires 18 multiplications and 12 reductions in $\mathbb{F}_p$ whereas the second requires 15 multiplications and 6 reductions. All these costs are summarized in Table 2. Moreover, since $t-1$ has bit-length 96 and Hamming weight around 48, the doubling step must be done 95 times in the Miller loop and the addition step 47 times. Finally, the final exponentiation is the same as for the Tate pairing.

	multiplications	reductions
lines 4 to 7	60	24
line 8'	18	12
line 9	30	12
doubling step	108	48
lines 12 to 15	66	30
line 16'	15	6
line 17	18	6
addition step	99	42
Miller loop	14913	6534
final exponentiation	1162	782
Ate pairing	16075	7316

Table 2. Number of $\mathbb{F}_p$ operations for the Ate pairing on MNT curves for 96 bits of security.

Then, the full Ate pairing computation requires 16075 multiplications but only 7316 reductions. This yields to 885972 word multiplications in radix representation but only 695046 in RNS which represents a gain of 21.5%.

Note that the Ate pairing is not really interesting for MNT curves because most of the computations are done in $\mathbb{F}_{p^3}$. This is not the case for BN curves because the twist used has order 6 so that the arithmetic involved in Ate pairing will be in $\mathbb{F}_{p^2}$. Moreover, we can again half the exponent in this case thanks to the R-Ate pairing.

5.2 Using RNS for BN curves

BN curves have a twist of order 6 so that all the improvements given in Section 4.4 can be used. This is one of the reasons of the current success of BN curves.

We assume in this section that E is an elliptic curve defined over $\mathbb{F}_p$ by an equation of the form

$$y^2 = x^3 + a_6$$

and obtained as described in Section 4.3. We also assume that $\mathbb{F}_{p^{12}}$ is built as a quadratic extension of a cubic extension of $\mathbb{F}_{p^2}$, which is easily compatible with the use of twists of order 6. More precisely, letting ν be an element in $\mathbb{F}_{p^2}$ which is not a sixth power, we build

$$\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[Y]/(Y^3 - \nu) = \mathbb{F}_{p^2}[\boldsymbol{\beta}], \quad \mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[Z]/(Z^2 - \boldsymbol{\beta}) = \mathbb{F}_{p^6}[\boldsymbol{\gamma}].$$

Thus $\mathbb{F}_{p^{12}}$ can also be defined by $\mathbb{F}_{p^2}[\boldsymbol{\gamma}]$.

Tate pairing

Thanks to the twist defined by ν, the second input of the Tate pairing can be written as

$$Q = (x_Q \boldsymbol{\gamma}^2, y_Q \boldsymbol{\gamma}^3) \quad \text{with } x_Q \text{ and } y_Q \in \mathbb{F}_{p^2}.$$

Applying the improvements explained in Section 4.4, the Tate pairing is given by Algorithm 5.2.

Algorithm 5.2. Tate(P, Q)

Require: $p = 36l^4 - 36l^3 + 24l^2 - 6l + 1$ prime,
E a BN elliptic curve defined over $\mathbb{F}_p$,
$\ell \mid \#E(\mathbb{F}_p)$ prime with binary representation $(\ell_{s-1}, \ldots, \ell_0)_2$,
$P = (x_P, y_P) \in E(\mathbb{F}_p)[\ell]$, $Q = (x_Q \boldsymbol{\gamma}^2, y_Q \boldsymbol{\gamma}^3)$ with $x_Q, y_Q \in \mathbb{F}_{p^2}$.

Ensure: $e_T(P, Q) \in \mathbb{F}_{p^{12}}$.

1: $T = (X_T, Y_T, Z_T) \leftarrow (x_P, y_P, 1)$
2: $f \leftarrow 1$
3: **for** i from $s-2$ downto 0 **do**
4: $\quad A = 3X_T^2, \quad C = 4X_T Y_T^2$
5: $\quad X_{2T} = A^2 - 2C$
6: $\quad Y_{2T} = A(C - X_{2T}) - 8Y_T^4$
7: $\quad Z_{2T} = 2Y_T Z_T$
8: $\quad g = Z_{2T} Z_T^2 y_Q \boldsymbol{\gamma}^3 - A Z_T^2 x_Q \boldsymbol{\gamma}^2 + A X_T - 2Y_T^2$
9: $\quad f \leftarrow f^2.g$
10: $\quad T \leftarrow [X_{2T}, Y_{2T}, Z_{2T}]$
11: $\quad$ **if** $\ell_i = 1$ **then**
12: $\quad\quad E = x_P Z_T^2 - X_T, \quad F = y_P Z_T^3 - Y_T$
13: $\quad\quad X_{T+P} = F^2 - 2X_T E^2 - E^3$
14: $\quad\quad Y_{T+P} = F(X_T E^2 - X_{T+P}) - Y_T E^3$
15: $\quad\quad Z_{T+P} = Z_T E$
16: $\quad\quad g = Z_{T+P} y_Q \boldsymbol{\gamma}^3 - F x_Q \boldsymbol{\gamma}^2 - Z_{T+P} y_P + F x_P)$
17: $\quad\quad f \leftarrow f.g$
18: $\quad\quad T \leftarrow [X_{T+P}, Y_{T+P}, Z_{T+P}]$
19: $\quad$ **end if**
20: **end for**
21: $f \leftarrow f^{p^6-1}$
22: $f \leftarrow f^{p^2+1}$
23: $f \leftarrow f^{p^3+(6l^2+1)p^2+(36l^3-18l^2+12l-1)p+36l^3-30l^2+18l-2}$
24: **return** f

Let us now analyze the complexity of each line of this algorithm.

- As explained for MNT curves, the lines 4 to 7 require 7 multiplications and 6 modular reductions whereas the lines 12 to 15 require 11 multiplications and 10 modular reductions.
- As x_Q and y_Q are in $\mathbb{F}_{p^2}$, line 8 requires 8 modular multiplications in $\mathbb{F}_p$ and lazy reduction cannot be used.
- In the same way, line 16 requires 6 multiplications but only 5 modular reductions because lazy reduction is used on the constant term.
- Line 9 involves both a squaring and a multiplication in $\mathbb{F}_{p^{12}}$. As explained in Section 3.5, such a squaring involves 36 multiplications and 12 modular reductions. We have seen in Section 4.4 that, thanks to its special form, the multiplication by g requires only 39 multiplications and 12 reductions. We deduce that the total complexity for this line is 75 multiplications and 24 reductions in $\mathbb{F}_p$.
- The situation for line 17 is similar and leads to 39 multiplications and 12 reductions.
- Line 21 is computed as $\frac{f^{p^6}}{f}$. As computing f^{p^6} is for free (conjugation), this step requires a multiplication and an inversion in $\mathbb{F}_{p^{12}}$. We have seen in Section 3.6 that an inversion can be done with only 1 inversion, 97 multiplications and 35 reductions in $\mathbb{F}_p$. Finally, this first step of the final exponentiation requires 151 multiplications, 47 modular reductions and 1 inversion in $\mathbb{F}_p$.
- Line 22 involves one multiplication in $\mathbb{F}_{p^{12}}$ and one powering to p^2. We have seen in Section 3.6 that the Frobenius map (and its iterations) requires 11 modular multiplications in $\mathbb{F}_p$. Thus this step requires 65 multiplications and 23 reductions in $\mathbb{F}_p$.
- The hard part of the final exponentiation is given by line 23. It can be computed using a multi-addition chain requiring 3 exponentiations to the l, 13 multiplications, 4 squarings, and 7 Frobenius actions in $\mathbb{F}_{p^{12}}$ [56]. Contrary to MNT curves, l can be chosen sparse for BN curves, so classical square-and-multiply can be used. Moreover, f has been raised to the power $(p^{12}-1)/\Phi_6(p^2)$ in line 21 and 22, so it is in $G_{\Phi_6}(\mathbb{F}_{p^2})$ and can be squared with only 18 multiplications and 12 reductions in $\mathbb{F}_p$ as explained in Section 3.6. For the (few) steps of these exponentiations corresponding to non-zero bits of the exponent, 54 additional multiplications and 12 additional reductions are necessary.

Again, it is necessary to fix the security level to have an idea of the overall complexity of the Tate pairing. We choose a 128-bit security level for which BN curves are well suited. We also have to fix the Hamming weight of l (and consequently

the one of ℓ). We assume here that l has weight 11 and ℓ has weight 90 as in the example given in Section 4.3. This is only an example and smaller values of these weight can be used for an optimal implementation as in [2, 15, 51], but it has negligible effects on our results. This means that lines 4 to 9 are done 255 times and lines 12 to 17, 89 times. Finally, the hard part of the final exponentiation involves 3 exponentiation by l which has bit-length 63 and Hamming weight 11 so each of them requires 62 doubling in $G_{\Phi_6}(\mathbb{F}_{p^2})$ and 10 multiplications in $\mathbb{F}_{p^{12}}$. This account is summarized in Table 3.

	multiplications	reductions
lines 4 to 7	7	6
line 8	8	8
line 9	75	24
doubling step	90	38
lines 12 to 15	11	10
line 16	6	5
line 17	39	12
addition step	56	27
Miller loop	27934	12093
line 21	151	47
line 22	65	23
f^l	1656	660
line 23	5819	2261
final exponentiation	6035	2331
Tate pairing	33969	14424

Table 3. Number of $\mathbb{F}_p$ operations for the Tate pairing on BN curves for 128 bits of security.

Then, the full Tate pairing computation requires 33969 multiplications but only 14424 reductions. Note that we obtain the same number of multiplications for the Miller loop as in [35] (since we use the same optimizations), but the number of reductions is much smaller.

For this level of security, 8 (32-bit) words are necessary so a radix implementation requires

$$33969 \times 8^2 + 14424 \times (8^2 + 8) = 3212544$$

word multiplications whereas an RNS implementation requires

$$1.1\left(33969\times 2\times 8+14424\times\left(\frac{7}{5}8^2+\frac{8}{5}8\right)\right)=2222574$$

word multiplications. This represents a gain of 30.8%.

Ate pairing

The algorithm is very similar to Algorithm 5.2, but the arguments P and Q are swapped. This means that operations of the lines 4 to 7 are done in $\mathbb{F}_{p^2}$ so multiplications are 3 times more expensive and reductions only 2 times. It is easy to prove that, if the coordinates of T are $(X_T\boldsymbol{\gamma}^2, Y_T\boldsymbol{\gamma}^3, Z_T)$, the lines 8 and 16 must be replaced by

$$8'' \quad g = Z_{2T}Z_T^2 y_P - AZ_T^2 x_P\boldsymbol{\gamma} + (AX_T - 2Y_T^2)\boldsymbol{\gamma}^3$$

$$16'' \quad g = Z_{T+Q}y_P - Fx_P\boldsymbol{\gamma} + (Fx_Q - Z_{T+Q}y_Q)\boldsymbol{\gamma}^3$$

where Z_{2T}, $A = 3X_T^2$, Y_T^2, Z_{T+P} and $F = y_Q Z_T^3 - Y_T$ were computed during the previous steps. The first requires 15 multiplications and 12 reductions in $\mathbb{F}_p$ whereas the second requires 10 multiplications and 6 reductions. Moreover, the value obtained for g has only terms in $\boldsymbol{\gamma}, \boldsymbol{\gamma}^3$ and a constant term, so that a multiplication by g requires only 39 multiplications instead of 54 as explained in Section 4.4.

Finally, in our example, $t-1$ has bit-length 128 and Hamming weight 29 and the final exponentiation is the same as for the Tate pairing. The number of operations required for all the steps of the computation of the Ate pairing is summarized in Table 4.

Then, the full Ate pairing computation requires 21836 multiplications but only 9491 reductions. This yields to 2080856 word multiplications in radix representation but only 1453380 in RNS which represents a gain of 30.1%.

R-Ate pairing

In the R-Ate pairing, the operations in the final exponentiation and in the (shorter) Miller loop are the same, but an additional step is necessary at the end of the Miller loop. This step is the computation of

$$f.\left(f.g_{(T,Q)}(P)\right)^p.g_{(\pi(T+Q),T)}(P),$$

where $T = (6l+2)Q$ is computed during the Miller loop and π is the Frobenius map on the curve. This step involves the following operations.

- One step of addition as in the Miller loop (computation of $T+Q$ and $g_{(T,Q)}(P)$) which requires 40 multiplications and 26 reductions in $\mathbb{F}_p$.

	multiplications	reductions
lines 4 to 7	17	12
line 8”	15	12
line 9	75	24
doubling step	107	48
lines 12 to 15	30	20
line 16”	10	6
line 17	39	12
addition step	79	38
Miller loop	15801	7160
final exponentiation	6035	2331
Ate pairing	21836	9491

Table 4. Number of $\mathbb{F}_p$ operations for the Ate pairing on BN curves for 128 bits of security.

- One application of the Frobenius map on the curve. As $p \equiv 1 \bmod 6$ for BN curves, this operation requires only 2 multiplications in $\mathbb{F}_{p^2}$ by precomputed values.
- One non-mixed addition step (computation of $g_{(\pi(T+Q),T)}(P)$). It is easy to prove that it requires 60 multiplications and 40 reductions in $\mathbb{F}_p$.
- Two multiplication by the previous results, both requiring 39 multiplications and 12 reductions in $\mathbb{F}_p$.
- One Frobenius requiring 11 modular multiplications.
- One full multiplication in $\mathbb{F}_{p^{12}}$ requiring 54 multiplications and 12 reductions in $\mathbb{F}_p$.

Then, this step requires 249 multiplications and 117 reductions in $\mathbb{F}_p$. Finally, in our example, $6l + 2$ has bit-length 66 and Hamming weight 9 so the cost of the R-Ate pairing is given in Table 5.

Then, the full R-Ate pairing computation requires 15019 multiplications but only 6405 reductions. This yields to 1310528 word multiplications in radix representation but only 905552 in RNS which represents a gain of 30.9%. Note that using the optimal Ate pairing as in [2, 15, 35] is only saving some multiplications in the additional step (both in RNS and in radix representation), so that we obtain a similar result.

	multiplications	reductions
doubling step	107	48
addition step	79	38
Miller loop	7587	3424
additional step	249	117
final exponentiation	6035	2331
R-Ate pairing	13871	5872

Table 5. Number of $\mathbb{F}_p$ operations for the R-Ate pairing on BN curves for 128 bits of security.

6 Conclusion

In this work we explained why the RNS arithmetic is particularly well suited for computation in extension fields essentially thanks to the use of lazy reduction. As a consequence, it is interesting to use such an arithmetic for pairing computation in large characteristic especially in contexts where the other advantages of the RNS arithmetic can be exploited (hardware architecture, parallel implementation). More precisely, we proved that using RNS for MNT curves when 96 bits of security are required involves 14.2 to 21.5% less basic operations. This gain reach more than 30% for BN curves with 128 bits of security whatever the pairing used (Tate, Ate, R-Ate, optimal). This is of course only an estimate which is made under assumptions (an RNS-digits product is equivalent to 1.1 classical digit product, additions in $\mathbb{F}_p$ have not been taken into account, 32-bit architecture, no need of extra word to handle lazy reduction, ...). Hence the real gain will depend on the implementation platform, but it shows that RNS arithmetic is very promising for pairing implementations.

Most of the gains are coming from arithmetic in extension fields so that choosing other systems of coordinate (affine, projective, Edwards, ...), other pairings or other high-level improvements in pairing implementations will not change these results. Moreover, it is beyond doubt that better gain will occur in other situations in pairing-based cryptography for two main reasons.

- For larger security levels or for curves of non-prime order (i.e., with $\frac{\log(p)}{\log(\ell)} > 1$), the number of words necessary to represent elements in $\mathbb{F}_p$ will increase, which is favorable to RNS arithmetic.
- Larger embedding degrees involve arithmetic in larger extension field, which is linear in RNS but quadratic in radix representation. Thus RNS can benefit from larger embedding degrees in comparison with radix representation.

Finally, RNS arithmetic is very promising for efficient pairing implementation and it would be attractive to develop a dedicated architecture such as the one described in [34].

Bibliography

[1] *GNU – Multiple-Precision Library*, `http://gmplib.org/`.

[2] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys and J. López, Faster explicit formulas for computing pairings over ordinary curves, *Cryptology ePrint Archive*, Report 2010/526, 2010, `http://eprint.iacr.org/`.

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

[4] J-C. Bajard, L-S. Didier and P. Kornerup, Modular multiplication and base extensions in residue number systems, in: *15th IEEE Symposium on Computer Arithmetic*, pp. 59–65, IEEE Computer Society Press, Los Alamitos, CA, 2001.

[5] J-C. Bajard, S. Duquesne and M. Ercegovac, Combining leak–resistant arithmetic for elliptic curves defined over $\mathbb{F}_p$ and RNS representation, *Cryptology ePrint Archive*, Report 2010/311, 2010, `http://eprint.iacr.org/`.

[6] J-C. Bajard, S. Duquesne, M. Ercegovac and N. Meloni, Residue systems efficiency for modular products summation: Application to Elliptic Curves Cryptography, in: *Advanced Signal Processing Algorithms, Architectures, and Implementations XVI*, pp. 631304.1–631304.11, Proceedings of SPIE 6313, SPIE, Bellingham, WA, 2006.

[7] J-C. Bajard and L. Imbert, A full RNS implementation of RSA, *IEEE Transactions on Computers* **53** (2004), 769–774.

[8] J-C. Bajard, M. Kaihara and T. Plantard, Selected RNS bases for modular multiplication, in: *19th IEEE International Symposium on Computer Arithmetic*, pp. 25–32, IEEE Computer Society Press, Los Alamitos, CA, 2009.

[9] J-C. Bajard, N. Meloni and T. Plantard, Efficient RNS bases for Cryptography, in: *Proceedings of the 17th IMACS World Congress: Scientific Computation, Applied Mathematics and Simulation*, Paris, France, July 11–15, 2005.

[10] P. Barreto, H. Kim, B. Lynn and M. Scott, Efficient algorithms for pairing-based cryptosystems, in: *Advances in Cryptology – CRYPTO 2002*, pp. 354–369, Lecture Notes in Computer Science 2442, Springer-Verlag, Berlin, Heidelberg, 2002.

[11] P. Barreto, B. Lynn and M. Scott, Constructing elliptic curves with prescribed embedding degrees, in: *Security in Communication Networks – SCN 2002*, pp. 257–267, Lecture Notes in Computer Science 2576, Springer-Verlag, Berlin, Heidelberg, 2003.

[12] P. Barreto, B. Lynn and M. Scott, Efficient implementation of pairing-based cryptosystems, *Journal of Cryptology* **17** (2004), 321–334.

[13] P. Barreto, B. Lynn and M. Scott, On the selection of pairing-friendly groups, in: *Selected Areas in Cryptography – SAC 2003*, Lecture Notes in Computer Science 3006, pp. 17–25, Springer-Verlag, Berlin, Heidelberg, 2004.

[14] P. Barreto and M. Naehrig, Pairing-friendly elliptic curves of prime order, in: *Selected Areas in Cryptography – SAC 2005*, Lecture Notes in Computer Science 3897, pp. 319–331, Springer-Verlag, Berlin, Heidelberg, 2006.

[15] J. L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez and T. Teruya, High-speed software implementation of the optimal Ate pairing over Barreto–Naehrig curves, in: *Pairing-Based Cryptography – Pairing 2010*, pp. 21–39, Springer-Verlag, Berlin, Heidelberg, 2010.

[16] I. F. Blake, G. Seroussi and N. P. Smart, *Advances in Elliptic Curve Cryptography*, Cambridge University Press, Cambridge, 2005.

[17] D. Boneh and M. Franklin, Identity-based encryption from the Weil pairing, in: *Advances in Cryptology – CRYPTO 2001*, Lecture Notes in Computer Science 2139, pp. 213–229, Springer-Verlag, Berlin, Heidelberg, 2001.

[18] D. Boneh, B. Lynn and H. Shacham, Short signatures from the Weil pairing, *Journal of Cryptology* **17** (2004), 297–319.

[19] A. Bosselaers, R. Govaerts and J. Vandewalle, Comparison of three modular reduction functions, in: *Advances in Cryptology – CRYPTO 94*, Lecture Notes in Computer Science 773, pp. 175–186, Springer-Verlag, Berlin, Heidelberg, 1994.

[20] R. P. Brent and H. T. Kung, The area-time complexity of binary multiplication, *Journal of the Association for Computing Machinery* **28** (1981), 521–534.

[21] F. Brezing and A. Weng, Elliptic curves suitable for pairing based cryptography, *Designs, Codes and Cryptography* **37** (2005), 133–141.

[22] J. Chung and A. Hasan, More generalized mersenne numbers, in: *Selected Areas in Cryptography – SAC 2003*, Lecture Notes in Computer Science 3006, pp. 335–347, Springer-Verlag, Berlin, Heidelberg, 2004.

[23] H. Cohen and G. Frey, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Discrete Mathematics and Its Applications 31, Chapman & Hall/CRC, Boca Raton, FL, 2006.

[24] C. Costello, T. Lange and M. Naehrig, Faster pairing computations on curves with high-degree twists, in: *Public Key Cryptography – PKC 2010*, Lecture Notes in Computer Science 6056, pp. 224–242, Springer-Verlag, Berlin, Heidelberg, 2010.

[25] A. J. Devegili, C. Ó hÉigeartaigh, M. Scott and R. Dahab, Multiplication and squaring on pairing-friendly fields, *Cryptography ePrint Archive*, Report 2006/471, 2006, `http://eprint.iacr.org/`.

[26] A. J. Devegili, M. Scott and R. Dahab, Implementing cryptographic pairings over Barreto–Naehrig curves, in: *Pairing-Based Cryptography – Pairing 2007*, pp. 197–207, Springer-Verlag, Berlin, Heidelberg,2007.

[27] D. Freeman, Constructing pairing-friendly elliptic curves with embedding degree 10, in: *Algorithmic Number Theory Symposium – ANTS-VII*, pp. 452–465, Lecture Notes in Computer Science 4076, Springer-Verlag, Berlin, 2006.

[28] D. Freeman, M. Scott and E. Teske, A taxonomy of pairing-friendly elliptic curves, *Journal of Cryptology* **23** (2010), 224–280.

[29] G. Frey and H. G. Rück, A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves, *Mathematics of Computation* **62** (1994), 865–874.

[30] S. Galbraith and M. Scott, Exponentiation in pairing-friendly groups using homomorphisms, in: *Pairing-Based Cryptography – Pairing 2008*, pp. 211–224, Springer-Verlag, Berlin, Heidelberg, 2008.

[31] S. D. Galbraith, J. F. McKee and P. C. Valenca, Ordinary abelian varieties having small embedding degree, *Finite Fields and Their Applications* **13** (2007), 800–814.

[32] H. L. Garner, The residue number system, *IRE Transactions on Electronic Computers, EL* **8** (1959), 140–147.

[33] R. Granger and M. Scott, Faster squaring in the cyclotomic subgroup of sixth degree extensions, in: *Public Key Cryptography – PKC 2010*, pp. 209–223, Lecture Notes in Computer Science 6056, pp. 224–242, Springer-Verlag, Berlin, Heidelberg, 2010.

[34] N. Guillermin, A high speed coprocessor for elliptic curve scalar multiplication over $\mathbb{F}_p$, in: *Cryptographic Hardware and Embedded Systems – CHES 2010*, Lecture Notes in Computer Science 6225, pp. 48–64, Springer-Verlag, Berlin, Heidelberg, 2010.

[35] D. Hankerson, A. Menezes and M. Scott, Software implementation of pairings, in: *Identity-Based Cryptography*, pp. 188–206, Cryptology and Information Security Series 2, IOS Press, Amsterdam, 2009.

[36] F. Hess, Pairing lattices, in: *Pairing-Based Cryptography – Pairing 2008*, pp. 18–38, Springer-Verlag, Berlin, Heidelberg, 2008.

[37] F. Hess, N. P. Smart and F. Vercauteren, The Eta pairing revisited, *IEEE Transactions on Information Theory* **52** (2006), 4595–4602.

[38] A. Joux, A one round protocol for tripartite Diffie–Hellman, *Journal of Cryptology* **17** (2004), 263–276.

[39] E. Kachisa, E. Schaefer and M. Scott, Constructing Brezing–Weng pairing-friendly elliptic curves using elements in the cyclotomic field, in: *Pairing-Based Cryptography – Pairing 2008*, pp. 126–135, Springer-Verlag, Berlin, Heidelberg, 2008.

[40] K. Karabina, Squaring in cyclotomic subgroups, *Cryptology ePrint Archive*, Report 2010/542, 2010, `http://eprint.iacr.org/`.

[41] D. E. Knuth, *Seminumerical Algorithms. The Art of Computer Programming. Volume 2*, Addison-Wesley, Reading, Mass., 1981.

[42] N. Koblitz and A. Menezes, Pairing-based cryptography at high security levels, in: *Proceedings of Cryptography and Coding 2005*, pp. 13–36, Lecture Notes in Computer Science 3796, Springer-Verlag, Berlin, Heidelberg, 2005.

[43] E. Lee, H. S. Lee and C. M. Park, Efficient and generalized pairing computation on abelian varieties, *IEEE Transactions on Information Theory* **55** (2009), 1793–1803.

[44] C. H. Lim and H. S. Hwang, Fast implementation of elliptic curve arithmetic in GF(p^n), in: *Public Key Cryptography – PKC 2000*, pp. 405–421, Lecture Notes in Computer Science 1751, Springer-Verlag, Berlin, Heidelberg, 2000.

[45] A. Menezes, T. Okamoto and S. Vanstone, Reducing elliptic curve logarithms to logarithms in a finite field, *IEEE Transactions on Information Theory* **39** (1993), 1639–1646.

[46] V. S. Miller, The Weil pairing, and its efficient calculation, *Journal of Cryptology* **17** (2004), 235–261.

[47] A. Miyaji, M. Nakabayashi and S. Takano, New explicit conditions of elliptic curve traces for FR-reduction, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* **84** (2001), 1234–1243.

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

[49] M. Naehrig, R. Niederhagen and P. Schwabe, New software speed records for cryptographic pairings, in: *LATINCRYPT*, pp. 109–123, Lecture Notes in Computer Science 6212, Springer-Verlag, Berlin, Heidelberg, 2010.

[50] D. Page, N. P. Smart and F. Vercauteren, A comparison of MNT curves and supersingular curves, *Applicable Algebra in Engineering, Communication and Computing* **17** (2006), 379–392.

[51] G. Pereira, M. Simplício Jr., M. Naehrig and P. Barreto, A family of implementation-friendly BN elliptic curves, to appear in *Journal of Systems and Software* (2011), *Cryptology ePrint Archive*, Report 2010/429, 2010.

[52] O. Schirokauer, The number field sieve for integers of low weight, *Mathematics of Computation* **79** (2010), 583–602.

[53] M. Scott, Computing the Tate pairing, in: *Topics in Cryptology – CT-RSA 2005* pp. 293–304, Lecture Notes in Computer Science 3376, Springer-Verlag, Berlin, Heidelberg, 2005.

[54] M. Scott, Implementing cryptographic pairings, in: *Pairing-based cryptography – Pairing 2007*, pp. 177–196, Lecture Notes in Computer Science 4575, Springer-Verlag, Berlin, Heidelberg, 2007.

[55] M. Scott and P. Barreto, Generating more MNT elliptic curves, *Designs, Codes and Cryptography* **38** (2006), 209–217.

[56] M. Scott, N. Benger, M. Charlemagne, L. Dominguez Perez and E. Kachisa, On the final exponentiation for calculating pairings on ordinary elliptic curves, in: *Pairing-Based Cryptography – Pairing 2009*, pp. 78–88, Lecture Notes in Computer Science 5671, Springer-Verlag, Berlin, Heidelberg, 2009.

[57] J. A. Solinas, Generalized mersenne numbers, Research Report CORR-99-39, Center for Applied Cryptographic Research, University of Waterloo, 1999.

[58] A. Svoboda and M. Valach, Operational circuits, *Stroje na Zpracovani Informaci, Sbornik III, Nakl. CSAV, Prague* (1955), 247–295 (in Czech).

[59] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*, McGraw-Hill, New York, 1967.

[60] F. Vercauteren, Optimal pairings, *IEEE Transactions on Information Theory* **56** (2010), 455–461.

[61] D. Weber and T. Denny, The solution of McCurley's discrete log challenge, in: *Advances in Cryptology – CRYPTO 98*, pp. 458–471, Lecture Notes in Computer Science 1462, Springer-Verlag, Berlin, Heidelberg, 1998.

Received October 30, 2010; revised February 17, 2011; accepted March 20, 2011.

Author information

Sylvain Duquesne, IRMAR, UMR CNRS 6625, Université Rennes 1,
Campus de Beaulieu, 35042 Rennes cedex, France.
E-mail: sylvain.duquesne@univ-rennes1.fr