Combining leak–resistant arithmetic for elliptic curves defined over $\mathbb{F}_p$ and RNS representation

JC Bajard[a], S. Duquesne[b], M Ercegovac[c]
[a] LIP6, UPMC Paris, France,
and LIRMM, CNRS, France;
[b] IRMAR, CNRS Université Rennes 1, France;
[c] UCLA, Computer Science Department, Los Angeles, California, USA

May 23, 2010

Abstract

In this paper we combine the residue number system (RNS) representation and the leak-resistant arithmetic on elliptic curves. These two techniques are relevant for implementation of elliptic curve cryptography on embedded devices.
It is well known that the RNS multiplication is very efficient whereas the reduction step is costly. Hence, we optimize formulae for basic operations arising in leak-resistant arithmetic on elliptic curves (unified addition, Montgomery ladder) in order to minimize the number of modular reductions. We also improve the complexity of the RNS modular reduction step. As a result, we show how to obtain a competitive secured implementation.
Finally, we show that, contrary to other approaches, ours takes optimally the advantage of a dedicated parallel architecture.

Keywords: Elliptic curves, Montgomery algorithm, leak-resistance, residue number system (RNS), modular multiplication, modular reduction.

1 Introduction

Because of their small key length, elliptic curve cryptosystems (ECC) have become popular to such a degree that they have recently been recommended by NSA. Their small key size is especially attractive for small cryptographic devices like smart cards. However, such devices are sensitive to side-channel attacks. These attacks consist of analyzing side-channel information, like timings [27], power consumptions [28] and electromagnetic radiations [35] of a device. They have become a serious threat that protecting ECC against them has become a whole research area in itself and has given rise to various countermeasures [17].

There are two types of such attacks, the simple power analysis (SPA) and the differential power analysis (DPA). The protection again DPA is relatively easy to obtain [17] and has no consequences on this paper. That is the reason why we concentrate on SPA. The weakness comes from the difference in complexity between the addition and the doubling on elliptic curves. There are two ways to deal with this. The first is to use representations of the curve for which the two operations are obtained with the same formulae as in [29], [25] or [11]. The second is to use an algorithm for the scalar multiplication due to Montgomery [31] for a family of curves defined over $\mathbb{F}_p$ and generalized in [19], [11] and [24]. This algorithm has many advantages for constrained environments: it is SPA-resistant, very simple to implement, modest in memory requirements and does not require precomputations. In classical multiprecision systems, the cost of these methods is given by the number of modular multiplications. As this number is high in the algorithms evoked above, we propose to use an alternative representation of numbers: the Residue Number Systems. Indeed, this representation allows a very efficient multiplication by distributing the computations on small independent integers. Nevertheless, even if the multiplication becomes linear (in word basic operations), the modular reduction step remains quadratic. The RNS has other advantages, in particular, it is easily parallelizable. Moreover, it is not specific to a value of p contrary to Mersenne number based arithmetic. This is an important point for circuit maker which must offer a tool useful for all customers.

The aim of this paper is to combine leak-resistance arithmetic on elliptic curve and the Residue Number System. The same work can of course be done for classical elliptic curve arithmetic. However, the RNS is particularly well-suited for hardware implementations so that we choose to concentrate on leak-resistant arithmetic.
To obtain an interesting and efficient combination, we work in two directions.

- We rewrite the formulae on elliptic curve for minimizing the number of modular reduction, eventually despite an increasing number of multiplication. In some cases, we even propose new formulae adapted to this new constraint.
- We improve the complexity in the RNS reduction step by using the properties of the RNS bases [6] which can be used for ECC size. This is significant in our context of small key size (in practice we deal with 6 to 16 words).

By this way, we show that it is very interesting to use RNS in leak-resistant elliptic curve cryptography especially if a parallel architecture is used.

In the following, $\mathbf{K}$ denotes a field of characteristic $\neq 2, 3$ (which is a prime field $\mathbb{F}_p$ in practice) and $|n|_2$ denotes the bit-length of n.

2 Background properties of the different representations and algorithms

2.1 Modular multiplication

The 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 a modular reduction. Because of the small size of the numbers used with ECC (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., $X = \sum_{i=0}^{n} x_i \beta^i$ with $0 \leq x_i < \beta$), then $A \times B$ 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 these by the equation

$$A \times B = b_0 A + \beta(b_1 A + \beta(b_2 A \cdots + \beta b_n A) \dots).$$

with a complexity of n^2 word operations. We note that for the current ECC key size, Karatsuba or Toom-Cook approaches remain costlier than the classical algorithm, as discussed in the study made by the GMP group [1].

The reduction of an integer k modulo another integer p consists of finding the remainder of the Euclidean division of k by p. This operation is costly. It can be substantially speeded up by using the Montgomery reduction (we will now review this method as used in RNS) or by using a special modulo.

Montgomery general reduction algorithm:

In [30], Montgomery proposed to substitute the reduction modulo p by a division by a power of the radix β (i.e., a simple shift). The result is not exactly $A \mod p$ but $A\beta^{-n} \mod p$. This problem can be overcome by using Montgomery representation where $A' = A \times \beta^n \mod p$ in this representation.

Algorithm 1: Montgomery$_p(R)$

Data: $R = A \times B < p\beta^n$ and $\beta^{n-1} \leq p < \beta^n$
and a precomputed value $(-p^{-1} \bmod \beta^n)$;
Result: (q, r) such that $r \equiv R\beta^{-n} \pmod p$, with $r < 2p$;
$q \leftarrow -R \times p^{-1} \bmod \beta^n$;
$r \leftarrow (R + qp)/\beta^n$;

The complexity of this reduction is n^2+n word operations [9]. Since all the computations can be done in Montgomery representations, we ignore the cost of the conversion from Montgomery to classic representation (and reciprocally). For $A < \beta^n$, its Montgomery representation is

[1] Intel Pentium-4 gmp-mparam.h
#define MUL_KARATSUBA_THRESHOLD 23
#define MUL_TOOM3_THRESHOLD 137

obtained via Algorithm 1 with $R = A \times (\beta^n \mod p)$. By the same way, if A' is the Montgomery representation of A, then we recover A using Algorithm 1 with $R = A'$. We note, as $r < 2p$, that a comparison and a final subtraction could occur, but the output of Algorithm 1 can be used as input by adding a condition on p, specifically $4p < \beta^n$.

Reduction using special modulo:

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 [38, 14]. This is used in most of the standards but the main drawback of this approach is that a dedicated architecture to a such a particular p cannot be used for other prime fields. Consequently, it is not practical in either software or hardware implementation and many customers prefer flexible products. For this reason we do not consider this restrictive approach in this paper.

2.2 Leak-resistant arithmetic in elliptic curve cryptography

In all elliptic curve based schemes (such as encryption/decryption or signature generation/verification), the dominant operation is the scalar multiplication of points on the curve. Hence, the efficiency of this operation is crucial in elliptic curve cryptography. This is usually done by using standard scalar multiplication methods such as double and add or sliding window methods combined with recoding of the exponent.

However, these methods are not SPA-resistant because of the difference in complexity between addition and doubling operations. There are some methods to protect these algorithms. For example, if one wants to protect a double and add algorithm against side-channel attacks, extra (useless) additions can be performed [17]. In this way, for each bit of the exponent, we perform both an addition and a doubling such that bits of the exponent are indistinguishable. Unfortunately this protection is expensive and also vulnerable to fault attacks.

Currently there are two means to perform SPA-resistant arithmetic on elliptic curves. The first uses unified addition formulae. This means that we use a representation of the curve for which the addition and the doubling can be performed using the same formulae. In the following, we will present unified formulae for three representations of the curve, i.e. the Hessian form, the Jacobi form (leading to the most efficient formulae but not applicable to all elliptic curves) and the short Weierstrass form (which is the general case). The second one uses the Montgomery ladder where both an addition and a doubling are performed at each step of the scalar multiplication algorithm. Again, in this case, the arithmetic is more efficient on restrictive models of the curve and we will present both the restrictive and general model.

We are not interested in this paper in the protection against DPA. Indeed, in elliptic curve cryptography, this problem is efficiently solved by randomizing the projective coordinates, the base point or the scalar [17].

Unified addition formulae

The use of the Hessian form for a leak-resistant arithmetic has been introduced in [25]. An elliptic curve over $\mathbb{F}_p$ is said to be in Hessian form if it is given by an equation of the form

$$X^3 + Y^3 + Z^3 = 3dXYZ$$

where $d \in \mathbb{F}_p$ and is not a third root of unity. Such curves have a point of 3-torsion, i.e., their cardinality is divisible by 3. Consequently, not all elliptic curves can be given in this form. In [25], Joye and Quisquater described formulae for the addition of two points (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) for the elliptic curve in such a representation:

$$\begin{cases} X_3 &= Y_1^2 X_2 Z_2 - Y_2^2 X_1 Z_1, \\ Y_3 &= X_1^2 Y_2 Z_2 - X_2^2 Y_1 Z_1, \\ Z_3 &= Z_1^2 X_2 Y_2 - Z_2^2 X_1 Y_1. \end{cases}$$

These formulae require 12 field multiplications and can be used for addition and doubling because we have

$$2(X, Y, Z) = (Z, X, Y) + (Y, Z, X).$$

At the same time, the use of the Jacobi model was introduced by Liardet and Smart in [29]. It is improved in [8] and, more recently, in [18]. It is easy to prove that any elliptic curve containing a 2-torsion point is birationally equivalent to the Jacobi quartic given by an equation of the form

$$Y^2 = \varepsilon X^4 - 2\delta X^2 Z^2 + Z^4,$$

where ε and δ are constants in $\mathbb{F}_p$. In this case, the formulae for the addition of two points (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) are also valid if the two points are the same.

$$\begin{cases} X_3 &= X_1 Z_1 Y_2 + Y_1 X_2 Z_2, \\ Y_3 &= \left(Z_1^2 Z_2^2 + \varepsilon X_1^2 X_2^2\right)(Y_1 Y_2 - 2\delta X_1 X_2 Z_1 Z_2) \\ &\quad +2\varepsilon X_1 X_2 Z_1 Z_2 \left(X_1^2 Z_2^2 + Z_1^2 X_2^2\right), \\ Z_3 &= Z_1^2 Z_2^2 - \varepsilon X_1^2 X_2^2. \end{cases}$$

In most cases, ε can be rescaled to a small value so that it is not too restrictive to neglect multiplication by ε. Thus these formulae also require 12 multiplications, as explained in [18]. However, this method cannot be applied to any elliptic curve since the cardinality of a Jacobi quartic is even.

In [11], Brier and Joye give unified formulae for a curve given in short Weierstrass form (which is not restrictive over $\mathbb{F}_p$, where p is a large prime number)

$$Y^2 Z = X^3 + aXZ^2 + bZ^3.$$

Again, formulae given for the addition of two points (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) are also valid if the two points are the same.

$$\begin{cases} X_3 &= 2\lambda_d \left(\lambda_n^2 - (X_1Z_2 + X_2Z_1)(Y_1Z_2 + Y_2Z_1)\lambda_d\right), \\ Y_3 &= \lambda_n \left(3(X_1Z_2 + X_2Z_1)(Y_1Z_2 + Y_2Z_1)\lambda_d - 2\lambda_n^2\right) - ((Y_1Z_2 + Y_2Z_1)\lambda_d)^2, \\ Z_3 &= 2\lambda_d^3, \end{cases}$$

where

$$\begin{aligned} \lambda_n &= (X_1Z_2 + X_2Z_1)^2 - X_1X_2Z_1Z_2 + aZ_1^2Z_2^2, \\ \lambda_d &= (Y_1Z_2 + Y_2Z_1)Z_1Z_2. \end{aligned}$$

These formulae are valid for all elliptic curves but are less efficient since they require 18 field multiplications. Note that by using an isomorphism or an isogeny as in [12], it is possible in most cases to rescale a to a small value. We will explain this in detail in Section 4.3 within Montgomery arithmetic context.

Montgomery scalar multiplication

Montgomery proposed in [31] to work only with the x-coordinate. Of course, the group law is lost but traces remain. So doubling is still possible and the addition of two points P and Q is possible if $P-Q$ is known. Montgomery gives formulae for operations when the curve admits a Montgomery form, which means that it can be defined by a (non general) equation of the type

$$By^2 = x^3 + Ax^2 + x.$$

Proposition 1 *Let E be an elliptic curve defined over $\mathbb{F}_p$ in Montgomery form. Let also $P = (X_p, Y_p, Z_p)$ and $Q = (X_q, Y_q, Z_q) \in E(\mathbf{K})$ given in projective coordinates. Assume that $P - Q = (x, y)$ is known in affine coordinates. Then the X and Z-coordinates for $P + Q$ and $2P$ are given by*

$$\begin{aligned} X_{p+q} &= ((X_p - Z_p)(X_q + Z_q) + (X_p + Z_p)(X_q - Z_q))^2, \\ Z_{p+q} &= x\left((X_p - Z_p)(X_q + Z_q) - (X_p + Z_p)(X_q - Z_q)\right)^2, \\ 4X_pZ_p &= \left((X_p + Z_p)^2 - (X_p - Z_p)^2\right), \\ X_{2p} &= (X_p + Z_p)^2(X_p - Z_p)^2, \\ Z_{2p} &= 4X_pZ_p\left((X_p - Z_p)^2 + \tfrac{A+2}{4}4X_pZ_p\right). \end{aligned}$$

In this way, both an addition and a doubling takes only 3 multiplications and 2 squares, which is much faster than usual operations ([16]). The fact that the difference $P-Q$ must be known to compute $P+Q$ implies that a new algorithm must be used to compute the scalar multiplication of a point G by an integer k. The solution is to use pairs of consecutive multiples of G, so that the difference between the two components of the pair is always known and equal to G. The scalar multiplication algorithm is as follows:

Algorithm 2: Montgomery_Scalar()

Data: $G \in E(\mathbb{F}_p)$ and $k \in \mathbb{Z}$

Result: x and z-coordinate of kG

Initialize $T = (\mathcal{O}, G)$ where $\mathcal{O}$ is the point at infinity;
For i from $|k|_2 - 1$ to 0 do;
  If $k_i = 0$ then $T = (2T[1], T[1] + T[2])$;
  If $k_i = 1$ then $T = (T[1] + T[2], 2T[2])$;
return $T[1]$;

Both an addition and a doubling are done for each bit of the exponent. So the cost of this algorithm is about $10|k|_2$ multiplications for a curve in Montgomery form which is better than other available algorithms.
Moreover, the operations we have to perform do not depend on the bits of the exponent, so this method has interesting leak-resistance properties.
Finally, x-coordinate of kG is usually sufficient but some cryptosystems, like ECDSA, require y-coordinate. It can easily be recovered as explained in [32].

Unfortunately, in odd characteristic, all elliptic curves cannot be transformed into Montgomery form. This is, for example, the case for most standards. The reason is that any curve which can be transformed into Montgomery form has a 2-torsion point so its cardinality is not prime (it is divisible by 2).

In general, the curve is defined by an equation of the form

$$y^2 = x^3 + ax + b. \tag{1}$$

This method can also be applied but is more time consuming ([11],[19] and [24]).

Proposition 2 *Let E be an elliptic curve defined over $\mathbb{F}_p$ by (1). Let also $P = (X_p, Y_p, Z_p)$ and $Q = (X_q, Y_q, Z_q) \in E(\mathbb{F}_p)$ given in projective coordinates. Assume that $P - Q = (x, y)$ is known in affine coordinates. Then we obtain X and Z-coordinates for $P + Q$ and $2P$ by the following formulae:*

$$\begin{aligned}
X_{p+q} &= -4bZ_pZ_q(X_pZ_q + X_qZ_p) + (X_pX_q - aZ_pZ_q)^2, \\
Z_{p+q} &= x(X_pZ_q - X_qZ_p)^2, \\
X_{2p} &= \left(X_p^2 - aZ_p^2\right)^2 - 8bX_pZ_p^3, \\
Z_{2p} &= 4Z_p\left(X_p^3 + aX_pZ_p^2 + bZ_p^3\right).
\end{aligned}$$

Addition can be performed in 10 multiplications and doubling in 9. Hence, the scalar multiplication can be performed in about $19|n|_2$ multiplications on $\mathbb{F}_p$ which is not interesting in terms of performance but it is in terms of leak-resistance. Note that the y-coordinate can also be recovered in this case ([11]).

Proposition 3 *Suppose that $Q = P+G$ with $G = (x, y)$, $P = (x_p, y_p)$ and $Q = (x_q, y_q)$. Then, if $y \neq 0$, one has*

$$y_p = -\frac{2b + (a + x.x_p)(x + x_p) - x_q(x - x_p)^2}{2y}$$

With the Montgomery scalar multiplication method, we always perform both an addition and a doubling for each bit of the exponent. Consequently, this method is resistant against side-channel attacks which is the reason why this method is interesting even with 19 multiplications per step.

In this paper, we will use Residue Number Systems (RNS) for the arithmetic on the base field. The consequence is that the cost of the multiplication becomes negligible compared to the cost of the modular reduction. Thus, it is necessary to rewrite the formulae given above in order to minimize the number of modular reductions. Let us now briefly review this representation system.

3 Residue Number Systems

3.1 Representation

The Residue Number Systems (RNS) 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 $a_i = a \bmod m_i = |a|_{m_i}$. We generally assume that $0 \leq a < M = \prod_{i=1}^{n} m_i$. The elements a_i are called RNS-digits, or simply digits if there is no ambiguity. The strongest advantage of a such system is that it distributes large integer operations on the small residue values. The operations are performed independently on the residues. These systems were introduced and developed in [21, 22, 39]. A good introduction can be found in [26].

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-digit products. A RNS-digit product is equivalent to a classical digit product followed by a modular reduction modulo m_i, which represents few additions (see [7, 6]).

$$r = (|a_1 \times b_1|_{m_1}, \ldots, |a_n \times b_n|_{m_n}) \equiv a \times b \pmod{M} \tag{2}$$

It is clear that if a product is followed by an addition (MAC operation), the cost is just increased by one addition on each modulo, and thus we consider that this operation is equivalent to a product.

$$r = (|a_1 \times b_1 + d_1|_{m_1}, \ldots, |a_n \times b_n + d_n|_{m_n}) \equiv a \times b + d \pmod{M} \tag{3}$$

In this paper, we consider RNS base $(m_1, ..., m_n)$ with elements such that, $m_i = 2^k - c_i$, where c_i is small and sparse, $c_i < 2^{k/2}$. For example, for $m_i < 2^{32}$, it is easy to find 16 coprime values with $c_i = 2^{t_i} \pm 1$, with $t_i = 0...16$ for $c_i = 2^{t_i} - 1$ and $t_i = 1...15$ if $c_i = 2^{t_i} + 1$. This ensures that $c_i < 2^{16}$. Then, if we want more co-prime values, we can consider the $c_i = 2^{t_i} \pm 2^{s_i} \pm 1$.

The reduction modulo m_i is, in this case, obtained with few shift and adds. For $r < 2^{2k}$ (for example the result of a MAC operation with operands lower than 2^k), we consider that $r = r_h 2^k + r_l \equiv c_i r_h + r_l \pmod{m_i}$ where the product by c_i is just a shift and add processing. Hence, if we note $r' = c_i r_h + r_l = r'_h 2^k + r'_l$ with $r' < 2^{3k/2}$ when $r < 2^{2k}$, we ensure that $r" = c_i r'_h + r'_l \leq 2(2^k - 2^{k/2}) < 2m_i$. This property ensures that the reduction part on each m_i, in the case of a product (or MAC) operations, represents around 10% of the cost [10, 7, 6]. In the following we consider that a RNS digit-product is equivalent to 1.1 word-product (word = k-bits).

We now focus on the multiplication modulo p using the Montgomery algorithm presented in [1, 2]. This algorithm for two numbers a and b given in RNS, actually evaluates $r = abM^{-1} \mod p$. To obtain the right result, we need to use it again with r and $(M^2 \mod p)$ as operands. To avoid this, we convert the values in a Montgomery representation where $a' = a \times M \mod p$ which is stable for Montgomery product and addition. This conversion is done once at the beginning by performing Montgomery product with a and $(M^2 \mod p)$ 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 [4].

3.2 RNS Montgomery reduction

This algorithm 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 β number system, the value β^n occurs in the reduction, division and Montgomery factor. In RNS, this value is replaced by M. Thus an auxiliary RNS base is needed to handle the inverse of M. Hence some operation as the initial product will be performed on the two bases, which cost $2n$ words-products.

Algorithm 3 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 3: MontgR_RNS(c, p)

Data: 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 positive integer p represented in RNS in both bases such that $0 < 4p < M$ and $\gcd(p, M) = 1$ (p is prime);

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

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

begin

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*;

end

Instructions 1 and 3 of Algorithm 3 deal with RNS operations as presented in the previous section, which are performed independently for each element of the basis, so they are very efficient. These two instructions are linear (or constant number of words-operations on a n cells architecture) Instructions 2 and 4 represent RNS base extensions which are quadratic (or linear on an n-cell architecture) are costly. To reduce this cost, we can use two different full RNS extensions as shown in [1, 2]. The extension to base $\mathcal{B}'$ of q (instruction 2), obtained in its RNS form $(q_1, \ldots, q_n)$ in the base $\mathcal{B}$. In other words, q is computed modulo M, and $(c + q \times p)$ is a multiple of M which can be divided by M in $\mathcal{B}'$ by multiplying with M^{-1}.

3.3 RNS Base extensions

We consider that we use bases of the following form: $m_i = 2^k - c_i$ with $c_i = 2^{t_i} \pm 1$ (or $c_i = 2^{t_i} \pm 2^{s_i} \pm 1$ if we need more elements, see Annexes) and $c_i < 2^{\frac{k}{2}}$ (for ECC k can be equal to 32, for keys up to 1024 bits and maybe more). Due to that fact, the best choice for the base extension is done with first a Mixed Radix conversion and then a Horner evaluation (it is related to a Newton interpolation approach).

The MRS Representation of $(\tilde{a}_1, \tilde{a}_2, \cdots, \tilde{a}_n)$ of the integer a given in its RNS representation $(a_1, a_2, \cdots, a_n)$ is obtained with:

$$
\begin{aligned}
\tilde{a}_1 &= a_1 \\
\tilde{a}_2 &= \left|(a_2 - \tilde{a}_1)m_{1,2}^{-1}\right|_{m_2} \\
\tilde{a}_3 &= \left|\left(\left|(x_3 - \tilde{a}_1)m_{1,3}^{-1}\right|_{m_3} - \tilde{a}_2\right)m_{2,3}^{-1}\right|_{m_3} \\
\tilde{a}_4 &= \left|\left(\left|\left(\left|(x_4 - \tilde{a}_1)m_{1,4}^{-1}\right|_{m_4} - \tilde{a}_2\right)m_{2,4}^{-1}\right)\right|_{m_4} - \tilde{a}_3\right)m_{3,4}^{-1}\Bigg|_{m_4} \\
&\vdots \\
\tilde{a}_n &= \left|\left(. \left|\left(\left|(x_n - \tilde{a}_1)m_{1,n}^{-1}\right|_{m_n} - \tilde{a}_2\right)m_{2,n}^{-1}\right)\right|_{m_n} - . - \tilde{a}_{n-1}\right)m_{n-1,n}^{-1}\Bigg|_{m_n}
\end{aligned}
\tag{4}
$$

where $m_{i,j}^{-1}$ is the inverse of m_i modulo m_j

Then, the reconstruction of A is given by:

$$
A = \tilde{a}_1 + m_1(\tilde{a}_2 + m_2(\tilde{a}_3 \cdots + m_{n-2}(\tilde{a}_{n-1} + m_{n-1}\tilde{a}_n)\cdots)) \tag{5}
$$

We point out that this transformation does not need any products, only shifts and adds. We transpose this equation modulo the elements of the new RNS base $\mathcal{B}'$ with $j = n+1...2n$.

$$
a_j = \left|\tilde{a}_1 + m_1|\tilde{a}_2 + m_2|\tilde{a}_3 \cdots + m_{n-2}|\tilde{a}_{n-1} + m_{n-1}\tilde{a}_n|_{m_j} \cdots |_{m_j}|_{m_j}\right|_{m_j} \tag{6}
$$

The two base extensions of Algorithm 3 are similar.

3.4 Analysis of the complexity

In the literature, the complexity is given in number of word-multiplications. So, we present here the cost of Algorithm 3 by counting the multiplications needed.

In step 1 and 3 the evaluations of q and r are made in RNS independently for each modulus. The value q is computed on base $\mathcal{B}$ that represents n multiplications by a constant value $|p|_{m_i}^{-1}$. The calculation of r is performed by $2n$ multiplications in base $\mathcal{B}'$.

Now for the base extension, multiplications occur only in the conversion to Mixed Radix given by Equation 4. The number of multiplications by a constant (the $|m_i|_{m_j}^{-1}$) is $\frac{n^2-n}{2}$.

$$
T_{RNS-MRS}(n) = \frac{n^2 - n}{2} \text{ RNSdigit-products} \tag{7}
$$

In the conversion from MRS to RNS, the basic operation $|a + m_i b|_{m_j}$ corresponds to few shifts and adds. Indeed, $a + m_i b = a + 2^k b - 2^{t_i} b \pm b$ can be done in two additions ($a + 2^k b$ is just a concatenation). We had seen that the reduction modulo m_j represents 3 additions ($c_i a'_h$ is a concatenation). We note another point: we can deal with values lower than $2m_j$ instead of m_j. In fact we only need a value lower than m_i in the RNS-MRS conversion part. This remark implies that $c_i < 2^{\frac{k}{2}-1}$ for a direct reuse of the result of one step in the Horner chain. Thus, the evaluation of each a_j needs $5n$ word-additions. As the product is the basic operation for most of the complexity studies found in the literature, we consider that these 5 word-additions

are equivalent to $\frac{1}{5}$ of an RNSdigit-product [10]. Hence the complexity of the conversion from MRS to RNS represents:

$$T_{MRS-RNS}(n) = \frac{1}{5}(n^2 - n) \text{ RNSdigit-products} \tag{8}$$

As we need two extensions in Algorithm 3, the total complexity of this algorithm is :

$$T_{Algo3}(n) = n^2 - n + \frac{2}{5}(n^2 - n) + 3n = \frac{7}{5}n^2 + \frac{8}{5}n \text{ RNSdigit-products} \tag{9}$$

This is asymptotically better than previous result which are in $O(n^2)$.

If we operate with an architecture of n basic word-arithmetic cells, Algorithm 3 can be performed in a parallel manner. In this case, due to the independency of the RNS, the evaluation requires, $(n-1)$ steps for the conversion RNS-MRS, and $\frac{1}{5}(n-1)$ for the RNS-MRS conversion, one step for each RNS product. Hence, a parallel evaluation of Algorithm 3 can be done in $\frac{12}{5}n + \frac{3}{5}$ steps.

3.5 Discussion of the advantages

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

However, if we count separately the multiplication and the modular reduction, with the property that a RNS multiplication needs only $2n$ RNSdigit-products or $2.2n$ word-products, the advantage of the RNS appears more clearly. This point is developed in the rest of the paper where by reformulating additions on elliptic curves, we propose solutions up to 30% better than the classical approaches.

Furthermore, 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, with RNS, computations can be performed in a random way and the architecture can be parallelized.

Parallelization of the architecture, with n basic operators, has a time complexity of 2 modular digit-operations for the multiplication (or multiplication-addition) and $\frac{12}{5}n + \frac{3}{5}$ for the modular reduction. This indicates that if we accumulate some operations (i.e., sum of products) before reduction we obtain an efficient implementation ([3]). We develop this approach in the next section with ECC.

The last advantage of the RNS is the flexibility of the architecture. With a given structure of n modular digit operators, it is possible to handle many values of p which satisfy : $4\times p < M$. If we refer to Algorithm 3, we note that the only values depending on p are: $\|p\|_{m_i}$ for $i = 1, ..., n$.

Thus, by reinitializing these pre-computed values, the system can be adapted for a new value of p. If p is relatively small as compared to M, we can adjust the RNS base by reducing it by some m_i. In this case, we will use partial RNS bases $(m_1, ..., m_{\tilde{n}})$ and $(m_{n+1}, ..., m_{n+\tilde{n}})$ with $\tilde{n} < n$. With a control part which takes $\tilde{n}$ into account, we can assume that the system performance depends on the size of p.

Hence, the RNS approach proposed in this paper offers different levels of adaptability and scalability. We also note that all the products which occur in the RNS conversions, are, at least, with a constant value. Another point is that we present a solution with no specific values for the inverses of the bases, but it is shown in [6] that specific bases exist. In this case, we can achieve a complexity lower than n^2 RNS digit-products for Algorithm 3, where many multiplications are replaced by additions if the inverses of the m_i modulo the following elements of the bases are sparse. However, these last cases are available for elements composed of more than 32 bits.

4 SPA-resistant arithmetic on elliptic curves optimized for RNS representation

The aim of this section is to rewrite or modify the formulae given in section 2.2 in order to minimize the number of modular reductions, since this is the most expensive operation in RNS representation. Thus we have to group together several multiplications and perform only one reduction.

4.1 Unified addition formulae

This can be well illustrated by the formulae for Hessian elliptic curves. We give here the steps required to compute the sum of two points (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2). The costs are given by the number of modular reductions.

step	operations	red.	mul.
computation of	$A = X_2Y_1, \quad B = Y_1Z_2, \quad C = X_1Y_2$	3	3
intermediate products	$D = Y_2Z_1, \quad E = X_1Z_2, \quad F = X_2Z_1$	3	3
computation of X_3	$AB - CD$	1	2
computation of Y_3	$EC - FA$	1	2
computation of Z_3	$EB - FD$	1	2

Thus the total cost in RNS representation is 9 modular reductions, which has to be compared to the 12 base field multiplication in standard representation.

Concerning the Jacobi quartic, the cost in terms of modular reductions (and the formulae) is given in [18] and is equal to 10, whereas 12 multiplications are necessary.

Finally, we give details of the steps for computing the sum of two points using unified addition formulae for a curve given in short Weierstrass form.

step	operations	red.	mul.
computation of λ_n	$A = X_2Z_1, \quad B = X_1Z_2,$	2	2
	$C = Z_1Z_2, \quad D = aC$	2	2
	$\lambda_n = (A+B)^2 - AB + CD$	1	3
computation of λ_d	$E = Y_1Z_2 + Y_2Z_1$	1	2
	$\lambda_d = EC$	1	1
intermediary	$F = E\lambda_d, \quad G = \lambda_n^2$	2	2
computations	$H = F(A+B)$	1	1
computation of X_3	$2\lambda_d(G-H)$	1	1
computation of Y_3	$\lambda_n(3H-2G) - F^2$	1	2
computation of Z_3	$2\lambda_d^3$	2	2

In this case, the total cost in RNS is 14 modular reductions, whereas 18 multiplications must be performed.

Thus, for all known unified formulae, computation of the addition requires fewer reductions than multiplications. This means thatusing the RNS representation can become attractive in terms of performance. In addition, it has all the advantages described in Section 3.5. A detailed comparison is given in Section 5 .

4.2 Montgomery formulae

In this section, we are interested in the Montgomery ladder. In the restrictive case of curves in Montgomery form (the cardinality of the curve is even), both 10 multiplications and modular reductions are required. Moreover, it is easy to see that it is not possible to have a better result because of the degree of the formulae. The general case of curves in Weierstrass form is much more interesting. Indeed, one can do better than re-using the formulae described in Section 2.2. Following the strategy used for the unified addition formulae leads to 16 modular reductions and 19 multiplications. Slightly rewriting the formulae given in Section 2.2 already makes it possible to perform only 15 modular reductions. It is actually possible to further reduce this complexity by resuming, from the beginning, the Montgomery ladder from a more theoretical standpoint.

The Montgomery ladder is based on the fact that the y-coordinate has only minor information. Indeed, it only allows to distinguish a point and its opposite (or equivalently a point and its image under the hyperelliptic involution). Thus the Montgomery ladder only deals with the x-coordinate. From a theoretical standpoint, this means that we are working on the quotient of the curve by the hyperelliptic involution: the Kummer variety. Of course, taking such a quotient implies that it is not possible to add two different points since $P+Q$ and $P-Q$ are

not equal in the Kummer variety. However, doubling is still possible (it is easy to discern $P+P$ and $P-P=\mathcal{O}$) and if $P-Q$ is known, it is possible to discern $P+Q$ and $P-Q$. More precisely, it is proved in [20] that there are biquadratic forms M_x, M_z and M_{xz}, such that for any points $P=(X_p,Z_p)$ and $Q=(X_q,Z_q)$ on the Kummer variety, we have

$$\begin{aligned} 2X_{p+q}X_{p-q} &= M_x \\ X_{p+q}Z_{p-q}+X_{p-q}Z_{p+q} &= M_{xz} \\ 2Z_{p+q}Z_{p-q} &= M_z \end{aligned}$$

with

$$\begin{aligned} M_x &= (X_pX_q-aZ_pZ_q)^2-4bZ_pZ_q(Z_pX_q+X_pZ_q), \\ M_{xz} &= X_pX_q(Z_pX_q+X_pZ_q)+Z_pZ_q(a(Z_pX_q+X_pZ_q)+2bZ_pZ_q) \\ M_z &= (Z_pX_q-X_pZ_q)^2 \end{aligned}$$

If $P-Q$ is known, one can easily deduce formulae to compute X_{p+q} and Z_{p+q} from these biquadratic forms. In fact, only two of the biquadratic forms are necessary. For instance, formulae obtained (in another way) by Brier and Joye in [11] and given in Proposition 2 can be easily deduced from M_x and M_z. Here, we will use M_{xz} and M_z, in order to minimize the number of modular reductions. In the context of the Montgomery ladder (Algorithm 2), the difference between the two points we want to add is always the base point $G=(x,y)$, which is given in affine coordinates so that $Z_{p-q}=1$ and $X_{p-q}=x$. Thus we obtain

$$\begin{aligned} X_{p+q} &= 2(M_{xz}-xZ_{p+q}) \\ Z_{p+q} &= M_z \end{aligned}$$

Let us note that the theory of Kummer varieties also provides formulae for doubling but these always lead to the same formulae as in Proposition 2. Therefore, we have the following theorem.

Theorem 1 *Let p be a prime number and E be an elliptic curve defined over $\mathbb{F}_p$ by (1). Let also $P=(X_p,Y_p,Z_p)$ and $Q=(X_q,Y_q,Z_q)\in E(\mathbb{F}_p)$ given in projective coordinates. Assume that $P-Q=(x,y)$ is known in affine coordinates. Then we obtain the X and Z-coordinates for $P+Q$ and $2P$ in terms of the X and Z-coordinates for P and Q by the following formulae:*

$$\begin{aligned} X_{p+q} &= 2\left(X_pX_q(Z_pX_q+X_pZ_q)+Z_pZ_q(a(Z_pX_q+X_pZ_q)+2bZ_pZ_q)-xZ_{p+q}\right), \\ Z_{p+q} &= (X_pZ_q+X_qZ_p)^2-4X_pX_qZ_pZ_q, \\ X_{2p} &= \left(X_p^2-aZ_p^2\right)^2-8bX_pZ_p^3, \\ Z_{2p} &= 4X_pZ_p\left(X_p^2+aZ_p^2\right)+4bZ_p^4. \end{aligned}$$

Finally, we give details of the steps for computing the sum of two points and the doubling of a point

step	operations	red.	mul.
preliminary computations	$A = Z_p X_q + X_p Z_q$	1	2
	$B = 2X_p X_q, \quad C = 2Z_p Z_q$	2	2
computation of Z_{p+q}	$A^2 - BC$	1	2
computation of X_{p+q}	$D = aA + bC$	1	2
	$BA + CD + 2xZ_{p+q}$	1	3
preliminary computations	$A = 2X_p Z_p$	1	1
	$B = X_p^2, \quad C = Z_p^2$	2	2
	$D = -4bA, \quad E = aA$	2	2
computation of X_{2p}	$BD + (C - E)^2$	1	2
computation of Z_{2p}	$2B(C + E) - AD$	1	2

In this case, the total cost for each bit of the exponent in RNS representation is 13 modular reductions and 20 multiplications whereas 19 base field multiplications must be performed in a standard representation. Hence the use of an arithmetic which complexity is concentrated on the reduction step (as the RNS) becomes very attractive with these formulae.
It is interesting to note that, contrary to the case of the standard representation, the extra cost for curves in short Weierstrass form compared to (more specific) curves in Montgomery form is not too large (33% in RNS representation compared to 90% in standard representation).
Lastly, if a (or b) is a small number, the cost becomes 12 modular reductions whereas 17 base field multiplications must be performed in a standard representation. Let us now show that we can almost always assume that either a or b is small.

4.3 Rescaling the constant to a small value

This section is not specific to the RNS representation and can be applied in other contexts. It is motivated by the fact that there are 2 multiplications by a in the general formulae for the Montgomery ladder. Thus, the gain will be attractive if a can be rescaled to a small value.

The standard way to perform such a rescaling is to find a small k such that $\frac{a}{k}$ is a fourth power u^4 in $\mathbb{F}_p$ and to use the isomorphism $(x, y) \mapsto \left(\frac{x}{u^2}, \frac{y}{u^3}\right)$ to send E on the curve E' defined by the equation $y^2 = x^3 + \frac{a}{u^4}x + \frac{b}{u^6}$. The probability for an element of $\mathbb{F}_p$ to be a fourth power is only $\frac{1}{4}$ and we can obtain a better result in the context of the Montgomery ladder. Indeed, y is not used in this representation so only u^2 will be used and it is actually sufficient that $\frac{a}{k}$ is a square in $\mathbb{F}_p$. The isomorphism is now defined over $\overline{\mathbb{F}_p}$, so that it is easier to use a change of variables to describe the rescaling.

Theorem 2 *Let E be an elliptic curve defined over $\mathbb{F}_p$ by (1) and k be a small integer such that $\frac{a}{k}$ is a square v^2 in $\mathbb{F}_p$. Let also $P = (X_p, Y_p, Z_p)$ and $Q = (X_q, Y_q, Z_q) \in E(\mathbb{F}_p)$ given in projective coordinates. Assume that $P - Q = (x, y)$ is known in affine coordinates. Put $Z' = vZ$. If $b' = \frac{b}{v^3}$ and $x' = \frac{x}{v}$ are precomputed, we obtain the X and Z'-coordinates for $P + Q$*

and $2P$ in terms of the X and Z'-coordinates for P and Q by the following formulae:

$$\begin{aligned} X_{p+q} &= -4b'Z'_pZ'_q(X_pZ'_q + X_qZ'_p) + (X_pX_q - kZ'_pZ'_q)^2, \\ Z'_{p+q} &= x'(X_pZ'_q - X_qZ'_p), \\ X_{2p} &= \left(X_p^2 - kZ_p'^2\right)^2 - 8b'X_pZ_p'^3, \\ Z'_{2p} &= 4Z'_p\left(X_p^3 + kX_pZ_p'^2 + b'Z_p'^3\right). \end{aligned}$$

In this case, addition can be performed in 9 multiplications and doubling in 8. The same idea can be applied to formulae optimized for the RNS representation given in Section 4.2.

As the constraint on k has been relaxed ($\frac{a}{k}$ must be a square, not necessarily a fourth power), it is easier to rescale a to a small value in the context of Montgomery ladder than in the general context. Using the properties of the Legendre symbol, it is easy to prove that k is either 1 or the smallest non-square in $\mathbb{F}_p$ and that the proportion of prime fields such that the n first prime numbers are squares is only $\frac{1}{2^n}$.

Anyway, if k is too large to assume that the multiplication by k can be neglected, there is another way to rescale a to a small value. This method uses isogenies and is explained in [12]. Finally, the method explained for rescaling a to a small value can also be applied to b if there is a small k such that $\frac{4b}{k}$ is a cube in $\mathbb{F}_p$ which leads to the same gain (2 multiplications).

In conclusion, the probability that neither a nor b can be rescaled (by using Z' or isogenies) to a small value is very low, especially in the Montgomery ladder context.

5 Performance Comparisons

In this section, we compare the complexities of our approach to those using Montgomery modular multiplication.
First, we summarize the complexities for base field operations in Table 1. Table 2 (resp. 3)

Operation	RNS (in RNSdigit-products)	Montgomery (in word-products)
Multiplication	$2n$	n^2
Reduction	$\frac{7}{5}n^2 + \frac{8}{5}n$	$n^2 + n$

Table 1: Number of word-products for performing a multiplication and a modular reduction in RNS and with Montgomery approach for two n-word integers. A RNSdigit-product is equivalent to 1.1 word-product (see section 3.1).

shows the number of operations required for a doubling **or** an addition (resp. a doubling **and** an addition) for the different representations of the curve we chose to deal with in this paper.

It is then easy to deduce the global complexity in each case. For instance, one step of the Montgomery exponentiation algorithm using the formulae given in Section 2.2 (for the

Curve representation	RNS representation	Standard representation
Hessian form	9 red. and 12 mul.	12 mul. and 9 red.
Jacobi form	10 red. and 12 mul.	12 mul. and 10 red.
Unified Weierstrass form	14 red. and 18 mul.	18 mul. and 14 red.

Table 2: Optimal number of operations in RNS and standard representation for an unified addition

Curve representation	RNS representation	Standard representation
Montgomery ladder	13 red. and 20 mul.	19 mul. and 16 red.
Montgomery ladder (a small)	12 red. and 18 mul.	17 mul. and 14 red.

Table 3: Optimal number of operations in RNS and standard representation for each bit of the exponent of a Montgomery ladder

Montgomery approach) or section 4.2 (for our approach) when a is small requires $17n^2+14(n^2+n)$ operations with Montgomery modular multiplication, and $\left(18(2n)+12(\frac{7}{5}n^2+\frac{8}{5}n)\right)\times 1.1$ using RNS. We summarize, in Table 4, the word complexity for each representation of the curve we considered in this paper (i.e., those having SPA-resistance properties). We also give these complexities for usual ECC sizes for a 32-bit architecture. All these complexities are given for one basic step of the scalar multiplication. For Montgomery ladder, such a step always requires both an addition and a doubling, so that the complexities are easy to compute. For unified formulae, we assume that this step requires 1.25 unified additions in average using, for example, a sliding window method with window size 3. This is not necessarily the best choice (for example in 512 bits) but this has no incidence on our comparisons.

It is interesting to note that the complexities we obtain in RNS are always asymptotically better than in the Montgomery representation. This is due to the fact that we optimized formulae on elliptic curves in order to minimize the number of reductions. As a consequence, our method becomes more interesting for high level of security. For example, the gain obtained is anecdotal for unified formulas in 192 bits but becomes interesting for higher level of security. The gain is even important for the Montgomery ladder because we discovered new formulae which are well adapted to the RNS representation of numbers. Moreover, all of the advantages of the RNS arithmetic become evident when a parallel architecture is used.

Indeed, assuming that we have an architecture equivalent to n word-operators on a single word-bus, Table 5 shows the complexities of the different approaches in number of word operations. Note that we only give these complexities in the case of the Montgomery ladder with a small in order to simplify the paper. The complexities for the other curves representations can be easily deduced from Table 4.

The estimation of the cost for the multiplication and for the Montgomery parallel product

Curve representation	size in bit	RNS	Montgomery	ratio in %
Hessian form	$32n$	$1.25\left(12.6n^2+38.4n\right)\times 1.1$	$1.25(21n^2+9n)$	
	192	940.5	1012	7 %
	256	1531.2	1770	13.5 %
	512	5280	6900	23.5 %
Jacobi form	32n	$1.25\left(14n^2+40n\right)\times 1.1$	$1.25(22n^2+10n)$	
	192	1023	1065	4 %
	256	1672	1860	10 %
	512	5808	7240	19.5 %
Unified Weierstrass form	32n	$1.25\left(19.6n^2+58.4n\right)\times 1.1$	$1.25(32n^2+14n)$	
	192	1452	1545	6 %
	256	2367.2	2700	12 %
	512	8184	10520	22 %
Montgomery ladder	32n	$\left(18.2n^2+60.8n\right)\times 1.1$	$35n^2+16n$	
	192	1122	1356	17 %
	256	1816.32	2368	23 %
	512	6195.2	9216	32.5 %
Montg. ladder (a small)	32n	$\left(16.8n^2+55.2n\right)\times 1.1$	$31n^2+14n$	
	192	1029.6	1200	14 %
	256	1668.48	2096	20 %
	512	5702.4	8160	30 %

Table 4: Cost in word-products (32-bits) of one scalar multiplication iteration

are based on systolic implementations [33] or on parallel implementations [13, 37], where the given architectures are respectively in $O(n^2/log(n)^2)$ and $O(n^2)$ for the area and $O(\log(n))$ for the time. As we did not find an explicit complexity for multiplication using a $O(n)$ area architecture, we give two values for the complexity. The first one is minimal but certainly not realistic. The second one, which is not necessarily optimal, takes into account that

- each product of a number by a digit will produce two numbers (high and low parts),
- a carry-save adder will need an extra register for storing the carry and a final adder for absorbing these carries,
- 32-bit words look-up tables are not reasonable.

Then, to get an idea with ECC key size, we compare three different implementations in table 6 for the number of operations required for one step of the Montgomery scalar multiplication on an elliptic curve in Weierstrass form with a small.

In this configuration, the RNS becomes very interesting compared to the Montgomery arithmetic in terms of efficiency for a leak-resistant implementation of elliptic curve cryptosystems, even if we use our non-realistic lower bound for the comparison.

Operation	RNS	Montgomery
Multiplication	2×1.1	$n \dots 2n$
Reduction	$\left(\frac{12}{5}n + \frac{3}{5}\right) \times 1.1$	$2n \dots 3n$
One iteration of algorithm 2	$31.68n + 47.52$	$44n \dots 75n$

Table 5: Number of cycles with parallel implementations on an n word-operators structure.

$\lvert p \rvert_2$	word	RNS	Montgomery	ratio inc
192	6	237.6	$264 \dots 450$	$10\% \dots 47\%$
256	8	300.96	$352 \dots 600$	$14.5\% \dots 50\%$
512	16	554.4	$704 \dots 1200$	$21\% \dots 54\%$

Table 6: Comparison of parallel implementations

6 Practical implementation

The aim of this paper was to study the interest of combining leak-resistant arithmetic on elliptic curves and RNS representation of numbers. Even if it was not the goal of this paper, a practical implementation was necessary to validate the good results obtained. Such an implementation, based on this paper, has now been done by Guillermin in [23]. The FPGA implementation obtained is the fastest one for elliptic curves defined over non-Mersenne prime field. This proves that the work done in this paper is not only theoretically interesting and that RNS representation is promising for elliptic curve cryptography.

7 Conclusion

Combining Residue Number System and SPA-resistant arithmetic on elliptic curves, we obtain an efficient and secure implementation of elliptic curves cryptosystems on embedded devices especially if a parallel architecture is used.

Since the expensive operation in RNS is the reduction, we proposed to rewrite formulae for elliptic curve SPA-resistant arithmetic in order to minimize the number of reductions even if the number of multiplications is increased. In the case of the Montgomery ladder on elliptic curves in Weierstrass form, we obtained new formulae which are better suited to RNS representation of numbers and we explain why multiplications by one of the coefficients of the curve can be neglected in most cases.

We also give an in-depth analysis of the complexity of the Montgomery reduction. We thus realized that some improvements could be made to obtain a final complexity of $\frac{7}{5}n^2 + \frac{8}{5}n$ for a n-word number.

It is clear that, without the results we obtained in these two directions, the combination of RNS arithmetic and elliptic curves will be possible but less convincing. Thus, we theoretically obtain an efficient leak-resistant arithmetic especially for high security levels and in the case of the Montgomery ladder on elliptic curves in Weierstrass form. Moreover these results have been recently successfully validated by a very efficient FPGA implementation [23]

Our approach is particularly interesting from a hardware standpoint since the RNS representation of numbers has many advantages (easy to implement and parallelize, flexibility). It is also very attractive in the case of a dedicated parallel architecture.

References

[1] Bajard, J.C., Didier, L.S., Kornerup, P.: A RNS Montgomery's Modular Multiplication. IEEE Transactions on Computers, volume 47, no. 7, July 1998.

[2] Bajard, J.C., Didier, L.S., Kornerup, P.: Modular multiplication and base extension in residue number systems. 15th IEEE Symposium on Computer Arithmetic, IEEE Computer Society Press (2001) 59–65.

[3] Bajard, J.C., Duquesne, S., Ercegovac M. and Meloni N.: Residue systems efficiency for modular products summation: Application to Elliptic Curves Cryptography, in Advanced Signal Processing Algorithms, Architectures, and Implementations XVI, part of the SPIE Optics & Photonics 2006 Symposium. August 2006 San Diego, USA.

[4] Bajard, J.C., Imbert, L.: A full RNS implementation of RSA. IEEE Transactions on Computers **53:6** (2004) 769–774.

[5] Bajard, J.C., Imbert, L., Liardet, P.Y., Teglia, Y.: Leak resistant arithmetic. CHES 2004, LNCS **3156** 59–65.

[6] Bajard, J.C., Kaihara, M., Plantard Th.: Selected RNS Bases for Modular Multiplication in Proceedings of the 19th IEEE symposium on Computer Arithmetic (ARITH 19) June 2009, Portland, USA.

[7] Bajard, J.C., Meloni, N., Plantard, T.: Efficient RNS bases for Cryptography IMACS'05, Applied Mathematics and Simulation, (2005).

[8] Billet, O., Joye, M.: The Jacobi Model of an Elliptic Curve and Side-Channel Analysis. Applied Algebra, Algorithms and Error-Correcting Codes, LNCS **2643** (2003) 34–42.

[9] Bosselaers, A., Govaerts, R., Vandewalle. J.: Comparison of the three modular reduction functions LNCS **773** (1994) 175–186.

[10] Brent, R.P., Kung, H.T.: The Area- Time Complexity of Binary Multiplication. Journal of the Association for Computing Machinery, Vol 28, No 3, July 1981, 521–534.

[11] Brier, E., Joye, M.: Weierstrass Elliptic Curves and Side-Channel Attacks. Public Key Cryptography, LNCS **2274** (2002) 335–345.

[12] Brier, E., Joye, M.: Fast Point Multiplication on Elliptic Curves Trough Isogenies, Applied Algebra, Algorithms and Error-Correcting Codes, Lecture Notes in Comput. Sci., vol.2643, Springer, Berlin, 2003, pp. 43–50.

[13] Bunimov, V., Schimmler, M.: Efficient Parallel Multiplication Algorithm for Large Integers Euro-Par 2003, International Conference on Parallel and Distributed Computing (2003) 923–928.

[14] Chung, J., Hasan, A.: More generalized mersenne numbers. SAC 2003, LNCS **3006** (2003) 335–347

[15] Ciet, M., Neve, M., Peeters, E., Quisquater, J.J.: Parallel FPGA implementation of RSA with residue number systems– can side-channel threats be avoided? 46th IEEE International Midwest Symposium on Circuits and Systems (2003).

[16] Cohen, H., Frey, G.: Handbook of elliptic and hyperelliptic curve cryptography. Discrete Math. Appl., Chapman & Hall/CRC (2006).

[17] Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. CHES'99, LNCS **1717** (1999) 292–302.

[18] Duquesne, S.: Improving the Arithmetic of Elliptic Curve in Jacobi Model. Information Processing Letters **104:3** (2007) 101–105.

[19] Fischer, W., Giraud, C., Knudsen, E.W., Seifert, J. P.: Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against Non-Differential Side-Channel Attacks. Preprint.

[20] Flynn, E.V.: An explicit theory of heights. Trans. Amer. Math. Soc. **347:8** (1995) 3003–3015.

[21] Svoboda, A. and Valach, M.: Operational Circuits. Stroje na Zpracovani Informaci, Sbornik III, Nakl. CSAV, Prague, (1955) 247-295.

[22] Garner, H.L.: The residue number system. IRE Transactions on Electronic Computers, EL **8:6** (1959) 140–147.

[23] Guillermin, N.: A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$. CHESS 2010, LNCS (2010)

[24] Izu, T., Takagi, T.: A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks. Public Key Cryptography, LNCS **2274** (2002) 280–296.

[25] Joye, M., Quisquater, J.J.: Hessian Elliptic Curves and Side-Channel Attacks. CHES 2001, LNCS **2162** 402–410.

[26] Knuth, D.: Seminumerical Algorithms. The Art of Computer Programming, vol. 2. Addison-Wesley (1981).

[27] Kocher, P.C.: Timing attacks on implementations of DH, RSA, DSS and other systems. CRYPTO'96, LNCS **1109** (1996) 104–113.

[28] Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. CRYPTO'99, LNCS **1666** (1999) 388–397.

[29] Liardet, P. Y., Smart, N.: Preventing SPA/DPA in ECC systems using the Jacobi form. CHESS 2001, LNCS **2162** 391–401.

[30] Montgomery, P.L.: Modular multiplication without trial division. Math. Comp. **44:170** (1985) 519–521.

[31] Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comp. **48:177** (1987) 243–164

[32] Okeya, O., Sakurai, K.: Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the y-Coordinate on a Montgomery-Form Elliptic Curve. Cryptographic Hardware and Embedded Systems, LNCS **2162** (2001) 126–141.

[33] G. Orlando and C. Paar. A scalable GF(p) elliptic curve processor architecture for programmable hardware. In Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001)

[34] Posch, K.C., Posch, R.: Modulo reduction in residue number systems. IEEE Transaction on Parallel and Distributed Systems **6:5** (1995) 449–454.

[35] Quisquater, J.J., Samyde, D.: ElectroMagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards. e-smart 2001, LNCS **2140** (2001) 200–210.

[36] Shenoy, A.P., Kumaresan, R.: Fast base extension using a redundant modulus in RNS. IEEE Transactions on Computer **38:2** (1989) 292–296.

[37] Sanu, M.O., Swartzlander, E.E., Chase, C.M.: Parallel Montgomery Multipliers. 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04) (2004) 63–72.

[38] Solinas, J.: Generalized Mersenne numbers. Research Report CORR-99-39, Center for Applied Cryptographic Research, University of Waterloo (1999)

[39] Szabo, N.S., Tanaka, R.I.: Residue Arithmetic and its Applications to Computer Technology. McGraw-Hill (1967)

Annexes

16 **coprimes of** 32 **bits with** $c_i = 2^{t_i} \pm 1 < 2^{12}$

$m_1 = 10000000000000000000000000000000$
$m_2 = 11111111111111111111111111111111$
$m_3 = 11111111111111111111111111111101$
$m_4 = 11111111111111111111111111111011$
$m_5 = 11111111111111111111111111110111$
$m_6 = 11111111111111111111111111110001$
$m_7 = 11111111111111111111111111101111$
$m_8 = 11111111111111111111111111011111$
$m_9 = 11111111111111111111111111000001$
$m_{10} = 11111111111111111111111110111111$
$m_{11} = 11111111111111111111111101111111$
$m_{12} = 11111111111111111111111100000001$
$m_{13} = 11111111111111111111110111111111$
$m_{14} = 11111111111111111111101111111111$
$m_{15} = 11111111111111111111011111111111$
$m_{16} = 11111111111111111111000000000001$

18 **coprimes of** 32 **bits with** $c_i = 2^{t_i} \pm 1 < 2^{15}$

$m_1 = 10000000000000000000000000000000$
$m_2 = 11111111111111111111111111111111$
$m_3 = 11111111111111111111111111111101$
$m_4 = 11111111111111111111111111111011$
$m_5 = 11111111111111111111111111110111$
$m_6 = 11111111111111111111111111110001$
$m_7 = 11111111111111111111111111101111$
$m_8 = 11111111111111111111111111011111$
$m_9 = 11111111111111111111111111000001$
$m_{10} = 11111111111111111111111110111111$
$m_{11} = 11111111111111111111111101111111$
$m_{12} = 11111111111111111111111100000001$
$m_{13} = 11111111111111111111110111111111$
$m_{14} = 11111111111111111111101111111111$
$m_{15} = 11111111111111111111011111111111$
$m_{16} = 11111111111111111111000000000001$
$m_{17} = 11111111111111111110111111111111$
$m_{18} = 11111111111111111101111111111111$

64 **coprimes of 32 bits with** $c_i = 2^{t_i} \pm 2^{s_i} \pm 1 < 2^{15}$

$$
\begin{aligned}
m_{1} &= 10000000000000000000000000000000 & m_{33} &= 11111111111111111111110001000001\\
m_{2} &= 11111111111111111111111111111111 & m_{34} &= 11111111111111111111110000010001\\
m_{3} &= 11111111111111111111111111111101 & m_{35} &= 11111111111111111111110000000111\\
m_{4} &= 11111111111111111111111111111011 & m_{36} &= 11111111111111111111101111111111\\
m_{5} &= 11111111111111111111111111110111 & m_{37} &= 11111111111111111111101000000001\\
m_{6} &= 11111111111111111111111111110001 & m_{38} &= 11111111111111111111100011111111\\
m_{7} &= 11111111111111111111111111101111 & m_{39} &= 11111111111111111111100010000001\\
m_{8} &= 11111111111111111111111111101001 & m_{40} &= 11111111111111111111100001000001\\
m_{9} &= 11111111111111111111111111100101 & m_{41} &= 11111111111111111111100000100001\\
m_{10} &= 11111111111111111111111111100011 & m_{42} &= 11111111111111111111100000001111\\
m_{11} &= 11111111111111111111111111011111 & m_{43} &= 11111111111111111111100000000011\\
m_{12} &= 11111111111111111111111111010001 & m_{44} &= 11111111111111111111011111111111\\
m_{13} &= 11111111111111111111111111000001 & m_{45} &= 11111111111111111111011111100001\\
m_{14} &= 11111111111111111111111110111111 & m_{46} &= 11111111111111111111000100000001\\
m_{15} &= 11111111111111111111111110100001 & m_{47} &= 11111111111111111111000001111111\\
m_{16} &= 11111111111111111111111110001001 & m_{48} &= 11111111111111111111000000000111\\
m_{17} &= 11111111111111111111111101111111 & m_{49} &= 11111111111111111110111111111111\\
m_{18} &= 11111111111111111111111101111001 & m_{50} &= 11111111111111111110111111111001\\
m_{19} &= 11111111111111111111111101110001 & m_{51} &= 11111111111111111110111110000001\\
m_{20} &= 11111111111111111111111101100001 & m_{52} &= 11111111111111111110111000000001\\
m_{21} &= 11111111111111111111111100011111 & m_{53} &= 11111111111111111110010000000001\\
m_{22} &= 11111111111111111111111100000001 & m_{54} &= 11111111111111111110000011111111\\
m_{23} &= 11111111111111111111111010000001 & m_{55} &= 11111111111111111110000010000001\\
m_{24} &= 11111111111111111111111000111111 & m_{56} &= 11111111111111111110000000111111\\
m_{25} &= 11111111111111111111111000100001 & m_{57} &= 11111111111111111110000000001001\\
m_{26} &= 11111111111111111111111000000101 & m_{58} &= 11111111111111111110000000000011\\
m_{27} &= 11111111111111111111111000000011 & m_{59} &= 11111111111111111101111111111111\\
m_{28} &= 11111111111111111111110111111111 & m_{60} &= 11111111111111111101111111000001\\
m_{29} &= 11111111111111111111110111110001 & m_{61} &= 11111111111111111101111100000001\\
m_{30} &= 11111111111111111111110111000001 & m_{62} &= 11111111111111111101100000000001\\
m_{31} &= 11111111111111111111110110000001 & m_{63} &= 11111111111111111101000000000001\\
m_{32} &= 11111111111111111111110001111111 & m_{64} &= 11111111111111111100000100000001
\end{aligned}
$$