New Multibase Non-Adjacent Form Scalar Multiplication and its Application to Elliptic Curve Cryptosystems (extended version)

Patrick Longa, and Ali Miri

Abstract. In this paper we present a new method for scalar multiplication that uses a generic multibase representation to reduce the number of required operations. Further, a multibase NAF-like algorithm that efficiently converts numbers to such representation without impacting memory or speed performance is developed and showed to be sublinear in terms of the number of nonzero terms. Additional representation reductions are discussed with the introduction of window-based variants that use an extended set of precomputations. To realize the proposed multibase scalar multiplication with or without precomputations in the setting of Elliptic Curve Cryptosystems (ECC) over prime fields, we also present a methodology to derive fast composite operations such as tripling or quintupling of a point that require less memory than previous point formulae. Point operations are then protected against simple side-channel attacks using a highly efficient atomic structure. Extensive testing is carried out to show that our multibase scalar multiplication is the fastest method to date in the setting of ECC and exhibits a small footprint, which makes it ideal for implementation on constrained devices.

Keywords: Scalar multiplication, multibase representation, multibase non-adjacent form, elliptic curve cryptosystem, composite operation, side-channel attacks.

1 Introduction

Scalar multiplication is the central and most time-demanding operation in many public-key curve-based systems such as Elliptic Curve (ECC), Hyperelliptic Curve (HECC) and Pairing-based cryptosystems. Its algorithmic and computational structure has been the focus of extensive research in recent years in a growing effort to reduce its time execution and power/memory requirements and, thus, make the corresponding system suitable for implementation in the myriad of new applications using ubiquitous devices such as PDAs, smartcards, cellphones, RFID tags and wireless sensor networks. Algorithms to implement this operation have traditionally relied on the binary expansion of numbers (e.g., Non-Adjacent Form (NAF) and Window-*w* Non-Adjacent Form (*w*NAF)) since the latter directly translates to a successive execution of the basic point operations, namely doubling and addition. However, the development of more complex operations has permitted the development of scalar multiplications using radices different than 2 (radix-*r* NAF [TYW04], generalized NAF [JY02]) or combinations of different bases (ternary/binary [CJL+06], Double-Base (DB) [DIM05],[DIM07], Triple-Base (TB) [MD07] methods) that permit a reduction in the length of the scalar expansion and, consequently, if the complex operations are efficient enough in a given setting, a reduction in the time required to execute the scalar multiplication. Throughout this work, we refer to these complex operations as *composite operations*,

---

This manuscript is also available at http://patricklonga.bravehost.com/publications.html.

Patrick Longa is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada (e-mail: plonga@uwaterloo.ca). Ali Miri is with the School of Information and Technology Engineering (SITE), University of Ottawa, Ottawa, Canada (e-mail: samiri@site.uottawa.ca).

since they are typically built on top of the basic doubling and addition.

In this work, we narrow our spectrum of applications to ECC over prime fields using standard curves [IEEE]. Elliptic curve cryptography, independently introduced by Koblitz and Miller in 1985, is becoming a major player in public-key infrastructures due to its shorter key length requirement in comparison with other cryptosystems such as RSA. It has been established that 160-bit ECC gives equivalent security to 1024-bit RSA, which highlights the potential increment in computing performance and savings in power/memory that can be achieved with this cryptosystem.

In this setting, the recent development of efficient composite operations [DIM05],[MD07],[LM07b] has permitted algorithms to surpass the performance of traditional scalar multiplications based solely on radix 2 by using DB and TB methods that mix bases 2 and 3, and 2, 3 and 5, respectively, to build shorter scalar expansions. Although the use of DB and TB has already been shown to reduce the computing cost of the scalar multiplication following a sublinear complexity of less than $(\log k / \log\log k)$ additions [DIM05],[DI06],[MD07], where k is the secret scalar, conversion to DB and TB has some shortcomings.

First, conversion of any scalar k to DB or TB representations can be relatively slow since it is based on an exhaustive search of all possible combinations of powers of (2,3) or (2,3,5), respectively (referred to as "Greedy" algorithm by [DIM05]). Some increase in speed can be obtained by applying a smart search, such as the method proposed in [DI06] using lexicographic-ordered tables. However, this method requires extra memory to store such tables, which can be prohibitive in some constrained environments. Refer to Table 1 in [DI06] for precise amounts of extra memory that would be required. Notice that if one stores only part of the precomputed data, then some delays potentially could be introduced during DB/TB conversions.

Second, the efficiency of DB/TB-based scalar multiplications depend heavily on the maximum bounds pre-established for exponents in bases 2, 3 and 5. Heuristic approximations have been proposed to estimate the values that would yield short expansions [DIM05],[DIM07]. However, in ECC scalar multiplication, the density of nonzero terms (i.e., the number of additions) is not the only parameter to take into account to achieve cheaper computations. In fact, since additional point operations (namely, tripling of a point $3P$, and quintupling of a point $5P$, for any point P on the curve) are included in the expansion, the right balance between point operations should be considered to reach a minimum computing cost. The complexity of this estimate is greatly increased if one needs to consider settings with different cost ratios among point operations.

Finally, there is no theoretical bound for the length or density of the expansion. This fact increases the difficulty of finding optimal values for the maximum bounds of exponents in the DB/TB-based scalar multiplication.

In this work, we generalize the idea of combining several radices to represent numbers and, then, solve the problem of finding short multibase representations in an efficient manner by proposing a multibase NAF-like representation (denoted by *mb*NAF) with no restriction in the number or selection of bases, that overcomes all the previous problems found in DB/TB. This is, to our knowledge, the first effort in the area to apply a generic multibase representation of the scalar k to reduce computational costs. Also, we show that our methods are sublinear with complexity of approximately $(\log k / \log\log k)$ point additions.

To realize the scalar multiplication using the new multibase NAF representation in the ECC setting, in the first part of this work we introduce a generic methodology to build composite operations of the form

dP for standard ECC curves over prime fields, where d is an odd prime ≥ 3 and P is a point on the elliptic curve. Additionally, we present an optimized quintupling formula that achieves the lowest cost reported in the literature. Further, we protect our operations against Simple-Side Channel Attacks (SSCA) using a variation of the efficient atomic structure presented in [LM07a] based on side-channel atomicity. We show that our unprotected and SSCA-protected composite operations reduce costs in comparison with previous formulae, and most remarkably, present reduced memory requirements. In particular, new tripling ($3P$), quintupling ($5P$) and septupling ($7P$) operations are made efficient enough to implement *mb*NAF scalar multiplications using radices 3, 5 and 7 besides 2. It will turn out that our new scalar multiplication is the fastest method known to date on generic curves over prime fields that simultaneously allows a highly efficient ECC implementation with reduced footprint.

It is important to remark that, even though we are focusing on ECC over prime fields using standard curves, the proposed *mb*NAF representations are generic and can be applied to other areas, or specifically, to any cryptosystem based on scalar multiplication. For the latter, the only requirement is to have efficient formulae to compute operations with radices other than 2.

For the rest of this work, M, S and A, in italics, stand for the computing costs of field multiplication, squaring, and addition or subtraction, respectively. Similarly, to denote the costs of point operations (typically expressed in terms of field operations), we use A, D, T and Q for point addition, doubling, tripling and quintupling, respectively.

To simplify our cost analyses in the different sections, we will consider two possible scenarios:

- Software-based implementations where squaring is usually faster than multiplication. In this case, we consider $1S = 0.8M$ [BHL+01],[GAS+05],[GG03], which is widely accepted in the literature.
- Implementations on hardware platforms or when some built-in hardware is used to accelerate EC-operations (e.g., using a modular hardware multiplier [XB01]). In this case, a multiplier executes both squaring and multiplication, and consequently, the ratio S/M is fixed at 1.

Although A/M ratios vary widely from application to application, to simplify comparisons we consider a low ratio for applications where multiplications are not favoured by a fast hardware multiplier. In such cases, we use $1A \approx 0.05M$, as achieved by [Ber06]. Where this is not the case, the cost of an addition can be non-negligible. We will then use the ratio achieved in [ITT+99], $1A \approx 0.2M$.

This paper is organized as follows. In Section 2, we introduce some basic concepts about ECC and summarize, for comparison purposes, the costs of the state-of-the-art point operation formulae. In Section 3, we present our innovative methodology that relies on the special addition operation using identical z-coordinate developed by [Mel06] to yield new composite operations: tripling ($3P$), quintupling ($5P$), septupling ($7P$), and so on. Our new atomic structure to protect against SSCA is introduced in the subsequent section and applied to the development of SSCA-protected formulae. In Section 5, we present new scalar multiplications that use multiple bases to represent scalars, including a discussion about their theoretical bounds in terms of density. Results of extensive tests with 160-bit random numbers showing the superiority of our scalar multiplication methods in SSCA-protected and unprotected scenarios are presented in Section 6. We end with some conclusions about the work presented.

2 Preliminaries

In this section we summarize basic notions about ECC. The reader is referred to [HMV04],[ACD+05] for further details. In general, ECC can be defined over different finite fields. Here we work with a prime field containing p elements, generically denoted by $\mathbb{F}_p$, where p is a large prime. In this case, the simplified Weierstrass equation is as follows

$$E: \quad y^2 = x^3 + ax + b\,, \tag{1}$$

where: a, $b \in \mathbb{F}_p$ and $\Delta = 4a^3 + 27b^2 \neq 0$ (Δ is the discriminant of E).

ECC computations are performed over the abelian group constituted by the set of pairs (x, y) that solves (1) and the point at infinity O. The latter acts as the identity for the Group Law and can be geometrically defined as the point "lying far out on the y-axis such that any line $x = c$, for some constant c, parallel to the y-axis passes through it" [ACD+05].

If we denote the elliptic curve over $\mathbb{F}_p$ by $E(\mathbb{F}_p)$, the central operation in ECC, known as scalar or point multiplication, can be represented by kP, where P is a point in $E(\mathbb{F}_p)$ and k is the secret scalar. In the following, we assume that $\#E(\mathbb{F}_p) = h.q$ (q prime and $h << q$, and hence $p \approx q$) and P and Q are points of order q [HMV04]. If k is a scalar randomly chosen in the range [1, $q - 1$] and $n = \log_2 q$, then the average length of k is $l \approx n - 1$. We refer indistinctly as density or Hamming weight to the number of nonzero elements of a given scalar representation. In particular, for scalar multiplication, the latter directly translates to the number of point additions required to compute kP using such representation.

In the following, we first summarize relevant methods for scalar multiplication, and then the point operations that allow the realization of these methods in ECC.

2.1 Previous Scalar Multiplication Methods

Considerable effort has been invested during the past two decades to developing fast algorithms to compute kP. The main idea is to use short representations for k that translate to efficient scalar multiplication methods. To this end, radix 2 or binary representations have been traditionally used because the expansion translates directly into a given sequence of point doubling ($2P$) and addition ($P + Q$), the basic ECC point operations. For instance, the binary method using elements $\{0,1\}$ is known to achieve a density of 1/2, which fixes the cost of the scalar multiplication to about $(n-1)\mathrm{D} + (n/2)\mathrm{A}$, where D and A represent the computing cost of doubling and addition, respectively. The density of the binary expansion can be effectively reduced with a signed representation that uses elements in the set {-1,0,1}, taking advantage of the fact that the cost of computing inverses of points (e.g., $-P$) in additive groups as $E(\mathbb{F}_p)$ is negligible. Among different signed radix2-based representations, NAF is a canonical representation with the fewest number of nonzero digits for any scalar k. The NAF representation of k, denoted by NAF(k), contains at most *one* nonzero digit among any two successive digits.

Algorithm 2.1 computes the NAF of a scalar k using $w = 2$. Similarly, Algorithm 2.2 uses NAF for scalar multiplication when $w = 2$ [HMV04].

Algorithm 2.1 Computing the *w*NAF (NAF) of a positive integer

INPUT: window w ($w = 2$ for NAF), scalar k

OUTPUT: $\mathrm{NAF}_w(k)$

1. $I = 0$
2. While $k \geq 1$ do
   2.1. If k is odd, then $k_i = k \bmod 2^w$, $k = k - k_i$
   2.2. Else $k_i = 0$
   2.3. $k = k/2$, $I = I + 1$
3. Return $(k_{l-1}, \ldots, k_1, k_0)_{\mathrm{NAF}_w}$

Algorithm 2.2 *w*NAF (NAF) method for scalar multiplication

INPUT: Window w ($w = 2$ for NAF), scalar $k = (k_{l-1}, \ldots, k_1, k_0)_{\mathrm{NAF}_w}$, $P \in E(\mathbb{F}_p)$

OUTPUT: kP

1. Compute $P_i = iP$ for $i \in \{1,\ 3,\ 5,\ \ldots\ ,\ (2^{w-1} - 1)\}$
2. $Q = O$
3. For $i = l - 1$ downto 0 do
   3.1. $Q = 2Q$
   3.2. If $k_i \neq 0$, then:
      If $k_i > 0$, then $Q = Q + P_{k_i}$
      Else $Q = Q - P_{k_i}$
4. Return (Q)

The function *mods* in Algorithm 2.1 represents the next computation ($w = 2$ for NAF):

$$\begin{cases} \text{If } k \bmod 2^w \geq 2^w/2, \text{ then:} \\ \quad k_i = (k \bmod 2^w) - 2^w \\ \text{Else,} \\ \quad k_i = k \bmod 2^w \end{cases} \tag{2}$$

The expected number of doublings and additions using Algorithm 2.2 ($w = 2$) is approximately ($n - 1$) and $n/3$, respectively. Thus, the cost of the NAF method is:

$$(n-1)\mathrm{D} + \left(\frac{n}{3}\right)\mathrm{A}. \tag{3}$$

If there is memory available, one can make use of precomputations to reduce the computing time for scalar multiplication. In such case, *w*NAF is the natural expansion of NAF. It basically exploits the availability of precomputed values to "insert" windows of width w, which permits the consecutive

execution of several doublings to reduce the density of the expansion. The *w*NAF representation of *k*, denoted by $\mathrm{NAF}_w(k)$, contains at most *one* nonzero digit among any *w* successive digits.

Algorithm 2.1 computes the *w*NAF representation of a scalar *k* when $w > 2$. Similarly, Algorithm 2.2 uses *w*NAF for scalar multiplication with $w > 2$. It can be observed directly from these algorithms that *w*NAF is simply a generalization of NAF to any window value, and that NAF is the only variant in such generalization that does not require precomputations P_i (hereafter we refer to non-trivial points not including $\{O, P\}$ as precomputed points).

The average density of nonzero digits in *w*NAF for a window of width *w* is $\frac{1}{w+1}$, and the number of required precomputed points is $(2^{w-2}-1)$. The cost of the *w*NAF method is approximately:

$$(n-1)\mathrm{D} + \left(\frac{n}{w+1}\right)\mathrm{A}. \tag{4}$$

Recently, new methods using other radices beside 2 were introduced. [DIM05] proposed to representing scalar *k* using mixed powers of 2 and 3, as follows:

$$k = \sum_{i=1}^{m} k_i 2^{b_i} 3^{c_i}, \tag{5}$$

where *m* is the length of the expansion, k_i is the sign (i.e., $k_i \in \{-1,1\}$), and b_i and c_i form decreasing sequences $b_{\max} \geq b_1 \geq b_2 \geq \ldots \geq b_m \geq 0$ and $c_{\max} \geq c_1 \geq c_2 \geq \ldots \geq c_m \geq 0$, respectively.

This Double-Base (DB) representation is highly sparse and, consequently, permits to reduce the Hamming weight of the expansion for the scalar. With the introduction of efficient tripling formulae [DIM05],[DIK06], these representations using ternary bases greatly reduce the execution time of scalar multiplication.

Later, [DI06] extended this approach, called *Extended DB*, to applications that can afford precomputations. In this case, k_i in (5) is allowed to have any value from a set of precomputed digits K_i, where the elements are prime numbers other than 3. For instance, $K_i = \{\pm 1, \pm 5, \pm 7, \pm 11\}$.

Finding short expansions using $\{2^{b_i}3^{c_i}\}$-terms has been defined as a difficult problem on its own. [DIM05] proposed to solve that problem by establishing "efficient" maximum bounds $b_{\max}$ and $c_{\max}$ for the powers of 2 and 3, respectively, and then executing an exhaustive search for closest terms $2^{b_i}3^{c_i}$ (referred to as "Greedy" algorithm).

An extension of the previous work (referred to as Triple-Base or TB) was presented in [MD07] by adding radix 5 to the previous approach. It is also based on a "Greedy" algorithm to find a scalar representation, and hence, exhibits the same shortcomings as DB.

2.2 Previous Point Operation Formulas

Since the representation of points on the curve *E* with two coordinates (x, y), namely affine coordinates, introduces expensive field inversions into the computation of point operations, in this work we will only consider the case of projective coordinates (X, Y, Z) to represent points. The latter solves the aforementioned problem by adding the third coordinate *Z* to replace inversions with a few other field operations. In particular, we focus on Jacobian coordinates, a special case of projective coordinates that has yielded very efficient point formulae [Elm06],[HMV04].

The foundation of these inversion-free coordinate systems can be explained by the concept of *equivalence classes*, which are defined in the following in the context of Jacobian coordinates.

Given a prime field $\mathbb{F}_p$, there is an equivalence relation ≡ among nonzero triplets over $\mathbb{F}_p$, such that [ACD+05]:

$$(X_1, Y_1, Z_1) \equiv (X_2, Y_2, Z_2) \Leftrightarrow X_1 = \lambda^2 X_2,\ Y_1 = \lambda^3 Y_2 \text{ and } Z_1 = \lambda Z_2, \text{ for some } \lambda \in \mathbb{F}_p^*.$$

Thus, the equivalence class of a *(Jacobian) projective point*, denoted by (X : Y : Z), is:

$$(X : Y : Z) = \{(\lambda^2 X,\ \lambda^3 Y,\ \lambda Z) : \lambda \in \mathbb{F}_p^*\}. \tag{6}$$

It is important to remark that any (X, Y, Z) in the equivalence class (6) can be used as a representative of a given (Jacobian) projective point.

In Table 1, we summarize computational and memory costs of previous formulae in Jacobian coordinates by different authors (referred to as previous operations) and improvements that we introduced in [LM07a] (referred to as fast operations). It is important to note that it has been suggested that the parameter a in equation (1) is fixed at −3 for efficiency purposes [IEEE], which has been shown not to impose significant restrictions to the cryptosystem [BJ03]. In this work, we will refer to the *special case* when $a = -3$, and the *general case* when the parameter is not fixed and can be any value in the field. For complete details about previous and fast point formulae the reader is referred to [LM07a]. Also, we have gathered the most efficient point operation formulas in Jacobian coordinates in an online database accessible at *http://patricklonga.bravehost.com/jacobian.html*.

To simplify comparisons, it is widely accepted that the computational cost of a point operation be expressed in terms of its required number of field multiplications and squarings. Regarding memory, the costs in Table 1 are expressed by the number of registers needed to store input/output point coordinates and intermediate computations (excepting mixed addition, these numbers correspond to the special case $a = -3$). Note that in this work we have developed very efficient memory allocation using low resources for the fast formulae by [LM07a]. The reader is referred to Appendices A1, A2 and B for complete details.

In addition, we present a new quintupling formula, detailed in Appendix C, which has been further accelerated by applying the technique of replacing multiplications by squarings used in [LM07a] to derive fast operations. This quintupling introduces significant reductions in comparison with the best previous formula due to [MD07]. For instance, if $1S = 0.8M$, the new quintupling costs $21.2M$ and $19.8M$ in contrast to $23M$ and $21.4M$ [MD07] for the general and special cases, respectively.

Operation costs in Table 1 will be used as a reference point for comparison with our new composite operations (Section 3). The improved fast operations with reduced memory requirements will be used to boost the performance of the *mb*NAF scalar multiplication (Section 5).

3 Composite Operations dP

If we add two points $P = (X_1, Y_1, Z)$ and $Q = (X_2, Y_2, Z)$ containing the same z-coordinate in Jacobian coordinates, the result $P + Q = (X_3, Y_3, Z_3)$ can be obtained as follows [Mel06]:

Table 1. Cost of point operations

Operation	general	$a = -3$	# reg.
Fast doubling [LM07a]	$2M + 8S$	$3M + 5S$	6
Previous doubling	$4M + 6S$	$4M + 4S$	6 (a)
Fast general addition [LM07a]	$11M + 5S$	-	N/A
Previous general addition	$12M + 4S$	-	N/A
Fast mixed addition [LM07a]	$7M + 4S$	-	6
Previous mixed addition	$8M + 3S$	-	7 (a)
Fast tripling [LM07a]	$6M + 10S$	$7M + 7S$	8
Previous tripling [DIM05]	$10M + 6S$	-	10 (b)
Fast quintupling (this work)	$10M + 14S$	$11M + 11S$	10
Previous quintupling [MD07]	$15M + 10S$	$15M + 8S$	N/A

(a) From [HMV04, pp. 91-92]
(b) SSCA-protected tripling from [DIM05],[DIM07]

$$
\begin{aligned}
X_3 &= \left(Y_2 - Y_1\right)^2 - \left(X_2 - X_1\right)^3 - 2X_1\left(X_2 - X_1\right)^2. \\
Y_3 &= \left(Y_2 - Y_1\right)\left(X_1\left(X_2 - X_1\right)^2 - X_3\right) - Y_1\left(X_2 - X_1\right)^3. \\
Z_3 &= Z\left(X_2 - X_1\right).
\end{aligned} \tag{7}
$$

The cost of formula (7) is only $5M + 2S$, which represents a significant reduction in comparison with a traditional mixed Jacobian-affine addition ($8M + 3S$), or even in comparison with the fast mixed addition ($7M + 4S$) (see Table 1). Obviously, it is not possible to directly replace traditional additions with this more efficient operation since it is expected that point additions are performed over operands with different z coordinates during scalar multiplication. Nevertheless, in [LM07b] we noticed that this special operation (referred to as addition with identical z-coordinate) can be used to generate composite operations of the form $dP+Q$ when using generic scalar multiplication methods over prime fields.

In the following, we extend the work presented in [LM07b] and introduce efficient composite operations of the form dP, where d is an odd prime integer ≥ 3 and P is any point on the elliptic curve $E(\mathbb{F}_p)$, by exploiting the advantages of this special addition (7).

3.1 Generalization to Composite Operations dP

We propose the following sequence to compute operations of the form dP, for any odd prime $d \geq 3$:

$$dP = 2P + (\ \ldots + (2P + (2P + (2P + P)))\ \ldots\ ), \tag{8}$$

which is executed backwards using only one doubling and $\left(\frac{d-1}{2}\right)$ additions with identical z-coordinate (7).

First, for the general case (random a), the doubling $2P = 2\left(X_1, Y_1, Z_1\right) = \left(X_2, Y_2, Z_2\right)$ is computed using the fast formulae developed in [LM07a], which has a cost of only $2M + 8S$:

$$
\begin{aligned}
X_2 &= \left(3X_1^2 + aZ_1^4\right)^2 - 4\left[\left(X_1 + Y_1^2\right)^2 - X_1^2 - Y_1^4\right]. \\
Y_2 &= \left(3X_1^2 + aZ_1^4\right)\left(2\left[\left(X_1 + Y_1^2\right)^2 - X_1^2 - Y_1^4\right] - X_2\right) - 8Y_1^4.
\end{aligned}
$$

$$Z_2 = \left(Y_1 + Z_1\right)^2 - Y_1^2 - Z_1^2 . \tag{9}$$

If we consider $a = -3$, then the doubling $2P$ is instead performed with the formulae due to [Ber01] and later revisited in [LM07a]:

$$\begin{aligned} X_2 &= \left[3\left(X_1 + Z_1^2\right)\left(X_1 - Z_1^2\right)\right]^2 - 8X_1Y_1^2 . \\ Y_2 &= \left[3\left(X_1 + Z_1^2\right)\left(X_1 - Z_1^2\right)\right]\left(4X_1Y_1^2 - X_2\right) - 8Y_1^4 . \\ Z_2 &= \left(Y_1 + Z_1\right)^2 - Y_1^2 - Z_1^2 . \end{aligned} \tag{10}$$

The cost of (10) is only $3M + 5S$. If we now fix $\lambda = 2Y_1$ in the equivalence class (6) to have the following new representation for P: $\left(X_1^{(1)}, Y_1^{(1)}, Z_1^{(1)}\right) = \left(X_1\left(4Y_1^2\right), Y_1\left(8Y_1^3\right), Z_1\left(2Y_1\right)\right) \equiv \left(X_1, Y_1, Z_1\right)$, we can use (7) for the addition of $2P$ and P following (8) since $\left(X_1^{(1)}, Y_1^{(1)}, Z_1^{(1)}\right)$, which is equivalent to the original point $P = \left(X_1, Y_1, Z_1\right)$, has the same z-coordinate as $\left(X_2, Y_2, Z_2\right)$. Remarkably, computation of the equivalent point $\left(X_1^{(1)}, Y_1^{(1)}, Z_1^{(1)}\right)$ does not introduce any extra cost because its coordinates have already been computed in (9) or (10).

Thus, we obtain the tripling of a point $3P = 2P + P = \left(X_2, Y_2, Z_2\right) + \left(X_1^{(1)}, Y_1^{(1)}, Z_1^{(1)}\right) = \left(X_3, Y_3, Z_3\right)$ with the following:

$$\begin{aligned} X_3 &= \left(Y_1^{(1)} - Y_2\right)^2 - \left(X_1^{(1)} - X_2\right)^3 - 2X_2\left(X_1^{(1)} - X_2\right)^2 . \\ Y_3 &= \left(Y_1^{(1)} - Y_2\right)\left(X_2\left(X_1^{(1)} - X_2\right)^2 - X_3\right) - Y_2\left(X_1^{(1)} - X_2\right)^3 . \\ Z_3 &= Z\left(X_1^{(1)} - X_2\right) . \end{aligned} \tag{11}$$

By fixing $\lambda = X_1^{(1)} - X_2$ in (6), point $2P = \left(X_2, Y_2, Z_2\right)$ becomes equivalent to $\left(X_2\left(X_1^{(1)} - X_2\right)^2, Y_2\left(X_1^{(1)} - X_2\right)^3, Z_2\left(X_1^{(1)} - X_2\right)\right)$, which has the same z-coordinate as $\left(X_3, Y_3, Z_3\right)$ in formula (11) and, thus, allows us to perform the next addition $2P + 3P$ in (8) using the special addition (7) to obtain the quintupling of a point. We can repeat the same procedure for extra additions with $2P$ and yield $7P$, $9P$, $11P$, and so on. In fact, we observe that every extra addition to $3P$ in (8) adjusts to the next generic formulae for $j = 1$ to $(d-3)/2$ having d as an odd prime > 3:

$$2P + \left(\underbrace{2P + \ldots + 2P}_{(j-1)\ terms} + (3P)\right) = \left(X_2^{(j)}, Y_2^{(j)}, Z_2^{(j)}\right) + \left(X_{j+2}, Y_{j+2}, Z_{j+2}\right) = \left(X_{j+3}, Y_{j+3}, Z_{j+3}\right):$$

$$\begin{aligned} X_{j+3} &= \left(Y_{j+2} - Y_2^{(j)}\right)^2 - \left(X_{j+2} - X_2^{(j)}\right)^3 - 2X_2^{(j)}\left(X_{j+2} - X_2^{(j)}\right)^2 , \\ Y_{j+3} &= \left(Y_{j+2} - Y_2^{(j)}\right)\left(X_2^{(j)}\left(X_{j+2} - X_2^{(j)}\right)^2 - X_{j+3}\right) - Y_2^{(j)}\left(X_{j+2} - X_2^{(j)}\right)^3 , \\ Z_{j+3} &= Z_2^{(j)}\left(X_{j+2} - X_2^{(j)}\right) , \end{aligned} \tag{12}$$

where $\left(X_2^{(j)}, Y_2^{(j)}, Z_2^{(j)}\right)$ denotes the equivalent point to $2P$ for the j^{th} addition. As we can see in (12), it holds that one always gets an equivalent point to $2P$ for the following addition by fixing:

$\left(X_2^{(j+1)}, Y_2^{(j+1)}, Z_2^{(j+1)}\right) = \left(X_2^{(j)}\left(X_{j+2} - X_2^{(j)}\right)^2, Y_2^{(j)}\left(X_{j+2} - X_2^{(j)}\right)^3, Z_2^{(j)}\left(X_{j+2} - X_2^{(j)}\right)\right)$, which is

equivalent to $\left(X_2^{(j)}, Y_2^{(j)}, Z_2^{(j)}\right)$ according to (6), and has the same z coordinate as (12).

The cost of our methodology (8) is given by:

$$1\mathrm{D} + \left(\frac{d-1}{2}\right)\mathrm{A'}, \tag{13}$$

where $d \geq 3$ is an odd prime integer for efficiency reasons, and D and A' denote the cost of doubling (as given by (9) or (10)) and addition with identical z-coordinate, respectively.

For instance, the cost of the quintupling ($d = 5$) is estimated as $(2M + 8S) + 2(5M + 2S) = 12M + 12S$, if a is randomly chosen. If $a = -3$, the cost of the new quintupling is reduced to $(3M + 5S) + 2(5M + 2S) = 13M + 9S$. Similar results are achieved for $d = 7$ and 11. Cost details for these cases are discussed in Section 3.2.

In Appendix A1, we present the pseudocode of the new composite operations of form dP, for $d \geq 3$ and $a = -3$, including memory allocation. An important feature of the new operations is that they present a reduced memory requirement fixed to only 6 registers, including registers to store input/output coordinates.

It is important to remark that the strategy presented in (8) is efficient for $d = 3, 5, 7, 11$ (tripling, quintupling, septupling and eleventupling, respectively). For computations with d greater than those values, it is more efficient to use the already efficient operations aforementioned. For instance, it is better to compute $13P$ as $2^2(3P) + P$, which requires two doublings, one tripling and one general addition. That represents a cost of $24M + 22S$ (cost has been further reduced by using fast point operations introduced in [LM07a]; see Table 1).

3.2 Performance comparison

Cost estimates of composite operations of the form dP using the proposed methodology are summarized in Table 2. Our results are compared with the best previous formulae summarized in Table 1.

We could not find formulae in the literature to compute dP for $d \geq 7$ in Jacobian coordinates over prime fields. Thus, for comparison purposes, in that case each "traditional" composite operation has been computed by combining available point operations from other works (i.e., traditional doubling and addition (Table 1; [HMV04]), tripling [DIM05] and quintupling [MD07]) in the most efficient way. Thus, $7P$ is computed by one doubling, one tripling and one general addition; and $11P$ by two doublings, one tripling and one general addition. The traditional approach has been slightly improved in the general case with a method to compute repeated doublings [IT99]. In that case, the cost of w successive doublings is expressed by $4wM + 2(2w + 1)S$. Also, we consider that performing a tripling after a doubling saves $2S$ in the general case.

In our case, composite operations are obtained following the proposed methodology (8). For tripling and quintupling, we show performance of an additional case: when these operations are computed with the fast formulas in [LM07a] (Appendix B) and Appendix C, respectively. We also include the cost of the doubling-addition operation (denoted by DA) introduced in [LM07b]. Note that we have extended the scope of the previous work and developed the memory allocation for this operation using minimum resources (see Appendix A3 for complete details).

Table 2. Cost and memory performance of composite operations

Composite Operation	Our work			Previous work		
	general	$a = -3$	# reg.	general	$a = -3$	# reg.
Tripling	(1) $6M + 10S$ (2) $7M + 10S$	$7M + 7S$ $8M + 7S$	8 6	(5) $10M + 6S$	-	10
Quintupling	(3) $10M + 14S$ (2) $12M + 12S$	$11M + 11S$ $13M + 9S$	10 6	(6) $15M + 10S$	$15M + 8S$	N/A
Septupling	(2) $17M + 14S$	$18M + 11S$	6	$26M + 14S$	$26M + 14S$	N/A
Eleventupling	(2) $27M + 18S$	$28M + 15S$	6	$30M + 18S$	$30M + 18S$	N/A
Doubling-Addition	(4) $11M + 7S$	$11M + 7S$	6	(7) $12M + 9S$	$12M + 7S$	7

(1) Fast tripling formula from [LM07a]; (2) Using methodology (8); (3) New quintupling (Appendix C);
(4) Doubling-addition from [LM07b]; (5) SSCA-protected tripling by [DIM05]; (6) Quintupling by [MD07];
(7) Using formulas by [HMV04, pp. 91-92]

After comparing the costs given in Table 2, we can conclude that our composite operations outperform the best previous formulae in both the general and special cases. It is important to note that the tripling from [LM07a] (Appendix B) and the new quintupling presented in Appendix C achieve the lowest computing costs, but with higher memory requirements in comparison with composite operations (8) presented in this section. Using the latter, it is possible to achieve a highly compact implementation with only 6 registers to perform operations.

In comparison with previous formulae, the fast tripling [LM07a] and the new tripling based on (8) are significantly superior to the one due to [DIM05]. Further, the memory requirement is also reduced. In particular, tripling in our second case is one multiplication more expensive but requires 2 fewer registers. Similarly, the new quintupling (Appendix C) and the quintupling based on (8) outperform the formula due to [DM07] not only in computing cost but also in memory usage. Significant improvements are also observed for $7P$ and $11P$. In the case of DA, we already showed its advantage in terms of costs by reducing 1 multiplication and up to 2 squarings (refer to [LM07b]). However, we have extended that advantage in this work by observing that our DA operation is not only faster but also requires less memory than the traditional execution using algorithms from [HMV04, pp. 91-92].

Our record of point operation formulas at *http://patricklonga.breavehost.com/jacobian.html* has been updated with these new composite operations.

These fast and memory-efficient composite operations will allow us to efficiently implement new *mb*NAF algorithms for the scalar multiplication presented in Section 5.

4 SSCA-Protected Point Arithmetic using Atomicity

Side-channel analysis is a real threat that has been proven to be highly useful for revealing sensitive information from crypto-operations running in electronic devices [Koc96],[KJJ99]. Through them, timing, faults, power, electromagnetic emissions and others are cleverly exploited, individually or in conjunction, to develop attacks that challenge the security of mathematically powerful cryptosystems, including ECC. In general, these attacks can be classified into two main strategies: Simple (SSCA) and

Differential (DSCA) Side-Channel Analysis attacks. Our work focuses on SSCA, which uses a single execution trace of an ECC scalar multiplication to distinguish the different point operations that are being executed and, thus, directly reveal the bits of the secret scalar.

Among several efforts to propose effective countermeasures to deal with these attacks (the reader is referred to [Ava04] for a complete survey, and to [LM07a, Section 1.1] for a discussion about the topic), side-channel atomicity [CCJ04] has been shown to give effective protection at reduced overhead. This method builds point operations on top of small homogenous blocks, known as atomic blocks, which contain the same structure of field operations and, consequently, cannot be distinguished from one another through SSCA. Since the attacker is only capable of seeing a sequence of homogenous blocks, he/she is not able to distinguish point operations. We assume here that transitions between blocks are carefully implemented to avoid any leakage that could reveal in which moment a point operation is beginning or ending.

The traditional atomic structure *M-A-N-A* (Multiplication-Addition-Negation-Addition) as proposed by [CCJ04] and later used by [DIM05],[Mis06] has three main drawbacks: parameter a in the elliptic curve equation (1) is not fixed to −3, the number of operations per atomic block is not optimal and the memory requirements (especially from previous tripling and quintupling operations) are high. To tackle these problems, a new atomic structure denoted by *M-N-A-M-N-A-A* is introduced to develop new atomic formulae for doubling, addition, doubling-addition, quintupling and the new composite operations introduced in Section 3.

4.1 New Atomic Formulae

The new atomic doubling and addition have been implemented with the proposed atomic structure using the formulae by [LM07a] detailed in Appendices A1 and A2. *Four* and *six M-N-A-M-N-A-A* blocks are required, which give total costs of $8M + 12A$ and $12M + 18A$ for the case of doubling and addition, respectively (note that the structure *M-N-A-M-N-A-A* contains $2M + 3A$ per block). The details are shown in Appendices D1 and D2. The number of registers required in both cases is limited to only 6 registers.

For the case of DA, by using the new structure we have developed atomic formulae based on the formula due to [LM07b] (see Appendix A3) with 9 blocks ($18M + 27A$) and 7 registers. The details are shown in Appendix D2.

Finally, efficient composite operations from Section 3 have also been protected with our atomic structure as shown in Appendix D3. Atomic tripling, quintupling and septupling require 8, 11 and 15 blocks, which translates to $16M + 24A$, $22M + 33A$ and $30M + 45A$, respectively. In terms of memory, the former operation requires only 6 registers, and the two latter, 7.

Table 3 summarizes costs and memory requirements for the new atomic operations and compares them with previous *M-A-N-A*-based operations. To the best of our knowledge, this is the first effort to develop atomic formulae for higher order operations such as quintupling, septupling, and so on. Hence, "traditional" composite operations shown in Table 3 are built by using the best atomic doubling, tripling and addition formulae found in the current literature. For instance, the cost of a "traditional" quintupling is estimated as $5P = 2^2P + P$ (two doublings and one general addition), and septupling as $7P = 2(3P) + P$ (one doubling, one tripling and one general addition). Note that the atomic formulae for general addition was developed by [CCJ04] with a cost of $16M + 32A$, and 9 registers.

For tripling, we present two cases: when using the fast formulae introduced in [LM07a] (see Appendix

E), and when applying the methodology (8) (Appendix D3). Similarly, for quintupling we present the SSCA-protected version for the new fast formula detailed in Appendix C and for the formula based on the methodology (8) (Appendix D3).

Table 3. Cost of atomic operations

Method	Previous work		This work	
	Cost	# reg.	Cost	# reg.
Doubling	$10M + 20A$ [CCJ04]	7	$8M + 12A$	5
w-doubling	$(8w + 2)M + 2(8w + 2)A$ [DIM05][DIM07]	7	$8wM + 12wA$	6
Mixed addition	$11M + 22A$ [Mis06]	8	$12M + 18A$	6
Tripling	$16M + 32A$ [DIM05][DIM07]	10	(1) $14M + 21A$ (2) $16M + 24A$	9 6
w-tripling	$(15w + 1)M + 2(15w + 1)A$ [DIM05][DIM07]	10	(1) $14wM + 21wA$ (2) $16wM + 24wA$	9 6
Quintupling	$34M + 68A$	9	(3) $22M + 33A$ (2) $22M + 33A$	9 7
Septupling	$42M + 84A$	9	(2) $30M + 45A$	7
Doubling-addition (DA)	$21M + 42A$	8	$18M + 27A$	7

(1) Using fast tripling from [LM07a]; (2) Using methodology (8); (3) New quintupling (Appendix C)

As we can see in Table 3, our atomic operations based on the new structure are significantly more efficient in terms of cost and memory than previous atomic operations using *M-A-N-A*, including cases where some savings can be achieved by successive execution of doublings or triplings [DIM05]. Regarding the latter, although our operations do not introduce any extra saving by repeated execution, they still present a performance superior to previous efforts.

Point addition is still one multiplication more expensive than the traditional formulae. However, we expect this disadvantage is minimized because this operation is rare during efficient scalar multiplications. Moreover, if every doubling followed by an addition is otherwise replaced by the new DA, then we gain $3M + 15A$ per nonzero term in the scalar expansion using *one* fewer register.

4.2 Performance Comparison

In the following, we compare performance for the case of scalar multiplications based on radix 2 (i.e., using only point doubling and addition). Performance achieved by scalar multiplications that use composite operations are discussed in Section 5 with the introduction of multibase algorithms, which take advantage of the efficiency of those operations.

To have a more precise idea of the improvement that can be achieved with our new atomic operations, we theoretically compare performance when using a traditional scalar multiplication with NAF method and scalar k of length $n = 160$ bits. Using (3), the cost is estimated in 159D + 53A. If we express the latter in terms of DA operations, then NAF has a estimated cost of (159−53)D+53DA = 106D+53DA.

The operation counting when using the proposed *M-N-A-M-N-A-A* atomic structure and the traditional approach is detailed in Table 4. The costs of atomic operations corresponding to each atomic structure have been taken from Table 3. Also, we include the performance of the new atomic structure when using the proposed atomic DA.

Table 4. Performance of proposed atomic operations (NAF method, n = 160 bits)

Method	Performance	
	Cost	# reg.
Traditional atomic operations	2173*M* + 4346*A*	8
Proposed atomic operations	1908*M* + 2862*A*	6
With proposed atomic DA	1802*M* + 2703*A*	7

As expected, the new *M-N-A-M-N-A-A* structure has reduced significantly the cost of the scalar multiplication in terms not only of field multiplications but also (and more significantly) of field additions. In particular, our atomic structure in combination with the proposed atomic DA yields the highest performance since the disadvantage of having a slower point addition has been completely cancelled by the efficient DA operation. Remarkably, the increment in performance is achieved with even fewer registers than the *M-A-N-A*-based approach.

For comparison purposes, let the ratio *A*/*M* be 0.2. Then, the new *M-N-A-M-N-A-A* structure presents a reduction of 18.5% in comparison with a scalar multiplication using previous atomic operations. When using DA with the *M-N-A-M-N-A-A* structure the reduction is further increased to 23%.

The fast, memory-efficient atomic formulae introduced in this section will be used to efficiently implement SSCA-protected scalar multiplication using the *mb*NAF method, presented in the following section.

5 New Multibase Scalar Multiplication Methods

In the following, we introduce our multibase scalar multiplication based on new NAF-like representations that generalize the use of several bases.

5.1 Multibase Non-Adjacent Form (*mb*NAF)

We propose the following signed multibase representation for the scalar k to construct the scalar multiplication:

$$k = \sum_{i=1}^{m} k_i \prod_{j=1}^{J} a_j^{c_{i(j)}}, \tag{14}$$

where: bases $a_1 \neq a_2 \neq \ldots \neq a_J$ are positive prime integers.
m is the length of the expansion.
k_i are signed digits from a given set D_i.

$c_i(j)$ are monotonically decreasing exponents, s.t. $c_1(j) \geq c_2(j) \geq \ldots \geq c_m(j) \geq 0$, for each j from 1 to J.

The last condition guarantees that an expansion of the form (14) is efficiently executed by a scalar multiplication scanning the digits from left to right.

It is important to note that there are no restrictions on the number of bases. This parameter is determined according to a specific application. In the case studied (i.e., ECC on standard curves over prime fields), we will show that for most of the cases the following parameter selection gives the highest performance: $J \leq 4$, with $a_1 = 2$, $a_2 = 3$, $a_3 = 5$, $a_4 = 7$.

Notice that the signed multibase representation (14) is not unique. In fact, the "Greedy" algorithm by [DIM05] yields an expansion with similar conditions, although limiting the number of bases to only two, namely $J = 2$, with $a_1 = 2$, $a_2 = 3$.

We now define a multibase NAF-like representation that is unique for every positive integer. Although it does not yield a canonical representation in all cases (representations with minimal number of terms using more than one radix are not necessarily efficient for scalar multiplication in all cases), it makes conversion to multibase a trivial task, and guarantees a short expansion for scalar multiplication.

Definition 5.1 Given a set of bases $\mathcal{A} = \{a_1, a_2, \ldots, a_J\}$, where $a_j \in \mathbb{Z}^+$ are positive primes for $1 \leq j \leq J$, a multibase non-adjacent form (*mb*NAF) of a positive integer k, denoted by $\left(k_m^{(a_m)}, k_{m-1}^{(a_{m-1})}, \ldots, k_2^{(a_2)}, k_1^{(a_1)}\right)$, where m is the length of the expansion, $k_i^{(a_i)}$ is the i^{th} digit and the superscript $a_i \in A$ denotes the base associated to the respective digit, has the following properties:

- Every positive integer k has a unique *mb*NAF representation for a given set of bases $\mathcal{A}$.
- No consecutive digits are nonzero.
- $k_i \in D_i = \left\{0, \pm 1, \pm 2, \ldots, \pm\left\lfloor \frac{a_1^2 - 1}{2} \right\rfloor\right\} \setminus \left\{\pm 1 a_1, \pm 2 a_1, \ldots, \pm\left\lfloor \frac{a_1 - 1}{2} \right\rfloor a_1\right\}$, for $1 \leq i \leq m$.
- The leftmost nonzero digit is positive, i.e., $k_m > 0$.

It is important to note that for the set of bases $\mathcal{A} = \{2\}$, the previous definition is identical to the traditional binary NAF.

According to the previous definition, the set of precomputed digits D_i works solely on base a_1, called the main base, to guarantee a minimal number of precomputations.

We propose Algorithm 5.1 to efficiently convert any positive integer to *mb*NAF representation. Notice that the proposed algorithm is a generalization of the traditional NAF to multibase digit representations.

Algorithm 5.1 Computing the *mb*NAF of a positive integer

INPUT: scalar k, bases $\mathcal{A} = \{a_1, a_2, \ldots, a_J\}$, where $a_j \in \mathbb{Z}^+$ are primes for $1 \le j \le J$

OUTPUT: the $(a_1, a_2, ..., a_J)$ NAF(k) = $(..., k_2^{(a_2)}, k_1^{(a_1)})$

1. $i = 0$
2. While $k > 0$ do
   - 2.1. If $k \bmod a_1 = 0$ or $k \bmod a_2 = 0$ or … or $k \bmod a_j = 0$, then $k_i = 0$
   - 2.2. Else:
     - 2.2.1 $k_i = k \bmod s a_1^2$
     - 2.2.2 $k = k - k_i$
   - 2.3
     - 2.3.1 If $k \bmod a_1 = 0$, then $k = k/a_1$, $k_i = k_i^{(a_1)}$
     - 2.3.2 elseif $k \bmod a_2 = 0$, then $k = k/a_2$, $k_i = k_i^{(a_2)}$
     - ⋮
     - 2.3.J elseif $k \bmod a_J = 0$, then $k = k/a_J$, $k_i = k_i^{(a_J)}$
   - 2.4 $i = i + 1$
3. Return $(..., k_2^{(a_2)}, k_1^{(a_1)})$

The function *mods* in Algorithm 5.1 represents the next computation ($w = 2$):

$$\begin{cases} \text{If } k \bmod a_1^w \ge a_1^w / 2, \text{ then:} \\ \quad k_i = \left(k \bmod a_1^w\right) - a_1^w \\ \text{Else,} \\ \quad k_i = k \bmod a_1^w \end{cases} \tag{15}$$

Algorithm 5.1 approximates every computation to the closest number divisible by the square of the main base, namely a_1. Thus, two consecutive operations by a_1 are guaranteed before the next addition. In this sense, it closely follows the structure of the expansion given by *r*NAF [TYW04]. The analogy is quite interesting since our main base a_1 acts as the radix r, using a similar construction for the table of precomputed points. However, the nonzero density of the expansion is further reduced in our case as the number of bases is increased since the algorithm looks for extra divisions by the remainder bases. Consequently, the nonzero density decreases with the number and size of the bases.

Following Definition 5.1, this method requires

$$\frac{(a_1 - 2)(a_1 + 1)}{2} \tag{16}$$

precomputed points without considering $\{0, \pm 1\}$. Further, preliminary analysis shows that the expected average nonzero density is asymptomatically

$$D = \frac{a_1 - 1}{2a_1 - 1} \tag{17}$$

with regard to the main base.

It is important to note that if a_1=2, no more precomputed points are required in comparison with the binary NAF representation, where the point at infinity and P must be stored.

Also, $a_1 = 2$ is expected to yield the most efficient scalar multiplication in terms of speed on EC standard curves, where point doubling is highly efficient in comparison with other operations. However, in new EC-based cryptosystems of characteristic 3 or pairing-based cryptosystems where triplings (or other composite operations) are more efficient, the expectation is that the case with $a_1 \neq 2$ provide a better performance.

5.2 Window-*w* Multibase Non-Adjacent Form (*wmb*NAF)

Similarly to NAF, it is possible to further reduce the density of the expansion of the proposed *mb*NAF by allowing additions by an extended set of precomputed digits D_i.

In the following, we define the window-*w* Non-Adjacent Form for multibase representations (*wmb*NAF).

Definition 5.2 Given a set of bases $\mathcal{A}$ = {a_1,a_2,…,a_J}, where $a_j \in \mathbb{Z}^+$ are positive primes for $1 \leq j \leq J$, the window-*w* multibase non-adjacent form (*wmb*NAF) of a positive integer k using window of width $w \geq 2$, denoted by $\left(k_m^{(a_m)}, k_{m-1}^{(a_{m-1})}, \ldots, k_2^{(a_2)}, k_1^{(a_1)}\right)$, where m is the length of the expansion, $k_i^{(a_i)}$ is the i^{th} digit and the superscript $a_i \in A$ denotes the base associated to the respective digit, has the following properties:

- Every positive integer k has a unique *wmb*NAF representation for a given set of bases $\mathcal{A}$ and window w.
- w adjacent digits contain at most one nonzero digit.
- $k_i \in D_i = \left\{0, \pm 1, \pm 2, \ldots, \pm\left\lfloor \frac{a_1^w - 1}{2} \right\rfloor\right\} \setminus \left\{\pm 1 a_1, \pm 2 a_1, \ldots, \pm\left\lfloor \frac{a_1^{w-1} - 1}{2} \right\rfloor a_1\right\}$, for $1 \leq i \leq m$.
- The leftmost nonzero digit is positive, i.e., $k_m > 0$.

Notice that for the set of bases $\mathcal{A}$ = {2}, the previous definition is identical to the traditional *w*NAF.

Similarly to *mb*NAF, the set of precomputed digits D_i are derived from the main base a_1, limiting the required number of precomputations.

We propose Algorithm 5.2 to efficiently convert the scalar k to the *wmb*NAF representation.

Function *mods* in Algorithm 5.2 is a generalization of function (15) to $w > 2$. Similarly to the relation between *w*NAF and NAF, *wmb*NAF is equivalent to *mb*NAF if we fix $w = 2$. Thus, Algorithm 5.2 defines windows of size w (only for the main base a_1) to which it approximates every computation. In this way, w consecutive operations by a_1 are guaranteed before the next addition.

Following Definition 5.2, this method requires

$$\frac{a_1^w - a_1^{w-1} - 2}{2} \tag{18}$$

Algorithm 5.2 Computing the *wmb*NAF of a positive integer

INPUT: scalar k, bases $\mathcal{A} = \{a_1,a_2,\ldots,a_J\}$, where $a_j \in \mathbb{Z}^+$ are primes for $1 \le j \le J$,
window $w \ge 2$, where $w \in \mathbb{Z}^+$

OUTPUT: the $(a_1, a_2, ..., a_J)\,\mathrm{NAF}_w(k) = (\ldots, k_2^{(a_2)}, k_1^{(a_1)})$

1. $i = 0$
2. While $k > 0$ do
   2.1. If $k \bmod a_1 = 0$ or $k \bmod a_2 = 0$ or … or $k \bmod a_j = 0$, then $k_i = 0$
   2.2. Else:
      2.2.1 $k_i = k \bmods a_1^w$
      2.2.2 $k = k - k_i$
   2.3
      2.3.1 If $k \bmod a_1 = 0$, then $k = k/a_1$, $k_i = k_i^{(a_1)}$
      2.3.2 elseif $k \bmod a_2 = 0$, then $k = k/a_2$, $k_i = k_i^{(a_2)}$
      ⋮
      2.3.J elseif $k \bmod a_J = 0$, then $k = k/a_J$, $k_i = k_i^{(a_J)}$
   2.4 $i = i + 1$
3. Return $(\ldots, k_2^{(a_2)}, k_1^{(a_1)})$

precomputed points without considering $\{0, \pm 1\}$. Further, preliminary analysis shows that the expected average nonzero density is asymptomatically

$$D = \frac{a_1 - 1}{w(a_1 - 1) + 1}. \tag{19}$$

As happens with *mb*NAF, defining $a_1 = 2$ is expected to yield the fastest scalar multiplication on EC standard curves, given the high efficiency of point doubling. In settings where tripling (or another composite operation) is more efficient, $a_1 = 3$ (or the value corresponding to the efficient composite operation) may achieve the highest performance.

5.3 Extended Window–*w* Multibase Non-Adjacent Form (*extended wmb*NAF)

In *mb*NAF and *wmb*NAF, we have restricted the internal approximation to numbers divisible by the base a_1, which has been referred to as the main base. However, curve-based cryptosystems other than ECC on standard curves offer different cost ratios among their point operations. To exploit the efficiency of different point operations in a given setting, we extend the proposed multibase representations by allowing internal approximations to numbers based on combinations of radices.

Definition 5.3 Given a set of bases $\mathcal{A} = \{a_1,a_2,\ldots,a_J\}$, where $a_j \in \mathbb{Z}^+$ are positive primes for $1 \ge j \ge J$, the extended multibase non-adjacent form (*extended mb*NAF) of a positive integer k using

window set $\mathcal{W} = \{w_1, w_2, \ldots, w_J\}$, where exponents w_j are positive integers ≥ 0 for all j from 1 to J, is denoted by $\left(k_m^{(a_m)}, k_{m-1}^{(a_{m-1})}, \ldots, k_2^{(a_2)}, k_1^{(a_1)}\right)$, where m is the length of the expansion, $k_i^{(a_i)}$ is the i^{th} digit and the superscript $a_i \in A$ denotes the base associated to the respective digit. The *extended mb*NAF has the following properties:

- Every positive integer k has a unique *extended mb*NAF representation for a given set of bases $\mathcal{A}$ and windows $\mathcal{W}$.
- There is at most one nonzero digit among $(w_1 + w_2 + \ldots + w_J)$ adjacent digits.
- $k_i \in D_i = \left\{0, \pm 1, \pm 2, \ldots, \pm\left\lfloor \frac{a-1}{2} \right\rfloor\right\} \setminus \ldots$
  $\ldots\left\{\pm 1a_1, \pm 2a_1, \ldots, \pm\left\lfloor \frac{a-1}{2a_1} \right\rfloor a_1, \pm 1a_2, \pm 2a_2, \ldots, \pm\left\lfloor \frac{a-1}{2a_2} \right\rfloor a_2, \ldots, \pm 1a_J, \pm 2a_J, \ldots, \pm\left\lfloor \frac{a-1}{2a_J} \right\rfloor a_J\right\}$, for $1 \leq i \leq m$.
- The leftmost nonzero digit is positive, i.e., $k_m > 0$.

We propose Algorithm 5.3 to efficiently convert the scalar k to *extended wmb*NAF representation.

Algorithm 5.3 Computing the *extended wmb*NAF of a positive integer

INPUT: scalar k, bases $\mathcal{A} = \{a_1, a_2, \ldots, a_J\}$, where $a_j \in \mathbb{Z}^+$ are primes for $1 \leq j \leq J$;
$a = a_1^{w_1} a_2^{w_2} \ldots a_J^{w_J}$, using window set $\mathcal{W} = \{w_1, w_2, \ldots, w_J\}$, where $w_j \geq 0$ for $1 \leq j \leq J$

OUTPUT: the $(a_1, a_2, \ldots, a_J)$ NAF$_{(w_1, w_2, \ldots, w_J)}(k) = (\ldots, k_2^{(a_2)}, k_1^{(a_1)})$

1. $i = 0$
2. While $k > 0$ do
   2.1. If $k \bmod a_1 = 0$ or $k \bmod a_2 = 0$ or … or $k \bmod a_j = 0$, then $k_i = 0$
   2.2. Else:
      2.2.1 $k_i = k \bmod s a$
      2.2.2 $k = k - k_i$
   2.3
      2.3.1 If $k \bmod a_1 = 0$, then $k = k/a_1$, $k_i = k_i^{(a_1)}$
      2.3.2 elseif $k \bmod a_2 = 0$, then $k = k/a_2$, $k_i = k_i^{(a_2)}$
      ⋮
      2.3.J elseif $k \bmod a_J = 0$, then $k = k/a_J$, $k_i = k_i^{(a_J)}$
   2.4 $i = i + 1$
3. Return $(\ldots, k_2^{(a_2)}, k_1^{(a_1)})$

The function *mods* in Algorithm 5.3 involves the next computation:

$$\begin{cases} \text{If } k \bmod a \geq a/2, \text{ then:} \\ \quad k_i = (k \bmod a) - a \\ \text{Else,} \\ \quad k_i = k \bmod a \end{cases} \tag{20}$$

Algorithm 5.3 guarantees that there is at most one nonzero digit among (w_1 + w_2 + … + w_J) consecutive digits. In contrast to Algorithm 5.2, this method establishes windows for different bases instead of limiting the window solely to base a_1. Thus, every computation is approximated to an extended value a, which we refer to as the global base, guaranteeing a given number of operations per radix before the next addition happens.

Notice that *wmb*NAF is in fact a particular case of the *extended wmb*NAF when the set of windows is limited to the main base, i.e., $\mathcal{W}=\{w_1\}$, where $w_i \geq 2$.

This extended representation is ideal for settings where more than one point operation is efficient. Moreover, its flexibility to define different window sizes for different bases allows us to determine windows according to specific cost ratios between point operations. For instance, for multibase representations using bases $\mathcal{A} = \{2,3\}$ on cryptosystems that have highly efficient tripling, it is more beneficial to use a relatively bigger window w_2 for the base 3 than to window w_1 for base 2.

*Extended wmb*NAF requires the next set of precomputed points:

$$k_i \in \left\{0, \pm 1, \pm 2, \ldots, \pm \left\lfloor \frac{a-1}{2} \right\rfloor \right\} \setminus \ldots$$
$$\ldots \left\{ \pm 1a_1, \pm 2a_1, \ldots, \pm \left\lfloor \frac{a-1}{2a_1} \right\rfloor a_1, \pm 1a_2, \pm 2a_2, \ldots, \pm \left\lfloor \frac{a-1}{2a_2} \right\rfloor a_2, \ldots, \pm 1a_J, \pm 2a_J, \ldots, \pm \left\lfloor \frac{a-1}{2a_J} \right\rfloor a_J \right\}$$

For instance, if we fix the number of bases to 3, we obtain the following expression to compute the number of required precomputed points:

$$\left\lfloor \frac{a}{2} - 1 \right\rfloor - \left\lfloor \frac{a}{2a_1} \right\rfloor - \left\lfloor \frac{a}{2a_2} \right\rfloor - \left\lfloor \frac{a}{2a_3} \right\rfloor + \left\lfloor \frac{a}{2a_1a_2} \right\rfloor - (n-2)\left\lfloor \frac{a}{2a_1a_2a_3} \right\rfloor \tag{21}$$

Notice that expression (21) only requires bases a_j that are present in the global base a. In other words, a given base a_j is not used in computation of (21) if $w_j = 0$.

Algorithm 5.3 searches for the closest number divisible by the window established by the global base a. Therefore, our approach's density is determined by every base in the global base. In fact:

$$D = D_{a_j} \times \#\left(O_{a_j}\right), \tag{22}$$

for any base a_j found in the global base a (i.e., $w_j = 0$ for a given j). D_{a_j} and $\#\left(O_{a_j}\right)$ represent density and number of operations associated to a given base a_j, respectively.

Extending the analysis of *mb*NAF and *wmb*NAF to this case, the following expression can be derived for the density D_{a_j}:

$$D_{a_j} = \frac{a_j - 1}{w_j\left(a_j - 1\right) + 1}. \tag{23}$$

5.4 Multibase Scalar Multiplication

Following, we propose a general multibase scalar multiplication for our three studied NAF-like expansions: *mb*NAF, *wmb*NAF and *extended wmb*NAF.

In Algorithm 5.4, $k_i^{(a_i)}$ is the i^{th} digit and the superscript $a_i \in A$ (the set of bases) denotes the base associated to the respective digit. Thus, the operation $Q = (a_i)Q$ in step 3.1 is any point operation with exception of addition (i.e., doubling, tripling, quintupling, and so on). In step 3.2, point addition is performed when a nonzero element is found in the multibase expansion. The operation is performed with a precomputed point P_i according to Definitions 5.1, 5.2 and 5.3, corresponding to *mb*NAF, *wmb*NAF and *extended wmb*NAF, respectively.

In the following section, we detail our extensive tests to determine the efficiency of this new multibase scalar multiplication and compare to the best previous efforts.

Algorithm 5.4 ***Extended wmb*NAF (*mb*NAF or *wmb*NAF) method for scalar multiplication**

INPUT: set $\mathcal{W}=\{w_1,w_2,\ldots,w_J\}$, where $w_j \geq 0$ ($w_1 = 2$, $w_j = 0$ for *mb*NAF, and $w_1 > 2$, $w_j = 0$ for *wmb*NAF; $j > 1$), scalar $k = (k_{l-1}^{(a_{l-1})},\ldots,k_2^{(a_2)},k_1^{(a_1)})$ from Alg. 5.1, 5.2 or 5.3, $P \in E(\mathbb{F}_p)$

OUTPUT: kP

1. Compute $P_i = iP$ for $i \in D_i$ (see Definitions 5.1, 5.2 or 5.3)
2. $Q = O$
3. For $i = l-1$ downto 0 do
   3.1. $Q = (a_i)Q$
   3.2. If $k_i^{(a_i)} \neq 0$, then:
      If $k_i^{(a_i)} > 0$, then $Q = Q + P_{k_i^{(a_i)}}$
      Else $Q = Q - P_{k_i^{(a_i)}}$
4. Return (Q)

6 Testing Results

The following algorithms have been implemented and run for 10000 random numbers with maximum bitlength n = 160 with the objective of estimating the average number of operations required by each of them:

- Algorithm 2.2 for the case $w = 2$ and $w \geq 2$, corresponding to NAF and *w*NAF, respectively.
- Algorithms 5.1, 5.2 and 5.3, corresponding to *mb*NAF, *wmb*NAF and *extended wmb*NAF, respectively.

The number of point operations required to compute scalar multiplication using *mb*NAF with different sets of bases $\mathcal{A}$ is displayed in Table 5, and compared with NAF. It can be seen that the nonzero density (i.e., number of point additions) strictly decreases with the number of bases (as well as the total number

of required operations), highlighting the potential increase in speed introduced by multibase representations. In fact, we can observe the sublinear nature of multibase expansions, which exhibit densities even below $(\log k / \log\log k)$ additions, in contrast to linearity in (log k) observed for NAF. Nevertheless, the decrease in density rapidly slows down for expansions using high number of bases. Ultimately, the efficiency of a given composite operation corresponding to each base will "decide" if it is beneficial to include it in the expansion of the scalar.

Next, using our test results we evaluate the performance of scalar multiplication using the *mb*NAF methods for two possible scenarios: unprotected implementations where SSCA is not a concern, and SSCA-protected implementations using side-channel atomicity as developed in Chapter 4.

Results are compared with performance reported by [DIM05] and [MD07] using the DB and TB methods, which show the previously fastest scalar multiplications on the standard curve (1).

Table 5. Performance of mbNAF and NAF in terms of point operations (10000 random numbers, n = 160 bits)

Method	Point operation cost (1)
NAF	158.7D + 52.8A
(2,3)NAF	113.5D + 28.4T + 37.7A
(2,3,5)NAF	96.7D + 24.3T + 10.1Q + 32.0A
(2,3,5,7)NAF	86.8D + 21.9T + 9.1Q + 5.7S + 28.7A
(2,3,5,7,11)NAF	81.1D + 20.4T + 8.5Q + 5.4S + 3.0E + 26.8A
(2,3,5,7,11,13)NAF	76.58D + 19.24T + 8.05Q + 5.16S + 2.83E + 2.31TH + 25.23A

(1) A: addition, D: doubling, T: tripling, Q: quintupling, S: septupling, E: eleventupling (11P), TH: thirteentupling (13P)

6.1 Unprotected Implementations

We have evaluated the performance of our multibase scalar multiplication methods for the following cases: $\mathcal{A}$={2,3}, $\mathcal{A}$={2,3,5}, $\mathcal{A}$={2,3,5,7}, $\mathcal{A}$={2,3,5,7,11} and $\mathcal{A}$={2,3,5,7,11,13}, with windows $2 \leq w \leq 6$. For the *extended wmb*NAF, we have considered a window set $\mathcal{W} = \{w_1,w_2,\ldots,w_6\}$, where $w_i \in \{0,1,2,3\}$.

We have further improved the performance of our scalar multiplication by using the efficient point operations introduced in [LM07a] and Sections 2 and 3 of this work (see Tables 1 and 2).

Table 6 details the performance of our new *mb*NAF method using bases $\mathcal{A}$={2,3}, $\mathcal{A}$={2,3,5} and $\mathcal{A}$={2,3,5,7}, which achieved the best results, and compares them to the traditional NAF, DB and TB. It can be seen that all our variants outperform previous scalar multiplications for a wide margin. In particular, (2,3,5)NAF and (2,3,5,7)NAF offer comparable performance with reductions of 11.9%, 10.8% and 9.4% in comparison with NAF, DB and TB, respectively. Hence, we can state that these are the fastest methods to compute scalar multiplications on standard curves over prime fields without precomputations. On the other hand, if memory is a primary concern, the performance of our algorithms using the composite operations derived from methodology (8) (see Section 3) is very competitive (still faster than any previous method) with the additional advantage of requiring only 6 registers, including storage for input/output point coordinates and intermediate computations.

Table 6. Performance of different scalar multiplications methods (n = 160, no precomputations)

Method	Performance	
	Cost	# reg.
(2,3)NAF	1514M 1542M	8 (1) 6 (2)
(2,3,5)NAF	1490M 1518M	10 (1) 6 (2)
(2,3,5,7)NAF	1491M 1517M	10 (1) 6 (2)
NAF	1691M	7 (3)
TB [MD07]	1645M	10
DB [DIM05]	1671M	10

(1) Using fast tripling from [LM07a] and new quintupling from Appendix C;
(2) Using methodology (8); (3) Using traditional point formulae [HMV04].

Table 7. Performance of different scalar multiplication methods using precomputations (n = 160)

Method	Cost (1 point) w = 3	Cost (3 point) w = 4	Cost (7 points) w = 5	Cost (15 points) w = 6	# reg.
(2,3)NAF$_w$	(1) 1460M (2) 1473M	1384M 1399M	1344M 1355M	1307M 1318M	R + 8 R + 6
(2,3,5)NAF$_w$	(1) 1444M (2) 1459M	1383M 1400M	1345M 1358M	1311M 1323M	R + 10 R + 6
(2,3,5,7)NAF$_w$	(1) 1449M (2) 1463M	1394M 1410M	1358M 1369M	1323M 1334M	R + 10 R + 6
NAF$_w$	(3) 1549M	1463M	1405M	1362M	R + 7
TB [MD07]	1569M	1515M	1552M	N/A	R + 10

R: number of registers to store precomputed points
(1) Using fast tripling from [LM07a] and new quintupling from Appendix C; (2) Using methodology (8);
(3) Using traditional point formulae [HMV04].

Similarly, in Table 7 we show performance of the *wmb*NAF method using bases $\mathcal{A}$={2,3}, $\mathcal{A}$={2,3,5} and $\mathcal{A}$={2,3,5,7} for cases w = 3, 4, 5 and 6, and compare them to wNAF and TB. In this case, (2,3,5)NAF$_w$ achieves the lowest computing cost, outperforming wNAF and TB methods. For instance, for w = 3, our method surpasses NAF$_3$ and TB by 6.8% and 8%, respectively. We can also observe that composite operations derived from (8) allow efficient implementations in terms of memory consumption. The memory requirement in this case is limited to only 6 registers in addition to the R registers necessary to store the precomputed points.

In addition, we have analyzed the performance of our method when applied to other families of curves where different composite operations are very efficient. As an example, we evaluate the computing costs for the special ECC curves with degree 3 isogenies presented by [DIK06]. In such scenario, costs of doubling, tripling and addition are 2M + 7S, 6M + 6S and 7M + 4S, respectively.

Given the efficiency of tripling and its particular ratio T/D, it can be expected to achieve the highest performance with our *extended wmb*NAF as this method allows us to flexibly fix window sizes for every

base. Table 8 details results of our tests, and compare them with results using NAF and the new *mb*NAF and *wmb*NAF methods.

Table 8. Performance of different scalar multiplications using ECC curves with degree 3 isogenies [DIK06] (n = 160 bits, $1S = 0.8M$)

Method	Points	Cost
*Extended wmb*NAF: (2,3)$NAF_{1,1}$	0	1505*M*
*mb*NAF: (2,3)NAF	0	1554*M*
NAF	0	1744*M*
*Extended wmb*NAF: (2,3)$NAF_{2,1}$	1	1446*M*
*wmb*NAF: (2,3)NAF_3	1	1540*M*
NAF_3	1	1605*M*
*Extended wmb*NAF: (2,3)$NAF_{1,2}$	2	1386*M*
*Extended wmb*NAF: (2,3)$NAF_{2,2}$	5	1356*M*
*wmb*NAF: (2,3)NAF_5	7	1421*M*
NAF_5	7	1462*M*
*Extended wmb*NAF: (2,3)$NAF_{1,3}$	8	1316*M*

As we can see in the table above, *extended wmb*NAF achieves the highest performance in all the studied cases. For instance, it achieves a significant reduction of about 13.7% in comparison with the NAF method without pre-computations. It is interesting to note that in most cases, our method achieves the lowest cost using windows relatively larger for radix 3 than for radix 2. As was previously discussed, this is due to the efficiency of the tripling on these curves. Also, comparing Tables 6, 7 and 8, we observe that the *extended wmb*NAF method allows the special curves proposed by [DIK06] to achieve comparable or even superior performance to the best cases with Jacobian coordinates (standard curves).

6.2 SSCA-Protected Implementations

We have evaluated the performance of SSCA-protected *mb*NAF scalar multiplications for the following cases: $\mathcal{A}=\{2,3\}$, $\mathcal{A}=\{2,3,5\}$ and $\mathcal{A}=\{2,3,5,7\}$, with windows $2 \leq w \leq 6$. For the *extended wmb*NAF, we have considered the window set $\mathcal{W} = \{w_1,w_2,\ldots,w_4\}$, where $w_i \in \{0,1,2,3\}$.

From our tests, we conclude that *M*-*N*-*A*-*M*-*N*-*A*-*A*-based formulae using atomic DA achieves the highest performance when assuming $1S = 1M$. In that case, Tables 9 and 10 compare our best implementation cases using *mb*NAF and *wmb*NAF, respectively, with bases $\mathcal{A}=\{2,3\}$, $\mathcal{A}=\{2,3,5\}$ and $\mathcal{A}=\{2,3,5,7\}$.

We can see in Table 9 that (2,3,5)NAF and (2,3,5,7)NAF again offer comparable costs and outperform the best previous approaches with protection against SSCA, reducing not only the number of field multiplications but also significantly reducing the number of additions. For instance, if we consider the ratio $A/M = 0.2$, (2,3,5)NAF presents reductions in terms of computing time of 29.1% and 15.6%

over NAF and DB, respectively.

If memory is scarce, then our methods using the SSCA-protected composite operations developed in Section 4 also outperform previous methods and offer reduced memory requirement, fixed at only 7 registers.

Table 9. Performance of different SSCA-protected scalar multiplication methods (n = 160, no precomputations)

Method	Performance			
	Cost ($1S = 1M$)	$1A = 0.05M$	$1S = 0.2M$	# reg.
(2,3)NAF	$1682M + 2524A$ $1739M + 2609A$	$1808M$ $1869M$	$2187M$ $2261M$	9 (1) 7 (2)
(2,3,5)NAF	$1655M + 2483A$ $1704M + 2556A$	$1779M$ $1832M$	$2152M$ $2215M$	9 (1) 7 (2)
(2,3,5,7)NAF	$1658M + 2487A$ $1702M + 2553A$	$1782M$ $1830M$	$2155M$ $2213M$	9 (1) 7 (2)
NAF	$2167M + 4334A$	$2384M$	$3034M$	8 (3)
DB [DIM05]	$1822M + 3645A$	$2004M$	$2551M$	10 (3)

(1) Using fast tripling from [LM07a] and new quintupling from Appendix C; (2) Using methodology (8);
(3) Using previous atomic formulae [CCJ04],[Mis06],[DIM05].

Table 10. Performance of different SSCA-protected scalar multiplication methods using precomputations (n = 160)

Method	Cost (1 point) $w = 3$	Cost (3 point) $w = 4$	Cost (7 points) $w = 5$	Cost (15 points) $w = 6$	# reg.
$(2,3)NAF_w$	(1) $2119M$ (2) $2153M$	$2015M$ $2054M$	$1961M$ $1990M$	$1912M$ $1940M$	R + 9 R + 7
$(2,3,5)NAF_w$	(1) $2094M$ (2) $2123M$	$2013M$ $2049M$	$1962M$ $1989M$	$1916M$ $1942M$	R + 9 R + 7
$(2,3,5,7)NAF_w$	(1) $2103M$ (2) $2130M$	$2029M$ $2063M$	$1978M$ $2003M$	$1932M$ $1956M$	R + 9 R + 7
NAF_w	(3) $2823M$	$2693M$	$2605M$	$2540M$	R + 8

R: number of registers to store precomputed points
(1) Using fast tripling from [LM07a] and new quintupling from Appendix C; (2) Using methodology (8);
(3) Using previous atomic formulae [CCJ04],[Mis06].

In the case of methods using pre-computations (Table 10), we observe that $(2,3,5)NAF_3$ surpasses NAF_3 by 25.8%, and that $(2,3,5)NAF_4$ outperforms NAF_4 by 25.3%. Also, $(2,3)NAF_5$ and $(2,3)NAF_6$ are the fastest for windows w = 5 and 6, and show reductions of 24.7%. A slightly slower performance can be achieved when using the composite operations derived from (8), although in this case the memory requirement is limited to only 7 registers in addition to the R registers necessary to store the precomputed points.

8 Conclusions

We have presented a sublinear scalar multiplication that uses the new multibase non-adjacent form method to reduce the number of required terms in the scalar expansion. Three variants are developed that differ in window size (and number of precomputations), and apply to different settings according to the availability of efficient composite operations. The algorithms presented to convert numbers to *mb*NAF, *wmb*NAF and *extended wmb*NAF representations solve the problem of finding short multibase expansions without consuming memory and/or degrading speed. Furthermore, to realize the new scalar multiplication in the ECC setting over prime fields, new composite operations (tripling, quintupling, septupling, and so on) that are faster and consume less memory are developed and protected against simple side-channel attacks using atomicity. Extensive tests with thousands of random numbers show that scalar multiplication using the multibase method is the fastest approach found in the current literature. Remarkably, this method is expected to achieve similar results in other settings such as Pairing-based cryptosystems, ECC over binary fields, ECC using Edwards curves or extended Jacobi quartics, and many others.

References

[ACD+05] R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen and F. Vercauteren, "Handbook of Elliptic and Hyperelliptic Curve Cryptography," CRC Press, 2005.

[Ava04] R. Avanzi, "Aspects of Hyperelliptic Curves over Large Prime Fields in Software Implementations," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES'04)*, LNCS Vol. 3156, pp. 148-162, Springer-Verlag, 2004.

[Ber01] D. Bernstein, "A Software Implementation of NIST P-224," presentation in *Elliptic Curve Cryptography (ECC'01)*, 2001.

[Ber06] D. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," in *Public Key Cryptography (PKC'06)*, LNCS Vol. 3958, pp. 229-240, Springer-Verlag, 2006.

[BHL+01] M. Brown, D. Hankerson, J. Lopez and A. Menezes, "Software Implementation of the NIST elliptic curves over prime fields," in *Progress in Cryptology CT-RSA 2001* , LNCS Vol. 2020, pp. 250-265, Springer-Verlag, 2001.

[BJ03] O. Billet and M. Joye, "Fast Point Multiplication on Elliptic Curves through Isogenies," in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, LNCS Vol. 2643, pp. 43–50, Springer-Verlag, 2003.

[CCJ04] B. Chevallier-Mames, M. Ciet and M. Joye, "Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity," in *IEEE Transactions on Computers*, Vol. 53, No 6, pp. 760-768, 2004.

[CJL+06] M. Ciet, M. Joye, K. Lauter and P. L. Montgomery, "Trading Inversions for Multiplications in Elliptic Curve Cryptography," in *Designs, Codes and Cryptography*, Vol. 39, No 2, pp.189-206, 2006.

[DI06] C. Doche and L. Imbert, "Extended Double-Base Number System with Applications to Elliptic Curve Cryptography," in *Progress in Cryptology (INDOCRYPT'06)*, LNCS Vol. 4329, pp 335-348, Springer-Verlag, 2006.

[DIK06] C. Doche, T. Icart and D. Kohel, "Efficient Scalar Multiplication by Isogeny Decompositions," in *Public Key Cryptography (PKC'06)*, LNCS Vol. 3958, pp. 191-206, Springer-Verlag, 2006.

[DIM05] V. Dimitrov, L. Imbert and P.K. Mishra, "Efficient and Secure Elliptic Curve Point Multiplication using Double-Base Chains," *Advances in Cryptology (ASIACRYPT'05)*, LNCS Vol. 3788, pp. 59–78, Springer-Verlag, 2005.

[DIM07] V. Dimitrov, L. Imbert and P.K. Mishra, "The Double-base Number System and its Application to Elliptic Curve Cryptography", to appear in *Mathematics of Computation*, 2007.

[DM07] V. Dimitrov and P.K. Mishra, "Efficient Quintuple Formulas for Elliptic Curves and Efficient Scalar Multiplication Using Multibase Number," in *International Conference on Information Security (ISC'04)*, LNCS Vol. 4779, pp. 390–406, Springer-Verlag, 2007.

[Elm06] L. Elmegaard-Fessel, "Efficient Scalar Multiplication and Security against Power Analysis in Cryptosystems based on the NIST Elliptic Curves over Prime Fields," *Master Thesis*, University of Copenhagen, 2006.

[GAS+05] J. Großschädl, R. Avanzi, E. Savaş and S. Tillich, "Energy-Efficient Software Implementation of Long Integer Modular Arithmetic," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES'05)*, LNCS Vol. 3659, pp. 75-90, Springer-Verlag, 2005.

[GG03] C.H. Gebotys and R.J. Gebotys, "Secure Elliptic Curve Implementations: An Analysis of Resistance to Power-Attacks in a DSP Processor," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES'03)*, LNCS Vol. 2523, pp. 114-128, Springer-Verlag, 2003.

[HMV04] D. Hankerson, A. Menezes and S. Vanstone, "Guide to Elliptic Curve Cryptography," Springer-Verlag, 2004.

[IEEE] IEEE Std 1363-2000. IEEE Standard Specifications for Public-Key Cryptography. *The Institute of Electrical and Electronics Engineers (IEEE)*, 2000.

[ITT+99] K. Itoh, M. Takenaka, N. Torii, S. Temma and Y. Kurihara, "Fast Implementation of Public-Key Cryptography on a DSP TMS320C6201," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS Vol. 1717, pp. 61-72, Springer-Verlag, 1999.

[JY02] M. Joye and S.-M. Yen, "New Minimal Modified Radix-r Representation with Applications to Smart Cards," in *Public Key Cryptography (PKC'02)*, LNCS Vol. 2274, pp. 375-384, Springer-Verlag, 2002.

[Koc96] C. Kocher, "Timing Attacks on Implementations of Diffie−Hellman, RSA, DSS, and Other Systems," in *CRYPTO'96*, LNCS Vol. 1109, pp.104−113, Springer-Verlag, 1996.

[KJJ99] C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *CRYPTO'99*, LNCS Vol. 1666, pp. 388−397, Springer-Verlag, 1999.

[LM07a] P. Longa and A. Miri, "Fast and Flexible Elliptic Curve Point Arithmetic," to appear in *IEEE Transactions on Computers*, 2007. Also available at http://doi.ieeecomputersociety.org/10.1109/TC.2007.70815.

[LM07b] P. Longa and A. Miri, "New Composite Operations and Precomputation Scheme for Elliptic Curve Cryptosystems over Prime Fields," accepted to *Public Key Cryptography (PKC'08)*, 2007.

[MD07] P. K. Mishra and V. Dimitrov, "Efficient Quintuple Formulas for Elliptic Curves and Efficient Scalar Multiplication using Multibase Number Representation," *Cryptology ePrint Archive*, Report 2007/040, 2007.

[Mel06] N. Meloni, "Fast and Secure Elliptic Curve Scalar Multiplication over Prime Fields using Special Addition Chains," *Cryptology ePrint Archive*, Report 2006/216, 2006.

[Mis06] P. K. Mishra, "Pipelined Computation of Scalar Multiplication in Elliptic Curve Cryptosystems," in *IEEE Transactions on Computers*, Vol. 25, No. 8, pp. 1000-1010, 2006.

[TYW04] T. Takagi, S-M. Yen and B-C. Wu, "Radix-r Non-Adjacent Form," in *International Conference on Information Security (ISC'04)*, LNCS Vol. 3225, pp. 99-110, Springer-Verlag, 2004.

[XB01] S.B. Xu and L. Batina, "Efficient Implementation of Elliptic Curve Cryptosystems on an ARM7 with Hardware Accelerator," in *Int. Conf. on Information and Communications Security (ICICS'01)*, LNCS Vol. 2200, pp. 266-279, Springer-Verlag, 2001.

APPENDIX A1: COMPOSITE OPERATIONS OF THE FORM *dP*

In this section, we present the pseudocode to compute the doubling of a point and the composite operations of the form *dP* introduced in Section 3. The doubling uses the fast formula by [LM07a] for the special case $a = -3$.

Algorithm 1: Composite Operation *dP*, where *d* is an odd prime ≥ 2, $(d)\mathcal{J} \rightarrow \mathcal{J}$, $E: y^2 = x^3 - 3x + b$

INPUT: point $P = (X_1, Y_1, Z_1)$ on $E(\mathbb{F}_p)$, $T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z_1$

OUTPUT: if $d = 2$: point $2P = (X_2, Y_2, Z_2)$; if $d \geq 3$: $dP = (X_{(d+3)/2}, Y_{(d+3)/2}, Z_{(d+3)/2})$

1. If $P = O$, then return (O)	
2. $T_4 = T_3^2$	$\{Z_1^2\}$
3. $T_5 = T_1 + T_4$	$\{X_1 + Z_1^2\}$
4. $T_6 = T_1 - T_4$	$\{X_1 - Z_1^2\}$
5. $T_5 = T_5 \times T_6$	$\{(X_1 + Z_1^2)(\{X_1 - Z_1^2\})\}$
6. $T_5 = 3T_5$	$\{\alpha = 3(X_1 + Z_1^2)(X_1 - Z_1^2)\}$
7. $T_3 = T_2 + T_3$	$\{Y_1 + Z_1\}$
8. $T_3 = T_3^2$	$\{(Y_1 + Z_1)^2\}$
9. $T_2 = T_2^2$	$\{Y_1^2\}$
10. $T_2 = T_2 + T_4$	$\{Y_1^2 + Z_1^2\}$
11. $T_3 = T_3 - T_4$	$\{Z_2 = (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2\}$
12. $T_1 = 4T_1$	$\{4X_1\}$
13. $T_4 = T_1 \times T_2$	$\{X_1^{(1)} = 4X_1Y_1^2\}$
14. $T_1 = 2T_4$	$\{8X_1Y_1^2\}$
15. $T_6 = T_5^2$	$\{\alpha^2\}$
16. $T_1 = T_6 - T_1$	$\{X_2 = \alpha^2 - 8X_1Y_1^2\}$
17. $T_6 = T_4 - T_1$	$\{X_1^{(1)} - X_2\}$
18. $T_5 = T_5 \times T_6$	$\{\alpha(X_1^{(1)} - X_2)\}$
19. $T_2 = T_2^2$	$\{Y_1^4\}$
20. $T_2 = 8T_2$	$\{Y_1^{(1)} = 8Y_1^4\}$
21. $T_5 = T_5 - T_2$	$\{Y_2 = \alpha(X_1^{(1)} - X_2) - 8Y_1^4\}$
22. If $d = 2$, then return $2P = (T_1, T_2, T_3) = (X_2, Y_2, Z_2)$	
23. If $2P = O$, then return $(X_1^{(1)}, Y_1^{(1)}, Z_2)$	
24. If $(X_1^{(1)}, Y_1^{(1)}, Z_2) = O$, then return (X_2, Y_2, Z_2)	
25. $T_3 = T_3 \times T_4$	$\{Z_3 = Z_2(X_1^{(1)} - X_2)\}$
26. $T_2 = T_2 - T_5$	$\{Y_1^{(1)} - Y_2\}$
27. $T_4 = T_6^2$	$\{(X_1^{(1)} - X_2)^2\}$
28. $T_6 = T_4 \times T_6$	$\{(X_1^{(1)} - X_2)^3\}$
29. $T_5 = T_5 \times T_6$	$\{Y_2^{(1)} = Y_2(X_1^{(1)} - X_2)^3\}$
30. $T_4 = T_1 \times T_4$	$\{X_2^{(1)} = X_2(X_1^{(1)} - X_2)^2\}$
31. $T_1 = T_2^2$	$\{(Y_1^{(1)} - Y_2)^2\}$
32. $T_1 = T_1 - T_6$	$\{(Y_1^{(1)} - Y_2)^2 - (X_1^{(1)} - X_2)^3\}$
33. $T_6 = 2T_4$	$\{2X_2(X_1^{(1)} - X_2)^2\}$
34. $T_1 = T_1 - T_6$	$\{X_3\}$
35. $T_6 = T_4 - T_1$	$\{X_2^{(1)} - X_3\}$
36. $T_2 = T_2 \times T_6$	$\{(Y_1^{(1)} - Y_2)(X_2^{(1)} - X_3)\}$
37. $T_2 = T_2 - T_5$	$\{Y_3 = (Y_1^{(1)} - Y_2)(X_2^{(1)} - X_3) - Y_2^{(1)}\}$
38. If $d = 3$, then return $3P = (T_1, T_2, T_3) = (X_3, Y_3, Z_3)$	
39. For $j = 1$ to $(d-3)/2$ do:	
40. If $(X_{j+2}, Y_{j+2}, Z_{j+2}) = O$, then return $(X_2^{(j)}, Y_2^{(j)}, Z_2^{(j)})$	
41. If $(X_2^{(j)}, Y_2^{(j)}, Z_2^{(j)}) = O$, then return $(X_{j+2}, Y_{j+2}, Z_{j+2})$	
39. $T_6 = -T_6$	$\{X_{j+2} - X_2^{(j)}\}$
40. $T_3 = T_1 \times T_3$	$\{Z_{j+3}\}$
41. $T_2 = T_2 - T_5$	$\{Y_{j+2} - Y_2^{(j)}\}$
42. $T_6 = T_1^2$	$\{(X_{j+2} - X_2^{(j)})^2\}$
43. $T_1 = T_1 \times T_6$	$\{(X_{j+2} - X_2^{(j)})^3\}$
44. $T_4 = T_4 \times T_6$	$\{X_2^{(j+1)} = X_2^{(j)}(X_{j+2} - X_2^{(j)})^2\}$
45. $T_6 = 2T_4$	$\{2X_2^{(j)}(X_{j+2} - X_2^{(j)})^2\}$
46. $T_5 = T_1 \times T_5$	$\{Y_2^{(j+1)} = Y_2^{(j)}(X_{j+2} - X_2^{(j)})^3\}$
47. $T_1 = T_1 + T_6$	$\{(X_{j+2} - X_2^{(j)})^3 + 2X_2^{(j)}(X_{j+2} - X_2^{(j)})^2\}$
48. $T_6 = T_2^2$	$\{(Y_{j+2} - Y_2^{(j)})^2\}$
49. $T_1 = T_6 - T_1$	$\{X_{j+3}\}$
50. $T_6 = T_4 - T_1$	$\{X_2^{(j+1)} - X_{j+3}\}$
51. $T_6 = T_2 \times T_6$	$\{(Y_{j+2} - Y_2^{(j)})(X_2^{(j+1)} - X_{j+3})\}$
52. $T_2 = T_2 - T_5$	$\{Y_{j+3}\}$
53. Return $dP = (T_1, T_2, T_3) = (X_{(d+3)/2}, Y_{(d+3)/2}, Z_{(d+3)/2})$	

The cost of performing a doubling with Algorithm 1 is $3M + 5S + 8A + 1(x2) + 1(x3) + 1(x4) + 1(x8)$. It requires only 6 registers to perform operations, including permanent registers to allocate input coordinates.

In the case of tripling, the cost is fixed to $8M + 7S + 13A + 2(x2) + 1(x3) + 1(x4) + 1(x8)$. Each extra addition with $2P$ to yield quintupling ($5P$), septupling ($7P$) and higher order operations, requires $5M +$

$2S + 5A + 1(x2)$. For instance, the cost of quintupling is given by $[8M + 7S + 13A + 2(x2) + 1(x3) + 1(x4) + 1(x8)] + [5M + 2S + 5A + 1(x2)] = 13M + 9S + 18A + 3(x2) + 1(x3) + 1(x4) + 1(x8)$. Similarly to doubling, composite operations have a requirement of only 6 registers in total.

APPENDIX A2: POINT MIXED ADDITION

The following pseudocode to compute a mixed addition uses the fast formula presented in [LM07a].

Algorithm 2: Point Mixed Addition, $\mathcal{J} + \mathcal{A} \rightarrow \mathcal{J}$, $E: y^2 = x^3 + ax + b$

INPUT: points $P = (X_1, Y_1, Z_1)$ and $Q = (x_2, y_2)$ on $E(\mathbb{F}_p)$, $T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z_1$, $T_x \leftarrow x_2$, $T_y \leftarrow y_2$

OUTPUT: point $P + Q = (X_3, Y_3, Z_3)$

1.	If $Q = O$, then return (X_1, Y_1, Z_1)	15.	$T_3 = T_3 - T_6$ $\{Z_3\}$
2.	If $P = O$, then return $(x_2, y_2, 1)$	16.	$T_6 = 8T_6$ $\{8\beta^2\}$
3.	$T_4 = T_3^2$ $\{Z_1^2\}$	17.	$T_5 = T_5 \times T_6$ $\{8\beta^3\}$
4.	$T_5 = T_x \times T_4$ $\{Z_1^2 x_2\}$	18.	$T_6 = T_1 \times T_6$ $\{8X_1\beta^2\}$
5.	$T_5 = T_5 - T_1$ $\{\beta = Z_1^2 x_2 - X_1\}$	19.	$T_2 = T_2 \times T_5$ $\{8Y_1\beta^3\}$
6.	$T_6 = T_3 + T_5$ $\{Z_1 + \beta\}$	20.	$T_5 = T_5 / 2$ $\{4\beta^3\}$
7.	$T_6 = T_6^2$ $\{(Z_1 + \beta)^2\}$	21.	$T_5 = T_5 + T_6$ $\{4\beta^3 + 8X_1\beta^2\}$
8.	$T_6 = T_6 - T_4$ $\{(Z_1 + \beta)^2 - Z_1^2\}$	22.	$T_1 = T_4^2$ $\{\alpha^2\}$
9.	$T_4 = T_3 \times T_4$ $\{Z_1^3\}$	23.	$T_1 = T_1 - T_5$ $\{X_3\}$
10.	$T_4 = T_y \times T_4$ $\{Z_1^3 y_2\}$	24.	$T_6 = T_6 / 2$ $\{4X_1\beta^2\}$
11.	$T_4 = T_4 - T_2$ $\{Z_1^3 y_2 - Y_1\}$	25.	$T_6 = T_6 - T_1$ $\{4X_1\beta^2 - X_3\}$
12.	$T_4 = 2T_4$ $\{\alpha = 2(Z_1^3 y_2 - Y_1)\}$	26.	$T_4 = T_4 \times T_6$ $\{\alpha(4X_1\beta^2 - X_3)\}$
13.	$T_3 = T_6$ $\{(Z_1 + \beta)^2 - Z_1^2\}$	27.	$T_2 = T_4 - T_2$ $\{Y_3\}$
14.	$T_6 = T_5^2$ $\{\beta^2\}$	28.	Return $P + Q = (T_1, T_2, T_3) = (X_3, Y_3, Z_3)$

The total cost of the mixed addition as given by Algorithm 2 is $7M + 4S + 9A + 1(x2) + 1(x8) + 2(\div 2)$. It requires only 6 registers to perform operations without considering storage for precomputed points (x_2, y_2).

APPENDIX A3: POINT DOUBLING-ADDITION

The following pseudocode corresponds to the doubling-addition (DA) formula presented in [LM07b].

Algorithm 3: Point Doubling-Addition, $\mathcal{J}+\mathcal{A}\rightarrow\mathcal{J}$, $E: y^2=x^3+ax+b$

INPUT: points $P=(X_1,Y_1,Z_1)$ and $Q=(x_2,y_2)$ on $E(\mathbb{F}_p)$, $T_1\leftarrow X_1$, $T_2\leftarrow Y_1$, $T_3\leftarrow Z_1$, $T_x\leftarrow x_2$, $T_y\leftarrow y_2$

OUTPUT: point $2P+Q=(X_3,Y_3,Z_3)$

1.	If $Q = O$, then compute Algorithm 1 ($d = 2$)		
2.	If $P = O$, then return Q		
3.	$T_4=T_3^2$	$\{Z_1^2\}$	
4.	$T_5=T_x\times T_4$	$\{Z_1^2x_2\}$	
5.	$T_5=T_5-T_1$	$\{\beta\}$	
6.	$T_6=T_3+T_5$	$\{Z_1+\beta\}$	
7.	$T_6=T_6^2$	$\{(Z_1+\beta)^2\}$	
8.	$T_6=T_6-T_4$	$\{(Z_1+\beta)^2-Z_1^2\}$	
9.	$T_4=T_3\times T_4$	$\{Z_1^3\}$	
10.	$T_4=T_y\times T_4$	$\{Z_1^3y_2\}$	
11.	$T_4=T_4-T_2$	$\{\alpha\}$	
12.	$T_3=T_5^2$	$\{\beta^2\}$	
13.	$T_6=T_6-T_3$	$\{Z_1^{(1)}\}$	
14.	$T_1=T_1\times T_3$	$\{X_1\beta^2\}$	
15.	$T_1=4T_1$	$\{X_1^{(1)}\}$	
16.	$T_3=T_3\times T_5$	$\{\beta^3\}$	
17.	$T_2=T_2\times T_3$	$\{Y_1\beta^3\}$	
18.	$T_2=8T_2$	$\{Y_1^{(1)}\}$	
19.	$T_5=T_4^2$	$\{\alpha^2\}$	
20.	$T_3=T_5-T_3$	$\{\alpha^2-\beta^3\}$	
21.	$T_3=4T_3$	$\{4\alpha^2-4\beta^3\}$	
22.	$T_3=T_3-T_1$	$\{4\alpha^2-4\beta^3-X_1^{(1)}\}$	
23.	$T_3=T_3-T_1$	$\{4\alpha^2-4\beta^3-2X_1^{(1)}\}$	
24.	$T_3=T_3-T_1$	$\{\theta\}$	
25.	$T_4=T_3+T_4$	$\{\alpha+\theta\}$	
26.	$T_4=T_4^2$	$\{(\alpha+\theta)^2\}$	
27.	$T_4=T_5-T_4$	$\{\alpha^2-(\alpha+\theta)^2\}$	
28.	$T_4=T_4-T_2$	$\{\alpha^2-Y_1^{(1)}-(\alpha+\theta)^2\}$	
29.	$T_4=T_4-T_2$	$\{\alpha^2-2Y_1^{(1)}-(\alpha+\theta)^2\}$	
30.	$T_5=T_3^2$	$\{\theta^2\}$	
31.	$T_4=T_4+T_5$	$\{\omega\}$	
32.	$T_1=T_1\times T_5$	$\{X_1^{(1)}\theta^2\}$	
33.	$T_5=T_3\times T_5$	$\{\theta^3\}$	
34.	$T_3=T_3\times T_6$	$\{Z_3\}$	
35.	$T_2=T_2\times T_5$	$\{Y_1^{(1)}\theta^3\}$	
36.	$T_5=3T_1$	$\{3X_1^{(1)}\theta^2\}$	
37.	$T_6=T_4^2$	$\{\omega^2\}$	
38.	$T_6=T_6-T_5$	$\{\omega^2-\theta^3\}$	
39.	$T_5=T_5-T_6$	$\{3X_1^{(1)}\theta^2-\omega^2+\theta^3\}$	
40.	$T_4=T_4\times T_5$	$\{\omega[3X_1^{(1)}\theta^2-\omega^2+\theta^3]\}$	
41.	$T_2=T_4-T_2$	$\{Y_3\}$	
42.	$T_1=T_1-T_5$	$\{X_3\}$	
43.	Return $2P+Q=(T_1,T_2,T_3)=(X_3,Y_3,Z_3)$		

The doubling-addition according to Algorithm 3 costs $11M + 7S + 18A + 1(x3) + 2(x4) + 1(x8)$, and requires only 6 registers to perform operations without considering storage for precomputed points (x_2, y_2).

APPENDIX B: POINT TRIPLING

The following pseudocode corresponds to the fast tripling formula presented in [LM07a].

Algorithm 4: Point Tripling (Jacobian Coordinates), $E: \quad y^2 = x^3 - 3x + b$

INPUT: point $P = (X_1, Y_1, Z_1)$ on $E(\mathbb{F}_p)$, $T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z_1$

OUTPUT: point $3P = (X_3, Y_3, Z_3)$

1.	If $P = O$, then return (O)	
2.	$T_4 = T_3^2$	$\{Z_1^2\}$
3.	$T_5 = T_1 + T_4$	$\{X_1 + Z_1^2\}$
4.	$T_6 = T_1 - T_4$	$\{X_1 - Z_1^2\}$
5.	$T_5 = T_5 \times T_6$	$\{(X_1 + Z_1^2)(X_1 - Z_1^2)\}$
6.	$T_5 = 3T_5$	$\{\theta = 3(X_1 + Z_1^2)(X_1 - Z_1^2)\}$
7.	$T_2 = 2T_2$	$\{2Y_1\}$
8.	$T_6 = T_2^2$	$\{4Y_1^2\}$
9.	$T_7 = T_1 \times T_6$	$\{4X_1Y_1^2\}$
10.	$T_7 = 3T_7$	$\{12X_1Y_1^2\}$
11.	$T_8 = T_5^2$	$\{\theta^2\}$
12.	$T_7 = T_7 - T_8$	$\{\omega = 12X_1Y_1^2 - \theta^2\}$
13.	$T_5 = T_5 + T_7$	$\{\theta + \omega\}$
14.	$T_5 = T_5^2$	$\{(\theta + \omega)^2\}$
15.	$T_5 = T_5 - T_8$	$\{(\theta + \omega)^2 - \theta^2\}$
16.	$T_3 = T_3 + T_7$	$\{Z_1 + \omega\}$
17.	$T_3 = T_3^2$	$\{(Z_1 + \omega)^2\}$
18.	$T_3 = T_3 - T_4$	$\{(Z_1 + \omega)^2 - Z_1^2\}$
19.	$T_4 = T_7^2$	$\{\omega^2\}$
20.	$T_3 = T_3 - T_4$	$\{Z_3\}$
21.	$T_5 = T_5 - T_4$	$\{2\alpha\}$
22.	$T_1 = T_1 \times T_4$	$\{X_1\omega^2\}$
23.	$T_4 = T_4 \times T_7$	$\{\omega^3\}$
24.	$T_7 = T_6^2$	$\{2\beta = 16Y_1^4\}$
25.	$T_5 = T_7 - T_5$	$\{2\beta - 2\alpha\}$
26.	$T_6 = T_5 \times T_6$	$\{4Y_1^2(2\beta - 2\alpha)\}$
27.	$T_1 = T_1 + T_6$	$\{4Y_1^2(2\beta - 2\alpha) + X_1\omega^2\}$
28.	$T_1 = 4T_1$	$\{X_3\}$
29.	$T_7 = T_5 + T_7$	$\{4\beta - 2\alpha\}$
30.	$T_5 = T_7 \times (-T_5)$	$\{(2\alpha - 2\beta)(4\beta - 2\alpha)\}$
31.	$T_5 = T_5 - T_4$	$\{(2\alpha - 2\beta)(4\beta - 2\alpha) - \omega^3\}$
32.	$T_2 = 4T_2$	$\{8Y_1\}$
33.	$T_2 = T_2 \times T_5$	$\{Y_3\}$
34.	Return $(T_1, T_2, T_3) = (X_3, Y_3, Z_3)$	

The total cost of the tripling as given by Algorithm 4 is 7*M* + 7*S* + 13*A* + 1(x2) + 2(x3) + 2(x4). It requires 8 registers to perform operations.

APPENDIX C: NEW POINT QUINTUPLING

The following formula allows the computation of the quintupling $5P = (X_5, Y_5, Z_5)$ of a point $P = (X_1, Y_1, Z_1)$:

$$X_5 = 4(X_1\gamma^2 - \rho\varphi), \quad Y_5 = 8Y_1[\gamma\omega^3(3\alpha^2 - \gamma) - \alpha^4(\omega^3 + \alpha\beta)], \quad Z_5 = (Z_1 + \gamma)^2 - Z_1^2 - \gamma^2,$$

where $\alpha = (\theta + \omega)^2 - (\theta^2 + \omega^2 + \beta)$, $\beta = 16Y_1^4$, $\theta = 3(X_1 + Z_1^2)(X_1 - Z_1^2)$, $\omega = 12X_1Y_1^2 - \theta^2$,
$\gamma = \alpha\beta - \omega^3$, $\phi = \gamma - \alpha^2$, $\varphi = \omega.\phi$, $\rho = 2[(Y_1^2 + \alpha)^2 - Y_1^4 - \alpha^2]$.

The validity of this formula has been verified with Magma. Its cost is $11M + 11S$ when considering the special case $a = -3$. The general case (parameter a with any value in the field) can be easily derived from the formula above. In this case, the cost is fixed at $10M + 14S$.

Following, we present the pseudocode for the quintupling formula (special case $a = -3$), including memory allocation.

Algorithm 5: Point Quintupling, $(5)\mathcal{J} \rightarrow \mathcal{J}$, $E: y^2 = x^3 - 3x + b$

INPUT: point $P = (X_1, Y_1, Z_1)$ on $E(\mathbb{F}_p)$, $T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z_1$

OUTPUT: point $5P = (X_5, Y_5, Z_5)$

1.	If $P = O$, then return (X_1, Y_1, Z_1)		26.	$T_6 = 2T_6$	$\{\rho\}$
2.	$T_4 = T_3^2$	$\{Z_1^2\}$	27.	$T_5 = T_5 \times T_{10}$	$\{\alpha.\beta\}$
3.	$T_5 = T_1 + T_4$	$\{X_1 + Z_1^2\}$	28.	$T_{10} = T_5 - T_8$	$\{\gamma\}$
4.	$T_6 = T_1 - T_4$	$\{X_1 - Z_1^2\}$	29.	$T_3 = T_3 + T_{10}$	$\{Z_1 + \gamma\}$
5.	$T_5 = T_5 \times T_6$	$\{(X_1 + Z_1^2)(X_1 - Z_1^2)\}$	30.	$T_3 = T_3^2$	$\{(Z_1 + \gamma)^2\}$
6.	$T_5 = 3T_5$	$\{\theta\}$	31.	$T_3 = T_3 - T_4$	$\{(Z_1 + \gamma)^2 - Z_1^2\}$
7.	$T_6 = T_2^2$	$\{Y_1^2\}$	32.	$T_4 = T_{10} - T_9$	$\{\phi\}$
8.	$T_7 = T_1 \times T_6$	$\{X_1Y_1^2\}$	33.	$T_7 = T_7 \times T_4$	$\{\varphi\}$
9.	$T_7 = 12T_7$	$\{12X_1Y_1^2\}$	34.	$T_6 = T_6 \times T_7$	$\{\rho.\varphi\}$
10.	$T_8 = T_5^2$	$\{\theta^2\}$	35.	$T_4 = T_{10}^2$	$\{\gamma^2\}$
11.	$T_7 = T_7 - T_8$	$\{\omega\}$	36.	$T_3 = T_3 - T_4$	$\{Z_5\}$
12.	$T_5 = T_5 + T_7$	$\{\theta + \omega\}$	37.	$T_1 = T_1 \times T_4$	$\{X_1\gamma^2\}$
13.	$T_5 = T_5^2$	$\{(\theta + \omega)^2\}$	38.	$T_1 = T_1 - T_6$	$\{X_1\gamma^2 - \rho.\varphi\}$
14.	$T_5 = T_5 - T_8$	$\{(\theta + \omega)^2 - \theta^2\}$	39.	$T_1 = 4T_1$	$\{X_5\}$
15.	$T_8 = T_7^2$	$\{\omega^2\}$	40.	$T_5 = T_5 + T_8$	$\{\omega^3 + \alpha.\beta\}$
16.	$T_5 = T_5 - T_8$	$\{(\theta + \omega)^2 - \theta^2 - \omega^2\}$	41.	$T_4 = T_9^2$	$\{\alpha^4\}$
17.	$T_8 = T_7 \times T_8$	$\{\omega^3\}$	42.	$T_4 = T_4 \times T_5$	$\{\alpha^4(\omega^3 + \alpha.\beta)\}$
18.	$T_9 = T_6^2$	$\{Y_1^4\}$	43.	$T_9 = 3T_9$	$\{3\alpha^2\}$
19.	$T_{10} = 16T_9$	$\{\beta = 16Y_1^4\}$	44.	$T_9 = T_9 - T_{10}$	$\{3\alpha^2 - \gamma\}$
20.	$T_5 = T_5 - T_{10}$	$\{\alpha\}$	45.	$T_9 = T_8 \times T_9$	$\{\omega^3(3\alpha^2 - \gamma)\}$
21.	$T_6 = T_5 + T_6$	$\{Y_1^2 + \alpha\}$	46.	$T_9 = T_9 \times T_{10}$	$\{\gamma\omega^3(3\alpha^2 - \gamma)\}$
22.	$T_6 = T_6^2$	$\{(Y_1^2 + \alpha)^2\}$	47.	$T_4 = T_9 - T_4$	$\{\gamma\omega^3(3\alpha^2 - \gamma) - \alpha^4(\omega^3 + \alpha.\beta)\}$
23.	$T_6 = T_6 - T_9$	$\{(Y_1^2 + \alpha)^2 - Y_1^4\}$	48.	$T_2 = 8T_2$	$\{8Y_1\}$
24.	$T_9 = T_5^2$	$\{\alpha^2\}$	49.	$T_2 = T_2 \times T_4$	$\{Y_5\}$
25.	$T_6 = T_6 - T_9$	$\{(Y_1^2 + \alpha)^2 - Y_1^4 - \alpha^2\}$	50.	Return $5P = (T_1, T_2, T_3) = (X_5, Y_5, Z_5)$	

The cost of performing a quintupling with Algorithm 5 is $11M + 11S + 19A + 1(x2) + 2(x3) + 1(x4) + 1(x8) + 1(x12) + 1(x16)$. It requires 10 registers to perform operations, including permanent registers to allocate input/output coordinates.

APPENDIX D1: ATOMIC POINT DOUBLING (*M-N-A-M-N-A-A*-BASED)

Input: $P = (X_1 , Y_1 , Z_1)$ Output: $2P = (X_3 , Y_3 , Z_3)$ $T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z_1$

Δ1	Δ2	Δ3	Δ4
$T_4 = T_3^2$ (Z_1^2)	$T_3 = T_2^2$ (Y_1^2)	$T_1 = T_4^2$ (α^2)	$T_3 = T_2^2$ $(4Y_1^4)$
*	*	*	$T_2 = -T_2$ $(-4Y_1^4)$
$T_5 = T_1 + T_4$ $(A = X_1 + Z_1^2)$	$T_2 = T_2 + T_2$ $(2Y_1^2)$	$T_5 = T_5 + T_5$ $(-2X_1)$	$T_5 = T_1 + T_5$ $(X_3 - \beta)$
$T_3 = T_2 \times T_3$ $(Y_1 Z_1)$	$T_4 = T_4 \times T_5$ $(A.B)$	$T_5 = T_2 \times T_5$ $(-\beta)$	$T_5 = T_4 \times T_5$ $(\alpha(X_3 - \beta))$
$T_4 = -T_4$ $(-Z_1^2)$	$T_5 = -T_1$ $(-X_1)$	*	$T_5 = -T_5$ $(\alpha(\beta - X_3))$
$T_4 = T_1 + T_4$ $(B = X_1 - Z_1^2)$	$T_4 = T_4 + T_4$ $(2A.B)$	$T_1 = T_1 + T_5$ $(\alpha^2 - \beta)$	$T_2 = T_2 + T_2$ $(-8Y_1^4)$
$T_3 = T_3 + T_3$ (Z_3)	$T_4 = T_4 + T_4$ (α)	$T_1 = T_1 + T_5$ (X_3)	$T_2 = T_2 + T_5$ (Y_3)

For the remainder of this paper, "*" represents a dummy field operation that depends on the step it is placed. For instance, the three "*" symbols in the second step of blocks Δ1 – 3 of the table given above represent dummy negations.

The cost of performing an atomic doubling is $8M + 12A$. It only requires 5 registers to perform operations, including permanent registers to allocate input/output coordinates.

APPENDIX D2: ATOMIC MIXED ADDITION AND DOUBLING-ADDITION (*M-N-A-M-N-A-A*-BASED)

Input: $P = (X_1, Y_1, Z_1)$ and $Q = (x_2 , y_2)$

Output: in the case of mixed addition, $P + Q = (X_3, Y_3, Z_3)$; in the case of DA, $2P + Q = (X_4, Y_4, Z_4)$

$T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z_1$, $T_x \leftarrow X_2$, $T_y \leftarrow Y_2$

Δ1	Δ2	Δ3	Δ4	Δ5
$T_4 = T_3^2$ (Z_1^2)	$T_4 = T_3 \times T_4$ (Z_1^3)	$T_3 = T_3 \times T_5$ (Z_3)	$T_6 = T_5 \times T_6$ (β^3)	$T_5 = T_7 \times T_5$ $(-Y_1^{(1)} = -Y_1\beta^3)$
*	*	*	$T_1 = -T_1$ (X_1)	$T_1 = -T_1$ $(-2X_1\beta^2 - \beta^3)$
*	*	*	*	*
$T_5 = T_x \times T_4$ $(Z_1^2 X_2)$	$T_4 = T_y \times T_4$ $(Z_1^3 Y_2)$	$T_6 = T_5^2$ (β^2)	$T_6 = T_1 \times T_6$ $(X_1^{(1)} = X_1\beta^2)$	$T_2 = T_4^2$ (α^2)
$T_1 = -T_1$ $(-X_1)$	$T_2 = -T_2$ $(-Y_1)$	$T_4 = -T_4$ $(-\alpha)$	*	$T_6 = -T_6$ $(-X_1^{(1)} = -X_1\beta^2)$
$T_5 = T_1 + T_5$ $(\beta = Z_1^2 X_2 - X_1)$	$T_4 = T_2 + T_4$ (α)	*	$T_1 = T_6 + T_6$ $(2X_1\beta^2)$	$T_1 = T_1 + T_2$ (X_3)
*	*	*	$T_1 = T_1 + T_5$ $(2X_1\beta^2 + \beta^3)$	$T_2 = T_1 + T_6$ $(X_3 - X_1^{(1)})$

Δ6	Δ7 (a)	Δ8 (a)	Δ9 (a)
$T_3 = T_3 \times T_4$ $(\alpha(X_1^{(1)} - X_3))$	$T_1 = T_4^2$ (θ^2)	$T_6 = T_1 \times T_6$ $(X_1^{(1)}\theta^2)$	$T_5 = T_4 \times T_5$ $(Y_1^{(1)}\theta^3)$
*	*	*	$T_7 = -T_1$ $(-X_4)$
$T_4 = T_1 + T_6$ $(\theta = X_3 - X_1^{(1)})$ (a)	*	$T_7 = T_6 + T_6$ $(2X_1^{(1)}\theta^2)$	$T_4 = T_6 + T_7$ $(X_1^{(1)}\theta^2 - X_4)$
$T_3 = T_3 \times T_4$ (Z_4) (a)	$T_4 = T_1 \times T_4$ (θ^3)	$T_1 = T_2^2$ (ϕ^2)	$T_2 = T_2 \times T_4$ $(\phi \cdot (X_1^{(1)}\theta^2 - X_4))$
*	*	$T_7 = -T_7$ $(-2X_1^{(1)}\theta^2)$	*
$T_2 = T_2 + T_5$ (Y_3)	*	$T_1 = T_1 + T_7$ $(\phi^2 - 2X_1^{(1)}\theta^2)$	$T_2 = T_2 + T_5$ (Y_4)
$T_2 = T_2 + T_5$ $(\phi = Y_3 - Y_1^{(1)})$ (a)	*	$T_1 = T_1 + T_4$ (X_4)	*

(a) Field operations are computed for the atomic doubling-addition.

The atomic mixed addition consists of atomic blocks Δ1 – 6. To perform an atomic doubling-addition, operations marked with (a) in Δ6 and atomic blocks Δ7 – 9 should be also included.

The costs of performing an atomic mixed addition and doubling-addition are $12M + 18A$ and $18M + 27A$, respectively. For the former, we require only 6 registers to perform computations. One extra register is necessary for the doubling-addition version, making a total requirement of 7 registers in this case.

The previous register counting considers registers to allocate input/output coordinates but does not include registers for storing precomputed points (x_2, y_2).

APPENDIX D3: ATOMIC COMPOSITE OPERATIONS OF THE FORM *dP* (*M-N-A-M-N-A-A*-BASED)

Input: $P = (X_1 , Y_1 , Z_1)$ Output: $dP = (X_{(d+3)/2}, Y_{(d+3)/2}, Z_{(d+3)/2})$

$T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z_1$

Δ1	Δ2	Δ3	Δ4
$T_4 = T_3^2$ (Z_1^2)	$T_4 = T_4 \times T_5$ $(A \cdot B)$	$T_1 = T_1 \times T_6$ $(2X_1Y_1^2)$	$T_6 = T_6^2$ $(4Y_1^4)$
*	*	*	$T_5 = -T_1$ $(-X_2)$
$T_3 = T_3 + T_3$ $(2Z_1)$	$T_5 = T_4 + T_4$ $(2A \cdot B)$	$T_2 = T_1 + T_1$ $(X_1^{(1)} = 4X_1Y_1^2)$	$T_5 = T_2 + T_5$ $(X_1^{(1)} - X_2)$
$T_3 = T_2 \times T_3$ (Z_2)	$T_6 = T_2^2$ (Y_1^2)	$T_1 = T_4^2$ $(3A \cdot B)^2$	$T_4 = T_4 \times T_5$ $((3A \cdot B)(X_1^{(1)} - X_2))$
$T_5 = -T_4$ $(-Z_1^2)$	*	$T_5 = -T_2$ $(-X_1^{(1)})$	$T_6 = -T_6$ $(-4Y_1^4)$
$T_5 = T_1 + T_5$ $(A = X_1 - Z_1^2)$	$T_4 = T_4 + T_5$ $(3A \cdot B)$	$T_5 = T_5 + T_5$ $(-2X_1^{(1)})$	$T_6 = T_6 + T_6$ $(-Y_1^{(1)})$
$T_4 = T_1 + T_4$ $(B = X_1 + Z_1^2)$	$T_6 = T_6 + T_6$ $(2Y_1^2)$	$T_1 = T_1 + T_5$ (X_2)	$T_2 = T_4 + T_6$ (Y_2)
Δ5	**Δ6**	**Δ7**	**Δ8**
$T_3 = T_3 \times T_5$ (Z_3)	$T_5 = T_4 \times T_5$ $((X_1^{(1)} - X_2)^3)$	$T_5 = T_2 \times T_5$ $(Y_2^{(1)})$	$T_2 = T_2 \times T_6$ $((Y_1^{(1)} - Y_2)(X_2^{(1)} - X_3))$
*	*	$T_4 = -T_4$ $(-X_2^{(1)})$	*
$T_6 = T_2 + T_6$ $(Y_2 - Y_1^{(1)})$	*	*	*
$T_4 = T_5^2$ $((X_1^{(1)} - X_2)^2)$	$T_4 = T_1 \times T_4$ $(X_2^{(1)})$	$T_2 = T_6^2$ $((Y_1^{(1)} - Y_2)^2)$	*
*	*	$T_1 = -T_1$ $(-(X_1^{(1)} - X_2)^3 - 2X_2^{(1)})$	$T_6 = -T_5$ $(-Y_2^{(1)})$
*	$T_1 = T_4 + T_4$ $(2X_2^{(1)})$	$T_1 = T_1 + T_2$ (X_3)	$T_2 = T_2 + T_6$ (Y_3)
*	$T_1 = T_1 + T_5$ $((X_1^{(1)} - X_2)^3 + 2X_2^{(1)})$	$T_2 = T_1 + T_4$ $(X_3 - X_2^{(1)})$	*

Δ1′	Δ2′	Δ3′	Δ4′
*	$T_6 = T_1^2$ $(X_{j+2} - X_2^{(j)})^2$	$T_6 = T_1 \times T_6$ $(X_{j+2} - X_2^{(j)})^3$	$T_2 = T_2 \times T_7$ $((Y_{j+2} - Y_2^{(j)})(X_{j+3} - X_2^{(j+1)}))$
*	*	$T_6 = -T_6$ $(-C = -(X_{j+2} - X_2^{(j)})^3)$	$T_2 = -T_2$ $((Y_{j+2} - Y_2^{(j)})(X_2^{(j+1)} - X_{j+3}))$
$T_1 = T_1 + T_4$ $(X_{j+2} - X_2^{(j)})$	*	$T_1 = T_6 + T_7$ $(-C - 2X_2^{(j+1)})$	*
$T_3 = T_1 \times T_3$ (Z_{j+3})	$T_4 = T_4 \times T_6$ $(-X_2^{(j+1)})$	$T_7 = T_2^2$ $(Y_{j+2} - Y_2^{(j)})^2$	$T_6 = T_5 \times T_6$ $(-Y_2^{(j+1)})$
*	*	*	$T_5 = -T_6$ $(Y_2^{(j+1)})$
*	$T_7 = T_4 + T_4$ $(-2X_2^{(j+1)})$	$T_1 = T_1 + T_7$ (X_{j+3})	$T_2 = T_2 + T_6$ (Y_{j+3})
$T_2 = T_2 + T_6$ $(Y_{j+2} - Y_2^{(j)})$	*	$T_7 = T_1 + T_4$ $(X_{j+3} - X_2^{(j+1)})$	*

The atomic tripling consists of blocks Δ1 – 8. To perform atomic composite operations *dP* with $d > 3$, we should execute blocks Δ1′ – 4′ for $j = 1$ to $(d–3)/2$.

The atomic tripling costs $16M + 24A$ and requires 6 registers to perform computations, including registers to allocate input/output coordinates. In the case of higher order composite operations, it is possible to reduce the cost further by merging blocks Δ8 and Δ1′ for $j = 1$. For instance, quintupling requires 11 blocks (i.e., $22M + 33A$); and septupling, 15 (i.e., $30M + 45A$). The memory requirement for these atomic composite operations is fixed to 7 registers to perform computations.

APPENDIX E: ATOMIC POINT TRIPLING (*M-N-A-M-N-A-A*-BASED)

Input: $P = (X_1 , Y_1 , Z_1)$ Output: $3P = (X_3 , Y_3 , Z_3)$

$T_1 \leftarrow X_1, T_2 \leftarrow Y_1, T_3 \leftarrow Z_1$

Δ1	Δ2	Δ3	Δ4
$T_4 = T_3^2 \quad (Z_1^2)$	$T_7 = T_1 \times T_9 \quad (4X_1Y_1^2)$	$T_4 = T_5^2 \quad (\theta^2)$	$T_5 = T_5 \times T_9 \quad (\alpha)$
*	*	*	$T_5 = -T_5 \quad (-\alpha)$
$T_5 = -T_4 \quad (-Z_1^2)$	$T_4 = T_1 + T_4 \quad (A = X_1 + Z_1^2)$	$T_8 = T_7 + T_7 \quad (8X_1Y_1^2)$	$T_6 = T_6 + T_6 \quad (\beta)$
$T_5 = T_1 + T_5 \quad (B = X_1 - Z_1^2)$	$T_4 = T_4 \times T_5 \quad (A.B)$	$T_6 = T_6^2 \quad (4Y_1^4)$	$T_3 = T_3 \times T_7 \quad (Z_3)$
$T_6 = T_2^2 \quad (Y_1^2)$	*	$T_4 = -T_4 \quad (-\theta^2)$	*
*	$T_5 = T_4 + T_4 \quad (2A.B)$	$T_7 = T_7 + T_8 \quad (12X_1Y_1^2)$	$T_5 = T_6 + T_5 \quad (\beta - \alpha)$
$T_6 = T_6 + T_6 \quad (2Y_1^2)$	$T_5 = T_4 + T_5 \quad (\theta = 3A.B)$	$T_7 = T_4 + T_7 \quad (\omega)$	$T_6 = T_6 + T_5 \quad (C = 2\beta - \alpha)$
$T_9 = T_6 + T_6 \quad (4Y_1^2)$			

Δ5	Δ6	Δ7
$T_4 = T_7^2 \quad (\omega^2)$	$T_1 = T_1 \times T_4 \quad (X_1\omega^2)$	$T_6 = T_6 \times T_5 \quad (4C.D)$
*	*	$T_6 = T_6 + T_4 \quad (4C.D - \omega^3)$
$T_9 = T_9 + T_9 \quad (8Y_1^2)$	$T_1 = T_1 + T_9 \quad (X_3)$	$T_2 = T_2 \times T_6 \quad (Y_3)$
$T_9 = T_5 \times T_9 \quad (8Y_1^2(\beta - \alpha))$	$T_4 = T_4 \times T_7 \quad (\omega^3)$	*
$T_5 = -T_5 \quad (D = \alpha - \beta)$	$T_4 = -T_4 \quad (-\omega^3)$	*
$T_5 = T_5 + T_5 \quad (2D)$	*	*
$T_5 = T_5 + T_5 \quad (4D)$	*	*

The atomic structure given above corresponds to the following tripling formulae given in [LM07a]:

$$X_3 = 8Y_1^2(\beta - \alpha) + X_1\omega^2, \quad Y_3 = Y_1\left[4(\alpha - \beta)(2\beta - \alpha) - \omega^3\right], \quad Z_3 = Z_1\omega,$$

where $\alpha = \theta\omega$, $\beta = 8Y_1^4$, $\theta = 3\left(X_1 + Z_1^2\right)\left(X_1 - Z_1^2\right)$, $\omega = 12X_1Y_1^2 - \theta^2$.

In this case, the atomic tripling costs $14M + 21A$ and requires 9 registers to perform computations, including registers to allocate input/output coordinates.