Accelerating the Scalar Multiplication on Elliptic Curve Cryptosystems over Prime Fields

by

Patrick Longa

Thesis submitted to
The Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa

Abstract

Elliptic curve cryptography (ECC), independently introduced by Koblitz and Miller in the 80's, has attracted increasing attention in recent years due to its shorter key length requirement in comparison with other public-key cryptosystems such as RSA. Shorter key length means reduced power consumption and computing effort, and less storage requirement, factors that are fundamental in ubiquitous portable devices such as PDAs, cellphones, smartcards, and many others. To that end, a lot of research has been carried out to speed-up and improve ECC implementations, mainly focusing on the most important and time-consuming ECC operation: scalar multiplication.

In this thesis, we focus in optimizing such ECC operation at the point and scalar arithmetic levels, specifically targeting standard curves over prime fields. At the point arithmetic level, we introduce *two* innovative methodologies to accelerate ECC formulae: the use of new composite operations, which are built on top of basic point doubling and addition operations; and the substitution of field multiplications by squarings and other cheaper operations. These techniques are efficiently exploited, individually or jointly, in several contexts: to accelerate computation of scalar multiplications, and the computation of pre-computed points for window-based scalar multiplications (up to 30% improvement in comparison with previous best method); to speed-up computations of simple side-channel attack (SSCA)-protected implementations using innovative atomic structures (up to 22% improvement in comparison with scalar multiplication using original atomic structures); and to develop parallel formulae for SIMD-based applications, which are able to execute *three* and *four* operations simultaneously (up to 72% of improvement in comparison with a

sequential scalar multiplication).

At the scalar arithmetic level, we develop new sublinear (in terms of Hamming weight) multibase scalar multiplications based on NAF-like conversion algorithms that are shown to be faster than any previous scalar multiplication method. For instance, proposed multibase scalar multiplications reduce computing times in 10.9% and 25.3% in comparison with traditional NAF for unprotected and SSCA-protected scenarios, respectively. Moreover, our conversion algorithms overcome the problem of converting any integer to multibase representation, solving an open problem that was defined as hard. Thus, our algorithms make the use of multiple bases practical for applications as ECC scalar multiplication for first time.

Acknowledgements

I would like to thank my wife and family for their support and understanding through all these years. Their love and confidence in me were the inspiration to give the best of myself to this work.

Special gratitude is to Dr. Miri. His faith in my work and capacity, and his constant support and guidance at all levels were priceless factors that helped me to achieve more than I thought possible.

To my wife, Veronica

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

DB	Double-Base Scalar Multiplication
DL	Discrete Logarithm
DSCA	Differential Side-Channel Attack
ECC	Elliptic Curve Cryptosystem
ECDLP	Elliptic Curve Discrete Logarithm Problem
HECC	Hyperelliptic Curve Cryptosystem
*mb*NAF	Multibase Non-Adjacent Form
NAF	Non-Adjacent Form
RSA	Rivest-Shamir-Adleman
SIMD	Single Instruction Multiple Data
SSCA	Simple Side-Channel Attack
*wmb*NAF	Window-*w* Multibase Non-Adjacent Form
*w*NAF	Window-*w* Non-Adjacent Form

Chapter 1

Introduction

1.1 Motivation

Elliptic curve cryptography (ECC) was independently introduced by Koblitz and Miller in 1985. Since then, this public-key cryptosystem has attracted increasing attention due to its shorter key size requirement in comparison with other established systems such as RSA and DL-based cryptosystems. For instance, it is widely accepted that 160-bit ECC offers equivalent security as 1024-bit RSA. This significant difference makes ECC especially attractive for applications on constrained environments as shorter key sizes are translated to less power and storage requirements, and reduced computing times.

Denoted by dP, where d is the secret key (scalar) and P a point on the elliptic curve, the scalar multiplication is the central operation of elliptic curve cryptosystems. The computation of this operation involves *three* mathematical levels: field arithmetic, point arithmetic and scalar arithmetic. Significant effort to optimize ECC operations through each of those levels has been carried out through the last few years. In this work, we concentrate efforts at the point and scalar arithmetic levels, specifically for the case of standard curves [NIST] over prime fields.

ECC point arithmetic involves the efficient execution of doubling and addition operations. At this level, several authors have been working on making those basic point operations as efficient as possible. Although the idea of combining basic operations to build more sophisticated point operations was around years ago, it mainly focused on affine coordinates formulae, which are particularly inefficient because they contain expensive field inversions. Only recently such approach became practical on standard curves over prime fields with the introduction of a tripling operation using inversion-free coordinates [DIM05].

In the scalar arithmetic level, efforts have mainly focused on developing efficient numeric expansions for integers (scalar d in our case) that reduce to the minimum the number of point operations required for the computation of the scalar multiplication. The well-known NAF and wNAF are traditional and efficient examples for the latter. Progress in the point arithmetic level with the introduction of the tripling operation mentioned previously has benefited this area with the appearance of the fastest scalar multiplication known in the literature, which uses radix 3 beside radix 2 (DB scalar multiplication [DIM05]). However, conversion to DB is either slow or require extra memory, drawbacks that are critical for constrained applications.

Beside efforts to speed-up scalar multiplication, there are two additional and important areas of research in ECC: side-channel attacks and efficient implementations on parallel architectures. First, side-channel information, such as power consumption and electromagnetic emission, leaked by electronic devices has been shown to be highly useful for revealing the secret key and effectively breaking public-key cryptosystems. One version of this attack, known as simple side-channel attack (SSCA), is based on the analysis of a single execution trace of the scalar multiplication to reveal the secret key through direct observation of the point operation sequence. More sophisticated attacks, known as differential side-channel attacks (DSCA), can be carried out through the use of statistical analysis on data from several execution traces. To avoid these and new variants of these attacks remains an open and challenging problem.

In particular, side-channel atomicity [CCJ04] has shown to give effective protection

against SSCA at reduced overhead. However, atomic structures, which are the basic blocks used to build SSCA-protected ECC formulae, are still sub-optimal in terms of computing cost and vulnerable to potential attacks that exploit differences between field squarings and multiplications.

Second, the appearance of multiprocessor/parallel architectures, which can execute several operations simultaneously, has become an important topic in recent years since current processor design is reaching its limits in terms of clock frequency. Some works already propose ECC formulae that can execute several operations simultaneously for architectures such as the well-known SIMD. The fastest methods in the literature at the point arithmetic level can execute *two* or *three* field operations in parallel [AHK$^+$01,IT02]. Also, SSCA-protected implementations exist with capability to execute *two* parallel computations [Mis06].

In this thesis, we deal with several of the open challenges described for the previous scenarios. In the following section, we summarize the accomplishments of this work.

1.2 Significance of this Work

Our work focuses on the optimization of the ECC scalar multiplication at point and scalar arithmetic levels for the case of standard curves over prime fields. However, it is important to note that some methodologies are applicable to other areas of cryptography or signal processing. Of particular importance are, for instance, the proposed multibase scalar multiplications, which can be applied to ECC over binary fields, Hyperelliptic Curve Cryptosystems (HECC), pairing-based cryptosystems, and others.

The contributions of this thesis are as follows:

1. Introduction of new composite operations of the form dP and $dP+Q$ that exploit the efficiency of a new special addition with identical z-coordinate [Mel06]. Of

particular interest are the introduction of new composite doubling-addition operation (denoted by DA) and efficient higher order operations such as quintupling (5*P*), septupling (7*P*) and others.

2. Speed-up of traditional and simultaneous scalar multiplication methods by exploiting the efficiency of the new DA operation. For instance, computation time of NAF has been reduced in 3.1% and 22.2% for the case of unprotected and SSCA-protected implementations, respectively.

3. Speed-up of computation of the pre-computed table for window-based scalar multiplications. We achieve up to 30% of improvement in terms of speed in comparison with the best previous method.

4. Introduction of an innovative methodology for accelerating the elliptic curve point formulae over prime fields. This flexible technique uses the substitution of multiplication with squaring and other cheaper operations by exploiting the fact that field squaring is generally less costly than multiplication. By applying this substitution to the traditional formulae, we obtain faster point operations in unprotected sequential implementations. Remarkably, our technique can be efficiently applied to other curve-based cryptosystems to achieve reduced computing cost.

5. Development of new SSCA-protected formulae using innovative atomic structures: *M-N-A-M-N-A-A* and *S-N-A-M-N-A-A*. We achieve up to 22% of improvement in terms of speed in comparison with scalar multiplications using traditional atomic structures. Additionally, *S-N-A-M-N-A-A* gives higher protection by distinguishing squarings from multiplications.

6. Development of the fastest formulae for SIMD-based schemes, which are capable of executing *three* and *four* operations simultaneously. Up to 72% of speed-up can be achieved with our parallel formulae in comparison with a sequential execution.

7. Development of a new parallel SSCA-protected scheme for multiprocessor/parallel

architectures by applying the new atomic structures presented in this work. Our parallel scheme is 24% faster than the best previous method [Mis06].

8. Development of new multibase scalar multiplications based on NAF-like conversion algorithms and the introduced composite operations. Our scalar multiplication methods are shown to exhibit sublinearity in their non-zero density and be faster than any previous method. In unprotected implementations, our method achieves up to 10.9% of improvement in terms of speed in comparison with NAF. In the case of SSCA-protected implementations using atomicity the improvement is as high as 25.7%. Moreover, conversion to multibase does not require extra memory and is relatively faster.

1.3 Thesis Outline

The organization of this work is detailed in the following.

Chapter 2 introduces basic concepts about elliptic curves and abstract algebra. Also, it gives the reader an introduction to each of the arithmetic levels that constitute the scalar multiplication.

In Chapter 3, we introduce our methodology to derive composite operations of the form dP and $dP+Q$. New doubling-addition, tripling-addition, quintupling, septupling and higher order operations are described, and their performance discussed for each case. The chapter ends with the description of two applications for the new operations: computation of pre-computed points and speed-up of traditional/simultaneous scalar multiplication methods.

In Chapter 4, we present our innovative methodology of replacing field multiplication by cheaper operations. Fast doubling, addition and tripling are presented and their performance compared against traditional formulae. Also, we apply this methodology to further reduce computing costs of new composite operations introduced in Chapter 3.

Chapter 5 deals with SSCA-protected implementations. First, we present improved *M-A-N-A*-based operations, and then, develop new atomic formulae on top of innovative atomic structures *M-N-A-M-N-A-A* and *S-N-A-M-N-A-A*. We end the chapter with a performance evaluation of the different structures and a comparison with the traditional approach.

In Chapter 6, we develop new 3- and 4-parallel ECC operations by applying the efficient methodology of substituting multiplications. Performance of these operations is compared with previous works in the case SSCA is not a concern. We then introduce a 2-parallel scheme protected against SSCA using atomicity, and compare it against the best previous efforts in the literature.

In Chapter 7, we present our scalar multiplication methods based on multiple bases and NAF-like expansions of the scalar: multibase (*mb*)-NAF, window-*w* multibase (*wmb*)-NAF and extended window-*w* multibase (*extended wmb*)-NAF scalar multiplications. We end the chapter with a comparison of the costs for the different scalar multiplication methods obtained from extensive tests with random numbers. Two scenarios are discussed: when SSCA is not a concern and when implementations should be protected using an efficient technique as atomicity.

Chapter 8 finishes this thesis with conclusions about our overall work, and makes suggestions for future work.

Chapter 2

Background

In this chapter, we introduce the basic theory behind Elliptic Curve Cryptosystems (ECC), from the abstract algebra of groups and finite fields to the arithmetic layers that constitute the computation of the ECC scalar multiplication.

This is intended as a brief introduction into the topic, which can be complemented by a more extensive review in [HMV04,ACD+05].

2.1 Preliminaries

Before beginning with the description of Elliptic Curve Cryptosystems, we first introduce some necessary concepts about groups and fields.

Groups

[HMV04] A set G is called an *abelian* group $(G,*)$ with a binary operation $* : G \times G \to G$ if it satisfies the following properties:

- Associativity: $(a * b) * c = a * (b * c)$ for all $a, b, c \in \mathrm{G}$.
- Commutativity: $a * b = b * a$ for all $a, b \in \mathrm{G}$.
- Identity: there exists $i \in \mathrm{G}$ such that $a * i = i * a = a$ for all $a \in \mathrm{G}$.
- Inverse: for each $a \in \mathrm{G}$, there exists $b \in \mathrm{G}$ such that $a * b = b * a = i$. Element b is called the inverse of a.

[HMV04] The group is called additive or multiplicative if the binary operation is called addition (+) or multiplication ($\bullet$), respectively. For the first case, the identity element is usually denoted by 0, and the additive inverse of an element a, by $-a$. For the second case, the identity element is usually denoted by 1, and the multiplicative inverse of an element a, by a^{-1}.

If the number of elements in the group is finite with q elements, G is called a finite group with order q.

Finite Fields

We will explain the concept of finite fields through the definition of the finite field $\mathbb{F}_p$ that is used through this work.

[HMV04] First, let's define the next finite group with order prime p, $\mathbb{F}_p$: $\{0,1,\ldots,p-1\}$. Then, $(\mathbb{F}_p,+)$ is an additive finite group of order p and additive identity 0, and $(\mathbb{F}_p^*,\bullet)$ is a multiplicative finite group of order p and multiplicative identity 1, where $\mathbb{F}_p^*$ denotes the non-zero elements in $\mathbb{F}_p$. The triple $(\mathbb{F}_p,+,\bullet)$, simply known as $\mathbb{F}_p$ or *prime field*, is a finite field.

As an extension of the definition of groups, the prime field $\mathbb{F}_p$ is finite since it contains a finite number of elements given by p, which also represents the order of the field. In a more general sense, given a field $\mathbb{F}_q$ of order q, it is said to be a finite field if and only if its order is a prime power $q = p^m$. In particular, we have seen that $\mathbb{F}_q$ is a prime field if $m = 1$.

We will see later that all the points belonging to an elliptic curve over such finite field $\mathbb{F}_p$ will be used to implement the EC-based cryptosystem.

2.2 Introduction to Elliptic Curves

An elliptic curve E over a field K (denoted by $E(K)$) is defined by the general Weierstrass equation [HMV04]:

$$E: \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

Where: $a_1, a_2, a_3, a_4, a_6 \in K$

$\Delta \neq 0$, where Δ is the discriminant of E.

The previous condition guarantees that there do not exist more than *one* tangent line for a given point on the curve, i.e., the curve is "smooth".

The set of pairs (x,y) that solves (2.1) and the point at infinity O, which is the identity for the group law, form an *abelian group* $(E(K),+)$ with binary operation denoted by addition. This group of points is used to implement the Elliptic Curve Cryptosystem (ECC).

We can define ECC over different finite fields K. Most important finite fields used to date to implement this cryptosystem have been binary, prime and extension fields.

In the present thesis, we work with a prime field, denoted by $\mathbb{F}_p$, where p is a large prime and also represents the number of elements of the field. In this case, the general Weierstrass equation simplifies to the following [HMV04]:

$$E: \quad y^2 = x^3 + ax + b \tag{2.2}$$

Where: $a,b \in \mathbb{F}_p$ and $\Delta = 4a^3 + 27b^2 \neq 0$

Consequently, the set of pairs (x, y) that solves (2.2), where $x, y \in \mathbb{F}_p$, and the point at infinity O form an *abelian group* $(E(\mathbb{F}_p),+)$, which ultimately contains all the possible elements for computations on ECC over prime fields:

$$(E(\mathbb{F}_p),+) = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : y^2 - x^3 - ax - b = 0\} \cup \{O\} \tag{2.3}$$

We remark without further proofs at this point that defined group $(E(\mathbb{F}_p),+)$ satisfies properties corresponding to *abelian groups* (see Section 2.1):

- Addition is commutative and associative for all points $P=(x_1,y_1) \in E(\mathbb{F}_p)$.
- The additive identity is denoted by O, such that $P+O = O+P = P$ for all $P \in E(\mathbb{F}_p)$.
- Every $P=(x_1,y_1) \in E(\mathbb{F}_p)$ has an additive inverse denoted by $-P$. We will see later that $-P = (x_1, -y_1)$.

2.2.1 Elliptic Curve Discrete Logarithm Problem (ECDLP)

Let E be the elliptic curve over the finite field $\mathbb{F}_p$. We can represent the main operation in ECC, namely scalar multiplication, as follows:

$$Q = dP \tag{2.4}$$

Where P and Q are points in $E(\mathbb{F}_p)$ of order q, and d is the secret scalar.

We define the Elliptic Curve Discrete Logarithm Problem (ECDLP) as the problem of determining scalar d, given P and Q.

Security of systems based on ECC relies on the hardness of this problem. In general, ECDLP has proven to be harder than other recognized problems such as the integer factorization problem and the discrete logarithm problem, which are the foundation of RSA (Rivest-Shamir-Adleman) and DH (Diffie-Hellman) cryptosystems, respectively.

Definition 2.1 [HMV04] Given an algorithm with input n, where n is an integer with size $l \approx \log_2 n$, its running time is:

$$L_n[a,c] = \mathcal{O}\left(e^{(c+\mathcal{O}(1))(\log_2 n)^a (\log_2 \log_2 n)^{1-a}}\right) \tag{2.5}$$

Where $c > 0$ and $0 \le a \le 1$ are constants.

The running time of (2.5) is said to be polynomial in l ($\mathcal{O}(l^c)$) if $a = 0$, exponential in l if $a = 1$, and subexponential in l if $0 < a < 1$.

In particular, there exists a subexponential attack, called Number Field Sieve (NFS), to solve the integer factorization problem and discrete logarithm problem in the following expected running time:

$$L_n[\frac{1}{3}, 1.923] \tag{2.6}$$

In contrast, the fastest known method to solve ECDLP is Pollard's rho, which is exponential with the following expected running time:

$$\frac{\sqrt{\pi q}}{2} \tag{2.7}$$

Where q is the order of P and Q in (2.4).

Therefore, it is expected that smaller key sizes are required for ECC for a given security level.

Table 2.1 shows the equivalent key sizes for EC, RSA and DL cryptosystems for an equivalent level of security. Estimates are based on the time to run the fastest algorithms that solve each problem (i.e., Pollard's rho and NFS). Security level is expressed by the key size in bits, meaning that for a key (scalar d) of size l, one would require 2^l steps to break the cryptosystem [HMV04].

Cryptosystem	Security level (bits)				
ECC	160	224	256	384	512
RSA / DL	1024	2048	3072	8192	15360

Table 2.1. Key sizes for EC, RSA and DL-based cryptosystems for equivalent security levels [HMV04]

From Table 2.1, we observe that ECC requires much smaller keys. This is directly translated to faster computations and reduced memory requirements, which make this cryptosystem ideal for devices with constrained resources such as smartcards, cellphones, PDAs, laptops, and many others.

In the following section, we succinctly describe the arithmetic layers that constitute the computation of the scalar multiplication. The interested reader is referred to [HMV04,ACD+05] for a more detailed look at the topic.

2.3 ECC Arithmetic

The mathematical hierarchy of the ECC scalar multiplication (2.4) consists of *three* levels: scalar arithmetic, point arithmetic and field arithmetic (see Figure 2.1).

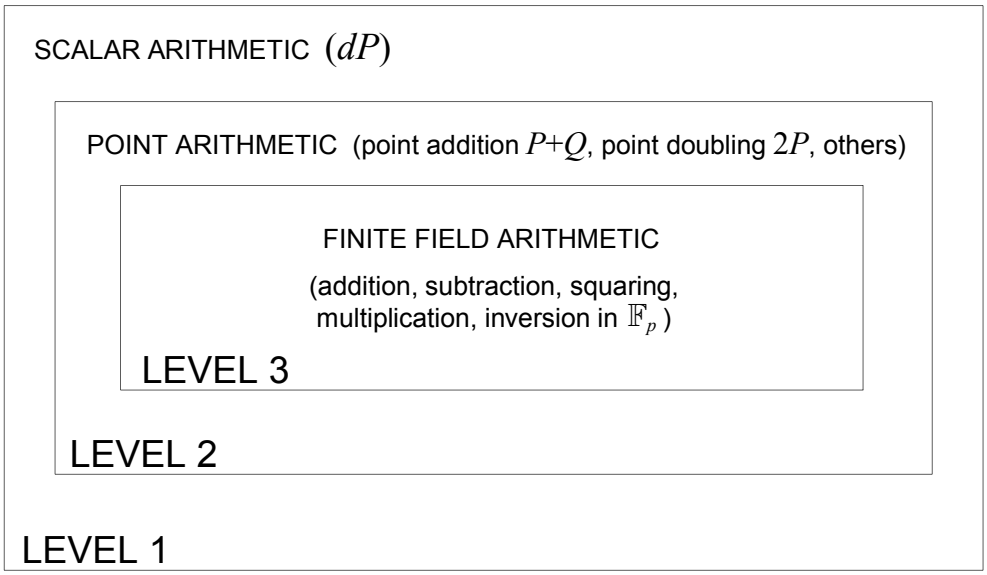


Figure 2.1. Elliptic curve mathematical hierarchy

2.3.1 Level 3: Finite Field Arithmetic

Basic curve operations in ECC over prime fields are performed using field operations. The latter consists of traditional arithmetic operations performed *modulo* the prime p:

- Addition: given $a, b \in \mathbb{F}_p$, $(a + b) \bmod p = r$, where r is the remainder of dividing $a + b$ by p, and $0 \le r \le p - 1$.

- Subtraction: given $a, b \in \mathbb{F}_p$, $(a - b) \bmod p = r$, where r is the remainder of dividing $a - b$ by p, and $0 \le r \le p - 1$. This operation is commonly replaced by an addition performed on a and $(-b)$, given that the negative of any element is easily obtained.

- Multiplication: given $a, b \in \mathbb{F}_p$, $(a.b) \bmod p = r$, where r is the remainder of dividing $a.b$ by p, and $0 \le r \le p - 1$.

- Squaring: given $a \in \mathbb{F}_p$, $(a^2) \bmod p = r$, where r is the remainder of dividing a^2 by p, and $0 \le r \le p - 1$. In some hardware implementations, this operation is replaced by a multiplication of the form $a.a$, given that there is normally only a hardware

multiplier at hand.

- Inversion: given a, a non-zero element in $\mathbb{F}_p$, $(a^{-1}) \bmod p = r$, is the unique integer $r \in \mathbb{F}_p$ for which $(a.r) \bmod p = 1$.

Example 2.2.1 Given the elements of $\mathbb{F}_{11}$: {0,1,2,…,10}, examples of field operations over such finite field are:

- $6 + 8 = 14 \equiv 3 \bmod 11$.
- $6 - 8 = -2 \equiv 9 \bmod 11$.
- $6 \,.\, 8 = 48 \equiv 4 \bmod 11$.
- $6^2 = 36 \equiv 3 \bmod 11$.
- $6^{-1} \equiv 2 \bmod 11$, since $6 \,.\, 2 = 12 \equiv 1 \bmod 11$.

NOTE: for the remainder of this work, we use the following notation in *italics* to specify the computing time (or computing cost) to perform field operations: A (field addition or subtraction), S (field squaring), M (field multiplication) and I (field inversion).

In this work, and to compare the various methods, we assume, as it is widely accepted, $1S = 0.6M$ or $1S = 0.8M$ for the case of software-based implementations [BHL+01,GAS+05,LH00,GG03,Ava04,Ber]. In hardware platforms, only one multiplier is usually available to execute both squaring and multiplication. Consequently, it is assumed $1S = 1M$ in that case. Cost of field addition/subtraction and field division/multiplication by small constants can be considered equivalent in terms of computation time [Ber,Ber06]. Also, their cost is widely assumed negligible in comparison with field multiplications and squarings. Consequently, these operations are not included in our analysis for simplification purposes.

Two exceptions to the previous assumption. We consider the cost of additions when two algorithms offer equivalent costs in terms of multiplications and squarings to ultimately establish which of these is more efficient. Also, in SSCA-protected implementations, we include field additions in the cost evaluation since our methods significantly reduce the required number of additions. In these cases we generally assume $1A = 0.05M$. For details of costs of efficient addition/subtractions and division/multiplications with small constants, the reader is referred to [Ber] and [Ber06].

2.3.2 Level 2: Point Arithmetic

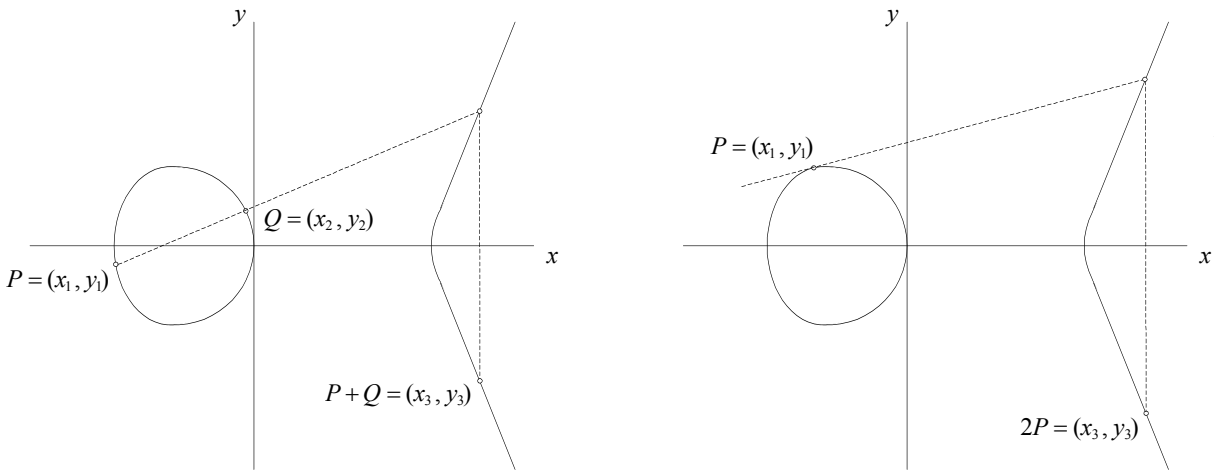
Scalar multiplication (2.4) directly depends on operations over points on the elliptic curve E. In general, traditional methods to compute the scalar multiplication rely on the execution of a given sequence of point doubling ($2P$) and point addition ($P+Q$) operations, where P and Q are points on the elliptic curve E.

Formulae to compute the previous elementary point operations are derived according to what is best known as group law.

Group Law

To best understand the way point formulae are derived, elementary point operations are typically described geometrically. The following description is based on the natural representation of points using x and y coordinates, which is called in the context of ECC *affine coordinates* representation.

Given an elliptic curve over a field K, $E(K)$, the resultant point $P+Q = (x_3, y_3)$ of adding two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in $E(K)$ is the reflection about the x-axis of the point that is intersected by the line crossing P and Q. Figure 2.2(a) permits to visualize such operation.



(a) Addition P+Q (b) Doubling 2P

Figure 2.2. Group law over $\mathbb{R}$

The exception to the previous procedure is the case where the points to be added have identical x-coordinate. Given equation (2.2), for any $x = x_1$, one has two solutions $y = \pm y_1$, and consequently, *two* points with the same x-coordinate, i.e., (x_1, y_1) and $(x_1, -y_1)$.

The latter is simply solved by using the identity of the group law, namely the point at infinity O, which 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].

Thus, the line crossing (x_1, y_1) and $(x_1, -y_1)$ obviously intersects the curve in the point at infinity O. Following the same definition, the reflection of O about the x-axis gives again the point at infinity O, so that $(x_1, y_1) + (x_1, -y_1) = O$. As a consequence, we can define the negative of $P = (x_1, y_1)$ as $-P = (x_1, -y_1)$.

By following the previous geometric description, formula for the point addition in affine coordinates has been derived and is described in the following [HMV04].

Let $E(\mathbb{F}_p)$ be an elliptic curve over the prime field $\mathbb{F}_p$, where $p > 3$. Given two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on $E(\mathbb{F}_p)$, where $P \neq \pm Q$, the addition $P + Q = (x_3, y_3)$ is

obtained as follows:

$$x_3 = \lambda^2 - x_1 - x_2, \qquad y_3 = \lambda(x_1 - x_3) - y_1 \tag{2.8}$$

Where: $\lambda = \dfrac{y_2 - y_1}{x_2 - x_1}$

The cost of the previous formula is $1I + 2M + 1S$.

Similarly, doubling of a point can be explained by its geometric description.

Given an elliptic curve over a field K, $E(K)$, the resultant point $2P = (x_3, y_3)$ of doubling the point $P = (x_1, y_1)$ in $E(K)$ is the reflection about the x-axis of the point that is intersected by the tangent line of P. Figure 2.2(b) permits to visualize such operation.

Formula for the point doubling in affine coordinates can be easily derived from the previous geometric description. It is described in the following [HMV04].

Let $E(\mathbb{F}_p)$ be an elliptic curve over the prime field $\mathbb{F}_p$, where $p > 3$. Given a point $P = (x_1, y_1)$ on $E(\mathbb{F}_p)$, where $P \neq -P$, the doubling $2P = (x_3, y_3)$ is obtained as follows:

$$x_3 = \lambda^2 - 2x_1, \qquad y_3 = \lambda(x_1 - x_3) - y_1 \tag{2.9}$$

Where: $\lambda = \dfrac{3x_1^2 + a}{2y_1}$

The cost of the previous formula is $1I + 2M + 2S$.

Inversion-free (projective) coordinates

The representation of points on the elliptic curve E with two coordinates (x,y), namely affine coordinates, introduces field inversions into the computation of point doubling and point addition. Inversions over prime fields are the most expensive field operation and are avoided as much as possible. Although their relative cost depends on the characteristics of a particular implementation, it has been observed that, especially in the case of efficient forms

for the prime p as recommended by [NIST], $1I > 30M$ [BHL+01, HMV04].

Projective coordinates (X, Y, Z) solve the previous problem by adding the third coordinate Z to replace inversions with a few other field operations. The foundation of these inversion-free coordinate systems can be explained by the concept of *equivalence class*, which is defined in the following.

Given a field K, there is an equivalent relation ≡ among non-zero triplets over K, such that [HMV04,ACD+05]:

$(X_1, Y_1, Z_1) \equiv (X_2, Y_2, Z_2) \Leftrightarrow X_2 = \lambda^c X_1$, $Y_2 = \lambda^d Y_1$ and $Z_2 = \lambda Z_1$, for some $\lambda \in K^*$, and $c, d \in \mathbb{Z}^+$

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

$$(X : Y : Z) = \{(\lambda^c X, \lambda^d Y, \lambda Z) : \lambda \in K^*, c, d \in \mathbb{Z}^+\} \quad (2.10)$$

It is important to remark that any (X, Y, Z) in the equivalence class (2.10) can be used as a representative of a given projective point.

In this case, there is only *one* element that can be represented with $Z = 0$, which is the point at infinity O. Also, there is only *one* element in (2.10) that can be represented with $Z = 1$, corresponding to the affine coordinates representative. The latter can be obtained as follows:

$$(X / Z^c, Y / Z^d, 1) \quad (2.11)$$

By exploiting relation (2.10), several coordinate systems (or combinations of coordinate systems) have been explored to yield inversion-free doubling/addition formulae with lower costs. The reader is referred to [CMO98,CPQ01] for further details.

In the present thesis, we work with Jacobian coordinates, a special case of projective coordinates that has yielded very efficient point doubling and addition formulae for ECC over prime fields [Elm06,HMV04].

Jacobian coordinates representation is obtained by fixing $c = 2$ and $d = 3$ in (2.10). Thus, the equivalence class for *Jacobian coordinates* is as follows:

$$(X : Y : Z) = \{(\lambda^2 X, \lambda^3 Y, \lambda Z) : \lambda \in K^*\} \tag{2.12}$$

By replacing $x = X / Z^2$ and $y = Y / Z^3$, and clearing denominators in formulae (2.8) and (2.9), corresponding to point addition and doubling in affine coordinates, respectively, one transforms affine coordinates formulae into Jacobian coordinates.

In the specific case of addition, representing one of the points in Jacobian and the other in affine coordinates has yielded the most efficient addition formula, which is known as mixed addition in affine-Jacobian coordinates [CMO98].

In the following, we introduce the (traditional) inversion-free point formulae, which can later be used for comparison purposes with the new composite operations presented in Chapter 3, and the fast point operations presented in Chapter 4.

It is important to note that in the case of doubling it has been suggested to fix the parameter a (see equation 2.2) to -3 for efficiency purposes. In fact, most curves recommended by public-key standards [IEEE] use $a = -3$, which has been shown to not impose significant restrictions to the cryptosystem [BJ03].

Thus, in the rest of this work we will refer as *special case* when $a = -3$, and *general case* when such parameter is not fixed and can be any value in the field.

Also, note that we apply such optimization to tripling to reduce its computing cost.

Point doubling in Jacobian coordinates

Let $P = (X_1, Y_1, Z_1)$ be a point in Jacobian coordinates on the elliptic curve E. The point doubling $2P = (X_3, Y_3, Z_3)$ can be computed by the next traditional formula:

$$X_3 = \alpha^2 - 2\beta, \qquad Y_3 = \alpha(\beta - X_3) - 8Y_1^4, \qquad Z_3 = 2Y_1 Z_1 \tag{2.13}$$

Where: $\alpha = 3X_1^2 + aZ_1^4$

$\beta = 4X_1Y_1^2$

Thus, the cost of a doubling is $4M + 6S$. As previously discussed, we can consider the efficient case of $a = -3$ without much loss of generality. In that special case, α is more efficiently computed as follows:

$$3X_1^2 + aZ_1^4 = 3\left(X_1 + Z_1^2\right)\left(X_1 - Z_1^2\right) \tag{2.14}$$

By using factorization (2.14) the cost of the doubling is reduced to $4M + 4S$.

Also, it is important to note that, in the general case, it is possible to reduce the cost when computing repeated doublings. The idea is to avoid intermediate operations during computation of the term aZ_3^4 in α by computing it as $(16Y_1^4)(aZ_1^4)$ using terms from the previous doubling. *Two* squarings are saved for each extra doubling, giving a total cost of $4wM + 2(2w + 1)S$ to compute w consecutive doublings.

(General) Point addition in Jacobian coordinates

Let $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ be points in Jacobian coordinates on the elliptic curve E. The point addition $P+Q = (X_3, Y_3, Z_3)$ can be computed with the next formula:

$$X_3 = \alpha^2 - \beta^3 - 2Z_2^2X_1\beta^2, \qquad Y_3 = \alpha\left(Z_2^2X_1\beta^2 - X_3\right) - Z_2^3Y_1\beta^3, \qquad Z_3 = Z_1Z_2\beta \tag{2.15}$$

Where: $\alpha = Z_1^3Y_2 - Z_2^3Y_1$

$\beta = Z_1^2X_2 - Z_2^2X_1$

Given formula (2.15), the cost of the general addition is $12M + 4S$.

Mixed addition in affine-Jacobian coordinates

Let $P = (X_1, Y_1, Z_1)$ and $Q = (x_2, y_2)$ be two points in Jacobian and affine coordinates,

respectively, on the elliptic curve E. The mixed addition $P+Q=(X_3,Y_3,Z_3)$ is traditionally obtained as follows:

$$X_3=\alpha^2-\beta^3-2X_1\beta^2,\qquad Y_3=\alpha\left(X_1\beta^2-X_3\right)-Y_1\beta^3,\qquad Z_3=Z_1\beta \tag{2.16}$$

Where: $\alpha=Z_1^3Y_2-Y_1$

$\beta=Z_1^2X_2-X_1$

With (2.16), the cost of a mixed addition is fixed in $8M + 3S$.

Point tripling in Jacobian coordinates

[DIM05] introduced a fast tripling formula that costs $10M + 6S$. Let $P=(X_1,Y_1,Z_1)$ be a point in Jacobian coordinates on the elliptic curve E. The point tripling $3P=(X_3,Y_3,Z_3)$ can be computed with the following:

$$X_3=8Y_1^2\left(\beta-\alpha\right)+X_1\omega^2,\qquad Y_3=Y_1\left[4\left(\alpha-\beta\right)\left(2\beta-\alpha\right)-\omega^3\right],\qquad Z_3=Z_1\omega \tag{2.17}$$

Where: $\alpha=\theta\omega$

$\beta=8Y_1^4$

$\theta=3X_1^2+aZ_1^4$

$\omega=12X_1Y_1^2-\theta^2$

[DIM05] proposed to accelerate the computation by avoiding intermediate operations during computation of the term aZ^4 when repeated triplings are to be computed. This idea is based on a similar approach given by [CMO98] with their modified Jacobian coordinates. Formula by [DIM05] to compute w consecutive triplings costs $(11w - 1)M + (4w + 2)S$. However, it is straightforward to note that applying another well-known technique (fixing $a = -3$) actually gives a better performance. Thus, by computing θ using the factorization technique given in (2.14), we reduce the cost of the tripling to $10M + 4S$, and the cost of w consecutive triplings to only $10wM + 4wS$.

Table 2.2 summarizes computing costs of traditional operations using Jacobian (or affine-Jacobian, in the case of addition) coordinates. These costs are used through this work as a reference point to calculate the performance improvement achieved by our new formulae.

Point operation	General case	$a = -3$
Doubling	$4M + 6S$	$4M + 4S$
w-doubling	$4wM + 2(2w + 1)S$	$4wM + 4wS$
Mixed addition	$8M + 3S$	-
Tripling [DIM05]	$10M + 6S$	$10M + 4S$
w-tripling [DIM05]	$(11w - 1)M + (4w + 2)S$	$10wM + 4wS$

Table 2.2. Costs of traditional point operations in Jacobian coordinates

NOTE: for the remainder of this work, we use the following notation to specify the computing time (or computing cost) to perform point operations: A (point addition, $P+Q$), D (point doubling, $2P$) and T (point tripling, $3P$). Costs of composite operations such as those introduced in Chapter 3 use the following notation: DA (point doubling-addition, $2P+Q$), TA (point tripling-addition, $3P+Q$), Q (point quintupling, $5P$), S (point septupling, $7P$), E ($11P$) and TH ($13P$).

2.3.3 Level 1: Scalar Arithmetic

This mathematical level deals with the efficient computation of dP using point operations introduced in the previous section.

A plethora of methods to efficiently compute the scalar multiplication can be found in the literature. In the remainder of this chapter, we succinctly describe the most popular ones based on their efficiency in terms of speed and/or advantageous memory requirements. Refer to [Gor97,HMV04] for an extensive survey in the topic.

In the remainder of this section, for an n-bit scalar multiplication dP (where n is the bitlength of the prime p corresponding to the prime field $\mathbb{F}_p$), we assume that P is of order $h.q$ (q prime and $h << q$) and $n \approx \log_2 q$ (i.e., $p \approx q$) [HMV04]. If d is a scalar randomly chosen in the range [1, $q - 1$], then the average length of d is $l \approx n - 1$. We refer as density or Hamming weight to the number of non-zero elements in a given integer representation. In particular, for scalar multiplication, the latter is directly translated to the number of required point additions to compute dP using such representation.

Binary method

This is the traditional scalar multiplication based on the binary expansion of the scalar d using $\{0,1\}$. Given a binary representation of d, the scalar multiplication can be computed by scanning the bits of d from left to right, as shown in Algorithm 2.1 [HMV04].

Algorithm 2.1 Left-to-right binary method for scalar multiplication

INPUT: $d = (d_{l-1},...,d_0)_2$, $P \in E(\mathbb{F}_p)$

OUTPUT: dP

1. $Q = O$
2. For $i = l - 1$ downto 0 do
   2.1. $Q = 2Q$
   2.2. If $d_i = 1$ then $Q = Q + P$
3. Return (Q)

The average number of doublings and additions using Algorithm 2.1 is $l \approx n - 1$ and $l/2 \approx n/2$, respectively, as d tends to infinity. Thus the cost of the binary method is approximately as follows:

$$(n-1)\mathrm{D} + \left(\frac{n}{2}\right)\mathrm{A} \tag{2.18}$$

Example 2.2.2 Let $d = 12632$ and P a point on the elliptic curve E. Given the binary expansion of d:

- $12632 = 2^{13} + 2^{12} + 2^8 + 2^6 + 2^4 + 2^3 \equiv (11000101011000)_2$

The scalar multiplication denoted by [12632]P using Algorithm 2.1 would be as follows:

- $[12632]P = 2^3\left(2\left(2^2\left(2^2\left(2^4\left(2P + P\right) + P\right) + P\right) + P\right) + P\right)$

Non-Adjacent Form (NAF) method

The density of the binary expansion can be effectively reduced with a signed representation that uses elements in the set {-1,0,1}.

Among different signed representations, NAF is a canonical representation with the fewest number of non-zero digits for any scalar d. The NAF representation of d, denoted by NAF(d), contains at most *one* non-zero digit among any two successive digits. Moreover, the length of NAF(d) is at most *one* more bit than its binary representation.

Algorithm 2.2 computes the NAF of a scalar d if w is fixed to two. Similarly, Algorithm 2.3 uses NAF for scalar multiplication when $w = 2$ [HMV04].

Algorithm 2.2 Computing the wNAF (NAF) of a positive integer

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

OUTPUT: $\text{NAF}_w(d)$

1. $i = 0$
2. While $d \geq 1$ do
   2.1. If d is odd, then $d_i = d \bmod 2^w$, $d = d - d_i$
   2.2. Else $d_i = 0$
   2.3. $d = d/2$, $i = i + 1$
3. Return $(d_{i-1}, \ldots, d_1, d_0)_{\text{NAF}_w}$

Algorithm 2.3 *w*NAF (NAF) method for scalar multiplication

INPUT: Window w ($w = 2$ for NAF), scalar $d = (d_{l-1}, \ldots, d_1, d_0)_{\text{NAF}_w}$, $P \in E(\mathbb{F}_p)$

OUTPUT: dP

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 $d_i \neq 0$, then:
      If $d_i > 0$, then $Q = Q + P_{d_i}$
      Else $Q = Q - P_{d_i}$
4. Return (Q)

Function *mods* in Algorithm 2.2 represents the next computation (for NAF $w = 2$):

$$\begin{cases} \text{If } d \bmod 2^w \geq 2^w / 2, \text{ then:} \\ \quad d_i = \left(d \bmod 2^w\right) - 2^w \\ \text{Else,} \\ \quad d_i = d \bmod 2^w \end{cases} \tag{2.19}$$

For random scalars d of average bitlength l, the expected number of doublings and additions using Algorithm 2.3 ($w = 2$) is approximately ($n - 1$) and $n/3$, respectively. Thus the cost of the NAF method is:

$$(n-1)\mathrm{D} + \left(\frac{n}{3}\right)\mathrm{A} \tag{2.20}$$

Example 2.2.3 Let $d = 12632$ and P a point on the elliptic curve E. Given the NAF expansion of d:

- $12632 = 2^{14} - 2^{12} + 2^9 - 2^7 - 2^5 - 2^3 \equiv (10 - 10010 - 10 - 10 - 1000)_{\text{NAF}}$

The scalar multiplication using Algorithm 2.3 would be computed as follows:

- $[12632]P = 2^2\left(2^2\left(2^2\left(2^3\left(2^2P - P\right) + P\right) - P\right) - P\right) - P\right)$

The cost of the previous computation is 14D + 5A, which follows (2.20) and contains the fewest number of additions of any scalar multiplication with $d = 12632$ on radix 2 when pre-computations are not allowed.

If we consider point operation costs as given by Table 2.2 (special case with $a = -3$), the previous scalar multiplication would cost $14(4M + 4S) + 5(8M + 3S) = 96M + 71S$. If in addition we consider $1S = 0.8M$, then such estimate represents a cost of $153M$.

Window-*w* Non-Adjacent Form (*w*NAF) method

If there is available memory, one can make use of pre-computations 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 pre-computed values to "insert" windows of width *w*, which permits the consecutive execution of several doublings and, thus, reduces the density of the expansion. In consequence, the *w*NAF representation of *d*, denoted by $\text{NAF}_w(d)$, contains at most *one* non-zero digit among any *w* successive digits.

Algorithm 2.2 computes the *w*NAF representation of a scalar *d* when $w > 2$. Similarly, Algorithm 2.3 uses *w*NAF for scalar multiplication if $w > 2$. It can be directly observed 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 pre-computations.

NOTE: in this work, we refer as pre-computed points to non-trivial points (i.e., excluding $\{0,1\}$) that are required to be computed and stored in memory before computation of the scalar multiplication itself.

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

$$(n-1)\mathrm{D}+\left(\frac{n}{w+1}\right)\mathrm{A} \tag{2.21}$$

Example 2.2.4 Let $d = 12632$ and P a point on the elliptic curve E. For $w = 3$, the NAF_3 expansion of d is:

- $12632 = 3\times 2^{12} + 2^9 - 3\times 2^6 + 3\times 2^3 \equiv (300100 - 3003000)_{\mathrm{NAF}_3}$

The scalar multiplication using Algorithm 2.3 ($w = 3$) would be computed as follows:

- $[12632]P = 2^3\left(2^3\left(2^3\left(2^3(3P)+P\right)-3P\right)+3P\right)$

Comparing with example 2.2.3, the cost of the scalar multiplication has been reduced to only 12D + 3A, which follows (2.21) and contains the fewest number of additions of any scalar multiplication with $d = 12632$ on radix 2 using the set of pre-computations {0, ±1, ±3}.

Double Base (DB) scalar multiplication

[DIM05] proposed to represent scalar d using mixed powers of 2 and 3, as follows:

$$d = \sum_{i=1}^{m} d_i 2^{b_i} 3^{c_i} \tag{2.22}$$

Where m is the length of the expansion, d_i is the sign, i.e., $d_i \in \{-1,1\}$, 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 representation is highly sparse and, consequently, it would permit to reduce the Hamming weight of the expansion for the scalar d. Moreover, with the introduction of efficient tripling formulae [DIM05,DIK06], the scalar multiplication would greatly benefit from expansions using ternary bases.

Later, [DI06] extended this approach, called *Extended DB*, to applications that can afford pre-computations. In this case, d_i in (2.22) is allowed to have any value from a set of pre-computed digits D_i, where the elements are prime numbers other than 3. For instance, D_i =

$\{\pm 1, \pm 5, \pm 7, \pm 11\}$.

Computing 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 two and three, respectively, and then executing an exhaustive search for closest terms $2^{b_i}3^{c_i}$ (Algorithm 2.4).

A more detailed analysis of the disadvantages of the previous approach is presented in Chapter 7.

Algorithm 2.4 "Greedy" algorithm for conversion to DB

INPUT: scalar d; $b_{\max}$, $c_{\max} > 0$

OUTPUT: the sequence (s_i, b_i, c_i), where $0 < i < m$; m is the length of the DB expansion

1. $s = 1$
2. While $d > 0$ do
   2.1. define $z = 2^b 3^c$, the best approximation of d with $0 \le b \le b_{\max}$ and $0 \le c \le c_{\max}$
   2.2. print (s, b, c)
   2.3 $b_{\max} = b$, $c_{\max} = c$
   2.4 If $d < z$, then $s = -s$
   2.5 $d = |d - z|$

Example 2.2.5 Let $d = 12632$ and P a point on the elliptic curve E. The DB expansion of d is:

- $12632 = 2^4 \times 3^6 + 2^2 \times 3^5 - 2^2$

The scalar multiplication would be computed as follows:

- $[12632]P = 2^2\left(3^5\left(2^2 \times 3P + P\right) - P\right)$

The cost of the scalar multiplication is fixed in 6T + 4D + 2A, which contains fewer

additions (reduced non-zero density) than NAF in Example 2.2.3.

By considering point operation costs given in Table 2.2 (special case with $a = -3$), the previous scalar multiplication would cost $86M + 58S$. If $1S = 0.8M$, then such estimate represents a cost of only $132M$, which is an improvement in comparison to NAF (Example 2.2.3).

Probably, the most important finding about this representation is given by Theorem 2.2.1 from [DJM98] that shows the sublinear nature of expansions using mixed powers of 2 and 3.

Theorem 2.2.1 Every positive integer d can be represented as the sum of at most $O\left(\frac{\log d}{\log\log d}\right)$ $\{2^b 3^c\}$- terms.

Radix-*r* Non-Adjacent Form (*r*NAF) method

Recently, [TYW04] generalized the binary NAF representation of numbers to any radix r. This particular NAF representation is called radix-r NAF (rNAF) and presents the minimal density for a given radix-based NAF representation, although at the extra cost of pre-computations.

Using the radix-r representation, a scalar d is represented as follows:

$$d = \sum_{i=0}^{l-1} d_i r^i \tag{2.23}$$

Where: $d_i \in \{\pm 0, \pm 1, ..., \pm(r-1)\}$ is the i^{th} digit of the radix-r NAF representation of d.

$d = (d_{l-1}, ..., d_0)$ is the radix-r representation of d.

l is the digit length of the radix-r representation of d.

It has been shown by [TYW04] that the density of the rNAF representation of a number is given by:

$$\frac{r-1}{2r-1}, \tag{2.24}$$

with a requirement of $\frac{(r-2)(r+1)}{2}$ pre-computed points.

The windowed version of this representation, called *wr*NAF, was also introduced in the same work, and it is a generalization of the well-known window-*w* NAF to any radix *r*. Thus, the density of the *wr*NAF is as follows:

$$\frac{r-1}{w(r-1)+1}, \tag{2.25}$$

with a requirement of $\frac{r^w - r^{w-1} - 2}{2}$ pre-computed points.

We must point out that a similar representation was previously given by [JY02], called *g*NAF (generalized NAF), in which fewer pre-computed points are required but the achieved density is higher. More precisely, one requires $(r-2)$ pre-computed points with a density of $\frac{r-1}{r+1}$ for the *g*NAF representation of a number.

Chapter 3

Efficient Point Arithmetic over Prime Fields via Composite Operations

As explained in Chapter 2, ECC scalar multiplications traditionally exploit the binary expansion of numbers. This is mainly due to two reasons: binary is a natural way to express numbers on computers, and its expansion is directly translated into computations using the simplest and elementary ECC point operations, namely point doubling and addition.

However, recent developments in the field have shown that it is possible to use more complex operations to accelerate the scalar multiplication [CJL+06,DIM05,DIK06]. Examples of those operations are tripling ($3P$, denoted by T) or quadrupling ($4P$) of a point, unified doubling-addition ($2P+Q$, denoted by DA), unified tripling-addition ($3P+Q$, denoted by TA), among others. Since, these new operations are inherently based on basic point doubling and addition, we refer to them as *composite* operations.

In the present chapter, we first detail most important efforts in the literature to yield efficient composite operations over prime fields. Then, we introduce our faster composite operations and generalize them to operations of the forms $dP+Q$ and dP. Following, we give example applications where these new operations can be efficiently used to accelerate ECC

computations. In particular, we show how to accelerate the pre-computation of points in window-based scalar multiplications, and then the computation of the scalar multiplication itself.

3.1 Previous Work

Recently, special effort has been put in developing efficient composite operations, which ultimately are expected to speed-up the scalar multiplication. This has a strong fundament: introducing bases different than two in the expansion of the scalar reduces the number of required terms and, consequently (if the required composite operations are efficient enough), reduces the overall computing cost.

Let's examine the following example to illustrate the latter:

$$101062 = 2^{17} - 2^{15} + 2^{12} - 2^{10} - 2^{8} - 2^{6} + 2^{3} - 2\,,$$

is the traditional NAF expansion using base 2. Then, a scalar multiplication using this expansion for the scalar *d* will require *seventeen* doublings and *seven* additions. If instead of 2 we use base 3, we obtain the following:

$$101062 = 3^{11} - 4 \times 3^{9} + 3^{7} + 2 \times 3^{5} - 3^{3} + 1$$

The previous expansion is denoted by 3NAF, and evidently contains fewer terms than the traditional binary NAF. In fact, it only requires *eleven* triplings and *five* additions if one computes the scalar multiplication in such a way. Therefore, if the point tripling is made efficient enough, we can potentially accelerate the scalar multiplication by using a ternary expansion. The reader must note that we are not including pre-computations in the previous analysis to simplify our explanation.

The previous example highlights the importance of finding efficient composite operations. Further reductions in the length of the expansion are expected if higher order composite

operations are included as in the generalized NAF (*g*NAF) [JY02] or radix-*r* NAF (*r*NAF) [TYW04] (see Section 2.3.3), or if one makes use of combination of different bases as in the ternary/binary approach [CJL+06] or the double-base number system (DBNS) [DIM05].

In such direction, [ELM03] developed formulae for doubling-addition (DA), tripling and tripling-addition (TA) in affine coordinates using the trick of replacing the point doubling by an addition (in this particular coordinate system point addition is slightly cheaper than doubling; see Section 2.3.2). For instance, $2P+Q$ is proposed to be computed as $(P+Q)+P$. Later, [CJL+06] improved these formulae by using the Montgomery's method to trade inversions for some cheaper multiplications. They extended the technique to yield quadrupling and quadrupling-addition formulae. The disadvantage of both works is that formulae for composite operations contain inversions, which are particularly expensive over prime fields. We pointed out in Chapter 2 that in the specific case of prime fields, it is recommended to get rid of these highly expensive field operations.

In [DIM05], the authors derived the first point tripling in projective (Jacobian) coordinates, making practical expansions with base 3 (and consequently, with multiples of 3) over prime fields when using standard curves. In fact, based on this finding, they proposed the double-base number system for the expansion of the scalar d, which contains terms of the form $2^a \times 3^b$.

We could not find in the current literature efficient formulae for other composite operations such as quintupling, septupling, doubling-addition (using free-inversion coordinates), and others, on standard curves over prime fields, which would be very attractive to implement expansions with reduced terms.

In the following, we propose, to our knowledge, the first two generic strategies to derive efficient composite operations of the form $dP+Q$ and dP for such case.

3.2 Composite Operations *dP + Q*

Our strategy to yield efficient composite operations of the form $dP+Q$ consists in replacing doubling operations by additions. Even though addition is generally more costly than doubling (in inversion-free coordinates), the overall cost is reduced because of use of highly efficient addition formulae with identical z-coordinate [Mel06], which is described in the following:

Let $P = (X_1, Y_1, Z)$ and $Q = (X_2, Y_2, Z)$ be two points with identical z-coordinate in Jacobian coordinates on the elliptic curve E. The addition $P+Q = (X_3, Y_3, Z_3)$ can be obtained using formulae (2.15) as follows:

$$\begin{aligned} X_3 &= \left(Z^3Y_2 - Z^3Y_1\right)^2 - \left(Z^2X_2 - Z^2X_1\right)^3 - 2Z^2X_1\left(Z^2X_2 - Z^2X_1\right)^2 \\ &= \left[\left(Y_2 - Y_1\right)^2 - \left(X_2 - X_1\right)^3 - 2X_1\left(X_2 - X_1\right)^2\right]Z^6 \end{aligned}$$

$$\begin{aligned} Y_3 &= \left(Z^3Y_2 - Z^3Y_1\right)\left(Z^2X_1\left(Z^2X_2 - Z^2X_1\right)^2 - X_3\right) - Z^3Y_1\left(Z^2X_2 - Z^2X_1\right)^3 \\ &= \left[\left(Y_2 - Y_1\right)\left(X_1\left(X_2 - X_1\right)^2 - X_3\right) - Y_1\left(X_2 - X_1\right)^3\right]Z^9 \end{aligned}$$

$$Z_3 = Z^2\left(Z^2X_2 - Z^2X_1\right) = \left[Z\left(X_2 - X_1\right)\right]Z^3$$

Because $Z^3, Z^6, Z^9 \in K^*$, it can be easily noticed that the terms in brackets are also representatives of the equivalence class for Jacobian coordinates corresponding to the point (X_3, Y_3, Z_3) (see Chapter 2, Section 2.3.2). Hence, the previous formulae are equivalent to the following:

$$\begin{aligned} X_3 &= \left(Y_2 - Y_1\right)^2 - \left(X_2 - X_1\right)^3 - 2X_1\left(X_2 - X_1\right)^2 \\ Y_3 &= \left(Y_2 - Y_1\right)\left(X_1\left(X_2 - X_1\right)^2 - X_3\right) - Y_1\left(X_2 - X_1\right)^3 \\ Z_3 &= Z\left(X_2 - X_1\right) \end{aligned} \tag{3.1}$$

These new addition formulae with identical z-coordinate only cost $5M + 2S$, which represents a significant reduction in comparison with $8M + 3S$ corresponding to the traditional addition with mixed affine-Jacobian coordinates (2.16). Sadly, it is not possible to directly replace traditional additions with this more efficient special operation since, evidently, it is expected that additions are computed over operands with different z-coordinate during the scalar multiplication.

[Mel06] applied his formulae to the context of scalar multiplication with star addition chains, where the particular sequence of operations allows the substitution of each traditional addition by the new special operation (3.1). However, we noticed that the new addition can in fact be applied to a wider context with traditional scalar multiplication algorithms. In the following, we develop faster composite operations by exploiting the advantages of this special addition with identical z-coordinate for Elliptic Curve Cryptosystems using generic scalar multiplications over prime fields.

3.2.1 Improved Doubling-Addition (DA) operation

$2P+Q$ is a recurrent operation in efficient scalar multiplications (see Chapter 2, Section 2.3.3). We propose to compute it as $(P+Q)+P$. We will show that, following this approach, the first addition in parenthesis can be computed with a traditional mixed addition (2.16), and the second operation, with the special addition with identical z-coordinate (3.1).

First, we have the mixed affine-Jacobian addition $P+Q = (X_1,Y_1,Z_1) + (X_2,Y_2) = (X_3,Y_3,Z_3)$, which is computed via formulae (2.16):

$$X_3 = \alpha^2 - \beta^3 - 2X_1\beta^2, \qquad Y_3 = \alpha\left(X_1\beta^2 - X_3\right) - Y_1\beta^3, \qquad Z_3 = Z_1\beta \tag{3.2}$$

Where: $\alpha = Z_1^3 Y_2 - Y_1$

$\beta = Z_1^2 X_2 - X_1$

Then, to apply the special addition for the second operation, we need $(P+Q)$ and P to have

the same *z*-coordinate. For this purpose, we follow the next procedure:

1st Choose an efficient equivalent point $\left(X_1', Y_1', Z_1'\right)$ for P from the equivalence class for Jacobian coordinates (2.12), s.t. $Z_1' = Z_3$ to comply the requirement of identical *z*-coordinate. The latter can be achieved by fixing $\lambda = Z_1^2 X_2 - X_1$ to have the following:

$$\left(X_1', Y_1', Z_1'\right) = \left(X_1\left(Z_1^2 X_2 - X_1\right)^2, Y_1\left(Z_1^2 X_2 - X_1\right)^3, Z_1\left(Z_1^2 X_2 - X_1\right)\right) \equiv \left(X_1, Y_1, Z_1\right) \quad (3.3)$$

➔ $Z_1' = Z_3 = Z$

Now we can apply the special addition since $\left(X_1', Y_1', Z_1'\right)$, which is equivalent to the original point $P = (X_1, Y_1, Z_1)$, has the same *z*-coordinate as $\left(X_3, Y_3, Z_3\right)$. Most importantly, the equivalent point $\left(X_1', Y_1', Z_1'\right)$ does not introduce any extra cost because it is already computed in the first addition (see (3.2)).

2nd Use formulae (3.1), corresponding to addition with identical *z*-coordinate, to compute $(P+Q)+P = \left(X_3, Y_3, Z_3\right) + \left(X_1', Y_1', Z_1'\right) = \left(X_4, Y_4, Z_4\right)$, as follows:

$$\begin{aligned} X_4 &= \left(Y_3 - Y_1'\right)^2 - \left(X_3 - X_1'\right)^3 - 2X_1'\left(X_3 - X_1'\right)^2 \\ Y_4 &= \left(Y_3 - Y_1'\right)\left(X_1'\left(X_3 - X_1'\right)^2 - X_4\right) - Y_1'\left(X_3 - X_1'\right)^3 \\ Z_4 &= Z\left(X_3 - X_1'\right) \end{aligned} \quad (3.4)$$

In overall, because the proposed DA operation involves *one* mixed and *one* special addition, the cost is fixed in (8*M* + 3*S*) + (5*M* + 2*S*) = 13*M* + 5*S*. In contrast, a traditional doubling (fixed $a = -3$, formulae (2.13)) followed by a mixed addition operation costs: (4*M* + 4*S*) + (8*M* + 3*S*) = 12*M* + 7*S*.

➔ the proposed DA operation trades *one* multiplication for *two* squarings.

Furthermore, because our DA does not involve a traditional doubling, the same aforementioned cost is achieved when the parameter *a* (equation (2.2)) is randomly chosen.

In contrast, a general doubling (2.13) followed by a mixed addition operation costs: (4*M* + 6*S*) + (8*M* + 3*S*) = 12*M* + 9*S*.

➔ the proposed DA trades *one* multiplication for *four* squarings.

Unified Doubling-Addition (DA) operation

We can reduce further the cost of the proposed DA operation in terms of field additions by unifying the two point additions (i.e., mixed and special additions) into the following unified DA formulae:

$$X_4 = \beta^2 - \alpha^3 - 2X_1'\alpha^2, \qquad Y_4 = \beta\left(X_1'\alpha^2 - X_4\right) - Y_1'\alpha^3, \qquad Z_4 = Z_1'\alpha \tag{3.5}$$

Where:

$$X_1' = X_1\left(Z_1^2X_2 - X_1\right)^2$$
$$Y_1' = Y_1\left(Z_1^2X_2 - X_1\right)^3$$
$$Z_1' = Z_1\left(Z_1^2X_2 - X_1\right)$$
$$\alpha = X_3 - X_1' = \left(Z_1^3Y_2 - Y_1\right)^2 - \left(Z_1^2X_2 - X_1\right)^3 - 3X_1\left(Z_1^2X_2 - X_1\right)^2$$
$$\beta = Y_3 - Y_1' = -\alpha\left(Z_1^3Y_2 - Y_1\right) - 2Y_1\left(Z_1^2X_2 - X_1\right)^3$$

Notice that, by avoiding the intermediate computation of X_3, Y_3 and Z_3 from the first point addition (3.2), we can directly compute $\alpha = X_3 - X_1'$ and $\beta = Y_3 - Y_1'$ and save *two* field additions.

The total cost is then fixed in 13*M* + 5*S* + 12*A*, which is superior to the traditional execution consisting of a doubling followed by a mixed addition: 12*M* + 7*S* + 16*A* if $a = -3$, and 12*M* + 9*S* + 15*A* if *a* is randomly chosen (see Section 2.3.2).

3.2.2 Improved Tripling-Addition (TA) operation

Similarly to DA, 3*P*+*Q* is proposed to be computed as (*P*+*Q*) + *P* + *P*. We first follow the

procedure given for DA to have $2P+Q$ with a cost of $13M + 5S$ using *one* mixed addition and *one* special addition. The result is expressed as follows (3.4):

$$(P+Q)+P=\left(X_3,Y_3,Z_3\right)+\left(X_1^{'},Y_1^{'},Z_1^{'}\right)=\left(X_4,Y_4,Z_4\right):$$

$$X_4=\left(Y_3-Y_1^{'}\right)^2-\left(X_3-X_1^{'}\right)^3-2X_1^{'}\left(X_3-X_1^{'}\right)^2$$
$$Y_4=\left(Y_3-Y_1^{'}\right)\left(X_1^{'}\left(X_3-X_1^{'}\right)^2-X_4\right)-Y_1^{'}\left(X_3-X_1^{'}\right)^3$$
$$Z_4=Z\left(X_3-X_1^{'}\right) \tag{3.6}$$

Similarly to DA, values for $\left(X_1^{'},Y_1^{'},Z_1^{'}\right)$ are computed according to (3.2) and (3.3).

Then, to apply the special addition given by (3.1) for the last operation, we require (P+Q) + P and P to have the same z-coordinate. Similarly to the procedure given in Section 3.2.1, we follow the next steps:

1st Choose an efficient equivalent point $\left(X_1^{''},Y_1^{''},Z_1^{''}\right)$ for P from the equivalence class for Jacobian coordinates (2.12), s.t. $Z_1^{''}=Z_4$ to comply the requirement of identical z-coordinate. The latter can be achieved by fixing $\lambda=X_3-X_1^{'}$ to have the following:

$$\left(X_1^{''},Y_1^{''},Z_1^{''}\right)=\left(X_1^{'}\left(X_3-X_1^{'}\right)^2,Y_1^{'}\left(X_3-X_1^{'}\right)^3,Z_1^{'}\left(X_3-X_1^{'}\right)\right)\equiv\left(X_1^{'},Y_1^{'},Z_1^{'}\right) \tag{3.7}$$
$\rightarrow\ Z_1^{''}=Z_4=Z$

Now we can use the special addition for the last addition operation since $\left(X_1^{''},Y_1^{''},Z_1^{''}\right)$, which is equivalent to the modified point $\left(X_1^{'},Y_1^{'},Z_1^{'}\right)$, has the same z-coordinate as $\left(X_4,Y_4,Z_4\right)$. Most importantly, computation of the equivalent point $\left(X_1^{''},Y_1^{''},Z_1^{''}\right)$ does not introduce any extra cost because it is already given internally by (3.6).

2nd Use formulae (3.1), corresponding to addition with identical z-coordinate, to compute $((P+Q)+P)+P=\left(X_4,Y_4,Z_4\right)+\left(X_1^{''},Y_1^{''},Z_1^{''}\right)=\left(X_5,Y_5,Z_5\right)$.

In overall, because the proposed TA operation involves *one* mixed and *two* special additions, the cost is fixed in $(8M + 3S) + 2(5M + 2S) = 18M + 7S$. This is similar to the cost achieved by a traditional tripling (formulae 2.17, fixed $a = -3$) followed by a mixed addition: $(10M + 4S) + (8M + 3S) = 18M + 7S$. We will show later that our TA is in fact slightly superior when considering field additions in the cost estimate.

Moreover, because our approach does not involve a traditional doubling, the same aforementioned cost is achieved when the parameter a is randomly chosen. In contrast, a traditional tripling [DIM05] followed by a mixed addition operation costs: $(10M + 6S) + (8M + 3S) = 18M + 9S$.

➔ the proposed TA saves *two* squarings.

Unified Tripling-Addition (TA) operation

We can reduce further the cost of the proposed TA operation in terms of field additions by unifying the point additions (i.e., mixed and special additions) into the following unified TA formulae:

$$X_5 = \omega^2 - \theta^3 - 2X_1''\theta^2, \qquad Y_5 = \omega\left(X_1''\theta^2 - X_5\right) - Y_1''\theta^3, \qquad Z_5 = Z_1''\theta \tag{3.8}$$

Where:
$$\theta = X_4 - X_1'' = \beta^2 - \alpha^3 - 3X_1'\alpha^2$$
$$\omega = Y_4 - Y_1'' = -\theta\beta - 2Y_1'\alpha^3$$
$$X_1'' = X_1'\alpha^2, \quad Y_1'' = Y_1'\alpha^3, \quad Z_1'' = Z_1'\alpha$$
$$X_1' = X_1\left(Z_1^2X_2 - X_1\right)^2$$
$$Y_1' = Y_1\left(Z_1^2X_2 - X_1\right)^3$$
$$Z_1' = Z_1\left(Z_1^2X_2 - X_1\right)$$
$$\alpha = X_3 - X_1' = \left(Z_1^3Y_2 - Y_1\right)^2 - \left(Z_1^2X_2 - X_1\right)^3 - 3X_1\left(Z_1^2X_2 - X_1\right)^2$$
$$\beta = Y_3 - Y_1' = -\alpha\left(Z_1^3Y_2 - Y_1\right) - 2Y_1\left(Z_1^2X_2 - X_1\right)^3$$

Notice that, by avoiding intermediate computations of the first two point additions, we can

directly compute $\theta = X_4 - X_1^{''}$, $\omega = Y_4 - Y_1^{''}$, $\alpha = X_3 - X_1^{'}$ and $\beta = Y_3 - Y_1^{'}$ and save *three* field additions.

The total cost is then fixed in $18M + 7S + 17A$, which is superior to the traditional execution consisting of a tripling followed by a mixed addition: $18M + 7S + 21A$ if $a = -3$ (special case), and $18M + 9S + 20A$ if a is randomly chosen (general case). See formulas 2.16 and 2.17 for cost details

3.2.3 Generalization to Composite Operations *dP + Q*

The approach presented in previous sections can be generalized as follows:

$$dP + Q = (P+Q) + P + P + \ldots \tag{3.9}$$

Where only the first addition in parenthesis is computed with a traditional mixed addition (2.16). Every extra addition out of the term in parenthesis can be computed with a special addition with identical z-coordinate (3.1).

Note that the latter is possible because P always has an equivalent point with the same z-coordinate as the resultant point of the precedent computation. For instance, in the case of TA from Section 3.2.2, $Z_1^{''} = Z_4$ if we fix the new equivalent of point P as $\left(X_1^{''}, Y_1^{''}, Z_1^{''}\right) = \left(X_1^{'}\left(X_3 - X_1^{'}\right)^2, Y_1^{'}\left(X_3 - X_1^{'}\right)^3, Z_1^{'}\left(X_3 - X_1^{'}\right)\right) \equiv \left(X_1^{'}, Y_1^{'}, Z_1^{'}\right)$ (see (3.7)).

We observe that every extra addition adjusts to the next generic formulae:

$$\left((2P+Q) + \underbrace{P + \ldots + P}_{j \text{ terms}}\right) + P = \left(X_{j+2}, Y_{j+2}, Z_{j+2}\right) + \left(X_1^{(j)}, Y_1^{(j)}, Z_1^{(j)}\right) = \left(X_{j+3}, Y_{j+3}, Z_{j+3}\right):$$

$$X_{j+3} = \left(Y_{j+2} - Y_1^{(j)}\right)^2 - \left(X_{j+2} - X_1^{(j)}\right)^3 - 2X_1^{(j)}\left(X_{j+2} - X_1^{(j)}\right)^2$$

$$Y_{j+3} = \left(Y_{j+2} - Y_1^{(j)}\right)\left(X_1^{(j)}\left(X_{j+2} - X_1^{(j)}\right)^2 - X_{j+3}\right) - Y_1^{(j)}\left(X_{j+2} - X_1^{(j)}\right)^3$$

$$Z_{j+3} = Z_1^{(j)}\left(X_{j+2} - X_1^{(j)}\right) \tag{3.10}$$

Where $\left(X_1^{(j)}, Y_1^{(j)}, Z_1^{(j)}\right)$ denotes the equivalent point of P for the current addition.

In (3.10), we can see that it is true that one always gets an equivalent point for P by fixing: $\left(X_1^{(j+1)}, Y_1^{(j+1)}, Z_1^{(j+1)}\right) = \left(X_1^{(j)}\left(X_{j+2} - X_1^{(j)}\right)^2, Y_1^{(j)}\left(X_{j+2} - X_1^{(j)}\right)^3, Z_1^{(j)}\left(X_{j+2} - X_1^{(j)}\right)\right)$, which is equivalent to $\left(X_1^{(j)}, Y_1^{(j)}, Z_1^{(j)}\right)$ according to (2.12), and has identical z-coordinate as (3.10).

The cost of generalization (3.9) is given by 1A + $(d-1)$A', where $d \in \mathbb{Z}^+$, $d \geq 2$, A denotes the cost of the traditional mixed addition and A' denotes the cost of the special addition with identical z-coordinate (i.e., $5M + 2S$).

Thus, the cost of the general composite operation in (3.9) is given by:

$$(8M + 3S) + (d-1)(5M + 2S) \tag{3.11}$$

It is important to remark that the previous generalization is efficient for d = 2, 3 (composite operations DA and TA) and prime numbers > 3. If we continue adding a point P at a time for even values of d, then the high efficiency of the special addition is cancelled by the increased number of operations to be performed.

For the latter, we propose a DA-based approach to yield efficient composite formulae by exploiting the high efficiency of the already improved DA. Thus, for d = 4 and 6, we propose to compute $dP+Q$ as follows:

- $4P+Q = (2P+Q) + 2P$, which involves a point doubling followed by a DA operation.
- $6P+Q = (3P+Q) + 3P$, which involves a point tripling followed by DA.

Notice that proposed computations follow the same structure of the new DA from Section 3.2.1, with the difference that these operations first require some pre-computation on point P (doubling or tripling) to afterwards compute the composite DA operation.

3.2.4 Performance comparison

Cost estimates using generalization (3.9), the proposed DA-based approach and the traditional formulae for different composite operations of form $dP+Q$ are summarized in Table 3.1. Since we could not find in the literature any effort to accelerate composite operations of the form $dP+Q$ in the case of projective (Jacobian) coordinates over prime fields, for the traditional case composite operations are computed by using basic point operations (i.e., doubling, tripling and addition) in the most efficient way. Thus, $2P+Q$ is computed by a doubling followed by a mixed addition; $3P+Q$, by a tripling followed by a mixed addition; $4P+Q$, by two consecutive doublings and a mixed addition; $5Q+P$, by two doublings followed by general and mixed additions; and $6P+Q$, by one doubling, one tripling and one mixed addition. Costs for those basic operations were given in Table 2.2 for the general (random a) and special ($a = -3$) cases. Note that the traditional approach has been slightly improved for the general case by saving some operations during computation of repeated doublings (see w-doubling in Table 2.2).

Method	$2P+Q$	$3P+Q$	$4P+Q$	$5P+Q$	$6P+Q$
Generalization (3.9)	$13M + 5S$	$18M + 7S$	$23M + 9S$	$28M + 11S$	$33M + 13S$
Using proposed DA	-	-	$17M + 9S$ [(a)] $17M + 11S$	$29M + 13S$ [(a)] $29M + 15S$	$22M + 11S$ [(a)] $23M + 11S$
Traditional	$12M + 7S$ [(a)] $12M + 9S$	$18M + 7S$ [(a)] $18M + 9S$	$16M + 11S$ [(a)] $16M + 13S$	$28M + 15S$ [(a)] $28M + 17S$	$22M + 11S$ [(a)] $22M + 15S$

(a) Parameter a in the doubling formulae is fixed ($a = -3$).

Table 3.1. Performance of proposed composite operations of form $dP+Q$ in comparison with traditional formulae

As mentioned before, when $d = 2$ and 5, our generalization (3.9) reduces costs in comparison with the traditional formulae by trading *one* multiplication for up to *six* squarings in comparison with a traditional approach. When $d = 3$, generalization (3.9) is superior since it performs better in the general case. On the other hand, for $d = 4$ and 6, the improved DA can be efficiently used to reduce costs by trading *one* multiplication for up to

two squarings.

3.3 Composite Operations *dP*

3.3.1 Improved Tripling (T) operation

3*P* is proposed to be computed as (2*P*)+*P*. In this case, we will show how the special addition with identical *z*-coordinate (3.1) can be used instead of a general addition operation to reduce costs.

First, we compute the point doubling $2P = 2(X_1,Y_1,Z_1) = (X_2,Y_2,Z_2)$, which is computed via formulae (2.13):

$$X_2 = \left[3\left(X_1 + Z_1^2\right)\left(X_1 - Z_1^2\right)\right]^2 - 2\left(4X_1Y_1^2\right)$$
$$Y_2 = \left[3\left(X_1 + Z_1^2\right)\left(X_1 - Z_1^2\right)\right]\left(4X_1Y_1^2 - X_2\right) - 8Y_1^4$$
$$Z_2 = 2Y_1Z_1 \qquad (3.12)$$

Then, to apply the special addition for the second operation, we require 2*P* and *P* to have the same *z*-coordinate. For this purpose, we follow the next procedure:

1st Choose an efficient equivalent point $\left(X_1', Y_1', Z_1'\right)$ for *P* from the equivalence class for Jacobian coordinates (2.12), s.t. $Z_1' = Z_2$ to comply the requirement of identical *z*-coordinate. The latter can be achieved by fixing $\lambda = 2Y_1$ to have the following:

$$\left(X_1', Y_1', Z_1'\right) = \left(X_1\left(4Y_1^2\right), Y_1\left(8Y_1^3\right), Z_1\left(2Y_1\right)\right) \equiv \left(X_1, Y_1, Z_1\right)$$
➔ $Z_1' = Z_2 = Z$

Now we can apply the special addition since $\left(X_1', Y_1', Z_1'\right)$, which is equivalent to the

original point $P=(X_1,Y_1,Z_1)$, has the same z-coordinate as (X_2,Y_2,Z_2). Most importantly, computation of the equivalent point $(X_1^{'},Y_1^{'},Z_1^{'})$ does not introduce any extra cost because it has already been computed by the first addition (see (3.12)).

2nd Use formulae (3.1), corresponding to addition with identical z-coordinate, to compute $(2P)+P=(X_2,Y_2,Z_2)+(X_1^{'},Y_1^{'},Z_1^{'})=(X_3,Y_3,Z_3)$, as follows:

$$X_3=(Y_1^{'}-Y_2)^2-(X_1^{'}-X_2)^3-2X_2(X_1^{'}-X_2)^2$$
$$Y_3=(Y_1^{'}-Y_2)(X_2(X_1^{'}-X_2)^2-X_3)-Y_2(X_1^{'}-X_2)^3$$
$$Z_3=Z(X_1^{'}-X_2) \qquad (3.13)$$

In overall, because the proposed tripling operation involves *one* doubling and *one* special addition, the cost is $(4M + 4S) + (5M + 2S) = 9M + 6S$, which is *one* multiplication cheaper than the tripling proposed by [DIM05]. However, in Chapter 2 (Section 2.3.2), we modified such operation for the special case $a = -3$, reducing the cost of the tripling to $10M + 4S$, which is faster than the proposed tripling. Moreover, by efficiently executing formulae (2.17), we have proposed in Appendix A a new algorithm for the computation of the point tripling. The overall cost of our algorithm is only $9M + 5S$, which makes it the most efficient for the execution of point tripling.

Nevertheless, the presented approach will be used in the following section to derive efficient point quintupling ($5P$) and septupling ($7P$) operations.

3.3.2 Generalization to Composite Operations *dP*

The approach described for the improved tripling can be generalized for any odd positive integer $d \geq 3$ (for efficiency reasons, d would be also prime), as follows:

$$dP = (2P + P) + 2P + 2P + \ldots \qquad (3.14)$$

Where each addition can be computed with a special addition with identical *z*-coordinate (3.1).

Note that the latter is possible because 2*P* always has an equivalent point with the same *z*-coordinate as the result of the current computation. For instance, for the case of quintupling (5*P*), the addition following the tripling operation from the previous section would be $(3P)+2P=(X_3,Y_3,Z_3)+(X_2,Y_2,Z_2)$. If we fix $\lambda = X_1^{'} - X_2$ in (2.12), point $2P=(X_2,Y_2,Z_2)$ would be equivalent to $\left(X_2\left(X_1^{'}-X_2\right)^2, Y_2\left(X_1^{'}-X_2\right)^3, Z_2\left(X_1^{'}-X_2\right)\right)$, which has the same *z*-coordinate as (X_3,Y_3,Z_3) in formulae (3.13) and, thus, allows us to use the special addition (3.1). Again, computing the equivalent point for 2*P* does not introduce any extra cost since all the required computations are internally computed in the previous addition (see (3.13)). We can repeat the same procedure for extra additions with 2*P*, and yield 7*P*, 9*P*, 11*P*, and so on.

The cost of generalization (3.14) is given by 1D + $\left(\frac{d-1}{2}\right)$A', where $d \geq 3$ is efficiently an odd positive prime, D denotes the cost of the traditional doubling and A' denotes the cost of the new addition with identical *z*-coordinate.

Thus, the cost of the general composite operation of the form *dP* is given by the following:

$$\mathrm{D} + \left(\frac{d-1}{2}\right)(5M + 2S) \tag{3.15}$$

Where D = 4*M* + 4*S* when considering special case $a = -3$. Otherwise, D = 4*M* + 6*S* (formulae (2.13)).

Let us estimate the costs of point quintupling (5*P*), septupling (7*P*) and eleventupling (11*P*) and compare them with costs using traditional operations.

By using (3.15), we can estimate the cost of the quintupling (d = 5) as (4*M* + 6*S*) + 2(5*M* + 2*S*) = 14*M* + 10*S*, if a is randomly chosen. If $a = -3$, the cost of the new quintupling is

reduced to $(4M + 4S) + 2(5M + 2S) = 14M + 8S$.

In contrast, traditional quintupling computed as $(2^2P + P)$ with random a costs: $(8M + 10S) + (12M + 4S) = 20M + 14S$. If $a = -3$, the cost would be $2(4M + 4S) + (12M + 4S) = 20M + 12S$.

➔ the proposed quintupling introduces a significant reduction of $6M + 4S$ for the mentioned cases.

For the case of septupling ($d = 7$), the cost is fixed in $(4M + 6S) + 3(5M + 2S) = 19M + 12S$, if a is randomly chosen. If $a = -3$, the cost of the new septupling is reduced to $(4M + 4S) + 3(5M + 2S) = 19M + 10S$.

In contrast, traditional septupling computed as $2(3P) + P$ with random a costs: $(4M + 6S) + (10M + 6S) + (12M + 4S) = 26M + 16S$. If $a = -3$, the cost would be $(4M + 4S) + (10M + 4S) + (12M + 4S) = 26M + 12S$.

➔ the proposed septupling introduces significant reductions of $7M + 4S$ and $7M + 2S$ for the mentioned cases, respectively.

For the case of eleventupling ($d = 11$), the cost is fixed in $(4M + 6S) + 5(5M + 2S) = 29M + 16S$, if a is randomly chosen. If $a = -3$, the cost of the new septupling is reduced to $(4M + 4S) + 5(5M + 2S) = 29M + 14S$.

In contrast, traditional eleventupling computed as $2^2(3P) - P$ with random a costs: $(8M + 10S) + (10M + 6S) + (12M + 4S) = 30M + 20S$. If $a = -3$, the cost would be $2(4M + 4S) + (10M + 4S) + (12M + 4S) = 30M + 16S$.

➔ the proposed eleventupling introduces reductions of $1M + 4S$ and $1M + 2S$ for the mentioned cases, respectively.

Combined approach for higher order composite operations *dP*

We remark that the previous generalization (3.14) is efficient for $d = 5, 7, 11$ (quintupling, septupling and eleventupling). For computations with d greater than those values, we propose a combined approach using the already efficient point tripling, quintupling and septupling operations. Moreover, we can nicely exploit an additional advantage of the presented approach: higher order composite operations internally compute lower composite operations. For instance, when computing the tripling of a point, one has the doubling for free. Similarly, when computing the quintupling of a point, one also obtains the doubling and tripling of such point, and so on.

Computation of higher order composite operations is described in the following (where d is an odd prime). We only show for the special case $a = -3$ (the general case easily follows). Also notice that for some computations involving triplings, the cost can be reduced further by applying the efficient tripling algorithm detailed in Appendix A.

- $13P = 2(5P) + 3P$, which requires one doubling, one quintupling and one general addition ➔ $30M + 16S$. We can slightly reduce this cost by using the efficient tripling (Appendix A) to compute $13P$ as $2^2(3P) + P$ ➔ $29M + 17S$.
- $17P = 2^4P + P$ ➔ $28M + 20S$ (scalars close to powers of two are evidently more efficient when only computed with doublings).
- $19P = 2(3^2P) + P$ ➔ $34M + 18S$ by using the efficient tripling given in Appendix A.

And so on.

Faster composite operation 5*P*+*Q* using 5*P*

In Section 3.2.3, we presented efficient $5P+Q$ using generalization $dP+Q$ (3.9). We can improve that result in the efficient case $a = -3$ by simply combining generalization dP (3.14) and traditional general additions.

Thus, by using our efficient quintupling followed by a general addition we obtain $5P+Q$ with only $26M + 12S$ for the special case ($a = -3$). This is even superior to our already improved $5P+Q$ from Section 3.2.3 ($28M + 11S$; see Table 3.1).

3.3.3 Performance comparison

Cost estimates of composite operations of form dP using generalization (3.14) are summarized in Table 3.2. Our best results are compared with traditional formulae.

With exception of tripling [DIM05], we could not find in the literature composite operations of the form dP in projective (Jacobian) coordinates over prime fields. Thus, for comparisons purposes, each "traditional" composite operation has been computed by using basic point operations (i.e., doubling, tripling and addition) in the most efficient way. $5P$ is computed by two consecutive doublings and a mixed addition; $7P$, by one doubling, one tripling and one addition; and $11P$ by two doublings, one tripling and one addition. Costs for the traditional basic operations were given in Chapter 2, Table 2.2, for the general (a random) and special ($a = -3$) cases.

Also, note that the traditional approach has been slightly improved for the general case by saving some operations during computation of repeated doublings (see Table 2.2).

In our case, we show cost of the tripling when computed with the efficient tripling algorithm in Appendix A. Quintupling, septupling and eleventupling were obtained from generalization (3.14).

Composite Operation	This work		Previous work	
	general	a = -3	general	a = –3
Tripling	9M + 7S	9M + 5S	10M + 6S	10M + 4S
Quintupling	14M + 10S	14M + 8S	20M + 14S	20M + 12S
Septupling	19M + 12S	19M + 10S	26M + 16S	26M + 12S
Eleventupling	29M + 16S	29M + 14S	30M + 20S	30M + 16S

Table 3.2. Cost performance of proposed composite operations of form dP in comparison with traditional formulae

After comparing costs given in Table 3.2, we can conclude that our composite operations outperform previous formulae in both the general and special cases. As one can see, the improvement is especially significant for quintupling and septupling, which will allow us to efficiently implement new multibase algorithms for the scalar multiplication in Chapter 7.

In the following section, we focus on some applications where our composite operations $dP+Q$ and dP can be efficiently applied.

3.4 Applications

In the present chapter, we have introduced efficient composite operations for ECC standard curves over prime fields by exploiting the special addition with identical z-coordinate introduced by [Mel06]. We explore *three* potential applications where these operations and the applied methodology can be nicely exploited:

1. For computation of pre-computed points, which are widely used in sliding window and window-w NAF scalar multiplications,

2. to speed-up existent scalar multiplications and,

3. to yield faster scalar multiplications using multiple bases.

In the following, we discuss the first two applications. The third one is extensively discussed in Chapter 7 with the introduction of new multibase scalar multiplications.

3.4.1 Computation of pre-computed points

Pre-computed points are extensively used to accelerate the scalar multiplication in applications where extra memory is available. Well-known methods in such category are window-*w* NAF and sliding window methods [HMV04]. Since these methods require more memory to store pre-computed points, constrained devices cannot normally afford the extra requirement. The problem is increased if those pre-computed points have to be computed only once at the start-up and then kept in memory for any required number of scalar multiplications. This is done to avoid the computation of the pre-computed table every time there is an execution of the scalar multiplication. In such case, there is a speed-up in the execution of the main operation, but the occupied memory is permanently unavailable to other internal operations.

A possible solution to the previous problem is to speed-up the execution of the pre-computed points as much as possible and, thus, makes its inclusion feasible at every execution of the scalar multiplication. In this way, extra memory would only be required during such operation and then released for other applications.

In the following, we apply the techniques introduced in this chapter to the pre-computation of points for the window-*w* NAF and the sliding window methods.

Window-*w* NAF (*w*NAF) and sliding window method

*w*NAF and sliding window methods rely on the pre-computation of points to reduce the Hamming weight of the binary expansion of the scalar d, and thus, reduce the cost during the scalar multiplication (see Chapter 2, Section 2.3.3). In particular, for *w*NAF one requires to build the next table [HMV04]:

$$\left\{1,3,5,...,2^{w-1}-1\right\} \tag{3.16}$$

And for the sliding window method:

$$\left\{1,3,5,...,\left[2\left(2^{w}-(-1)^{w}\right)/3\right]-1\right\} \tag{3.17}$$

We propose to compute the table by applying generalization (3.14). The only difference with our approach to derive composite operations (Section 3.3.2) is that in this case point P is assumed to be originally in affine coordinates. Thus, the first doubling to compute $2P$ should be in affine coordinates too.

By applying the mixed coordinates approach proposed in [CMO98], we can efficiently compute such first doubling in affine coordinates and yield the result in Jacobian coordinates, as follows:

$$X_2 = (3x_1^2 + a)^2 - 8x_1y_1^2, \qquad Y_2 = (3x_1^2 + a)(4x_1y_1^2 - X_2) - 8y_1^4, \qquad Z_2 = 2y_1 \tag{3.18}$$

Where the input and result are $P = (x_1, y_1)$ and $2P = (X_2, Y_2, Z_2)$, respectively.

Formula (3.18) is easily derived from (2.9) by applying (2.12) with $\lambda = 2y_1$, and it has a computing cost of only $2M + 4S$.

Then, the first addition (i.e., $3P=2P+P$) can be computed with a traditional mixed addition with Jacobian-affine coordinates (2.16):

$$\begin{aligned} X_3 &= (Z_2^3Y_1 - Y_2)^2 - (Z_2^2X_1 - X_2)^3 - 2X_2(Z_2^2X_1 - X_2)^2 \\ Y_3 &= (Z_2^3Y_1 - Y_2)\left[X_2(Z_2^2X_1 - X_2)^2 - X_3\right] - Y_2(Z_2^2X_1 - X_2)^3 \\ Z_3 &= Z_2(Z_2^2X_1 - X_2) \end{aligned} \tag{3.19}$$

Note that we can apply the special addition with identical z-coordinate (3.1) to compute ($3P+2P$) by choosing the following representative for $2P$ from (2.12) with $\lambda = Z_2^2X_1 - X_2$:

$$(X_2', Y_2', Z_2') = \left(X_2(Z_2^2 X_1 - X_2)^2, Y_2(Z_2^2 X_1 - X_2)^3, Z_2(Z_2^2 X_1 - X_2)\right) \equiv (X_2, Y_2, Z_2)$$

It is straightforward to note that, similarly to generalization (3.14), every extra addition can be computed with the special addition (3.1).

Finally, every point in the table has to be converted back to affine coordinates since this would permit to use the efficient mixed addition (2.16) during the scalar multiplication. This can be achieved by means of (2.11) for Jacobian coordinates:

$$(X / Z^2, Y / Z^3, 1) \tag{3.20}$$

Using (3.20) naively to convert back all points in the table to affine coordinates would require several expensive inversions. We can use the method due to Montgomery, called simultaneous inversion [HMV04], to replace inversions by only *one* inversion and some extra multiplications.

A pseudocode for Montgomery's method is given in the following:

1st Compute $R = (Z_1 \times Z_2 \times \ldots \times Z_n)^{-1}$, where $Z_1, Z_2, \ldots, Z_n$ are *z*-coordinates of *n* points to be converted to affine coordinates.

2nd For $j = 1$ to n do: $Z_j^{-1} = R \times \prod_{\substack{i=1 \\ i \neq j}}^{n} Z_i$.

The previous algorithm requires only *one* inversion and $(3n - 3)$ multiplications.

Once Z_j^{-1} is computed for each point using Montgomery's method, *x*- and *y*-coordinates are easily derived by using (3.20) with a cost of $3M+1S$ per point.

To summarize, the cost of the described approach would involve one doubling (3.18), one mixed addition, $\left(\frac{L-3}{2}\right)$ special additions, one inversion, $\left(3\left(\frac{L-1}{2}\right)-3\right)$ multiplications and $(3M+1S)\left(\frac{L-1}{2}\right)$. Note that $\left(\frac{L-1}{2}\right)$ is the number of points in pre-computed tables (3.16) and (3.17), where *L* denotes the last point.

In the case of *w*NAF, the overall cost can be expressed by:

$$1I + (11 \times 2^{w-2} - 9)M + (3 \times 2^{w-2} + 2)S \tag{3.21}$$

Now, let's compare our approach with previous methods in the literature.

A naïve approach to compute pre-computed points in (3.16) and (3.17) would require one doubling and $\left(\frac{L-1}{2}\right)$ additions in affine coordinates. For *w*NAF, this is equivalent to 1D + $(2^{w-2}-1)$ A, where D and A are costs of point operations in affine coordinates (see (2.8) and (2.9)).

[CMO98] proposed a methodology (later revisited by [Elm06]) to compute the pre-computed table for *w*NAF by applying the Montgomery's method. Their approach is, to our knowledge, the fastest in the literature, and requires a cost given by:

$$(w-1)I + (5 \times 2^{w-2} + 2w - 12)M + (2^{w-2} + 2w - 5)S \tag{3.22}$$

In Table 3.3, we show costs for *w*NAF using the different methods for w = 3, 4, 5 and 6. Estimates to the right of operation costs consider $1S = 0.8M$ and $1I = 30M$.

As it has been previously discussed, inversions over prime fields are highly expensive. In particular, for efficient implementations it is recommended to use special forms for the prime p such as those recommended by NIST [NIST] or those known as Mersenne primes [Sol99]. In such case, the cost ratio I/M is generally greater than 30 [HMV04,BHL$^+$01,Ava04] and justifies to replace inversions as much as possible.

*w*NAF	This work		Naïve		[CMO98] [Elm06]	
$w = 3$	$1I + 13M + 8S$	$49M$	$2I + 4M + 3S$	$66M$	$2I + 4M + 3S$	$66M$
$w = 4$	$1I + 35M + 14S$	$76M$	$4I + 8M + 5S$	$132M$	$3I + 16M + 7S$	$112M$
$w = 5$	$1I + 79M + 26S$	$130M$	$8I + 16M + 9S$	$263M$	$4I + 38M + 13S$	$168M$
$w = 6$	$1I + 167M + 50S$	$237M$	$16I + 32M + 17S$	$526M$	$5I + 80M + 23S$	$248M$

Table 3.3. Performance of proposed approach and previous methods to compute pre-computed points for *w*NAF

As we can see in Table 3.3, our approach is superior to previous methods in all cases when using *w*NAF. In the most efficient case, improvement over best previous effort surpasses 30%. Similar results are observed for sliding window method.

3.4.2 Speeding-up existent scalar multiplications

Composite doubling-addition (DA) operation is a very common operation in most well-known scalar multiplication algorithms, where the binary representation of the scalar is efficiently scanned from left-to-right. Hence, the computation consists of successive doublings with intermittent additions that happen when a non-zero element is found. The number of additions, and consequently, the number of DAs in the expansion, tightly depends on the Hamming weight of the scalar.

Similarly, in simultaneous scalar multiplications of the form $dP + lQ$, where P, Q are points on the elliptic curve and d, l scalars, several doublings are executed before an addition with a term extracted from a pre-computed table is computed. For more details about simultaneous scalar multiplication (or multiple point multiplication), the reader is referred to [HMV04].

Thus, the introduced DA (Section 3.2.1) can be effectively used to accelerate the scalar multiplication by replacing every doubling followed by an addition. As mentioned, the improvement tightly depends on the Hamming weight (or joint Hamming weight in the case

of the simultaneous scalar multiplication) since it determines the number of doubling/addition pairs that can be replaced for our composite operation.

Table 3.4 [HMV04] gives theoretical estimates for the density using some of the most common methods, and the corresponding savings that are achieved by using the proposed DA.

Method	w	Pre-computed points	additions	savings
NAF	1	0	$0.3333n$	$32M$
Shamir's trick	1	2	$0.75n$	$72M$
Shamir's trick with JSF	-	3	$0.50n$	$48M$

Table 3.4. Savings introduced by using the proposed DA operation for different scalar multiplications (n = 160bits, $1S = 0.8M$)

Savings in Table 3.4 are estimated by multiplying the average number of additions expected in a given scalar multiplication method with the savings achieved when trading one doubling/addition pair for the proposed DA ($2S–1M=0.6M$ per DA; see Table 3.1).

Evidently, the improvement is proportional to the density of a particular method, and hence, more efficient when pre-computation is restricted as happens in memory-constrained implementations.

Chapter 4

Fast and Flexible Point Arithmetic over Prime Fields

In Chapter 3, we proposed to accelerate ECC computations by devising efficient composite operations. In the present chapter, we introduce an alternative methodology for accelerating the elliptic curve point formulae over prime fields.

Our innovative technique uses the substitution of multiplication with squaring and other cheaper operations, by exploiting the fact that field squaring is generally less costly than multiplication. By applying this substitution to traditional formulae, we obtain faster point operations in unprotected sequential implementations.

Moreover, we apply the technique to efficient composite operations introduced in the previous chapter to achieve further optimization.

Previous work presented in [Ber06] makes use of a direct algebraic substitution to replace one "even" field multiplication (i.e., a multiplication accompanied by a multiple of two such as 2*ab*) by *one* squaring and *three* field additions/subtractions in doubling and general addition formulae. However, our technique first efficiently modifies current formulae for a

specific point operation in such a way that allows maximum gain through the mentioned algebraic substitution, which is applied right after.

It is important to note that it is widely accepted the fact that one squaring is less computationally expensive than one multiplication in software platforms such as Pentium, SPARC, ARM and many others. In general, most implementations report $1S \approx 0.6M - 0.8M$ [BHL+01,GAS+05,LH00,GG03,Ava04]. In particular, some implementations using special primes and OEFs (Optimal Extension Fields) have reported *S*/*M* ratios as low as 0.6 – 0.67 [Ber,GPW+04,Woo01].

In the following, we first introduce our innovative methodology, and then apply it to the improvement of traditional and composite operations. The chapter finishes with a discussion about the cost reductions that were achieved.

4.1 Flexible Methodology for Replacing Multiplications by Cheaper Operations

The next algebraic substitution holds for elements in a given prime field:

$$ab = \frac{1}{2}\left[(a+b)^2 - a^2 - b^2\right] \tag{4.1}$$

The first observation is that if we apply (4.1) to traditional formulae given in Chapter 2, *one* field multiplication would be replaced by *three* squarings, *three* addition/subtractions and *one* division by two. Consequently, a direct replacement is inefficient, since we are considering $1S = 0.8M$ or $1S = 0.6M$. However, we will show that redundancy in the ECC point arithmetic formulae over prime fields avoids the necessity of computing *two* out of the *three* squarings, reducing the total cost to *one* squaring, *three* addition/subtractions and *one* division by two. At this point, this trade would be advantageous, recalling the fact that field addition, subtraction and division by two are negligible in comparison to field multiplication

and squaring over large prime fields.

We still have to decide how to divide a given number by two. A possible solution is to transform the division by two to its inverse $2^{-1} \bmod p$ and then multiply the result (or compute several additions, depending on its value). However, because it is assumed that we are working with a big prime p for the underlying field, the inverse $2^{-1} \bmod p$ would be a very large number, and consequently, require too many additions, or at most a whole field multiplication.

To solve this problem more efficiently, we propose to choose another representative from the projective equivalence class (see Chapter 2, Section 2.3.2) that inserts multiples of two into the formulae.

The equivalence class that contains the Jacobian coordinates (X, Y, Z) is given by (2.12). Thus, by defining $\lambda = 2$ in (2.12), we get an equivalent representative for the point (X, Y, Z) of form $(2^3 X, 2^2 Y, 2Z)$, which efficiently inserts multiples of two into the formulae.

The latter would permit the transformation of the original algebraic substitution (4.1) to the next form for an "even" field multiplication, eliminating the division by two:

$$2ab = (a+b)^2 - a^2 - b^2 \tag{4.2}$$

Our flexible methodology can be summarized in two steps:

1st Modify, if necessary, the point formulae by inserting multiples of two via selection of the following representative of the equivalence class for Jacobian coordinates:

$$(X, Y, Z) \equiv (2^2 X, 2^3 Y, 2Z) \tag{4.3}$$

2nd Replace inserted or existent "even" field multiplications by applying the algebraic substitution given in (4.2), depending on the requirements of the targeted application.

The high flexibility of this methodology permits to efficiently adapt the point formulae to each application in such a way that the maximum cost reduction is achieved.

In particular, we will show that substitutions in step 2 of our methodology are closely related to the targeted application. For instance, in unprotected sequential operations, we must replace *one* "even" multiplication by only *one* squaring so that the cost (and number of operations) is kept to the minimum possible, whereas in parallel implementations (see Chapter 6), in some cases we can replace *one* "even" multiplication by up to *two* squarings to take advantage of the multiple processing units, and this way, reduce the cost further.

4.1.1 Fast Point Formulae for Traditional Operations

In this section, we apply the methodology introduced in the previous section to derive fast formulae for basic ECC point operations. As explained previously, to achieve maximum gain in a sequential software-based implementation, we should replace *one* multiplication for only *one* squaring.

Fast point doubling

Observing (2.13), we can easily detect two multiplications that can be directly replaced by squarings using the algebraic substitution (4.2): $Z_3 = 2Y_1Z_1$ and $\beta = 4X_1Y_1^2$. We do not have to worry about divisions by two because these two terms already contain multiples of two:

$$Z_3 = 2Y_1Z_1 = \left(Y_1 + Z_1\right)^2 - Y_1^2 - Z_1^2$$

$$\beta = 4X_1Y_1^2 = 2\left[\left(X_1 + Y_1^2\right)^2 - X_1^2 - Y_1^4\right]$$

Note that each of the previous multiplications is replaced by only one squaring and some extra additions, since the rest of the squarings (Y_1^2, Z_1^2, X_1^2 and Y_1^4) are already computed in the doubling formulae. No other multiplication can be efficiently replaced by squarings

because it would add more than one squaring to the formula.

The revised doubling formula is as follows:

$$X_3 = \alpha^2 - 2\beta, \qquad Y_3 = \alpha(\beta - X_3) - 8Y_1^4, \qquad Z_3 = (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2 \tag{4.4}$$

Where: $\alpha = 3X_1^2 + aZ_1^4$

$\beta = 2\left[(X_1 + Y_1^2)^2 - X_1^2 - Y_1^4\right]$

Given (4.4), the cost of a doubling is reduced from 4*M* + 6*S* to 2*M* + 8*S*, trading *two* field squarings for *two* multiplications.

If we fix $a = -3$, there is a computing reduction by applying the factorization technique in (2.14). Note that computation of X_1^2 is avoided, and consequently, computing $\beta = 2\left[(X_1 + Y_1^2)^2 - X_1^2 - Y_1^4\right]$ is not an improvement anymore since one multiplication would be replaced by two squarings instead of only one.

The doubling formula for the special case $a = -3$ is as follows:

$$X_3 = \alpha^2 - 2\beta, \qquad Y_3 = \alpha(\beta - X_3) - 8Y_1^4, \qquad Z_3 = (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2 \tag{4.5}$$

Where: $\alpha = 3(X_1 + Z^2)(X_1 - Z^2)$

$\beta = 4X_1Y_1^2$

And the cost of a doubling is reduced from 4*M* + 4*S* to 3*M* + 5*S*.

Further improvement can be achieved for implementations where squaring is relatively cheap in comparison with multiplication (i.e., 1*S* = 0.6*M*). In this case, considering *a* sparse (sparse meaning very low Hamming weight), (4.4) is the most efficient doubling formula with a cost of only 1*M* + 8*S*, given that cost of a multiplication by a constant can be considered negligible [Ber,Ber06]. This is even more efficient than the case $a = -3$, and less restrictive in the choice of the parameter *a*.

Fast point addition

Observing (2.15), we can quickly detect one multiplication that can be replaced by one squaring using the algebraic substitution (4.1): Z_1Z_2 in Z_3. However, this term lacks a multiple of two to avoid the division by two. To solve this problem, we follow the methodology given in Section 4.1. Thus, the formulae are first transformed as follows:

$$X_3 = \alpha^2 - 4\beta^3 - 8Z_2^2X_1\beta^2, \quad Y_3 = \alpha\left(4Z_2^2X_1\beta^2 - X_3\right) - 8Z_2^3Y_1\beta^3, \quad Z_3 = 2Z_1Z_2\beta \tag{4.6}$$

Where: $\alpha = 2\left(Z_1^3Y_2 - Z_2^3Y_1\right)$

$\beta = Z_1^2X_2 - Z_2^2X_1$

In this modified formulae, the term Z_1Z_2 has been replaced by $2Z_1Z_2$ allowing the computation: $2Z_1Z_2 = \left(Z_1 + Z_2\right)^2 - Z_1^2 - Z_2^2$. Note that *one* multiplication is being replaced by only *one* squaring and some extra additions, since the rest of squarings (Z_1^2 and Z_2^2) is already computed in the addition formula. Again, no other multiplication can be efficiently replaced by squaring because it would add more than one squaring to the formula.

The revised addition formula is as follows:

$$X_3 = \alpha^2 - 4\beta^3 - 8Z_2^2X_1\beta^2, \quad Y_3 = \alpha\left(Z_2^2X_1\beta^2 - X_3\right) - Z_2^3Y_1\beta^3, \quad Z_3 = \theta\beta \tag{4.7}$$

Where: $\alpha = 2\left(Z_1^3Y_2 - Z_2^3Y_1\right)$

$\beta = Z_1^2X_2 - Z_2^2X_1$

$\theta = \left(Z_1 + Z_2\right)^2 - Z_1^2 - Z_2^2$

Given (4.7), the cost of an addition is reduced to $11M + 5S$, trading *one* multiplication for *one* squaring in the traditional formula.

Fast mixed addition

Following the methodology given for the general point addition, we can derive a more

efficient mixed addition formula by trading multiplications by squarings. In this case, the revised formula is as follows:

$$X_3 = \alpha^2 - 4\beta^3 - 8X_1\beta^2, \quad Y_3 = \alpha\left(4X_1\beta^2 - X_3\right) - 8Y_1\beta^3, \quad Z_3 = \left(Z_1 + \beta\right)^2 - Z_1^2 - \beta^2 \qquad (4.8)$$

Where: $\alpha = 2\left(Z_1^3 Y_2 - Y_1\right)$

$\beta = Z_1^2 X_2 - X_1$

It is important to note that the term $Z_3 = 2Z_1\beta$ (from (4.6), with $Z_2 = 1$) has been replaced by $Z_3 = \left(Z_1 + \beta\right)^2 - Z_1^2 - \beta^2$, reducing the cost of the mixed addition to $7M + 4S$.

4.1.2 Fast Point Formulae for Composite Operations

In the following, we reduce the computing time to perform the new composite operations presented in Chapter 3. These improvements highlight the potential of combining introduced methodologies to obtain highly optimized point operations.

Fast point tripling

Substituting squarings for multiplications gives the greatest reduction in costs in the tripling formula, given the rich redundancy found in this operation. Observing (2.17), we can quickly detect up to *three* multiplications that can be replaced by squarings using the algebraic substitution (4.1): $Z_3 = Z_1\omega$, $\alpha = \theta\omega$ and $12X_1Y_1^2$. To insert the necessary multiples of two to the two former terms, we first apply step 1 of our flexible methodology to formula (2.17):

$$X_3 = 16Y_1^2\left(2\beta - 2\alpha\right) + 4X_1\omega^2, \qquad Y_3 = 8Y_1\left[\left(2\alpha - 2\beta\right)\left(4\beta - 2\alpha\right) - \omega^3\right], \qquad Z_3 = 2Z_1\omega$$

Where: $2\alpha = 2\theta\omega$

$2\beta = 16Y_1^4$

$$\theta = 3X_1^2 + aZ_1^4$$
$$\omega = 12X_1Y_1^2 - \theta^2$$

Now we can apply the algebraic replacement (4.2) as follows:

$$X_3 = 16Y_1^2\left(2\beta - 2\alpha\right) + 4X_1\omega^2 \tag{4.9}$$
$$Y_3 = 8Y_1\left[\left(2\alpha - 2\beta\right)\left(4\beta - 2\alpha\right) - \omega^3\right]$$
$$Z_3 = \left(Z_1 + \omega\right)^2 - Z_1^2 - \omega^2$$

Where:
$$2\alpha = \left(\theta + \omega\right)^2 - \theta^2 - \omega^2$$
$$2\beta = 16Y_1^4$$
$$\theta = 3X_1^2 + aZ_1^4$$
$$\omega = 6\left[\left(X_1 + Y_1^2\right)^2 - X_1^2 - Y_1^4\right] - \theta^2$$

Thus, the cost of a tripling has been efficiently reduced from $10M + 6S$ to $6M + 10S$.

A more efficient result is achieved by fixing a as a sparse number. In that case, the cost is further reduced to $5M + 10S$.

Again, for the case when $a = -3$, there is an additional reduction if we compute θ using the factorization technique given in (2.14). Note that computation of X_1^2 is avoided, and consequently, computing $\omega = 12X_1Y_1^2 - \theta^2 = 6\left[\left(X_1 + Y_1^2\right)^2 - X_1^2 - Y_1^4\right] - \theta^2$ is not an improvement anymore since one multiplication would be replaced by two squarings instead of only one.

Thus, the revised tripling formulae when $a = -3$ is as follows:

$$X_3 = 16Y_1^2\left(2\beta - 2\alpha\right) + 4X_1\omega^2 \tag{4.10}$$
$$Y_3 = 8Y_1\left[\left(2\alpha - 2\beta\right)\left(4\beta - 2\alpha\right) - \omega^3\right]$$

$$Z_3 = (Z_1 + \omega)^2 - Z_1^2 - \omega^2$$

Where: $2\alpha = (\theta + \omega)^2 - \theta^2 - \omega^2$

$2\beta = 16Y_1^4, \quad \theta = 3(X + Z^2)(X - Z^2)$

$\omega = 12X_1Y_1^2 - \theta^2$

In this case, the cost of the tripling is reduced further to 7*M* + 7*S*. Note that this formulae is even more efficient than the tripling algorithm proposed in Appendix A (9*M* + 5*S*), since it trades *two* expensive field multiplications for *two* squarings.

Fast point composite operations

In Chapter 3, Section 3.3.2, we introduced generalization (3.14) to yield composite operations of the form *dP*. We notice that this generalization consists of one traditional doubling followed by several special additions (3.1). Therefore, we can speed-up the internal doubling with our methodology, as shown in Section 4.1.1, and trade *two* multiplications for *two* squarings in the general case, and *one* multiplication for o*n*e squaring in the efficient case $a = -3$. It is important to note that there is no multiplication that can be efficiently replaced by squaring in the special addition with identical *z*-coordinate (3.1).

Thus, the cost formula (3.15) for composite operations of the form *dP* keeps the same, with the difference that D (cost of a doubling) is adjusted to D = 3*M* + 5*S* when considering $a = -3$ (formulae (4.5)). Otherwise, D = 2*M* + 8*S* (formulae (4.4)).

Applying these modifications to (3.15), the new cost for point quintupling, septupling and eleventupling are estimated as follows:

- Cost of the quintupling is reduced from 14*M* + 8*S* (Table 3.2) to 13*M* + 9*S* when $a = -3$, and from 14*M* + 10*S* to 12*M* + 12*S*, when *a* is randomly chosen.

- Cost of the septupling is reduced from 19*M* + 10*S* (Table 3.2) to 18*M* + 11*S* when $a = -3$, and from 19*M* + 12*S* to 17*M* + 14*S*, when *a* is randomly chosen.

- Cost of the eleventupling is reduced from $29M + 14S$ (Table 3.2) to $28M + 15S$ when $a = -3$, and from $29M + 16S$ to $27M + 18S$, when a is randomly chosen.

4.2 Performance Comparison

Tables 4.1 and 4.2 summarize our achievements with the new methodology of replacing multiplications for squarings. We distinguish three cases: when a has any possible value (first column, general case), when a is defined as sparse (second column) and for the special case $a = -3$ (third column). Costs for traditional operations are taken from Tables 2.1 and 3.2. Where applicable, costs of "traditional" composite operations are estimated using approach detailed in Chapter 3, Section 3.3.3.

As shown in those tables, our formulae replace expensive multiplications by squarings. In some highly efficient cases, such as the proposed fast tripling (Table 4.2), we replace up to *four* squarings with *four* multiplications in the original formulae given by [DIM05].

Operation	general case	a sparse	$a = -3$
Fast doubling	$2M + 8S$	$1M + 8S$	$3M + 5S$
Traditional doubling	$4M + 6S$	$3M + 6S$	$4M + 4S$
Fast addition	$11M + 5S$	-	-
Traditional addition (2.15)	$12M + 4S$	-	-
Fast mixed addition	$7M + 4S$	-	-
Traditional mixed addition	$8M + 3S$	-	-

Table 4.1. Cost of proposed Fast Point Operations in comparison with traditional formulae

Operation	general case	a sparse	$a = -3$
Fast tripling	$6M + 10S$	$5M + 10S$	$7M + 7S$
Revised tripling (App. A)	$9M + 7S$	$8M + 7S$	$9M + 5S$
Tripling [DIM05]	$10M + 6S$	-	$10M + 4S$
Fast w-tripling	$(6w)M + (10w)S$	$(5w)M + (10w)S$	$(7w)M + (7w)S$
w-tripling [DIM05]	$(11w-1)M + (4w+2)S$	-	-
Fast quintupling	$12M + 12S$	-	$13M + 9S$
Quintupling (Table 3.2)	$14M + 10S$	-	$14M + 8S$
Traditional quintupling	$20M + 14S$	-	$20M + 12S$
Fast septupling	$17M + 14S$	-	$18M + 11S$
Septupling (Table 3.2)	$19M + 12S$	-	$19M + 10S$
Traditional septupling	$26M + 16S$	-	$26M + 12S$
Fast eleventupling	$27M + 18S$	-	$28M + 15S$
Eleventupling (Table 3.2)	$29M + 16S$	-	$29M + 14S$
Traditional eleventupling	$30M + 20S$	-	$30M + 16S$

w = number of repeated triplings.

Table 4.2. Costs of Fast Composite Operations in comparison with traditional formulae

Proposed formulae are shown to be more efficient than traditional formulae in all cases, assuming that squaring is cheaper than multiplication. However, we remark that the improvement is more significant in applications where squarings are relatively very cheap in comparison with multiplications (i.e., $1S = 0.6M$).

Also, we must point out that defining a sparse in doublings and triplings does not represent the most efficient case in the traditional operations. However, in our fast point formulae, a sparse yields the most efficient implementation in cases where squarings are relatively cheap. For instance, considering a sparse in our fast doubling and tripling operations is more efficient than the general case or when $a = -3$, if $1S = 0.6M$.

In addition, [DIM05] proposed a repeated tripling formula that efficiently trades *one* multiplication for *two* squarings at every repeated tripling. However, we can see from Table 4.2 that our fast tripling, without extra modifications needed, is more efficient if consecutive triplings are to be computed. For instance, if $1S = 0.8M$, [DIM05] needs $(14.2w+0.6)$ field multiplications, whereas our formula only requires $(14w)$ field multiplications. If $1S = 0.6M$, the advantage of our formula is further increased. Furthermore, the reader must note that more efficient repeated triplings can be computed when our formula fixes a sparse or $a = -3$.

Chapter 5

SSCA-Protected Point Arithmetic using Side-Channel Atomicity

In the two previous chapters we introduced innovative methodologies to accelerate ECC computations via composite operations and by replacing multiplications for cheaper operations. In the present chapter, we focus on optimizing SSCA-protected formulae for ECC over prime fields on top of developed formulae.

Since introduced by [CCJ04], side-channel atomicity has been the preferred technique to protect implementations against Simple Side-Channel Attacks (SSCA), mainly because of its inexpensive overhead cost when compared against previous techniques. However, atomic blocks as originally proposed for curves over prime fields have two disadvantages that avoid further reduction in their computing time: parameter a is not fixed to an efficient value (i.e., $a = -3$), and structure of the atomic block is not optimal.

First, we will tackle the first problem, and propose, to our knowledge, the first atomic formulae that fix $a = -3$ to achieve reduced costs. Then, we will accelerate the formulae by introducing *two* new atomic structures that minimize the number of required field operations per block. For the latter, our improvement will be shown to be two-fold: on one side, we

reduce further the cost of the atomic formulae by minimizing the number of required field additions and introducing for first time the use of squarings, which are generally cheaper than multiplications; and, on the other side, we potentially increase the security offered by atomicity by distinguishing squarings from multiplications.

We end the chapter with a discussion of the significant increment in performance that is achieved using the new atomic formulae for the computation of the scalar multiplication.

5.1 Side-Channel Attacks

Side-channel information, such as power dissipation and electromagnetic emission, leaked by real-world devices has been shown to be highly useful for guessing private keys and effectively breaking the otherwise mathematically-strong ECC cryptosystem [Koc96,KJJ99,Ava05].

There are two main strategies to these attacks: simple (SSCA) and differential (DSCA) side-channel attacks. In this work, we focus in SSCA, which is based on the analysis of a single execution trace of a scalar multiplication to guess the secret key by revealing the sequence in the execution of point operations.

Extensive research has been carried out to yield effective countermeasures to deal with SSCA. Among them, we could mention indistinguishable operations via dummy instructions, scalar multiplication with a fixed sequence of group operations (e.g., Coron's countermeasure double-and-add-always method [Cor99]), unified addition and doubling formulae (e.g., the Jacobi and Hessian forms [LS01,BJ02,Sma01]) and side-channel atomicity [CCJ04].

The two first methods are in general highly expensive and quite susceptible to fault attacks. Using unified addition and doubling formulae has the drawback of being expensive and/or relying on special curves that are different from the standard curve (2.2). A highly

efficient variation of the scalar multiplication with a fixed sequence of group operations is based on the Montgomery Ladder [FGK+02,IT02b,IT05]. However, similarly to the unified operation approach, the most efficient version of the Montgomery Ladder also relies on a non-standardized curve form, namely the Montgomery curve.

Side-channel atomicity, proposed in [CCJ04], dissolves point operations into small homogenous blocks, known as atomic blocks, which cannot be distinguished from one another through simple side-channel analysis because each one contains the same pattern of basic field operations. Furthermore, atomic blocks are made sufficiently small to make this approach inexpensive. [CCJ04] proposed the *M-A-N-A* (field multiplication, addition, negation, addition) structure to build SSCA-protected doubling and general addition formulae over prime fields. [Mis06] and [DIM05] optimized the addition case by introducing the atomic formulae for mixed addition. Authors in [DIM05] also developed the atomic tripling formulae.

However, traditional *M-A-N-A* structure has two main drawbacks: parameter a in elliptic curve (2.2) is not fixed to -3, and the number of blocks per atomic block is not optimal. In that sense, *M-A-N-A*-based formulae with $a = -3$ and new atomic structures *M-N-A-M-N-A-A* and *S-N-A-M-N-A-A* are introduced to tackle both problems.

In addition, one potential drawback in the traditional *M-A-N-A* structure is that it relies on the assumption that field multiplication and squaring are indistinguishable from each other. In software implementations, timing and power consumption have been shown to be quite different for squarings and multiplications, making these operations directly distinguishable through power analysis [GAS+05,GG03]. The next attack can be conceived in such a case: by observing only one electromagnetic (EM) or power trace, an attacker may be able to detect which portions of the scalar multiplication are in fact executing a squaring. With that knowledge, he/she can now gain access to the point doubling/addition sequence (and, consequently to the secret key), given that the atomic addition has far more multiplications than squarings and its pattern of squaring/multiplications is very different from the pattern for the atomic doubling.

Hardware platforms (or implementations with an ECC co-processor) can be thought to be invulnerable to such attack because one hardware multiplier executes both field squaring and multiplication. However, reported attacks have shown that in fact multiplying two identical numbers (as happens during squaring in a modular hardware multiplier) can behave differently than multiplying two different numbers. For instance, research presented in [WT01] shows how Montgomery multipliers can be effectively attacked to distinguish multiplications from squarings by exploiting the difference in the occurrence of one conditional subtraction for those operations. This, or other not so evident vulnerabilities in a hardware multiplier, can be exploited by clever attackers to distinguish the field operations mentioned. Also, [Wal01] proposed a high-order DPA attack to defeat an RSA implementation by distinguishing multiplications with pre-computed points from squarings and multiplications with random numbers through power analysis. This work suggests that, similarly, power traces of multiplications with random numbers can be distinguished from power traces of squarings.

The previous conclusions can be directly applied to ECC cryptosystems, and exploited to implement the attack described previously. Although more research is required to assess effectiveness of these and related attacks in hardware implementations, the smart developer would take into account some precautions when implementing side-channel atomicity.

In the last part of the present chapter (Section 5.4), we exploit the flexibility given by our technique of replacing multiplications by squarings to propose a more efficient atomic structure (*S*-*N*-*A*-*M*-*N*-*A*-*A*) that effectively takes into account squarings in its formulation. The latter makes our atomic structure not only faster but also invulnerable to attacks that exploit distinctions between multiplications and squarings, as the one described in this section.

This way, new atomic formulae for point doubling, mixed addition and tripling have been developed and shown to be considerably faster than previous efforts. The increase in speed comes from the fact that squarings are generally less expensive than multiplications and because our methodology, in combination with the improved atomic structure *S*-*N*-*A*-*M*-*N*-

A-*A*, permits us to pack more field operations in one atomic block.

5.2 Improved Atomic Formulae for Traditional Doubling and Tripling Operations

[CCJ04] introduced the concept of atomicity to protect public-key cryptosystems against simple side-channel attacks. They proposed atomic doublings and additions built with *ten* and *sixteen M*-*A*-*N*-*A* atomic blocks, respectively. Thus, a doubling costs $10M + 20A$, and an addition, $16M + 32A$. [Mis06] presented an improved atomic operation for the case of mixed addition. The number of atomic blocks for addition was reduced to *eleven*, with a total cost of $11M + 22A$. Later, [DIM05] presented a fast tripling formula with *sixteen* atomic blocks using the same atomic structure, with a total cost of $16M + 32A$.

We have derived the atomic doubling for the case $a = -3$ using formulae (2.13). Appendix B1 shows the new efficient SSCA-protected doubling. The number of blocks is reduced from *ten* to *eight*, which gives a total cost of $8M + 16A$.

Similarly, we have derived the efficient case with $a = -3$ for the point tripling using the improved algorithm detailed in Appendix A. The required number of blocks is reduced from *sixteen* to only *fourteen* blocks, which gives a total cost of $14M + 28A$. The details of the SSCA-protected tripling are shown in Appendix B9.

5.3 Enhanced Atomic Structure

5.3.1 Atomic Formulae for Traditional Operations

By a careful analysis of previously improved atomic point operations, we can notice that

doubling and tripling have an even number of blocks, which would potentially allow dividing each atomic block to approximately the half. Most importantly, the number of necessary dummy additions to complete the atomic formulae (see Appendices B1, B3 and B9) using *M-A-N-A* structure is relatively high. Hence, if we divide each block we could potentially reduce the number of required additions.

The only obstacle to the latter is that atomic addition has an odd number of blocks (*eleven*). Dividing this operation to the half would forcedly increase the number of field multiplications. Nevertheless, point additions are scarce in efficient scalar multiplications, and the significant reduction in the number of field additions would balance such increment in field multiplications.

We propose the next structure to further improve atomic formulae by reducing the number of required field additions: *M-N-A-M-N-A-A* (Multiplication-Negation-Addition-Multiplication-Negation-Addition-Addition).

Details of the new atomic formulae using this efficient structure are given in Appendices B2, B4 and B10. Atomic point doubling, tripling and addition have been derived from formulas (2.13), (2.16) and (2.17), respectively. With this structure, the number of blocks for doubling, tripling and addition has been reduced to *four*, *seven* and *six*, respectively, which gives the following costs (note that *M-N-A-M-N-A-A* structure contains $2M + 3A$ per block):

- Doubling: 4 blocks × $(2M + 3A)$ ➔ $8M + 12A$.
- Tripling: 7 blocks × $(2M + 3A)$ ➔ $14M + 21A$.
- Addition: 6 blocks × $(2M + 3A)$ ➔ $12M + 18A$.

5.3.2 Atomic Formulae for Composite Operations

In Section 3.2.1, we introduced a highly efficient doubling-addition (DA) operation by

taking advantage of the special addition with identical z-coordinate (3.1). If we protect this new composite operation against SSCA by applying atomicity, we should expect reduced costs in comparison with a consecutive execution of traditional atomic doubling followed by addition.

As a direct consequence of performance analysis in Section 3.4.2, improvement of such composite operation in its SSCA-protected version would lead to a reduction in computing costs directly proportional to the Hamming weight of the scalar multiplication being used. Similarly, in the present case the gain obtained from using this composite operation is maximized for high density values, making this technique very useful for applications with constrained resources where scalar multiplications cannot typically make use of pre-computations.

Following generalization (3.9) for the case $2P+Q$, DA consists of a mixed addition followed by a special addition with identical z-coordinate. Hence, by executing atomic formulae in Appendix B4 (corresponding to atomic mixed addition) followed by formulae in Appendix B6 (corresponding to atomic special addition), we achieve an SSCA-protected DA operation using the proposed *M-N-A-M-N-A-A* structure. Similarly, by unifying formulae in Appendices B3 and B5, we achieve the analogue for *M-A-N-A*.

It is important to note that availability of dummy operations in the last block of the atomic mixed addition can be efficiently used by field operations from the special addition to reduce *one* atomic block. Thus, first block ($\Delta 1$) of the atomic special addition can be completely integrated with blocks $\Delta 6$ or $\Delta 11$ for *M-N-A-M-N-A-A* and *M-A-N-A* cases, respectively. See Appendices B5 and B6 for more details.

Thus, the costs for the atomic DA are given by:

- *M-N-A-M-N-A-A* structure: 9 blocks × $(2M + 3A)$ ➔ $18M + 27A$.
- *M-A-N-A* structure: 18 blocks × $(1M + 2A)$ ➔ $18M + 36A$.

The previous results should be compared with the combined cost of one atomic doubling and one atomic addition. In the traditional *M-A-N-A* case, the cost of a doubling followed by an addition is $21M + 42A$; with the improved atomic operations from Section 5.1, the cost is only $19M + 38A$; and with the new *M-N-A-M-N-A-A* structure the cost is reduced further to $20M + 30A$ (Section 5.2.1).

Given the previous results, we can conclude that the new atomic DA introduces a significant reduction, especially in the case of the new *M-N-A-M-N-A-A* structure.

Also note that improvement is even more significant for applications that consider the general case (i.e., a is not fixed to -3), since the new DA does not depend on parameter a and, consequently, its cost keeps the same. In such case, the comparison is limited to only the traditional *M-A-N-A*, and the reduction is fixed in $3M + 15A$ (best case with *M-N-A-M-N-A-A* structure).

Composite operations *dP*

Similarly to the atomic DA, the tripling of a point is obtained by performing a doubling followed by a special addition (3.1). By unifying the atomic doubling operations given in Appendices B1 and B2 with the corresponding atomic formulae for the special addition (Appendices B5 and B6), we achieve atomic formulae for the tripling introduced in Section 3.3.1. The costs for the atomic tripling are fixed in $16M + 24A$ and $15M + 30A$ for *M-N-A-M-N-A-A* and *M-A-N-A* structures, respectively, and the details shown in Appendices B8 and B7.

However, we have already introduced highly efficient atomic formulae for tripling with costs of $14M + 21A$ and $14M + 28A$ for the same atomic structures (see Section 5.1 and 5.2.1, and Appendices B9 and B10).

Nevertheless, the atomic tripling can be used to derive efficient atomic formulae for quintuplings and septuplings, which is detailed in the following paragraphs.

Following generalization (3.14), the quintupling of a point consists of a tripling ($2P + P$) followed by a special addition (3.1) with $2P$. Thus, by unifying atomic tripling formulae from Appendices B7 and B8 (for *M-A-N-A* and *M-N-A-M-N-A-A* structures, respectively) with corresponding formulae in Appendices B5 and B6 for the special addition, we obtain the atomic quintupling. Note that, similarly to the atomic DA, the special addition can be reduced in *one* atomic block for each case by integrating its first atomic block with the last atomic block of the tripling, as shown in Appendices B7 and B8 for each structure.

Atomic formulae for higher composite operations can be derived similarly to the quintupling by simply following generalization (3.14). Thus, for the septupling is only necessary to add one special addition (3.1) with $2P$. Details are shown in Appendices B7 and B8 for each structure.

The total cost for an atomic operation of the form dP, where $d \geq 5$ is an odd prime, is given by the following cost formulas:

$$\begin{aligned} &M\text{-}N\text{-}A\text{-}M\text{-}N\text{-}A\text{-}A \text{ structure:} \quad \mathrm{T}_a + \left\lfloor \frac{d-1}{4} \right\rfloor \mathrm{A}'_a + \left\lfloor \frac{d-3}{4} \right\rfloor \mathrm{A}'_b \\ &M\text{-}A\text{-}N\text{-}A \text{ structure:} \quad \mathrm{T}_a + \left(\frac{d-3}{2} \right) \mathrm{A}'_a \end{aligned} \tag{5.1}$$

Where T_a denotes the cost of an atomic tripling using generalization (3.14), i.e., $16M + 24A$ and $15M + 30A$ for *M-N-A-M-N-A-A* and *M-A-N-A* structures, respectively. A'_b denotes the cost of an atomic special addition, and A'_a denotes the cost of an atomic special addition reduced in *one* atomic block. The last reduction is possible due to the fact that every extra addition can save one atomic block by integrating its first atomic block with the last block of the precedent operation for the *M-A-N-A* structure (see Appendix B5). For the *M-N-A-M-N-A-A* structure, such reduction can be achieved every two operations (see Appendix B6 for details). Thus, the effective costs of the atomic special addition are fixed to: $\mathrm{A}'_b = 8M + 12A$, and $\mathrm{A}'_a = 6M + 9A$ or $7M + 14A$ for *M-N-A-M-N-A-A* and *M-A-N-A* structures, respectively.

Using (5.1), the costs for the atomic quintupling are given by:

- *M-N-A-M-N-A-A* structure: $(16M + 24A) + (6M + 9A) \rightarrow 22M + 33A$.
- *M-A-N-A* structure: $(15M + 30A) + (7M + 14A) \rightarrow 22M + 44A$.

And for the atomic septupling:

- *M-N-A-M-N-A-A* structure: $(16M + 24A) + (6M + 9A) + (8M + 12A) \rightarrow 30M + 45A$.
- *M-A-N-A* structure: $(15M + 30A) + 2(7M + 14A) \rightarrow 29M + 58A$.

Table 5.1 summarizes costs of introduced composite operations and compares them with traditional atomic formulae. Since, to our knowledge, this is the first effort to develop atomic formulae for higher order operations such as quintupling, septupling, and so on, "traditional" composite operations shown in the table are built by using the best atomic doubling, tripling and addition formulae existent in the current literature (see Tables 5.2 and 5.3, previous work). For instance, the cost of a "traditional" quintupling is estimated as $5P = 2^2P + P$ (two doublings and one general addition).

As we can see, new atomic formulae are significantly more efficient than a traditional implementation. Let the ratio A/M be 0.05 for a very efficient implementation as the one presented by [Ber]. Our atomic formulae (best case, *M-N-A-M-N-A-A* structure) reduce computing costs in 40%, 30% and 17% for atomic quintupling, septupling and eleventupling, respectively.

Method	Previous work	This work [1]	This work [2]
Quintupling	$36M + 72A$	$22M + 44A$	$22M + 33A$
Septupling	$42M + 84A$	$29M + 58A$	$30M + 45A$
Eleventupling	$52M + 104A$	$43M + 86A$	$44M + 66A$

1: using *M-A-N-A* structure.
2: using *M-N-A-M-N-A-A* structure.

Table 5.1. Performance of new atomic composite operations using *M-A-N-A* and *M-N-A-M-N-A-A* structures, in comparison with traditional atomic formulae

5.4 New Highly-Secure Atomic Structure with Squarings

As previously pointed out, atomic blocks were originally *M-A-N-A* (Multiplication-Addition-Negation-Addition)–based structures for the case of point operations over prime fields [CCJ04]. For efficiency purposes, squarings and multiplications are considered side-channel equivalent [CCJ04,DIM05,Mis06], and consequently the atomic block efficiency heavily depends on cost of field multiplication.

The general assumption is that multiplication and squaring are indistinguishable from a side-channel analysis point of view. However, that is not the case in most efficient software implementations, where squaring is less expensive and obviously distinguishable from multiplication. Thus, an efficient atomic block should consider squaring in its structure. With previous formulae (traditional operations, Chapter 2; composite operations, Chapter 3) such consideration would have been very inefficient and expensive. However, the flexible methodology presented in Chapter 4 permits to modify the point formulae in such a way that allows us to easily balance the number of squarings and multiplications and, thus, to efficiently introduce squarings into the formulation.

In the remainder of this chapter, we present an innovative atomic structure based on *S-N-A-M-N-A-A* (Squaring-Negation-Addition-Multiplication-Negation-Addition-Addition) for the addition, doubling and tripling operations. We remark that this new structure is more efficient and truly protects against simple side-channel attacks because it takes into account

differences between field squarings and multiplications. Note that this structure only differs in the first field operation in comparison with the *M-N-A-M-N-A-A* structure presented in Section 5.3. Consequently, it also gains in efficiency by reducing the number of dummy operations required for field additions.

In the next paragraphs, we detail our methodology to derive atomic formulae with the new atomic structure, first for the case of scalar multiplication using only radix 2 for the scalar expansion, and then for the case of expansions including ternary bases.

Case with binary bases

The first step to achieving cheaper atomic blocks that include squarings and multiplications for a given point operation is to try to balance the number of each of these field operations in such point operations.

In the traditional point doubling formula, we observe that the number of multiplications and squarings is already balanced with the minimum number of operations when considering $a = -3$ (formulae (2.13)): $4M + 4S$. Thus, working with the aforementioned atomic structure (*S-N-A-M-N-A-A*), *four* atomic blocks are sufficient to accommodate the balanced doubling formula as each atomic block contains *one* field multiplication and *one* field squaring. Therefore, the cost of the atomic doubling is $4M + 4S$. The details are shown in Appendix C1.

For the case of mixed addition, we have the fast formula (4.8) whose cost is $7M + 4S$. We first need to balance the number of multiplications and squarings to achieve an efficient atomic structure. That can be achieved if one multiplication is replaced by two squarings as follows: $2Z_1^3Y_2 = \left(Z_1^3 + Y_2\right)^2 - Z_1^6 - Y_2^2$, with the term $-Y_2^2$ pre-computed.

Thus, the balanced formula is as follows:

$$X_3 = \alpha^2 - 4\beta^3 - 8X_1\beta^2, \quad Y_3 = \alpha\left(4X_1\beta^2 - X_3\right) - 8Y_1\beta^3, \quad Z_3 = \left(Z_1 + \beta\right)^2 - Z_1^2 - \beta^2 \qquad (5.2)$$

Where: $\alpha = \left(Z_1^3 + Y_2\right)^2 - Z_1^6 - Y_2^2 - 2Y_1$

$\beta = Z_1^2 X_2 - X_1$

The cost for this formulae is now 6*M* + 6*S*, containing the minimum possible number of operations when the number of field multiplications and squarings is equivalent. The cost of the atomic addition is fixed at 6*M* + 6*S*, as only *six* atomic blocks are required to accommodate formulae (5.2). Details of the new atomic addition are shown in Appendix C2.

Case with ternary bases

Scalar multiplications that take advantage of ternary bases to accelerate computations [CJL+06,DIM05] require tripling of a point beside doubling and addition. Given the number of field additions found in the tripling formulae (2.17) or (4.10), it is not possible to accommodate this operation with the optimal number of atomic blocks using the proposed *S-N-A-M-N-A-A* structure. Thus, an additional field addition per atomic block has been added to the previous atomic structure to permit high efficiency: *S-N-A-A-M-N-A-A*. In this way, we can accommodate the tripling formula without requiring extra multiplications or squarings, resulting in the cheapest atomic implementation known to date.

In the following paragraphs, we will present the modified atomic operations for scalar multiplications that require triplings, doublings and additions, as is the case of the double-base chains [DIM05] or the new multibase algorithms presented in Chapter 7.

In the case of the tripling operation, the optimal balance between multiplications and squarings can be found in formula (4.10), when $a = -3$, with a cost of 7*M* + 7*S*. Then, *seven* atomic blocks would be required to accommodate all these operations. However, because of the tripling structure and the internal dependences between field operations, one extra atomic block is necessary to accommodate the whole formula. Thus, *eight* atomic blocks are required in total for the tripling operation. If repeated triplings are computed, it is possible to reduce the cost to (7*w*+1)*M* + (7*w*+1)*S*, where *w* is the number of repeated triplings, by merging the last block of the current tripling with the first atomic block of the following

tripling, saving *one* field multiplication and *one* field squaring at every additional tripling operation. Details of the atomic tripling and atomic repeated tripling are presented in Appendix C3.

In addition, point doubling and addition operations must be modified according to the new atomic structure to make them suitable for scalar multiplications that include triplings in their computation. Basically, *four* and *six* extra additions has to be added to the previous (*S-N-A-M-N-A-A*)-based atomic doubling and mixed addition, as detailed in Appendices C1 and C2.

5.5 Performance comparison

In this section, we discuss the reduction in terms of computing costs achieved by the improved atomic doubling and tripling with fixed $a = -3$, the new atomic formulae for DA and the innovative atomic structures for the computation of the scalar multiplication. For all cases, *M-A-N-A*-based formulae are compared against the performance offered by the new *M-N-A-M-N-A-A* and *S-N-A-M-N-A-A* structures.

Table 5.2 summarizes performance of revised formulae when only point addition and doubling are used, as is the case for the traditional binary NAF and sliding window scalar multiplications. For the case where ternary bases are included into the computation of the scalar multiplication, Table 5.3 summarizes our results.

Method	Previous work	This work (1,a)	This work (2,a)	This work (3,a)
Doubling [CCJ04]	$10M + 20A$	$8M + 16A$	$8M + 12A$	$4M + 4S + 12A$
w-doubling [DIM05]	$(8w + 2)M + 2(8w + 2)A$	$8wM + 16wA$	$8wM + 12wA$	$4wM + 4wS + 12wA$
Mix. addition [Mis06]	$11M + 22A$	-	$12M + 18A$	$6M + 6S + 18A$

Table 5.2. Performance of proposed atomic operations using *M-A-N-A*, *M-N-A-M-N-A-A* and *S-N-A-M-N-A-A* structures, in comparison with traditional atomic operations

Method	Previous work	This work (1,a)	This work (2,a)	This work (3,a)
Doubling [CCJ04]	$10M + 20A$	$8M + 16A$	$8M + 12A$	$4M + 4S + 16A$
w-doubling [DIM05]	$(8w + 2)M + 2(8w + 2)A$	$8wM + 16wA$	$8wM + 12wA$	$4wM + 4wS + 6wA$
Mix. addition [Mis06]	$11M + 22A$	-	$12M + 18A$	$6M + 6S + 24A$
Tripling [DIM05]	$16M + 32A$	$14M + 28A$	$14M + 21A$	$8M + 8S + 32A$
w-tripling [DIM05]	$(15w + 1)M + 2(15w + 1)A$	$14wM + 8wA$	$14wM + 21wA$	$(7w + 1)M + (7w + 1)S + (28w + 4)A$

(a) Parameter a is fixed ($a = -3$).
(1) Improved atomic doubling and tripling operations with *M-A-N-A* structure.
(2) Improved atomic operations with *M-N-A-M-N-A-A* structure.
(3) Improved atomic operations with *S-N-A-M-N-A-A-A* structure.
w = number of repeated doublings or triplings.

Table 5.3. Cost of new atomic operations with *M-A-N-A*, *M-N-A-M-N-A-A* and *S-N-A-M-N-A-A-A* structures for scalar multiplications based on ternary bases in comparison with previous atomic operations with *M-A-N-A* atomic structure

As we can see in both tables, our atomic operations based on new structures are more efficient in all cases when compared against 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 (with exception of our atomic w-tripling for *S-N-A-M-N-A-A-A*), they still present a superior performance than previous efforts.

Also, it can be seen that point doubling and addition using the new *S-N-A-M-N-A-A*

structure achieve the lowest cost by minimizing the number of required field additions and replacing multiplications by squarings. Remarkably, we also observe that even in applications where squarings are considered as costly as multiplications (i.e., $1S = 1M$; if the same hardware multiplier is used to perform multiplication and squaring), our *S-N-A-M-N-A-A*-based atomic doubling and repeated doubling present a reduction of at least *two* field multiplications and *four* field additions in comparison to traditional formulae. In the case of triplings, the cost would be equivalent ($16M + 32A$), but when repeated triplings are computed, again our approach is superior, reducing the required number of multiplications and additions from $(15w+1)M+(30w+2)A$ to $(14w+2)M+ (28w+4)A$, which means an overall reduction of $(w-1)$ field multiplications and $2(w-1)$ field additions.

For the case including ternary bases, *M-N-A-M-N-A-A* structure offers the fastest tripling and the minimal number of field additions in all operations, highlighting the potential efficiency of this structure for multibase scalar multiplications using radix 3. In this case, point addition is still *one* multiplication more expensive than the traditional formulae. However, we expect that such disadvantage is minimized due to the scarcely occurrence of these operations during efficient scalar multiplications.

In the following, we compare performance when using scalar multiplications based on radix 2 (i.e., only using point doubling and addition). Performance achieved by scalar multiplications that use higher order composite operations are discussed in Chapter 7 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 *three* sets of atomic operations, and to determine which of these offers the best results, we compare performance when using a traditional scalar multiplication with NAF method and scalar d of length $n = 160$ bits. We already pointed out that NAF has a density of non-zero terms of approximately 1/3 (Section 2.3.3). Thus, for a 160-bit NAF scalar multiplication, one approximately requires 159D + 53A using (2.20).

The number of required operations when using our three sets of point operations and the

traditional approach are detailed in Table 5.4 for cases where a hardware multiplier executes both multiplications and squarings (and consequently, $1S = 1M$) and the general case for software implementations ($1S = 0.8M$). On the bottom of the table, we specify the ratio field addition/field multiplication for which our two first sets (1) and (2) offer equivalent performance.

Method	Cost ($1S = 1M$)	Cost ($1S = 0.8M$)
Traditional atomic operations	$2173M + 4346A$	$2173M + 4346A$
This work (1)	$1855M + 3710A$	$1855M + 3710A$
This work (2)	$1908M + 2862A$	$1908M + 2862A$
This work (3)	$1908M + 2862A$	$1717M + 2862A$
A/M between (1) and (2)	0.063	0.063

(1) Using improved atomic operations with *M-A-N-A* structure.
(2) Using improved atomic operations with *M-N-A-M-N-A-A* structure.
(3) Using improved atomic operations with *S-N-A-M-N-A-A* structure.

Table 5.4. Performance of proposed atomic operations using *M-A-N-A*, *M-N-A-M-N-A-A* and *S-N-A-M-N-A-A* structures, in comparison with traditional atomic operations (NAF method, $n = 160$ bits)

As it can be seen in Table 5.4, our atomic operations perform significantly better than the traditional operations. In particular, for the case $1S = 0.8M$, *S-N-A-M-N-A-A*-based formulae achieve the highest performance in terms of computing cost.

For the case $1S = 1M$, the relative ratio A/M would ultimately decide if set (1) or sets (2) and (3) offer the lowest cost. The break point between the first and the other two sets of proposed atomic operations for 160-bit NAF is given by the relative value $1A = 0.063M$. Specifically, improved *M-A-N-A* offers the highest performance if $1A < 0.063M$ in a given implementation. But if $1A > 0.063M$, the new *M-N-A-M-N-A-A* and *S-N-A-M-N-A-A* structures represent the best options.

Let the ratio A/M be 0.05 for a very efficient implementation as the one presented by

[Ber,Ber06]. Then, the improved *M-A-N-A* and the new *M-N-A-M-N-A-A* structure present a reduction of 14.6% and 14.2% in comparison with a NAF scalar multiplication using previous atomic operations based on *M-A-N-A*. On the other hand, if $1S = 0.8M$, the new *S-N-A-M-N-A-A* structure presents a significant reduction of about 22.2%. More dramatic improvement is expected on applications where $1S = 0.6M$.

Using the proposed DA operation

In Chapter 3, Section 3.2.1, we introduced a faster composite DA operation with a cost of only $13M + 5S$. Previously, we showed that this composite operation can be protected against SSCA using atomicity, yielding the fastest atomic implementation using a unified doubling/addition formulae. Table 5.5 summarizes the costs of the improved DA, and compares it with the traditional execution of a doubling followed by an addition.

Method	Cost
Traditional atomic Doubling/Addition	$21M + 42A$
Proposed atomic DA (1)	$18M + 36A$
Proposed atomic DA (2)	$18M + 27A$

(1) Using *M-A-N-A* structure.
(2) Using *M-N-A-M-N-A-A* structure.

Table 5.5. Performance of proposed atomic DA using *M-A-N-A* and *M-N-A-M-N-A-A* structures, in comparison with approach using traditional atomic operations

To determine which structure is more convenient for the proposed DA, we compare performance using the NAF method. For a 160-bit NAF scalar multiplication (see (2.20)), we approximately require $159\text{D} + 53\text{A} = (159–53)\text{D} + 53\text{DA} = 106\text{D} + 53\text{DA}$. The operation counting when using the proposed DA for both atomic structures and the traditional approach are detailed in Table 5.6. The cost of the doubling corresponding to each atomic structure for the proposed DA case has been taken from Table 5.2.

Also, we show previous results (from Table 5.4) obtained with our basic atomic operations for both atomic structures.

Method	Cost
Traditional atomic operations	$2173M + 4346A$
With proposed atomic DA (1)	$1802M + 3604A$
With proposed atomic DA (2)	$1802M + 2703A$
Using basic atomic operations (1)	$1855M + 3710A$
Using basic atomic operations (2)	$1908M + 2862A$

(1) Using *M-A-N-A* structure.
(2) Using *M-N-A-M-N-A-A* structure.

Table 5.6. Performance of proposed atomic DA using *M-A-N-A* and *M-N-A-M-N-A-A* structures, in comparison with the traditional approach (NAF method, $n = 160$ bits)

As expected, the efficient atomic DA has significantly reduced the cost of the scalar multiplication. In particular, using the new *M-N-A-M-N-A-A* structure yields the highest performance since the disadvantage of having a slower addition has been cancelled by the composite DA operation, which is more efficient in this structure than with the traditional *M-A-N-A*.

For comparison purposes, let the ratio A/M be 0.05. Then, the improved scalar multiplication using DA with *M-A-N-A* and *M-N-A-M-N-A-A* structures present a reduction of 17.1% and 19%, respectively, in comparison with a NAF scalar multiplication using previous atomic operations based on *M-A-N-A*.

Previous conclusions hold for $1S = 1M$. However, notice that for case $1S = 0.8M$, *S-N-A-M-N-A-A*-based approach with a estimated cost of $1717M + 2862A$ (see Table 5.4) offers the lowest computing cost, beside increased protection when differences between squarings and multiplications are a security concern.

Chapter 6

Efficient Parallel Point Arithmetic

In recent years a new paradigm has arisen in the design concept with the appearance of multiprocessor/parallel architectures, which can execute several operations simultaneously. This topic is becoming increasingly important since single processor design is reaching its limits in terms of clock frequency. Among several parallel architectures, Single-Instruction Multiple-Data (SIMD) has become highly attractive since it generally avoids the higher hardware complexity needed in parallel architectures such as superscalar computers by leaving to the programmer the task of parallelizing the program execution. Hence, we can already find SIMD-based schemes in many popular processors such as Pentium, SPARC and PowerPC.

In the previous chapter, we showed how to take advantage of the flexibility offered by our methodology of replacing multiplications to include squarings into the atomic formulae. In the present chapter, we take advantage of such methodology but this time with the objective of deriving faster ECC formulae for parallel architectures such as SIMD.

Similarly to other systems, ECC can be adapted to parallel architectures at different mathematical levels (see Section 2.3). In this work, we focus our efforts to parallelize ECC formulae at the point arithmetic level.

The organization of this chapter is as follows. First, we describe most important efforts in the literature to derive efficient parallel point formulae. Then, we present our new parallel formulae that exploit the flexibility of methodology given in Chapter 4 and a new highly-efficient coordinate system. Thus, unprotected point operations with *three* and *four* simultaneous operations are developed for SIMD-based architectures where SSCA attacks are not a concern. Following, we propose a parallel architecture based on SSCA-protected operations that compute *two* operations simultaneously and is protected by using our highly-secure atomic structure (Chapter 5, Section 5.4). We end with performance analysis of our formulae for both scenarios: SSCA-protected and unprotected.

6.1 Previous Work

Several efforts appeared during the last few years to parallelize ECC point formulae. [AHK+01] and [IT02] introduced efficient parallel point operations targeting SIMD-based processors.

In [IT02], authors presented formulae for 2-processor architectures. However, its parallel doubling is not the most efficient since it only considers the general doubling case (the constant a is not fixed; see formula (2.13)).

[AHK+01] introduced modified Jacobian coordinates (X,Y,Z,Z^2) to develop fast parallel formulae for platforms that can execute *two* and *three* operations simultaneously. Its parallel formulae are, to our knowledge, the fastest.

However, the limitation of the previous works is that they rely on traditional point operation formulae, which are restricted to a fixed number of squarings and multiplications (Chapter 2). The methodology introduced in Chapter 4 will be shown to allow higher flexibility to develop superior parallel operations that are more flexibly adaptable to multiprocessor or parallel execution. In this regard, we propose faster parallel formulae for multiprocessor platforms that are able to execute *three* and *four* operations in parallel.

On the other hand, we have investigated SSCA-protected implementations for parallel architectures. In [FGK+02] and [IT02b], authors presented efficient parallel schemes on generic curves over prime fields using the Montgomery Ladder, which is intrinsically protected against SSCA because every iteration in the main loop involves *one* doubling and *one* addition. An advantage of this method is that formulae involve computation with the *x*-coordinate only. In particular, reference [FGK+02] presented a more attractive scheme since it parallelizes doublings and additions at the field operation level, whereas [IT02b] parallelizes at the point operation level in every iteration of the main loop. The latter has the limitation that the cost of every iteration is determined by the most costly point operation, namely addition.

Later, reference [IT05] improved the previous proposals and introduced a unified Doubling-Addition formula for the Montgomery Ladder method. The composite formula was then efficiently parallelized.

In [Mis06], the authors proposed a pipelined approach for generic curves over prime fields using the standard point arithmetic. In this scheme, each point operation is protected against SSCA using atomicity, and the atomic block execution is done through a pipeline, where up to two atomic blocks can be computed simultaneously. Because a pipelined atomic operation can begin its execution before the previous atomic operation is complete, the total throughput is significantly reduced to a few atomic blocks.

In this work, we propose a faster 2-processor SSCA-protected scheme that introduces further cost reductions by using the innovative atomic structure with squarings: *S*-*N*-*A*-*M*-*N*-*A*-*A* (Chapter 5, Section 5.4). As previously explained, our atomic structure not only offers true protection against SSCA by distinguishing multiplications from squarings, but also allows us to pack more field operations in each block.

6.2 Unprotected Parallel Point Arithmetic

In this case, we will show that our methodology of replacing multiplications (Chapter 4) permits to flexibly modify point doubling, addition and tripling formulae to make them more efficient when implemented in a parallel architecture such as SIMD. In the following, we will present formulae to compute *three* and *four* operations in parallel. It is important to note that in the 4-processor case, *one* multiplication can be replaced by up to *two* squarings since more computing resources are available, and the introduction of squarings would permit to reduce the costs further.

Also, a new coordinate system that takes advantage of the inserted squarings and, thus, minimizes computing time in parallel implementations is introduced: $(X, Y, Z, X^2, Z^2, Z^3 / Z^4)$. The fourth coordinate X^2 will be required for doublings and triplings, and the sixth coordinate will be Z^3 if the current operation is an addition, or Z^4 if the current operation is a doubling or tripling.

6.2.1 Three-processor formulae

Parallel Point Doubling

For the parallel doubling operation using a 3-processor architecture, we use the fast formula (4.4), considering $a = -3$ for efficiency purposes. However, we do not use the factorization technique given in (2.14) with the objective of taking advantage of the new coordinate system which already includes X^2 and Z^4 as pre-computed terms. This will be shown to reduce the overall cost.

The parallel doubling formula is shown in Table 6.1. Only squarings and multiplications are shown. For a detailed description the reader is referred to Appendix D1.

Doubling: $2(X_1, Y_1, Z_1, X_1^2, Z_1^2, Z_1^4) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^3 / Z_3^4)$			
Operation	Processor1	Processor2	Processor3
1. Squaring	α^2	$\left(Y_1 + Z_1\right)^2$	Y_1^2
2. Squaring	Y_1^4	Z_3^2	$\left(X_1 + Y_1^2\right)^2$
3. Multiplication	X_3^2	$\alpha \cdot \left(\beta - X_3\right)$	Z_3^3 / Z_3^4 (a)

(a) Z_3^3 if next operation is a point addition, or Z_3^4 if next operation is a doubling or tripling.

Table 6.1. Three-Processor Point Doubling

The total cost of the parallel doubling is $1M + 2S + 11A$. This cost is reduced by one addition to $1M + 2S + 10A$ if repeated doublings are being computed or the following operation is a tripling. This reduction is possible because the last 3-parallel operations of the doubling can be merged with the first 3-parallel operations of the following doubling or tripling.

Parallel Point Addition

For the parallel addition formula, we consider the efficient case with mixed coordinates given by the fast addition formula (4.8).

The parallel addition formula is shown in Table 6.2, with a cost of $3M + 2S + 8A$. This cost is reduced by one addition to $3M + 2S + 7A$ if a doubling or tripling is computed right after the addition. Similarly to the 3-processor doubling, the last 3-parallel operations of the addition can be merged with the first 3-parallel operations to the following doubling or tripling. Again, only squarings and multiplications are shown in Table 6.2, but a detailed description is presented in Appendix D2.

Addition: $(X_1, Y_1, Z_1, Z_1^2, Z_1^3) + (X_2, Y_2) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^4)$			
Operation	Processor1	Processor2	Processor3
1. Multiplication	$Z_1^3 . Y_2$	$Z_1^2 . X_2$	*
2. Squaring	α^2	$(Z_1 + \beta)^2$	β^2
3. Multiplication	$4X_1 . \beta^2$	$4\beta . \beta^2 = 4\beta^3$	$2\beta . Y_1$
4. Squaring	X_3^2	*	Z_3^2
4. Multiplication	$4Y_1\beta . 2\beta^2 = 8Y_1\beta^3$	$\alpha . (4X_1\beta^2 - X_3)$	Z_3^4

Table 6.2. Three-Processor Point Addition

Parallel Point Tripling

For the parallel tripling operation, we consider formula (4.9) with $a = -3$. The parallel tripling formula is shown in Table 6.3, and similarly to the doubling case, a is defined as sparse with a fixed value of -3 but the factorization technique in (2.14) is not applied to achieve maximal utilization of the introduced coordinate system. A detailed description of this parallel computation can be found in Appendix D3. The total cost of the parallel tripling is $3M + 3S + 14A$.

Tripling: $3(X_1, Y_1, Z_1, X_1^2, Z_1^2, Z_1^4) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^3 / Z_3^4)$			
Operation	Processor1	Processor2	Processor3
1. Squaring	θ^2	Y_1^2	Y_1^2
2. Squaring	Y_1^4	$(X_1 + Y_1^2)^2$	$4Y_1^4$
3. Squaring	$(\theta + \omega)^2$	$(Z_1 + \omega)^2$	ω^2
4. Multiplication	$4X_1 . \omega^2$	$(4Y_1)^2 = 16Y_1^2$	Z_3^2
5. Multiplication	$16Y_1^2 . (2\beta - 2\alpha)$	$C . D = (2\beta - 2\alpha) . (4\beta - 2\alpha)$	$\omega . \omega^2 = \omega^3$
6. Multiplication	X_3^2	$Y_3 = 8Y_1 . (-C . D - \omega^3)$	Z_3^3 / Z_3^4 (a)

(a) Z_3^3 if next operation is a point addition, or Z_3^4 if next operation is a doubling or tripling.

Table 6.3. Three-Processor Point Tripling

6.2.2 Four-processor formulae

Parallel Point Doubling

For the parallel doubling using a 4-processor architecture and, similarly to the case of three processors, we use formula (4.4) with $a = -3$ and avoid factorization technique (2.14). However, this time we can maximize processor utilization by replacing additional multiplications by two squarings. As it was discussed previously, in a sequential architecture this would lead to higher costs. However, we will show that this strategy leads to further computing time reduction in 4-processor architectures since not all processors are being used all the time and extra squarings can be accommodated by inactive processors.

Before proceeding, we have to modify the fast doubling formula (4.4) to make it suitable for more squaring-for-multiplication replacements. The revised formula would be as follows if we apply the strategy given in Chapter 4:

$$X_3 = 4\alpha^2 - 8\beta, \qquad Y_3 = 2\alpha\left(4\beta - X_3\right) - 64Y_1^4, \qquad Z_3 = 2\left[\left(Y_1 + Z_1\right)^2 - Y_1^2 - Z_1^2\right] \tag{6.1}$$

Where: $\alpha = 3X_1^2 + aZ_1^4$

$4\beta = 8\left[\left(X_1 + Y_1^2\right)^2 - X_1^2 - Y_1^4\right]$

$2\alpha\left(4\beta - X_3\right)$ is computed as $\left(\alpha + 4\beta - X_3\right)^2 - \alpha^2 - \left(4\beta - X_3\right)^2$.

We must notice that, although the number of squarings (and the cost for the sequential implementation) has been increased in (6.1), in a 4-processor architecture this leads to higher processor utilization and reduced or nil number of multiplications. The parallel doubling formula is shown in Table 6.4, with a total cost of $3S + 13A$. Only squarings and multiplications are shown, but a complete description is presented in Appendix D4. If the following operation is an addition, then the cost is slightly increased to $1M + 2S + 13A$, because we need to compute Z_3^3 in the third step, Processor 4.

Additionally, the cost is reduced by *two* field additions to $3S + 11A$ if repeated doublings are performed or the following operation is a tripling, because the last two 4-parallel field

operations can be merged with the first two 4-parallel operations of the following doubling or tripling.

Doubling: $2(X_1, Y_1, Z_1, X_1^2, Z_1^2, Z_1^4) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^3 / Z_3^4)$				
Operation	Processor1	Processor2	Processor3	Processor4
1. Squaring	α^2	$\left(Y_1 + Z_1\right)^2$	Y_1^2	*
2. Squaring	*	Z_3^2	$4\left(X_1 + Y_1^2\right)^2$	$4Y_1^4$
3. Squaring/Multiplication	X_3^2	Z_3^3 / Z_3^4 (a)	$\left(\alpha + 4\beta - X_3\right)^2$	$\left(4\beta - X_3\right)^2$

(a) Z_3^3 if next operation is a point addition, or Z_3^4 if next operation is a doubling or tripling.

Table 6.4. Four-Processor Point Doubling

Parallel Point Addition

We use the fast mixed addition given by (4.8). Following the strategy previously applied to the doubling formula, we modify (4.8) as follows:

$$X_3 = 4\alpha^2 - 4\beta^3 - 8X_1\beta^2, \quad Y_3 = 2\alpha\left(4X_1\beta^2 - X_3\right) - 8Y_1\beta^3, \quad Z_3 = \left(Z_1 + \beta\right)^2 - Z_1^2 - \beta^2 \qquad (6.2)$$

Where: $\alpha = Z_1^3 Y_2 - Y_1$

$\beta = Z_1^2 X_2 - X_1$

$2Y_1\beta$ is computed as $\left(Y_1 + \beta\right)^2 - Y_1^2 - \beta^2$.

$2\alpha\left(4X_1\beta^2 - X_3\right)$ is computed as $\left(\alpha + 4X_1\beta^2 - X_3\right)^2 - \alpha^2 - \left(4X_1\beta^2 - X_3\right)^2$.

Again, the number of squarings has been increased, but the extra squarings can be easily computed by inactive processors, and thus, costly multiplications are avoided as much as possible.

The parallel addition formula is presented in Table 6.5, where only squarings and multiplications are shown. For a detailed description the reader is referred to Appendix D5. The total cost of the parallel addition is $2M + 2S + 9A$. If the following operation is a

doubling or tripling, the cost is reduced by *two* additions to $2M + 2S + 7A$.

Addition: $(X_1, Y_1, Z_1, Z_1^2, Z_1^3) + (X_2, Y_2) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^4)$				
Operation	Processor1	Processor2	Processor3	Processor4
1. Multiplication	$Z_1^3 . Y_2$	$Z_1^2 . X_2$	Y_1^2	*
2. Squaring	α^2	$(Z_1 + \beta)^2$	$(Y_1 + \beta)^2$	β^2
3. Multiplication	$2\beta . 2\beta^2 = 4\beta^3$	Z_3^2	$2Y_1\beta . 4\beta^2$	$2X_1 . 2\beta^2 = 4X_1\beta^2$
4. Squaring	X_3^2	Z_3^4	$(4X_1\beta^2 - X_3)^2$	$(\alpha + 4X_1\beta^2 - X_3)^2$

Table 6.5. Four-Processor Mixed Addition

Parallel Point Tripling

In the case of the tripling, this operation can be implemented with the fast formula given by (4.9), fixing $a = -3$. We again follow the strategy applied to doubling and addition and show that in this case the replacement of all multiplications by squarings leads to higher performance.

Formula (4.9) is modified as follows:

$$X_3 = 16Y_1^2(2\beta - 2\alpha) + 4X_1\omega^2, \qquad Y_3 = 4Y_1\mu, \qquad Z_3 = (Z_1 + \omega)^2 - Z_1^2 - \omega^2 \tag{6.3}$$

Where:

$$2\alpha = (\theta + \omega)^2 - \theta^2 - \omega^2$$
$$2\beta = 16Y_1^4$$
$$\theta = 3X_1^2 + aZ_1^4$$
$$\omega = 6\left[(X_1 + Y_1^2)^2 - X_1^2 - Y_1^4\right] - \theta^2$$
$$\mu = 2(2\alpha - 2\beta)(4\beta - 2\alpha) - 2\omega^3$$

The next multiplications are computed as follows:

$$4X_1\omega^2 = 2\left[(X_1 + \omega^2)^2 - X_1^2 - \omega^4\right]$$

$$2\omega^3 = \left(\omega + \omega^2\right)^2 - \omega^2 - \omega^4$$
$$16Y_1^2\left(2\beta - 2\alpha\right) = \left(8Y_1^2 + 2\beta - 2\alpha\right)^2 - 64Y_1^4 - \left(2\beta - 2\alpha\right)^2$$
$$4Y_1\mu = \left(4Y_1 + \mu\right)^2 - 16Y_1^2 - \mu^2$$
$$2\left(2\alpha - 2\beta\right)\left(4\beta - 2\alpha\right) = 4\beta^2 - \left(2\alpha - 2\beta\right)^2 - \left(4\beta - 2\alpha\right)^2$$

The parallel tripling formula has a total cost of 6*S* + 17*A*. Only squarings and multiplications are shown in Table 6.6, but a detailed description can be found in Appendix D6. If the operation following a tripling is an addition, the cost is slightly increased to 1*M* + 5*S* + 17*A*. On the other hand, the cost is reduced by *two* additions to 6*S* + 15*A* if repeated triplings are performed or the following operation is a doubling.

Tripling: $3(X_1, Y_1, Z_1, X_1^2, Z_1^2, Z_1^4) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^3 / Z_3^4)$

Operation	Processor1	Processor2	Processor3	Processor4
1. Squaring	θ^2	Y_1^2	$4Y_1^2$	*
2. Squaring	$4Y_1^4$	$4\left(X_1 + Y_1^2\right)^2$	Y_1^4	$\left(X_1 + Y_1^2\right)^2$
3. Squaring	ω^2	$\left(Z_1 + \omega\right)^2$	$\left(\theta + \omega\right)^2$	$4\beta^2$
4. Squaring	$\left(X_1 + \omega^2\right)^2$	ω^4	$\left(\omega + \omega^2\right)^2$	$\left(2\alpha - 2\beta\right)^2$
5. Squaring	$\left(2\beta - 2\alpha\right)^2$	Z_3^2	$\left(4\beta - 2\alpha\right)^2$	$\left(8Y_1^2 + 2\beta - 2\alpha\right)^2$
6. Squaring/Multiplication	X_3^2	Z_3^3 / Z_3^4 (a)	$\left(4Y_1 + \mu\right)^2$	μ^2

(a) Z_3^3 if next operation is a point addition, or Z_3^4 if next operation is a doubling or tripling.

Table 6.6. Four-Processor Point Tripling

6.2.3 Performance comparison

Table 6.7 summarizes the cost of the parallel point operations presented for the cases where *three* and *four* operations are executed simultaneously on a SIMD-based architecture.

Results are compared to previous proposals given by [AHK+01] and [IT02]. The most

efficient scenario was given by [AHK+01], which developed cheaper 3-processor SIMD doubling and mixed addition operations using the modified coordinate system (X, Y, Z, Z^2). Similarly to our case, they used $a = -3$ for the doubling formula, without applying the factorization technique in (2.14). However, our flexible methodology has allowed further cost reductions by replacing some costly multiplications for squarings. This permits to reduce doublings to $1M + 2S$ in the case of 3-parallel operations, in comparison to the cost of $2M + 1S$ for the doubling given by [AHK+01].

For instance, for an n-bit NAF scalar multiplication, with an approximate cost of $(n-1)$ doublings and $(n/3)$ additions, [AHK+01] would require roughly $3.3nM + 1nS$, whereas our formulae requires $2nM + 2.6nS$. When $1S = 0.8M$, there are no significant differences between both formulae (about 661 field multiplications each, if $n = 160$). However, for the case $1S = 0.6M$, [AHK+01] costs $627M$, and ours only $574M$.

In comparison to costs using traditional sequential formulae (Chapter 2, Table 2.2): $1954M$ (if $1S = 0.8M$) and $1731M$ (if $1S = 0.6M$), we get computing time reductions of about 66% and 67%, respectively. In comparison to our fast sequential formulae presented in Chapter 4 (Table 4.1): $1657M$ (if $1S = 0.8M$) and $1455M$ (if $1S = 0.6M$), we get computing time reductions of about 60% and 61%, respectively.

For the tripling, we have proposed, to our knowledge, the first approach for a parallel implementation. On a 3-processor SIMD scheme, the proposed tripling performs more than *two* times faster. For instance, if $1S = 0.8M$, the traditional tripling costs $13.2M$, and our fast tripling in a sequential fashion, $12.6M$ (see Table 4.2). By contrast, the proposed 3-processor tripling only costs $5.4M$.

Work	Doubling	Mixed Addition	Tripling
This work: 4 processors	$3S$ / $1M + 2S$	$2M + 2S$	$6S$ / $1M + 5S$
This work: 3 processors	$1M + 2S$	$3M + 2S$	$3M + 3S$
[AHK+01] : 3 processors	$2M + 1S$	$4M$	-
[AHK+01] : 2 processors	$2M + 3S$	$5M + 1S$	-
[IT02] : 2 processors	$2M + 3S$	$4M + 2S$	-
Fast sequential (Ch. 4)	$3M + 5S$	$7M + 4S$	$7M + 7S$
Traditional sequential (Ch. 2)	$4M + 6S$	$8M + 3S$	$10M + 4S$

Table 6.7. Comparison of different parallel and sequential point operations

Furthermore, our methodology makes point operations suitable for architectures that compute *four* operations simultaneously. We further reduce our 3-parallel formulae, which are the most efficient to our knowledge, to achieve faster 4-parallel formulae by trading *one* squaring for *one* multiplication in the case of doubling, reducing *one* field multiplication in the case of mixed addition, and trading *three* squarings for *three* multiplications in the case of tripling.

For comparison purposes, if we consider a 160-bit NAF scalar multiplication, our 4-processor formulae would require approximately $584M$ (if $1S = 0.8M$) and $478M$ (if $1S = 0.6M$). That means reductions of about 11% and 17% for each case, respectively, in comparison with the 3-processor implementation. In comparison to the traditional sequential approach (Chapter 2), we obtain reductions of about 70% and 72% for each case, respectively, which mean that the 4-processor scheme is more than *three* times faster than the traditional sequential implementation.

On a 4-processor scheme, the proposed tripling formula performs almost *three* times faster. For instance, if $1S = 0.8M$, the proposed 4-processor tripling costs in most cases $6S \equiv 4.8M$, whereas the traditional tripling costs $13.2M$, and our fast tripling in a sequential fashion, $12.6M$ (see Table 4.2).

6.3 Parallel SSCA-protected Point Arithmetic

Operations presented in the previous section are oriented to achieve the maximum speed-up on SIMD-based implementations when side-channel attacks are not a problem. In the present section, we propose a scheme with 2-processor point operations protected against simple-side-channel attacks. Again, atomicity has been used to achieve the required level of security.

We have investigated dependencies among field operations in each point operation, and concluded that architectures that are designed for executing *two* operations simultaneously are more efficiently exploited if squarings are also considered in the formulae.

In Chapter 5, our highly flexible methodology of replacing multiplications for squarings has already been applied to yield improved formulae that permit the efficient introduction of squarings into the atomic block structure. Thus, our scheme arranges *two* field operations in parallel at each step, following the atomic structure given by *S-N-A-A-M-N-A-A* (introduced in Section 5.4), which has been found to efficiently accommodate all the point operations in 2-processor architectures.

In the following, we call a block a *parallel* atomic block, if it is able to execute two operations in parallel and follows the aforementioned atomic structure to protect against SSCA.

In the next paragraphs, we describe each parallel point operation. For each formula, the order of execution has been carefully arranged to yield the cheapest possible atomic point operations.

As explained in Chapter 5, to achieve minimum costs, we require formulae with a balanced number of field multiplications and squarings. For the case of doubling, the traditional formula given by (2.13), with $a = -3$, is already balanced with a cost of $4M + 4S$. Thus, we only require 2 *parallel* atomic blocks, as each of these is capable of executing two field multiplications and two field squarings. The cost of the 2-parallel atomic doubling is

fixed in $2M + 2S + 8A$. Details of this operation are shown in Appendix E1.

For the case of mixed addition, in Section 6.1, Chapter 6, we derived the balanced formula (5.2) with a cost of $6M + 6S$. We use this formula to obtain our 2-parallel atomic addition with only 4 *parallel* atomic blocks. Thus, the total cost of the 2-parallel addition is given by $4M + 4S + 16A$. Details of this formula are shown in Appendix E2.

Similarly, in Section 4.1.2, Chapter 4, we presented a balanced tripling formula given by (4.10) with a cost of $7M + 7S$. Using this formula, we derive a 2-parallel tripling protected against SSCA with 5 *parallel* atomic blocks and a cost of $5M + 5S + 20A$. Details of the 2-parallel protected tripling are shown in Appendix E3.

In Table 6.8, we show a sample execution of consecutive tripling, doubling and addition operations in our proposed 2-parallel SSCA-protected scheme, where one point operation is executed in parallel at a time. We denote the *parallel* atomic block x executed by processor y by $\Delta x,y$. Thus, *five*, *two* and *four parallel* atomic blocks are required to complete a tripling, a doubling and an addition, respectively.

Time	Processor1	Processor2
t	⋮	⋮
t + 1	$\Delta 1,1^{(c)}$	$\Delta 1,2^{(c)}$
t + 2	$\Delta 2,1^{(c)}$	$\Delta 2,2^{(c)}$
t + 3	$\Delta 3,1^{(c)}$	$\Delta 3,2^{(c)}$
t + 4	$\Delta 4,1^{(c)}$	$\Delta 4,2^{(c)}$
t + 5	$\Delta 5,1^{(c)}$	$\Delta 5,2^{(c)}$
t + 6	$\Delta 1,1^{(b)}$	$\Delta 1,2^{(b)}$
t + 7	$\Delta 2,1^{(b)}$	$\Delta 2,2^{(b)}$
t + 8	$\Delta 1,1^{(a)}$	$\Delta 1,2^{(a)}$
t + 9	$\Delta 2,1^{(a)}$	$\Delta 2,2^{(a)}$
t + 10	$\Delta 3,1^{(a)}$	$\Delta 3,2^{(a)}$
t + 11	$\Delta 4,1^{(a)}$	$\Delta 4,2^{(a)}$
t + 12	⋮	⋮

(a) Atomic point addition,
(b) Atomic point doubling,
(c) Atomic point tripling.

Table 6.8. Parallel execution of ECC point operations in the proposed 2-parallel SSCA-protected scheme

6.3.1 Performance comparison

Several efficient efforts to protect parallel implementations against SSCA can be found in the literature.

In [FGK+02] and [IT02b], authors presented efficient parallel SSCA-protected schemes using the Montgomery Ladder method over prime fields. In general, for an n-bit scalar multiplication dP, the Montgomery ladder method requires $(n - 1)$ iterations. [FGK+02] presented a parallel doubling and addition execution with a cost of *ten* field multiplications. Thus, the scalar multiplication would cost $10(n - 1)M$. If $n = 160$ bits, then the total cost is $1590M$, with $9.9M$ per bit.

On the other hand, the method given by [IT02b] fixes the cost of every iteration to one

point addition. Doubling and addition formulae in [IT02b] cost $6M + 3S$ and $8M + 2S$, respectively. Since one extra doubling is required to complete the scalar multiplication, the cost of the scalar multiplication using the Montgomery Ladder would be one doubling and $(n - 1)$ additions, which is equivalent to $(6M + 3S) + (n - 1)(8M + 2S)$. If we consider $1S = 0.8M$, the total cost is fixed in $1535M$, with $9.6M$ per bit.

Later, [IT05] improved proposals by [FGK$^+$02] and [IT02b] and proposed a unified Doubling-Addition formula for the Montgomery Ladder. The composite formula was efficiently parallelized with a cost of $7M + 2S$. Thus, a scalar multiplication would cost $(n - 1)(7M + 2S) = 1367M$, with $8.5M$ per bit.

In [Mis06], the authors proposed a pipelined approach for generic curves over prime fields using the standard point arithmetic. The pipeline scheme was protected against SSCA using atomicity. Because a pipelined atomic operation can begin its execution before the previous atomic operation is complete, the total throughput is reduced to only *six* atomic blocks. In this work, each atomic block had the traditional *M-A-N-A* structure. Since the cost using the NAF method is $(n - 1)$ doublings and $(n/3)$ additions (according to 2.20), the pipelined method costs $10 + 6(n + n/3 - 2)$ atomic blocks [Mis06], which is equivalent to 1278 atomic blocks when $n = 160$. If we consider each atomic block equivalent to $1.1M$ (each atomic block contains one multiplication and two additions, where $1A = 0.05M$ [Ber]), then the cost using the NAF method is fixed in $1406M$, with $8.8M$ per bit.

In contrast, our proposed parallel SSCA-protected scheme introduces further cost reductions by including squarings in the atomic structure, and hence, efficiently reduces the number of required atomic blocks. As it was shown in previous sections, our atomic structure not only offers true protection against SSCA by distinguishing multiplications from squarings, but also allows us to pack more field operations per block. Furthermore, our parallel approach is showing to be superior to the pipelined one in [Mis06]. The pipeline reduces the throughput to *six* atomic blocks by beginning each point operation as soon as possible, with a maximum of two processes being computed simultaneously. Hence, the main obstacle to achieve cheaper execution is given by inter-operation dependencies

(dependencies found between consecutive point operations). By contrast, our approach parallelizes field operations inside each point operation, and hence, cost of the atomic formulae is mainly defined by intra-operation dependencies (dependencies inside each point operation). We have carefully analyzed both kinds of dependency, and concluded that inter-operation dependencies in the point arithmetic over prime fields are more restrictive than intra-operation dependencies. Thus, even though throughput is effectively reduced to six atomic blocks (about *six* field multiplications) in [Mis06], with the parallel approach we have doublings and additions that are executed with only $2M + 2S$ and $4M + 4S$, respectively. The reader must note that doublings are more frequently required in efficient scalar multiplication methods, and hence, our method would be superior even in the case $1S = 1M$, making our approach not only more secure but also faster in hardware implementations where a hardware multiplier executes both multiplications and squarings.

The next comparison holds for implementations where squarings are cheaper than multiplications, as is generally the case in software platforms. In a 160-bit NAF scalar multiplication, our scheme costs $2(n - 1)$ atomic blocks and $4(n/3)$ atomic blocks, since doublings and additions require *two* and *four parallel* atomic blocks, respectively. Consequently, our parallel approach requires 531 *parallel* atomic blocks. If we consider the cost of each *parallel* atomic block to be $2M$ (each *parallel* atomic block costs one multiplication, one squaring and four additions, where $1S = 0.8M$ and $1A = 0.05M$ [Ber]), then the total cost using the NAF method would be $1062M$, with $6.6M$ per bit. Thus, our approach introduces computing time reductions of approximately 33%, 31% and 22% in comparison with the Montgomery Ladder methods proposed by [FGK+02], [IT02b] and [IT05], respectively. The reader must note that field additions have not been included in the cost estimates for the previous Montgomery Ladder methods. Consequently, cost reductions of our approach in comparison with those methods are even greater than estimated values.

In comparison with the pipelined scheme presented by [Mis06], our parallel approach introduces a significant reduction of approximately 24%.

Work	Cost
Proposed 2-parallel scheme (NAF method)	1062*M*
Pipelined [Mis06] (NAF method)	1406*M*
[IT05] (Montgomery ladder) [(a)]	1367*M*
[IT02b] (Montgomery ladder) [(a)]	1535*M*
[FGK$^+$02] (Montgomery ladder) [(a)]	1590*M*
S-*N*-*A*-*M*-*N*-*A*-*A*-based sequential (Table 5.4, Ch. 5, NAF method)	1860*M*
Traditional sequential (Table 5.4, Ch. 5, NAF method)	2390*M*

(a) Field additions are not included in the cost.

Table 6.9. Comparison of performance of parallel SSCA-protected methods with $n = 160$ ($1S = 0.8M$ and $1A = 0.05M$)

Table 6.9 compares performance of previous parallel SSCA-protected methods with the proposed 2-parallel SSCA-protected scheme. For comparison purposes, sequential atomic methods are also presented. Our parallel approach introduces a speed-up of about 56% and 43% in comparison to the traditional and new *S*-*N*-*A*-*M*-*N*-*A*-*A*-based (Chapter 5, Section 5.4) atomic implementations, respectively.

Again, the speed-up of our method would be even more significant if $1S = 0.6M$.

Chapter 7

New Multibase Scalar Multiplication

In Chapter 3, we introduced new formulae for composite operations such as quintupling, septupling and others. We also showed how these operations can be cleverly exploited to speed-up ECC computations.

In the present chapter, new scalar multiplication methods are developed on top of those new operations by allowing an extended set of bases. Our methodology works with a new multibase NAF-like representation for the scalar d. Moreover, we propose efficient algorithms that convert any positive integer to such multibase representation and define their theoretical performance in terms of length and Hamming weight.

In the remainder of this chapter, we first discuss previous efforts to accelerate the scalar multiplication by using more than one radix in the scalar expansion. Then, *three* new multibase representations are introduced with their corresponding scalar multiplication algorithms. We end the chapter with extensive tests that follow the theoretical estimates and confirm the superiority of our scalar multiplications in comparison to any previous method in the current literature.

7.1 Previous Work

Since [DIM05] introduced the idea of using mixed powers of two and three to shorten the scalar expansion for the ECC scalar multiplication, many works investigating and expanding this idea have appeared during recent years. The specific representation using bases two and three as presented by [DIM05], and its scalar multiplication, are referred in this work as DB and DB scalar multiplication, respectively (see Chapter 2, Section 2.3.3).

One notable area where these ideas are producing interesting results is in ECC scalar multiplication on Koblitz curves [AS06,ADD+06], where new scalar multiplication methods are sublinear with a complexity of less than $\left(\frac{\log d}{\log\log d}\right)$ additions.

On standard ECC curves over prime and binary fields, the use of double bases has been already shown to reduce the computing cost of established methods as NAF and *w*NAF [DIM05,DI06]. However, DB has some shortcomings in such scenario.

First, conversion of any scalar d to DB can be relatively slow since it is based on an exhaustive search of all possible combinations of powers of two and three (see "Greedy" Algorithm 2.4). Some speed-up can be injected 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 for 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 pre-computed data, then some delays could be potentially introduced during DB conversions.

Second, the efficiency of DB scalar multiplication heavily depends on the maximum bounds $b_{\max}$, $c_{\max}$ (see Algorithm 2.4). Heuristic approximations have been proposed to estimate the values that would yield short expansions [DIM05]. However, in ECC scalar multiplication, the density of non-zero terms (i.e., the number of additions) is not the only parameter to take into account to achieve cheaper computations. In fact, since an additional point operation (namely tripling) is included in the expansion, a balanced number of doublings, triplings and additions should be considered to reach an optimal 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 to find optimal values for $b_{\max}$, $c_{\max}$ in the DB scalar multiplication.

In this work, we expand the idea of using only two bases to any efficient number of bases, and propose a multibase NAF-like representation that overcomes all the previous problems found in DB. This is, to our knowledge, the first effort in the area to apply a multibase representation of the scalar d to reduce ECC computing costs. Also, we show that our methods are sublinear with complexity of approximately $\left(\frac{\log d}{\log\log d}\right)$ point additions.

As it has been through all this work, we focus on applying proposed methods to ECC over prime fields using standard curves (2.2). However, it is important to note that proposed representations are generic and can be applied to other areas, or specifically, to any cryptosystem based on exponentiation or scalar multiplication. For the latter, the only requisite is to have efficient formulae to compute operations with radices > 2.

7.2 New Multibase Scalar Multiplication Methods

In the following, we introduce our multibase scalar multiplications based on new NAF-like representations that use an extended number of bases.

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

We propose the next signed multibase representation for the scalar d to construct the scalar multiplication:

$$d = \sum_{i=1}^{m} d_i \prod_{j=1}^{J} a_j^{c_{i(j)}} \tag{7.1}$$

Where: bases $a_1 \neq a_2 \neq \ldots \neq a_J$ are positive prime integers.

m is the length of the expansion.

d_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 (7.1) is efficiently executed by a scalar multiplication scanning the digits from left-to-right.

It is important to note that there is no restriction in the selection or number of bases. These parameters are determined according to a specific application. In our case, for ECC on standard curves over prime fields, we will show that efficiently for most of the cases: $J \leq 4$, with $a_1 = 2$, $a_2 = 3$, $a_3 = 5$, $a_4 = 7$.

Notice that signed multibase representation as presented in (7.1) is not unique. In fact, the "Greedy" algorithm by [DIM05] gives an expansion for the scalar d with similar conditions, although in their case they restrict the number of bases to 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 in all cases a canonical representation (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 7.1 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 (mbNAF) of a positive integer d, denoted by $\left(d_m^{(a_m)}, d_{m-1}^{(a_{m-1})}, \ldots, d_2^{(a_2)}, d_1^{(a_1)}\right)$, where m is the length of the expansion, $d_i^{(a_i)}$ 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 d has a unique *mb*NAF representation for a given set of bases $\mathcal{A}$.
- No consecutive digits are non-zero.
- $d_i \in D_i = \left\{0, \pm 1, \pm 2, \ldots, \pm \left\lfloor \frac{a_1^2 - 1}{2} \right\rfloor \right\} \setminus \left\{ \pm 1 a_1, \pm 2 a_1, \ldots, \pm \left\lfloor \frac{a_1 - 1}{2} \right\rfloor a_1 \right\}$, for $1 \leq i \leq m$.
- The leftmost non-zero digit is positive, i.e., $d_m > 0$.

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 pre-computed digits D_i works solely on base a_1, called the main base, to guarantee a minimal number of pre-computations.

We propose Algorithm 7.1 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 7.1 Computing the *mb*NAF of a positive integer

INPUT: scalar d, 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$(d) = (..., d_2^{(a_2)}, d_1^{(a_1)})$

1. $i = 0$
2. While $d > 0$ do
   2.1. If $d \bmod a_1 = 0$ or $d \bmod a_2 = 0$ or … or $d \bmod a_j = 0$, then $d_i = 0$
   2.2. Else:
      2.2.1 $d_i = d \bmods a_1^2$
      2.2.2 $d = d - d_i$
   2.3
      2.3.1 If $d \bmod a_1 = 0$, then $d = d/a_1$, $d_i = d_i^{(a_1)}$
      2.3.2 elseif $d \bmod a_2 = 0$, then $d = d/a_2$, $d_i = d_i^{(a_2)}$
      ⋮
      2.3.J elseif $d \bmod a_J = 0$, then $d = d/a_J$, $d_i = d_i^{(a_J)}$
   2.4 $i = i + 1$
3. Return $(..., d_2^{(a_2)}, d_1^{(a_1)})$

Function *mods* in Algorithm 7.1 represents the next computation:

$$\begin{cases} \text{If } d \bmod a_1^2 \ge a_1^2/2, \text{ then:} \\ \quad d_i = (d \bmod a_1^2) - a_1^2 \\ \text{Else,} \\ \quad d_i = d \bmod a_1^2 \end{cases} \tag{7.2}$$

Algorithm 7.1 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 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 pre-computed points. However, the non-zero density of the expansion is further reduced in our case as the number of bases is increased since the

algorithm searches for extra divisions by the remainder bases. Consequently, the non-zero density decreases with the number and size of the bases.

[TYW04] showed that the non-zero density for a radix-r using rNAF is given by (2.24). Consequently, our approach's density is asymptotically given by:

$$D = \frac{a_1 - 1}{2a_1 - 1}, \tag{7.3}$$

with regards to the length of the base a_1, and requires $\frac{(a_1 - 2)(a_1 + 1)}{2}$ pre-computed points without considering $\{0, \pm 1\}$. The reader is referred to [TYW04] for a mathematical proof.

It is important to note that if $a_1 = 2$, no extra pre-computed points are required in comparison with the binary or binary NAF representation, where the point at infinity and P should be stored.

Also, $a_1 = 2$ is expected to yield the most efficient scalar multiplication in terms of speed on EC-based standard curves, where point doubling is highly efficient in comparison with other operations (see Table 2.2). However, in new EC-based cryptosystems of characteristic 3 or pairing-based cryptosystems where triplings (or other composite operations) are highly efficient, it is expected to achieve better results with $a_1 \neq 2$.

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

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

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

Definition 7.2 Given a set of bases $\mathcal{A} = \{a_1, a_2, \ldots, a_J\}$, where $a_j \in \mathbb{Z}^+$ are positive primes for $1 \leq j \leq J$, the window-*w* multibase non-adjacent form (*wmb*NAF) of a positive integer d

using window of width $w \geq 2$, denoted by $\left(d_m^{(a_m)}, d_{m-1}^{(a_{m-1})}, \ldots, d_2^{(a_2)}, d_1^{(a_1)}\right)$, where m is the length of the expansion, $d_i^{(a_i)}$ 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 d has a unique *wmb*NAF representation for a given set of bases $\mathcal{A}$ and window w.
- w adjacent digits contain at most one non-zero digit.
- $d_i \in D_i = \left\{0, \pm 1, \pm 2, \ldots, \pm\left\lfloor \frac{a_1^w - 1}{2} \right\rfloor\right\} \setminus \left\{\pm 1 a_1, \pm 2 a_1, \ldots, \pm\left\lfloor \frac{a_1^{w-1} - 1}{2} \right\rfloor a_1\right\}$, for $1 \leq i \leq m$.
- The leftmost non-zero digit is positive, i.e., $d_m > 0$.

Notice that for the set of bases $\mathcal{A} = \{2\}$, the previous definition is identical to the traditional binary *w*NAF (see Chapter 2, Section 2.2.3).

Similarly to *mb*NAF, the set of pre-computed digits D_i are derived from the main base a_1, limiting the required number of pre-computations.

We propose Algorithm 7.2 to efficiently convert the scalar d to *wmb*NAF representation. Again, note that the proposed algorithm is a generalization of the traditional *w*NAF to multibase digit representations.

Algorithm 7.2 Computing the *wmb*NAF of a positive integer

INPUT: scalar d, 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 $w > 2$, where $w \in \mathbb{Z}^+$

OUTPUT: the $(a_1, a_2, ..., a_J)\text{NAF}_w(d) = (..., d_2^{(a_2)}, d_1^{(a_1)})$

1. $i = 0$
2. While $d > 0$ do
   2.1. If $d \bmod a_1 = 0$ or $d \bmod a_2 = 0$ or … or $d \bmod a_j = 0$, then $d_i = 0$
   2.2. Else:
      2.2.1 $d_i = d \bmods a_1^w$
      2.2.2 $d = d - d_i$
   2.3
      2.3.1 If $d \bmod a_1 = 0$, then $d = d/a_1$, $d_i = d_i^{(a_1)}$
      2.3.2 elseif $d \bmod a_2 = 0$, then $d = d/a_2$, $d_i = d_i^{(a_2)}$
      ⋮
      2.3.J elseif $d \bmod a_J = 0$, then $d = d/a_J$, $d_i = d_i^{(a_J)}$
   2.4 $i = i + 1$
3. Return $(..., d_2^{(a_2)}, d_1^{(a_1)})$

Function *mods* in Algorithm 7.2 is a generalization of function (7.2) to any window w:

$$\begin{cases} \text{If } d \bmod a_l^w \ge a_l^w/2, \text{ then:} \\ \quad d_i = (d \bmod a_l^w) - a_l^w \\ \text{Else,} \\ \quad d_i = d \bmod a_l^w \end{cases} \tag{7.4}$$

As we can see, similarly to relation between *w*NAF and NAF, *wmb*NAF is equivalent to *mb*NAF if we fix $w = 2$. Thus, Algorithm 7.2 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. As happens with *mb*NAF, the non-zero density of the expansion closely follows that of *wr*NAF [TYW04], which is the windowed version of *r*NAF (see Section 2.3.3, Chapter 2). Again, our method's density is further

reduced as the number of bases is increased since the algorithm searches for extra divisions by the remainder bases. Consequently, the non-zero density also decreases with the number and size of the bases.

[TYW04] showed that the non-zero density for *wr*NAF is given by (2.25). Consequently, our approach's density is asymptotically given by the following:

$$D = \frac{a_1 - 1}{w(a_1 - 1) + 1}, \tag{7.5}$$

with regards to the length of the base a_1, and requires $\frac{a_1^w - a_1^{w-1} - 2}{2}$ pre-computed points without considering $\{0, \pm 1\}$. For further details, the reader is referred to [TYW04].

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

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

Definition 7.3 Given a set of bases $\mathcal{A} = \{a_1, a_2, \ldots, a_J\}$, where $a_j \in \mathbb{Z}^+$ are positive primes for $1 \geq j \geq J$, the extended multibase non-adjacent form (*extended mb*NAF) of a positive integer d 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 $\left(d_m^{(\mathrm{a}_m)}, d_{m-1}^{(\mathrm{a}_{m-1})}, \ldots, d_2^{(\mathrm{a}_2)}, d_1^{(\mathrm{a}_1)}\right)$, where m is the length of the expansion, $d_i^{(\mathrm{a}_i)}$ is the i^{th} digit and the superscript $\mathrm{a}_i \in A$ denotes the base associated to the respective digit. The *extended mb*NAF has the following properties:

- Every positive integer d has a unique *extended mb*NAF representation for a given set of bases $\mathcal{A}$ and windows $\mathcal{W}$.

- There is at most one non-zero digit among $(w_1 + w_2 + \ldots + w_J)$ adjacent digits.
- $d_i \in D_i = \left\{0, \pm 1, \pm 2, \ldots, \pm \left\lfloor \frac{a-1}{2} \right\rfloor \right\} \setminus \ldots$
  $\ldots \left\{ \pm 1a_1, \pm 2a_1, \ldots, \pm \left\lfloor \frac{a-1}{2a_1} \right\rfloor a_1, \pm 1a_2, \pm 2a_2, \ldots, \pm \left\lfloor \frac{a-1}{2a_2} \right\rfloor a_2, \ldots, \pm 1a_J, \pm 2a_J, \ldots, \pm \left\lfloor \frac{a-1}{2a_J} \right\rfloor a_J \right\}$,
  for $1 \le i \le m$.
- The leftmost non-zero digit is positive, i.e., $d_m > 0$.

We propose Algorithm 7.3 to efficiently convert the scalar d to *extended wmb*NAF representation.

Algorithm 7.3 Computing the *extended wmb*NAF of a positive integer

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

OUTPUT: the $(a_1, a_2, ..., a_J)\text{NAF}_{(w_1, w_2, \ldots, w_J)}(d) = (..., d_2^{(a_2)}, d_1^{(a_1)})$

1. $i = 0$
2. While $d > 0$ do
   2.1. If $d \bmod a_1 = 0$ or $d \bmod a_2 = 0$ or … or $d \bmod a_j = 0$, then $d_i = 0$
   2.2. Else:
      2.2.1 $d_i = d \bmods a$
      2.2.2 $d = d - d_i$
   2.3
      2.3.1 If $d \bmod a_1 = 0$, then $d = d/a_1$, $d_i = d_i^{(a_1)}$
      2.3.2 elseif $d \bmod a_2 = 0$, then $d = d/a_2$, $d_i = d_i^{(a_2)}$
      ⋮
      2.3.J elseif $d \bmod a_J = 0$, then $d = d/a_J$, $d_i = d_i^{(a_J)}$
   2.4 $i = i + 1$
3. Return $(..., d_2^{(a_2)}, d_1^{(a_1)})$

Function *mods* in Algorithm 7.3 involves the next computation:

$$\begin{cases} \text{If } d \bmod a \geq a/2, \text{ then:} \\ \quad d_i = (d \bmod a) - a \\ \text{Else,} \\ \quad d_i = d \bmod a \end{cases} \tag{7.6}$$

Algorithm 7.3 guarantees that there is at most one non-zero digit among ($w_1 + w_2 + \ldots + w_J$) successive digits.

In contrast to Algorithm 7.2, this algorithm establishes windows for different bases instead of limiting the window to only base a_1. Thus, every computation is approximated to a global value a, which we refer to as global base, guarantying 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 solely the main base, i.e., $\mathcal{W}=\{w_1\}$, where $w_1 > 2$.

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 determining 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, in comparison to window w_1 for base 2.

*Extended wmb*NAF requires the next set of pre-computed points:

$$d_i \in \left\{0, \pm 1, \pm 2, \ldots, \pm \left\lfloor \frac{a-1}{2} \right\rfloor\right\} \setminus \ldots$$

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

The previous table requires $\left\lfloor \frac{a}{2} - 1 \right\rfloor$ points for the first term without including $\{0,1\}$. Then,

we have to discard multiples of each base in the respective range, as indicated by the second term of the expression. We have $\left\lfloor \frac{a}{2a_j} \right\rfloor$ points for each j from 0 to J. However, some multiples are shared between different bases. If we fix the number of bases to three, we obtain an expression as the following to compute the number of required pre computed points:

$$\left\lfloor \frac{a}{2} - 1 \right\rfloor - \left\lfloor \frac{a}{2a_1} \right\rfloor - \left\lfloor \frac{a}{2a_2} \right\rfloor - \left\lfloor \frac{a}{2a_3} \right\rfloor + \left\lfloor \frac{a}{2a_1a_2} \right\rfloor - (n-2)\left\lfloor \frac{a}{2a_1a_2a_3} \right\rfloor \tag{7.7}$$

Notice that expression (7.7) 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 (7.7) if $w_j = 0$.

Algorithm 7.3 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 a. In fact, the number of additions $\#(\mathrm{A})$ in the expansion is given by:

$$\#(\mathrm{A}) = D_{a_j} \times \#\left(O_{a_j}\right), \tag{7.8}$$

for any base a_j found in the global base a (i.e., $w_j \neq 0$ 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.

By expanding previous cases to this setting with several bases, density D_{a_j} satisfies the following expression:

$$D_{a_j} = \frac{a_j - 1}{w_j\left(a_j - 1\right) + 1} \tag{7.9}$$

This is also similar to *r*NAF [TYW04]. However, our approach gains further reduction in terms of density due to the use of several bases with flexible windows that introduce successive point doublings, triplings or others before the next addition.

Multibase Scalar Multiplication

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

Algorithm 7.4 *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 $d = (d_{l-1}^{(a_{l-1})},\ldots,d_2^{(a_2)},d_1^{(a_1)})$ from Alg. 7.1, 7.2 or 7.3, $P \in E(\mathbb{F}_p)$

OUTPUT: dP

1. Compute $P_i = iP$ for $i \in D_i$ (see Definitions 7.1, 7.2 or 7.3)
2. $Q = O$
3. For $i = l-1$ downto 0 do
   3.1. $Q = (a_i)Q$
   3.2. If $d_i^{(a_i)} \neq 0$, then:
      If $d_i^{(a_i)} > 0$, then $Q = Q + P_{d_i^{(a_i)}}$
      Else $Q = Q - P_{d_i^{(a_i)}}$
4. Return (Q)

In Algorithm 7.4, $d_i^{(a_i)}$ is the i^{th} digit and the superscript $a_i \in A$ (set of bases) denotes the base associated to the respective digit. Thus, operation $Q = (a_i)Q$ in 3.1 is any point operation with exception of addition (i.e., doubling, tripling, quintupling, and so on). In 3.2, point addition is performed when a non-zero element is found in the multibase expansion. The operation is performed with a pre-computed point P_i according to Definitions 7.1, 7.2 and 7.3, 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 in comparison with previous efforts.

7.3 Testing Results

In this section, we compare performance of our new multibase scalar multiplications with other relevant methods. On top of that, we demonstrate the further reduction in terms of computing costs that can be achieved by using the efficient composite operations introduced in Chapter 3 and the new fast point formulae presented in Chapter 4.

Performance is evaluated for two possible scenarios: unprotected implementations when SSCA is not a concern, and SSCA-protected implementations using side-channel atomicity as developed in Chapter 5.

In the following, we detail our methodology to compare the various scalar multiplication methods.

7.3.1 Methodology

The focus of this section is to determine the fastest algorithms to compute scalar multiplication in two cases: when pre-computations are and are not allowed; and for the two aforementioned scenarios with regards to protection against SSCA.

As it has been common practice through this work, we consider that computing costs of point and scalar arithmetic are independent of the field arithmetic to be used. In general, the previous assumption is valid since different optimizations in the finite field can be generally applied to upper arithmetic layers.

Under that assumption, the cost of a given scalar multiplication method only depends on the number of point operations (and their corresponding cost in terms of field operations), which allows us to greatly simplify comparisons.

Thus, we first determine the expansions for a random number d_i using the various methods to be compared, and then count the number of point operations that would be

required to execute the scalar multiplication d_iP in each case. After counting operations for thousands of random numbers, we average the results for each method and compare one another. To simplify comparisons, costs are ultimately expressed in terms of field multiplications.

To estimate costs in terms of field multiplications in the case of unprotected implementations, the following general assumptions hold:

- $1S = 0.8M$.
- Cost of field addition and subtraction are negligible in comparison to field multiplication and squaring, and they are not counted in the cost estimate.
- Cost of field divisions and multiplications by small constants are comparable to the cost of field addition, and consequently, they are not included in the cost estimate.

To estimate costs in terms of field multiplications for the case of SSCA-protected implementations, the following general assumptions hold:

- There are two possible cases: $1S = 0.8M$ or $1S = 1M$. The latter corresponds to implementations where a hardware multiplier executes both multiplication and squaring.
- $1A = 0.05M$.

Notice for the last assumption that, although cost of field additions is very low in comparison with other operations, our methodology using atomicity significantly reduces the required number of such operations. If one considers a scalar multiplication where thousands of those addition operations are required, then it is relevant to include them in the operation counting.

7.3.2 Comparison

All algorithms have been implemented in Matlab 7r14 using the extended symbolic math

toolbox to take advantage of the Maple kernel for computation with large variable-precision numbers. Algorithms have been run for 10000 different random numbers with maximum bitlength $n = 160$ bits.

The next algorithms have been implemented:

- Algorithm 2.2 for the case $w = 2$ and $w > 2$, corresponding to NAF and wNAF, respectively.
- Algorithm 2.4 corresponding to the "Greedy" algorithm for conversion to DB.
- Algorithms 7.1, 7.2 and 7.3, corresponding to *mb*NAF, *wmb*NAF and *extended wmb*NAF, respectively.

In the following, we discuss cost performance of implemented algorithms, first in the unprotected case and then in the SSCA-protected scenario.

Unprotected Implementations

We have investigated performance of our multibase scalar multiplication methods for the next 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 also consider window $w \in \{0,1\}$.

Table 7.1 details the number of point operations required by introduced multibase methods in comparison with NAF.

Method	Point operation cost
NAF	158.67D + 52.77A
(2,3)NAF	113.50D + 28.41T + 37.67A
(2,3,5)NAF	96.69D + 24.30T + 10.07Q + 31.98A
(2,3,5,7)NAF	86.80D + 21.90T + 9.05Q + 5.71S + 28.68A
(2,3,5,7,11)NAF	81.07D + 20.39T + 8.45Q + 5.40S + 3.00E + 26.77A
(2,3,5,7,11,13)NAF	76.58D + 19.24T + 8.05Q + 5.16S + 2.83E + 2.31TH + 25.23A

Table 7.1. Computing costs in terms of point operations for *mb*NAF scalar multiplications in comparison with NAF

It can be seen in Table 7.1 that non-zero density (i.e., number of point additions) strictly decreases with the number of bases, highlighting the potential reduction introduced by multibase methods. In fact, we can observe the sublinear nature of multibase expansions, which exhibits densities even below $\left(\frac{\log d}{\log\log d}\right)$ additions in some cases (satisfying Theorem 2.2.1, Chapter 2), in contrast to linearity in ($\log d$) observed for NAF. Also, it is interesting to observe that multibase expansions satisfy the theoretical bound given by (7.3) with regards to density.

Nevertheless, the decrement in density rapidly slows down for expansions using high number of bases. Ultimately, in ECC scalar multiplications, efficiency of the composite operation for a specific base will "decide" if it is beneficial to include it into the expansion of the scalar.

To evaluate costs of studied multibase expansions, we followed the next considerations.

For point addition, we consider the efficient case of mixed addition with affine-Jacobian coordinates (2.16). In the case of point operations of the form dP ($d \geq 2$), cost estimates are only considered for the special case when parameter a in equation (2.2) is fixed to -3. The general case easily follows, and should show slightly worse performance for all cases.

Three cases are considered for standard curves over prime fields (2.2):

- Using traditional operations ("Traditional" case). Costs for point doubling, addition and tripling are taken from Table 2.2. Since there are no previous proposals for quintupling and septupling, they are built by using elementary point operations as described in Section 3.3.3. Costs for those operations are taken from Table 3.2 (previous work).

- Using fast point operations as detailed in Tables 4.1 and 4.2 ("Fast" case). They reflect improved formulae using our methodology of replacing multiplications for squarings in the case of doubling and tripling (Chapter 4). In the case of higher order operations, those tables reflect improvements achieved by applying the previous methodology to new composite operations introduced in Chapter 3.

- Using new composite operation DA ("Fast using DA" case) introduced in Section 3.2.1, and following approach given in Section 3.4.2. In this case, the only difference with the "Fast" case is that every doubling followed by an addition is replaced by a DA operation to achieve further cost reductions.

Figures 7.1 and 7.2 shows the way different non-windowed and windowed scalar multiplication methods, respectively, are improved with the previous techniques.

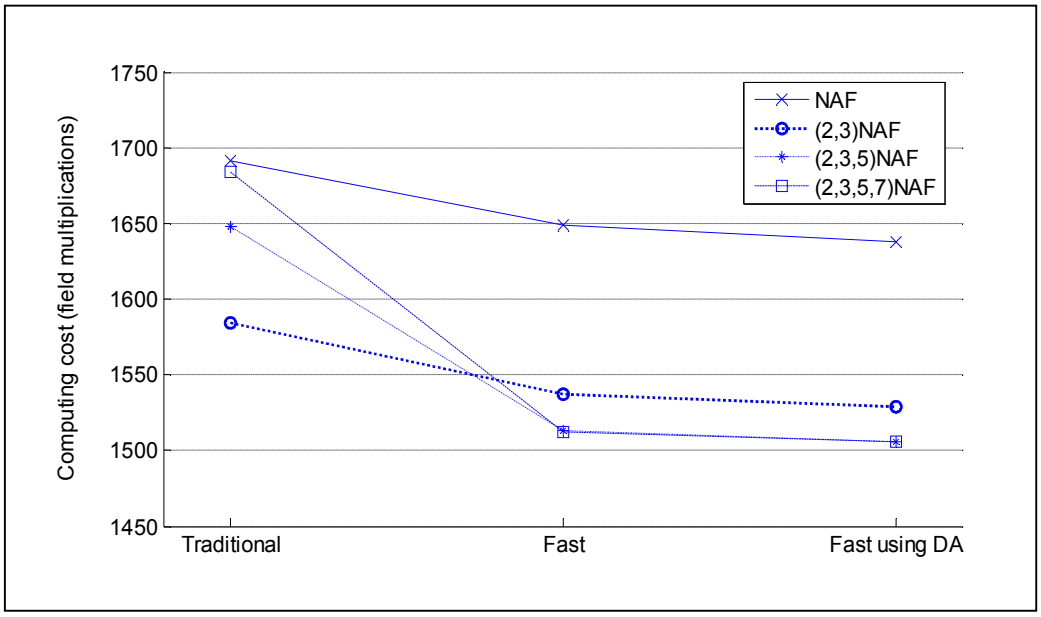


Figure 7.1. Improvement of performance of scalar multiplications using proposed techniques

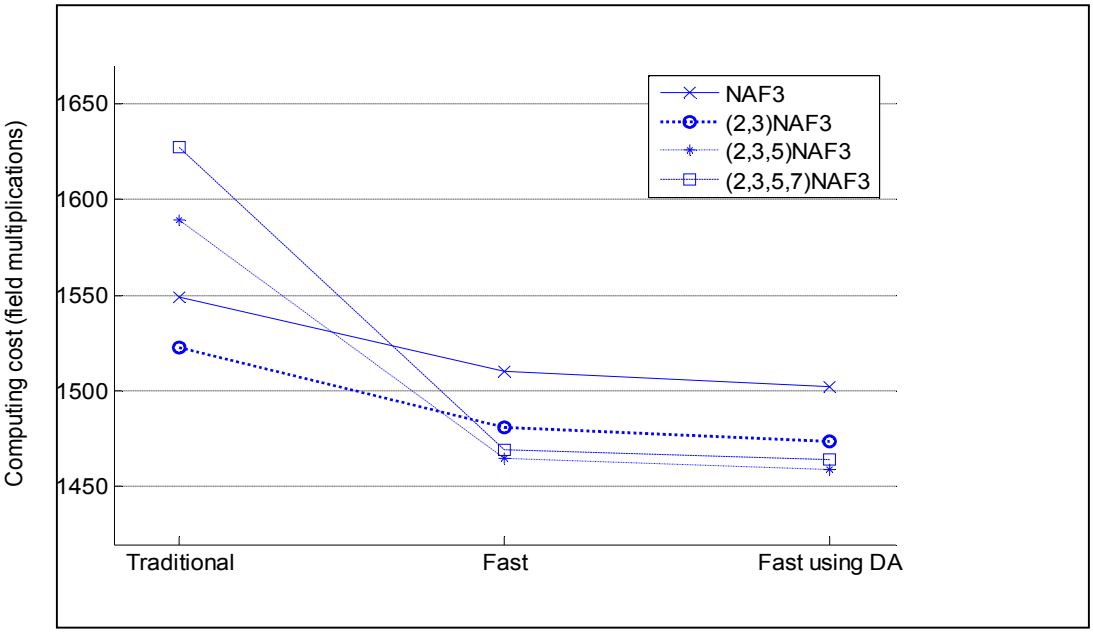


Figure 7.2. Improvement of performance of window-based scalar multiplications ($w = 3$) using new techniques

We can see in Tables 7.2 and 7.3 that fast formulae from Chapter 4 in combination with composite operations from Chapter 3 give an important speed-up. In particular, composite operation DA is found to further reduce costs.

Also, notice that our techniques give the most significant reduction to multibase methods using $\mathcal{A}$={2,3,5} and $\mathcal{A}$={2,3,5,7}. This is due to the high efficiency of new quintupling and septupling operations, which have made bases 5 and 7 practical for scalar expansions.

Figure 7.3 details performance of our new *mb*NAF methods using bases $\mathcal{A}$={2,3}, $\mathcal{A}$={2,3,5} and $\mathcal{A}$={2,3,5,7}, and compares them against NAF and DB. It can be seen that all our methods outperform previous scalar multiplications. In particular, (2,3,5)NAF and (2,3,5,7)NAF offer comparable performance with reductions of 10.9% and 4.9% in comparison with traditional NAF and DB, respectively. Thus, we can state that those two algorithms are the fastest to compute scalar multiplications on standard curves over prime fields without pre-computations.

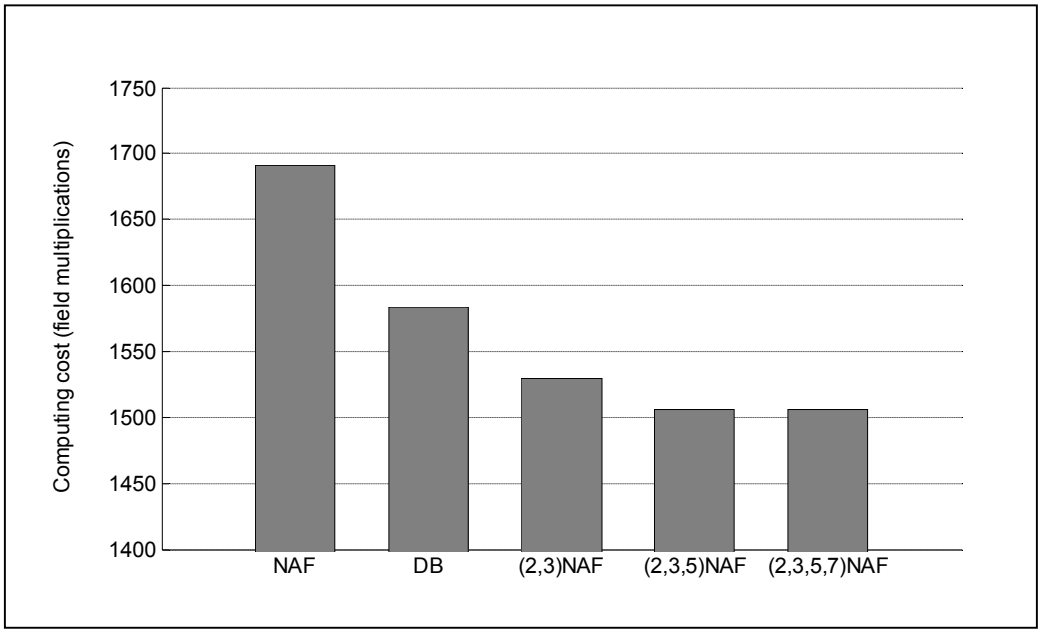


Figure 7.3. Comparison of performance of *mb*NAF methods with NAF and DB scalar multiplications

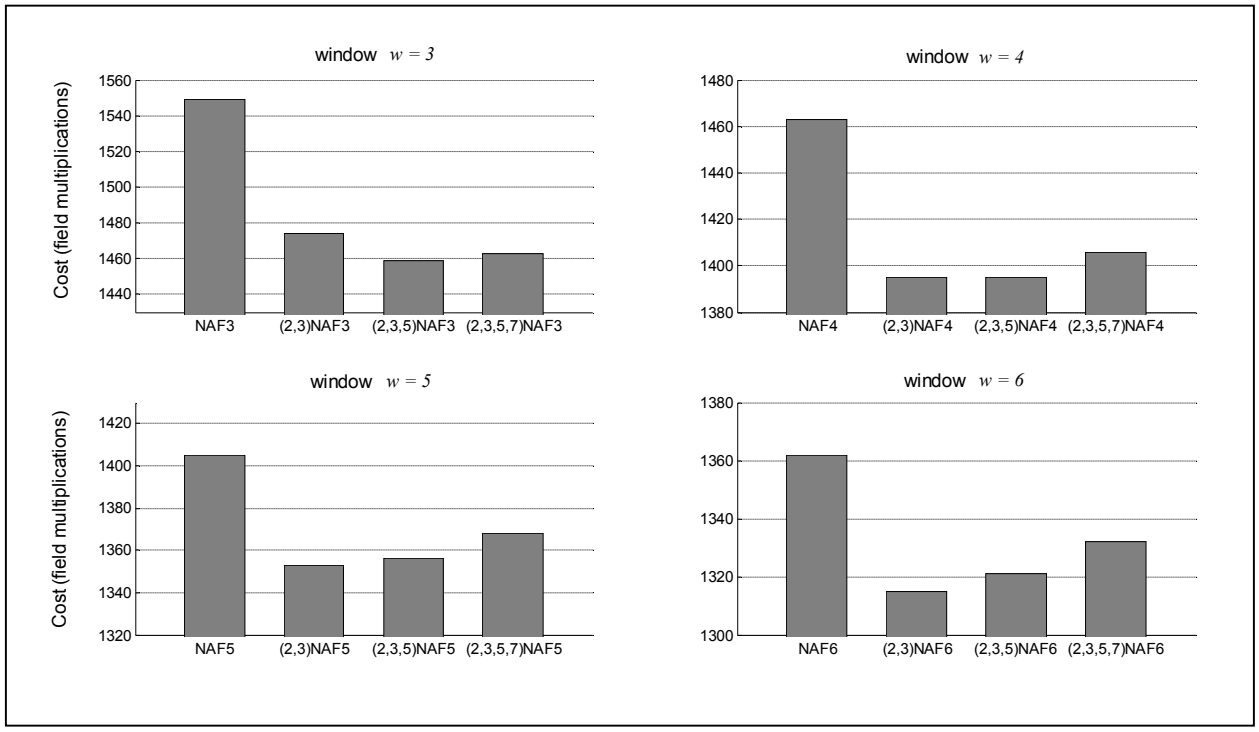


Figure 7.4. Comparison of performance of *wmb*NAF methods with *w*NAF for $w = 3, 4, 5$ and 6

Similarly, in Figure 7.4 we show performance of our *wmb*NAF methods 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 against *w*NAF. We see that (2,3,5)NAF_3 surpasses NAF_3 in 5.8%. (2,3)NAF_4 reduces computation time of NAF_4 in about 4.7%. And (2,3)NAF_5 and (2,3)NAF_6 present reductions of 3.7% and 3.4% in comparison to NAF_5 and NAF_6, respectively.

To our knowledge, for the case of scalar multiplication with pre-computations, *wmb*NAF shows superior performance that any other method in the literature.

Additionally, we wanted to analyze the cost achieved by other families of curves where composite operations are efficiently computed. As an example, we analyze the computing costs for the special ECC curves presented by [DIK06]. In such scenario, costs of doubling, tripling and addition are $5M + 4S$, $6M + 6S$ and $8M + 3S$, respectively.

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

Table 7.2 details performance results of our tests, and compare them with results using DB, ternary/binary approach [CJL+06] and *r*NAF [TYW04] (see Section 2.3.3) as detailed by [DIK06].

Method	Points	Cost
NAF [(a)]	0	1818*M*
DB [(a)]	0	1562*M*
DB [(b)]	0	1643*M*
Ternary/binary [CJL+06] [(b)]	0	1541*M*
This work: (2,3)$NAF_{1,1}$	0	1541*M*
NAF_3 [(a)]	1	1676*M*
This work: (2,3)$NAF_{2,1}$	1	1487*M*
$3NAF_2$ [(b)]	2	1507*M*
This work: (2,3)$NAF_{1,2}$	2	1413*M*
$6NAF_2$ [(b)]	5	1457*M*
This work: (2,3)$NAF_{1,2}$	5	1387*M*
$3NAF_3$ [(b)]	8	1392*M*
This work: (2,3)$NAF_{1,3}$	8	1337*M*

(a) Costs estimated in this work.
(b) Costs as estimated by [DIK06] with special parameters.

Table 7.2. Comparison of *wmb*NAF method with different scalar multiplications using special curves [DIK06] (n = 160 bits, 1S = 0.8M)

As we can see in the table above, *extended wmb*NAF achieves the highest performance among all methods including NAF, *w*NAF, DB and *r*NAF. Only in the case without pre-

computations the ternary/binary approach shows equivalent performance.

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 in this particular curve.

In the following, we summarize the main observations derived from our tests. For complete results of previous and other cases for our multibase methods, the reader is referred to Appendix F1.

- Among methods with no pre-computed points on standard curves, *mb*NAF using bases $\mathcal{A}$={2,3,5} and $\mathcal{A}$={2,3,5,7} presents the lowest cost. It introduces an improvement of about 10.9% in comparison with the traditional NAF.

- *wmb*NAF with w = 3, 4, 5 and 6 offers the best performance in terms of computing costs among methods requiring {1,3,7,15} pre-computed points on standard curves. In this case, using bases $\mathcal{A}$={2,3} and $\mathcal{A}$={2,3,5} offer comparable performance. Notice that, on standard curves using pre-computations, *wmb*NAF is superior to *extended wmb*NAF in every case and requires fewer pre-computed points.

- New composite operations (namely tripling, quintupling and septupling) presented in this work have reduced significantly cost of multibase methods including radices 3, 5 and 7, in comparison with multibase scalar multiplications using traditional operations.

- Fast formulae exploiting new composite DA reduces further the computing costs for NAF and multibase NAF methods.

- In the case of special curves [DIK06] with no pre-computations, *extended wmb*NAF using bases $\mathcal{A}$={2,3} have permitted a reduction of about 15.2% in comparison to traditional NAF.

- *Extended wmb*NAF using bases $\mathcal{A}$={2,3} offers the best performance in terms of

computing costs among methods requiring {1,2,5,8} pre-computed points on special curves. Most importantly, *extended wmb*NAF on special curves shows superior performance than any method on standard curves when requiring {2,5,8} pre-computed points. If one compares *extended wmb*NAF with the previous number of pre-computed points against best cases for *wmb*NAF on standard curves using {3,7,15} pre-computed points, performance is comparable and the latter requires extra memory in each case.

Protected Implementations

We have studied s*ix* possible cases for implementation of SSCA-protected scalar multiplications:

- Point operations with traditional *M-A-N-A* structure.
- Improved *M-A-N-A*-based formulae (Sections 5.2 and 5.3.2).
- New *M-N-A-M-N-A-A*-based formulae (Section 5.3).
- Improved *M-A-N-A*-based formulae using atomic DA operation (Section 5.3.2).
- *M-N-A-M-N-A-A*-based formulae using atomic DA (Section 5.3.2).
- New *S-N-A-M-N-A-A*-based formulae (Section 5.4).

We have investigated performance of SSCA-protected multibase scalar multiplication methods for the next 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, window w can be also $\{0,1\}$.

From our tests, we concluded that *M-N-A-M-N-A-A*-based formulae using atomic DA achieves the highest performance when assuming $1S = 1M$. In that case, Figure 7.5 and Figure 7.6 compare our best implementation case using *mb*NAF and *wmb*NAF, respectively, with bases $\mathcal{A}=\{2,3\}$, $\mathcal{A}=\{2,3,5\}$ and $\mathcal{A}=\{2,3,5,7\}$.

On the other hand, *S*-*N*-*A*-*M*-*N*-*A*-*A*-based formulae shows the lowest cost for implementations with ratio $S/M = 0.8$. In particular for this case, *w*NAF is further optimized and performs better than any other scalar multiplication method.

Complete details of our tests can be found in Appendix F2.

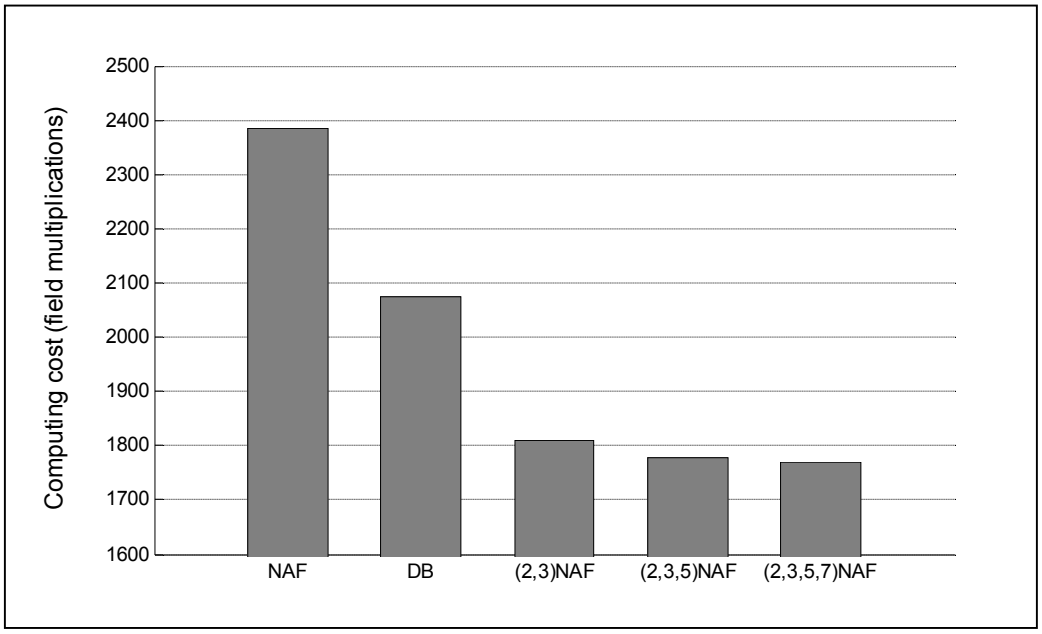


Figure 7.5. Comparison of performance of *mb*NAF with NAF and DB scalar multiplications protected against SSCA

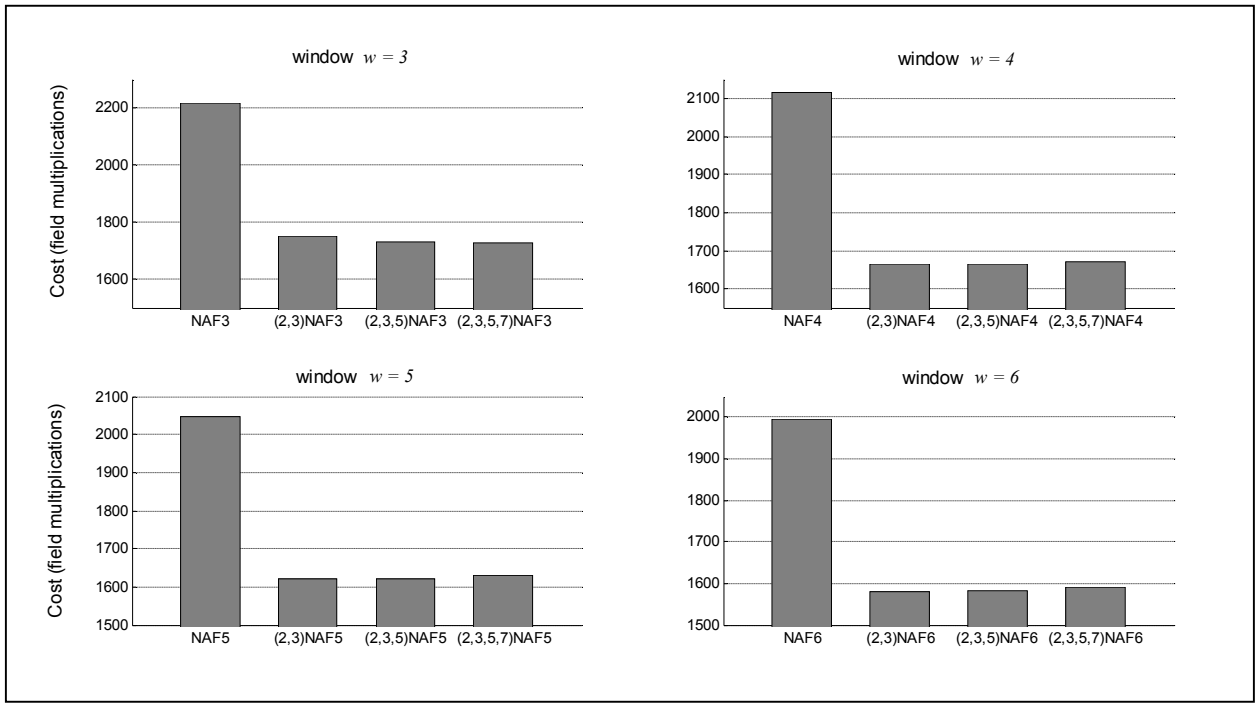


Figure 7.6. Comparison of performance of SSCA-protected *wmb*NAF and *w*NAF methods for $w = 3, 4, 5$ and 6

We can see in Table 7.5 that (2,3,5)NAF offers the best performance for a protected implementation without pre-computations. It presents a reduction in terms of computing time of 25.3% and 14.2% in comparison with NAF and DB, respectively. Also, in Table 7.6 we observe that (2,3,5)NAF_3 surpasses NAF_3 in 21.9%, and that (2,3,5)NAF_4 improves NAF_4 in 21.3%. Finally, (2,3)NAF_5 and (2,3)NAF_6 are the fastest for windows $w = 5$ and 6, and show reductions of 20.7% and 20.8%, respectively.

In conclusion, we can state that *mb*NAF and *wmb*NAF are the fastest methods for the scalar multiplication with protection against SSCA.

Chapter 8

Conclusions

8.1 Concluding Remarks

Scalar multiplication is the central and most time-consuming operation in curve-based cryptosystems such as ECC and HECC. In this work, we have tackled the problem of optimizing such operation on ECC standard curves over prime fields mainly in terms of speed. In such sense, we have proposed two innovative methodologies: optimization through utilization of composite operations, and reduction of computing costs by replacing field multiplications for squarings.

In the first case, new composite operations of the form dP and $dP+Q$ have been developed and shown to be superior to previous efforts. In particular new doubling addition (DA), quintupling (Q) and septupling (S) operations are shown to introduce significant savings in the computation of traditional or simultaneous scalar multiplication methods, and in the computation of pre-computed points in window-based approaches.

For the second case, we showed the way our methodology can be used to accelerate traditional formulae. In particular, this technique was shown to greatly improve performance of SSCA-protected and parallel/multiprocessor implementations. Performance comparison

with other methods indicated that our SSCA-protected implementations, which use innovative atomic structures, are not only faster but also more secure since they include squarings into the formulation, making them invulnerable to potential attacks that could exploit differences between field multiplications and squarings. For SIMD-like architectures, we developed ECC parallel formulae capable of executing *three* and *four* operations simultaneously. When compared against previous efforts to parallelize formulae at the point arithmetic level, our approaches showed reduction in computing times for all cases. Also, a new SSCA-protected scheme capable of executing *two* field operations in parallel was developed. In this scenario, our scheme was shown to be faster than best previous methods using Montgomery ladder or pipelining techniques.

Most remarkably, our substitution technique was applied to already efficient composite operations such as quintupling and septupling to further improve performance.

It is important to note that our methodology is generic and can be applied to any other curve-based cryptosystem working over prime fields.

Finally, by exploiting efficiency of new composite operations, we developed *three* new methods for the scalar multiplication based on multiple bases, which are sublinear in terms of Hamming weight. Our approach was made practical with the introduction of NAF-like algorithms that convert any integer to multibase, solving a previous problem found in the literature with double-base methods. In this sense, our conversion algorithms are efficient and do not consume extra memory. Most importantly, they can be applied to other areas of cryptography such as ECC over binary fields, HECC and pairing-based cryptosystems. As an example, we demonstrated performance improvement by using our multibase methods in the case of special curves developed by [DIK06].

Extensive tests with thousands of random numbers using 0, 1, 3, 7, 15 pre-computations indicate that our methods are superior to previous methods such as NAF, DB and *w*NAF for windowed and non-windowed scenarios. For instance, *mb*NAF using bases {2,3,5} or {2,3,5,7} were found to be the fastest when pre-computations are not allowed, introducing an improvement of 10.9% in comparison with NAF; and *wmb*NAF using $w = 3$ and bases

{2,3,5} was similarly shown to be the fastest when using *one* pre-computed point. For the latter, the savings are about 5.8%. On the other hand, *extended mb*NAF has shown high efficiency in settings where triplings are particularly efficient. For instance, for special curves [DIK06], *extended wmb*NAF achieves the lowest computing costs for 0, 1, 2, 5, 8 pre-computations. Furthermore, in the case of 2, 5 and 8 pre-computations *extended wmb*NAF over special curves surpasses performance of the best method over standard curves.

Greater improvements were found with our methods in SSCA-protected implementations using atomicity. In such case, *mb*NAF using bases {2,3,5} was also found to be the fastest with an improvement of 25.3% in comparison with the traditional NAF. On the other hand, in window-based approaches *wmb*NAF gave the highest performance with reductions of up to 21.9% in comparison with *w*NAF.

In conclusion, we can state that we have the most efficient scalar multiplication methods in terms of both speed and memory usage.

8.2 Future Work

Several research opportunities can be derived from this work.

We mainly concentrated efforts on optimizing the ECC scalar multiplication on standard curves over prime fields. Future work can be focused on applying introduced techniques to the optimization of point formulae on curves other than the standard ones. Of special interest would be to apply the substitution of multiplication for squarings to HECC formulae and to operations on special curves such as those proposed by [DIK06].

New multibase scalar multiplication methods can also be subject of further research. If more efficient high-order composite operations are developed, then it would greatly improve performance of our methods and/or justify the use of other bases beside the ones found

efficient in this work. Also, efficient composite operations on other cryptosystems would make increasingly interesting the use of these multibase methods on those settings. We did not explore in detail implementations of those cryptosystems where some efficient composite already exist. Further work could also focus in this topic.

Finally, we theoretically showed performance improvements at the point arithmetic level and confirmed findings with testing of thousands of random scalars and the evaluation of their expansions for the various representation methods. A complete hardware and/or software implementation would be highly useful to demonstrate in practice superiority of the multibase scalar multiplication using the new methodologies exposed in this work.

APPENDICES

A: OPTIMIZED ALGORITHM FOR POINT TRIPLING

Algorithm A: Optimized Point Tripling (Jacobian Coordinates), $E: \; y^2 = x^3 + ax + b$

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. If $a = -3$, then:	
3.1 $T_5 = T_1 + T_4$	$\{X_1 + Z_1^2\}$
3.2 $T_4 = T_1 - T_4$	$\{X_1 - Z_1^2\}$
3.3 $T_4 = T_4 \times T_5$	$\{(X_1 + Z_1^2)(X_1 - Z_1^2)\}$
3.4 $T_4 = 3T_4$	$\{\theta = 3(X_1 + Z_1^2)(X_1 - Z_1^2)\}$
4. Else:	
4.1 $T_4 = T_4^2$	$\{Z_1^4\}$
4.2 $T_5 = T_1^2$	$\{X_1^2\}$
4.3 $T_5 = 3T_5$	$\{3X_1^2\}$
4.4 $T_4 = a \times T_4$	$\{aZ_1^4\}$
4.5 $T_4 = T_4 + T_5$	$\{\theta = 3X_1^2 + aZ_1^4\}$
5. $T_5 = T_4^2$	$\{\theta^2\}$
6. $T_6 = T_2^2$	$\{Y_1^2\}$
7. $T_7 = 12T_6$	$\{12Y_1^2\}$
8. $T_7 = T_1 \times T_7$	$\{12X_1Y_1^2\}$
9. $T_7 = T_7 - T_5$	$\{\omega = 12X_1Y_1^2 - \theta^2\}$
10. $T_3 = T_3 \times T_7$	$\{Z_3 = Z_1\omega\}$
11. $T_4 = T_4 \times T_7$	$\{\alpha = \theta\omega\}$
12. $T_5 = T_6^2$	$\{Y_1^4\}$
13. $T_5 = 8T_5$	$\{\beta = 8Y_1^4\}$

14. $T_4 = T_5 - T_4$	$\{\beta - \alpha\}$
15. $T_8 = T_7^2$	$\{\omega^2\}$
16. $T_1 = T_1 \times T_8$	$\{X_1\omega^2\}$
17. $T_6 = 8T_6$	$\{8Y_1^2\}$
18. $T_6 = T_4 \times T_6$	$\{8Y_1^2(\beta - \alpha)\}$
19. $T_1 = T_1 + T_6$	$\{8Y_1^2(\beta - \alpha) + X_1\omega^2\}$
20. $T_5 = T_4 + T_5$	$\{2\beta - \alpha\}$
21. $T_4 = -T_4 \times T_5$	$\{(\alpha - \beta)(2\beta - \alpha)\}$
22. $T_4 = 4T_4$	$\{4(\alpha - \beta)(2\beta - \alpha)\}$
23. $T_5 = T_7 \times T_8$	$\{\omega^3\}$
24. $T_4 = T_4 - T_5$	$\{4(\alpha - \beta)(2\beta - \alpha) - \omega^3\}$
25. $T_2 = T_2 \times T_4$	
$\{Y_2 = Y_1[4(\alpha - \beta)(2\beta - \alpha) - \omega^3]\}$	
26. Return $(T_1, T_2, T_3) = (X_3, Y_3, Z_3)$	

This algorithm to compute the tripling of a point costs only $9M + 5S + 12A$ and $9M + 7S + 11A$ for the special ($a = -3$) and general case, respectively. As stated previously, we consider field multiplications by small constants and additions approximately equivalent, and their cost negligible in comparison with squaring and multiplication.

B1: ATOMIC POINT DOUBLING
(*M-A-N-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
$T_4 = T_3^2 \quad (Z_1^2)$	$T_5 = T_4 \times T_5 \quad (A.B)$	$T_5 = T_2^2 \quad (Y_1^2)$	$T_3 = T_2 \times T_3 \quad (Z_3)$
$T_5 = T_1 + T_4 \quad (A = X_1 + Z_1^2)$	$T_4 = T_5 + T_5 \quad (2A.B)$	$T_2 = T_2 + T_2 \quad (2Y_1)$	$T_2 = T_5 + T_5 \quad (4Y_1^2)$
$T_4 = -T_4 \quad (-Z_1^2)$	*	*	*
$T_4 = T_1 + T_4 \quad (B = X_1 - Z_1^2)$	$T_4 = T_4 + T_5 \quad (\alpha)$	$T_5 = T_5 + T_5 \quad (2Y_1^2)$	*
Δ5	**Δ6**	**Δ7**	**Δ8**
$T_2 = T_1 \times T_2 \quad (\beta)$	$T_6 = T_4^2 \quad (\alpha^2)$	$T_5 = T_5^2 \quad (4Y_1^4)$	$T_2 = T_2 \times T_4 \quad (\alpha(\beta - X_3))$
$T_1 = T_2 + T_2 \quad (2\beta)$	$T_1 = T_1 + T_6 \quad (X_3)$	$T_5 = T_5 + T_5 \quad (8Y_1^4)$	$T_5 = -T_5 \quad (-8Y_1^4)$
$T_1 = -T_1 \quad (-2\beta)$	*	$T_6 = -T_1 \quad (-X_3)$	$T_2 = T_2 + T_5 \quad (Y_3)$
*	*	$T_2 = T_2 + T_6 \quad (\beta - X_3)$	*

For the remainder of this work, "*" represents a dummy field operation that depends on the step it is placed. For instance, the three "*" in the third step of the upper half of the table represent dummy negations. "Δ*i*" represents the i^{th} atomic block. In a sequential implementation, atomic blocks are executed one at a time according to positions *i* in the formulae.

B2: ATOMIC POINT DOUBLING (*M-N-A-M-N-A-A*-BASED)

Input: $P = (X_1 , Y_1 , Z_1)$
Output: $2P = (X_3 , Y_3 , Z_3)$

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

$\Delta 1$	$\Delta 2$	$\Delta 3$	$\Delta 4$
$T_4 = T_3^2 \quad (Z_1^2)$	$T_5 = T_4 \times T_5 \quad (A.B)$	$T_5 = T_4^2 \quad (\alpha^2)$	$T_2 = T_2^2 \quad (4Y_1^4)$
*	*	*	$T_5 = -T_1 \quad (-X_3)$
$T_5 = T_1 + T_4 \ (A = X_1 + Z_1^2)$	$T_4 = T_5 + T_5 \quad (2A.B)$	$T_6 = T_2 + T_2 \quad (4Y_1^2)$	$T_5 = T_5 + T_6 \quad (\beta - X_3)$
$T_6 = T_2^2 \quad (Y_1^2)$	$T_3 = T_2 \times T_3 \quad (Z_3)$	$T_6 = T_1 \times T_6 \quad (\beta)$	$T_5 = T_4 \times T_5 \quad (\alpha(\beta - X_3))$
$T_4 = -T_4 \quad (-Z_1^2)$	*	$T_1 = -T_6 \quad (-\beta)$	$T_2 = -T_2 \quad (-4Y_1^4)$
$T_2 = T_2 + T_2 \quad (2Y_1)$	$T_4 = T_4 + T_5 \quad (\alpha)$	$T_1 = T_1 + T_1 \quad (-2\beta)$	$T_2 = T_2 + T_2 \quad (-8Y_1^4)$
$T_4 = T_1 + T_4 \ (B = X_1 - Z_1^2)$	$T_2 = T_6 + T_6 \quad (2Y_1^2)$	$T_1 = T_1 + T_5 \quad (X_3)$	$T_2 = T_2 + T_4 \quad (Y_3)$

B3: ATOMIC MIXED ADDITION (*M-A-N-A*-BASED)

Input: $P = (X_1 , Y_1 , Z_1)$ and $Q = (X_2 , Y_2)$
Output: $P + Q = (X_3, Y_3, Z_3, X_1', Y_1')$

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

Δ1	Δ2	Δ3	Δ4
$T_4 = T_3^2 \quad (Z_1^2)$	$T_5 = T_x \times T_4 \quad (Z_1^2 X_2)$	$T_6 = T_5^2 \quad (A^2)$	$T_7 = T_1 \times T_6 \quad (X_1')$
*	*	*	$T_8 = T_1 + T_1 \quad (2X_1')$
*	$T_6 = -T_1 \quad (-X_1)$	*	*
*	$T_5 = T_5 + T_6 \ (A = Z_1^2 X_2 - X_1)$	*	*
Δ5	**Δ6**	**Δ7**	**Δ8**
$T_9 = T_5 \times T_6 \quad (A^3)$	$T_4 = T_3 \times T_4 \quad (Z_1^3)$	$T_4 = T_y \times T_4 \quad (Z_1^3 Y_2)$	$T_{10} = T_4^2 \quad (B^2)$
$T_8 = T_8 + T_9 \quad (A^3 + 2X_1')$	*	*	*
*	*	$T_{10} = -T_2 \quad (-Y_1)$	$T_8 = -T_8 \quad (-A^3 - 2X_1')$
*	*	$T_4 = T_4 + T_{10} \quad (B = Z_1^3 Y_2 - Y_1)$	$T_1 = T_6 + T_8 \quad (X_3)$
Δ9	**Δ10**	**Δ11**	
$T_8 = T_2 \times T_9 \quad (Y_1')$	$T_6 = T_6 \times T_{10} \ (B(X_1' - X_3))$	$T_3 = T_3 \times T_5 \quad (Z_3)$	
*	*	*	
$T_6 = -T_1 \quad (-X_3)$	$T_9 = -T_8 \quad (-Y_1')$	$T_4 = -T_7 \quad (-X_1')$ (a)	
$T_6 = T_6 + T_7 \quad (X_1' - X_3)$	$T_2 = T_6 + T_9 \quad (Y_3)$	$T_4 = T_1 + T_4 \quad (X_3 - X_1')$ (a)	

(a) Field operations in Δ10 and Δ11 if a special addition follows.

B4: ATOMIC MIXED ADDITION (*M-N-A-M-N-A-A*-BASED)

Input: $P = (X_1 , Y_1 , Z_1)$ and $Q = (X_2 , Y_2)$
Output: $P + Q = (X_3, Y_3, Z_3, X_1', Y_1')$

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

Δ1	Δ2	Δ3	Δ4
$T_4 = T_3^2$ $\left(Z_1^2\right)$	$T_6 = T_5^2$ $\left(A^2\right)$	$T_9 = T_5 \times T_6$ $\left(A^3\right)$	$T_4 = T_y \times T_4$ $\left(Z_1^3 Y_2\right)$
*	*	*	$T_{10} = -T_2$ $\left(-Y_1\right)$
*	*	$T_8 = T_8 + T_9$ $\left(A^3 + 2X_1'\right)$	$T_4 = T_4 + T_{10}$ $\left(B = Z_1^3 Y_2 - Y_1\right)$
$T_5 = T_x \times T_4$ $\left(Z_1^2 X_2\right)$	$T_7 = T_1 \times T_6$ $\left(X_1'\right)$	$T_4 = T_3 \times T_4$ $\left(Z_1^3\right)$	$T_{10} = T_4^2$ $\left(B^2\right)$
$T_6 = -T_1$ $\left(-X_1\right)$	*	*	$T_8 = -T_8$ $\left(-A^3 - 2X_1'\right)$
$T_5 = T_5 + T_6$ $\left(A = Z_1^2 X_2 - X_1\right)$	$T_8 = T_1 + T_1$ $\left(2X_1'\right)$	*	$T_1 = T_6 + T_8$ $\left(X_3\right)$
*	*	*	*

Δ5	Δ6
$T_8 = T_2 \times T_9$ $\left(Y_1'\right)$	$T_3 = T_3 \times T_5$ $\left(Z_3\right)$
$T_6 = -T_1$ $\left(-X_3\right)$	$T_4 = -T_7$ $\left(-X_1'\right)$ (a)
$T_6 = T_6 + T_7$ $\left(X_1' - X_3\right)$	$T_4 = T_1 + T_4$ $\left(X_3 - X_1'\right)$ (a)
$T_6 = T_6 \times T_{10}$ $\left(B\left(X_1' - X_3\right)\right)$	$T_5 = T_4^2$ $\left(A^2\right)$ (a)
$T_9 = -T_8$ $\left(-Y_1'\right)$	$T_6 = -T_8$ $\left(-Y_1'\right)$ (a)
$T_2 = T_6 + T_9$ $\left(Y_3\right)$	$T_6 = T_2 + T_6$ $\left(B\right)$ (a)
*	*

(a) Field operations in Δ6 if a special addition follows.

B5: ATOMIC SPECIAL ADDITION WITH IDENTICAL *Z*-COORDINATE (*M-A-N-A*-BASED)

Input: $P = (X_1 , Y_1 , Z)$ and $Q = (X_2 , Y_2 , Z)$
Output: $P + Q = (X_3, Y_3, Z_3)$

$T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z$, $T_7 \leftarrow X_2$, $T_8 \leftarrow Y_2$

$\Delta 1$	$\Delta 2$	$\Delta 3$	$\Delta 4$
*	$T_5 = T_4^2 \quad (A^2)$	$T_7 = T_5 \times T_7 \quad (X_1' A^2)$	$T_5 = T_4 \times T_5 \quad (A^3)$
*	*	$T_9 = T_7 + T_7 \quad (2X_1' A^2)$	$T_9 = T_5 + T_9 \ (A^3 + 2X_1' A^2)$
$T_4 = -T_7 \quad (-X_1')$	$T_6 = -T_8 \quad (-Y_1')$	*	*
$T_4 = T_1 + T_4 \ (A = X_2 - X_1')$	$T_6 = T_2 + T_6 \ (B = Y_2 - Y_1')$	*	*
$\Delta 5$	**$\Delta 6$**	**$\Delta 7$**	**$\Delta 8$**
$T_1 = T_6^2 \quad (B^2)$	$T_2 = T_5 \times T_8 \quad (Y_1' A^3)$	$T_5 = T_5 \times T_6 \quad (B.C)$	$T_3 = T_3 \times T_4 \quad (Z_3)$
*	*	*	*
$T_9 = -T_9 \ (-A^3 - 2X_1' A^2)$	$T_5 = -T_1 \quad (-X_3)$	$T_2 = -T_2 \quad (-Y_1' A^3)$	*
$T_1 = T_1 + T_9 \quad (X_3)$	$T_5 = T_5 + T_7 \ (C = X_1' A^2 - X_3)$	$T_2 = T_2 + T_5 \quad (Y_3)$	*

Atomic DA using *M-A-N-A* can be built by consecutive execution of formulas in Appendices B3 (mixed addition) and B5 (special addition). In this case, $\Delta 11$ from the former can be merged with $\Delta 1$ from the latter as indicated by operations in (a) (see $\Delta 11$ in Appendix B3). Thus, the cost of atomic DA is reduced to only 18 atomic blocks.

B6: ATOMIC SPECIAL ADDITION WITH IDENTICAL *Z*-COORDINATE (*M-N-A-M-N-A-A*-BASED)

Input: $P = (X_1 , Y_1 , Z)$ and $Q = (X_2 , Y_2 , Z)$
Output: $P + Q = (X_3, Y_3, Z_3)$

$T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z$, $T_7 \leftarrow X_2$, $T_8 \leftarrow Y_2$

$\Delta 1$	$\Delta 2$	$\Delta 3$	$\Delta 4$
*	$T_7 = T_5 \times T_7 \quad (X_1' A^2)$	$T_1 = T_6^2 \quad (B^2)$	$T_5 = T_5 \times T_6 \quad (B.C)$
$T_4 = -T_7 \quad (-X_1')$	*	$T_9 = -T_9 \quad (-A^3 - 2X_1' A^2)$	$T_2 = -T_2 \quad (-Y_1' A^3)$
$T_4 = T_1 + T_4 \ (A = X_2 - X_1')$	$T_9 = T_7 + T_7 \quad (2X_1' A^2)$	$T_1 = T_1 + T_9 \quad (X_3)$	$T_2 = T_2 + T_5 \quad (Y_3)$
$T_5 = T_4^2 \quad (A^2)$	$T_5 = T_4 \times T_5 \quad (A^3)$	$T_2 = T_5 \times T_8 \quad (Y_1' A^3)$	$T_3 = T_3 \times T_4 \quad (Z_3)$
$T_6 = -T_8 \quad (-Y_1')$	*	$T_5 = -T_1 \quad (-X_3)$	*
$T_6 = T_2 + T_6 \ (B = Y_2 - Y_1')$	$T_9 = T_5 + T_9 \ (A^3 + 2X_1' A^2)$	$T_5 = T_5 + T_7 \ (C = X_1' A^2 - X_3)$	*
*	*	*	*

Atomic DA using *M-N-A-M-N-A-A* can be built by consecutive execution of formulas in Appendices B4 (mixed addition) and B6 (special addition). In this case, $\Delta 6$ from the former can be merged with $\Delta 1$ from the latter as indicated by operations in (a) (see $\Delta 6$ in Appendix B4). Thus, the cost of atomic DA is reduced to only 9 atomic blocks.

B7: ATOMIC POINT TRIPLING (*M-A-N-A*-BASED)

Input: $P = (X_1 , Y_1 , Z_1)$
Output: $3P = (X_3 , Y_3 , Z_3 , X_2 , Y_2)$

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

Δ1	Δ2	Δ3	Δ4
$T_4 = T_3^2 \quad (Z_1^2)$	$T_3 = T_2 \times T_3 \quad (Z_1')$	$T_4 = T_4 \times T_5 \quad (A.B)$	$T_2 = T_2^2 \quad (Y_1^2)$
$T_3 = T_3 + T_3 \quad (2Z_1)$	$T_4 = T_1 + T_4 \quad (A = X_1 + Z_1^2)$	$T_5 = T_4 + T_4 \quad (2A.B)$	$T_4 = T_4 + T_5 \quad (C = 3A.B)$
$T_5 = -T_4 \quad (-Z_1^2)$	*	*	*
*	$T_5 = T_1 + T_5 \quad (B = X_1 - Z_1^2)$	*	$T_2 = T_2 + T_2 \quad (2Y_1^2)$
Δ5	**Δ6**	**Δ7**	**Δ8**
$T_5 = T_4^2 \quad (C^2)$	$T_6 = T_1 \times T_2 \quad (X_1')$	$T_2 = T_2^2 \quad (4Y_1^4)$	$T_4 = T_1 \times T_4 \quad (C.D)$
$T_1 = T_1 + T_1 \quad (2X_1)$	*	$T_7 = T_1 + T_5 \quad (X_2)$	*
*	$T_1 = -T_6 \quad (-X_1')$	$T_1 = -T_7 \quad (-X_2)$	$T_2 = -T_2 \quad (-4Y_1^4)$
*	$T_1 = T_1 + T_1 \quad (-2X_1')$	$T_1 = T_1 + T_6 \quad (D = X_1' - X_2)$	$T_2 = T_2 + T_2 \quad (-Y_1')$
Δ9	**Δ10**	**Δ11**	**Δ12**
$T_5 = T_1^2 \quad (D^2)$	$T_6 = T_5 \times T_6 \quad (F = X_1' D^2)$	$T_5 = T_1 \times T_5 \quad (-D^3)$	$T_{10} = T_4^2 \quad (E^2)$
$T_8 = T_2 + T_4 \quad (Y_2)$	*	$T_1 = T_6 + T_6 \quad (2F)$	$T_1 = T_1 + T_{10} \quad (X_3)$
*	$T_9 = -T_1 \quad (-D)$	$T_1 = -T_1 \quad (-2F)$	$T_{10} = -T_1 \quad (-X_3)$
$T_4 = T_2 + T_8 \quad (E = Y_2 - Y_1')$	*	$T_1 = T_1 + T_5 \quad (-D^3 - 2F)$	*
Δ13	**Δ14**	**Δ15**	
$T_5 = T_2 \times T_5 \quad (Y_1' D^3)$	$T_2 = T_2 \times T_4 \quad (E(F - X_3))$	$T_3 = T_3 \times T_9 \quad (Z_3)$	
$T_2 = T_6 + T_{10} \quad (F - X_3)$	*	*	
*	$T_5 = -T_5 \quad (-Y_1' D^3)$	$T_4 = -T_7 \quad (-X_2)$ (a)	
*	$T_2 = T_2 + T_5 \quad (Y_3)$	$T_4 = T_1 + T_4 \quad (X_3 - X_2)$ (a)	

(a) Field operations in Δ15 if follows a special addition.

Atomic quintupling using *M-A-N-A* can be built by consecutive execution of formulas in Appendices B7 (tripling) and B5 (special addition). In this case, Δ15 from the former can be merged with Δ1 from the latter as indicated by operations in (a). Thus, the cost of atomic quintupling is reduced to only 22 atomic blocks.

Similarly, atomic septupling (and higher order operations) can be built by executing extra additions from Appendix B5 (special addition). In every case, operations in $\Delta 1$ (see Appendix B5) can be merged with the last atomic block of the precedent operation. Thus, the cost of the atomic septupling is reduced to 29 atomic blocks.

B8: ATOMIC POINT TRIPLING (*M-N-A-M-N-A-A*-BASED)

Input: $P = (X_1 , Y_1 , Z_1)$
Output: $3P = (X_3 , Y_3 , Z_3 , X_2 , Y_2)$

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

Δ1	Δ2	Δ3	Δ4
$T_4 = T_3^2$ $\left(Z_1^2\right)$	$T_4 = T_4 \times T_5$ $\left(A.B\right)$	$T_5 = T_4^2$ $\left(C^2\right)$	$T_2 = T_2^2$ $\left(4Y_1^4\right)$
*	*	*	$T_1 = -T_7$ $\left(-X_2\right)$
$T_3 = T_3 + T_3$ $\left(2Z_1\right)$	$T_5 = T_4 + T_4$ $\left(2A.B\right)$	$T_1 = T_1 + T_1$ $\left(2X_1\right)$	$T_1 = T_1 + T_6$ $\left(D = X_1' - X_2\right)$
$T_3 = T_2 \times T_3$ $\left(Z_1'\right)$	$T_2 = T_2^2$ $\left(Y_1^2\right)$	$T_6 = T_1 \times T_2$ $\left(X_1'\right)$	$T_4 = T_1 \times T_4$ $\left(C.D\right)$
$T_5 = -T_4$ $\left(-Z_1^2\right)$	*	$T_1 = -T_6$ $\left(-X_1'\right)$	$T_2 = -T_2$ $\left(-4Y_1^4\right)$
$T_4 = T_1 + T_4$ $\left(A = X_1 + Z_1^2\right)$	$T_4 = T_4 + T_5$ $\left(C = 3A.B\right)$	$T_1 = T_1 + T_1$ $\left(-2X_1'\right)$	$T_2 = T_2 + T_2$ $\left(-Y_1'\right)$
$T_5 = T_1 + T_5$ $\left(B = X_1 - Z_1^2\right)$	$T_2 = T_2 + T_2$ $\left(2Y_1^2\right)$	$T_7 = T_1 + T_5$ $\left(X_2\right)$	$T_8 = T_2 + T_4$ $\left(Y_2\right)$
Δ5	**Δ6**	**Δ7**	**Δ8**
$T_5 = T_1^2$ $\left(D^2\right)$	$T_5 = T_1 \times T_5$ $\left(-D^3\right)$	$T_5 = T_2 \times T_5$ $\left(Y_1' D^3\right)$	$T_3 = T_3 \times T_9$ $\left(Z_3\right)$
*	*	$T_{10} = -T_1$ $\left(-X_3\right)$	$T_4 = -T_7$ $\left(-X_2\right)$ (a)
$T_4 = T_2 + T_8$ $\left(E = Y_2 - Y_1'\right)$	$T_1 = T_6 + T_6$ $\left(2F\right)$	$T_2 = T_6 + T_{10}$ $\left(F - X_3\right)$	$T_4 = T_1 + T_4$ $\left(X_3 - X_2\right)$ (a)
$T_6 = T_5 \times T_6$ $\left(F = X_1' D^2\right)$	$T_{10} = T_4^2$ $\left(E^2\right)$	$T_2 = T_2 \times T_4$ $\left(E\left(F - X_3\right)\right)$	$T_5 = T_4^2$ $\left(A^2\right)$ (a)
$T_9 = -T_1$ $\left(-D\right)$	$T_1 = -T_1$ $\left(-2F\right)$	$T_5 = -T_5$ $\left(-Y_1' D^3\right)$	$T_6 = -T_8$ $\left(-Y_2\right)$ (a)
*	$T_1 = T_1 + T_5$ $\left(-D^3 - 2F\right)$	$T_2 = T_2 + T_5$ $\left(Y_3\right)$	$T_6 = T_2 + T_6$ $\left(B\right)$ (a)
*	$T_1 = T_1 + T_{10}$ $\left(X_3\right)$	*	*

(a) Field operations in Δ8 if follows a special addition.

Atomic quintupling using *M-N-A-M-N-A-A* can be built by consecutive execution of formulas in Appendices B8 (tripling) and B6 (special addition). In this case, Δ8 from the former can be merged with Δ1 from the latter as indicated by operations in (a). Thus, the cost of atomic quintupling is reduced to only 11 atomic blocks.

Similarly, atomic septupling (and higher order operations) can be built by executing extra additions from Appendix B6 (special addition). It is important to note that every two point operations, operations in Δ1 (see Appendix B6) can be merged with the last atomic block of the precedent operation. Thus, cost of the atomic septupling is fixed to 15 atomic blocks.

B9: OPTIMIZED ATOMIC POINT TRIPLING (*M-A-N-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
$T_4 = T_3^2 \quad (Z_1^2)$	$T_6 = T_2^2 \quad (Y_1^2)$	$T_7 = T_1 \times T_9 \quad (4X_1Y_1^2)$	$T_4 = T_4 \times T_5 \quad (A.B)$
*	$T_6 = T_6 + T_6 \quad (2Y_1^2)$	$T_4 = T_1 + T_4 \quad (A = X_1 + Z_1^2)$	$T_5 = T_4 + T_4 \quad (2A.B)$
$T_5 = -T_4 \quad (-Z_1^2)$	*	*	*
$T_5 = T_1 + T_5 \quad (B = X_1 - Z_1^2)$	$T_9 = T_6 + T_6 \quad (4Y_1^2)$	*	$T_5 = T_4 + T_5 \quad (\theta = 3A.B)$

Δ5	Δ6	Δ7	Δ8
$T_4 = T_5^2 \quad (\theta^2)$	$T_6 = T_6^2 \quad (4Y_1^4)$	$T_5 = T_5 \times T_9 \quad (\alpha)$	$T_3 = T_3 \times T_7 \quad (Z_3)$
$T_8 = T_7 + T_7 \quad (8X_1Y_1^2)$	$T_7 = T_7 + T_8 \quad (12X_1Y_1^2)$	$T_6 = T_6 + T_6 \quad (\beta)$	$T_5 = T_6 + T_5 \quad (\beta - \alpha)$
*	$T_4 = -T_4 \quad (-\theta^2)$	$T_5 = -T_5 \quad (-\alpha)$	*
*	$T_7 = T_4 + T_7 \quad (\omega)$	*	$T_6 = T_6 + T_5 \quad (C = 2\beta - \alpha)$

Δ9	Δ10	Δ11	Δ12
$T_4 = T_7^2 \quad (\omega^2)$	$T_9 = T_5 \times T_9 \quad (8Y_1^2(\beta - \alpha))$	$T_1 = T_1 \times T_4 \quad (X_1\omega^2)$	$T_4 = T_4 \times T_7 \quad (\omega^3)$
$T_9 = T_9 + T_9 \quad (8Y_1^2)$	*	$T_5 = T_5 + T_5 \quad (4D)$	*
*	$T_5 = -T_5 \quad (D = \alpha - \beta)$	*	$T_4 = -T_4 \quad (-\omega^3)$
*	$T_5 = T_5 + T_5 \quad (2D)$	$T_1 = T_1 + T_9 \quad (X_3)$	*

Δ13	Δ14
$T_6 = T_6 \times T_5 \quad (4C.D)$	$T_2 = T_2 \times T_6 \quad (Y_3)$
$T_6 = T_6 + T_4 \quad (4C.D - \omega^3)$	*
*	*
*	*

B10: OPTIMIZED 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
$T_4 = T_3^2 \quad (Z_1^2)$	$T_7 = T_1 \times T_9 \quad (4X_1Y_1^2)$	$T_4 = T_5^2 \quad (\theta^2)$	$T_5 = T_5 \times T_9 \quad (\alpha)$
$T_5 = -T_4 \quad (-Z_1^2)$	*	*	$T_5 = -T_5 \quad (-\alpha)$
$T_5 = T_1 + T_5 \quad (B = X_1 - Z_1^2)$	$T_4 = T_1 + T_4 \quad (A = X_1 + Z_1^2)$	$T_8 = T_7 + T_7 \quad (8X_1Y_1^2)$	$T_6 = T_6 + T_6 \quad (\beta)$
$T_6 = T_2^2 \quad (Y_1^2)$	$T_4 = T_4 \times T_5 \quad (A.B)$	$T_6 = T_6^2 \quad (4Y_1^4)$	$T_3 = T_3 \times T_7 \quad (Z_3)$
*	*	$T_4 = -T_4 \quad (-\theta^2)$	*
$T_6 = T_6 + T_6 \quad (2Y_1^2)$	$T_5 = T_4 + T_4 \quad (2A.B)$	$T_7 = T_7 + T_8 \quad (12X_1Y_1^2)$	$T_5 = T_6 + T_5 \quad (\beta - \alpha)$
$T_9 = T_6 + T_6 \quad (4Y_1^2)$	$T_5 = T_4 + T_5 \quad (\theta = 3A.B)$	$T_7 = T_4 + T_7 \quad (\omega)$	$T_6 = T_6 + T_5 \quad (C = 2\beta - \alpha)$

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

C1: ATOMIC POINT DOUBLING
(*S-N-A-M-N-A-A* OR *S-N-A-A-M-N-A-A*)

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
$T_4 = T_3^2 \quad (Z_1^2)$	$T_2 = T_2^2 \quad (Y_1^2)$	$T_1 = T_4^2 \quad (\alpha^2)$	$T_2 = T_2^2 \quad (4Y_1^4)$
*	*	*	$T_2 = -T_2 \quad (-4Y_1^4)$
$T_5 = T_1 + T_4 \quad (A = X_1 + Z_1^2)$	$T_2 = T_2 + T_2 \quad (2Y_1^2)$	$T_5 = T_5 + T_5 \quad (-2X_1)$	$T_5 = T_1 + T_5 \quad (X_3 - \beta)$
* (a)	* (a)	* (a)	* (a)
$T_3 = T_2 \times T_3 \quad (Y_1 Z_1)$	$T_4 = T_4 \times T_5 \quad (A.B)$	$T_5 = T_2 \times T_5 \quad (-\beta)$	$T_5 = T_4 \times T_5 \, (\alpha (X_3 - \beta))$
$T_4 = -T_4 \quad (-Z_1^2)$	$T_5 = -T_1 \quad (-X_1)$	*	$T_5 = -T_5 \quad (\alpha (\beta - X_3))$
$T_4 = T_1 + T_4 \quad (B = X_1 - Z_1^2)$	$T_4 = T_4 + T_4 \quad (2A.B)$	$T_1 = T_1 + T_5 \quad (\alpha^2 - \beta)$	$T_2 = T_2 + T_2 \quad (-8Y_1^4)$
$T_3 = T_3 + T_3 \quad (Z_3)$	$T_4 = T_4 + T_4 \quad (\alpha)$	$T_1 = T_1 + T_5 \quad (X_3)$	$T_2 = T_2 + T_5 \quad (Y_3)$

(a) Dummy field additions are added for the case of the scalar multiplication using ternary bases.

When the previous formula is used as a sequential SSCA-protected atomic point doubling for traditional scalar multiplications that only involve doublings and additions in their execution, the step with dummy operations in (a) is not considered. Thus, we have the atomic structure: *S-N-A-M-N-A-A*.

Dummy field additions are added in (a) to make this formula suitable for scalar multiplications that involve tripling, doubling and addition, as is the case of multibase scalar multiplication with bases $\mathcal{A} = \{2,3\}$. Thus, we have the atomic structure: *S-N-A-A-M-N-A-A*.

C2: ATOMIC POINT ADDITION
(*S-N-A-M-N-A-A* OR *S-N-A-A-M-N-A-A*)

Input: $P = (X_1 , Y_1 , Z_1)$ and $Q = (X_2 , Y_2)$
Output: $P + Q = (X_3, Y_3, Z_3)$

$T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z_1$, $T_x \leftarrow X_2$, $T_y \leftarrow Y_2$, and $-Y_2^2$ is pre-computed

Δ1		Δ2		Δ3	
$T_4 = T_3^2$	$\left(Z_1^2\right)$	$T_1 = T_5^2$	$\left(\beta^2\right)$	$T_4 = T_4^2$	$\left(A^2\right)$
*		*		*	
$T_2 = T_2 + T_2$	$\left(2Y_1\right)$	$T_8 = T_1 + T_4$	$\left(Z_1^2 + \beta^2\right)$	$T_4 = T_4 - Y_2^2$	$\left(A^2 - Y_2^2\right)$
*	(a)	*	(a)	*	(a)
$T_5 = T_4 \times T_x$	$\left(Z_1^2 X_2\right)$	$T_3 = T_3 \times T_4$	$\left(Z_1^3\right)$	$T_6 = T_1 \times T_6$	$\left(-2X_1\beta^2\right)$
$T_6 = -T_1$	$\left(-X_1\right)$	*		$T_2 = -T_2$	$\left(-2Y_1\right)$
$T_5 = T_5 + T_6$	$\left(\beta\right)$	$T_4 = T_3 + T_y$	$\left(A = Z_1^3 + Y_2\right)$	$T_6 = T_6 + T_6$	$\left(-4X_1\beta^2\right)$
$T_7 = T_3 + T_5$	$\left(Z_1 + \beta\right)$	$T_1 = T_1 + T_1$	$\left(2\beta^2\right)$	$T_4 = T_2 + T_4$	$\left(A^2 - Y_2^2 - 2Y_1\right)$
Δ4		**Δ5**		**Δ6**	
$T_3 = T_3^2$	$\left(Z_1^6\right)$	$T_5 = T_4^2$	$\left(\alpha^2\right)$	$T_7 = T_7^2$	$\left(\left(Z_1 + \beta\right)^2\right)$
$T_3 = -T_3$	$\left(-Z_1^6\right)$	$T_1 = -T_1$	$\left(-4\beta^3\right)$	$T_8 = -T_8$	$\left(-Z_1^2 - \beta^2\right)$
$T_4 = T_3 + T_4$	$\left(\alpha\right)$	$T_5 = T_1 + T_5$	$\left(\alpha^2 - 4\beta^3\right)$	$T_3 = T_7 + T_8$	$\left(Z_3\right)$
*	(a)	*	(a)	*	(a)
$T_1 = T_1 \times T_5$	$\left(2\beta^3\right)$	$T_2 = T_1 \times T_2$	$\left(8Y_1\beta^3\right)$	$T_5 = T_4 \times T_5$	$\left(\alpha\left(X_3 - 4X_1\beta^2\right)\right)$
*		$T_2 = -T_2$	$\left(-8Y_1\beta^3\right)$	$T_5 = -T_5$	$\left(\alpha\left(4X_1\beta^2 - X_3\right)\right)$
$T_1 = T_1 + T_1$	$\left(4\beta^3\right)$	$T_1 = T_3 + T_5$	$\left(X_3\right)$	$T_2 = T_2 + T_5$	$\left(Y_3\right)$
$T_3 = T_6 + T_6$	$\left(-8X_1\beta^2\right)$	$T_5 = T_1 + T_6$	$\left(X_3 - 4X_1\beta^2\right)$	*	

(a) Dummy field additions are added for the case of scalar multiplication using ternary bases.

Similarly to Appendix C1, dummy field additions are added in (a) to make this formula suitable for scalar multiplications that involve tripling, doubling and addition, as is the case of multibase scalar multiplication with bases $\mathcal{A} = \{2,3\}$.

C3: ATOMIC POINT TRIPLING (*S-N-A-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
$T_4 = T_3^2$ $\left(Z_1^2\right)$	$T_7 = T_2^2$ $\left(4Y_1^2\right)$	$T_6 = T_5^2$ $\left(\theta^2\right)$	$T_9 = T_8^2$ $\left(\omega^2\right)$
$T_5 = -T_4$ $\left(-Z_1^2\right)$	*	$T_3 = -T_3$ $\left(-Z_1\right)$	$T_8 = -T_8$ $\left(\omega\right)$
$T_5 = T_1 + T_5$ $\left(B=X_1-Z_1^2\right)$	*	$T_8 = T_7 + T_7$ $\left(8Y_1^2\right)$	$T_6 = T_6 + T_9$ $\left(\theta^2+\omega^2\right)$
$T_6 = T_1 + T_4$ $\left(A=X_1+Z_1^2\right)$	*	$T_8 = T_7 + T_8$ $\left(12Y_1^2\right)$	$T_5 = T_5 + T_8$ $\left(\theta+\omega\right)$
*	$T_5 = T_5 \times T_6$ $\left(A.B\right)$	$T_8 = T_1 \times T_8$ $\left(12X_1Y_1^2\right)$	$T_8 = T_8 \times T_9$ $\left(\omega^3\right)$
*	*	$T_8 = -T_8$ $\left(-12X_1Y_1^2\right)$	*
$T_2 = T_2 + T_2$ $\left(2Y_1\right)$	$T_5 = T_5 + T_5$ $\left(2A.B\right)$	$T_8 = T_6 + T_8$ $\left(-\omega\right)$	$T_4 = T_4 + T_9$ $\left(Z_1^2+\omega^2\right)$
*	$T_5 = T_5 + T_5$ $\left(\theta\right)$	$T_3 = T_3 + T_8$ $\left(-Z_1-\omega\right)$	$T_1 = T_1 + T_1$ $\left(2X_1\right)$
Δ5	**Δ6**	**Δ7**	**Δ8**
$T_5 = T_5^2$ $\left((\theta+\omega)^2\right)$	$T_7 = T_7^2$ $\left(2\beta\right)$	$T_3 = T_3^2$ $\left((Z_1+\omega)^2\right)$	$T_4 = T_3^2$ $\left(Z_3^2\right)$ (a)
$T_6 = -T_6$ $\left(-\theta^2-\omega^2\right)$	$T_5 = -T_5$ $\left(-2\alpha\right)$	$T_5 = -T_5$ $\left(E=2\alpha-2\beta\right)$	$T_5 = -T_4$ $\left(-Z_3^2\right)$ (a)
$T_5 = T_5 + T_6$ $\left(2\alpha\right)$	$T_5 = T_5 + T_7$ $\left(C=2\beta-2\alpha\right)$	$T_3 = T_3 + T_4$ $\left(Z_3\right)$	$T_5 = T_1 + T_5$ $\left(B=X_3-Z_3^2\right)$ (a)
$T_1 = T_1 + T_1$ $\left(4X_1\right)$	$T_6 = T_6 + T_6$ $\left(16Y_1^2\right)$	*	$T_6 = T_1 + T_4$ $\left(A=X_3+Z_3^2\right)$ (a)
$T_1 = T_1 \times T_9$ $\left(4X_1\omega^2\right)$	$T_6 = T_5 \times T_6$ $\left(16Y_1^2C\right)$	$T_5 = T_5 \times T_7$ $\left(D.E\right)$	$T_2 = T_2 \times T_8$ $\left(Y_3\right)$
*	$T_4 = -T_4$ $\left(-Z_1^2-\omega^2\right)$	$T_8 = -T_8$ $\left(-\omega^3\right)$	*
$T_2 = T_2 + T_2$ $\left(4Y_1\right)$	$T_7 = T_5 + T_7$ $\left(D=4\beta-2\alpha\right)$	$T_8 = T_5 + T_8$ $\left(D.E-\omega^3\right)$	$T_2 = T_2 + T_2$ $\left(2Y_3\right)$ (a)
$T_6 = T_7 + T_7$ $\left(8Y_1^2\right)$	$T_1 = T_1 + T_6$ $\left(X_3\right)$	$T_2 = T_2 + T_2$ $\left(8Y_1\right)$	*

(a) Field operations in Δ8 that correspond to the first atomic block of the following tripling.

Every tripling right after another tripling saves *one* atomic block by merging its first atomic block with the last atomic block of the previous tripling. Hence, field operations (a) in Δ8 are executed when repeated triplings are computed. Otherwise, dummy operations are executed in (a).

D1: THREE-PROCESSOR DOUBLING

Doubling: $2(X_1, Y_1, Z_1, X_1^2, Z_1^2, Z_1^4) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^3 / Z_3^4)$

$T_1 \leftarrow X_1$, $T_2 \leftarrow Y_1$, $T_3 \leftarrow Z_1$, $T_4 \leftarrow X_1^2$, $T_5 \leftarrow Z_1^2$, $T_6 \leftarrow Z_1^4$

Processor1	Processor2	Processor3
1. $T_7 = T_4 + T_4$ $\left(2X_1^2\right)$	*	$T_8 = T_6 + T_6$ $\left(2Z_1^4\right)$
2. $T_7 = T_4 + T_7$ $\left(3X_1^2\right)$	*	$T_6 = T_6 + T_8$ $\left(3Z_1^4\right)$
3. $T_7 = T_6 + T_7$ (α)	$T_3 = T_2 + T_3$ $(Y_1 + Z_1)$	*
4. $T_6 = T_7^2$ $\left(\alpha^2\right)$	$T_3 = T_3^2$ $\left((Y_1 + Z_1)^2\right)$	$T_2 = T_2^2$ $\left(Y_1^2\right)$
5. *	$T_5 = T_2 + T_5$ $\left(Y_1^2 + Z_1^2\right)$	$T_1 = T_1 + T_2$ $\left(X_1 + Y_1^2\right)$
6. *	$T_3 = T_3 - T_5$ (Z_3)	*
7. $T_2 = T_2^2$ $\left(Y_1^4\right)$	$T_5 = T_3^2$ $\left(Z_3^2\right)$	$T_1 = T_1^2$ $\left(\left(X_1 + Y_1^2\right)^2\right)$
8. $T_4 = T_2 + T_4$ $\left(X_1^2 + Y_1^4\right)$	$T_{10} = T_2 + T_2$ $\left(2Y_1^4\right)$	$T_1 = T_1 + T_1$ $\left(2\left(X_1 + Y_1^2\right)^2\right)$
9. $T_4 = T_4 + T_4$ $\left(2\left(X_1^2 + Y_1^4\right)\right)$	$T_{10} = T_{10} + T_{10}$ $\left(4Y_1^4\right)$	$T_2 = T_1 + T_1$ $\left(4\left(X_1 + Y_1^2\right)^2\right)$
10. $T_8 = T_4 + T_4$ $\left(4\left(X_1^2 + Y_1^4\right)\right)$	$T_{10} = T_{10} + T_{10}$ $\left(8Y_1^4\right)$	$T_1 = T_1 + T_2$ $\left(6\left(X_1 + Y_1^2\right)^2\right)$
11. $T_4 = T_4 + T_8$ $\left(6\left(X_1^2 + Y_1^4\right)\right)$	$T_2 = T_1 - T_6$ $\left(6\left(X_1 + Y_1^2\right)^2 - \alpha^2\right)$	$T_9 = T_2 - T_8$ (2β)
12. $T_1 = T_6 - T_9$ (X_3)	$T_2 = T_2 - T_4$ $(\beta - X_3)$	*
13. $T_4 = T_1^2$ $\left(X_3^2\right)$	$T_2 = T_2 \times T_7$ $(\alpha(\beta - X_3))$	$T_6 = T_3 \times T_5$ $\left(Z_3^3\right)$ | $T_6 = T_5^2$ $\left(Z_3^4\right)$ (a)
14. $T_7 = T_4 + T_4$ $\left(2X_3^2\right)$ (b)	$T_2 = T_2 - T_{10}$ (Y_3)	$T_8 = T_6 + T_6$ $\left(2Z_3^4\right)$ (b)

(a) Z_3^3 if next operation is a point addition, or Z_3^4 if next operation is a doubling or tripling.

(b) Field additions from the first step of a following doubling or tripling.

If a doubling or tripling (Appendix D3) comes right after a doubling, field additions from the first step of the following doubling or tripling can be merged with the last step of the doubling formulae, saving one field addition.

The previous formula requires $1M + 2S + 11A$ and 10 variables. If there are memory constraints, then we could replace steps from 8 to 14 by the block given below, to have a slightly more costly formula with $1M + 2S + 12A$ but with a memory requirement of only 8 variables:

8. $T_4 = T_2 + T_4 \quad \left(X_1^2 + Y_1^4\right)$	*	*
9. $T_4 = T_1 - T_4 \quad \left(\left(X_1 + Y_1^2\right)^2 - X_1^2 - Y_1^4\right)$	*	*
10. $T_4 = T_4 + T_4 \quad (\beta)$	$T_2 = T_2 + T_2 \quad \left(2Y_1^4\right)$	*
11. $T_1 = T_4 + T_4 \quad (2\beta)$	$T_2 = T_2 + T_2 \quad \left(4Y_1^4\right)$	*
12. $T_1 = T_6 - T_1 \quad (X_3)$	$T_2 = T_2 + T_2 \quad \left(8Y_1^4\right)$	*
13. *	$T_8 = T_4 - T_1 \quad (\beta - X_3)$	*
14. $T_4 = T_1^2 \quad \left(X_3^2\right)$	$T_7 = T_7 \times T_8 \quad (\alpha(\beta - X_3))$	$T_6 = T_3 \times T_5 \ \left(Z_3^3\right) \mid T_6 = T_5^2 \ \left(Z_3^4\right)$ (a)
15. $T_7 = T_4 + T_4 \quad \left(2X_3^2\right)$ (b)	$T_2 = T_7 - T_2 \quad (Y_3)$	$T_8 = T_6 + T_6 \quad \left(2Z_3^4\right)$ (b)

(a) Z_3^3 if next operation is a point addition, or Z_3^4 if next operation is a doubling or tripling.

(b) Field additions from the first step of a following doubling or tripling.

D2: THREE-PROCESSOR ADDITION

Mixed Addition : $(X_1, Y_1, Z_1, Z_1^2, Z_1^3) + (X_2, Y_2) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^4)$

$T_1 \leftarrow X_1,\ T_2 \leftarrow Y_1,\ T_3 \leftarrow Z_1,\ T_5 \leftarrow Z_1^2,\ T_6 \leftarrow Z_1^3,\ T_x \leftarrow X_2,\ T_y \leftarrow Y_2$

Processor1		Processor2		Processor3	
1. $T_8 = T_8 \times T_y$	$(Z_1^3 Y_2)$	$T_7 = T_5 \times T_x$	$(Z_1^2 X_2)$	*	
2. $T_8 = T_8 - T_2$	$(Z_1^3 Y_2 - Y_1)$	$T_7 = T_7 - T_1$	(β)	*	
3. $T_8 = T_8 + T_8$	(α)	$T_3 = T_3 + T_7$	$(Z_1 + \beta)$	*	
4. $T_6 = T_8^2$	(α^2)	$T_3 = T_3^2$	$((Z_1 + \beta)^2)$	$T_4 = T_7^2$	(β^2)
5. $T_1 = T_1 + T_1$	$(2X_1)$	$T_9 = T_7 + T_7$	(2β)	$T_5 = T_4 + T_5$	$(Z_1^2 + \beta^2)$
6. $T_1 = T_1 + T_1$	$(4X_1)$	$T_7 = T_9 + T_9$	(4β)	$T_3 = T_3 - T_5$	(Z_3)
7. $T_5 = T_1 \times T_4$	$(4X_1\beta^2)$	$T_7 = T_4 \times T_7$	$(4\beta^3)$	$T_9 = T_2 \times T_9$	$(2Y_1\beta)$
8. $T_1 = T_5 + T_5$	$(8X_1\beta^2)$	$T_6 = T_6 - T_7$	$(\alpha^2 - 4\beta^3)$	$T_9 = T_9 + T_9$	$(4Y_1\beta)$
9. $T_1 = T_6 - T_1$	(X_3)	*		$T_7 = T_4 + T_4$	$(2\beta^2)$
10. *		$T_2 = T_5 - T_1$	$(4X_1\beta^2 - X_3)$	*	
11. $T_4 = T_1^2$	(X_3^2)	*		$T_5 = T_3^2$	(Z_3^2)
12. $T_9 = T_7 \times T_9$	$(8Y_1\beta^3)$	$T_2 = T_2 \times T_8$	$(\alpha(4X_1\beta^2 - X_3))$	$T_6 = T_5^2$	(Z_3^4)
13. $T_7 = T_4 + T_4$	$(2X_3^2)$ (a)	$T_2 = T_2 - T_9$	(Y_3)	$T_8 = T_6 + T_6$	$(2Z_3^4)$ (a)

(a) Field additions from the first step of a following doubling or tripling.

If a doubling (Appendix D1) or tripling (Appendix D3) comes right after an addition, field additions from the first step of the following doubling or tripling can be merged with the last step of the addition formulae, saving one field addition.

D3: THREE-PROCESSOR TRIPLING

Tripling : $3(X_1, Y_1, Z_1, X_1^2, Z_1^2, Z_1^4) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^3 / Z_3^4)$

$T_1 \leftarrow X_1,\ T_2 \leftarrow Y_1,\ T_3 \leftarrow Z_1,\ T_4 \leftarrow X_1^2,\ T_5 \leftarrow Z_1^2,\ T_6 \leftarrow Z_1^4$

Processor1	Processor2	Processor3
1. $T_7 = T_4 + T_4$ $\left(2X_1^2\right)$	*	$T_8 = T_6 + T_6$ $\left(2Z_1^4\right)$
2. $T_7 = T_4 + T_7$ $\left(3X_1^2\right)$	*	$T_6 = T_6 + T_8$ $\left(3Z_1^4\right)$
3. $T_7 = T_7 - T_6$ (θ)	*	*
4. $T_6 = T_7^2$ $\left(\theta^2\right)$	$T_8 = T_2^2$ $\left(Y_1^2\right)$	$T_{10} = T_2^2$ $\left(Y_1^2\right)$
5. *	$T_9 = T_1 + T_8$ $\left(X_1 + Y_1^2\right)$	$T_{10} = T_{10} + T_{10}$ $\left(2Y_1^2\right)$
6. $T_8 = T_8^2$ $\left(Y_1^4\right)$	$T_9 = T_9^2$ $\left(A = \left(X_1 + Y_1^2\right)^2\right)$	$T_{10} = T_{10}^2$ $\left(4Y_1^4\right)$
7. $T_8 = T_8 + T_4$ $\left(X_1^2 + Y_1^4\right)$	*	$T_{10} = T_{10} + T_{10}$ $\left(8Y_1^4\right)$
8. $T_8 = T_9 - T_8$ $\left(B = A - X_1^2 - Y_1^4\right)$	*	$T_{10} = T_{10} + T_{10}$ (2β)
9. $T_8 = T_8 + T_8$ $(2B)$	*	*
10. $T_9 = T_8 + T_8$ $(4B)$	$T_1 = T_1 + T_1$ $(2X_1)$	$T_4 = T_8 - T_6$ $\left(2B - \theta^2\right)$
11. $T_8 = T_4 + T_9$ (ω)	$T_1 = T_1 + T_1$ $(4X_1)$	$T_2 = T_2 + T_2$ $(2Y_1)$
12. $T_7 = T_7 + T_8$ $(\theta + \omega)$	$T_3 = T_3 + T_8$ $(Z_1 + \omega)$	$T_2 = T_2 + T_2$ $(4Y_1)$
13. $T_7 = T_7^2$ $\left((\theta + \omega)^2\right)$	$T_3 = T_3^2$ $\left((Z_1 + \omega)^2\right)$	$T_4 = T_8^2$ $\left(\omega^2\right)$
14. $T_6 = T_4 + T_6$ $\left(\theta^2 + \omega^2\right)$	*	$T_5 = T_4 + T_5$ $\left(Z_1^2 + \omega^2\right)$
15. $T_7 = T_7 - T_6$ (2α)	$T_9 = T_{10} + T_{10}$ (4β)	$T_3 = T_3 - T_5$ (Z_3)
16. $T_{10} = T_{10} - T_7$ $(C = 2\beta - 2\alpha)$	$T_9 = T_9 - T_7$ $(D = 4\beta - 2\alpha)$	*
17. $T_1 = T_1 \times T_4$ $\left(4X_1\omega^2\right)$	$T_6 = T_2^2$ $\left(16Y_1^2\right)$	$T_5 = T_3^2$ $\left(Z_3^2\right)$
18. $T_6 = T_6 \times T_{10}$ $\left(16Y_1^2 C\right)$	$T_9 = T_9 \times T_{10}$ $(C.D)$	$T_8 = T_4 \times T_8$ $\left(\omega^3\right)$
19. $T_1 = T_1 + T_6$ (X_3)	$T_9 = -T_8 - T_9$ $\left(-C.D - \omega^3\right)$	$T_2 = T_2 + T_2$ $(8Y_1)$
20. $T_4 = T_1^2$ $\left(X_3^2\right)$	$T_2 = T_2 \times T_9$ (Y_3)	$T_6 = T_3 \times T_5$ $\left(Z_3^3\right)$ | $T_6 = T_5^2$ $\left(Z_3^4\right)$ (a)

(a) Z_3^3 if next operation is a point addition, or Z_3^4 if next operation is a doubling or tripling.

D4: FOUR-PROCESSOR DOUBLING

Doubling: $2(X_1, Y_1, Z_1, X_1^2, Z_1^2, Z_1^4) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^3 / Z_3^4)$

$T_1 \leftarrow X_1,\ T_2 \leftarrow Y_1,\ T_3 \leftarrow Z_1,\ T_4 \leftarrow X_1^2,\ T_5 \leftarrow Z_1^2,\ T_6 \leftarrow Z_1^4$

Processor1	Processor2	Processor3	Processor4
$T_7 = T_4 + T_4 \quad (2X_1^2)$	$T_8 = T_6 + T_6 \quad (2Z_1^4)$	*	*
$T_4 = T_4 + T_7 \quad (3X_1^2)$	$T_6 = T_6 + T_8 \quad (3Z_1^4)$	*	*
$T_4 = T_4 - T_6 \quad (\alpha)$	$T_3 = T_2 + T_3 \quad (Y_1 + Z_1)$	$T_7 = T_7 + T_7 \quad (4X_1^2)$	*
$T_6 = T_4^2 \quad (\alpha^2)$	$T_3 = T_3^2 \quad (A = (Y_1 + Z_1)^2)$	$T_2 = T_2^2 \quad (Y_1^2)$	*
*	$T_5 = T_2 + T_5 \quad (B = Y_1^2 + Z_1^2)$	$T_8 = T_2 + T_2 \quad (2Y_1^2)$	$T_1 = T_1 + T_1 \quad (2X_1)$
*	$T_3 = T_3 - T_5 \quad (A - B)$	$T_1 = T_1 + T_8 \quad (2X_1 + 2Y_1^2)$	*
$T_2 = T_6 + T_6 \quad (2\alpha^2)$	$T_3 = T_3 + T_3 \quad (Z_3)$	$T_7 = T_7 + T_7 \quad (8X_1^2)$	*
*	$T_5 = T_3^2 \quad (Z_3^2)$	$T_1 = T_1^2 \quad (C=4(X_1+Y_1^2)^2)$	$T_8 = T_8^2 \quad (4Y_1^4)$
$T_2 = T_2 + T_7 \quad (2\alpha^2 + 8X_1^2)$	*	$T_1 = T_1 + T_1 \quad (2C)$	$T_8 = T_8 + T_8 \quad (8Y_1^4)$
$T_2 = T_2 + T_2 \quad (4\alpha^2 + 16X_1^2)$	*	$T_1 = T_1 - T_8 \quad (2C - 8Y_1^4)$	$T_9 = T_8 + T_8 \quad (16Y_1^4)$
$T_8 = T_1 + T_1 \quad (4C - 16Y_1^4)$	*	$T_7 = T_1 - T_7 \quad (4\beta)$	$T_9 = T_9 + T_9 \quad (32Y_1^4)$
$T_1 = T_2 - T_8 \quad (X_3)$	*	$T_4 = T_4 + T_7 \quad (D=\alpha+4\beta)$	$T_9 = T_9 + T_9 \quad (64Y_1^4)$
*	$T_9 = T_6 + T_9 \quad (G = \alpha^2 + 64Y_1^4)$	$T_2 = T_4 - T_1 \quad (D - X_3)$	$T_7 = T_7 - T_1 \quad (F=4\beta-X_3)$
$T_4 = T_1^2 \quad (X_3^2)$	$T_6=T_3\times T_5 \quad (Z_3^3) \mid T_6=T_5^2 \quad (Z_3^4)$ [a]	$T_2 = T_2^2 \quad (E = (D - X_3)^2)$	$T_{10} = T_7^2 \quad (F^2)$
$T_7 = T_4 + T_4 \quad (2X_3^2)$ [b]	$T_8 = T_6 + T_6 \quad (2Z_3^4)$ [b]	$T_9 = T_9 + T_{10} \quad (F^2 + G)$	*
$T_4 = T_4 + T_7 \quad (3X_3^2)$ [b]	$T_6 = T_6 + T_8 \quad (3Z_3^4)$ [b]	$T_2 = T_2 - T_9 \quad (Y_3)$	*

(a) Z_3^3 if next operation is a point addition, or Z_3^4 if next operation is a doubling or tripling.

(b) Field additions from the first two steps of a following doubling or tripling.

If a doubling or tripling (Appendix D6) comes right after a doubling, field additions from the first two steps of the following doubling or tripling can be merged with the last two steps of the doubling formulae, saving two field additions.

D5: FOUR-PROCESSOR ADDITION

Mixed Addition : $(X_1, Y_1, Z_1, Z_1^2, Z_1^3) + (X_2, Y_2) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^4)$

$T_1 \leftarrow X_1,\ T_2 \leftarrow Y_1,\ T_3 \leftarrow Z_1,\ T_5 \leftarrow Z_1^2,\ T_6 \leftarrow Z_1^3,\ T_x \leftarrow X_2,\ T_y \leftarrow Y_2$

Processor1	Processor2	Processor3	Processor4
$T_6 = T_6 \times T_y \quad (Z_1^3 Y_2)$	$T_7 = T_5 \times T_x \quad (Z_1^2 X_2)$	$T_4 = T_2^2 \quad (Y_1^2)$	*
$T_6 = T_6 - T_2 \quad (\alpha)$	$T_7 = T_7 - T_1 \quad (\beta)$	*	*
*	$T_3 = T_3 + T_7 \quad (Z_1 + \beta)$	$T_2 = T_2 + T_7 \quad (Y_1 + \beta)$	$T_1 = T_1 + T_1 \quad (2X_1)$
$T_{10} = T_6^2 \quad (\alpha^2)$	$T_3 = T_3^2 \quad ((Z_1 + \beta)^2)$	$T_2 = T_2^2 \quad ((Y_1 + \beta)^2)$	$T_9 = T_7^2 \quad (\beta^2)$
$T_8 = T_{10} + T_{10} \quad (2\alpha^2)$	$T_5 = T_5 + T_9 \quad (Z_1^2 + \beta^2)$	$T_4 = T_4 + T_9 \quad (Y_1^2 + \beta^2)$	$T_7 = T_7 + T_7 \quad (2\beta)$
$T_8 = T_8 + T_8 \quad (4\alpha^2)$	$T_3 = T_3 - T_5 \quad (Z_3)$	$T_2 = T_2 - T_4 \quad (2Y_1\beta)$	$T_9 = T_9 + T_9 \quad (2\beta^2)$
$T_7 = T_7 \times T_9 \quad (4\beta^3)$	$T_5 = T_3^2 \quad (Z_3^2)$	$T_2 = T_2 \times T_9 \quad (4Y_1\beta^3)$	$T_4 = T_1 \times T_9 \quad (4X_1\beta^2)$
$T_1 = T_8 - T_7 \quad (4\alpha^2 - 4\beta^3)$	*	$T_2 = T_2 + T_2 \quad (8Y_1\beta^3)$	$T_9 = T_4 + T_4 \quad (8X_1\beta^2)$
$T_1 = T_1 - T_9 \quad (X_3)$	*	$T_2 = T_2 + T_{10} \quad (\alpha^2 + 8Y_1\beta^3)$	$T_6 = T_4 + T_6 \quad (A = \alpha + 4X_1\beta^2)$
*	*	$T_{10} = T_4 - T_1 \quad (4X_1\beta^2 - X_3)$	$T_9 = T_6 - T_1 \quad (A - X_3)$
$T_4 = T_1^2 \quad (X_3^2)$	$T_6 = T_5^2 \quad (Z_3^4)$	$T_{10} = T_{10}^2 \quad ((4X_1\beta^2 - X_3)^2)$	$T_9 = T_9^2 \quad (B = (A - X_3)^2)$
$T_7 = T_4 + T_4 \quad (2X_3^2)$ (a)	$T_8 = T_6 + T_6 \quad (2Z_3^4)$ (a)	*	$T_9 = T_9 - T_{10} \quad (B - (4X_1\beta^2 - X_3)^2)$
$T_4 = T_4 + T_7 \quad (3X_3^2)$ (a)	$T_6 = T_6 + T_8 \quad (3Z_3^4)$ (a)	*	$T_2 = T_9 - T_2 \quad (Y_3)$

(a) Field additions from the first two steps of a following doubling or tripling

If a doubling (Appendix D4) or tripling (Appendix D6) comes right after an addition, field additions from the first two steps of the following doubling or tripling can be merged with the last two steps of the addition formulae, saving two field additions.

D6: FOUR-PROCESSOR TRIPLING

Tripling : $3(X_1, Y_1, Z_1, X_1^2, Z_1^2, Z_1^4) = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^3 / Z_3^4)$

$T_1 \leftarrow X_1,\ T_2 \leftarrow Y_1,\ T_3 \leftarrow Z_1,\ T_4 \leftarrow X_1^2,\ T_5 \leftarrow Z_1^2,\ T_6 \leftarrow Z_1^4$

Processor1	Processor2	Processor3	Processor4
$T_7 = T_4 + T_4 \quad (2X_1^2)$	$T_8 = T_6 + T_6 \quad (2Z_1^4)$	*	*
$T_4 = T_4 + T_7 \quad (3X_1^2)$	$T_6 = T_6 + T_8 \quad (3Z_1^4)$	*	*
$T_6 = T_4 - T_6 \quad (\theta)$	*	$T_{14} = T_2 + T_2 \quad (2Y_1)$	*
$T_8 = T_6^2 \quad (\theta^2)$	$T_2 = T_2^2 \quad (Y_1^2)$	$T_9 = T_{14}^2 \quad (4Y_1^2)$	*
$T_{10} = T_2 + T_2 \quad (2Y_1^2)$	$T_{11} = T_1 + T_2 \quad (A = X_1 + Y_1^2)$	$T_9 = T_9 + T_9 \quad (8Y_1^2)$	*
$T_4 = T_4 + T_4 \quad (6X_1^2)$	$T_{12} = T_{11} + T_{11} \quad (2A)$	*	*
$T_{10} = T_{10}^2 \quad (4Y_1^4)$	$T_{12} = T_{12}^2 \quad (4A^2)$	$T_2 = T_2^2 \quad (Y_1^4)$	$T_{11} = T_{11}^2 \quad (A^2)$
$T_{13} = T_8 + T_{10} \quad (4Y_1^4 + \theta^2)$	$T_{12} = T_{12} - T_4 \quad (4A^2 - 6X_1^2)$	$T_2 = T_2 + T_2 \quad (2Y_1^4)$	$T_{11} = T_{11} + T_{11} \quad (2A^2)$
$T_{13} = T_2 + T_{13} \quad (6Y_1^4 + \theta^2)$	$T_{12} = T_{11} + T_{12} \quad (6A^2 - 6X_1^2)$	$T_{14} = T_{14} + T_{14} \quad (4Y_1)$	$T_{10} = T_{10} + T_{10} \quad (8Y_1^4)$
$T_{13} = T_{12} - T_{13} \quad (\omega)$	*	*	$T_{10} = T_{10} + T_{10} \quad (2\beta)$
$T_8 = T_8 + T_{10} \quad (2\beta + \theta^2)$	$T_3 = T_3 + T_{13} \quad (Z_1 + \omega)$	$T_6 = T_6 + T_{13} \quad (\theta + \omega)$	$T_{12} = T_{10} + T_{10} \quad (4\beta)$
$T_2 = T_{13}^2 \quad (\omega^2)$	$T_3 = T_3^2 \quad ((Z_1 + \omega)^2)$	$T_6 = T_6^2 \quad (B = (\theta + \omega)^2)$	$T_{11} = T_{10}^2 \quad (4\beta^2)$
$T_{13} = T_2 + T_{13} \quad (\omega + \omega^2)$	$T_5 = T_2 + T_5 \quad (Z_1^2 + \omega^2)$	$T_6 = T_6 - T_2 \quad (B - \omega^2)$	$T_{10} = T_8 + T_{10} \quad (4\beta + \theta^2)$
$T_1 = T_1 + T_2 \quad (E = X_1 + \omega^2)$	$T_3 = T_3 - T_5 \quad (Z_3)$	$T_8 = T_8 - T_6 \quad (C = 2\beta - 2\alpha)$	$T_{10} = T_{10} - T_6 \quad (D = 4\beta - 2\alpha)$
$T_1 = T_1^2 \quad (E^2)$	$T_6 = T_2^2 \quad (\omega^4)$	$T_{13} = T_{13}^2 \quad (H = (\omega + \omega^2)^2)$	$T_5 = T_8^2 \quad (C^2)$
$T_1 = T_1 - T_6 \quad (F = E^2 - \omega^4)$	$T_2 = T_2 + T_6 \quad (G = \omega^2 + \omega^4)$	$T_{11} = T_{11} - T_{13} \quad (K = 4\beta^2 - H)$	$T_{12} = T_{12} + T_{12} \quad (64Y_1^4)$
$T_1 = T_1 + T_1 \quad (2F)$	$T_2 = T_2 + T_{11} \quad (L = K + G)$	$T_{12} = T_5 + T_{12} \quad (C^2 + 64Y_1^4)$	$T_{13} = T_8 + T_9 \quad (8Y_1^2 + C)$
$T_{11} = T_8^2 \quad (C^2)$	$T_5 = T_3^2 \quad (Z_3^2)$	$T_{10} = T_{10}^2 \quad (D^2)$	$T_{13} = T_{13}^2 \quad (I = (8Y_1^2 + C)^2)$
$T_1 = T_1 - T_7 \quad (4X_1\omega^2)$	$T_{11} = T_{10} + T_{11} \quad (J = C^2 + D^2)$	$T_{14} = T_2 + T_{14} \quad (M = 4Y_1 + L)$	$T_{13} = T_{13} - T_{12} \quad (I - C^2 - 64Y_1^4)$
$T_1 = T_1 + T_{13} \quad (X_3)$	$T_9 = T_9 + T_9 \quad (16Y_1^2)$	$T_7 = T_{14} - T_{11} \quad (N = M - J)$	$T_2 = T_2 - T_{11} \quad (\mu = L - J)$
$T_4 = T_1^2 \quad (X_3^2)$	$T_6 = T_3 \times T_5 \quad (Z_3^3)$ $\mid$ $T_6 = T_5^2 \quad (Z_3^4)$ (a)	$T_{11} = T_7^2 \quad (N^2)$	$T_2 = T_2^2 \quad (\mu^2)$
$T_7 = T_4 + T_4 \quad (2X_3^2)$ (b)	$T_8 = T_6 + T_6 \quad (2Z_3^4)$ (b)	$T_9 = T_{11} - T_9 \quad (N^2 - 16Y_1^2)$	*
$T_4 = T_4 + T_7 \quad (3X_3^2)$ (b)	$T_6 = T_6 + T_8 \quad (3Z_3^4)$ (b)	$T_2 = T_9 - T_2 \quad (Y_3)$	*

(a) Z_3^3 if next operation is a point addition, or Z_3^4 if next operation is a doubling or tripling.

(b) Field additions from the first two steps of a following doubling or tripling.

If a doubling (Appendix D4) or tripling comes right after a tripling, field additions from the first two steps of the following doubling or tripling can be merged with the last two steps of the tripling formulae, saving two field additions.

E1: TWO-PARALLEL SSCA-PROTECTED DOUBLING

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	
Processor1	Processor2	Processor1	Processor2
$T_4 = T_3^2 \quad (Z_1^2)$	$T_5 = T_2^2 \quad (Y_1^2)$	$T_6 = T_4^2 \quad (\alpha^2)$	$T_5 = T_5^2 \quad (4Y_1^4)$
$T_6 = -T_4 \quad (-Z_1^2)$	*	*	$T_5 = -T_5 \quad (-4Y_1^4)$
$T_4 = T_1 + T_4 \; (A = X_1 + Z_1^2)$	$T_5 = T_5 + T_5 \quad (2Y_1^2)$	$T_1 = T_1 + T_6 \quad (X_3)$	$T_5 = T_5 + T_5 \quad (-8Y_1^4)$
$T_6 = T_1 + T_6 \; (B = X_1 - Z_1^2)$	$T_7 = T_1 + T_1 \quad (2X_1)$	$T_6 = T_1 + T_7 \quad (X_3 - \beta)$	$T_2 = T_2 + T_2 \quad (2Y_1)$
$T_4 = T_4 \times T_6 \quad (A.B)$	$T_7 = T_5 \times T_7 \quad (\beta)$	$T_6 = T_4 \times T_6 \; (\alpha(X_3 - \beta))$	$T_3 = T_2 \times T_3 \quad (Z_3)$
*	$T_7 = -T_7 \quad (-\beta)$	$T_6 = -T_6 \quad (\alpha(\beta - X_3))$	*
$T_4 = T_4 + T_4 \quad (2A.B)$	$T_1 = T_7 + T_7 \quad (-2\beta)$	$T_2 = T_5 + T_6 \quad (Y_3)$	*
$T_4 = T_4 + T_4 \quad (\alpha)$	*		*

For Appendices E1, E2 and E3, two processing units execute in parallel one *parallel* atomic block at a time (Δ1, Δ2, Δ3, and so on). Each *parallel* atomic block has the atomic structure: *S-N-A-A-M-N-A-A* to protect against simple side-channel attacks.

E2: TWO-PARALLEL SSCA-PROTECTED ADDITION

Input: $P = (X_1 , Y_1 , Z_1)$ and $Q = (X_2 , Y_2)$
Output: $P + Q = (X_3 , Y_3 , Z_3)$

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

$\Delta 1$		$\Delta 2$	
Processor1	Processor2	Processor1	Processor2
$T_4 = T_3^2 \quad (Z_1^2)$	$T_5 = T_y^2 \quad (Y_2^2)$	$T_8 = T_7^2 \quad (\beta^2)$	$T_3 = T_3^2 \quad ((Z_1+\beta)^2)$
*	*	$T_8 = -T_8 \quad (-\beta^2)$	$T_4 = -T_4 \quad (-Z_1^2)$
*	*	$T_1 = T_1 + T_1 \quad (2X_1)$	$T_4 = T_4 + T_8 \quad (-Z_1^2 - \beta^2)$
*	*	$T_1 = T_1 + T_1 \quad (4X_1)$	$T_3 = T_3 + T_4 \quad (Z_3)$
$T_6 = T_3 \times T_4 \quad (Z_1^3)$	$T_7 = T_4 \times T_x \quad (Z_1^2 X_2)$	$T_4 = T_1 \times T_8 \quad (A = -4X_1\beta^2)$	$T_7 = T_7 \times T_8 \quad (-\beta^3)$
*	$T_8 = -T_1 \quad (-X_1)$	*	*
*	$T_7 = T_7 + T_8 \quad (\beta)$	$T_1 = T_4 + T_4 \quad (B = -8X_1\beta^2)$	$T_7 = T_7 + T_7 \quad (-2\beta^3)$
*	$T_3 = T_3 + T_7 \quad (Z_1 + \beta)$	$T_8 = T_6 + T_y \quad (Z_1^3 + Y_2)$	$T_7 = T_7 + T_7 \quad (-4\beta^3)$

$\Delta 3$		$\Delta 4$	
Processor1	Processor2	Processor1	Processor2
$T_6 = T_6^2 \quad (Z_1^6)$	$T_8 = T_8^2 \quad ((Z_1^3 + Y_2)^2)$	$T_5 = T_6^2 \quad (\alpha^2)$	*
*	*	*	*
$T_6 = T_5 + T_6 \quad (C = Z_1^6 + Y_2^2)$	$T_2 = T_2 + T_2 \quad (2Y_1)$	$T_1 = T_1 + T_5 \quad (X_3)$	*
$T_6 = T_2 + T_6 \quad (C + 2Y_1)$	*	$T_4 = T_1 + T_4 \quad (X_3 + A)$	*
*	$T_2 = T_2 \times T_7 \quad (-8Y_1\beta^3)$	$T_4 = T_4 \times T_6 \quad (\alpha(X_3 + A))$	*
$T_6 = -T_6 \quad (-C - 2Y_1)$	*	$T_4 = -T_4 \quad \alpha(-A - X_3)$	*
$T_6 = T_8 + T_6 \quad (\alpha)$	$T_1 = T_1 + T_7 \quad (-4\beta^3 + B)$	$T_2 = T_2 + T_4 \quad (Y_3)$	*
*	*	*	*

E3: TWO-PARALLEL SSCA-PROTECTED TRIPLING

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	
Processor1	Processor2
$T_4 = T_3^2 \quad (Z_1^2)$	$T_5 = T_2^2 \quad (Y_1^2)$
$T_6 = -T_4 \quad (-Z_1^2)$	*
$T_4 = T_1 + T_4 \quad (A = X_1 + Z_1^2)$	$T_5 = T_5 + T_5 \quad (2Y_1^2)$
$T_6 = T_1 + T_6 \quad (B = X_1 - Z_1^2)$	$T_5 = T_5 + T_5 \quad (4Y_1^2)$
$T_4 = T_4 \times T_6 \quad (A.B)$	$T_7 = T_1 \times T_5 \quad (4X_1Y_1^2)$
*	*
$T_4 = T_4 + T_4 \quad (2A.B)$	$T_6 = T_7 + T_7 \quad (8X_1Y_1^2)$
$T_4 = T_4 + T_4 \quad (\theta)$	$T_7 = T_6 + T_7 \quad (12X_1Y_1^2)$

Δ2	
Processor1	Processor2
$T_5 = T_5^2 \quad (2\beta)$	$T_6 = T_4^2 \quad (\theta^2)$
*	$T_6 = -T_6 \quad (-\theta^2)$
$T_2 = T_2 + T_2 \quad (2Y_1)$	$T_7 = T_6 + T_7 \quad (\omega)$
$T_4 = T_4 + T_7 \quad (\theta + \omega)$	*
*	*
*	*
$T_2 = T_2 + T_2 \quad (4Y_1)$	*
*	*

Δ3	
Processor1	Processor2
$T_4 = T_4^2 \quad ((\theta + \omega)^2)$	$T_8 = T_7^2 \quad (\omega^2)$
*	$T_8 = -T_8 \quad (-\omega^2)$
$T_1 = T_1 + T_1 \quad (2X_1)$	$T_6 = T_6 + T_8 \quad (-\theta^2 - \omega^2)$
$T_1 = T_1 + T_1 \quad (4X_1)$	$T_6 = T_4 + T_6 \quad (2\alpha)$
$T_1 = T_1 \times T_8 \quad (-4X_1\omega^2)$	$T_4 = T_7 \times T_8 \quad (-\omega^3)$
$T_1 = -T_1 \quad (4X_1\omega^2)$	*
*	$T_7 = T_3 + T_7 \quad (Z_1 + \omega)$
*	*

Δ4	
Processor1	Processor2
$T_7 = T_7^2 \quad ((Z_1 + \omega)^2)$	$T_3 = T_3^2 \quad (Z_1^2)$
$T_6 = -T_6 \quad (-2\alpha)$	$T_3 = -T_3 \quad (-Z_1^2)$
$T_6 = T_5 + T_6 \quad (C = 2\beta - 2\alpha)$	$T_3 = T_3 + T_8 \quad (-Z_1^2 - \omega^2)$
$T_5 = T_5 + T_6 \quad (D = 4\beta - 2\alpha)$	$T_3 = T_3 + T_7 \quad (Z_3)$
$T_5 = T_5 \times T_6 \quad (C.D)$	*
$T_5 = -T_5 \quad (-C.D)$	*
$T_5 = T_4 + T_5 \quad (-C.D - \omega^3)$	*
*	*

Δ5	
Processor1	Processor2
$T_4 = T_2^2 \quad (16Y_1^2)$	*
*	$T_2 = T_2 + T_2 \quad (8Y_1)$
*	*
*	*
$T_4 = T_4 \times T_6 \quad (16Y_1^2C)$	$T_2 = T_2 \times T_5 \quad (Y_3)$
*	*
$T_1 = T_1 + T_4 \quad (X_3)$	*
*	*

F1: TEST RESULTS FOR UNPROTECTED IMPLEMENTATIONS

Method	Points	Standard curves (2.2)			Special curves [DIK06]
		Traditional	Fast	Fast using DA	
NAF	0	1691.26*M*	1648.97*M*	1638.42*M*	1818.19*M*
(2,3)NAF	0	1583.91*M*	1536.63*M*	1529.10*M*	1606.53*M*
(2,3,5)NAF	0	1647.68*M*	1512.68*M*	1506.29*M*	-
(2,3,5,7)NAF	0	1683.59*M*	1512.04*M*	1506.30*M*	-
$(2,3)NAF_{1,1}$	0	1583.91*M*	1536.63*M*	1529.10*M*	1541.00*M*
$(2,3,5)NAF_{1,1,0}$	0	1667.89*M*	1533.21*M*	1527.01*M*	-
NAF_3	1	1549.48*M*	1509.95*M*	1502.06*M*	1676.08*M*
$(2,3)NAF_3$	1	1523.23*M*	1480.92*M*	1474.05*M*	1602.14*M*
$(2,3,5)NAF_3$	1	1589.00*M*	1465.26*M*	1459.35*M*	-
$(2,3,5,7)NAF_3$	1	1627.23*M*	1468.87*M*	1463.52*M*	-
$(2,3)NAF_{2,1}$	1	1522.38*M*	1472.40*M*	1466.55*M*	1487.11*M*
$(2,3,5)NAF_{2,1,0}$	1	1559.04*M*	1469.55*M*	1464.08*M*	-
$(2,3)NAF_{1,2}$	2	1528.91*M*	1473.36*M*	1468.10*M*	1412.96*M*
$(2,3)NAF_{0,2}$	2	1649.68*M*	1586.96*M*	1582.29*M*	1463.42*M*
$(2,3,5)NAF_{1,2,0}$	2	1569.78*M*	1469.11*M*	1464.24*M*	-
NAF_4	3	1462.99*M*	1425.14*M*	1418.86*M*	1589.21*M*
$(2,3)NAF_4$	3	1441.27*M*	1400.07*M*	1394.74*M*	1512.03*M*
$(2,3,5)NAF_4$	3	1492.90*M*	1423.67*M*	1395.49*M*	-
$(2,3,5,7)NAF_4$	3	1511.16*M*	1442.64*M*	1405.78*M*	-
$(2,3)NAF_{3,1}$	3	1457.42*M*	1410.45*M*	1405.55*M*	1447.42*M*
$(2,3,5)NAF_{3,1,0}$	3	1505.10*M*	1410.68*M*	1406.15*M*	-
$(2,3,5)NAF_{1,1,1}$	3	1765.70*M*	1479.58*M*	1475.28*M*	-
$(2,3)NAF_{2,2}$	5	1468.65*M*	1416.67*M*	1412.18*M*	1387.49*M*
$(2,3,5)NAF_{2,2,0}$	5	1514.34*M*	1414.66*M*	1410.54*M*	-

NAF_5	7	1404.61*M*	1367.92*M*	1362.69*M*	1530.50*M*
$(2,3)NAF_5$	7	1397.00*M*	1357.65*M*	1353.00*M*	1482.04*M*
$(2,3,5)NAF_5$	7	1445.12*M*	1360.15*M*	1355.86*M*	-
$(2,3,5,7)NAF_5$	7	1482.62*M*	1371.53*M*	1367.50*M*	-
$(2,3)NAF_{4,1}$	7	1409.92*M*	1365.15*M*	1360.92*M*	1418.07*M*
$(2,3,5)NAF_{4,1,0}$	7	1459.32*M*	1366.78*M*	1362.89*M*	-
$(2,3,5)NAF_{2,1,1}$	7	1685.17*M*	1430.20*M*	1426.36*M*	-
$(2,3)NAF_{1,3}$	8	1477.40*M*	1421.24*M*	1417.11*M*	1336.61*M*
$(2,3)NAF_{0,3}$	8	1526.92*M*	1467.34*M*	1461.96*M*	1350.39*M*
$(2,3,5)NAF_{1,3,0}$	8	1517.43*M*	1419.41*M*	1415.57*M*	-
$(2,3,5)NAF_{0,3,0}$	8	1567.43*M*	1461.63*M*	1456.69*M*	-
$(2,3)NAF_{3,2}$	11	1422.24*M*	1372.98*M*	1369.07*M*	1367.29*M*
$(2,3,5)NAF_{3,2,0}$	11	1462.75*M*	1374.89*M*	1371.23*M*	-
NAF_6	15	1361.86*M*	1326.01*M*	1321.55*M*	1487.38*M*
$(2,3)NAF_6$	15	1357.80*M*	1319.37*M*	1315.38*M*	1444.40*M*
$(2,3,5)NAF_6$	15	1405.65*M*	1324.39*M*	1320.69*M*	-
$(2,3,5,7)NAF_6$	15	1439.06*M*	1335.21*M*	1331.719*M*	-

This table details the results of tests with 10000 160-bit scalars chosen randomly. We compare NAF and *w*NAF with proposed *mb*NAF, *wmb*NAF and *extended wmb*NAF assuming $1S = 0.8M$ for two cases: standard curves (2.2) and special curves [DIK06]. We fairly classify scalar multiplications according to the number of required pre-computations, which (for the case of our multibase methods) has been determined by using (7.3), (7.5) and (7.7) and confirmed by test results.

In the case of standard curves, it can be noticed the improvement when applying the methodology of replacing multiplications for squarings ("Fast" case) and when using new DA operation on top of the latter ("Fast using DA" case).

F2: TEST RESULTS FOR SSCA-PROTECTED IMPLEMENTATIONS

Method	Points	Traditional *M-A-N-A*	(1)	(2)	(1) with DA	(2) with DA	(3) $1S = 0.8M$
NAF	0	2383.91*M*	2034.84*M*	2045.33*M*	1976.79*M*	1931.87*M*	1855.07*M*
(2,3)NAF	0	2204.23*M*	1892.04*M*	1889.53*M*	1850.60*M*	1808.54*M*	1814.52*M*
(2,3,5)NAF	0	2277.11*M*	1855.82*M*	1848.02*M*	1820.64*M*	1779.26*M*	-
(2,3,5,7)NAF	0	2309.60*M*	1849.43*M*	1844.38*M*	1817.88*M*	1782.71*M*	-
$(2,3)NAF_{1,1}$	0	2188.78*M*	1909.75*M*	1905.07*M*	1870.14*M*	1827.64*M*	1880.61*M*
$(2,3,5)NAF_{1,1,0}$	0	2258.11*M*	1873.31*M*	1864.06*M*	1839.20*M*	1797.40*M*	-
NAF_3	1	2217.85*M*	1869.71*M*	1869.61*M*	1826.34*M*	1784.83*M*	1695.70*M*
$(2,3)NAF_3$	1	2163.34*M*	1831.12*M*	1826.45*M*	1793.32*M*	1752.56*M*	1725.21*M*
$(2,3,5)NAF_3$	1	2234.27*M*	1804.59*M*	1795.38*M*	1772.04*M*	1731.77*M*	-
$(2,3,5,7)NAF_3$	1	2267.87*M*	1803.24*M*	1796.66*M*	1773.80*M*	1739.11*M*	-
$(2,3)NAF_{2,1}$	1	2101.81*M*	1810.46*M*	1800.74*M*	1778.30*M*	1737.89*M*	1763.37*M*
$(2,3,5)NAF_{2,1,0}$	1	2146.81*M*	1804.09*M*	1792.49*M*	1774.00*M*	1733.68*M*	-
$(2,3)NAF_{1,2}$	2	2066.64*M*	1804.15*M*	1791.42*M*	1775.22*M*	1734.88*M*	1798.55*M*
$(2,3)NAF_{0,2}$	2	2172.13*M*	1932.84*M*	1925.52*M*	1907.13*M*	1875.28*M*	1961.98*M*
$(2,3,5)NAF_{1,2,0}$	2	2119.73*M*	1796.13*M*	1781.48*M*	1769.36*M*	1729.15*M*	-
NAF_4	3	2115.99*M*	1768.88*M*	1762.48*M*	1734.30*M*	1694.88*M*	1598.53*M*
$(2,3)NAF_4$	3	2062.09*M*	1734.16*M*	1723.39*M*	1704.85*M*	1666.10*M*	1633.47*M*
$(2,3,5)NAF_4$	3	2119.82*M*	1729.74*M*	1716.64*M*	1702.91*M*	1664.20*M*	-
$(2,3,5,7)NAF_4$	3	2154.31*M*	1737.93*M*	1726.84*M*	1712.67*M*	1677.47*M*	-
$(2,3)NAF_{3,1}$	3	2038.35*M*	1739.06*M*	1725.91*M*	1712.09*M*	1673.17*M*	1679.77*M*
$(2,3,5)NAF_{3,1,0}$	3	2095.23*M*	1735.50*M*	1720.40*M*	1710.59*M*	1671.72*M*	-
$(2,3,5)NAF_{1,1,1}$	3	2381.06*M*	1797.98*M*	1780.20*M*	1774.37*M*	1734.04*M*	-
$(2,3)NAF_{2,2}$	5	2013.68*M*	1739.87*M*	1724.48*M*	1715.15*M*	1676.17*M*	1717.12*M*
$(2,3,5)NAF_{2,2}$	5	2071.12*M*	1734.08*M*	1716.86*M*	1711.38*M*	1672.49*M*	-
NAF_5	7	2046.99*M*	1700.79*M*	1690.21*M*	1672.06*M*	1634.06*M*	1532.98*M*

(2,3)NAF_5	7	2017.57M	1685.04M	1671.70M	1659.50M	1621.79M	1577.50M
(2,3,5)NAF_5	7	2070.81M	1683.65M	1668.45M	1660.04M	1622.31M	-
(2,3,5,7)NAF_5	7	2107.49M	1693.22M	1680.14M	1671.08M	1636.87M	-
(2,3)$NAF_{4,1}$	7	1991.55M	1686.78M	1671.18M	1663.52M	1625.71M	1618.76M
(2,3,5)$NAF_{4,1,0}$	7	2049.33M	1684.67M	1667.29M	1663.27M	1625.46M	-
(2,3,5)$NAF_{2,1,1}$	7	2296.73M	1743.39M	1724.00M	1722.69M	1683.54M	-
(2,3)$NAF_{1,3}$	8	1992.20M	1739.77M	1722.47M	1717.01M	1677.99M	1747.66M
(2,3)$NAF_{0,3}$	8	2032.05M	1791.37M	1775.90M	1765.55M	1725.43M	1817.02M
(2,3,5)$NAF_{1,3,0}$	8	2044.69M	1734.97M	1716.19M	1713.84M	1674.89M	-
(2,3)$NAF_{3,2}$	11	1972.35M	1690.23M	1672.85M	1668.70M	1630.78M	1654.60M
(2,3,5)$NAF_{3,2,0}$	11	2022.04M	1689.61M	1670.89M	1669.46M	1631.52M	-
NAF_6	15	1996.04M	1650.85M	1637.33M	1626.30M	1589.34M	1485.02M
(2,3)NAF_6	15	1971.49M	1639.53M	1623.71M	1617.59M	1580.83M	1531.75M
(2,3,5)NAF_6	15	2024.64M	1641.41M	1624.03M	1621.03M	1584.19M	-
(2,3,5,7)NAF_6	15	2057.36M	1650.82M	1635.45M	1631.53M	1597.77M	-

This table details results from tests with 10000 160-bit scalars chosen randomly and protected against SSCA using atomicity. We compare NAF and *w*NAF with proposed *mb*NAF, *wmb*NAF and *extended wmb*NAF when using standard curves.

Six cases are considered for standard curves over prime fields (2.2):

- Using point operations with traditional *M-A-N-A* structure ("Traditional *M-A-N-A*" case). Costs for point doubling, addition and tripling are taken from Table 5.3. Since there are no previous proposals for quintupling and septupling, they are built by using elementary point operations as described in Section 5.3.1. Costs for those operations are taken from Table 5.1 (previous work).
- Using improved *M-A-N-A*-based formulae as detailed in Sections 5.2 and 5.3.2 ("(1)" case). Costs of atomic doubling, tripling and addition are taken from Table 5.3. Table 5.1 shows computing costs for quintupling and septupling.

- Using new *M-N-A-M-N-A-A*-based formulae introduced in Section 5.3 ("(2)" case). Similarly, costs for atomic doubling, tripling and addition are taken from Table 5.3, and costs of septupling and quintupling from Table 5.1.
- Similar to the second case, but this time every doubling followed by an addition is replaced by an efficient *M-A-N-A*-based atomic DA operation presented in Section 5.3.2, following approach discussed in Section 3.4.2.
- Similar to the third case, but this time every doubling followed by an addition is replaced by an *M-N-A-M-N-A-A*-based atomic DA (see Section 5.3.2), following approach discussed in Section 3.4.2.
- Using new *S-N-A-M-N-A-A*-based formulae introduced in Section 5.4 ("(3)" case). Costs for atomic doubling, tripling and addition are taken from Table 5.3.

For most cases, we assume $1S = 1M$, with exception of the last column corresponding to *S-N-A-M-N-A-A*-based formulae, which show the lowest computing costs for applications with ratio $S/M = 0.8$.

Similarly to Appendix F1, we fairly classify scalar multiplications according to the number of required pre-computations, which has been determined by using (7.3), (7.5) and (7.7) and confirmed with test results.

Bibliography

[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.

[ADD+06] R, Avanzi, V. Dimitrov, C. Doche and F. Sica, "Extending Scalar Multiplication Using Double Bases," Proc. *AsiaCrypt'06*, LNCS Vol. 4284, pp. 130-144, 2006.

[AHK+01] K. Aoki, F. Hoshino, T. Kobayashi and H. Oguro, "Elliptic Curve Arithmetic Using SIMD," *ISC2001*, Vol. 2200 of Lecture Notes in Computer Science, pp.235-247, Springer-Verlag, 2001.

[AS06] R. Avanzi and F. Sica, "Scalar Multiplication on Koblitz Curves using Double Bases," *Technical Report Number 2006/067*, Cryptology ePrint Archive, 2006.

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

[Ava05] R. Avanzi, "Side Channel Attacks on Implementations of Curve-Based Cryptographic Primitives," *Cryptology ePrint Archive*, Report 2005/017, 2005. Available at: http://eprint.iacr.org/

[Ber] D. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," available at: http://cr.yp.to/talks.html

[Ber06] D. Bernstein, "High-Speed Diffie-Hellman, Part 2," presentation in *INDOCRYPT'06*, tutorial session, 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* , Vol. 2020 of Lecture Notes in Computer Science, pp. 250-265, Springer-Verlag, 2001.

[BJ02] O. Billet and M. Joye, "The Jacobi Model of an Elliptic Curve and Side-Channel Analysis," *Cryptology ePrint Archive*, Report 2002/125, 2002. Available at: http://eprint.iacr.org/2002/125/

[BJ03] O. Billet and M. Joye, "Fast Point Multiplication on Elliptic Curves through Isogenies," *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," *IEEE Transactions on Computers*, 53(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.

[CMO98] H. Cohen, A. Miyaji and T. Ono, "Efficient Elliptic Curve Exponentiation using Mixed Coordinates," *Advances in Cryptology – ASIACRYPT '98*, Vol. 1514 of Lecture Notes in Computer Science, pp. 51–65, Springer-Verlag, 1998.

[Cor99] J.S. Coron, "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Springer-Verlag, pp. 292-302, 1999.

[CPQ01] M. Ciet, G. Piret and J. Quisquater, "Several Optimizations for Elliptic Curves Implementation on Smart Card," *Technical Report*, UCL Crypto Group, 2001.

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

[DIK06] C. Doche, T. Icart and D. Kohel, "Efficient Scalar Multiplication by Isogeny Decompositions," in Proc. *PKC 2006*, LNCS 3958, 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*, Vol. 3788 of Lecture Notes in Computer Science, pp. 59–78, Springer-Verlag, 2005.

[DJM98] V. S. Dimitrov, G. A. Jullien, and W. C. Miller, "An Algorithm for Modular Exponentiation," *Information Processing Letters*, 66(3):155–159, 1998.

[ELM03] K. Eisentraeger, K. Lauter and P. Montgomery, "Fast Elliptic Curve Arithmetic and Improved Weil Pairing Evaluation," in *Topics in Cryptology – CT-RSA'2003*, Vol. 2612 of Lecture Notes in Computer Science, pp. 343–354, Springer-Verlag, 2003.

[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.

[FGK+02] W. Fischer, C. Giraud, E.W. Knudsen and J.-P. Seifert, "Parallel Scalar Multiplication on General Elliptic Curves over Fp Hedged against Non-Differential Side-Channel Attacks," in *IACR ePrint archive,* Report 2002/007, 2002. Available at: http://www.iacr.org

[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)*, Vol. 3659 of Lecture Notes in Computer Science, Springer-Verlag, pp. 75-90, 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)*, Vol. 2523 of Lecture Notes in Computer Science, Springer-Verlag, pp. 114-128, 2003.

[Gor98] D. Gordon, "A Survey of Fast Exponentiation Methods," *Journal of Algorithms*, Vol. 27, pp. 129-146, 1998.

[GPW+04] N. Gura, A. Patel, A. Wander, H. Eberle and S.C. Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES'04)*, Vol. 3156 of Lecture Notes in Computer Science, Springer-Verlag, pp. 119-132, 2004.

[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.

[IT02] T. Izu and T. Takagi, "Fast Elliptic Curve Multiplications with SIMD Operations," in *4th International Conference on Information and Communications Security (ICICS'02)*, Vol. 2513 of Lecture Notes in Computer Science, pp. 217-230, Springer-Verlag, 2002.

[IT02b] T. Izu and T. Takagi, "A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks," in *Public Key Cryptography (PKC '02)*, Vol. 2274 of Lecture Notes in Computer Science, pp. 280-296, Springer-Verlag, 2002.

[IT05] T. Izu and T. Takagi, "Fast Elliptic Curve Multiplications Resistant against Side Channel Attacks," in *IEICE Trans. Fundamentals*, Vol. E88-A, No. 1, pp. 161–171, 2005.

[JY02] M. Joye and S. -M. Yen, "New Minimal Modified Radix-r Representation with Applications to Smart Cards," *PKC 2002*, LNCS 2274, pp. 375-384, 2002.

[Koc96] P. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems,” in *Proc. CRYPTO’96*, LNCS Vol. 1109, pp. 104–113, 1996.

[KJJ99] C. Kocher, J. Jaffe and B. Jun, “Differential Power Analysis,” *Advances in Cryptology - CRYPTO ’99*, LNCS Vol. 1666, pp. 388-397, 1999.

[LH00] C.H. Lim, and H.S. Hwang, “Fast implementation of Elliptic Curve Arithmetic in GF(p^m),” in Proc. *PKC’00*, LNCS Vol. 1751, pp. 405-421, Springer-Verlag, 2000.

[LS01] P.Y. Liardet and N.P. Smart, “Preventing SPA/DPA in ECC systems using the Jacobi form,” in *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001)*, Vol. 2162 of Lecture Notes in Computer Science, pp. 401-411, Springer-Verlag, 2001.

[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,” *IEEE Transactions on Computers*, Vol. 25, No. 8, pp. 1000-1010, 2006.

[NIST] FIPS PUB 186-2. Digital Signature Standard (DSS). *National Institute of Standards and Technology (NIST)*, 2000.

[Sma01] N. P. Smart, “The Hessian Form of an Elliptic Curve,” in *Workshop on Cryptographic Hardware and Embedded Systems (CHES’01)*, Vol. 2162 of Lecture Notes in Computer Science, pp. 118-125, Springer-Verlag, 2001.

[Sol99] J. Solinas, “ Generalized Mersenne Numbers,” *Technical Report CORR-99-39*, Dept. of C&O, University of Waterloo, 1999.

[TYW04] T. Takagi, S-M. Yen and B-C. Wu, “ Radix-r Non-Adjacent Form,” *ISC 2004*, LNCS Vol. 3225, pp. 99-110, Springer-Verlag, 2004.

[Wal01] C.D. Walter, “Sliding Windows succumbs to Big Mac Attack,” in *Workshop on Cryptographic Hardware and Embedded Systems (CHES’01)*, Vol. 2162 of Lecture Notes in Computer Science, pp. 286-299, Springer-Verlag, 2001.

[Woo01] A. Woodbury, “Efficient Algorithms for Elliptic Curve Cryptosystems on Embedded Systems,” *MSc. Thesis*, Worcester Polytechnic Institute, 2001.

[WT01] C.D. Walter and S. Thompson, “Distinguishing Exponent Digits by Observing Modular Subtractions,” in *Topics in Cryptology − CT-RSA 2001*, Vol. 2020 of Lecture Notes in Computer Science, pp. 192-207, Springer-Verlag, 2001.