Applied Mathematics and Computation 334 (2018) 41–59

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Fast elliptic curve point multiplication based on window Non-Adjacent Form method

Denis Khleborodov

Mathematical Studies in Information Security Section, Lomonosov Moscow State University, Moscow, Russia

ARTICLE INFO

Keywords:
Elliptic curves
Scalar multiplication
Lightweight cryptography
Sliding window NAF

ABSTRACT

This article presents window Non-Adjacent Form (*w*NAF) of scalar representation method for developing algorithms of computing scalar multiplication in groups of points on an elliptic curve over finite fields. A new efficient *w*NAF-based algorithm has been presented. The algorithm was developed based on simple and composite operations with a point and also based on affine and Jacobi coordinates systems taking into account the latest achievements in computing cost reduction. The theorem concerning its computational complexity is formulated and proved for this new algorithm. In the end of this article, an efficiency analysis of the proposed algorithm depending on various parameters is presented, namely, the characteristic of the field over which the curve is defined, the scalar length and the window width of the precomputations and the coordinates of the precomputed points. A comparative analysis of the new algorithm and previously known efficient algorithm is also presented.

© 2018 Elsevier Inc. All rights reserved.

1. Introduction

Elliptic Curve Cryptography (ECC), independently introduced by Miller [1] and Koblitz [2] in 1980s, is a promising technology for new cryptographic applications. ECC can achieve high security levels as that of RSA, but with much smaller key sizes and possibly faster computation, which result in lower power consumption as well as better memory and bandwidth savings (see Table 1 [3]).

The main advantage of ECC is the possibility to use short keys and to provide a high security level at the same time. A significant feature of the approach used in ECC, in comparison with other approaches used in FFC (Finite Field Cryptography) and IFC (Integer Factorization Cryptography), is the ability to provide the same security level with significantly shorter keys (Table 1). It allows reduced CPU performance, power consumption and key storage requirements for hardware platforms. Memory is not a scarce resource today, however, memory usually is usually limited at IoT (Internet of Things) devices, smart cards and SIM (Subscriber Identification Module) cards due to the desire to reduce linear dimensions of such devices. Elliptic curve based transformations are also widely represented in security standards and security regulatory requirements (FIPS PUB 186 [4], ANSI X9.62 [5], SECG [6]).

Constraints

Despite the significant progress of extensive research in this field, an interest in obtaining new results related to the acceleration of ECC transformations remains undying due to the practical significance of the problems outlined above. A

E-mail address: dkhleborodov@gmail.com

https://doi.org/10.1016/j.amc.2018.03.112
0096-3003/© 2018 Elsevier Inc. All rights reserved.

Table 1
Relation of security levels (SL) in symmetric (SC) and asymmetric (FFC, IFC, ECC) cryptosystems.

SL	SC	FFC (DSA, DH), *bit*	IFC (RSA), *bit*	ECC (ECDS), *f bit*
80	2TDEA	$L = 1024,\ N = 160$	1024	160...223
112	3TDEA	$L = 2048,\ N = 224$	2048	224...255
128	AES-128	$L = 3072,\ N = 256$	3072	256...383
192	AES-192	$L = 7680,\ N = 384$	7680	384...511
256	AES-256	$L = 15360,\ N = 512$	15360	512+

significant drawback of transformations based on elliptic curves is a high computational complexity. This shortcoming can be reduced by applying new effective algorithms.

Two classes can be distinguished among all algorithms for computing the main operation in ECC, i.e., algorithms with and without precomputations. Algorithms from the first class are associated with the idea of separating some calculations from the main part of the algorithm. Such calculations can be performed only once, and their results are then used in subsequent calculations on the elliptic curve in the main part of algorithm. Such an approach can significantly reduce the entire computing complexity, but at the stage of precomputations, additional memory is needed to store the results.

Contribution

Despite the fact that operations with a point in different coordinate systems are well studied and presented in various sources, it is possible to develop algorithms for computing transformations based on elliptic curves, relying not only on the complexity of such operations, but also taking into account the following factors.

1. Properties of the scalar representation allow the following.
   (a) The use of various volumes of precomputations (a different window w size) to reduce computing complexity of the main stage.
   (b) The use of various coordinates to represent precomputed points.
   (c) Joining operations with a point into composite operations and to use them at the main stages of the algorithm (DA is "double and addition" operation, T is tripling operation). Composite operations have less computational complexity than the sequential application of simple operations with a point (ADD is point addition, DBL is point doubling).
   (d) The use of special composite operations at various stages of the algorithm, which reduces the overall computational complexity.
2. The use of mixed coordinate systems and point representations.
3. Various ratios of basic operations in finite field: multiplication (M), squaring (S) and multiplicative inverse (I) lookup.

For algorithms that use pre-computations, factors such as the volume of pre-computations and the coordinate type of pre-computed points have a significant influence on the overall complexity.

Thus, a consideration of the factors described above will allow one to obtain more efficient computational algorithms on an elliptical curve than algorithms that take into account only the computational complexity of simple operations with a point in a single coordinate system.

In this article based on the statements already reviewed the following new results are obtained. These conclusions are all new and have not been presented in the literature.

1. A new fast algorithm based on window NAF scalar representation method with the use of affine and Jacobi coordinate systems is proposed.
2. A theorem on the average computational complexity of proposed algorithm is formulated and proved.
3. An analysis of the computational complexity of the proposed algorithm with various parameter values (the characteristic of the field over which the curve is defined and the scalar lengths, the volume of precomputations, the ratio of the computational complexity of the basic operations in the field) is conducted.
4. For fields with various characteristics and scalar length, scenarios of the proposed algorithm providing the least total computational complexity is presented.
5. The results of the comparative analysis of the proposed algorithm and the previously results are given.

The main definitions and related results are presented in the section below.

2. Background

This section presents the main definitions that will be used for further analysis.

Definition 2.1 (An elliptic curve over field K in general case)**.** Let K be a field, K its algebraic closure. An elliptic curve E over K is denoted by E/K and is given by the Weierstrass equation

$$E(K): y^2 \oplus a_1xy \oplus a_3y = x^3 \oplus a_2x^2 \oplus a_4x \oplus a_6, \tag{2.1}$$

where $\{a_1, a_2, a_3, a_4, a_6\} \in K$, $\oplus$, $\cdot$ are field operations and for each point (x, y) with coordinates in K satisfying 2.1 and the partial derivatives $2y \oplus a_1x \oplus a_3$ and $3x^2 \oplus 2a_2x \oplus a_4 \ominus a_1y$ do not vanish simultaneously at that point.

The criterion that both partial derivatives do not vanish simultaneously means that the curve is non-singular. A method for testing a curve for non-singularity involves a quantity called the discriminant of the curve. The discriminant of the curve E/K denoted by Δ is defined to be

$$\begin{aligned}
\Delta &= \ominus d_2^2 d_8 \ominus 8d_4^3 \ominus 27d_6^2 \oplus 9d_2d_4d_6,\\
d_2 &= a_1^2 \oplus 4a_2,\\
d_4 &= 2a_4 \oplus a_1a_3,\\
d_6 &= a_3^2 \oplus 4a_6,\\
d_8 &= a_1^2a_6 \ominus a_1a_3a_4 \oplus a_2a_3^2 \ominus a_4^2.
\end{aligned} \tag{2.2}$$

The condition $\Delta \neq 0$ (2.2) ensures that the elliptic curve E/K is smooth. It means that there are no points at which the curve has two or more distinct tangent lines. The properties of the elliptic curve depend on field and characteristics of Eq. (2.1).

Definition 2.2 (An elliptic curve over a prime field)**.** Let $\mathbb{F}_p$ is a prime field and for $a, b \in \mathbb{F}_p$ the inequality $4a^3 + 27b^2 \not\equiv 0 \pmod p$ is satisfied. Then an elliptic curve $E(\mathbb{F}_p)$ over $\mathbb{F}_p$ defined by the parameters $a, b \in \mathbb{F}_p$ consists of the set of solutions or points $P = (x, y)$, $x, y \in \mathbb{F}_p$ of the following equation:

$$E(\mathbb{F}_p): y^2 \equiv x^3 + ax + b \pmod p, \tag{2.3}$$

along with singular point $\mathcal{O}$ called the point at infinity. For the point $P = (x_P, y_P)$, x_P is the x-coordinate of the point P and the y_P is the y-coordinate of the point P. Such an interpretation of a point is called affine coordinates.

Eq. (2.3) is called the defining equation of $E(\mathbb{F}_p)$ (the short Weierstrass model). The short Weierstrass model is general in the sense that it can be used to describe all elliptic curves defined over large prime fields. This includes all such curves in standards as well as those currently under discussion for future standards. On the other hand, the twisted Edwards and Montgomery models can only be used for a subset of elliptic curves; these models cannot be used for prime order curves (i.e., all of the curves defined over prime fields in current standards). The advantage of prime order curves is that they are backwards compatible with implementations that support the most popular standardized curves.

Definition 2.3 (A group of points)**.** Consequently, the set of pairs (x, y) that solves (2.3), where $x, y \in \mathbb{F}_p$, and the point at infinity $\mathcal{O}$ form an abelian group $(E(\mathbb{F}_p), \boxplus)$, which ultimately contains all the possible elements for computations on ECC over prime fields:

$$(E(\mathbb{F}_p), \boxplus) = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : y^2 \equiv x^3 + ax + b \pmod p\} \cup \{\mathcal{O}\}.$$

It is possible to determine the law of addition of points on E in affine coordinates. The law of addition is defined as follows [6]:

1. The rule of the addition of the infinity point:

$$\mathcal{O} \boxplus \mathcal{O} = \mathcal{O}.$$

2. The addition of the infinity point to any another point:

$$(x, y) \boxplus \mathcal{O} = \mathcal{O} \boxplus (x, y) = (x, y), \forall (x, y) \in E(\mathbb{F}_p).$$

3. The rule of the addition of two points with the identical x-coordinates when the points are either different or have the y-coordinate equal to 0:

$$(x, y) \boxplus (x, \ominus y) = \mathcal{O}, \forall (x, y), (x, \ominus y) \in E(\mathbb{F}_p),$$

4. The rule of the addition of two points with the different x-coordinates: let $(x_1, y_1) \in E(\mathbb{F}_p)$ and $(x_2, y_2) \in E(\mathbb{F}_p)$ be the points for which $x_1 \neq x_2$. Then $(x_1, y_1) \boxplus (x_2, y_2) = (x_3, y_3)$, where:

$$\begin{aligned}
x_3 &\equiv \lambda^2 - x_1 - x_2 \pmod p,\\
y_3 &\equiv \lambda(x_1 - x_3) - y_1 \pmod p,\\
\lambda &\equiv \frac{y_2 - y_1}{x_2 - x_1} \pmod p.
\end{aligned}$$

5. The rule of the addition of a point to itself (point doubling): let $(x_1, y_1) \in E(\mathbb{F}_p)$ be a point and $y_1 \neq 0$. Then $(x_1, y_1) \boxplus (x_2, y_2) = (x_3, y_3)$, where:

$$x_3 \equiv \lambda^2 - 2x_1 \pmod{p},$$
$$y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p},$$
$$\lambda \equiv \frac{3x_1^2 + a}{2y_1} \pmod{p}.$$

Definition 2.4. (A group order) The number of points on the elliptic curve $E(\mathbb{F}_p)$ is denoted as $\#E(\mathbb{F}_p)$.

Definition 2.5 (Hasse–Weil bound)**.** Let $E(\mathbb{F}_p)$ be an elliptic curve. Then

$$p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p}.$$

Since $\sqrt{p}$ is small relative to p, we have $\#E(\mathbb{F}_p) \approx p$. An estimate for scalar d involved in scalar multiplication dP is as follows:

$$d \leq \#G \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p},$$

where $G = \langle P \rangle$, $G \subset E(\mathbb{F}_p)$ is the cyclic subgroup of $E(\mathbb{F}_p)$. Elliptic curves can be represented in several different ways. The standard projective model of an elliptic curve is useful for improving the efficiency of computation on elliptic curves, but was surpassed by another projective model introduced by Chudnovsky and Chudnovsky [7]. These projective coordinate systems lead to more efficient arithmetic and rely on a generalized definition of projective space.

Definition 2.6 (Scalar multiplication)**.** Basis of cryptographic operations founded on elliptic curves is scalar point multiplication. Given an integer d and the point $P \in E(\mathbb{F}_p)$, scalar multiplication is the process of the addition of the point P with itself d times. The result of scalar multiplication is denoted as by $dP = \underbrace{P \boxplus \ldots \boxplus P}_{d \text{ times}}$.

Elliptic curves can be represented in several different ways. The standard projective model of an elliptic curve is useful for improving the efficiency of computation on elliptic curves, but was surpassed by another projective model introduced by Chudnovsky and Chudnovsky [7]. These projective coordinate systems lead to more efficient arithmetic and rely on a generalized definition of projective space.

Definition 2.7 (Jacobi form)**.** For a field K, let us define an equivalence relation $K\cong_{(c,\,d)}$ on $K^3 \setminus \{(0, 0, 0)\}$ as follows:

$$(X_1, Y_1, Z_1) \cong (X_2, Y_2, Z_2) \text{ if } X_1 = \lambda^c X_2, Y_1 = \lambda^d Y_2, Z_1 = \lambda Z_2,$$

for some $\lambda \in K^*, c, d \in \mathbb{N}$.

We call the equivalence class containing a triple (X, Y, Z) a (c, d)-projective point and denote it as (X: Y: Z). The set of all such points

$$\{(X : Y : Z) | (X, Y, Z) \neq (0, 0, 0)\} / \langle \cong_{(c,d)} \rangle$$

is called (c, d)-projective 2-space.

If $Z \neq 0$, then a unique representative for (X: Y: Z) is (X/Z^c, Y/Z^d, 1). This gives a one-to-one correspondence between affine points and projective points with $Z \neq 0$.

With different values of c and d, the resulting equation is homogeneous with the following measure of degree:

$$\deg(X) = c, \deg(Y) = d \text{ and } \deg(Z) = 1.$$

To obtain the (c, d)-projective version of an affine curve $E : y_2 + a_1xy + a_3y = x_3 + a_2x_2 + a_4x + a_6$, substitute $x \mapsto X/Z^c$ and $y \mapsto X/Z^d$, then multiply through by the highest power of Z in the denominator.

If $3c = 2d$, the resulting (c, d)-projective curve is as follows:

$$E : Y_2 + a_1XYZ^{d-c} + a_3YZ^3 = X^3 + a_2X^2Z^c + a_4X^2Z^c + a_6Z^{3c}.$$

Solving for $Z = 0$ shows that there is the only (c, d)-projective point with $Z = 0$. This corresponds to the point at infinity in the affine coordinate system. To convert back to the affine representation, substitute 1 for Z. The affine coordinate system is denoted as $\mathcal{A}$.

The Jacobian projective coordinates are as follows: $c = 2$, $d = 3$. The Jacobian and Chudnovsky Jacobian projective coordinates were proposed by Chudnovsky and Chudnovsky [7]. In the Jacobian coordinates, point doubling can be performed in fewer finite field operations than in projective coordinates. Jacobian-based coordinate systems are mainly used for elliptic curves over prime fields. This coordinate system will be denoted as $\mathcal{J}$. In this system, 1 is represented by (1: 1: 0).

To reduce the computational complexity of scalar point multiplication on an elliptic curve algorithm, we will use composite operations.

Definition 2.8 (A composite operation)**.** Composite operations are operations that are based on a finite number of simpler point operations, such as doubling and addition. Composite operations include operations with form d_1P and $d_2P \boxplus Q$, where $d_1 > 2$ and $d_2 > 2$.

We will refer to the question of the number of non-zero values in scalar representation; therefore, we introduce the following notation for it.

Definition 2.9 (Hamming weight)**.** The function $\mathcal{H}(d)$, $d \in \mathbb{Z}^+$ denotes the number of non-zero values in the representation of scalar d.

Below, we present an overview of the known results associated with this study.

Related works

Since this study is related to conclusions obtained by using affine and Jacobi coordinates, the most important results for this study relating only to these coordinates will be mentioned. The computational complexity of the operation will be indicated, as well as the operation itself, but will be highlighted in bold.

Formulas for the point doubling operation with $2\mathcal{A} \longrightarrow \mathcal{J}$ structure were proposed in [8]. The results were derived from the doubling formula in [9] by applying $\lambda = 2y_1$. The cost was reduced by replacing the multiplication $4x_1y_1^2$ by one squaring and other less expensive operations. The operation has total cost $1\mathbf{M} + 5\mathbf{S}$.

Formulas for another point doubling operation with $2\mathcal{J} \longrightarrow \mathcal{J}$ structure were proposed in [10]. The results were obtained taking into account the fact that two multiplications could be directly replaced by a single squaring using the algebraic substitution (4.2 in [10]): $Z_3 = 2Y_1Z_1$ and $\beta = 4X_1Y_1^2$. Division by two is not necessary because these two terms already contain multiples of two. The total cost of the operation is $3\mathbf{M} + 5\mathbf{S}$.

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 the affine-Jacobian coordinates [11]. The mixed coordinates approach was proposed in [12]. It is important to note that in the case of doubling it has been suggested to fix parameter a in (2.3) to $a = -3$ for efficiency purposes. In fact, most curves recommended by public-key standards [13] use an $a = -3$, which has been shown not to impose significant restrictions on the cryptosystem [14].

Therefore, below in this study we will refer to the special case when $a = -3$ and the general case when this parameter is not fixed and can take on any value in the field.

Formulas for point tripling operation with $3\mathcal{A} \longrightarrow \mathcal{A}$ structure was introduced by Eisentrager et al. [15] and improved by Ciet et al. [16]. These formulas were used by Dimitrov et al. [17] to derive the following formula for tripling in the Jacobian coordinates. The formulas for $3\mathcal{A} \longrightarrow \mathcal{A}$ were proposed in [18]. The complexity of this operation is $7\mathbf{M} + 5\mathbf{S}$ or $\mathbf{S}$ when $2P = \mathcal{O}$ (the infinity point).

Formulas for "double and addition" operation with $\mathcal{J} \boxplus \mathcal{A} \boxplus \mathcal{J} \longrightarrow \mathcal{J}$ structure were introduced in [19] and [8]. The computing complexity of such operation is $11\mathbf{M} + 5\mathbf{S}$ [8]. This result was obtained due to the use of mixed addition [20] and addition with identical z-coordinates [21].

The formulas for the addition of points $\mathcal{J} \boxplus \mathcal{J} \longrightarrow \mathcal{J}$ structure were introduced by Longa [10]. The total cost of the operation is $11\mathbf{M} + 5\mathbf{S}$.

Properties of the wNAF scalar representation were analyzed by Moller [22]. An algorithm for converting a scalar to a wNAF representation form was proposed by Hankerson et al. [9, Alg. 3.35]. An algorithm for scalar point multiplication based on the wNAF method was proposed by Hankerson et al. [9, Alg. 3.36]. Another signed binary encoding, wMOF (window Mutual Opposite Form), was introduced by Okeya et al. [23].

Having introduced the basic definitions and the related results, we present the main results of this study related to the computation of scalar point multiplication on an elliptic curve, when neither point nor scalar is known in advance.

3. Scalar Point Multiplication

In this section, we present a new effective scalar point multiplication algorithm based on the wNAF scalar representation method. Theorem on their computational complexity and proof will also be given.

Definition 3.1 (window Non-Adjacent Form)**.** Let $w \geq 2$ be a positive integer. The width-w NAF of the positive integer d is described by the following expression

$$d = \sum_{i=0}^{l-1} k_i 2^i,$$

where each non-zero coefficient k_i is odd, $|k_i| \leq 2^{w-1}$, $k_{n-1} \neq 0$, and at most one of any w consecutive digits is nonzero. The length of the width-w NAF is l.

This representation (Definition 3.1) is a generalization of the binary NAF representation.

Properties of this representation are summarized in Theorem 3.2. These points were summarized by Muir and Stinson [24].

Theorem 3.2 (properties of the width-w NAF scalar representation)**.** $d \in \mathbb{Z}^+ \backslash \{0\}$, $d = \{d_{n-1}, \ldots, d_0\}_2$, $d_i \in \{0, 1\}$, $\forall i = \overline{0, n-1}$, $w \in \mathbb{Z}^+ \backslash \{0\}$ *then.*

(1) *The positive integer d has a unique width-w NAF representation denoted* $\mathrm{NAF}_w(d) = \{k_{l-1}, \ldots, k_0\}_{\mathrm{NAF}_w}$, $|k_i| \le 2^{w-1}$, $\forall i = \overline{0, l-1}$;
(2) $\mathrm{NAF}_2(d) = \mathrm{NAF}(d)$;
(3) *The* $\mathrm{NAF}(d)$ *representation has the fewest non-zero digits of any signed digit representation of d;*
(4) $l \le n + 1$;
(5) $\mathcal{H}(\mathrm{NAF}_w(d)) \approx \frac{l}{w+1}$.

Koyama and Tsuruoka introduced a NAF method in [25]. Moller [22] and Bosma [12] discuss methods to eliminate the length expansion of a window-w NAF expansion in around half the cases.

The representation $\mathrm{NAF}_w(d)$ can be efficiently computed using Algorithm 1 (algorithm was adapted by Hankerson et al. [9, Alg. 3.35], where $d \mod 2^w$ denotes the integer u satisfying the inequality $-2^{w-1} \le u \le 2^{w-1}$. The digits of $\mathrm{NAF}_w(d)$ are obtained by repeatedly dividing d by 2 allowing remainders in $[-2^{w-1}+1, 2^{w-1}-1]$ if d is odd and the remainder $d \mod 2^w$ is chosen, then $\frac{d-u}{2}$ will be divisible by 2^{w-1} ensuring that the next $w-1$ digits are zero.

Algorithm 1 Computing $\mathrm{NAF}(d)_w$, $d \in \mathbb{Z}^+ \backslash \{0\}$.

Input - Integer $d \in \mathbb{Z}^+ \backslash \{0\}$.
- Window $w \in \mathbb{Z}^+ \backslash \{0\}, w \ge 2$.
Output - Representation of d: $\mathrm{NAF}_w(d) = \{k_{l-1}, \ldots, k_0\}_{\mathrm{NAF}_w}$.
begin
1: $l \leftarrow 0$;
2: **while** $d \ge 1$ **do**
3: **if** $d \mod 2 \not\equiv 0$ **then**
4: $k_l \leftarrow 2 - (d \mod 2^w)$ (signed);
5: $d \leftarrow d - k_l$;
6: **else**
7: $k_l \leftarrow 0$;
8: **end if**
9: $d \leftarrow d/2$;
10: $l \leftarrow l + 1$;
11: **end while**
12: **return** $(k_{l-1}, \ldots, k_0)$**end**

The digit k_i of the set for this representation is the odd integer between -2^{w-1} and 2^{w-1}. If the points k_iP are precomputed for every positive k_i in the digit set , the binary algorithm can again be applied. The advantage is that in this case the number of non-zero digits and hence the number of additions needed are lower. This leads to Algorithm 2 for computing a scalar multiple of a point, given the wNAF of the multiplier adapted from Hankerson et al. [9, Alg. 3.36].

The affirmation regarding the computational complexity of Algorithm 2 is formulated in Theorem 3.3. All statements in the theorem are valid for $\mathbf{S}/\mathbf{M} = 0.8$ and $\mathbf{I}/\mathbf{M} = 80$. These ratios are the most frequently encountered in practice and are often cited elsewhere.

Theorem 3.3. *For Algorithm 2 of scalar point multiplication dP, where* $d \in Z^+ \backslash \{0\}$, $\mathrm{NAF}_w(d) = \{k_{l-1}, \ldots, k_0\}_{\mathrm{NAF}_w}$, $|k_i| \le 2^{w-1}$, $\forall i = \overline{0, l-1}$, *where at most one of any* $w \ge 2$ *consecutive digits is non-zero, the following items are true:*

1. *The main part of the algorithm can be performed with the least average computational complexity:*

$$\widehat{L}_{Alg.2}/m_2(l, w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + \left(\left(\frac{8}{w+1} + 3\right)l - 10\right)\mathbf{M} + \left(\left(\frac{2}{w+1} + 5\right)l - 6\right)\mathbf{S}$$

for all n and all width-w windows. In this case, precomputations can be performed with complexity:

$$\widehat{L}_{Alg.2/pre_2}(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + 9\left(2^{w-2} - 1\right)\mathbf{M} + \left(3 \cdot 2^{w-2} + 2\right)\mathbf{S}$$

or

$$\widehat{L}_{Alg.2/pre_3}(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + 9\left(2^{w-2} - 1\right)\mathbf{M} + \left(2 \cdot 2^{w-2} + 4\right)\mathbf{S}.$$

2. *The lowest total computational complexity is achieved with window width* $w = w_3 = 5$ *for* $n \in \{192, 224, 256\}$ *and* $w = w_4 = 6$ *for* $n \in \{384, 521\}$. *In this case, the computational complexity of precomputations is:*

$$\widehat{L}_{Alg.2/pre_3}(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + 9\left(2^{w-2} - 1\right)\mathbf{M} + \left(2 \cdot 2^{w-2} + 4\right)\mathbf{S}.$$

Computing complexity of the main part will be:

$$\widehat{L}_{Alg.2/m_2}(l, w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + \left(\left(\frac{8}{w+1} + 3\right)l - 10\right)\mathbf{M} + \left(\left(\frac{2}{w+1} + 5\right)l - 6\right)\mathbf{S}.$$

Algorithm 2 Point multiplication based on window NAF method.

Input - Elliptic curve $E(\mathbb{F}_p)$.
- Scalar d representation $\mathrm{NAF}_w(d) = \{k_{l-1}, \ldots, k_0\}_{\mathrm{NAF}_w}$, $d \in \mathbb{Z}^+ \backslash \{0\}$.
- Point $P \in E(\mathbb{F}_p)$ in $\mathcal{A}$ form.

Output - Result of scalar multiplication $dP \in E(\mathbb{F}_p)$ in $\mathcal{A}$ form.

begin
// Precomputations
1: $P_1 \leftarrow P$;
2: $P_2 \leftarrow \mathbf{dbl}(P)$;
3: **foreach**$(i = 3, 5, \ldots, 2^{w-1} - 1)$ {
4: $P_i \leftarrow \mathbf{add}(P_{i-2}, P_2)$;
5: }
// Main Computations
6: $Q \leftarrow P_{l-1}$;
7: $i \leftarrow n - 2$;
8: **while** $i \geq 0$ **do**
9: **if** $k_i = 1$ **then**
10: $Q \leftarrow \mathbf{da}(Q, P_{k_i})$;
11: **else**
12: **if** $k_i = -1$ **then**
13: $Q \leftarrow \mathbf{da}(Q, \boxminus P_{k_i})$;
14: **else**
15: $Q \leftarrow \mathbf{dbl}(Q)$;
16: **end if**
17: **end if**
18: $i \leftarrow i - 1$;
19: **end while**
20: **return** (Q)**end**

3. *The lowest total computation complexity is achieved for all n when windows $w > w_5$ ($w > 7$) with the complexity of the precomputations:*

$$\widehat{L}_{Alg.2/pre_1}(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + \left(5 \cdot 2^{w-2} - 4\right)\mathbf{M} + \left(2^{w-1} + 3\right)\mathbf{S}$$

and average computational complexity of the main part:

$$\widehat{L}_{Alg.2/m_1}(l, w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + \left(\left(3 - \frac{1}{w+1}\left(\frac{12}{2^w} - 11\right)\right)l + \frac{12}{2^w} - 13\right)\mathbf{M} + \left(\left(5 - \frac{1}{w+1}\left(\frac{12}{2^w} - 5\right)\right)l + \frac{12}{2^w} - 9\right)\mathbf{S}.$$

Proof. Since the overall computational complexity of the algorithm consists of the complexity of the precomputations and the complexity of the main part. We describe it in a general form by the following expression:

$$\widehat{L}_{Alg.2}(l, w, \mathbf{ADD}, \mathbf{DBL}, x\mathbf{SCALE}, \mathbf{SCALE}) = \widehat{L}_{Alg.2/pre}(w, \mathbf{ADD}, \mathbf{DBL}, x\mathbf{SCALE}) + \widehat{L}_{Alg.2/m}(l, \mathbf{ADD}, \mathbf{DBL}, \mathbf{SCALE}).$$

We will start by considering precomputations. A rough estimate of the complexity of the precomputation is:

$$\widehat{L}_{Alg.2/pre}(w, \mathbf{ADD}, \mathbf{DBL}) = \mathbf{DBL} + (2^{w-2} - 1)\mathbf{ADD}.$$

Let us explain this estimate. Consider the following calculation method. Since the

$$k_i \in K, \text{ where } K = \{1, 3, 5, \ldots, 2^{w-1} - 1\}.$$

Then will calculate the precomputation table as follows:

$$k_i P = 2P \boxplus \ldots \boxplus 2P \boxplus 2P \boxplus P. \tag{3.1}$$

All additions in (3.1) can be computed with the special addition with identical z-coordinate.

The result of the precomputation can be represented in affine or Jacobi coordinates. Obtaining a result in the Jacobi coordinates requires less computation than in affine coordinates, however, at the main stage of the algorithm, slower operations with a point will have to be used. Converting the results of precomputations from Jacobi coordinates to affine coordinates requires additional computations, since it includes computation of the multiplicative inverse in the prime field.

Next, we will show how the complexity of the precomputations and main computations correlates for various parameters n, the width of the window w, and the coordinates in which the result of the precomputation will be presented.

We split the precomputation stage into two steps. The first step will allow one to obtain results of calculating the table in Jacobi coordinates, and in the second step, to convert to affine coordinates. For the second step, we will show two conversion schemes. As a result, three ways of precomputations and estimating the complexity for each of them will be presented. The general scheme of coordinate transformations looks as follows:

$$Scenario_1 - pre_1 : m_1 \quad \mathcal{A} \xrightarrow{Step_1} \mathcal{J} \xrightarrow{Main_1} \mathcal{A}$$

$$Scenario_2 - pre_2 : m_2 \quad \mathcal{A} \xrightarrow{Step_1} \mathcal{J} \xrightarrow[(scheme_1)]{Step_2} \mathcal{A} \xrightarrow{Main_2} \mathcal{A}$$

$$Scenario_3 - pre_3 : m_2 \quad \mathcal{A} \xrightarrow{Step_1} \mathcal{J} \xrightarrow[(scheme_2)]{Step_2} \mathcal{A} \xrightarrow{Main_2} \mathcal{A}$$

Consider the **first step**, where we calculate the precomputations in the Jacobi coordinates.

Point P is assumed to be originally in $\mathcal{A}$. Thus, if we want to use the special addition, we should translate computations to $\mathcal{J}$.

By applying the mixed coordinates approach proposed in [11], we can compute the doubling $2P$ in $\mathcal{A}$ and yield the result in $\mathcal{J}$ as follows:

$$\begin{aligned} X_2 &= (3x_1^2 \oplus a) \ominus 2\alpha, \\ Y_2 &= (3x_1^2 \oplus a)(\alpha \ominus X_2) \ominus 8y_1^4, \\ Z_2 &= 2y_1, \end{aligned} \tag{3.2}$$

with $\alpha = 4x_1y_1^2 = 2[(x_1 \oplus y_1^2)^2 \ominus x_1^2 \ominus y_1^2]$, where the input and result are $P = (x_1, y_1)$ and $2P = (X_2, Y_2, Z_2)$, respectively.

Formula (3.2) is easily derived from the doubling formula in $\mathcal{A}$ [9] by applying $\lambda = 2y_1$, and has a cost of only $1\mathbf{M} + 5\mathbf{S}$. Note that we have reduced the cost of (3.2) by replacing the multiplication $4x_1y_1^2$ by one squaring and other cheaper operations.

Then, by fixing $\lambda = 2y_1$ we can assume the following equivalent point to P:

$$P^{(1)} = \left(X_1^{(1)}, Y_1^{(1)}, Z_1^{(1)}\right) = (4x_1y_1^2, 8y_1^4, 2y_1) \equiv (X_1, Y_1, Z_1),$$

which does not introduce extra costs since its coordinates have already been computed in (3.2). Following additions to compute digits k_i would be performed using (3.1) as follows.

On *1st stage*:

$$3P = 2P \boxplus P^{(1)} = (X_2, Y_2, Z_2) \boxplus \left(X_1^{(1)}, Y_1^{(1)}, Z_1^{(1)}\right) \equiv (X_3, Y_3, Z_3):$$

$$\begin{aligned} X_3 &= \left(Y_1^{(1)} \ominus Y_2\right)^2 \ominus \left(X_1^{(1)} \ominus X_2\right)^3 \ominus 2X_2\left(X_1^{(1)} \ominus X_2\right)^2, \\ Y_3 &= \left(Y_1^{(1)} \ominus Y_2\right)\left(X_2\left(X_1^{(1)} \ominus X_2\right)^2 \ominus X_3\right) \ominus Y_2\left(X_1^{(1)} \ominus X_2\right)^3, \\ Z_3 &= Z_2\left(X_1^{(1)} \ominus X_2\right). \end{aligned}$$

On *2nd stage*:

$$\begin{aligned} 2P^{(1)} &= \left(X_2^{(1)}, Y_2^{(1)}, Z_2^{(1)}\right) \\ &= \left(X_2\left(X_1^{(1)} \ominus X_2\right)^2, Y_2\left(X_1^{(1)} \ominus X_2\right)^3, Z_2\left(X_1^{(1)} \ominus X_2\right)\right) \\ &\equiv (X_2, Y_2, Z_2). \end{aligned}$$

$$5P = 2P^{(1)} \boxplus 3P = \left(X_2^{(1)}, Y_2^{(1)}, Z_2^{(1)}\right) \boxplus (X_3, Y_3, Z_3) = (X_4, Y_4, Z_4),$$

where:

$$\begin{aligned} X_4 &= \left(Y_3 \ominus Y_2^{(1)}\right)^2 \ominus \left(X_3 \ominus X_2^{(1)}\right)^3 \ominus 2X_2^{(1)}\left(X_2 \ominus X_2^{(1)}\right)^2, \\ Y_4 &= \left(Y_2 \ominus Y_2^{(1)}\right)\left(X_2^{(1)}\left(X_3 \ominus X_2^{(1)}\right)^2 \ominus X_4\right) \ominus Y_2^{(1)}\left(X_3 \ominus X_2^{(1)}\right)^3, \\ Z_4 &= Z_2^{(1)}\left(X_3 \ominus X_2^{(1)}\right). \\ A_4 &= \left(X_3 \ominus X_2^{(1)}\right), \\ B_4 &= \left(X_3 \ominus X_2^{(1)}\right)^2, \\ C_4 &= \left(X_3 \ominus X_2^{(1)}\right)^3. \\ &\dots \end{aligned}$$

On $\left(2^{w-2}-1\right)th$ *stage*:

$$\begin{aligned}
2P^{(2^{w-2}-2)} &= \left(X_2^{(2^{w-2}-2)}, Y_2^{(2^{w-2}-2)}, Z_2^{(2^{w-2}-2)}\right)\\
&= \left(X_2^{(2^{w-2}-3)}\left(X_{2^{w-2}-1} \ominus X_2^{(2^{w-2}-3)}\right)^2,\right.\\
&\quad Y_2^{(2^{w-2}-3)}\left(X_{2^{w-2}-1} \ominus X_2^{(2^{w-2}-3)}\right)^3,\\
&\quad \left. Z_2^{(2^{w-2}-3)}\left(X_{2^{w-2}-1} \ominus X_2^{(2^{w-2}-3)}\right)\right)\\
&\equiv \left(X_2^{(2^{w-2}-3)}, Y_2^{(2^{w-2}-3)}, Z_2^{(2^{w-2}-3)}\right).\\
(2^{w-1}-1)P &= 2P^{(2^{w-2}-2)} \boxplus (2^{w-1}-3)P\\
&= \left(X_2^{(2^{w-2}-3)}, Y_2^{(2^{w-2}-3)}, Z_2^{(2^{w-2}-3)}\right) \boxplus (X_{2^{w-2}}, Y_{2^{w-2}}, Z_{2^{w-2}}),
\end{aligned}$$

where:

$$\begin{aligned}
X_{2^{w-2}+1} &= \left(Y_{2^{w-2}} \ominus Y_2^{(2^{w-2}-2)}\right)^2 \ominus \left(X_{2^{w-2}} \ominus \ominus X_2^{(2^{w-2}-2)}\right)^3 \ominus 2X_2^{(2^{w-2}-2)}\left(X_{2^{w-2}} \ominus X_2^{(2^{w-2}-2)}\right)^2,\\
Y_{2^{w-2}+1} &= \left(Y_{2^{w-2}} \ominus Y_2^{(2^{w-2}-2)}\right)\left(X_2^{(2^{w-2}-2)}\left(X_{2^{w-2}} \ominus \ominus X_2^{(2^{w-2}-2)}\right)^2 \ominus X_{2^{w-2}}\right) \ominus Y_2^{(2^{w-2}-2)}\left(X_{2^{w-2}} \ominus X_2^{(2^{w-2}-2)}\right)^3,\\
Z_{2^{w-2}+1} &= Z_2^{(2^{w-2}-2)}\left(X_{2^{w-2}} \ominus X_2^{(2^{w-2}-2)}\right).\\
A_{2^{w-2}+1} &= \left(X_{2^{w-2}} \ominus X_2^{(2^{w-2}-2)}\right),\\
B_{2^{w-2}+1} &= \left(X_{2^{w-2}} \ominus X_2^{(2^{w-2}-2)}\right)^2,\\
C_{2^{w-2}+1} &= \left(X_{2^{w-2}} \ominus X_2^{(2^{w-2}-2)}\right)^3.
\end{aligned}$$

Values A_i and (B_i, C_i), for $i = 4$ to $2^{w-2}+1$, are stored for Schemes 1 and 2, respectively, and used in Step 2 to save some computations when converting points to $\mathcal{A}$.

Therefore, the total cost of pre_1 computations.

$$\begin{aligned}
\widehat{L}_{Alg.2/pre_1}(w, \mathbf{DBL}_1, m\mathbf{ADD}) &= \mathbf{DBL}_1 + m\mathbf{ADD} + (2^{w-2}-2)m\mathbf{ADD}.\\
\widehat{L}_{Alg.2/pre_1}(w, \mathbf{DBL}_1, m\mathbf{ADD}) &= \mathbf{DBL}_1 + m\mathbf{ADD} + (2^{w-2}-2)m\mathbf{ADD}.\\
\widehat{L}_{Alg.2/pre_1}(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) &= (1\mathbf{M}+5\mathbf{S}) + (5\mathbf{M}+2\mathbf{S}) + (5\mathbf{M}+2\mathbf{S})(2^{w-2}-2)\\
&= (5\cdot 2^{w-2}-4)\mathbf{M} + (2^{w-1}+3)\mathbf{S}.
\end{aligned}$$

The second step of precomputation is the conversion of points from the Jacobi coordinates to affine coordinates. Since the complexity of the basic calculations depends on the coordinates in which the results of the precomputation are presented, it is necessary to investigate in detail all the possibilities of this step. The end result will depend on many other parameters, which we will establish below. Next, sets of parameters will be established, under which the theoretical complexity of the calculations will be minimal. In the practical implementation of these algorithms, the calculation method can be selected dynamically based on the input data and available memory.

In the **second step**, we will consider the scaling schemes of the pre-calculated points.

Points (X_i, Y_i, Z_i) from Step 1, for i from 3 to $2^{w-2}+1$, have to be converted back to $\mathcal{A}$ since this would allow the use of the efficient mixed addition during the scalar multiplication. This can be achieved by means of the following:

$$\left(X_i(Z_i^2), Y_i(Z_i^3), 1\right). \tag{3.3}$$

To avoid the computation of several expensive inversions when using (3.3) for each point in the case $w > 2$, we use the method due to Montgomery, called simultaneous inversion [9], to limit the requirement to only one inversion.

In Scheme 1, we first compute the inverse $r = Z_{2^{w-2}+1}^{-1}$, and then recover every point using (3.3) as follows:

For point $(2^{w-1}-1)P$:

$$r = Z_{2^{w-2}+1}^{-1},$$

$$x_{2^{w-2}+1} = r^2 X_{2^{w-2}+1},$$
$$y_{2^{w-2}+1} = r^3 Y_{2^{w-2}+1}.$$

For point $(2^{w-1}-3)P$:

$$r = rA_{2^{w-2}+1},$$
$$x_{2^{w-2}} = r^2 X_{2^{w-2}},$$
$$y_{2^{w-2}} = r^3 Y_{2^{w-2}}.$$

For point $3P$:

$$r = rA_4,$$
$$x_3 = r^2 X_3,$$
$$y_3 = r^3 Y_3.$$

It is important to observe that $Z_j = Z_3 \times \prod_{i=4}^{j} A_i$ for $j = 4$ to $(2^{w-1}+2)$, according to Step 1, and hence, for $i = (2^{w-2}-3)$ down to 3, $Z^{-1}_{\frac{i+3}{2}}$ for each point iP is recovered at every multiplication $rA_{\frac{i+5}{2}}$.

In total, Scheme 1 has the following cost when computing the precomputed table:

$$\widehat{L}_{Alg.2/pre_2}(w, \mathbf{DBL}_1, m\mathbf{ADD}, x\mathbf{SCALE}_1) = \mathbf{DBL}_1 + m\mathbf{ADD} + (2^{w-2}-2)m\mathbf{ADD} + x\mathbf{SCALE}_1.$$

Since

$$x\mathbf{SCALE}_1(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + (3\mathbf{M}+1\mathbf{S}) + (4\mathbf{M}+1\mathbf{S})(2^{w-2}-2),$$

then

$$\widehat{L}_{Alg.2/pre_2}(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = (1\mathbf{M}+5\mathbf{S}) + (5\mathbf{M}+2\mathbf{S}) + (2^{w-2}-2)(5\mathbf{M}+2\mathbf{S}) + \mathbf{I} + (3\mathbf{M}+1\mathbf{S}) + (4\mathbf{M}+1\mathbf{S})(2^{w-2}-2).$$

$$\widehat{L}_{Alg.2/pre_2}(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + 9(2^{w-2}-2)\mathbf{M} + (3\cdot 2^{w-2}+2)\mathbf{S}.$$

In Scheme 2, we first compute $r_1 = (Z^{-1}_{2^{w-2}+1})^2$ and $r_2 = (Z^{-1}_{2^{w-2}+1})^3$, and then recover every point as follows:

For point $(2^{w-1}-1)P$:

$$x_{2^{w-2}+1} = r_1 X_{2^{w-2}+1},$$
$$y_{2^{w-2}+1} = r_2 Y_{2^{w-2}+1}.$$

For point $(2^{w-1}-3)P$:

$$r_1 = r_1 B_{2^{w-2}+1},$$
$$r_2 = r_2 C_{2^{w-2}+1},$$
$$x_{2^{w-2}} = r_1 X_{2^{w-2}},$$
$$y_{2^{w-2}} = r_2 Y_{2^{w-2}}.$$

For point $3P$:

$$r_1 = r_1 B_4,$$
$$r_2 = r_2 C_4,$$
$$x_3 = r_1 X_3,$$
$$y_3 = r_2 Y_3.$$

In this case $Z^2_j = Z^2_3 \times \prod_{i=4}^{j} B_i$ and $Z^2_j = Z^3_3 \times \prod_{i=4}^{j} C_i$ for $j = 4$ to $(2^{w-2}+1)$, according to Step 1, and hence, for $j = (2^{w-2}-3)$ down to 3, the pair $(Z^{-2}_{\frac{i+3}{2}}, Z^{-3}_{\frac{i+3}{2}})$ for each point iP is recovered at every multiplication $r_1 B_{\frac{i+5}{2}}$ and $r_1 C_{\frac{i+5}{2}}$.

Where $2^{w-2}-1$ represents the number of points. In terms of memory usage, Scheme 1 requires $3(2^{w-2}-1)+3$ registers for temporary calculations and the storing of the precomputed points. We will show later that this requirement does not exceed the number of available registers for the scalar multiplication for practical values of $2^{w-2}-1$.

In the case of Scheme 2, the cost is as follows:

$$\widehat{L}_{Alg.2/pre_3}(w, \mathbf{DBL}_1, m\mathbf{ADD}, x\mathbf{SCALE}_2) = \mathbf{DBL}_1 + m\mathbf{ADD} + (2^{w-2}-2)m\mathbf{ADD} + x\mathbf{SCALE}_2.$$

Since

$$x\mathbf{SCALE}_2(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + (3\mathbf{M}+1\mathbf{S}) + 4\mathbf{M}(2^{w-2}-2),$$

then

$$\widehat{L}_{Alg.2/pre_3}(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = (1\mathbf{M}+5\mathbf{S}) + (5\mathbf{M}+2\mathbf{S}) + (2^{w-2}-2)(5\mathbf{M}+2\mathbf{S}) + \mathbf{I} + (3\mathbf{M}+1\mathbf{S}) + 4\mathbf{M}(2^{w-2}-2).$$

Table 2
Computing cost of precomputations (pre_i) in Algorithm 2.

Pre	Steps	Complexity $\widehat{L}_{pre_i}(w, \mathbf{I}, \mathbf{M}, \mathbf{S})$	In	Out
pre_1	St_1	$(5\cdot 2^{w-2}-4)\mathbf{M}+(2^{w-1}+3)\mathbf{S}$	$\mathcal{A}$	$\mathcal{J}$
pre_2	$St_1+St_2(Sch_1)$	$\mathbf{I}+9(2^{w-2}-2)\mathbf{M}+(3\cdot 2^{w-2}+2)\mathbf{S}$	$\mathcal{A}$	$\mathcal{A}$
pre_3	$St_1+St_2(Sch_2)$	$\mathbf{I}+9(2^{w-2}-1)\mathbf{M}+2(2^{w-2}+4)\mathbf{S}$	$\mathcal{A}$	$\mathcal{A}$

$$\widehat{L}_{Alg.2/pre_3}(w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I}+9(2^{w-2}-1)\mathbf{M}+2(2^{w-2}+4)\mathbf{S}.$$

All precomputation results are summarized in Table 2.

Next, we investigate the computational complexity of the main part of the algorithm.

According to Theorem 3.2 (5) the density of the non-zero width-w NAFs of length d is approximately $\frac{l}{w+1}$. This means that the general computing cost of main part of Algorithm 2 is approximately:

$$\widehat{L}_{Alg.2/main}(l, w, \mathbf{DBL}, \mathbf{ADD}) = \left(\frac{l}{w+1}-1\right)\mathbf{ADD}+(l-1)\mathbf{DBL}+\mathbf{SCALE}.$$

For the main computation, there are approximately $w+1$ times as many doublings as additions, therefore Jacobian coordinates are used for the temporary variable.

If the precomputed values are kept in Jacobian coordinates (precomputations pre_1), the double-and-add (for digits other than ± 1) will be performed with $2\mathcal{J}\boxplus\mathcal{J}\longrightarrow\mathcal{J}$ ($2\mathcal{J}\boxminus\mathcal{J}\longrightarrow\mathcal{J}$) and cost $\mathbf{DA}_2(\mathbf{I}, \mathbf{M}, \mathbf{S}) = 14\mathbf{M}+10\mathbf{S}$. Another option is to use simultaneous inversion to convert the precomputed values to affine coordinates (precomputations $pre_{2,3}$). Converting the precomputed points to affine coordinates would allow each addition to use $\mathcal{A}\boxplus\mathcal{J}\boxplus\mathcal{A}\longrightarrow\mathcal{J}$ ($\mathcal{A}\boxminus\mathcal{J}\boxplus\mathcal{A}\longrightarrow\mathcal{J}$), costing only $\mathbf{DA}_1(\mathbf{I}, \mathbf{M}, \mathbf{S}) = 11\mathbf{M}+7\mathbf{S}$. The trade-off for this is dependent on the size of the precomputed set and number of additions required.

With precomputed values in Jacobian form (pre_1), the rest of the algorithm has the following estimates.

On stage $Q\leftarrow 2Q\circ P$ executes:

$$\begin{cases}\mathcal{J}\boxplus\mathcal{A}\boxplus\mathcal{J}\longrightarrow\mathcal{J}, & \circ=\boxplus,\ i=n-2\\ \mathcal{J}\boxminus\mathcal{A}\boxplus\mathcal{J}\longrightarrow\mathcal{J}, & \circ=\boxminus,\ i=n-2\\ 2\mathcal{J}\boxplus\mathcal{J}\longrightarrow\mathcal{J}, & \circ=\boxplus,\ i\neq n-2\\ 2\mathcal{J}\boxminus\mathcal{J}\longrightarrow\mathcal{J}, & \circ=\boxminus,\ i\neq n-2\end{cases}$$

On stage $Q\leftarrow 2Q$ executes:

$$\begin{cases}2\mathcal{A}\longrightarrow\mathcal{J}, & \text{first step}\\ 2\mathcal{J}\longrightarrow\mathcal{J}, & \text{other steps}\end{cases}$$

On the last stage of the algorithm point Q is in Jacobian form, so it needs to be translated to affine form. Operator **return**(Q) makes translation from Jacobian to Affine form:

$$\mathcal{J}\longrightarrow\mathcal{A}.$$

This algorithm takes the following number of operations on average:

$$\widehat{L}_{Alg.2/m_1}(l, w, \mathbf{DA}_{1,2}, \mathbf{DBL}_{1,2}, \mathbf{SCALE}) = \left(\frac{1}{2^{w-2}}\right)\left(\frac{l}{w+1}-1\right)\mathbf{DA}_1+\left(\frac{2^{w-2}-1}{2^{w-2}}\right)\mathbf{DA}_2+\mathbf{DBL}_1$$
$$+\left(l-\frac{l}{w+1}-1\right)\mathbf{DBL}_2+\mathbf{SCALE}. \tag{3.4}$$

Next, consider the structure, formulas and computational complexity of operations used in expression (3.4).

Double and addition operation $P_3\leftarrow \mathrm{DA}_1(P_1, P_2)$, where $P_1=(X_1, Y_1, Z_1)\in E(\mathbb{F}_p)$, $P_2=(x_2, y_2)\in E(\mathbb{F}_p)$ and $P_3=(X_4, Y_4, Z_4)\in E(\mathbb{F}_p)$ has the following structure $\mathcal{J}\boxplus\mathcal{A}\boxplus\mathcal{J}\longrightarrow\mathcal{J}$ and calculated as follows:

$$\begin{aligned}X_4&=\omega^2\ominus\theta^3\ominus 2X_1^{(1)}\theta^2,\\ Y_4&=\omega(X_1^{(1)}\theta^2\ominus X_4)\ominus Y_1^{(1)}\theta^3,\\ Z_4&=Z_1^{(1)}\theta,\ \textit{where}\end{aligned} \tag{3.5}$$

$$\begin{aligned}\alpha&=Z_1^2y_2\ominus Y_1,\\ \beta&=Z_1^2x_2\ominus X_1,\\ X_1^{(1)}&=4X_1\beta^2,\\ Y_1^{(1)}&=8Y_1\beta^3,\end{aligned}$$

$$Z_1^{(1)} = (Z_1 \oplus \beta)^2 \ominus Z_1^2 \ominus \beta^2,$$
$$\theta = X_3 \ominus X_1^{(1)} = 4(\alpha^2 \ominus \beta^3 \ominus 3X_1\beta^2),$$
$$\omega = Y_3 \ominus Y_1^{(1)} = \alpha^2 \oplus \theta^2 \ominus (\alpha \oplus \theta)^2 \ominus 16Y_1\beta^3.$$

Note that we directly compute $\theta = X_3 \ominus X_1^{(1)}$ and $\omega = Y_3 \ominus Y_1^{(1)}$ to avoid the intermediate computations of X_3, Y_3 and Z_3 from the first addition, saving some field additions and trading one multiplication for one squaring. Therefore, the cost of the unified "double-and-add" operation has cost $\mathbf{DA}_1(\mathbf{I}, \mathbf{M}, \mathbf{S}) = (6\mathbf{M} + 5\mathbf{S}) + (5\mathbf{M} + 2\mathbf{S}) = 11\mathbf{M} + 7\mathbf{S}$ (3.5).

The point doubling operation $P_2 \leftarrow \mathrm{DBL}_1(P_1)$, where $P_1 = (x_1, y_1) \in E(\mathbb{F}_p)$, $P_2 = (X_1, Y_1, Z_1) \in E(\mathbb{F}_p)$ has structure $2\mathcal{A} \longrightarrow \mathcal{J}$ and computes as follows:

$$\begin{aligned} X_2 &= (3x_1^2 \oplus a)^2 \ominus 2\alpha, \\ Y_2 &= (3x_1^2 \oplus a)(\alpha \ominus X_2) \ominus 8y_1^4, \\ Z_2 &= 2y_1, \ \text{where} \\ \alpha &= 4x_1y_1^2 = 2[(x_1 \oplus y_1^2)^2 \ominus x_1^2 \ominus y_1^2]. \end{aligned} \tag{3.6}$$

The computational complexity of point doubling is $\mathbf{DBL}_1(\mathbf{I}, \mathbf{M}, \mathbf{S}) = 1\mathbf{M} + 5\mathbf{S}$ (3.6).

The point doubling operation $P_2 \leftarrow \mathrm{DBL}_2(P_1)$, where $P_1 = (X_1, Y_1, Z_1) \in E(\mathbb{F}_p)$, $P_2 = (X_2, Y_2, Z_2) \in E(\mathbb{F}_p)$ has structure $2\mathcal{J} \longrightarrow \mathcal{J}$ and computes as follows:

$$\begin{aligned} X_2 &= \alpha^2 \ominus 2\beta, \\ Y_2 &= \alpha(\beta \ominus X_2) \ominus 8Y_1^4, \\ Z_2 &= (Y_1 \oplus Z_1)^2 \ominus Y_1^2 \ominus Z_1^2, \ \textit{where} \\ \alpha &= 3(X_1 \oplus Z_1^2)(X_1 \ominus Z_1^2), \\ \beta &= 4X_1Y_1^2. \end{aligned} \tag{3.7}$$

The computational complexity of point doubling is $\mathbf{DBL}_2(\mathbf{I}, \mathbf{M}, \mathbf{S}) = 3\mathbf{M} + 5\mathbf{S}$ (3.7).

Another double-and-add operation $P_3 \leftarrow \mathrm{DA}_2(P_1, P_2)$, where $P_1 = (X_1, Y_1, Z_1) \in E(\mathbb{F}_p)$ and $P_1 = (X_2, Y_2, Z_2) \in E(\mathbb{F}_p)$ has the following structure $2\mathcal{J} \boxplus \mathcal{J} \longrightarrow \mathcal{J}$. The first step can be calculated by DBL_2. On final step we can apply $\mathbf{ADD}_1$ operation with $\mathcal{J} \boxplus \mathcal{J} \longrightarrow \mathcal{J}$ structure. Thus, the cost of the unified double-and-add operation has cost $\mathbf{DA}_2(\mathbf{I}, \mathbf{M}, \mathbf{S}) = (3\mathbf{M} + 5\mathbf{S}) + (11\mathbf{M} + 5\mathbf{S}) = 14\mathbf{M} + 10\mathbf{S}$.

The point scaling operation $P_2 \leftarrow \mathrm{SCALE}_1(P_1)$, where $P_1 = (X_1, Y_1, Z_1) \in E(\mathbb{F}_p)$, $P_2 = (x_3, y_2) \in E(\mathbb{F}_p)$ has structure $\mathcal{J} \longrightarrow \mathcal{A}$ and computes as follows:

$$\begin{aligned} x_2 &= X_1/Z_1^2, \\ y_2 &= X_1/Z_1^3, \end{aligned} \tag{3.8}$$

where $1/Z_1^2$ and $1/Z_1^3$ multiplicative inverse elements in field $\mathbb{F}_p$ for Z_1^2 and Z_1^3, respectively.

The computational complexity of point scaling operation is $\mathbf{SCALE}(\mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + 3\mathbf{M} + \mathbf{S}$ (3.8).

Then the computational complexity of the Algorithm 2 is:

$$\widehat{L}_{Alg.2/m_1}(l, w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \left(\frac{1}{2^{w-2}}\right)\left(\frac{l}{w+1} - 1\right)(11\mathbf{M} + 7\mathbf{S}) + \left(\frac{2^{w-2} - 1}{2^{w-2}}\right)(14\mathbf{M} + 10\mathbf{S}) + (1\mathbf{M} + 5\mathbf{S}) + \left(l - \frac{l}{w+1} - 1\right)(3\mathbf{M} + 5\mathbf{S}) + (\mathbf{I} + 3\mathbf{M} + \mathbf{S}).$$

$$\widehat{L}_{Alg.2/m_1}(l, w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + \left(3l - \frac{12l}{2^w(w+1)} + \frac{11l}{w+1} + \frac{12}{2^w} - 13\right)\mathbf{M} + \left(5l - \frac{12l}{2^w(w+1)} + \frac{5l}{w+1} + \frac{12}{2^w} - 9\right)\mathbf{S}.$$

$$\widehat{L}_{Alg.2/m_1}(l, w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + \left(\left(3 - \frac{1}{w+1}\left(\frac{12}{2^w} - 11\right)\right)l + \frac{12}{2^w} - 13\right)\mathbf{M} + \left(\left(5 - \frac{1}{w+1}\left(\frac{12}{2^w} - 5\right)\right)l + \frac{12}{2^w} - 9\right)\mathbf{S}.$$

Once all the precomputed values are in affine coordinates. On stage $Q \leftarrow 2Q \circ P$ executes:

$$\begin{cases} \mathcal{J} \boxplus \mathcal{A} \boxplus \mathcal{J} \longrightarrow \mathcal{J}, & \circ = \boxplus \\ \mathcal{J} \boxminus \mathcal{A} \boxplus \mathcal{J} \longrightarrow \mathcal{J}, & \circ = \boxminus \end{cases}$$

The operation $\circ = \boxplus$ in case $k_i > 0$ and $\circ = \boxminus$ in case $k_i < 0$. Since operations $\circ = \boxplus$ and $\circ = \boxminus$ have equal complexity (for affine and Jacobi forms) accurate to number of field additions, we will regard it as a single operation without loss of generality.

On stage $Q \leftarrow 2Q$ executes:

$$\begin{cases} 2\mathcal{A} \longrightarrow \mathcal{J}, & \text{first step} \\ 2\mathcal{J} \longrightarrow \mathcal{J}, & \text{other steps} \end{cases}$$

Table 3
Average computing M-cost of Algorithm 2 (pre_1: $main_1$ scenario).

w	w_1	w_2	w_3	w_4	w_5	w_6
pre_1	11.60**M**	24.80**M**	51.20**M**	104.00**M**	209.60**M**	420.80**M**
m_1^{192}	2006.98**M**	1939.04**M**	1872.26**M**	1815.40**M**	1768.77**M**	1730.74**M**
m_1^{224}	2329.38**M**	2250.40**M**	2172.66**M**	2106.43**M**	2052.10**M**	2007.78**M**
m_1^{256}	2651.78**M**	2561.76**M**	2473.06**M**	2397.46**M**	2335.42**M**	2284.81**M**
m_1^{384}	3941.38**M**	3807.20**M**	3674.66**M**	3561.58**M**	3468.72**M**	3392.94**M**
m_1^{521}	5321.65**M**	5140.21**M**	4960.75**M**	4807.54**M**	4681.71**M**	4578.99**M**
t^{192}	2018.58**M**	1963.84**M**	1923.46**M**	1919.40**M**	1978.37**M**	2151.54**M**
t^{224}	2340.98**M**	2275.20**M**	2223.86**M**	2210.43**M**	2261.70**M**	2428.58**M**
t^{256}	2663.38**M**	2586.56**M**	2524.26**M**	2501.46**M**	2545.02**M**	2705.61**M**
t^{384}	3952.98**M**	3832.00**M**	3725.86**M**	3665.58**M**	3678.32**M**	3813.74**M**
t^{521}	5333.25**M**	5165.01**M**	5011.95**M**	4911.54**M**	4891.31**M**	4999.79**M**

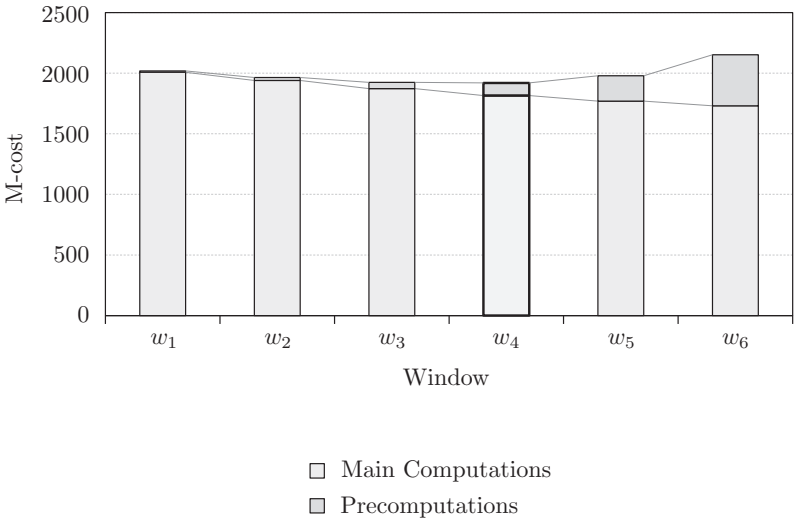

Fig. 1. Average computing M-cost of Algorithm 2 (pre_1: $main_1$ scenario) for $n = 192$.

On the last stage of the algorithm point Q is in Jacobian form, so it needs to be translated to affine form. Operator **return**(Q) makes translation from Jacobian to Affine form:

$$\mathcal{J} \longrightarrow \mathcal{A}.$$

When k_i is non-zero, $k_i = \pm 1$ happens with probability around $\frac{1}{2^{w-2}}$. With the precomputed values in affine form ($pre_{2,3}$), the rest of the algorithm has field cost:

$$\widehat{L}_{Alg.2/m_2}(l, w, \mathbf{DA}_1, \mathbf{DBL}_{1,2}, \mathbf{SCALE}) = \left(\frac{l}{w+1} - 1\right)\mathbf{DA}_1 + \mathbf{DBL}_1 + \left(l - \frac{l}{w+1} - 1\right)\mathbf{DBL}_2 + \mathbf{SCALE}.$$

After substitution:

$$\widehat{L}_{Alg.2/m_2}(l, w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \left(\frac{l}{w+1} - 1\right)(11\mathbf{M} + 7\mathbf{S}) + (1\mathbf{M} + 5\mathbf{S}) + \left(l - \frac{l}{w+1} - 1\right)(3\mathbf{M} + 5\mathbf{S}) + (\mathbf{I} + 3\mathbf{M} + \mathbf{S}).$$

$$\widehat{L}_{Alg.2/m_2}(l, w, \mathbf{I}, \mathbf{M}, \mathbf{S}) = \mathbf{I} + \left(\left(\frac{8}{w+1} + 3\right)l - 10\right)\mathbf{M} + \left(\left(\frac{2}{w+1} + 5\right)l - 6\right)\mathbf{S}.$$

Algorithm 2 requires the same number of temporary field elements as elements being inverted. For space-constrained implementations, this seems to pose a problem.

Next, we will analyze the effectiveness of the algorithm. Table 3 describes the average field cost (precomputations, main computations and total) of NAF scalar multiplication for d corresponding to the 5 NIST primes in pre_1: $main_1$ scenario.

The graphical interpretation of Table 3 is presented in Figs. 1 and 2. Table 4 describes the average field cost of NAF scalar multiplication for d corresponding to the 5 NIST primes in pre_2: $main_2$ scenario.

The graphical interpretation of Table 4 is presented in Figs. 3 and 4. Table 5 describes the average field cost of NAF scalar multiplication for d corresponding to the 5 NIST primes in pre_3: $main_2$ scenario.

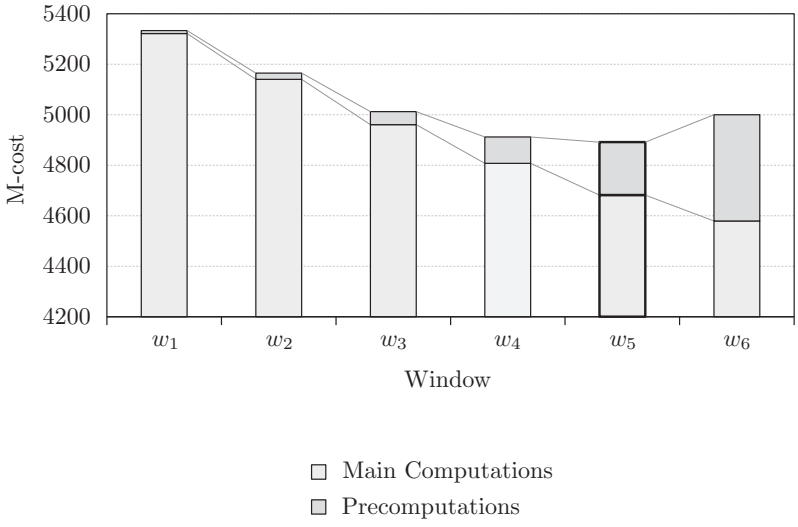

Fig. 2. Average computing M-cost of Algorithm 2 (pre_1: $main_1$ scenario) for $n = 521$.

Table 4
Average computing M-cost of Algorithm 2 (pre_2: $main_2$ scenario).

w	w_1	w_2	w_3	w_4	w_5	w_6
pre_2	95.40**M**	118.20**M**	163.80**M**	255.00**M**	437.40**M**	802.20**M**
m_2^{192}	1879.40**M**	1786.76**M**	1725.00**M**	1680.89**M**	1647.80**M**	1622.07**M**
m_2^{224}	2180.20**M**	2072.20**M**	2000.20**M**	1948.77**M**	1910.20**M**	1880.20**M**
m_2^{256}	2481.00**M**	2357.64**M**	2275.40**M**	2216.66**M**	2172.60**M**	2138.33**M**
m_2^{384}	3684.20**M**	3499.40**M**	3376.20**M**	3288.20**M**	3222.20**M**	3170.87**M**
m_2^{521}	4972.00**M**	4721.44**M**	4554.40**M**	4435.09**M**	4345.60**M**	4276.00**M**
t^{192}	1974.80**M**	1904.96**M**	1888.80**M**	1935.89**M**	2085.20**M**	2424.27**M**
t^{224}	2275.60**M**	2190.40**M**	2164.00**M**	2203.77**M**	2347.60**M**	2682.40**M**
t^{256}	2576.40**M**	2475.84**M**	2439.20**M**	2471.66**M**	2610.00**M**	2940.53**M**
t^{384}	3779.60**M**	3617.60**M**	3540.00**M**	3543.20**M**	3659.60**M**	3973.07**M**
t^{521}	5067.40**M**	4839.64**M**	4718.20**M**	4690.09**M**	4783.00**M**	5078.20**M**

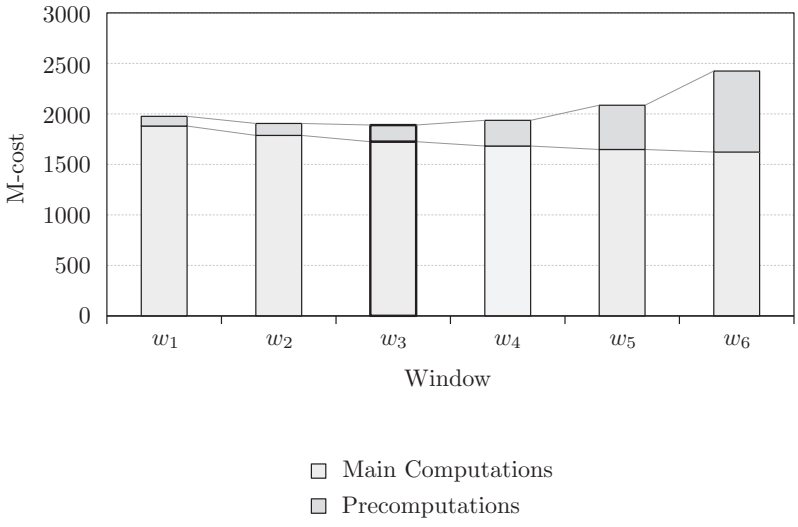

Fig. 3. Average computing M-cost of Algorithm 2 (pre_2: $main_2$ scenario) for $n = 192$.

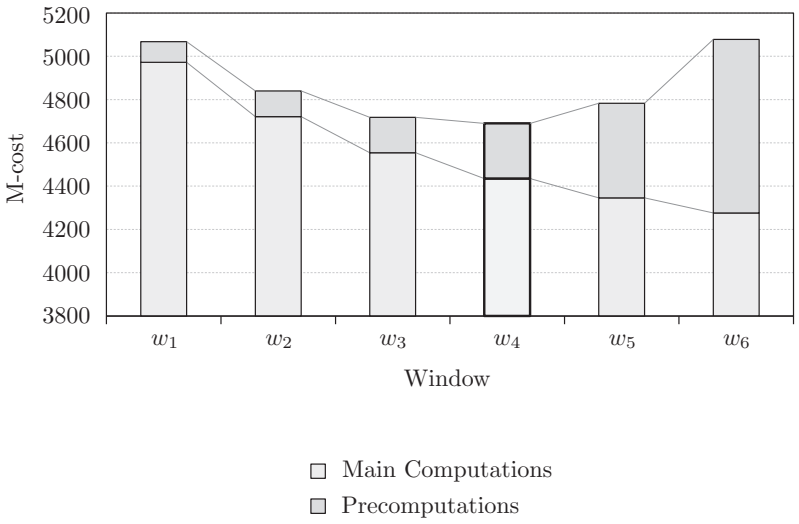

Fig. 4. Average computing M-cost of Algorithm 2 (pre_2: $main_2$ scenario) for $n = 521$.

Table 5
Average computing M-cost of Algorithm 2 (pre_3: $main_2$ scenario)

w	w_1	w_2	w_3	w_4	w_5	w_6
pre_3	93.80**M**	114.20**M**	155.00**M**	236.60**M**	399.80**M**	726,20**M**
m_2^{192}	1879.40**M**	1786.76**M**	1725.00**M**	1680.89**M**	1647.80**M**	1622.07**M**
m_2^{224}	2180.20**M**	2072.20**M**	2000.20**M**	1948.77**M**	1910.20**M**	1880.20**M**
m_2^{256}	2481.00**M**	2357.64**M**	2275.40**M**	2216.66**M**	2172.60**M**	2138.33**M**
m_2^{384}	3684.20**M**	3499.40**M**	3376.20**M**	3288.20**M**	3222.20**M**	3170.87**M**
m_2^{521}	4972.00**M**	4721.44**M**	4554.40**M**	4435.09**M**	4345.60**M**	4276.00**M**
t^{192}	1973.20**M**	1900.96**M**	1880.00**M**	1917.49**M**	2047.60**M**	2348.27**M**
t^{224}	2274.00**M**	2186.40**M**	2155.20**M**	2185.37**M**	2310.00**M**	2606.40**M**
t^{256}	2574.80**M**	2471.84**M**	2430.40**M**	2453.26**M**	2572.40**M**	2864.53**M**
t^{384}	3778.00**M**	3613.60**M**	3531.20**M**	3524.80**M**	3622.00**M**	3897.07**M**
t^{521}	5065.80**M**	4835.64**M**	4709.40**M**	4671.69**M**	4745.40**M**	5002.20**M**

Table 6
Minimal average computing M-cost of main part of Algorithm 2.

w	w_1	w_2	w_3	w_4	w_5	w_6
m^{192}	1879.40**M**	1786.76**M**	1725.00**M**	1680.89**M**	1647.80**M**	1622.07**M**
m^{224}	2180.20**M**	2072.20**M**	2000.20**M**	1948.77**M**	1910.20**M**	1880.20**M**
m^{256}	2481.00**M**	2357.64**M**	2275.40**M**	2216.66**M**	2172.60**M**	2138.33**M**
m^{384}	3684.20**M**	3499.40**M**	3376.20**M**	3288.20**M**	3222.20**M**	3170.87**M**
m^{521}	4972.00**M**	4721.44**M**	4554.40**M**	4435.09**M**	4345.60**M**	4276.00**M**
m^{192}	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$
m^{224}	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$
m^{256}	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$
m^{384}	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$
m^{521}	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$

The graphical interpretation of Table 5 is represent in Figs. 5 and 6. As can be seen from Tables 3–5, the minimum computational complexity of the main part of Algorithm 2 is achievable in the case of $main_2$, at which precomputations can be performed using $pre_{2,3}$. It proves the first item of the Theorem 3.2. The computational complexity of $main_2$ is given in Table 6.

Table 7 represents scenarios that allow one to achieve the minimum overall computational complexity for different values of n and w. At the top of the table, the minimum total computational complexity among all possible scenarios is presented, in the middle part of the table the type of precomputations is given, and in the lower part the type of main computations is presented.

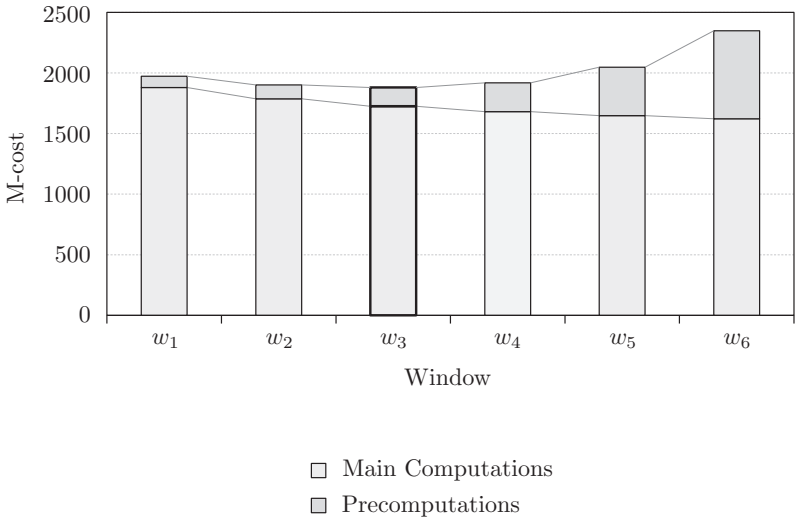

Fig. 5. Average computing M-cost of Algorithm 2 (pre_3: $main_2$ scenario) for $n = 192$.

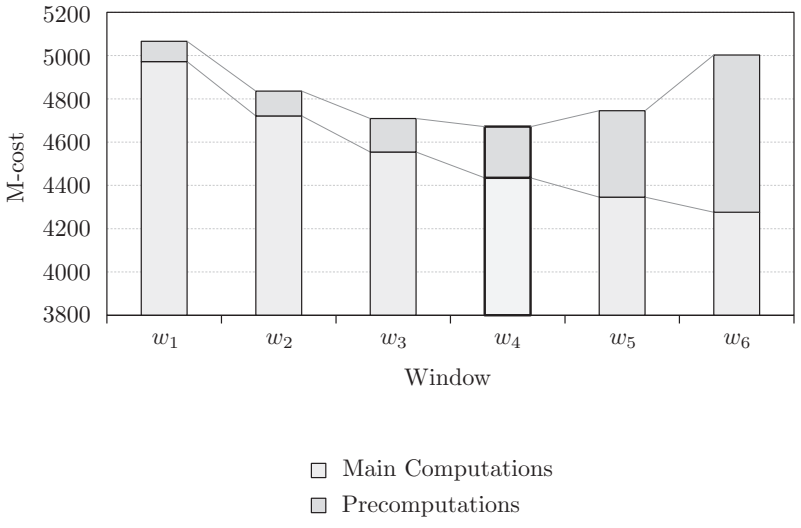

Fig. 6. Average computing M-cost of Algorithm 2 (pre_3: $main_2$ scenario) for $n = 521$.

Table 7
Minimal total average computing M-cost of Algorithm 2.

w	w_1	w_2	w_3	w_4	w_5	w_6
t^{192}	1985.50**M**	1900.96**M**	1880.00**M**	1884.53**M**	1947.01**M**	2123.37**M**
t^{224}	2274.00**M**	2186.40**M**	2155.20**M**	2169.56**M**	2224.91**M**	2395.50**M**
t^{256}	2574.80**M**	2471.84**M**	2430.40**M**	2453.26**M**	2502.81**M**	2667.63**M**
t^{384}	3778.00**M**	3613.60**M**	3531.20**M**	3524.80**M**	3614.41**M**	3756.17**M**
t^{521}	5065.80**M**	4835.64**M**	4709.40**M**	4671.69**M**	4745.40**M**	4921.24**M**
pre^{192}	pre_3	pre_3	pre_3	pre_3	pre_1	pre_1
pre^{224}	pre_3	pre_3	pre_3	pre_3	pre_1	pre_1
pre^{256}	pre_3	pre_3	pre_3	pre_3	pre_1	pre_1
pre^{384}	pre_3	pre_3	pre_3	pre_3	pre_1	pre_1
pre^{521}	pre_3	pre_3	pre_3	pre_3	pre_3	pre_1
m^{192}	$main_2$	$main_2$	$main_2$	$main_2$	$main_1$	$main_1$
m^{224}	$main_2$	$main_2$	$main_2$	$main_2$	$main_1$	$main_1$
m^{256}	$main_2$	$main_2$	$main_2$	$main_2$	$main_1$	$main_1$
m^{384}	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$	$main_1$
m^{521}	$main_2$	$main_2$	$main_2$	$main_2$	$main_2$	$main_1$

Table 8
Computing complexity of known algorithm [18, Alg. 17, $pre_{\mathcal{J}} : main_{\mathcal{J}}$ scenario].

w	w_1	w_2	w_3	w_4	w_5	w_6
$pre_{\mathcal{J}}$	17.40**M**	44.20**M**	97.80**M**	205.00**M**	419.40**M**	848.20**M**
$m_{\mathcal{J}}^{192}$	2026.48**M**	1939.84**M**	1870.15**M**	1815.28**M**	1771.91**M**	1737.20**M**
$m_{\mathcal{J}}^{224}$	2352.08**M**	2251.20**M**	2170.01**M**	2106.08**M**	2055.53**M**	2015.08**M**
$m_{\mathcal{J}}^{256}$	2677.68**M**	2562.56**M**	2469.88**M**	2396.88**M**	2339.16**M**	2292.95**M**
$m_{\mathcal{J}}^{384}$	3980.08**M**	3808.00**M**	3669.35**M**	3560.08**M**	3473.66**M**	3404.46**M**
$m_{\mathcal{J}}^{521}$	5374.05**M**	5141.01**M**	4953.15**M**	4805.06**M**	4687.93**M**	4594.13**M**
t^{192}	1973.00**M**	1984.04**M**	1967.95**M**	2020.28**M**	2191.31**M**	2585.40**M**
t^{224}	2369.48**M**	2295.40**M**	2267.81**M**	2311.08**M**	2474.93**M**	2863.28**M**
t^{256}	2695.08**M**	2606.76**M**	2567.68**M**	2601.88**M**	2758.56**M**	3141.15**M**
t^{384}	3997.48**M**	3852.20**M**	3767.15**M**	3765.08**M**	3893.06**M**	4252.66**M**
t^{521}	5391.45**M**	5185.21**M**	5050.95**M**	5010.06**M**	5107.33**M**	5442.33**M**

Table 9
Computing complexity of known algorithm [18, Alg. 17, $pre_{\mathcal{A}} : main_{\mathcal{J}}$ scenario].

w	w_1	w_2	w_3	w_4	w_5	w_6
$pre_{\mathcal{A}}$	101.20**M**	141.6**M**	222.4**M**	384**M**	707.2**M**	1353.6**M**
$m_{\mathcal{J}}^{192}$	1955.60**M**	1855.24**M**	1788.33**M**	1740.54**M**	1704.70**M**	1676.82**M**
$m_{\mathcal{J}}^{224}$	2269.20**M**	2152.20**M**	2074.20**M**	2018.49**M**	1976.70**M**	1944.20**M**
$m_{\mathcal{J}}^{256}$	2582.80**M**	2449.16**M**	2360.07**M**	2296.43**M**	2248.70**M**	2211.58**M**
$m_{\mathcal{J}}^{384}$	3837.20**M**	3637.00**M**	3503.53**M**	3408.20**M**	3336.70**M**	3281.09**M**
$m_{\mathcal{J}}^{521}$	5179.80**M**	4908.36**M**	4727.40**M**	4598.14**M**	4501.20**M**	4425.80**M**
t^{192}	2056.80**M**	1996.84**M**	2010.73**M**	2124.54**M**	2411.90**M**	3030.42**M**
t^{224}	2370.40**M**	2293.80**M**	2296.60**M**	2402.49**M**	2683.90**M**	3297.80**M**
t^{256}	2684.00**M**	2590.76**M**	2582.47**M**	2680.43**M**	2955.90**M**	3565.18**M**
t^{384}	3938.40**M**	3778.60**M**	3725.93**M**	3792.20**M**	4043.90**M**	4634.69**M**
t^{521}	5281.00**M**	5049.96**M**	4949.80**M**	4982.14**M**	5208.40**M**	5779.40**M**

As can be seen from Table 7, for $n = \{192, 224, 256\}$ the smallest computational complexity is achieved in the scenario pre_3: $main_2$ with the window width $w = w_3 = 5$ and for $n = \{384, 521\}$ in the scenario pre_3: $main_2$ with the window width $w = w_4 = 6$. It proves the second item of Theorem 3.3.

We can also see from Table 7 that for windows of the width $w > w_5$ $(w > 7)$ the most effective scenario is pre_1: $main_1$ for all values of n, which proves the third item of Theorem 3.3.

Another important result is that Table 7 shows scenarios of Algorithm 2 that can be used depending on n and available memory for storing results of precomputations in order to achieve the maximum computational efficiency.

For instance, for $n = 192$ and the window width $w = w_1$, the best scenario for proposed Algorithm 2 will be pre_3: $main_2$. This will allow one to dynamically run the necessary scenarios and achieve the minimum average computational complexity of scalar multiplication:

$$\widehat{L}_{min}(n, w) = \min_{pre_i : main_j, \forall i,j} \widehat{L}_{Alg.2}(n, w).$$

Therefore, Algorithm 2 can adapt to available environment resources during operation.

Thus, we have proved all three items of Theorem 3.3. Next, we compare the effectiveness of the well-known algorithm and that of the algorithm proposed in this article. □

4. Result comparison

We compare the scenarios of the scalar point multiplication algorithm based on the window NAF method (Algorithm 2) and the scenarios for using another well-known scalar multiplication algorithm proposed in this article [18, Alg. 17], which is based on the same method and uses point operations in affine and Jacobi coordinate systems.

Table 8 presents the computational complexity of the well-known algorithm [18, Alg. 17, $pre_{\mathcal{J}} : main_{\mathcal{J}}$ scenario]. Table 9 presents the computational complexity of the well-known algorithm [18, Alg. 17, $pre_{\mathcal{A}} : main_{\mathcal{J}}$ scenario].

Next, we will compare the effectiveness of the well-known and proposed algorithm in this article. We will compare the scenarios when the results of precomputations are applied to the input of the main part in the affine coordinates as the most effective (Algorithm 2 pre_2: $main_2$ scenario versus [18, Alg. 17, $pre_{\mathcal{A}} : main_{\mathcal{J}}$ scenario] in Table 10 and Algorithm 2 pre_3: $main_2$ scenario versus [18, Alg. 17, $pre_{\mathcal{A}} : main_{\mathcal{J}}$ scenario] in Table 11).

Table 10 shows that the greatest advantage of Algorithm 2 (pre_2: $main_2$ scenario with effective configuration, see Theorem 3.3(2) reaches 6.06% when $n = 192$ in total. In this configuration, precomputations are more effective by 26.35% when $n = \{192, 224, 256, 384\}$ and by 33.59% when $n = 521$.

Table 10
Effectiveness comparison of the well-known [18, Alg. 17, $pre_{\mathcal{A}}$: $main_{\mathcal{J}}$ scenario] and proposed Algorithm 2 (pre_2: $main_2$ scenario).

w	$w_1(\%)$	$w_2(\%)$	$w_3(\%)$	$w_4(\%)$	$w_5(\%)$	$w_6(\%)$
pre	5.73	16.53	26.35	33.59	38.15	40.74
t^{192}	3.99	4.60	6.06	8.88	13.55	20.00
t^{224}	4.00	4.51	5.77	8.27	12.53	18.66
t^{256}	4.01	4.44	5.55	7.79	11.70	17.52
t^{384}	4.03	4.26	4.99	6.57	9.50	14.28
t^{521}	4.04	4.16	4.68	5.86	8.17	12.13

Table 11
Effectiveness comparison of the well-known [18, Alg. 17, $pre_{\mathcal{A}}$: $main_{\mathcal{J}}$ scenario] and proposed Algorithm 2 (pre_3: $main_2$ scenario).

w	$w_1(\%)$	$w_2(\%)$	$w_3(\%)$	$w_4(\%)$	$w_5(\%)$	$w_6(\%)$
pre	7.31	19.35	30.31	38.39	43.47	46.35
t^{192}	4.06	4.80	6.50	9.75	15.10	22.51
t^{224}	4.07	4.68	6.16	9.04	13.93	20.97
t^{256}	4.07	4.59	5.89	8.48	12.97	19.65
t^{384}	4.07	4.37	5.23	7.05	10.43	15.92
t^{521}	4.07	4.24	4.86	6.23	8.89	13.45

Table 11 shows that the greatest advantage of Algorithm 2 (pre_3: $main_2$ scenario with effective configuration, see Theorem 3.3 (2) reaches 7.05% when $n = 384$ in total. In this configuration, precomputations are more effective by 30.31%, when $n = \{192, 224, 256\}$ and by 38.39% when $n = \{384, 521\}$.

5. Conclusion

Scalar multiplication is the main and the most important operation in curve-based cryptosystems. In this article, we have concentrated on the problem of optimizing such an operation on ECC standard curves over prime fields in terms of computational complexity.

The algorithm for scalar point multiplication based on the wNAF method has been proposed and considered.

Based on the results of this study, we can draw the following conclusions.

The average computing complexity of scalar point multiplication based on the window NAF method depends on the following factors: the parameter domain of the elliptic curve, the scalar representation and its length, the scalar scan direction, the volume of the precomputations, the type of coordinates in which the precomputation results are presented, the usage of composite operations, the usage of mixed coordinate systems and the ratios of finite field operations complexity.

The new algorithm for scalar point multiplication has been proposed and the theorem on the average computational complexity of precomputation and the main parts is formulated and proved.

The analysis of all scenarios for the new algorithm has been carried out. For each value of n and window w, the scenario has been found that allows one to compute the scalar multiplication of the point with the least complexity. This will make it possible to use the proposed algorithm in a dynamically changing environment and to achieve the greatest efficiency of its operation.

The comparative analysis of the proposed and the well-known algorithms has been carried out. Depending on the scenario, the optimal window w and the parameter n, the proposed Algorithm 2 is more efficient than the well-known algorithm [18, Alg. 17] by 26.35% to 50.02%at the precomputation step and by 4.53–7.05% in total. The maximum advantage is equal to 50.39% when $w = w_6$ and $n = 192$ at the precomputation step and to 22.51% in total.

References

[1] V. Miller, Use of elliptic curves in cryptography, in: Proceedings of the Advances in Cryptology–CRYPTO '85, 1986, pp. 417–426.
[2] N. Koblitz, Elliptic curve cryptosystems, Math. Comput. 48 (1987) 203–209.
[3] National Institute of Standards and Technology (NIST), Recommendation for Key Management Part 1: General (Revision 3). NIST PUB 800-57. National Institute of Standards and Technology (NIST), 2012.
[4] National Institute of Standards and Technology (NIST), Digital Signature Standard (DSS). FIPS PUB 186-2. National Institute of Standards and Technology (NIST), 2000.
[5] ANSI X9.62:2005, American National Standard for Financial Services – Public key cryptography for the financial services industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), American Bankers Association, Washington, DC, 2005.
[6] Certicom Corp., SEC 1: Elliptic Curve Cryptography. Standards for Efficient Cryptography. Certicom Corp., 2009.
[7] D. Chudnovsky, G. Chudnovsky, Sequences of numbers generated by addition in formal groups and new primality and factorization tests, Adv. Appl. Math. 7 (1986) 385–434.
[8] P. Longa, A. Miri, New composite operations and precomputation scheme for elliptic curve cryptosystems over prime fields, in: Proceedings of the Public Key Cryptography, 2008, pp. 229–247.
[9] D. Hankerson, A. Menezes, S. Vanstone, Guide to Elliptic Curve Cryptography, Springer-Verlag, 2004.

[10] P. Longa, Accelerating the Scalar Multiplication on Elliptic Curve Cryptosystems Over Prime Fields, 2008, 2007, p. 100. IACR Cryptology ePrint Archive
[11] H. Cohen, A. Miyaji, T. Ono, Efficient elliptic curve exponentiation using mixed coordinates, in: Proceedings of the Advances in Cryptology ASIACRYPT 98, in: Lecture Notes in Computer Science, 1514, Springer-Verlag, 1998, pp. 51–65.
[12] W. Bosma, Signed bits and fast exponentiation, J. Theor. Nr. Bordx. 13 (2001) 27–41.
[13] The Institute of Electrical and Electronics Engineers (IEEE), IEEE Standard Specifications for Public-Key Cryptography. IEEE Std 1363–2000, The Institute of Electrical and Electronics Engineers (IEEE), 2000.
[14] O. Billet, M. Joye, Fast point multiplication on elliptic curves through isogenies, in: Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, in: Lecture Notes in Computer Science, 2643, Springer-Verlag, 2003, pp. 43–50.
[15] K. Eisentrager, K. Lauter, P. Montgomery, Fast elliptic curve arithmetic and improved weil pairing evaluation, in: Proceedings of the Topics in CryptographyCT-RSA 2003, in: Lecture Notes in Computer Science, vol. 2612, 2003, pp. 343–354. 230
[16] M. Ciet, M. Joye, K. Lauter, P. Montgomery, Trading inversions for multiplications in elliptic curve cryptography, Des. Codes Cryptogr. 39 (2) (2006) 189–206.
[17] V. Dimitrov, L. Imbert, P.K. Mishra, Efficient and secure elliptic curve point multiplication using double-base chains, in: Proceedings of the Advances in Cryptology ASIACRYPT 05, in: Lecture Notes In Computer Science, vol. 3788, 2005, pp. 59–78.
[18] N.T. Sullivan, Fast Algorithms for Arithmetic on Elliptic Curves Over Prime Fields, University of Calgary, Canada, 2008 Master's thesis.
[19] B. Fay, Double-and-add with relative Jacobian coordinates. Cryptology ePrint Archive, Report 2014/1014 (2014), http://eprint.iacr.org/.
[20] P. Longa, A. Miri, Fast and flexible elliptic curve point arithmetic over prime fields, IEEE Trans. Comput. 57 (3) (2008) 289–302.
[21] N. Meloni, Fast and secure elliptic curve scalar multiplication over prime fields using special addition chains 2006 (2006) 216. IACR Cryptology ePrint Archive.
[22] B. Möller, Improved Techniques for Fast Exponentiation, in: P.J. Lee, C.H. Lim (Eds.), Information Security and Cryptology – ICISC 2002. ICISC 2002. Lecture Notes in Computer Science, 2587, Springer, Berlin, Heidelberg, 2003.
[23] K. Okeya, K. Schmidt-Samoa, C. Spahn, T. Takagi, Signed Binary Representations Revisited, in: M. Franklin, C.H. Lim (Eds.), Advances in Cryptology CRYPTO 2004. CRYPTO 2004. Lecture Notes in Computer Science, 3152, Springer, Berlin, Heidelberg, 2004.
[24] J.A. Muir, D.R. Stinson, Minimality and other properties of the width-w nonadjacent form, Math. Comput. 75 (2006) 369–384.
[25] K. Koyama, Y. Tsuruoka, Speeding up elliptic cryptosystems by using a signed binary window method, in: Proceedings of the Advances in Cryptology–CRYPTO' 92, 1993, pp. 345–357.