Comparative analysis of the scalar point multiplication algorithms in the NIST FIPS 186 elliptic curve cryptography

Mikhail Babenko[1,2†], Andrei Tchernykh[2,3,4‡], Aziz Redvanov[1††] and Anvar Djurabaev[5‡‡]

[1] North-Caucasus Center for Mathematical Research, North-Caucasus Federal University, 1, Pushkin Street, 355017, Stavropol, Russia
[2] Institute for System Programming of the Russian Academy of Sciences, 109004, Moscow, Russia
[3] CICESE Research Center, carr. Tijuana-Ensenada 3918, 22860, Ensenada, BC, Mexico
[4] South Ural State University, Prospekt Lenina 76, 454080, Chelyabinsk, Russia.
[5] ITMO University, 9, Lomonosova Street, Saints-Petersburg, 191002, Russia

E-mail: †mgbabenko@ncfu.ru, ‡chernykh@cicese.mx, ††azizredvanov@yandex.ru, ‡‡a.djurabaev@yandex.ru

Abstract. In today's world, the problem of information security is becoming critical. One of the most common cryptographic approaches is the elliptic curve cryptosystem. However, in elliptic curve arithmetic, the scalar point multiplication is the most expensive compared to the others. In this paper, we analyze the efficiency of the scalar multiplication on elliptic curves comparing Affine, Projective, Jacobian, Jacobi-Chudnovsky, and Modified Jacobian representations of an elliptic curve. For each coordinate system, we compare Fast exponentiation, Nonadjacent form (NAF), and Window methods. We show that the Window method is the best providing lower execution time on considered coordinate systems.

1. Introduction

Throughout history, humanity has experienced the need to protect information. Many cryptosystems of different complexity were created. However, in the last 20 years, due to the growth in the computing power, most algorithms based on the complexity of factorization have become more vulnerable to cryptographic attacks. As a result, it became necessary to use alternative approaches to build cryptosystems with more advanced protocols. Currently, one of the most secure systems in cryptography is a cryptosystem based on elliptic curves (Elliptic Curve Cryptosystems -ECC). Elliptic cryptography is secure because there are currently no subexponential algorithms for solving the discrete logarithm problem [1, 2].

The most expensive and widely used operation on ECC is scalar point multiplication because it is used for key generation, encryption/decryption of data, and signing/verification of digital signatures. Scalar multiplication denoted by nP where P represents a point on the ellipic curve and n represents a scalar. Efficient implementation of multiplying the points of an elliptic curve by a scalar is important.

This research aims to study the methods of scalar point multiplication of NIST FIPS 186 elliptic curves and their implementation in the C++ programming language.

The rest of this paper is organized as follows. Section 2 briefly introduces elliptic curve concept, affine, projective, Jacobian, Jacobi-Chudnovsky, and modified Jacobian coordinate systems. Section 3 describes the actual research and algorithms of scalar multiplication on elliptic curves. The main libraries the calculation of integer values of arbitrary length are reviewed in Sections 4. Section 5 provides experimental analysis on NIST FIPS 186 standard. Finally, we conclude and discuss future works in Section 6.

2. Elliptic curve

In the standards such as GOST 34.10-2018, NIST FIPS 186, the elliptical curve is described by the following parameters:

- p – field characteristic, prime number;
- a, b – constants that are parameters equation of the curve $y^2 = x^3 + ax + b$;
- $G = (Gx, Gy)$ – point of a large-order elliptic curve;
- n – point order G;
- h – cofactor determined by the ratio of the total number of points on a curve to the order of a point G, which should be the smallest possible.

An elliptic curve over the field F_p is the set of points (x, y) satisfying the following equation [2-4]:

$$E(F_p):\ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

To achieve efficiency of elliptic cryptosystem implementations, curves over a simple field defined in the Weierstrass form are used:

$$E(F_p):\ y^2 = x^3 + ax + b$$

where $a, b \in F_p$ – constants satisfying $4a^3 + 27b^2 \not\equiv 0\ (mod\ p)$ and $p > 3$ and p – prime number.

The set of points on the EC form a group (namely, an Abelian group) with the following properties:

- if P and $Q \in F_p$, then $P + Q \in F_p$;
- unit element O (point at infinity): $P + O = P$;
- inverse value of the point $P(x; y)$ – that point $-P(x; -y)$;
- commutativity: $P + Q = Q + P$;
- associativity: $P + (Q + R) = (P + Q) + R = P + Q + R = O$.

2.1. Affine coordinate system

Let the points $P(x_P, y_P)$, $Q(x_Q, y_Q) \in E(F_p)$, then the addition of points $P(x_P; y_P)$ and $Q(x_Q; y_Q)$ is called the point $R(x_R; y_R) = P + Q$ then the coordinates of the point R will be calculated as follows [2]:

$$x_R = s^2 - x_P - x_Q \pmod p,$$
$$y_R = -y_P + s \cdot (x_P - x_R) \pmod p,$$

where $s = \begin{cases} \frac{y_P - y_Q}{x_P - x_Q} \pmod p, & \text{if } P \neq Q \\ \frac{3 * x_P^2 + a}{2 * y_P} \pmod p, & \text{if } P = Q \end{cases}$

In practical calculations, an affine coordinate system is inefficient since it requires the calculation of the inverse element in the field F_p.

A practical solution to this problem can be a transition to other coordinate systems (mainly to projective ones) by eliminating modular inversion.

2.2. Projective coordinate system

In the projective coordinate system, the point of an elliptic curve is given by $P = (X_P: Y_P: Z_P)$. If $Z_p = 0$ than $P = O$ – is a point at infinity. If $Z_P \neq 0$, then the point P can be put in the affine coordinate system: $x_P = X_P/Z_P$ и $y_P = Y_P/Z_P$ [5, 6].

The equation of an elliptic curve in a projective coordinate system has the following form:

$$E(F_p):\ Y^2 \cdot \mathrm{Z} = X^3 + aX \cdot Z^2 + bZ^3$$

The addition operation of points $(P\ +\ Q)$ on the elliptic curve in a projective coordinate system has the form:

$$X_R = B \cdot D$$
$$Y_R = A \cdot (X_P \cdot Z_Q \cdot B^2 - D) - B^3 \cdot Y_P \cdot Z_Q$$
$$Z_R = \mathrm{B}^3 \cdot Z_P \cdot Z_Q$$

where $A = Y_Q \cdot Z_P - Y_P \cdot Z_Q$, $B = X_Q \cdot Z_P - X_P \cdot Z_Q$, $C = X_Q \cdot Z_P + X_P \cdot Z_Q$ and $D = A^2 \cdot Z_P \cdot Z_Q - B^2 \cdot \mathrm{C}$.

The operation of doubling the point $(2P)$ is given as follows:

$$X_R = 2v \cdot o$$
$$Y_R = u \cdot (4t - o) - 8Y_P^2 \cdot v^2$$
$$Z_R = 8v^3$$

where $u = 3X_P^2 + aZ_P^2, v = Y_P \cdot Z_P, t = X_P \cdot Y_P \cdot B$ and $o = A^2 - 8C$.

2.3. Projective Jacobi coordinate system

In the projective Jacobi coordinate system, the point of the elliptic curve is given by $P = (X_P : Y_P : Z_P)$. If $Z_p = 0$ then $P = O$ – a point at infinity. If $Z_P \neq 0$, then P can put a point in the affine coordinate system: $x_P = X_P/Z_P^2$ и $y_P = Y_P/Z_P^3$ [6-8].

The algorithm addition of points on an elliptic curve in projective Jacobi coordinates has the form:

$$X_R = -C^3 - 2 \cdot A_1 \cdot C^2 + D^2$$
$$Y_R = -B_1 \cdot C^3 + D \cdot (A_1 \cdot C^2 - X_R)$$
$$Z_R = \mathrm{C} \cdot Z_P \cdot Z_Q$$

where $A_1 = X_P \cdot Z_Q^2, A_2 = X_Q \cdot Z_P^2, B_1 = Y_P \cdot Z_Q^3, B_2 = Y_Q \cdot Z_P^3, C = A_2 - A_1$ и $D = B_2 - B_1$.

The point doubling algorithm is specified as follows:

$$X_R = \mathrm{t}$$
$$Y_R = -8Y_P^4 + v \cdot (u - t)$$
$$Z_R = 2Y_P \cdot Z_P$$

where $u = 4X_P \cdot Y_P^2, v = 3X_P^2 + \mathrm{a} \cdot Z_P^4, t = -2u + v^2$.

2.4. Jacobi-Chudnovsky coordinate system

In the Jacobi-Chudnovsky projective coordinate system, the point on an elliptic curve is given by five coordinates $P = (X_P : Y_P : Z_P : Z_P^2 : Z_P^3)$ [7, 8]. If $Z_P = 0$ then $P = O$ – a point at infinity. If $Z_P \neq 0$, then P can put a point in the affine coordinate system: $x_P = X_P/Z_P^2$ and $y_P = Y_P/Z_P^3$.

2.5. Modified Jacobi coordinate system

In the modified projective Jacobi coordinate system, the point on the elliptic curve is given by four coordinates $P = (X_P : Y_P : Z_P : aZ_P^4)$ [7, 8]. If $Z_P = 0$ then $P = O$ – a point at infinity. If $Z_P \neq 0$, то P can put a point in the affine coordinate system: $x_P = X_P/Z_P^2$ и $y_P = Y_P/Z_P^3$.

3. Scalar multiplication on elliptic curves

3.1. Scalar multiplication by Fast Exponentiation

Algorithm 1 presents the operation of the scalar point multiplication by the Fast Exponentiation (FE) method.

Let n – be a scalar, $P \in E(F_p)$, then nP is calculated by the formula $nP = \sum_{i=0}^{k-1} n_i \cdot 2^i \cdot P$, where n_i – are the binary digits of the number n.

Algorithm 1. Scalar multiplication of an elliptic curve point by FE method [5-12]

Input: $n = (n_{k-1}, n_{k-2}, \dots, n_1, n_0)$ and $P \in E(F_p)$.
Output: $R = nP$.

1. $R = O$
2. $i = 0$
3. $while\ i < k$:
3.1. $if\ n_i == 1$:
3.1.1. $R += P$
3.2. $P = 2P$
3.3. $i += 1$
4. $Return\ R.$

3.2. Scalar multiplication by the NAF method

NAF (nonadjacent form) is obtained from a modification of the fast exponentiation method by introducing an additional operation of subtracting a point, thereby reducing the amount of operations by representing a number in NAF with coefficients $\{-1,0,1\}$.

Algorithm 2 presents the number representation in NAF, and Algorithm 3 describes the scalar point multiplication operation [11-17].

Algorithm 2. NAF(n).

Input: n
Output: $NAF(n) = (n_0, n_1, \dots, n_{k-2}, n_{k-1})$

1. $i = 0$
2. $while\ n \geq 1$:
2.1. $if\ n\ mod\ 2$:
2.1.1. $n_i = 2 - (n\ mod\ 4)$
2.1.2. $n -= n_i$
2.2. *else:* $n_i = 0$
2.3. $n /= 2$
2.4. $i += 1$
3. $Return\ (n_0, n_1, \dots, n_{k-2}, n_{k-1})$.

Algorithm 3. Scalar multiplication by the NAF method

Input: $NAF(n) = (n_0, n_1, \dots, n_{k-2}, n_{k-1})$ и $P \in E(F_p)$.
Output: $R = nP$.

1. $R = O$
2. $i = 0$
3. $while\ i < k$:
3.1. $if\ n_i == 1$:
3.1.1. $R += P$
3.2. $if\ n_i == -1$:
3.2.1. $R -= P$
3.3. $P = 2P$
3.4. $i += 1$
4. $Return\ R$.

3.3. Scalar multiplication by the window method

If there is enough memory, then window-based methods (window methods) can be used to speed up the operation of multiplying an elliptic curve point by a scalar. It was proposed by Brauer in 1939. The idea is to cut the scalar k into digits and process W digits at the same time.

The scalar k in window methods is represented in the base 2^w (or 2^r in the radix-r method), where $w, r > 1$. The algorithms in this method would significantly improve the speed of scalar multiplication. It processes the W-bit k at a time, at the cost of 2^{w-2} points in the memory lookup table.

For example, computing ECSM using the windowing method introduced by Thurber requires the set $iP, i \in \{1, 3, 5, 7, \cdots, 2^{w-1}\}$, the points must be precomputed and stored in memory. A typical standard method for computing ECSM in radix-r is shown in Algorithm 4. In this algorithm, the average density of non-zero digits is $\left(\frac{r-1}{r}\right)$.

Algorithm 4 shows that the addition operation at step 1.2.1 and subtraction operation at step 1.3 begin only after completing repeated doubling operations at step 1.1.

Algorithm 4. Radix-r standard signed scalar multiplication from left to right [18, 19]

Input: l-bit Radix-r of k and dot $P \in E(\mathbb{F})$, где, $k = (R_{l-1}, R_{l-2}, \dots, R_1, R_0)_r$, $R_i \in \{0,1,2, \dots, (r-1)\}$.

Output: Point $Q = kP$.

Pre-calculations: $|R_i|P$ for all $R_i \in \{1,2, \dots, (r-1)\}$.
Initialize: $Q = \mathcal{O}$;
Step 1: For $i = l - 1$ down to 0 do
Step 1.1: $Q = rQ$; /* Perform a re-doubling operation */
Step 1.2: If $R_i \geq 0$ Then
Step 1.2.1: $Q = Q + R_i P$; /* Perform the addition operation */
Step 1.3: Else $Q = Q - R_i P$ /* Perform the subtraction operation */
Step 2: End For
Step 3: Return Q.

It is a pure sequential method in computing operations with elliptic curve points at the low level of the addition group.

Note that in order to make these window methods feasible for implementations that support parallel processing at the low level of the addition group, all precomputed points must be doubled $w - 1$ times at each iteration.

Let us denote the cost of the ECSM computational operation using kP_{cost}. Then the kP_{const} of the s-bit scalar k using the windows method is approximately equal to:

$$kP_{cost} = (s-1)DBL + \left(\frac{s}{w+1}\right)ADD.$$

Notably, the Window Methods (WM) described here do not provide resilience to SCAs. These techniques must be performed in a regular manner to counter most SCAs.

4. Long arithmetic library

Nowadays, cryptography is very often faced with tasks that require the calculation of integer values of arbitrary length. But not all programming languages have built-in tools for such calculations. Hence, you have to either create your solutions or use additional libraries such as GMP, NTL, CLN, MIRACL, etc., which help in calculations with acceptable accuracy.

The independent **GMP** library, which is one of the fastest libraries and supports most of the latest platforms (Mac OS X/Darwin, GNU/Linux, Windows, and others), has the following advantages [20, 21]:

- Practically there are no limits on the accuracy of calculations, except for a limited amount of available memory
- user-friendly programming interface
- rich set of various functions
- supports most modern platforms: Unix-like operating systems such as GNU / Linux, Solaris, HP-UX, Mac OS X / Darwin, BSD, AIX; Windows. There are 32-bit and 64-bit versions of GMP
- uses the most efficient algorithms and assembly code optimized for various modern processor systems in all internal cycles.

The main areas of application of the library are cryptographic systems and research, security of inter-network interactions, algebraic packages. The GMP library is part of the GNU Project.

The **CLN** library, which allows you to perform operations using all existing numeric types, uses GMP as the computing core and is slightly inferior in the speed of calculations [22]. This library implements classes for integers, rational fractions, float numbers, complex numbers, univariant polynomials, modulo calculations. It has a user-friendly interface, a rich set of supported types, transparent mechanisms of interaction, and transformation of various data structures into each other, together with a sufficiently high computation speed, ensure the widespread use of this library. It uses in scientific research by many software products (Octave, maxima, Scilab, etc.).

MIRACL – is a library with a commercial license, which has the following advantages:

- rich library of specialized functions for calculations in the field of cryptography on elliptic curves;
- availability of implemented C / C++ interfaces and various algorithms for solving problems to choose the optimal option for current needs [23].

The High-performance C ++ **NTL** library is famous for its high speed in algorithms for factorization and determination of the order of elliptic curves and polynomial arithmetic [24]. NTL represents structure and algorithms for integers, float numbers, polynomials, vectors, and matrices of various lengths and accuracy. All NTL algorithms are implemented in C ++, which ensures the high portability of this library. As a computing core, not only the NTL computing core but also the GMP library.

Since NTL is one of the fastest today, it has been used more than once in setting "world records" in the speed of factorization algorithms and determining the order of elliptic curves. As a library for the implementation of long arithmetic in the C ++ programming language, we will use it.

5. Comparative analysis

For the experiment, the following curves are selected from the NIST FIPS 186 standard [25-27]: $P-192$, $P-224$, $P-256$, $P-384$, and $P-521$.

Curve $P-192$

p = 6277101735386680763835789423207666416083908700390324961279
n = 6277101735386680763835789423176059013767194773182842284081
a = -3
b = 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1
Gx = 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012
Gy = 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811

Curve $P-224$

p = 26959946667150639794667015087019630673557916260026308143510066298881
n = 26959946667150639794667015087019625940457807714424391721682722368061
a = -3
b = b4050a85 0c04b3ab f5413256 5044b0b7 d7bfd8ba 270b3943 2355ffb4
Gx = b70e0cbd 6bb4bf7f 321390b9 4a03c1d3 56c21122 343280d6 115c1d21
Gy = bd376388 b5f723fb 4c22dfe6 cd4375a0 5a074764 44d58199 85007e34

Curve $P-256$

p = 115792089210356248762697446949407573530086143415290314195533631308867097853951

n = 115792089210356248762697446949407573529996955224135760342422259061068512044369
a = -3
b = 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b
Gx = 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296
Gy = 4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5

Curve $P-384$

p = 39402006196394479212279040100143613805079739270465446667948293404245721771496870329047266088258938001861606973112319
n = 39402006196394479212279040100143613805079739270465446667946905279627659399113263569398956308152294913554433653942643
a = -3
b = b3312fa7 e23ee7e4 988e056b e3f82d19 181d9c6e fe814112 0314088f 5013875a c656398d8a2ed19d 2a85c8ed d3ec2aef
Gx = aa87ca22 be8b0537 8eb1c71e f320ad74 6e1d3b62 8ba79b98 59f741e0 82542a38 5502f25d bf55296c 3a545e3872760ab7
Gy = 3617de4a 96262c6f 5d9e98bf 9292dc29 f8f41dbd 289a147c e9da3113 b5f0b8c0 0a60b1ce 1d7e819d 7a431d7c 90ea0e5f

Curve $P-521$

p= 6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151
n = 6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449
a = -3
b = 051 953eb961 8e1c9a1f 929a21a0 b68540ee a2da725b 99b315f3 b8b48991 8ef109e1 56193951 ec7e937b 1652c0bd 3bb1bf07 3573df88 3d2c34f1 ef451fd4 6b503f00
Gx = c6 858e06b7 0404e9cd 9e3ecb66 2395b442 9c648139 053fb521 f828af60 6b4d3dba a14b5e77 efe75928 fe1dc127 a2ffa8de 3348b3c1 856a429b f97e7e31 c2e5bd66
Gy = 118 39296a78 9a3bc004 5c8a5fb4 2c7d1bd9 98f54449 579b4468 17afbd17 273e662c 97ee7299 5ef42640 c550b901 3fad0761 353c7086 a272c240 88be9476 9fd16650.

All algorithms are implemented on an Intel Core i3-2310M dual-core 2.10 GHz processor and 8.00 GB of RAM. The points of each curve are multiplied by 1000 different constants in the range $[n/2\,;n)$, where n is the order of the point. Also, for the window method, the window size is equal to 23.

Table 1-5 and figure 1 shows the execution time of the algorithms for the scalar point multiplication in the coordinate systems described earlier: Affine, Projective, Jacobian, Chudnovsky Jacobian, and Modified Jacobian.

Tables 1-5 show execution time in seconds of the scalar multiplication in five coordinate systems, using the curve $P-192$, $P-224$, $P-256$, $P-384$, and $P-521$ respectively.

For curve $P-192$ (see, table 1), the best method is VM method showing best time 4.948s for Modified Jacobian coordinate systems and worst time 30.318 for Affine. FE method shows results that are 13.56%- 24.14% worse than WM. NAF method shows results that are 1.63%- 7.72% worse than WM.

Table 1. Execution time in seconds of the scalar multiplication of the point of an elliptic curve in different coordinate systems, using the curve $P-192$

	FE	**NAF**	**WM**	**FE/WM-1**	**NAF/WM-1**
Affine	34.428	30.812	30.318	13.56%	1.63%
Projective	5.849	5.116	4.976	17.54%	2.81%
Jacobian	5.614	4.722	4.537	23.74%	4.08%
Chudnovsky Jacobian	5.662	4.913	4.561	24.14%	7.72%
Modified Jacobian	6.067	5.059	4.948	22.62%	2.24%

For curve $P-224$ (see, table 2), the best method is VM method showing best time 5.534s for Jacobian coordinate system and worst time 46.301s for Affine. FE method shows results that are 13.91% - 28.66% worse than WM. NAF method shows results that are 2.63%- 7.35% worse than WM.

Table 2. Execution time in seconds of the scalar multiplication of the point of an elliptic curve in different coordinate systems, using the curve $P-224$

	FE	NAF	WM	FE/WM-1	NAF/WM-1
Affine	52.743	47.521	46.301	13.91%	2.63%
Projective	7.55	6.488	6.184	22.09%	4.92%
Jacobian	7.12	5.941	5.534	28.66%	7.35%
Chudnovsky Jacobian	7.131	6.16	5.847	21.96%	5.35%
Modified Jacobian	7.739	6.403	6.032	28.30%	6.15%

For curve $P-256$ (see, table 3), the best method is VM method showing best time 5.534s for Jacobian coordinate system and worst time 46.301s for Affine. FE method shows results that are 13.91% - 28.66% worse than WM. NAF method shows results that are 2.63%- 7.35% worse than WM.

Table 3. Execution time in seconds of the scalar multiplication of the point of an elliptic curve in different coordinate systems, using the curve $P-256$

	FE	NAF	WM	FE/WM-1	NAF/WM-1
Affine	72.03	63.348	62.154	15.89%	1.92%
Projective	9.907	8.339	7.648	29.54%	9.04%
Jacobian	9.263	7.673	7.157	29.43%	7.21%
Chudnovsky Jacobian	9.238	7.871	7.488	23.37%	5.11%
Modified Jacobian	10.041	8.39	7.892	27.23%	6.31%

For curve $P-384$ (see. table 4), the best methods are VM and NAF methods showing best times about 21.131s for Projective coordinate system and worst time 259.123s for Affine. FE method shows results that are 15.78% - 24.94% worse than WM. NAF method shows results that are 0.00% - 7.35% worse than WM.

Table 4. Execution time in seconds of the scalar multiplication of the point of an elliptic curve in different coordinate systems, using the curve $P-384$

	FE	NAF	WM	FE/WM-1	NAF/WM-1
Affine	300.02	267.299	259.123	15.78%	3.16%
Projective	24.835	21.131	21.132	17.52%	0.00%
Jacobian	22.954	19.093	18.98	20.94%	0.60%
Chudnovsky Jacobian	22.527	19.167	19.156	17.60%	0.06%
Modified Jacobian	25.017	20.702	20.023	24.94%	3.39%

For curve $P-521$ (see, table 5), the best methods are VM method showing best times about 36.489s for **Chudnovsky Jacobian** coordinate system and worst time 786.516s for Affine. FE method shows results that are 14.30% - 24.30% worse than WM. NAF method shows results that are 0.74% - 4.18% worse than WM.

Table 5. Execution time in seconds of the scalar multiplication of the point of an elliptic curve in different coordinate systems, using the curve $P-521$

	FE	NAF	WM	FE/WM-1	NAF/WM-1
Affine	898.963	799.708	786.516	14.30%	1.68%
Projective	49.963	41.874	40.194	24.30%	4.18%
Jacobian	45.889	37.864	37.431	22.60%	1.16%
Chudnovsky Jacobian	43.882	37.468	36.489	20.26%	2.68%
Modified Jacobian	50.348	41.117	40.813	23.36%	0.74%

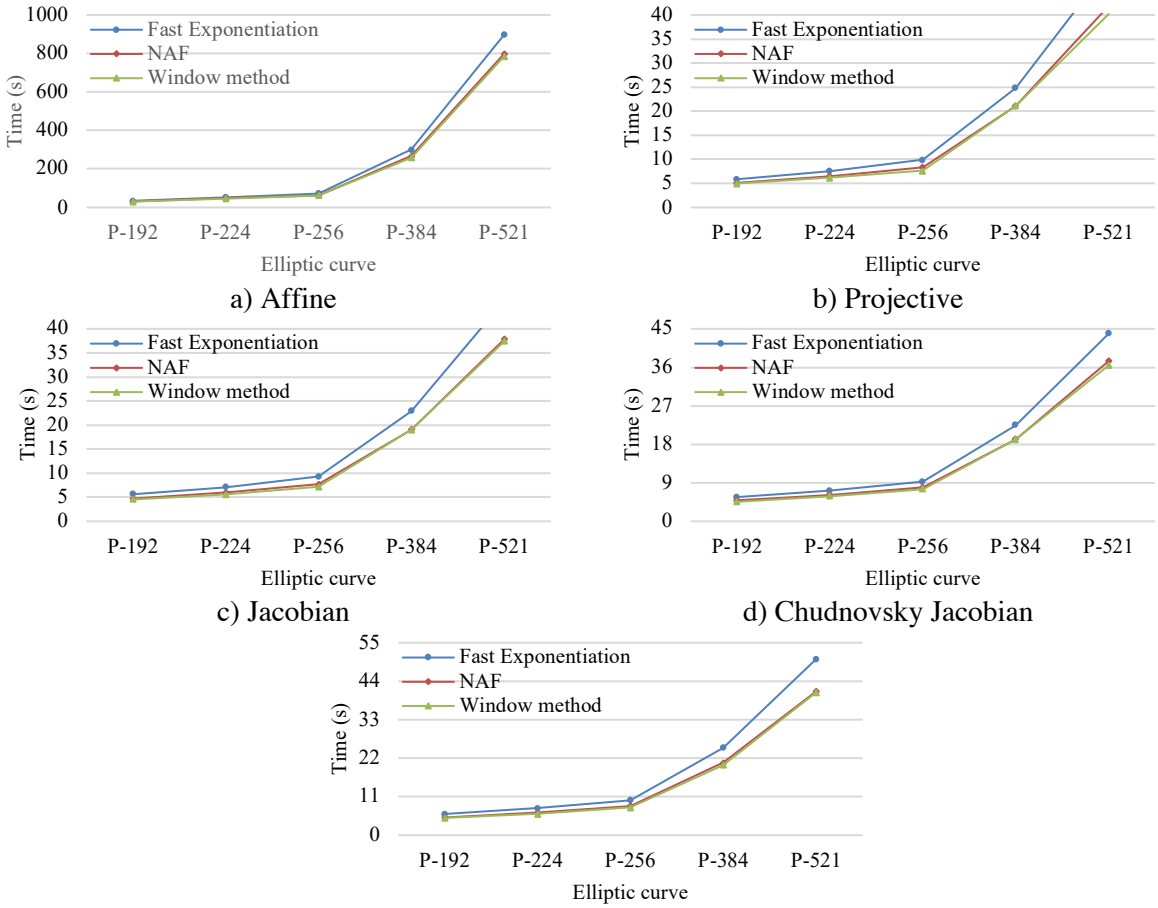

Figure 1. Execution time in seconds of the scalar multiplication of the point of an elliptic curve in different elliptic curves: $P-192, P-224, P-256, P-384, P-521$, using the coordinate systems: a) Affine; b) Projective; c) Jacobian; d) Chudnovsky Jacobian; e) Modified Jacobian.

Figure 1 shows that the time grows significantly with increasing the bit size of the curve. Results are obtained with the parameter $a = 3$ in the elliptic curve equation.

6. Conclusion

Experimental evaluation shows that the WM method is an efficient method of the scalar integer multiplication of an elliptic curve point, especially if there is enough free space in the computer memory (at least for storing 2^{22} points of the elliptic curve). It gives the better execution time comparing with the Fast Exponentiation and Nonadjacent Form methods in five coordinated systems: Affine, Projective, Jacobian, Chudnovsky Jacobian, and Modified Jacobian.

Acknowledgments. The reported study was funded by RFBR, project number 20-37-70023.

References

[1] Gong G and Lam C 2002 Linear recursive sequences over elliptic curves. *In: Sequences and their applications*. (London: Springer) pp 182–196

[2] Hankerson D, Vanstone S and Menezes A 2004 Guide to Elliptic Curve Cryptography (Springer-Verlag/New York)

[3] Johnson D, Menezes A and Vanstone S 2001 The elliptic curve digital signature algorithm (ECDSA) *International journal of information security* v 1 no, 1 pp 36-63

[4] Silverman J H 2009 The arithmetic of elliptic curves. (Springer Science and Business Media) v 106
[5] Okeya K, Miyazaki K and Sakurai K 2001 A fast scalar multiplication method with randomized projective coordinates on a Montgomery-form elliptic curve secure against side channel attacks *Int. Conf. on Information Security and Cryptology*. (Springer, Berlin and Heidelberg) pp 428-39
[6] Cohen H et al. 2005 Handbook of elliptic and hyperelliptic curve cryptography (Chapman and Hall/CRC)
[7] Gutub A A A and Arabia S 2010 Remodeling of elliptic curve cryptography scalar multiplication architecture using parallel jacobian coordinate system *Int. Journal of Computer Science and Security (IJCSS)* v **4** no. 4 p 409
[8] Gallant R P, Lambert R J and Vanstone S A 2001 Faster point multiplication on elliptic curves with efficient endomorphisms *Annual Int. Cryptology Conf.* (Springer, Berlin and Heidelberg) pp 190-200
[9] Avanzi R, Cohen H, Doche C, Frey G, Lange T, Nguyen K and Vercauteren F 2005 Handbook of elliptic and hyperelliptic curve cryptography *CRC press*
[10] Daly A, Marnane W, Kerins T and Popovici E 2004 An FPGA implementation of a GF (p) ALU for encryption processors *Microprocessors and Microsystems* v 28 no. 5-6 pp 253-60
[11] Washington L C 2003 Elliptic curves: number theory and cryptography *Discrete Mathematics and Its Applications* (Chapman & Hall/CRC)
[12] Dormale G M 2007 Destructive and constructive aspects of efficient algorithms and implementation of cryptographic hardware (Catholic University of Louvain, Louvain-la-Neuve and Belgium)
[13] Trappe W and Washington L C 2007 Introduction to Cryptography
[14] Verneuil V 2012 Elliptic curve cryptography and security of embedded devices
[15] Galbraith S D, Lin X and Scott M 2011 Endomorphisms for faster elliptic curve cryptography on a large class of curves *Journal of cryptology* v **24** no. **3** pp 446-69
[16] Molahosseini A S, De Sousa L S and Chang C H 2017 Embedded systems design with special arithmetic and number systems (New York: Springer) p 390
[17] Okeya K and Takagi T 2003 The width-w NAF method provides small memory and fast elliptic scalar multiplications secure against side channel attacks *Cryptographers' Track at the RSA Conf.* (Springer, Berlin and Heidelberg) pp 328-43.
[18] Taverne J, Faz-Hernandez A, Aranha D F, Rodriguez-Henriquez F, Hankerson D and Lopez J 2011 Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction *Journal of Cryptographic Engineering* v **1** no. **3** p 187
[19] Han D G and Takagi T 2005 Some Analysis of Radix-r Representations *IACR Cryptology ePrint Archive* p 402.
[20] Hart W B 2010 Fast library for number theory: an introduction *International Congress on Mathematical Software* (Springer, Berlin and Heidelberg) pp 88-91
[21] GMP «Arithmetic without limitation» *The GNU Multiple Precision Arithmetic Library (Electronic resource)* (Electron. dat. - Access mode: http://gmplib.org/) free Title from the title screen
[22] CLN-Class Library for Numbers *(Electronic resource)* (Electron. dat. - Access mode: http://www.ginac.de/CLN/) free. - Title from the title screen
[23] Multiprecision Integer and Rational Arithmetic C/C++ Library *(Electronic resource)* (Electron. dat. - Access mode: http://www.shamus.ie/) free. - Title from the title screen.
[24] NTL: A Library for doing Number Theory *(Electronic resource)* (Electron. dat. - Access mode: http://www.shoup.net/ntl/) free. - Title from the title screen.
[25] Digital Signature Standard (DSS) *(Electronic resource)* (Electron. dat. - Access mode: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf) free. - Title from the title screen.

[26] Barker E, Mouha N 2017 Recommendation for the triple data encryption algorithm (TDEA) block cipher *National Institute of Standards and Technology (no. NIST Special Publication (SP))* pp 800-67
[27] Brown D R L 2010 Standards for efficient cryptography sec 2: Recommended elliptic curve domain parameters *Certicom Research, Certicom Corp*