; **00**:1–18

Accelerated Precomputation Points Based Scalar Reduction on Elliptic Curve Cryptography for Wireless Sensor Networks

International Journal of Communication Systems†

Youssou Faye [1*] , Hervé Guyennet[1], Shou Yanbo[1], Ibrahima Niang[2]

[1]*Franche Comte University, Femto-st, 16, route de Gray, BP 25030 Besancon France*
[2]*Department of Mathematics and Computer Sciences, Cheikh Anta Diop University, BP 5026 Dakar, Senegal*

SUMMARY

Sensor devices are limited resource power and energy, thus providing security services for sensor networks is very difficult. Elliptic Curve Cryptography (ECC) is one of the most famous asymmetric cryptographic schemes which offers the same level of security with much shorter keys compared to the other widely used asymmetric cryptographic algorithm, RSA. In ECC, the main and most-heavily used operation is the scalar multiplication kP, where the scalar value k is a private integer and must be secured. In this work, we present a new approach to accelerate the main scalar multiplication on ECC over prime fields for sensor networks. This approach uses an equivalent representation of points and can act as a support for existing schemes in a selected interval. The simulation results showed that the proposed technique increases the efficiency of the computation time. For example, on this scalar multiplication we obtain a gain of 4 bits in 161 bits for 6.25% of the scalars. This gain can sometimes reach 100% in some cases. After this significant reduction of the scalar k, we present a fast precomputation algorithm in a distributed scalar multiplication on kP to avoid storage of precomputation points, which requires extra memory. ©

...

KEY WORDS: Wireless Sensor Networks, Elliptic Curve Cryptography, Fast Scalar Multiplication, Precomputation Point.

1. INTRODUCTION

Wireless Sensor Networks (WSN) presents physical and technological vulnerabilities which expose them to the risks of many kinds of attacks [1]. Symmetric cryptographic algorithms usually have low computational cost, but in WSN key management is a critical issue especially in the presence of compromised nodes. In asymmetric cryptography, key management becomes easier, but the computational cost is more expensive [2]. In this paper, we use the Elliptic Curve Cryptography (ECC), a famous asymmetric cryptographic scheme which is feasible and more flexible for WSN. It has attracted increasing attention recently because of its shorter key length requirement compared

*Correspondence to: Franche Comte University, Femto-st, 16, route de Gray, BP 25030 Besancon France. E-mail: yfaye@univ-zig.sn

to the other widely used asymmetric cryptographic algorithm RSA [3]. The security of ECC relies mainly on the difficulty of discrete logarithm problems. For example: $Q = kP$ where Q and P are 2 points on the curve and k is a positive integer. It is extremely difficult to compute the value of k given Q and P if k is big enough.

The scalar multiplication kP is the central and most time-consuming computation in ECC, it is involved in all major operations: key generation, encryption of data, decryption of data, signature and verification of messages. To permit fast computation of scalar multiplication, much research has been carried out on point arithmetic levels [6],[11], [10], [9], [21], [29], [28], [31], [32]. Some solutions use technical implementation on different microcontroller architectures [4], [33] and others use multiprocessor architectures to perform several operations, but they need to store precomputed points [5], [30], [18], [27].
Our main contributions are as follows:

1. We first propose a scalar reduction technique to run fast computation of scalar multiplication. In the scalar multiplication kP, we make an equivalent representation tP with a scalar $t < k$. This technique is based on the negative of point and the point order. According to our knowledge, this is the first method based on this technique. Existing algorithms can basically use our technique to accelerate computation.
2. After a significant reduction of the scalar k, we present a fast precomputation algorithm in a parallel scalar multiplication on kP to accelerate computation. Most existing methods based on precomputation require some extra memory [5], [27], [8], [22]. Our goal is to avoid the storage of precomputation points. To deal with this, we propose to accelerate the precomputation phase. This method is an improvement of the Double-and-Add and quadruple-and-quadruple algorithm and based on an algorithm to scan the scalar[6]. This method involves the level of scalar arithmetic and point arithmetic in Jacobian coordinates.

The discussion on this article proceeds as: In Section 2 we start with the background on ECC over prime fields. Section 3 gives related work on fast and parallel scalar multiplication, followed by the description of our new scalar reduction and an efficiency analysis on it in Section 4. Section 5 proposes our method to accelerate precomputation point. After defining the partitioning scalar method in (Section 5.1) and discussing its reliability in (Section 5.2), we present the acceleratiion of the precomputation points in (Section 5.3) and make a comparison in (Section 5.4). The conclusion and perspectives are given in the last section.

2. PRELIMINARIES ON ELLIPTIC CURVE CRYPTOGRAPHY

In this section, we give a brief overview on ECC over finite prime fields. An elliptic curve E over finite field $\mathbb{F}$ (of order n) denoted by $E(\mathbb{F})$ can be defined by the Weierstrass equation [16]:

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

where a and $b \in \mathbb{F}_p$, and $\mathbb{F}_p$ a prime field.
Most important finite fields used to date to implement cryptosystem have been binary, prime and

extension fields. In this paper, we work with a prime field $\mathbb{F}_p$, where $p > 3$ and p= q^r, with *r=1* and q a prime number called the characteristic of $\mathbb{F}_p$.

To be used for cryptography, the necessary condition is the discriminant of polynomial:

$$f(x) = x^3 + ax + b, \triangle = 4a^3 + 27b^2 \neq 0. \tag{2}$$

The set of pairs (x, y) that solves (1), where x,y $\in \mathbb{F}_p$, and the point at infinity (denoted ∞) form an abelian group. The scalar multiplication directly depends on two basic operations over points on an elliptic curve: point doubling (2*P*) and point addition *(P+Q)* where *P* and *Q* are two different points on the elliptic curve.

If $P = (x_p, y_p)$ and $Q = (x_q, y_q)$ 2 points ($\neq \infty$) on the elliptic curve over $\mathbb{F}_p$ denoted by $E(\mathbb{F}_p)$, then point addition $P + Q = (x_{pq}, y_{pq})$ or point doubling $2P = P + Q = (x_{pq}, y_{pq})$ if $P = Q$ can be calculated as:

$$\begin{cases} x_{pq} = \lambda^2 - x_p - x_q \\ y_{pq} = \lambda(x_p - x_{p+q}) - y_p \end{cases} \tag{3}$$

$$\begin{cases} \lambda = \dfrac{y_q - y_p}{x_q - x_p} & if\ P \neq Q \\ \lambda = \dfrac{3x_p^2 + a}{2y_p} & if\ P = Q \end{cases} \tag{4}$$

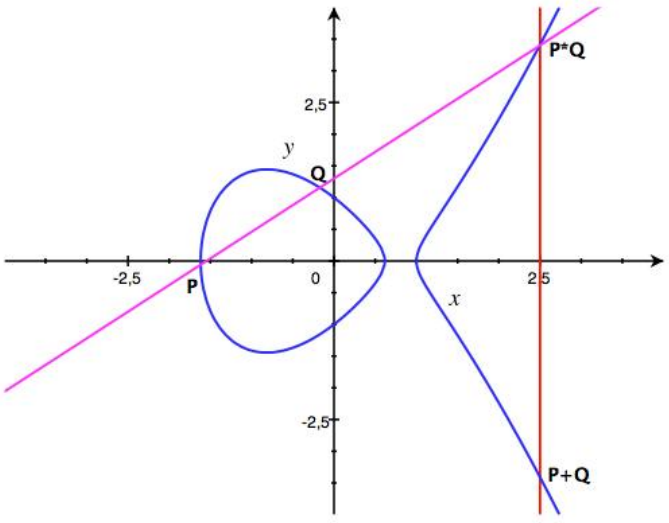

Figure 1. Points addition in ECC

The negative of a point $P = (x_p, y_p)$ is the point *-P* $(x_p, -y_p)$, where *P* and *-P* are two points on the elliptic curve.

3. RELATED WORK

To perform fast computation of scalar multiplication, much research has focused on the arithmetic level and parallel computing.

On the scalar arithmetic level, traditional methods (Double-and-Add algorithm, Non-Adjacent Form (NAF), sliding window algorithm) use binary representation of the scalar k and compute a sequence of given point doubling and point addition operations and reduce effectively the number of point operations [11], [6]. Other algorithms like Double-Base Number System have been developed to accelerate scalar multiplication by using binary and ternary representation [10], [9], [21]. Unlike the previous ones where the scalar is represented in a one base number system, in the Double-Base Number System [10], the scalar is represented in a two bases number system. Optimization is done by some approaches which also use the binary representation of the scalar k [29], [28], [31]. For other solutions, optimization is based on selecting a set of elliptic curves for cryptography (Weierstrass curve, twisted Edwards curve) on which scalar multiplication is faster than the recent implementation record on the corresponding NIST curve.

On point arithmetic, algebraic substitutions can be used. For example, the multiplication operation can be substituted by squaring or other cheaper field operations such as addition, subtraction and multiplication or division by a small constant [20]. Some other schemes use the concept of optimizing formulas like point addition formulas by computing the sum of any two points on any odd-order elliptic curve with q>= 5 [34]. The authors claim that the cost of their addition formulas is 12 field multiplications (12M), while the fastest known addition formulas require 14 field multiplications (12M+2S) in homogeneous coordinates and 16 field multiplications (11M + 5S) in Jacobian coordinates.

The parallel computing of scalar multiplication has become an important topic in cryptography. In the literature, various solutions focus on the scalar arithmetic level and base on precomputation, but most of them are hardware implementations using FPGA or multi-core architectures [24], [25]. Parallel computing of fast exponentiation is introduced in [18]. The paper [8] presents a fast exponentiation method using precomputed table. Another method based on point precomputation is proposed in [22].

Recently, has been developed concept of using multiprocessor architectures to compute the scalar multiplication by several operations simultaneously. At the scalar arithmetic level, the concept of parallelization is used to compute the series of doubling and addition operations with two processor architectures using a shared memory [5]. The first processor computes the point doubling operation and stores the result in the shared memory, and the second processor computes the point addition operation. The same technique is used in [33] with two scalar multipliers. Each scalar multiplier has one processor and a buffer residing between the two processors.This parallelization is used in [30], [18] by partitioning the scalar into v equal-length bit substrings on multiprocessor architectures. This technique is also used on sensor nodes in [27] by partitioning the scalar in v blocks for v sensor nodes, where each sensor node computes one block. At the point arithmetic level, some works already propose the parallelization of ECC formulas for architectures such as the well-known SIMD [20]. In this case, parallel computing of terms in the formula can be used according to the order of priority in the point operation formula.

4. NEW SCALAR REDUCTION METHOD (SR)

4.1. Context

We present a new technique to accelerate computation on the scalar arithmetic level. This improvement is based on the negative of a point and a specific reduction of the scalar in a selected interval. Using negation is a well-known trick in cryptanalysis (e.g. the negation map optimization in Pollard rho for computation of discrete logarithms) as well as in cryptography for computation of scalar multiplication with addition-subtraction chains. [13], [14].

Assume that the characteristic of $\mathbb{F}_p$ is greater than 3. Let $E(\mathbb{F}_p)$ be an elliptic curve over $\mathbb{F}_p$ and $\#E(\mathbb{F}_p)$ denote the number of points of $E(\mathbb{F}_p)$. $\#E(\mathbb{F}_p)$ is also called the order of the group of points. The theorem of Hasse indicates that [16]: $|\#E(\mathbb{F}_p) - p - 1 \leq 2\sqrt{p}|$. Let $\mathbb{G}$ be a cyclic group of $E(\mathbb{F}p)$ of order n generated by a base point P (namely, the generator point). The points in $\mathbb{G}$ are expressed as multiples of P: $\mathbb{G}=\langle P\rangle=\{\infty, P, 2P, ..., (n-2)P, (n-1)P\}\subseteq E(\mathbb{F}_p)$ with $nP=\infty$. The order of point P (denoted by $\#P$) is n.

4.2. Description of the scalar reduction method

We propose to replace the point kP by an equivalent representation of point tP in the main scalar multiplication operation where k and t are scalars and $k > t$. This technique is used in the interval $[\lfloor n/2\rfloor+1, n-1]$, where $\lfloor n/2\rfloor$ denotes the integer-part function of $n/2$. As the negative of a point is obtained freely, we use it to make fast computation. Given the point $P=(x_p, y_p)$ in affine coordinates, to compute the negative of the point $kP=(x_{kp}, y_{kp})$, we can compute $kP=(x_{kp}, y_{kp})$ and then change the sign on the y-coordinate (y_{kp}). Thus, by kP we get equivalent point tP through equation(5).

$$\begin{cases} 1. \quad \text{If k} \in]\lfloor \frac{n}{2}\rfloor, \text{n-1}], \; kP = tP \quad \text{where } t = (k-n) \\ 2. \quad \text{If k} \in]0, \lfloor \frac{n}{2}\rfloor], \; kP = tP \quad \text{where } t = k \end{cases} \tag{5}$$

Without losing generality, we use an example in order to better express the reduction method. Assume that p =23 is a prime number; this is only for explaining our new method, but in real life p is much bigger than this. If we consider an elliptic E over $\mathbb{F}_{23}$ defined by $E(\mathbb{F}_{23})$: $y^2=x^3+x+1$, then $\#E(\mathbb{F}_{23})=28$, $E(\mathbb{F}_{23})$ is a cyclic group and $P(0, 1)$ is a generator point.

In Figure 2, we make an equivalent representation of all points in $[\lfloor n/2\rfloor+1, n-1]$. Thus we can see that computing the points $16P$, $22P$ and $27P$ can be replaced respectively by $-12P$, $-7P$, and $-P$. In this case, computing $27P$ is replaced by computing $-P$ and is almost free.

4.3. Analytical evaluation

For WSN, replacing computation kP by tP using equation (5.1) in $[\lfloor \frac{n}{2}\rfloor+1, n-1]$ can accelerate scalar multiplication. From equation(6), we can scan all scalars:

$$\sum_{k=1}^{n-1} kP = \sum_{k=1}^{\lfloor n/2\rfloor-1} kP + \lfloor \frac{n}{2}\rfloor P + \sum_{k=\lfloor n/2\rfloor+1}^{n-1} kP \tag{6}$$

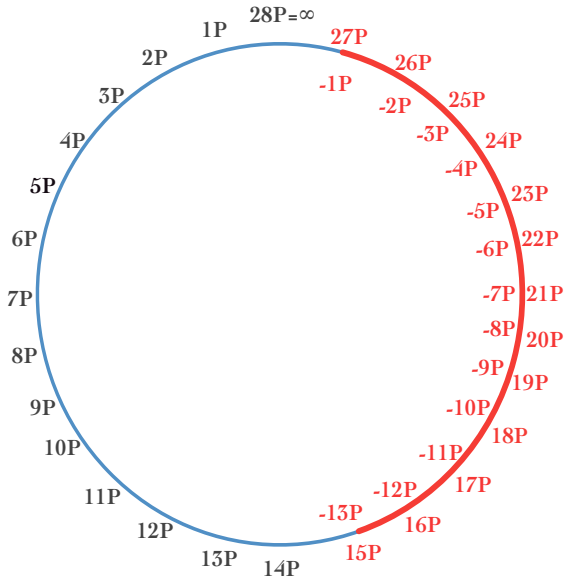

Figure 2. Circular representation of points over elliptic curve cryptography

In the interval [$\lfloor \frac{n}{2} \rfloor$+1, n-1], based on the Figure 2 we have:

$$\begin{aligned}
[15]P &= [13]P + 2([1]P);\\
[16]P &= [12]P + 2([2]P);\\
[17]P &= [11]P + 2([3]P);\\
\ldots\ldots\ldots &= \ldots\ldots\ldots\ldots\ldots;\\
\ldots\ldots\ldots &= \ldots\ldots\ldots\ldots\ldots;\\
[26]P &= [2]P + 2([12]P);\\
[27]P &= [1]P + 2([13]P).
\end{aligned}$$

It can be inferred that: $\sum_{k=\lfloor n/2 \rfloor+1}^{n-1} kP = \sum_{k=1}^{\lfloor n/2 \rfloor-1} kP + 2\sum_{k=1}^{\lfloor n/2 \rfloor-1} kP$

Thus:

$$\sum_{k=1}^{n-1} kP = 2\sum_{k=1}^{\lfloor n/2 \rfloor-1} kP + \lfloor \frac{n}{2} \rfloor P + 2\sum_{k=1}^{\lfloor n/2 \rfloor-1} kP \tag{7}$$

In our technique, we can replace respectively [15]P, [16]P, ..., [26]P, [27]P] by [-13]P, [-12]P, ..., [-2]P,[-1]P in interval [$\lfloor \frac{n}{2} \rfloor$+1, n-1]. The expression $\sum_{k=\lfloor n/2 \rfloor+1}^{n-1} kP$ can be replaced by $\sum_{k=1}^{\lfloor n/2 \rfloor-1} |k|P$, and the equation(7) can be replaced by equation(8).

$$\sum_{k=1}^{n-1} kP = \sum_{k=1}^{\lfloor n/2 \rfloor-1} kP + \sum_{k=1}^{\lfloor n/2 \rfloor-1} |k|P + \lfloor \frac{n}{2} \rfloor P. \tag{8}$$

For *kP* in [$\lfloor \frac{n}{2} \rfloor$+1, n-1], we gain a rate of $\sum_{k=1}^{\lfloor n/2 \rfloor - 1} 2kP$ from equation(7) and equation(8). The gain α for a given scalar *kP* is:

$$\alpha = 2(k - (\frac{n}{2})) \tag{9}$$

From Figure 2:

- Computing *kP*=24*P*, can be replaced by computation of 4*P*, the gain is *2(24-(28/2))= 20P* since *20P+ 4P= 24P*.
- Computing ([26]*P*=[24]*P*+[2]*P*) is equal to compute [2]*P*, the gain is [22]*P= 2(26-(28/2))*.

The complexity of scalar multiplication can be determined by the bit length of *k* which is equal to $\lfloor log_2(k) \rfloor$+1, or $log_2(k)$ if k=2^x, where *x* is an integer. In binary representation, $log_2(k)$ can be replaced in our Scalar Reduction technique by:

$$log_2(k - 2(k - \frac{n}{2})) = log_2(k) + log_2(1 + \frac{n - 2k}{k}) \tag{10}$$

Thus, the gain α in bit length is:

$$\alpha = |log_2(1 + \frac{n - 2k}{k})| = |log_2(\frac{|t|}{k})|. \tag{11}$$

Optimization can be done in interval [$\lfloor \frac{n}{2} \rfloor$+1, n-1] if the order *n* is an even number .

- If *n* >2 is even: $\sum_{k=\lfloor n/2 \rfloor + 1}^{n-1} kP$=3$\sum_{k=1}^{\lfloor n/2 \rfloor - 1} kP$
- If *n* ≥3 is odd: $3\sum_{k=1}^{\lfloor n/2 \rfloor - 1} kP > \sum_{k=\lfloor n/2 \rfloor + 1}^{n-1} kP \geq 2\sum_{k=1}^{\lfloor n/2 \rfloor - 1} kP$.

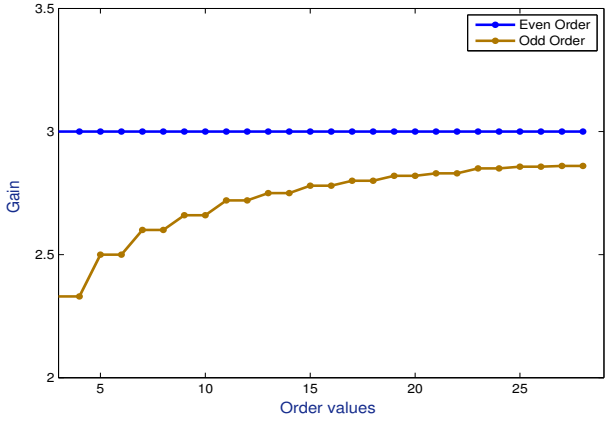

Figure 3. Gain between even order and odd order

From Figure 3, we can see that if the order *n* is odd, the line *y=3* is a horizontal asymptote for the gain curve.

To achieve a good security level of ECC, the scalar k will be chosen in the interval [0, n-1], and should be coded in 160 bits according to NIST- 192 recommended parameters.

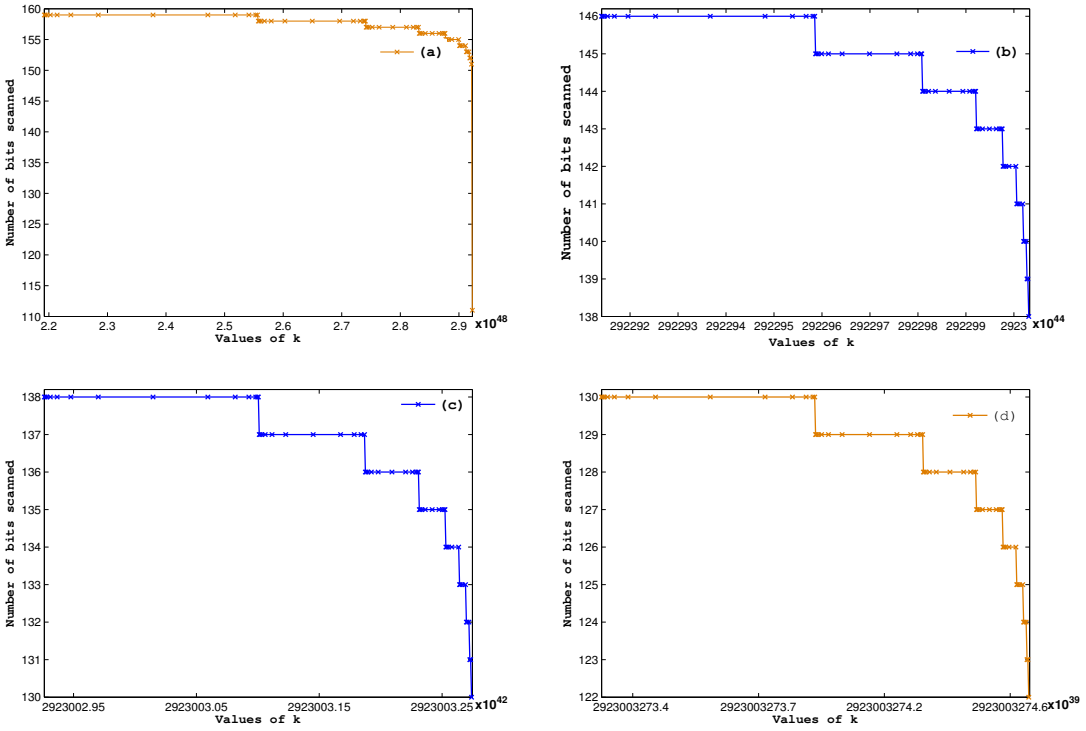

Figure 4. The number of bits scanned depending upon the values of k

For example: given an order n= $2^{161} \approx 2923.10^{44}$ coded in 161 bits, if k is the biggest scalar coded in 160 bits, k= $2^{160} \approx 1461510^{44}$ and k $\in$[0, 2^{161}]. Note that one half of the values of interval [0, 2^{161}-1] are in the interval [0, 2^{160}[and the other half are in [2^{160}, 2^{161}]. In the case of ECC, k $\in$[2^{160}, 2^{161}-1], and in this interval there are 14615.10^{44} values.

Using our scalar reduction method, we can examine closely the number of bits to scan for some values of the scalar k in the interval [2^{160}, 2^{161}-1]. The results are given in Table I.

Table I. Number of bits to scan and gain for some values of scalar k

Values of k	$2^{161} - 2^{159}$	$2^{161} - 2^{146}$	$2^{161} - 2^{110}$	$2^{161} - 2^{10}$	$2^{161} - 2$
Bits scanned	159bits	146bits	110bits	10bits	1bit
Gain	1bits	14bits	50bits	150bits	159bits

Figure (4-a) shows the number of bits to scan based on a set of values of k coded with the same number of bits. We can see significant evolution for scalars $> 2^{161} - 2^{146} \approx 29229.10^{44}$. In this interval there are 44601.10^{44} scalars. From Figures (4-b), (4-c) and (d), we can see more details after 146 bits scanned. Table II shows the gain expressed in number of bits and the corresponding percentage. We can see that, 6.25% of scalars have a gain $\geq$ 4 bits. Note that there are less and less values in intervals where the gain of bits is more important. To choose a scalar k, we can use a random number generator function based on a non-uniform probability distribution per interval.

Thus for an interval where the gain of bits is important, the scalars are generated by high-probability functions.

Table II. Gain (bits) and its corresponding scalar percentage

Gain (bits)	1bit	2bits	3 bits	4bits	5bits	6bits	$\geq$7bits
% of k values	50%	25%	12,5%	6,25%	3,125%	1,5625%	1,5625%

4.4. Performance Evaluation

We have implemented a simulator in Java to test the performance of the scalar reduction technique on an elliptic curve over $\mathbb{F}_p$ using NIST-192 recommended parameters which are given in Table III. $P(x_P, y_P)$ is the generator point with order n. For implementation, we have chosen 6 values of 192

Table III. NIST-192 recommended elliptic curve parameters

Parameter	**NIST-192 recommended values**
p	$2^{192} - 2^{64} - 1$
a	-3
b	0x64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1
x_P	0x188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012
y_P	0x07192b95ffc8da78631011ed6b24cdd573f977a11e794811
n	0xffffffffffffffffffffffff99def836146bc9b1b4d22831

bits for k which are distributed uniformly in $]0, n-1]$ (see Table IV).

Under these assumptions, we have tested the scalar reduction method using affine and Jacobian coordinates. The scalars are in binary and NAF form combined with the Scalar Reduction approach. The results are given in Tables V and VI and illustrated in Figure 5 and 6.

Table IV. Values of k chosen for performance evaluation

k	**Values in hexadecimal**
$\frac{n}{6}$	0x 2aaaaaaa aaaaaaaa aaaaaaaa 99a5295e 58bca19d 9e2306b2
$\frac{n}{3}$	0x555555555555555555555555334a52bcb179433b3c460d65
$\frac{n}{2}$	0x7fffffffffffffffffffffccef7c1b0a35e4d8da691418
$\frac{2n}{3}$	0xaaaaaaaaaaaaaaaaaaaaaaaa6694a57962f28676788c1aca
$\frac{5n}{6}$	0xd555555555555555555550039ced7bbaf281416af217a
n-1	0xffffffffffffffffffffff99def836146bc9b1b4d22830

Table V. Running times (ms) using affine coordinates (SR: Scalar reduction)

NAF	SR	DA	Gain	$\frac{n}{6}$	$\frac{n}{3}$	$\frac{n}{2}$	$\frac{2n}{3}$	$\frac{5n}{6}$	n-1
		$\surd$		6579	6572	7604	6555	6931	7471
$\surd$				6282	6326	5317	6239	6698	5114
	$\surd$			6578	6573	7600	6416	6600	27
$\surd$	$\surd$			6279	6325	5320	6100	6556	27
	$\surd$	$\surd$	$\alpha_{(sr/da)}$				2,12%	4,77%	99,63%
$\surd$	$\surd$		$\alpha_{(sr/naf)}$					1,46%	99,47%
$\surd$	$\surd$	$\surd$	$\alpha_{(sr-naf/da)}$				6,94%	5,41%	99,63%

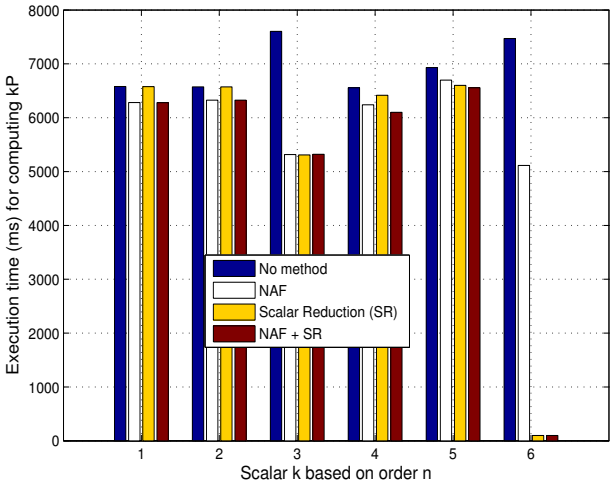

Figure 5. Running times (ms) using affine coordinates

Table VI. Running times (ms) in jacobian coordinates (SR: Scalar reduction)

NAF	SR	DA	Gain	$\frac{n}{6}$	$\frac{n}{3}$	$\frac{n}{2}$	$\frac{2n}{3}$	$\frac{5n}{6}$	n-1
		√		3066	3102	3621	3072	3202	3520
√				3053	3074	3592	3071	3189	3541
	√			3070	3100	3622	3030	3107	9
√	√			3050	3075	3597	3029	3094	9
	√	√	$\alpha_{(sr/da)}$				1,36%	2,96%	99,74%
√	√		$\alpha_{(sr/naf)}$				1,33%	2,57%	99,74%
√	√	√	$\alpha_{(sr-naf/da)}$				1,39%	3,37%	99,74%

As shown in results of performance evaluation respectively in Tables V and VI, we can notice that when $k \in]0, \frac{n}{2}]$, the proposed SR is not better solution. However, when $k \in]\frac{n}{2}, n-1]$, the computation task can be simplified and carried out more quickly in SR. In Jacobian coordinates, as we don't need to repeat the modular inverse, computation strongly run faster than case in affine coordinates as illustrated in Figure 5 and 6. The acceleration rate strongly depends on scalar value of k. We can see that the computation in NAF form is faster than the one in binary form. We can notice that in both cases, if $k \in]\frac{n}{2}, n-1]$, scalar reduction accelerate slightly the computation. Especially if k is close to $n-1$, the computation can be done instantaneously since $(n-1)P = -P$.

The gain rate depends on the value of k. For comparison, we define $(\alpha_{rs/da})$, $(\alpha_{rs/naf})$, $(\alpha_{rs-naf/da})$ respectively as the gain rate of the Scalar Reduction (SR) method compared to Double-and-Add (DA), NAF and SR combined with NAF compared to DA. If T_{da}, T_{naf}, T_{rs} and T_{rs-naf} respectively are computation time for the binary method (DA), NAF method, and Scalar Reduction (RS) combined with the NAF, then the percentage of the gain in terms of computation time for an algorithm *x* compared to an algorithm *y* can be expressed as:

$$\alpha_{x/y} = \frac{(T_y - T_x) * 100}{T_x} \tag{12}$$

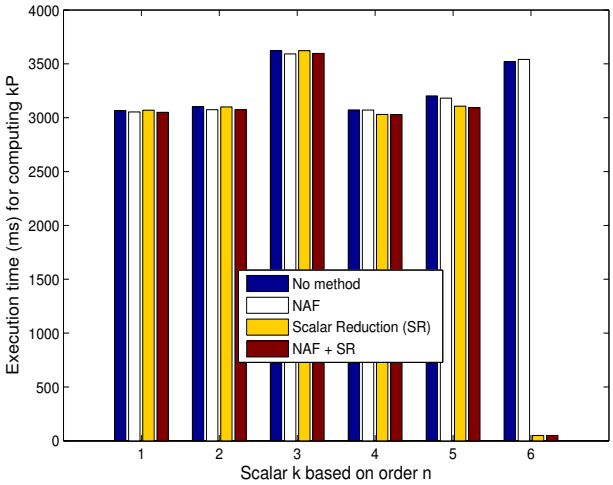

Figure 6. Running times (ms) using Jacobian coordinates

The results are given in Tables V and VI, and we can notice that there is no gain in the interval $]0, \frac{n}{2}]$. However, we can see that the gain rate of SR combined with NAF compared to DA is the most significant and approaches 100 % as *k* tends to *(n-1)*. Thus the SR method keeps its performances when combined with others schemes.

5. EFFICIENT PRECOMPUTATION ALGORITHMS

In this section, after a significant reduction of the scalar *k*, we present a fast precomputation algorithm in a parallel scalar multiplication on *kP* to accelerate computation. Most existing methods based on precomputation require some extra memory. Our goal is to avoid the storage of precomputation points. To deal with this, we propose to accelerate the precomputation phase. Our method is an improvement of the Double-and-Add algorithm. This method involves the level of scalar arithmetic and point arithmetic in Jacobian coordinates.

5.1. Partitioning scalar

To make a parallel computation of *k*P between *N* sensor nodes, we can split the scalar *k* to *m* blocks of length *v* = *k/N* bits, and each block is computed by one sensor node. We also split the optimization algorithm (Double-and-Add, NAF, etc.) to *m* blocks and each block m_i of the algorithm operates on one block m_i of the scalar. Algorithm 1 and Algorithm 2 show respectively block *i* for Double-and-Add and NAF algorithms.

Algorithm 1 Double-And-Add for node *i*

input : d= $(d_{v-1},.....,d_1,d_0)_2$, P $\in E(\mathbb{F}_p)$

output: dP

begin

$Q \longleftarrow \infty$ **for** $j \leftarrow 0$ **to** $v-1$ **do**

// begin scan from right to left step by step

if d_j=*1* **then**

$Q \longleftarrow Q + 2^{vi}P$ // 2^{vi} is the pre-computed point

$P \longleftarrow 2P$

Return(Q)

Algorithm 2 NAF for node *i*

input : NAF(d)=$(d_{v-1},.....,d_1,d_0)_2$, P $\in E(\mathbb{F}_p)$

output: dP

begin

$Q \longleftarrow \infty$ **for** $j \longleftarrow 0$ **to** $v-1$ **do**

// begin scan from right to left step by step

$P \longleftarrow 2Q$

if d_j=*1* **then**

$Q \longleftarrow Q + 2^{vi}P$ // $2^{vi}P$ is the pre-computed point

if d_j=*-1* **then**

$Q \longleftarrow Q - 2^{vi}P$ // $2^{vi}P$ is the pre-computed point

Return(Q)

5.2. Reliability of partitioning scalar

After partitioning the scalar *k* to *m* blocks of length *v*, the node which leads computation copies one of the *m* blocks into its local memory and distributes (*m-1*)blocks to others nodes. In this case, one possibility is to send the (*m-1*) blocks in secure mode by symmetric encryption. If they are sent randomly in clear mode, the intruder, after gaining (*m-1*) blocks of the m blocks, will compute *(m!*2^v*)P* to find scalar *k*. Moreover, if the intruder gains the (*m-1*) results sent by other nodes, security is not compromised. It is as difficult to find *k* from *kP* as *k* from the (*m-1*) points resulting from computation scalar multiplication on (*m-1*) blocks. For each block d_iP, the intruder needs to find d_i. After that, it also needs to compute *(m!*2^v*)P* before getting scalar *k*.

5.3. Accelerating precomputation points

In the case of parallel scalar multiplication on *kP*, precomputation points are in the form *dP*=$2^{vi}P$ for the block *i*, where *d*=2^{vi}. From (*vi+1*) bits representating the integer 2^{vi}, only the most significant bit is 1; the last significant vi bits are equal to 0. In the rest of this paper, notations used to specify the computation time (or computation cost) are listed below: A (addition or subtraction), M (multiplication), S (field squaring) and I(inversion). It is widely accepted that *1S = 0.6M* or *1S*

= *0.8M* and *1S* = *1M* [7], [12], [19]. In Jacobian coordinates, doubling is cheaper than adding [20]; in the general case, the cost of doubling is *8M+3S* and the cost of adding is *12M+ 4S*. We can only use point doubling operations since the Hamming weight is next to nothing in pre-computed points. The concept of the direct computation of several repeated doublings was first suggested in [15] on elliptic curves over $\mathbb{F}_{2^n}$ in affine coordinates. This concept increases the speed of repeated doubling by computing directly 2^xP (with x integer) from P $\in$ $E(\mathbb{F}_p)$ without computing the intermediate points $2P$, 2^2P, ..., $2^{x-1}P$ [23]. However, the known formula works only with small x (2, 3 or 4). Paper [26] gives efficient formulas to compute directly 2^xP from $P \in E(\mathbb{F}_p)$ for all k>=1. The existing formulas for doubling, quadrupling and computing 2^xP repeated doubling in the Jacobian coordinate system are defined follows:

$$\begin{cases} \alpha_1 = 3X_1^2 + aZ_1^4 \\ \beta_1 = 4X_1Y_1^2 \\ \gamma_1 = 8Y_1^4 \\ X_2 = \alpha^2 - 2\beta \\ Y_2 = \alpha(\beta - X_2) - \gamma \\ Z_2 = 2Y_1Z_1 \end{cases} \quad \textit{Doubling} \tag{13}$$

$$\begin{cases} \alpha = 3X_1^2 + aZ_1^4 \\ \beta = \alpha^2 - 8X_1Y_1^2 \\ \gamma = -8Y_1^4 + \alpha(12X_1Y_1^2 - \alpha^2) \\ \omega = 16aY_1^4Z_1^4 + 3\beta^2 \\ X_4 = -8\beta\gamma^2 + \omega^2 \\ Y_4 = -8\gamma^4 + \omega(12\beta\gamma^2 - \omega^2) \\ Z_4 = 4Y_1Z_1\gamma \end{cases} \quad \textit{Quadrupling} \tag{14}$$

$$\begin{cases} \alpha_1 = X_1 \\ \beta_1 = 3X_1^2 + a \\ \gamma_1 = -Y_1 \\ For\ i\ from\ 2\ to\ x,\ compute\ \alpha_i,\ \beta_i,\ \gamma_i \\ \alpha_i = \beta_{i-1}^2 - 8\alpha_{i-1}\gamma_{i-1}^2 \\ \beta_i = 3\alpha_i^2 + 16^{i-1}a(\prod_{j=1}^{i-1}\gamma_j)^4 \\ \gamma_i = -8\gamma_{i-1}^4 - \beta_{i-1}(\alpha_i - 4\alpha_{i-1}\gamma_{i-1}^2) \\ \omega_x = 12\alpha_x\gamma_x^2 - \beta_x^2 \\ X_{2^x} = \beta_x^2 - 8\alpha_x\gamma_x^2 \\ Y_{2^x} = 8\gamma_x^4 - \beta_x\omega_x \\ Z_{2^x} = 2x\prod_{i=1}^{x}\gamma_i \end{cases} \quad 2^x\text{-}uple \tag{15}$$

In our proposed algorithm, namely Double-and-Double, from $(vi+1)$ bits representing the integer 2^{vi},we scan the vi bits from left to right and only use repeated doubling to replace addition in the Double-and-Add algorithm by doubling. To accelerate computation, we can optimize by using the existing formula of quadruple operation of a point [15] and build a new algorithm named Quadruple-and-Quadruple. We can also make a generalization of this algorithm by using the formula of 2^x*-uple* point where *x* is an integer. From Algorithm 3, Algorithm 4 and Algorithm 5, we can see respectively our proposed *Double-and-Double* algorithm, *Quadruple-and-Quadruple* algorithm and the general 2^x*-uple-and-*2^x*-uple* algorithm.

Algorithm 3 Double-And-Double for node *i*

input : d=2^v=(d_v,d_{v-1},.....,d_1,d_0)$_2$, P $\in$ $E(\mathbb{F}_p)$

output: dP

begin

$Q \longleftarrow P$ **for** $j \leftarrow v-1$**to** 0 **do**

// begin scanning on the (v-1)th bit with single step

$Q \longleftarrow 2Q$ // doubling operation

Return(Q)

5.4. Overhead comparisons and performance evaluation

[20] shows that the total cost to compute *w* consecutive doublings is *4wM+2(2w+1)S*. Theoretically, the efficiency of the formula using Jacobian coordinates can be determined by the number of multiplication (M) and of square (S) operations which compose it. Recall that operations like addition, subtraction and multiplication with a constant are negligible when faced with square and multiplication of 2 variables. Table VII shows the cost of each formula.

Algorithm 4 Quadruple-and-Quadruple for node *i*

input : d=2^v=(d_v,d_{v-1},.....,d_1,d_0)$_2$, P $\in$ $E(\mathbb{F}_p)$
output: dP

```
begin
    Q ⟵ P
    j ⟵ v
    repeat
        j ⟵ j − 2   // begin scanning on the (v-1)-th bit with steps of
        two bits
        if j<0 then
            Q ⟵ 2Q                                   // doubling operation
        else
            Q ⟵ 4Q                                 // quadrupling operation
    until j <=0
    Return(Q)
```

Algorithm 5 2^x-uple-and-2^x-uple for node *i*

] **input** : d=2^v=(d_v,d_{v-1},.....,d_1,d_0)$_2$, P $\in$ $E(\mathbb{F}_p)$
output: dP

```
begin
    Q ⟵ P
    j ⟵ v   // begin scanning on the (v-1)th bit with step of x-bits
    repeat
        if j<x then
            Q ⟵ 2^j Q                              // 2^j-upling operation
        else
            Q ⟵ 2^x Q                              // 2^x-upling operation
        j ⟵ j − x
    until j <=0
    Return(Q)
```

Table VII. Cost of each formula

Formulas	Number of operations
Double	4M + 4S
Quadruple	9M + 10S
2^x-uple(x>=2)	6M + 8S +(x-2)(5M + 5S)

Generally speaking, we assume that the cost of square is equivalent to 0.6 to 0.8 of the cost of multiplication. Hence, for a scalar multiplication with a scalar of length of *n* bits, we can determine the ratio (*r*=S/M) from which each formula can justify the best efficiency. As we can see in figure 7, the formula of quadruple is more efficient with the ratio $r >=0.5$, and the ratio of 2^x-uple is more efficient than the one of double when $r = 9.5$ approximately.

We use a Java simulator with recommended parameters found in Table III (see section 4.5) to test the performance of our method on an elliptic curve over $\mathbb{F}_p$ using NIST- 192. The length of

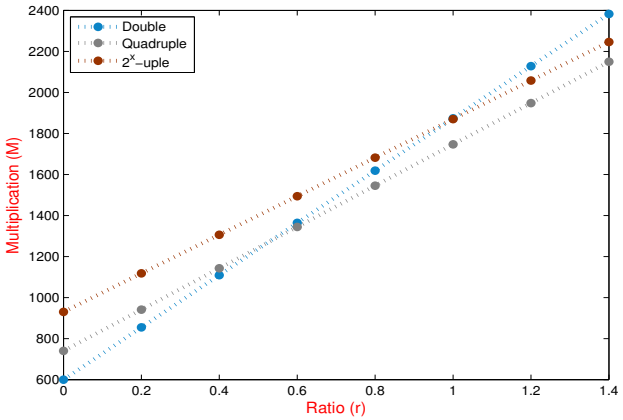

Figure 7. Number of multiplications based on ratio r = S/M

Table VIII. Running times (ms) using Jacobian coordinates

Method	**1 Processor**	**2 Processors**	**3 Processors**	**4 Processors**
Double	3508	1690	1111	841
2^3-uple	4032	2016	1344	1008
Quadruple	3141	1504	995	744
2^4-uple	3744	1872	1248	936
2^x-uple	17435	4779	2400	1506

the scalar used is 192 bits; the program is first run on one processor. Then we perform parallel computing using 2, 3 and 4 processors. The test results in Jacobian coordinates are given in Table VIII.

Theoretically, the 2^x-uple formula seems to be very promising, but we can see that the quadruple formula is the most efficient in both cases.

The theoretical efficiency of our scanning algorithm seems to depend on the step-size of every iteration and on all the elemental arithmetic operations (addition, subtraction, multiplication and reverse in finite fields) which compose it. If the step-size is much larger, our scanning algorithm will be more efficient. For a scalar of d_i=2^{vi}, the objective is to find an efficient formula which can perform scanning using a step-size greater or equal to vi. Thus, the main difficulty is to find a formula with a big step-size.

6. CONCLUSION

This paper presents a new technique to run quickly the computation of scalar multiplication ECC. This approach, based on the negative and order of points reduces computation time in the interval $[\lfloor n/2 \rfloor+1, n\text{-}1]$. We show that the usage of an even order is more efficient. This technique can be easily applied to almost all existing fast scalar multiplication methods (as shown in NAF) and is suitable for use in embedded devices such as WSN. Through simulations based on evaluations, we show that the proposed solution does accelerate the computation of scalar multiplication on NIST-192 parameters for ECC. We obtain a gain of 4 bits in 161 bits for 6.25% of the scalars. This

gain is greater than 7 bits for 1.5625% of the scalars and can sometimes reach 100% (160bits) in some cases. In our second contribution, we propose efficient precomputation algorithms to avoid storage of precomputation, which requires extra memory in sensor networks. We show that our improvement method based on the Double-and-Add algorithm is very efficient, especially when it is used in Jacobian coordinates. Even the 2^x-uple formula seems to be very promising theoretically; through simulation, we show that the quadruple formula is the most efficient. Note that, we use existing formulas in our scanning algorithm. Our future research could be to use our approach on real sensor nodes with elliptic curves over finite prime fields using recommended parameters to maintain a good level of security. We will also discover a new efficient formula for 2^x-uple which seems to be very promising.

REFERENCES

1. Walters, J. P., Liang, Z. Q., Shi W. S., & al., (2007). Wireless sensor network security: A survey. Security in distributed, grid, mobile, and pervasive computing, Auerbach Publications, 1-50.
2. Zhou, Y., Fang, Y., & Zhang, Y. (2008). Securing wireless sensor networks: a survey. IEEE Communications Surveys and Tutorials, 10(3), 6-28.
3. Gura, N., Patel, A., Wander, A., Eberle, H., & Shantz, S. C. (2004, August). Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In International Workshop on Cryptographic Hardware and Embedded Systems, 119-132, Springer.
4. Düll, M. and Haase, B. & Hinterwälder, G., Hutter, M., Paar, C., Snchez, A. H., and Schwabe, P. (2015). High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. Designs, Codes and Cryptography, 77(2-3), 493-514.
5. Ansari, B., & Wu, H. (2005, May). Parallel scalar multiplication for elliptic curve cryptosystems. In Proceedings 2005 International Conference on Communications, Circuits and Systems, 2005. (Vol. 1, 71-73) IEEE.
6. Blake, I. F., Seroussi, G., and Smart, N. (1999). Elliptic curves in cryptography (Vol. 265). Cambridge university press.
7. D. Bernstein, D. Hankerson, J. López & A. Menezes. (2001, April). Software implementation of the NIST elliptic curves over prime fields. In Cryptographers Track at the RSA Conference, 250-265. Springer.
8. Brickell, E. F., Gordon, D. M., McCurley, K. S., & Wilson, D. B. (1992, May). Fast exponentiation with precomputation. In Workshop on the Theory and Application of of Cryptographic Techniques, 200-207. Springer.
9. Ciet, M., Joye, M., Lauter, K., & Montgomery, P. L. (2006). Trading inversions for multiplications in elliptic curve cryptography. Designs, codes and cryptography, 39(2), 189-206.
10. Dimitrov, V., Imbert, L., & Mishra, P. K. (2005, December). Efficient and secure elliptic curve point multiplication using double-base chains. In International Conference on the Theory and Application of Cryptology and Information Security, 59-78. Springer.
11. Gordon, D. M. (1998). A survey of fast exponentiation methods. Journal of algorithms, 27(1), 129-146.
12. Großschädl, J., Avanzi, R. M., Sava?, E., & Tillich, S. (2005, August). Energy-efficient software implementation of long integer modular arithmetic. In International Workshop on Cryptographic Hardware and Embedded Systems, 75-90. Springer.
13. Bernstein, D. J., Lange, T., & Schwabe, P. (2011, March). On the correct use of the negation map in the Pollard rho method. In International Workshop on Public Key Cryptography, 128-146. Springer.
14. Morain, F., & Olivos, J. (1990). Speeding up the computations on an elliptic curve using addition-subtraction chains. Informatique theorique et Applications, 24(6), 531-543.
15. Guajardo J. & Paarf C.. Efficient algorithms for elliptic curve cryptosystems, Cryptology CRYPTO97, LNCS, 1294, 342–356, Springer, (1997)
16. Hankerson, D., Menezes, A. J., & Vanstone, S. (2006). Guide to elliptic curve cryptography. Springer Science & Business Media.
17. Robshaw, M. J. B., and Yin, Y. L. (1997). Elliptic curve cryptosystems. An RSA Laboratories Technical Note, 1, 997.
18. Lim, C. H., & Lee, P. J. (1994, August). More flexible exponentiation with precomputation. In Annual International Cryptology Conference (pp. 95-107). Springer.

19. Lim, C. H., & Hwang, H. S. (2000, January). Fast implementation of elliptic curve arithmetic in GF (p n). In International Workshop on Public Key Cryptography, 405-421. Springer.
20. Longa, P., & Miri, A. (2008). Fast and flexible elliptic curve point arithmetic over prime fields. IEEE Transactions on computers, 57(3), 289-302.
21. Meloni, N., & Hasan, M. A. (2009). Elliptic curve scalar multiplication combining yaos algorithm and double bases. In Cryptographic Hardware and Embedded Systems-CHES 2009, 304-316. Springer.
22. Miyaji, A., Ono, T., & Cohen, H. (1997). Efficient elliptic curve exponentiation. Information and Communications Security, 282-290.
23. Müller, V. (1998). Efficient algorithms for multiplication on elliptic curves. In Chipkarten (pp. 135-145). Vieweg+ Teubner Verlag.
24. Panda, B., & Khilar, P. M. (2011, February). Fpga based implementation of parallel ecc processor. In Proceedings of the 2011 International Conference on Communication, Computing & Security, 453-45. ACM.
25. Purnaprajna, M., Puttmann, C., & Porrmann, M. (2008, March). Power aware reconfigurable multiprocessor for elliptic curve cryptography. In 2008 Design, Automation and Test in Europe, 1462-1467. IEEE.
26. Sakai, Y., & Sakurai, K. (2000, July). Efficient scalar multiplications on elliptic curves without repeated doublings and their practical performance. In Australasian Conference on Information Security and Privacy (pp. 59-73). Springer Berlin Heidelberg.
27. Shou, Y., Guyennet, H., & Lehsaini, M. (2013, January). Parallel scalar multiplication on elliptic curves in wireless sensor networks. In International Conference on Distributed Computing and Networking, 300-314. Springer.
28. Suppakitpaisarn, V., Imai, H., & Masato, E. (2012, June). Fastest multi-scalar multiplication based on optimal double-base chains. In Internet Security (WorldCIS), 2012 World Congress on 93-98. IEEE.
29. Tian, M., Wang, J., Wang, Y., & Xu, S. (2010, April). An efficient elliptic curve scalar multiplication algorithm suitable for wireless network. In 2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing, 95–98.
30. Wu, K., Li, D., Li, H., Chen, T., & Yu, F. (2009, December). Partitioned Computation to Accelerate Scalar Multiplication for Elliptic Curve Cryptosystems. In 15th International Conference on Parallel and Distributed Systems (ICPADS), 551-555, IEEE.
31. Faye, Y., Guyennet, H., Niang, I., & Shou, Y. (2013). Fast scalar multiplication on elliptic curve cryptography in selected intervals suitable for wireless sensor networks. In Cyberspace Safety and Security, 171-182. Springer International Publishing.
32. Al-Somani, T. F. (2015). Highly Efficient Generic-Point Parallel Scalar Multiplication using Concurrent Precomputations. Journal of Applied Sciences, 15(10), 1261.
33. Al-Somani, T. F. (2016). High-Performance Generic-Point Parallel Scalar Multiplication. Arabian Journal for Science and Engineering, 1-6.
34. Renes, J., Costello, C., & Batina, L. (2016, May). Complete addition formulas for prime order elliptic curves. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, 403-428. Springer.
35. Bos, J. W., Costello, C., Longa, P., & Naehrig, M. (2015). Selecting elliptic curves for cryptography: An efficiency and security analysis. Journal of Cryptographic Engineering, 1-28.