Hindawi
Security and Communication Networks
Volume 2020, Article ID 4087873, 10 pages
https://doi.org/10.1155/2020/4087873

Research Article

Parallel and Regular Algorithm of Elliptic Curve Scalar Multiplication over Binary Fields

Xingran Li[1,2,3] **Wei Yu**[1,3] **and Bao Li**[1,2]

[1]*State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing 100093, China*
[2]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China*
[3]*Data Assurances and Communications Security, Institute of Information Engineering, CAS, Beijing 100093, China*

Correspondence should be addressed to Wei Yu; yuwei_1_yw@163.com

Received 17 October 2019; Accepted 8 May 2020; Published 24 June 2020

Academic Editor: Benjamin Aziz

Accelerating scalar multiplication has always been a significant topic when people talk about the elliptic curve cryptosystem. Many approaches have been come up with to achieve this aim. An interesting perspective is that computers nowadays usually have multicore processors which could be used to do cryptographic computations in parallel style. Inspired by this idea, we present a new parallel and efficient algorithm to speed up scalar multiplication. First, we introduce a new regular halve-and-add method which is very efficient by utilizing λ projective coordinate. Then, we compare many different algorithms calculating double-and-add and halve-and-add. Finally, we combine the best double-and-add and halve-and-add methods to get a new faster parallel algorithm which costs around 12.0% less than the previous best. Furthermore, our algorithm is regular without any dummy operations, so it naturally provides protection against simple side-channel attacks.

1. Introduction

The elliptic curve was first imported into the world of cryptography by Neal Koblitz and Victor Miller independently in 1985 [1, 2] and is now increasingly used for a wide range of cryptography primitives in practice such as public encryption and digital signature. More than 30 years after its introduction to the cryptography field, the practical advantages of elliptic curve cryptosystem (ECC) are clear and well-known: it has richer algebraic structures, a smaller key size, and relatively faster implementations to achieve the same level of security compared with other deployed schemes such as RSA. Based on the above benefits, ECC is particularly suitable for resource-constrained devices.

The efficiency of ECC is dominated by the speed of calculating scalar multiplication. Namely, given a rational point P of order r on elliptic curves, it requires to compute $kP = P + P + \cdots + P$ (k times), for a given scalar $k \in [0, r)$. Obviously, there are similar features between scalar multiplication and exponentiation in a general multiplicative finite group. Therefore, inspired by the repeated "square-and-multiply" algorithm, the normally used binary method called "double-and-add" for scalar multiplication over elliptic curves has been regarded as a fundamental technique.

In constrained environments, scalar multiplication is easily implemented by "double-and-add" variant of Horner's rule, providing binary expansion of scalar $k = \sum_{i=0}^{l} k_i \cdot 2^i$. However, each bit of k implies different algorithmic path during each iteration, that is, if $k_i = 0$, only a point doubling is necessary. Whereas if $k_i = 1$, a point doubling followed by a point addition is involved. As a consequence, different power and time consumption of this two prominent building blocks can be detected by simple power analysis (SPA) [3] and timing attack—this naive implementation leads to information leakage of secret scalar k.

Protecting against simple side-channel attacks (SSCA) can be achieved by recoding scalars in a regular manner, meaning that scalar multiplications are executed in the same instructions in the same order for any input value. Coron introduced a countermeasure against SSCA named "double-and-add always" algorithm [4]. By inserting a dummy operation when necessary, it evaluates scalar multiplication by executing a doubling and an addition in each loop. However, it was soon found to be vulnerable to safe-error fault attacks [5, 6]. By timely inducing a fault at one iteration during the point addition, an adversary can determine whether the operation is dummy or not by checking the correctness of the output.

A measurement against safe-error fault attacks performs scalar multiplication in a predictable pattern. Besides the most commonly used Montgomery-ladder algorithm [7], another efficient method is m-ary recoding [8]. This algorithm recodes a scalar in a sequence of $m-1$ zeros and a nonzero with the percentage of nonzero numbers $(1/m)$. However, scanning from look-up table could be dangerous if this step cannot be proceeded in constant-time.

Another increased interest-focused field of regular executing scalar multiplication is exploiting efficient curve forms that allow complete addition law. For any pair of k-rational points on elliptic curves (or in desired subgroup), complete addition law can compute the correct result, ignoring whether two addends are identical or not. As a corollary of the main results in [9], elliptic curves embedded in any projective spaces of dimension n by a symmetric line bundle admit a complete system of addition laws of bidegree $(2,2)$. The later work of Bosma and Lenstra [10] shows that, when suitably chosen, a single addition law is able to act as add operation for all pairs of k-rational elliptic points. One of the well-studied examples is Edwards curves [11, 12], of which exceptional pairs for addition law exist outside k-rational points. A recent work [13] proposed an optimized algorithm that adds any pair of k-rational points for prime order elliptic curves defined over field of characteristic different from 2 and 3.

In [14], the authors introduce a new approach for scalar multiplication called Montgomery-halving algorithm which is a variation of the original Montgomery-ladder point multiplication. Besides, they present a new strategy for parallel implementation of point multiplication over elliptic curves by running the Montgomery-halving algorithm with the original Montgomery-ladder algorithm in parallel to calculate scalar multiplication concurrently. Moreover, this parallel algorithm can achieve protecting against SSCA. However, in their scheme, affine coordinate has to be used for halving, because the projective form of the Montgomery-halving algorithm could not be used to save operations.

In this paper, we provide a similar parallel implementation method using regular recoding technique which should be highly efficient by parallel processing doubling and halving operations in two different coprocessors. It can be concluded as two main contributions.

The first contribution is that we give a new regular algorithm computing halving operation called zero-less signed-digit (ZSD) halve-and-add which saves around 32.7% and 33.0% cost compared with Montgomery-halving method in [14] with $m = 233$ and $m = 409$. The λ projective coordinate system could offer projective coordinates saving inversions. This is especially useful for our ZSD halve-and-add algorithm (Algorithm 1). For halving operation, the best coordinate is λ affine coordinate. For the following addition operation, the better choice is λ projective coordinate. The Montgomery-halving algorithm in [14] has to exploit affine coordinate for its special structure without other choices, while our Algorithm 1 could make use of λ projective coordinate for its different structure design, where R_1 can always be in λ affine coordinate for halving and R_0 can always be in λ projective coordinate for addition so that λ projective mixed addition law could be used and no more coordinate transformation needed. In addition, the regular recoding technique ensures the secure implementation of scalar multiplication against SSCA.

The second contribution concerns the new mixed-parallel algorithm. After analyzing all the algorithms in Table 1, we combine the fastest double-and-add method and Montgomery double-and-add method, in [14], and the fastest halve-and-add method, our ZSD halve-and-add algorithm, in this paper. A new efficient and secure mixed-parallel algorithm just comes into being, the mixed-parallel method, which costs around 11.7% and 12.0% less than Montgomery-Parallel approach in [14] when $m = 233$ and $m = 409$, respectively. The more thorough analysis will be exhibited in Section 4, and the related estimate results are all displayed in Tables 1 and 2.

The rest of this paper is organized as follows. In the next section, we introduce the related arithmetic knowledge of binary elliptic curves, especially on efficient λ coordinate point representation, twisted μ_4-normal form, and how to evaluate scalar multiplication in parallel by combining point halving and doubling operations. In Section 3, our new regular algorithm for halve-and-add is provided. Moreover, a similar parallel strategy as the one detailed in [14] shows how to efficiently implement scalar multiplication in a regular and parallel manner. Cost comparison and expected performance analysis are presented in Section 4. Finally, we conclude this paper and give the new mixed-parallel algorithm after analyzing.

2. Preliminaries

We focus on elliptic curves E defined over binary fields $\mathscr{F}_{2^m}$, by the Weierstrass equation:

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

where $a, b \in \mathscr{F}_{2^m} = \mathscr{F}_2[t]/(f(t))$, $f(t) \in \mathscr{F}_2[t]$ is an irreducible polynomial of degree m. Isomorphic to the divisor class group of degree 0, the rational points $P(x, y)$ on E together with the point at infinity $\mathcal{O}$ form an abelian group, of which the basic group operation—addition—is algebraically interpreted by the tangent-and-chord law.

Given two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on $E(\mathscr{F}_{2^m})$, where $P_1 = \pm P_2$, if the addition of the two points is presented by $Q = P_1 + P_2$, then the coordinates of $Q = (x_3, y_3)$ can be computed according to the following formula:

$$\begin{cases} x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \\ y_3 = \lambda(x_1 + x_3) + x_3 + y_1, \end{cases} \tag{2}$$

with $\lambda = (y_1 + y_2)/(x_1 + x_2)$.

```
    Input: $P \in E(\mathcal{F}_{2^m})$ of odd order $r$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$ with $k_{n-1} = 1$
    Output: $2^{-n}k \cdot P$
(1) $R_0 \longleftarrow P/2$; $R_1 \leftarrow P/2$
(2) For $i = n-1$ down to 1 do
(3)     $t \longleftarrow (-1)^{1+k_i}$
(4)     $R_1 \longleftarrow R_1/2$; $R_0 \leftarrow R_0 + t \cdot R_1$
(5) End for
(6)   $R_{k_0} \longleftarrow R_{k_0} - R_{1-k_0}$
(7) Return $R_0$
```

Algorithm 1: Regular ZSD halve-and-add (left-to-right) method.

Table 1: Complexity comparison for m-bit k in different single algorithms.

Method	Point operations	Field operations ($I = 10M$)	$m = 233$	$m = 409$
Montgomery-D	$mD_M + mA_M$	$6mM + 10M + \mathrm{I}$	$1418M$	$2474M$
Montgomery-H	$mH_a + mA_a$	$3mM + m\ (2M + \mathrm{I})$	$3495M$	$6135M$
Algorithm 2-D (λ-projective)	$mD_{\lambda_P} + mA_{\lambda_P}$	$10mM + 11M + \mathrm{I}$	$2351M$	$4111M$
Algorithm 2-D (twisted μ_4)	$mD_{\mu_4} + mA_{\mu_4}$	$9mM + 11M + \mathrm{I}$	$2118M$	$3702M$
Algorithm 1-H	$mH_{\lambda_a} + mA_{\lambda_P}$	$10mM + 11M + \mathrm{I}$	$2351M$	$4111M$

Montgomery-D = Montgomery double-and-add algorithm, Montgomery-H = Montgomery halve-and-add algorithm, Algorithm. 2-D (λ-Projective) = Algorithm 2 using the λ-projective coordinate system, Algorithm. 2-D (twisted μ_4) = Algorithm 2 using the twisted μ_4 coordinate system, Algorithm. 1-H = Algorithm 1 for halve-and-add.

Table 2: Complexity comparison for m-bit k in different parallel algorithms.

Algorithm	Split (t)	Estimate ($m = 233$)	Split (t)	Estimate ($m = 409$)
Montgomery-parallel	67	$1028M$	118	$1782M$
Our mixed-parallel	87	$908M$	153	$1568M$

Similarly, given $P = (x_1, y_1) \in E(\mathcal{F}_{2^m})$, where $P \neq -P$, if the doubling of the point P is presented by $2 \cdot P$, then the coordinates of $2P = (x_3, y_3)$ can be computed according to the following formula [15]:

$$\begin{cases} x_3 = \lambda^2 + \lambda + a = x_1^2 + \dfrac{b}{x_1^2}, \\ \\ y_3 = x_1^2 + \lambda x_3 + x_3, \end{cases} \tag{3}$$

with $\lambda = x_1 + y_1/x_1$.

From the above formulas, it is easy to notice that there are inevitable inversion operations in the base field, which would consume much time. Usually, the projective coordinate system is more welcome for its inclusion of no field inversions. In practice, various kinds of coordinate systems are already available to be used. The work in this paper prefers to exploit the state-of-the-art coordinate systems: λ coordinate and the projective coordinate system of twisted μ_4-normal form. They perform excellently in different situations.

2.1. λ Coordinates. Efficient point representation is of great importance to accelerate scalar multiplication. Inversion in the base field takes a large amount of time; however, they are indispensable if points are represented in affine coordinate. The homogeneous projective coordinate system (also called standard projective coordinate system) is usually used to eliminate this obstacle by injecting any k-rational affine point $P(x, y) \in \mathbb{A}^2$ into one of its projective copies $(X, Y, Z) = (x, y, 1) \in \mathbb{P}^2$, where $x = X/Z, y = Y/Z, Z \neq 0$. When one of the projective copies $(X, Y, Z) = (x, y, 1)$ corresponds to the affine point (x, y), where $x = X/Z^2, y = Y/Z^3, Z \neq 0$, it is the Jacobian projective coordinate system. Later, López and Dahab proposed a new and efficient projective coordinate system. Compared with the above coordinate systems, the difference is $x = X/Z, y = Y/Z^2, Z \neq 0$ here [16], denoted as LD coordinate for short. Later, Kim and Kim presented a four-dimensional LD coordinate system for binary curves which represents P as (X, Y, Z, T), with $x = X/Z$, $y = Y/T$, $T = Z^2$, and $Z \neq 0$.

The λ coordinate system was firstly noticed by Knudsen [17] when studying halving operations on binary elliptic curves. Oliveira [18] further surveyed its comprehensive arithmetic. Given a point $P = (x, y) \in E(\mathcal{F}_{2^m})$ with $x = 0$, the λ affine representation of P is defined as (x, λ), where $\lambda = x + y/x$. So, it is easy to derive point addition and doubling formulas of points in λ affine coordinates from the normal affine ones. Let $P = (x_P, \lambda_P)$ and $Q = (x_Q, \lambda_Q)$ be two points on $E(\mathcal{F}_{2^m})$, where $P \neq \pm Q$, then the formula for $P + Q = (x_{P+Q}, \lambda_{P+Q})$ can be given by the following formula:

$$\begin{cases} x_{P+Q} = \dfrac{x_P \cdot x_Q}{\left(x_p + x_Q\right)^2}\left(\lambda_P + \lambda_Q\right), \\ \\ \lambda_{P+Q} = \dfrac{x_Q \cdot \left(x_{P+Q} + x_P\right)^2}{x_{P+Q} \cdot x_P} + \lambda_P + 1. \end{cases} \tag{4}$$

Referring to doubling operation, $2P = (x_{2P}, \lambda_{2P})$ is given as follows:

$$\begin{cases} x_{2P} = \lambda_P^2 + \lambda_P + a, \\ \lambda_{2P} = \dfrac{x_P^2}{x_{2P}} + \lambda_P^2 + a + 1. \end{cases} \quad (5)$$

As for projective conditions, the translation between affine representation (x, y) and λ projective representation (X, L, Z) is defined by $x = X/Z$, $\lambda = L/Z$ with $\lambda = x + y/x$. The negative element of (X, L, Z) is $(X, L + Z, Z)$. Assumed two points $P(X_P, L_P, Z_P)$ and $Q(X_Q, L_Q, Z_Q)$ represented in λ model on binary elliptic curves, similar to the affine case, the addition arithmetic could be described as the following formulas:

$$\begin{cases} A = L_P \cdot Z_Q + L_Q \cdot Z_P, \\ B = \left(X_P \cdot Z_Q + X_Q \cdot Z_P\right)^2, \\ X_{P+Q} = A \cdot \left(X_P \cdot Z_Q\right) \cdot \left(X_Q \cdot Z_P\right) \cdot A, \\ L_{P+Q} = \left(A \cdot \left(X_Q \cdot Z_P\right) + B\right)^2 + \left(A \cdot B \cdot Z_Q\right) \cdot \left(L_P + Z_P\right), \\ Z_{P+Q} = \left(A \cdot B \cdot Z_Q\right) \cdot Z_P, \end{cases} \quad (6)$$

and for $2P = (X_{2P}, L_{2P}, Z_{2P})$, it could be given as follows:

$$\begin{cases} T = L_P^2 + \left(L_P \cdot Z_P\right) + a \cdot Z_P^2, \\ X_{2P} = T^2, \\ Z_{2P} = T \cdot Z_P^2, \\ L_{2P} = \left(X_P \cdot Z_P\right)^2 + X_{2P} + T \cdot \left(L_P \cdot Z_P\right) + Z_{2P}. \end{cases} \quad (7)$$

The associated group addition $P + Q$ and doubling $2 \cdot P$ operations can be calculated by $11M + 2S$ and $3M + 4S$, respectively, where M denotes a field multiplication and S denotes a squaring.

Having the above formulas, a direct thought is to combine doubling and addition formulas to obtain a formula evaluating $2Q + P$, which is of great importance in the latter part of this paper.

Let $P = (x_P, \lambda_P)$ and $Q = (X_Q, L_Q, Z_Q)$ be points of $E(\mathcal{F}_{2^m})$, then $2Q + P = (X_{2Q+P}, L_{2Q+P}, Z_{2Q+P})$ can be computed as follows:

$$\begin{cases} T = L_Q^2 + \left(L_Q \cdot Z_Q\right) + a \cdot Z_Q^2, \\ A = X_Q^2 \cdot Z_Q^2 + T \cdot \left(L_Q^2 + \left(a + 1 + \lambda_P\right) \cdot Z_Q^2\right), \\ B = \left(x_P \cdot Z_Q^2 + T\right)^2, \\ X_{2Q+P} = \left(x_P \cdot Z_Q^2\right) \cdot A^2, \\ Z_{2Q+P} = \left(A \cdot B \cdot Z_Q^2\right), \\ L_{2Q+P} = T \cdot (A + B)^2 + \left(\lambda_P + 1\right) \cdot Z_{2Q+P}. \end{cases} \quad (8)$$

Using this, $2Q + P$ operations can be calculated efficiently by $10M + 6S$ instead of $11M + 2S + 3M + 4S = 14M + 6S$, where M denotes a field multiplication and S denotes a squaring [18].

2.2. Twisted μ_4-Normal Form.

Twisted μ_4-normal form [19] can be seen as the complement and extension of μ_4-normal form [20]. The related definitions, theorems, equation forms, and group laws of twisted μ_4-normal form and μ_4-normal form are given by Kohel's series of papers [19–22]. There are three forms for (twisted) μ_4-normal form, called (twisted) μ_4-normal form, (twisted) semisplit μ_4-normal form, and (twisted) split μ_4-normal form separately. Yet, for practical consideration, only twisted spilt μ_4-normal form will be used here.

Let C^t be an elliptic curve over characteristic-two finite field in the twisted split μ_4-normal form:

$$\begin{aligned} X_0^2 + X_2^2 &= c^2\left(X_1X_3 + a\left(X_1 + X_3\right)^2\right), \\ X_1^2 + X_3^2 &= c^2 X_0 X_2, \end{aligned} \quad (9)$$

and let (X_1, X_2, X_3, X_4) and (Y_1, Y_2, Y_3, Y_4) be two points on the curve. A complete system of addition laws is given by the two following two maps:

$$\begin{gathered} \left(\left(U_{00} + U_{22}\right)^2, c\left(U_{00}U_{11} + U_{22}U_{33} + aG\right), \left(U_{11} + U_{33}\right)^2, \right. \\ \left. c\left(U_{00}U_{33} + U_{11}U_{22} + aG\right)\right), \\ \left(\left(U_{13} + U_{31}\right)^2, c\left(U_{02}U_{31} + U_{20}U_{13} + aF\right), \left(U_{02} + U_{20}\right)^2, \right. \\ \left. c\left(U_{02}U_{13} + U_{20}U_{31} + aF\right)\right), \end{gathered} \quad (10)$$

respectively, where

$$\begin{aligned} U_{jk} &= X_j Y_k, \\ F &= V\left(U_{02} + U_{20}\right), \\ G &= V\left(U_{00} + U_{22}\right), \\ V &= \left(X_1 + X_3\right)\left(Y_1 + Y_3\right). \end{aligned} \quad (11)$$

For the point (X_1, X_2, X_3, X_4), the doubling map sends it to

$$\left(\left(X_0^4 + X_2^4\right) : c\left(X_0^2X_1^2 + X_2^2X_3^2\right) : \left(X_1^4 + X_3^4\right) : c\left(X_0^2X_3^2 + X_1^2X_2^2\right)\right), \quad (12)$$

if $a = 0$, and to

$$\left(\left(X_0^4 + X_2^4\right) : c\left(X_0^2X_3^2 + X_1^2X_2^2\right) : \left(X_1^4 + X_3^4\right) : c\left(X_0^2X_1^2 + X_2^2X_3^2\right)\right), \quad (13)$$

if $a = 1$. In twisted split μ_4-normal form, addition operations of generic points can be evaluated by $9M + 2S$ and doubling operations of a generic point can be evaluated by $2M + 5S$ with notations M for field multiplication and S for squaring [19].

Among all the studied coordinate systems on binary curves, twisted μ_4-normal form and λ projective coordinate appear to be faster. The difference is twisted μ_4-normal form is better calculating double-and-add, while λ projective coordinate can be used in halving operation. The costs of different point operations using various point representing systems are shown in Table 3.

Table 3: Cost comparison.

	Homogeneous	Jacobian	LD	λ	Twisted μ_4
Addition	$14M+1S$	$14M+5S$	$13M+4S$	$11M+2S$	$9M+2S$
Mixed-addition	$11M+1S$	$10M+3S$	$8M+5S$	$8M+2S$	$7M+2S$
Doubling	$7M+3S$	$4M+5S$	$3M+5S$	$3M+4S$	$2M+S$

2.3. Halving Operation. The main ingredient we consider is a cyclic subgroup in $E(\mathcal{F}_{2^m})$ of odd order r, denoted as $\mathcal{G}$. The multiple-by-2 isogeny [2]: $P \longrightarrow 2P$ on $\mathcal{G}$ is an isomorphism, so is its inverse map halving operation [1/2]: $P \longrightarrow (1/2)P$. The use of point halving to speedup scalar multiplication was firstly investigated by Knudsen [17]. Given a point $Q = (u, v) \in \mathcal{G}$, it allows to compute another point $P = (x, y) \in \mathcal{G}$ satisfying $Q = 2P$ in the cost of a field multiplication, calculating a square root and solving a quadratic equation, which could be directly understood from the formulas below:

$$\begin{cases} \lambda = \dfrac{x+y}{x}, \\ u = \lambda^2 + \lambda + a, \\ v = x^2 + u(\lambda + 1). \end{cases} \tag{14}$$

The most commonly used method is to solve the second equation for λ, then the third one for x, and finally the first one for y.

When λ coordinate like (x, λ) is used instead of affine coordinate (x, y), where $P = (x, \lambda_P)$ and $Q = (u, \lambda_Q)$, the halving operation formulas would be changed as follows:

$$\begin{cases} u = \lambda_P^2 + \lambda_P + a, \\ x^2 = u\left(u + \lambda_Q + \lambda_P + 1\right). \end{cases} \tag{15}$$

This time we just need two steps, that is to say, solve the first equation for λ_P and then the second one for x. Without computing y, the halving point coordinates $P = (x, \lambda_P)$ of $Q = (u, \lambda_Q)$ can be obtained more simply.

As proved in [23], solving a quadratic equation $x^2 + x = t$ on binary curves with $\mathrm{Tr}(a) = 1$ equivalents to computing the half-trace function $H(t) = \sum_{i=0}^{(m-1)/2} t^{2^{2i}}$. Although extra memory resources are needed, Fong et al. [23] showed a technique to significantly reduce the required time and space. With dedicated implementation, a point halving is approximately twice the time of a field multiplication, significantly faster than the customarily used point doubling.

From the algorithmic view, the halve-and-add method [17] expands a scalar k in radix-(1/2) representation system. Let l be the binary length of r, first compute $k' = k \cdot 2^l \bmod r = \sum_{i=0}^{l-1} k'_i \cdot 2^i$, that is, $k = k'/2^l \bmod r = \sum_{i=0}^{l-1} k'_i \cdot 2^i/2^l = \sum_{i=0}^{l-1} k'_{l-i-1}(1/2)^{i+1}$. Much similar to double-and-add, point multiplication,

$$kP = \sum_{i=0}^{l-1} k'_{l-i-1} \frac{P}{2^{i+1}} = k'_{l-1} \frac{P}{2} + k'_{l-2} \frac{P}{2^2} + \cdots + k'_0 \frac{P}{2^l}, \tag{16}$$

can be efficiently computed by applying point halving on an accumulator. It can be further optimized combining methods like ω-NAF to get a better implementation performance, as shown in [23].

Enlightened by the treatment in halve-and-add, if we choose an appropriate number less than l, the scalar k can be split into two parts naturally. In consequence, the halve-and-add method is easy to be concurrently implemented with the double-and-add algorithm in parallel model, making use of increasing cores in modern processors, which would be a lot faster than applying one algorithm without parallel implementation (some inevitable computation load should be considered in advance). Specifically speaking, if the lengths of r is l and a proper t has been chosen, the scalar k can be split into two portions applying halve-and-add and double-and-add algorithms simultaneously, which can be indicated as follows—the length of each part (t and $l - t$) depends on actual implementation speed of halving and doubling which can be found experimentally:

$$k' = 2^t \cdot k \bmod r. \tag{17}$$

If we already have the binary expression of $k' = \sum_{i=0}^{l-1} k'_i 2^i$ with odd order r, then it is easily derived that $k = k' \cdot 2^{-t} \bmod r = (k'_{t-1} \cdot 2^{-1} + \cdots + k'_0 \cdot 2^{-t}) + (k'_{l-1} \cdot 2^{l-t-1} + \cdots + k'_t) \bmod r$. The scalar multiplication $k \cdot P$ of P is then split into two parts directly:

$$\begin{aligned} k \cdot P = &\left(k'_{t-1} \cdot 2^{-1} + \cdots + k'_0 \cdot 2^{-t}\right) \cdot P \\ &+ \left(k'_{l-1} \cdot 2^{l-t-1} + \cdots + k'_t\right) \cdot P. \end{aligned} \tag{18}$$

The first part is easily executed in the halve-and-add method; meanwhile, the second part can be performed through a double-and-add approach, in two different threads.

As far as side-channel attacks being concerned, noticing that double-and-add can be implemented using Montgomery-ladder point multiplication, Negre and Robert [14] presented analogous Montgomery-halving algorithm. During each iteration, two registers hold fixed difference---$2P$, and the algorithm processes a point halving and an addition in each iteration. However, as noticed by the authors, this parallel algorithm can only be implemented in affine coordinate, since halving operation cannot be implemented in the projective coordinate efficiently. To overcome this drawback, we present another regular recoding algorithm that can be used when implementing parallel halve-and-add/double-and-add in the projective coordinate system.

3. Regular Implementation

Protecting the implementation of scalar multiplication against SSCA can be achieved by many methods. Compared with unprotected implementation, algorithmic countermeasures like recoding scalars in a regular manner always sacrifice efficiency, yet may be easily mitigated by taking advantage of inherent parallelism of modern processors.

3.1. Zero-Less Signed-Digit Expansion. In general, point addition and doubling of elliptic curves are very different from the usual arithmetic operations, which are so complicated and time consuming that plenty of scholars have been sparing no effort to find efficient approaches to speed them up like work in this paper. As is well known, the negative of a point is a very cheap operation ensuring subtraction of points on elliptic curves being just as efficient as addition. This motivates modifying the binary method to signed-digit representations, that is to say, the scalar k is usually represented by digits in the set of $\{-1, 0, 1\}$ instead of $\{0, 1\}$. As we all know, there are many kinds of signed-digit representations. For achieving our aim, in this paper, zero-less signed-digit expansion is chosen to be used to come up with regular algorithms improving the resistance of scalar multiplication against timing attack and SPA.

Zero-less signed-digit expansion [24] (ZSD) is a highly regular scalar recoding algorithm that expresses an odd integer k with digits in $\{-1, 1\}$. -1 is usually denoted as $\overline{1}$. Since bit 0 is avoided in recoded sequence, each iteration of point multiplication requires a double-and-add operation, providing a natural protection against timing attack and SPA.

Let $(k_{n-1}, k_{n-2}, \dots, k_0)$ be the binary expansion of a scalar k. Note that for any sequence of consecutive ω bits $00\cdots01$, the above expansion can be rewritten as $1\overline{1}\overline{1}\cdots\overline{1}$, i.e., $P = 2^{\omega}P - 2^{\omega-1}P - \cdots - 2P - P$. Similar treatment is able to be applied to radix-$(1/2)$ expansion of k, since $P/2^{\omega-1} = P - (P/2) - (P/2^2) - \cdots - (P/2^{\omega-1})$. When applying halve-and-add algorithm, any consecutive ω bits $00\cdots01$ can be rewritten as ω bits $1\overline{1}\overline{1}\cdots\overline{1}$ as well. So if k is an odd integer, its radix-2 ZSD expansion $k = \sum_{i=0}^{n-1}\widetilde{k}_i \cdot 2^i$ (or its corresponding one based on radix-$(1/2)$ represented by $2^{-n}k = \sum_{i=0}^{n-1}\widetilde{k}_i \cdot (1/2^{n-i})$) with $\widetilde{k}_i \in \{-1, 1\}$ can be obtained from

$$\begin{cases} \widetilde{k}_{n-1} = 1, \\ \widetilde{k}_i = (-1)^{k_{i+1}+1}, \\ \text{for } 0 \le i \le n-2. \end{cases} \tag{19}$$

From a security standpoint, every bit should be nonzero. When k is even, it requires a special treatment. This can be circumvented by computing kP with the least significant bit of k forced into 1 and finally subtracting P (or $(1/2^n)P$ in the corresponding condition) from the so-obtained result if bit k_0 is zero. The three algorithms in this paper applied this way to deal with kP or $2^{-n}kP$ correctly whether the input k is even or not.

Having known enough about ZSD expansion, we will get regular algorithms combining ZSD expansion and common binary methods to calculate the scalar multiplication. Algorithm 2 illustrates the regular ZSD double-and-add method based on radix-2 expansion from left to right, while Algorithm 3 does it from the opposite side.

Algorithms 2 and 3 give regular binary methods to evaluate elliptic scalar multiplication based on radix-2 expansion. When it comes to calculating $2^{-n}kP$, a similar condition based on radix-1/2 has to be considered, for which the halve-and-add method is needed. Referring to Algorithms 2 and 3, with a slight modification, we get Algorithm 1 for regular halving operation.

3.2. Parallelized Regular Scalar Multiplication. Let $P \in E(\mathscr{F}_{2^m})$ be the point of odd order r with bit length l and a scalar $k \in [0, r-1]$. The parallelized double-and-add/halve-and-add algorithm for scalar multiplication can be described in the following three parts including preprocessing, implementing, and postprocessing. Moreover, we may have a better view of the whole process from Figure 1.

Preprocessing: select a proper $t \in (0, l)$ and compute $k' = 2^t \cdot k \bmod r$.

So $k = 2^{-t} \cdot k' = [k'/2^t] + (k' \bmod 2^t) \cdot 2^{-t} = k_1{}' + k_2' \cdot 2^{-t}$, where k_1' is the most significant $l - t$ bits and k_2' is the least significant t bits of k'. This equation indicates $kP = (k'/2^t)P = k_1'P + (k_2'/2^t)P$.

Implementing: point multiplication can be done by concurrently implementing $k_1'P$ in the binary method, $(k_2'/2^t)P$ in radix-1/2 method in two different threads. In detail,

(1) Feed parameters k_1' and P as inputs to the regular double-and-add algorithm, exploiting Algorithm 2 or Algorithm 3, in one thread. The final result point P_1 is stored in the register.

(2) In the meanwhile, feed parameters k_2' and P as inputs to regular halve-and-add algorithm, Algorithm 1, in another thread. The final result point P_2 is stored in the register.

Postprocessing: a single-point addition $P_1 + P_2$ is operated to obtain the correct result of scalar multiplication.

4. Comparison and Expected Performance

Numerous standards have included NIST-recommended curves as implementation abelian groups for cryptographic protocols. The general conclusion in Tables 1 and 2 is specifically for NIST-recommended random curves having the form $y^2 + xy = x^3 + x^2 + b$, where b is an element in $\mathscr{F}_2(t)$. To allow easy comparison, the two considered curves with estimate results in this section are NIST B-233 and NIST B-409, defined by $f(t) = t^{233} + t^{74} + 1$ and $f(t) = t^{409} + t^{87} + 1$ over $\mathscr{F}_2(t)$, respectively.

```
     Input: $P \in E(\mathcal{F}_{2^m})$ of odd order $r$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$ with $k_{n-1} = 1$
     Output: $k \cdot P$
(1) $R_0 \longleftarrow P$; $R_1 \longleftarrow P$
(2) For $i = n - 1$ down to 1 do
(3)     $t \longleftarrow (-1)^{1+k_i}$
(4)     $R_0 \longleftarrow 2R_0 + t \cdot R_1$
(5) End for
(6) $R_{k_0} \longleftarrow R_{k_0} - R_{1-k_0}$
(7) Return $R_0$
```

ALGORITHM 2: Regular ZSD double-and-add (left-to-right) method.

```
      Input: $P \in E(\mathcal{F}_{2^m})$ of odd order $r$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$ with $k_{n-1} = 1$
      Output: $k \cdot P$
 (1)    $R_1 \longleftarrow P$
 (2)  If $k_0 = 0$ then
 (3)      $R_0 \longleftarrow -P$
 (4)  Else
 (5)      $R_0 \longleftarrow O$
 (6)  For $i = 1$ to $n - 1$ do
 (7)      $t \longleftarrow (-1)^{1+k_i}$
 (8)      $R_0 \longleftarrow R_0 + t \cdot R_1$; $R_1 \longleftarrow 2R_1$
 (9)  End for
(10)    $R_0 \longleftarrow R_0 + R_1$
(11)  Return $R_0$
```

ALGORITHM 3: Regular ZSD double-and-add (right-to-left) method.

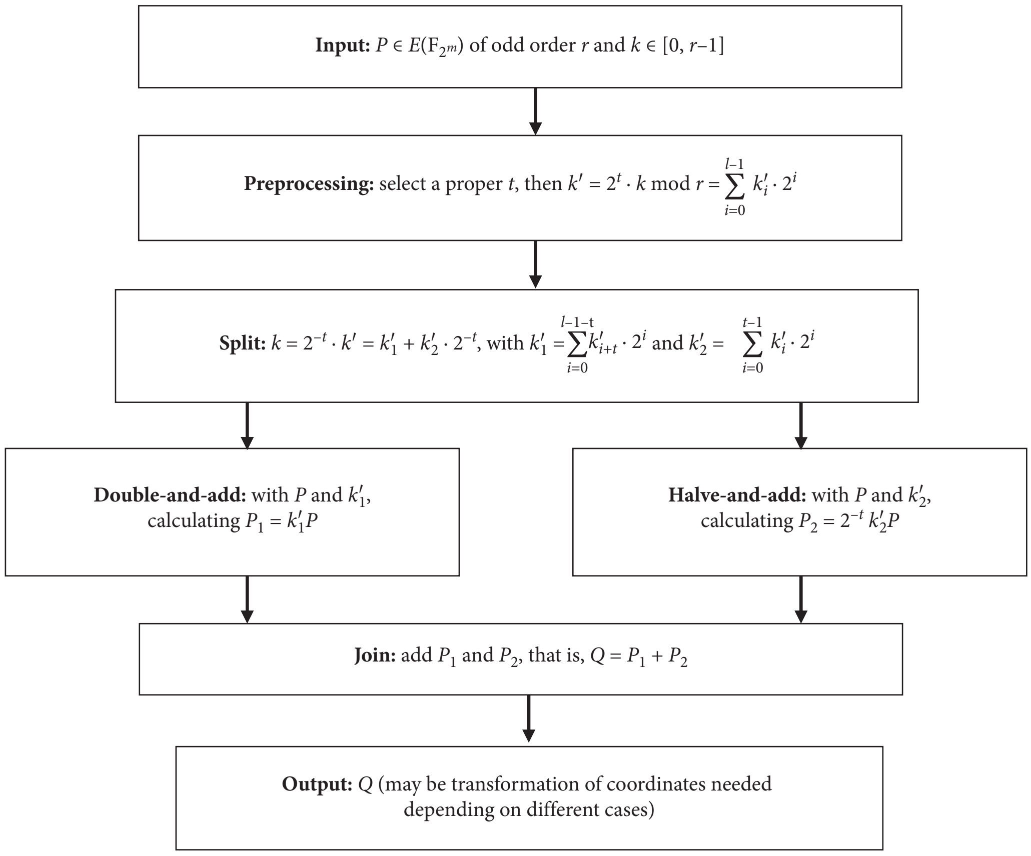
Input: $P \in E(\mathrm{F}_{2^m})$ of odd order r and $k \in [0, r-1]$
Preprocessing: select a proper t, then $k' = 2^t \cdot k \bmod r = \sum_{i=0}^{l-1} k'_i \cdot 2^i$
Split: $k = 2^{-t} \cdot k' = k'_1 + k'_2 \cdot 2^{-t}$, with $k'_1 = \sum_{i=0}^{l-1-t} k'_{i+t} \cdot 2^i$ and $k'_2 = \sum_{i=0}^{t-1} k'_i \cdot 2^i$
Double-and-add: with P and k'_1, calculating $P_1 = k'_1 P$
Halve-and-add: with P and k'_2, calculating $P_2 = 2^{-t} k'_2 P$
Join: add P_1 and P_2, that is, $Q = P_1 + P_2$
Output: Q (may be transformation of coordinates needed depending on different cases)

FIGURE 1: Parallel double-and-add/halve-and-add scalar multiplication.

4.1. Analysis. The theoretic complexity analysis of the four considered scalar multiplication approaches is reported in Table 1. Our work is to improve the algorithms in [14] and give a better new parallel algorithm for evaluating scalar multiplication (Algorithms 2 and 3 have the similar complexity, and just Algorithm 2 will be talked about in the following parts.)

For regular implementation against SSCA, the Montgomery methods and our new methods here both need m doubling and m addition point operations for double-and-add algorithms and m halving and m addition point operations for halve-and-add algorithms. To be specific, in Montgomery-D, D_M, and A_M mean doubling and addition operations of a very efficient Montgomery double-and-add algorithm in [25]. It is so excellent that only $6mM + 10M + 1I$ field operations are enough for Montgomery-D, where M and I represent field multiplication and inversion. In Montgomery-H, H_a and A_a are halving and addition operations in the affine coordinate. Halving usually includes computing field multiplication, trace, solving the quadratic equation, and computing the square root operations. According to the analysis and experimental results in [14, 23], we can assume halving in affine coordinate needs $3M$ field operations while $2M$ field operations for λ projective coordinate. Besides, A_a, addition in the affine coordinate needs $2M + 1I$ field operations. Unavoidably, the structure of Montgomery-H algorithm requires to use affine coordinate only, because no proper projective coordinate could be applied here so far, which influences its efficiency significantly. It can be easily seen from the estimate results later.

In Algorithm 2, D_{λ_p} and A_{λ_p} represent doubling and addition in λ projective coordinate separately. D_{μ_4} and A_{μ_4} represent doubling and addition in twisted μ_4 projective coordinate. As for their corresponding field operations, D_{λ_p} and A_{λ_p} requires $3M$ and $11M$, while D_{μ_4} and A_{μ_4} require $2M$ and $8M$. In Algorithm 1, H_{λ_a} means halving in λ affine coordinate and requires $2M$. Specially, if the mixed addition operation and the formula of calculating $2Q + P$ in Section 2.1 could be exploited, the field operations of Algorithm 2 will be $10mM + 8M + 3M + 1I$ for λ projective coordinate, in which $8M$ is the cost of final step mixed addition in Algorithm 2 and $3M + 1I$ is the cost of transforming the final result from λ projective coordinate to affine coordinate. When it turns to twisted μ_4 projective coordinate, $9mM + 7M + 2M + 1I$ field operations are needed, in which $7M$ is the cost of final step mixed addition in Algorithm 2 and $2M + 1I$ is the cost of transforming the final result from twisted μ_4 projective coordinate to affine coordinate. As for Algorithm 1, the mixed λ projective coordinate system could be applied saving inversion operations owing to the different algorithm structures of Algorithm 1 from Montgomery-H. Similarly, $10mM + 8M + 3M + 1I$ field operations are supposed to be consumed here.

In this work, we assume $I = 10M$ and ignore S for squaring multiplication here referring to [14, 23]. In fact, squaring is nearly the fastest among all the field operations we talk about in this paper and usually S is less than $0.2M$, so we can ignore it. For $I = 10M$, it is a commonly used reference value. Yet on most occasions, I may be bigger than $10M$, where Montgomery-H will be influenced most while the other three methods are almost unaffected. This is also the benefit of using the projective coordinate system. Having known all above cost comparison, two examples of NIST B-233 and B-409 are illustrated in Table 1 for easier understanding.

For double-and-add, the Montgomery-D algorithm is so outstanding that Algorithm 2 still could not catch up with the speed of it even using the twisted μ_4 projective coordinate system, which is the fastest to date. For halve-and-add, Algorithm 3 saves 32.7% and 33.0% cost compared with Montgomery-H with $m = 233$ and $m = 409$. That means our algorithm for regular halve-and-add is much more useful in practice by using projective coordinate. When making use of the faster algorithm, the parallel method would also be much more efficient.

One may ask why the mixed λ projective coordinate system could not be applied to Montgomery-H. It seems that comparing these two algorithms in different coordinate systems is so unfair. To be honest, it is not our tricks to do this on purpose. If we take a good look at Montgomery-H in [14], supposing that we already have Q_0 in λ affine coordinate and Q_1 in λ projective coordinate when $k_i' = 0$, we would meet the dilemma of transforming Q_1 into λ affine coordinate for halving operation and Q_0 into λ projective coordinate for mixed addition operation in order to save inversions when ${k_{i+1}}' = 1$. Every time the consecutive two bits are different, the transformation has to be done. For a random m bits binary number, if its leftmost bit is 1, then the average number of 0 next to 1 or 1 next to 0 is approximately $(m - 1)/2$. Transforming from λ projective coordinate into λ affine coordinate equals to $2M + 1I$ field operations. Taking these costs into account, Montgomery-H has to use around $(16m + 7)M$ field operations, which is more than applying affine coordinate. So the best solution to deal with the problem is to use a new structure like Algorithm 1.

4.2. Parallel and New Discovery. Negre and Robert [14] get inspiration from [26] and utilize a split technique similar to the one introduced in [26]. They also provide a Montgomery-halving algorithm like the original Montgomery-ladder scalar multiplication method. By carrying out these knowledge, a parallel method using Montgomery-D and Montgomery-H algorithms is presented. It is a pity that the Montgomery-H method from [14] can only use affine coordinate for its special structure. Aiming at solving this, we come up with a new regular parallel approach including Montgomery-D and Algorithm 1, which we call it mixed-parallel.

After analyzing each algorithm in Section 4.1, we can take a suitable split t to see complexity in parallel condition. The specific results are shown in Table 2. In the Algorithm column, Montgomery-parallel is the parallel algorithm in [14] meaning executing Montgomery-D and Montgomery-H concurrently in two different threads. Our mixed-parallel in the last line is the new united algorithm which applies Montgomery-D and Algorithm 1 simultaneously in different coprocessors.

It is evident that Montgomery-D has the least cost among all the algorithms in Table 1. However, either parallel method in Table 2 has less cost than Montgomery-D. Let us compare the Montgomery-parallel and Montgomery-D first. It turns out that Montgomery-parallel algorithm saves 27.5% and 28.0% cost than of Montgomery-D when $m = 233$ and $m = 409$. As a consequence, parallel is indeed a good idea for computing scalar multiplication. Furthermore, if we combine the best double-and-add Montgomery-D algorithm and the best halve-and-add Algorithm 1-H, a new efficient parallel method, mixed-parallel, jumps into our sight giving new hope. Estimating results demonstrate that our mixed-parallel method costs 11.7% and 12.0% less than that of Montgomery-parallel when $m = 233$ and $m = 409$, respectively. This is a new discovery and record.

5. Conclusion

In this paper, we present a new parallel algorithm to improve the Montgomery algorithm in [14]. The two methods both take advantage of inherent parallelism of modern processors constructing parallel approaches. Instead of using Montgomery-like idea, a regular recoding technique is applied in our approach which is supposed to be highly efficient by processing double-and-add and halve-and-add in a parallel way. The regular method could protect the computing process against SSCA like Montgomery thought.

After the careful analysis of these algorithms, we could draw the conclusion that our regular halve-and-add approach, Algorithm 1, could use λ projective coordinate making up for the disadvantage of Montgomery-H saving about 32.7% and 33.0% cost compared with that of Montgomery-H with $m = 233$ and $m = 409$.

As a result, combining Montgomery-D and Algorithm 1, a new preferable parallel approach is born, our mixed-parallel. It costs 11.7% and 12.0% less than that of Montgomery-parallel when $m = 233$ and $m = 409$, respectively. This is a new record as well as a good improvement and supplement to the previous excellent work of [14].

Data Availability

All data generated or analyzed during this study are included in this published article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Nos. 61872442, 61772515, 61502487, and U1936209); the National Cryptography Development Fund (No. MMJJ20180216); and the Beijing Municipal Science & Technology Commission (Project no. Z191100007119006).

References

[1] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[2] V. S. Miller, "Use of elliptic curves in cryptography," in *Proceedings of the Conference on the Theory and Application of Cryptographic Techniques*, Springer, Linz, Austria, April 1985.

[3] P. Kocher, J. Jaffe, and J. Benjamin, "Differential power analysis," in *Proceedings of the Annual International Cryptology Conference*, Springer, Santa Barbara, CA, USA, August 1999.

[4] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, Worcester, MA, USA, August 1999.

[5] S.-M. Yen and M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," *IEEE Transactions on Computers*, vol. 49, no. 9, pp. 967–970, 2000.

[6] Y. Sung-Ming, "A countermeasure against one physical cryptanalysis may benefit another attack," in *Proceedings of the International Conference on Information Security and Cryptology*, Springer, Seoul, Korea, December 2001.

[7] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of Computation*, vol. 48, no. 177, p. 243, 1987.

[8] M. Joye and M. Tunstall, "Exponent recoding and regular exponentiation algorithms," in *Proceedings of the International Conference on Cryptology in Africa*, Springer, Gammarth, Tunisia, June 2009.

[9] H. Lange and W. Ruppert, "Complete systems of addition laws on abelian varieties," *Inventiones Mathematicae*, vol. 79, no. 3, pp. 603–610, 1985.

[10] W. Bosma and H. W. Lenstra, "Complete systems of two addition laws for elliptic curves," *Journal of Number Theory*, vol. 53, no. 2, pp. 229–240, 1995.

[11] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, Springer, Kuching, Malaysia, December 2007.

[12] H. Hisil, "Twisted edwards curves revisited," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, Springer, Melbourne, Australia, December 2008.

[13] J. Renes, C. Craig, and L. Batina, "Complete addition formulas for prime order elliptic curves," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, Vienna, Austria, May 2016.

[14] C. Negre and J.-M. Robert, "New parallel approaches for scalar multiplication in elliptic curve over fields of small characteristic," *IEEE Transactions on Computers*, vol. 64, no. 10, pp. 2875–2890, 2015.

[15] D. Hankerson, A. J. Menezes, and V. Scott, *Guide to Elliptic Curve Cryptography*, Springer Science & Business Media, Berlin, Germany, 2006.

[16] J. López and R. Dahab, "Improved algorithms for elliptic curve arithmetic in GF (2 n)," in *Proceedings of the International Workshop on Selected Areas in Cryptography*, Springer, Kingston, Canada, August 1998.

[17] E. W. Knudsen, "Elliptic scalar multiplication using point halving," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, Springer, Singapore, November 1999.

[18] T. Oliveira, "Lambda coordinates for binary elliptic curves," in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, Santa Barbara, CA, USA, August 2013.

[19] D. Kohel, "Twisted μ_4-normal form for elliptic curves," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, Paris, France, April 2017.

[20] D. Kohel, "Efficient arithmetic on elliptic curves in characteristic 2," in *Proceedings of the International Conference on Cryptology in India*, Springer, Chennai, India, 2012.

[21] D. Kohel, "A normal form for elliptic curves in characteristic 2," in *Arithmetic, Geometry, Cryptography and Coding Theory (AGCT 2011)*, Springer, Berlin, Germany, 2011.

[22] D. Kohel, "Addition law structure of elliptic curves," *Journal of Number Theory*, vol. 131, no. 5, pp. 894–919, 2011.

[23] K. Fong, D. Hankerson, J. Lopez, and A. Menezes, "Field inversion and point halving revisited," *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 1047–1059, 2004.

[24] R. R. Goundar, M. Joye, A. Miyaji, M. Rivain, and A. Venelli, "Scalar multiplication on weierstraβ elliptic curves from Co-Z arithmetic," *Journal of Cryptographic Engineering*, vol. 1, no. 2, pp. 161–176, 2011.

[25] J. López and R. Dahab, "Fast multiplication on elliptic curves over GF (2 m) without precomputation," in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, Worcester, MA, USA, August 1999.

[26] J. Taverne, A. Faz-Hernández, D. F. Aranha, F. Rodríguez-Henríquez, D. Hankerson, and J. López, "Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction," *Journal of Cryptographic Engineering*, vol. 1, no. 3, pp. 187–199, 2011.